



開發人員指南，第 1 版

AWS IoT Greengrass



AWS IoT Greengrass: 開發人員指南，第 1 版

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

.....	xx
什麼是 AWS IoT Greengrass ?	1
AWS IoT Greengrass 核心軟體	3
AWS IoT Greengrass 核心軟體版本	3
AWS IoT Greengrass 群組	12
中的裝置 AWS IoT Greengrass	14
軟體開發套件	16
支援平台和需求	17
AWS IoT Greengrass 下載	29
AWS IoT Greengrass 核心軟體	29
AWS IoT Greengrass snap 軟體	36
AWS IoT Greengrass Docker 軟體	36
AWS IoT Greengrass 核心 SDK	38
支援的 Machine Learning 執行時間和程式庫	39
AWS IoT Greengrass ML 開發套件軟體	40
我們希望傾聽您的意見	40
安裝 AWS IoT Greengrass 核心軟體	40
下載並擷取 tar.gz 檔案	41
執行 Greengrass 裝置設定指定碼	41
從 APT 儲存庫安裝	41
AWS IoT Greengrass 在 Docker 容器中執行	43
AWS IoT Greengrass 快速執行	44
封存核心軟體安裝	54
設定 AWS IoT Greengrass 核心	56
AWS IoT Greengrass 核心組態檔案	57
服務端點必須符合憑證類型	105
連線至連接埠 443 或透過網路代理	106
設定寫入目錄	114
配置 MQTT 設定	117
啟動自動 IP 偵測	133
在系統開機啟動 Greengrass	136
另請參閱	137
AWS IoT Greengrass V1 維護政策	138
AWS IoT Greengrass 版本控制方案	138

Core AWS IoT Greengrass 軟體的生命週期階段	138
AWS IoT Greengrass Core 軟體的維護政策	139
維護階段排程	139
棄用排程	139
Lambda 函數的支援政策	140
AWS IoT Device Tester for 的支援政策 AWS IoT Greengrass V1	140
結束維護排程	140
Core AWS IoT Greengrass 軟體 v1.x Docker 映像的維護結束	37
Core AWS IoT Greengrass 軟體 v1.x APT 儲存庫的維護結束	142
Core AWS IoT Greengrass 軟體 v1.11.x Snap 的維護結束	142
入門 AWS IoT Greengrass	143
選擇如何開始使用	143
要求	146
建立 AWS 帳戶	147
註冊 AWS 帳戶	147
建立具有管理存取權的使用者	148
快速入門：Greengrass 裝置安裝	149
要求	150
執行 Greengrass 裝置安裝	150
疑難排解 問題	154
Greengrass 裝置安裝組態選項	154
單元 1：Greengrass 的環境設定	163
設定 Raspberry Pi	164
設定 Amazon EC2 執行個體	172
設定其他裝置	176
單元 2：安裝 AWS IoT Greengrass 核心軟體	179
佈建要用作 Greengrass 核心的 AWS IoT 物件	180
建立 Greengrass 群組	183
AWS IoT Greengrass 在核心裝置上安裝和執行	184
單元 3（第 1 部分）：上的 Lambda 函數 AWS IoT Greengrass	190
建立和封裝 Lambda 函數	191
設定的 Lambda 函數 AWS IoT Greengrass	195
部署雲端組態到核心裝置	198
驗證 Lambda 函數是否在核心裝置上執行	199
單元 3（第 2 部分）：Lambda 函數開啟 AWS IoT Greengrass	200
建立和封裝 Lambda 函數	201

為 設定長期 Lambda 函數 AWS IoT Greengrass	205
測試長期 Lambda 函數	205
測試隨需 Lambda 函數	208
單元 4：與 AWS IoT Greengrass 群組中的用戶端裝置互動	212
在 AWS IoT Greengrass 群組中建立用戶端裝置	214
設定訂閱	216
安裝 AWS IoT Device SDK for Python	217
測試通訊	223
第五單元：與裝置陰影互動	227
設定裝置和訂閱	229
下載必要的檔案	231
測試通訊 (停用裝置同步)	231
測試通訊 (啟用裝置同步)	235
單元 6：存取其他 AWS 服務	235
設定群組角色	237
建立並設定 Lambda 函式	239
設定訂閱	242
測試通訊	242
單元 7：模擬硬體安全整合	244
安裝 SoftHSM	245
設定 SoftHSM	245
匯入私有金鑰	247
設定 Greengrass 核心	248
測試組態	251
另請參閱	252
AWS IoT Greengrass Core 軟體的 OTA 更新	253
要求	253
OTA 更新的 IAM 許可	254
考量事項	257
Greengrass OTA 更新代理程式	257
與初始化系統整合	258
透過 OTA 更新受管的 respawn	259
建立 OTA 更新	260
CreateSoftwareUpdateJob API	263
部署 AWS IoT Greengrass 群組	266
部署群組 (主控台)	267

部署群組 (API)	268
取得群組 ID	269
群組物件模型概觀	271
群組	271
群組版本	271
群組元件	272
更新群組	273
另請參閱	274
取得部署通知	275
群組部署狀態變更事件	276
建立 EventBridge 規則的先決條件	277
設定部署通知 (主控台)	277
設定部署通知 (CLI)	279
設定部署通知 (CloudFormation)	279
另請參閱	280
重設部署	280
從 AWS IoT 主控台重設部署	280
使用 AWS IoT Greengrass API 重設部署	281
另請參閱	282
建立大量部署	282
先決條件	283
建立和上傳大量部署輸入檔	283
為大量部署建立和設定 IAM 執行角色	285
允許執行角色存取 S3 儲存貯體	288
部署群組	289
測試部署	291
大量部署故障診斷	293
另請參閱	295
執行本機 Lambda 函數	296
軟體開發套件	297
遷移雲端型 Lambda 函數	299
依別名或版本參考函數	300
控制 Greengrass Lambda 函數執行	300
群組專屬組態設定	301
執行 Lambda 函數做為根	304
選擇 Lambda 函數容器化時的考量事項	306

為群組中的 Lambda 函數設定預設存取身分	309
為群組中的 Lambda 函數設定預設容器化	310
通訊流程	311
使用 MQTT 訊息進行通訊	311
其他通訊流程	311
擷取輸入主題 (或主旨)	312
生命週期組態	314
Lambda 可執行檔	315
建立 Lambda 可執行檔	317
AWS IoT Greengrass 在 Docker 容器中執行	318
先決條件	319
從 Amazon ECR 取得 AWS IoT Greengrass 容器映像	320
建立及設定 Greengrass 群組和核心	324
在 AWS IoT Greengrass 本機執行	324
為群組的容器化設定「No container」(無容器)	327
將 Lambda 函數部署至 Docker 容器	328
(選用) 在 Docker 容器中部署與 Greengrass 互動的用戶端裝置	328
停止 AWS IoT Greengrass Docker 容器	328
在 Docker AWS IoT Greengrass 容器中進行故障診斷	328
存取本機資源	332
支援的資源類型	332
需求	333
在 /proc 目錄下的磁碟區資源	334
群組擁有者檔案存取許可	334
另請參閱	334
使用 CLI	335
建立本機資源	335
建立 Greengrass 函數	337
將 Lambda 函數新增至群組	338
疑難排解	340
使用主控台	341
先決條件	342
建立 Lambda 函數部署套件	342
建立和發佈 Lambda 函數	344
將 Lambda 函數新增至群組	346
新增本機資源至群組	347

新增訂閱到群組	347
部署群組	348
測試本機資源存取	350
執行機器學習推論	352
AWS IoT Greengrass ML 推論的運作方式	352
機器學習資源	353
支援的模型來源	353
要求	355
適用於 ML 推論的執行時間和程式庫	356
SageMaker AI Neo 深度學習執行期	356
MXNet 版本控制	356
在 Raspberry Pi 中的 MXNet	356
在 Raspberry Pi 上的 TensorFlow 模型服務限制	357
存取機器學習資源	357
機器學習資源的存取權限	358
定義 Lambda 函數的存取許可（主控台）	360
定義 Lambda 函數 (API) 的存取許可	360
從 Lambda 函數程式碼存取機器學習資源	363
疑難排解	364
另請參閱	366
如何設定機器學習推論	366
先決條件	367
設定 Raspberry Pi	368
安裝 MXNet 架構	369
建立模型套件	370
建立和發佈 Lambda 函數	370
將 Lambda 函數新增至群組	373
新增資源到群組	375
新增訂閱到群組	377
部署群組	378
測試應用程式	379
後續步驟	382
設定 Intel Atom	382
設定 NVIDIA Jetson TX2	385
如何設定最佳化的機器學習推論	390
先決條件	367

設定 Raspberry Pi	391
安裝 Neo 深度學習執行時間	393
建立推論 Lambda 函數	394
將 Lambda 函數新增至群組	397
將 Neo 最佳化模型資源新增至群組	399
將您的相機裝置資源新增至群組	401
新增訂閱到群組	402
部署群組	402
測試範例	403
設定 Intel Atom	404
設定 NVIDIA Jetson TX2	406
針對 AWS IoT Greengrass ML 推論進行故障診斷	380
後續步驟	412
管理資料串流	413
串流管理工作流程	413
要求	415
資料安全	416
本機資料安全性	416
用戶端身分驗證	417
另請參閱	417
設定串流管理員	417
串流管理員參數	418
進行設定 (主控台)	420
進行設定 (CLI)	423
另請參閱	432
使用 StreamManagerClient 搭配串流	432
建立訊息串流	433
附加訊息	437
讀取訊息	444
列出串流	446
描述訊息串流	448
更新訊息串流	450
刪除訊息串流	454
另請參閱	455
匯出支援 AWS 雲端 目的地的組態	456
匯出資料串流 (主控台)	471

先決條件	471
建立 Lambda 函數部署套件	473
建立 Lambda 函式	477
將函數新增到群組	479
啟用串流管理員	479
設定本機記錄	480
部署群組	480
測試應用程式。	481
另請參閱	482
匯出資料串流 (CLI)	483
先決條件	483
建立 Lambda 函數部署套件	486
建立 Lambda 函數	489
建立函數定義和版本	491
建立記錄器定義和版本	493
取得您核心定義版本的 ARN	494
建立群組版本	495
建立部署	496
測試應用程式。	497
另請參閱	498
將私密部署至核心	499
秘密金鑰加密	500
要求	501
指定用於秘密加密的私有金鑰	502
允許 AWS IoT Greengrass 取得秘密值	503
另請參閱	504
使用秘密資源	505
建立和管理秘密	505
使用本機私密	509
如何建立秘密資源 (主控台)	512
先決條件	513
建立 Secrets Manager 秘密	513
將私密資源新增至群組	514
建立 Lambda 函數部署套件	515
建立 Lambda 函式	517
新增 函數至群組	519

將秘密資源連接到函數	520
新增訂閱到群組	521
部署群組	521
測試 Lambda 函數	522
另請參閱	523
使用連接器整合服務和通訊協定	524
需求	525
使用 Greengrass 連接器	525
組態參數	527
用於存取群組資源的參數	527
更新連接器參數	528
輸入和輸出	528
輸入主題	529
容器化支援	529
升級連接器版本	530
日誌	531
AWS提供的 Greengrass 連接器	531
CloudWatch Metrics	534
Device Defender	549
Docker 應用程式部署	555
IoT Analytics	595
IoT 乙太網路 IP 通訊協定轉接器	609
IoT SiteWise	614
Kinesis Firehose	629
ML 意見回饋	645
ML 影像分類	662
ML 物件偵測	687
Modbus-RTU 通訊協定轉接器	703
Modbus-TCP 通訊協定轉接器	720
Raspberry Pi GPIO	726
序列串流	735
ServiceNow MetricBase 整合	748
SNS	761
Splunk 整合	772
Twilio 通知	785
連接器入門 (主控台)	801

先決條件	802
建立 Secrets Manager 秘密	803
將私密資源新增至群組	804
將連接器新增到群組	805
建立 Lambda 函數部署套件	805
建立 Lambda 函式	807
將函數新增到群組	809
新增訂閱到群組	809
部署群組	810
測試解決方案	811
另請參閱	812
連接器入門 (CLI)	812
先決條件	814
建立 Secrets Manager 秘密	815
建立資源定義和版本	816
建立連接器定義和版本	817
建立 Lambda 函數部署套件	818
建立 Lambda 函式	819
建立函數定義和版本	821
建立訂閱定義和版本	822
建立群組版本	824
建立部署	825
測試解決方案	826
另請參閱	828
Greengrass Discovery RESTful API	829
請求	829
回應	830
探索授權	830
範例 Discover 回應文件	831
安全	834
安全性概觀 AWS IoT Greengrass	834
裝置連線工作流程	836
設定 AWS IoT Greengrass 安全性	836
安全性主體	837
MQTT 簡訊工作流程中的受管訂閱	840
TLS 密碼套件支援	840

資料保護	842
資料加密	843
硬體安全整合	846
裝置身分驗證和授權	862
X.509 憑證	862
AWS IoT 政策	864
核心裝置的最低 AWS IoT 政策	866
身分與存取管理	870
目標對象	870
使用身分驗證	870
使用政策管理存取權	871
另請參閱	873
AWS IoT Greengrass 如何使用 IAM	873
Greengrass 服務角色	880
Greengrass 群組角色	887
預防跨服務混淆代理人	897
身分型政策範例	897
針對識別和存取問題進行故障診斷	900
法規遵循驗證	903
恢復能力	903
基礎架構安全	904
組態與漏洞分析	904
VPC 端點 (AWS PrivateLink)	905
AWS IoT Greengrass VPC 端點的考量事項	905
建立 AWS IoT Greengrass 控制平面操作的介面 VPC 端點	906
為 建立 VPC 端點政策 AWS IoT Greengrass	906
安全最佳實務	907
盡可能授予最低的許可	907
請勿在 Lambda 函數中硬式編碼登入資料	907
請勿記錄敏感資訊	907
建立目標訂閱	908
讓裝置的時鐘保持同步	908
使用 Greengrass 核心管理裝置身分驗證	908
另請參閱	909
日誌記錄和監控	910
監控工具	910

另請參閱	911
使用 AWS IoT Greengrass 日誌進行監控	911
存取 CloudWatch Logs	911
存取檔案系統日誌	913
預設日誌記錄組態	914
設定 的記錄 AWS IoT Greengrass	914
記錄限制	917
CloudTrail 日誌	919
使用 記錄 AWS IoT Greengrass API 呼叫 AWS CloudTrail	919
AWS IoT Greengrass CloudTrail 中的資訊	919
了解 AWS IoT Greengrass 日誌檔案項目	920
另請參閱	923
收集系統運作狀態遙測資料	923
設定遙測設定	926
訂閱以接收遙測資料	930
對 AWS IoT Greengrass 遙測進行故障診斷	936
呼叫本機運作狀態檢查 API	936
取得所有工作者的健康資訊	936
取得指定工作者的運作狀態資訊	938
工作者運作狀態資訊	940
標記您的 Greengrass 資源	943
標籤基本概念	943
標記支援 (主控台)	943
標記支援 (API)	943
搭配 IAM 政策使用標籤	945
範例 IAM 政策	946
另請參閱	948
CloudFormation 支援 AWS IoT Greengrass	949
建立 資源	949
部署資源	950
範例 範本	951
支援的 AWS 區域	963
使用 AWS IoT Device Tester for AWS IoT Greengrass V1	965
AWS IoT Greengrass 資格套件	965
自訂測試套件	966
AWS IoT Device Tester for AWS IoT Greengrass V1 的支援版本	966

for 不支援的 IDT 版本 AWS IoT Greengrass	966
使用 IDT AWS IoT Greengrass 執行資格套件	971
測試套件版本	972
測試群組描述	973
先決條件	977
設定您的裝置以執行 IDT 測試	985
設定 IDT 設定	1005
執行 AWS IoT Greengrass 資格套件	1019
了解結果和日誌	1024
使用 IDT 來開發和執行您自己的測試套件	1027
下載最新版本的 IDT for AWS IoT Greengrass	977
測試套件建立工作流程	1028
教學課程：建置並執行範例 IDT 測試套件	1029
教學課程：開發簡單的 IDT 測試套件	1034
建立 IDT 測試套件組態檔案	1043
設定 IDT 狀態機器	1050
建立 IDT 測試案例可執行檔	1071
使用 IDT 內容	1077
設定測試執行器的設定	1081
偵錯並執行自訂測試套件	1092
檢閱 IDT 測試結果和日誌	1094
IDT 用量指標	1100
IDT 故障 AWS IoT Greengrass 診斷	1106
錯誤代碼	1106
解決 IDT AWS IoT Greengrass 錯誤	1124
AWS IoT Device Tester for 的支援政策 AWS IoT Greengrass V1	1128
疑難排解	1129
AWS IoT Greengrass 核心問題	1129
錯誤：組態檔案缺少 CaPath、CertPath 或 KeyPath。Greengrass 協助程式在 [pid = <pid>] 結束狀態下處理。	1131
錯誤：無法剖析 /<greengrass-root>/config/config.json。	1131
錯誤：產生 TLS 組態時發生錯誤：ErrUnknownURIScheme	1132
錯誤：執行時間無法開始：無法啟動工作者：容器測試逾時。	1132
錯誤：無法在本機 Cloudwatch 叫用 PutLogEvents，logGroup: /GreengrassSystem/ connection_manager，錯誤：RequestError：傳送請求失敗，原因為：Post http://<path>/ cloudwatch/logs/: dial tcp <address>: getsockopt：連線遭拒，回應：{ }。	1133

錯誤：無法建立伺服器，因為：無法載入群組：chmod /<greengrass-root>/ggc/deployment/ lambda/arn : aws : lambda : <region> : <account-id> : function : <function-name> : <version>/<file-name> : 沒有此類檔案或目錄。	1133
AWS IoT Greengrass 核心軟體不會在您從沒有容器化的執行變更為在 Greengrass 容器中執 行後啟動。	1133
錯誤：多工緩衝處理應至少為 262144 位元組。	1134
錯誤：[ERROR]-雲端傳訊錯誤：嘗試發佈訊息時發生錯誤。{"errorString": "操作逾時"}	1134
錯誤：container_linux.go:344 : 啟動容器程序造成 "process_linux.go:424 : 容器 init 造成 \"rootfs_linux.go:64 : 將 \"/greengrass/ggc/socket/greengrass_ipc.sock\" 裝載至 rootfs \"/ greengrass/ggc/packages/<version>/rootfs/merged\" (位於 \"/greengrass_ipc.sock\") 造成 \"stat /greengrass/ggc/socket/greengrass_ipc.sock: permission denied\"\"\"。	1135
錯誤：具 PID : <process-id> 的執行中 Greengrass 協助程式。有些系統元件無法啟動。檢查 'runtime.log' 是否有錯誤。	1135
裝置陰影與雲端不同步。	901
錯誤：無法接受 TCP 連線。接受 tcp [::]:8000: accept4: 太多開啟的檔案。	1135
錯誤：執行時間執行錯誤：無法啟動 lambda 容器。container_linux.go:259 : 啟動容器程序造 成 "process_linux.go:345 : 容器 init 造成 \"rootfs_linux.go:50 : 準備 rootfs 造成 \"\"permission denied\"\"\"\"。	1136
警告：[WARN]-[5]GK 遠端：擷取公有金鑰資料發生錯誤：ErrPrincipalNotConfigured : 未設 定 MqttCertificate 的私有金鑰。	1136
錯誤：嘗試使用角色 arn : aws : iam : : <account-id> : role/<role-name> 存取 s3 url https://<region>-greengrass-updates.s3.<region>.amazonaws.com/core/<architecture>/ greengrass-core-<distribution-version>.tar.gz 時，許可遭拒。	901
AWS IoT Greengrass 核心設定為使用網路代理，您的 Lambda 函數無法進行傳出連線。 ..	1137
此核心位於連線/中斷連線的無限迴圈。runtime.log 檔案包含連線和中斷連線項目的連續系 列。	1137
錯誤：無法啟動 lambda 容器。container_linux.go: 259: 啟動容器程序導致 「process_linux.go: 345: container init 導致「rootfs_linux.go: 62: 將「proc」掛載到 rootfs」	1138
【ERROR】-執行時間執行錯誤：無法啟動 lambda 容器。{"errorString": "無法初始化容器 掛載：無法在浮水印上方 dir 中遮罩 greengrass 根目錄：無法在目錄 <ggc-path> : 檔案存 在"} 建立遮罩裝置	1139
【ERROR】-Deployment 失敗。{"deploymentId": "<deployment-id>", "errorString": "container test process with pid <pid> failed : container process status : exit status 1"} ...	1139
錯誤：[ERROR -執行時間執行錯誤：無法啟動 lambda 容器。{"errorString": "無法初 始化容器掛載：無法為容器建立覆蓋 fs : 正在將覆蓋掛載於 /greengrass/ggc/ packages/	

<ggc-version>/rootfs/merged 失敗：無法利用 args 來源進行掛載="no_source \" dest=\"/greengrass/ggc/packages/<ggc-version>/rootfs/merged\" fstype=\"overlay \" flags=\"0\" data=\"lowerdir=/greengrass/ggc/packages/<ggc-version>/dns:/,upperdir=/ greengrass/ggc/packages/<ggc-version>/rootfs/upper,workdir=/greengrass/ggc/packages/ <ggc-version>/rootfs/work\": too many levels of symbolic links\"}	1140
錯誤：[DEBUG]-無法取得路由。捨棄訊息。	1141
錯誤:[Errno 24] 太多開啟的 <lambda-function>，[Errno 24] 太多開啟的檔案	1141
錯誤：ds 伺服器無法開始接聽通訊端：接聽 unix <ggc-path>/ggc/socket/ greengrass_ipc.sock：bind：無效引數	1141
【INFO】(Copier) aws.greengrass.StreamManager：stdout。原因： com.fasterxml.jackson.databind.JsonMappingException：立即超過最小或最大立即	1141
GPG error: https://dnw9lb6lzp2d8.cloudfront.net stable InRelease: The following signatures were invalid: EXPKEYSIG 68D644ABD2327D47 AWS Greengrass Master Key	1142
部署問題	1142
您目前的部署無法運作，因此您想要回復到之前可正常運作的部署。	1144
您會在日誌中看到部署 403 Forbidden (403 禁止) 錯誤。	1146
當您第一次執行 create-deployment 命令時，就會發生 ConcurrentDeployment 錯誤。	1146
錯誤：Greengrass 未獲授權，無法擔任與此帳戶關聯的服務角色，或錯誤：失敗：TES 服務 角色並未與此帳戶關聯。	901
錯誤：無法在部署中執行下載步驟，下載時發生錯誤：下載群組定義檔案時發生錯誤：... x509：憑證已過期或無效	1146
部署尚未完成。	1146
錯誤：找不到 java 或 java8 可執行檔，或錯誤：適用於群組 <group-id> 的類型 NewDeployment 部署 <deployment-id> 失敗 錯誤：<worker-id> 的背景工作者無法初始化， 原因是安裝的 Java 版本必須大於或等於 8	1147
此部署沒有完成，且 runtime.log 含有多個「等待 1 秒讓容器停止」項目。	1147
部署未完成，且 runtime.log 包含 "[ERROR]-Greengrass 部署錯誤：無法將部署狀態回報 給雲端 {"deploymentId": "<deployment-id>", "errorString": "無法初始化 PUT，端點: https:// <deployment-status>，錯誤: Put https://<deployment-status>: proxyconnect tcp: x509: 未知 授權機構簽署的憑證"}"	1148
錯誤：適用於群組 <guid> 的類型 NewDeployment 部署 <deployment-id> 失敗錯誤：處理時 發生錯誤。群組設定無效：112 或 [119 0] 在檔案上沒有 rw 許可：<path>。	1149
錯誤：<list-of-function-arns> 設定為以根執行，但 Greengrass 未設定為以根許可執行 Lambda 函數。	1149
錯誤：適用於群組 <group-id> 的類型 NewDeployment 部署 <deployment-id> 失敗錯誤： Greengrass 部署錯誤：無法在部署中執行下載步驟。處理時發生錯誤：無法載入下載的群組	

檔案：無法根據使用者名稱 userName 找到 UID：userName：ggc_user：使用者：不明的使用者 ggc_user。	1149
錯誤：[ERROR] 執行時間執行錯誤：無法啟動 lambda 容器。{"errorString": "無法初始化容器掛載：無法在覆蓋上層目錄中遮罩 Greengrass 根目錄：無法在目錄 <ggc-path> 上建立遮罩裝置：檔案存在"}	1150
錯誤：適用於群組 <group-id> 類型 NewDeployment 的部署 <deployment-id> 失敗錯誤：程序開始失敗：container_linux.go:259：啟動容器程序造成 "process_linux.go:250：為 init 執行 exec setns 程序造成 \"等待：沒有子程序\"。	1150
錯誤：【WARN】-MQTT【client】撥號 tcp：lookup <host-prefix>-ats.iot.<region>.amazonaws.com：無此類主機... 【ERROR】-Greengrass 部署錯誤：無法將部署狀態回報給雲端 ... net/http：請求在等待連線時取消（在等待標頭時超過 Client.Timeout)	1150
建立群組和建立函數問題	1151
錯誤：群組的 'IsolationMode' 設定無效。	1151
錯誤：arn 為 <function-arn> 的 'IsolationMode' 設定無效。	1152
錯誤：IsolationMode=NoContainer 中不允許 arn 為 <function-arn> 的函數 MemorySize 設定。	1152
錯誤：IsolationMode=NoContainer 中不允許 arn 為 <function-arn> 的函數 Access Sysfs 設定。	1152
錯誤：IsolationMode=GreengrassContainer 中需要 arn 為 <function-arn> 的函數 MemorySize 設定。	1152
錯誤：函數 <function-arn> 會參考 IsolationMode = NoContainer 中不允許的類型 <resource-type> 資源。	1152
錯誤：不允許 arn 為 <function-arn> 的函數執行組態。	1153
Discovery 問題	1153
錯誤：裝置是太多群組的成員，裝置不得在超過 10 個群組中	1153
機器學習資源問題	1153
InvalidMLModelOwner - GroupOwnerSetting 在 ML 模型資源中提供，但是 GroupOwner 或 GroupPermission 不存在	365
附加 Machine Learning 資源時，NoContainer 函數無法設定權限。<function-arn> 係指在資源存取政策中，具有權限 <ro/rw> 的 Machine Learning 資源 <resource-id>。	365
函數 <function-arn> 係指在資源 <resource-id> ResourceAccessPolicy 和資源 OwnerSetting 中具有遺失權限的 Machine Learning 資源。	365
函數 <function-arn> 係指具有 \"rw\" 權限的 Machine Learning 資源 <resource-id>，而資源擁有者設定 GroupPermission 只允許 \"ro\"。	365
NoContainer 函數 <function-arn> 係指巢狀目的地路徑的資源。	365

Lambda <function-arn> 透過共用相同群組擁有者 ID 來獲得資源 <resource-id> 的存取權	366
AWS IoT Greengrass Docker 問題的核心	1155
錯誤：不明選項：-no-include-email。	329
警告：IPv4 已停用。網路將無法運作。	329
錯誤：防火牆封鎖了 Windows 和容器之間的檔案共用。	329
錯誤：呼叫 GetAuthorizationToken 操作時發生錯誤 (AccessDeniedException)：	
User：arn：aws：iam：：<account-id>：user/<user-name> 未獲授權在資源上執行：ecr：	
GetAuthorizationToken：*	329
錯誤：無法為服務 greengrass 建立容器：衝突。容器名稱 "/aws-iot-greengrass" 已在使用	
中。	1157
錯誤：[FATAL]-因為意外錯誤，無法重設執行緒的掛載命名空間：「不允許操作」。為了保持	
一致性，GGC 會當機且需要手動重新啟動。	1157
日誌故障診斷	1157
對儲存體問題進行故障診斷	1158
訊息故障診斷	1159
陰影同步逾時問題故障診斷	1159
檢查 AWS re：Post	1160
文件歷史紀錄	1161
舊版更新	1178

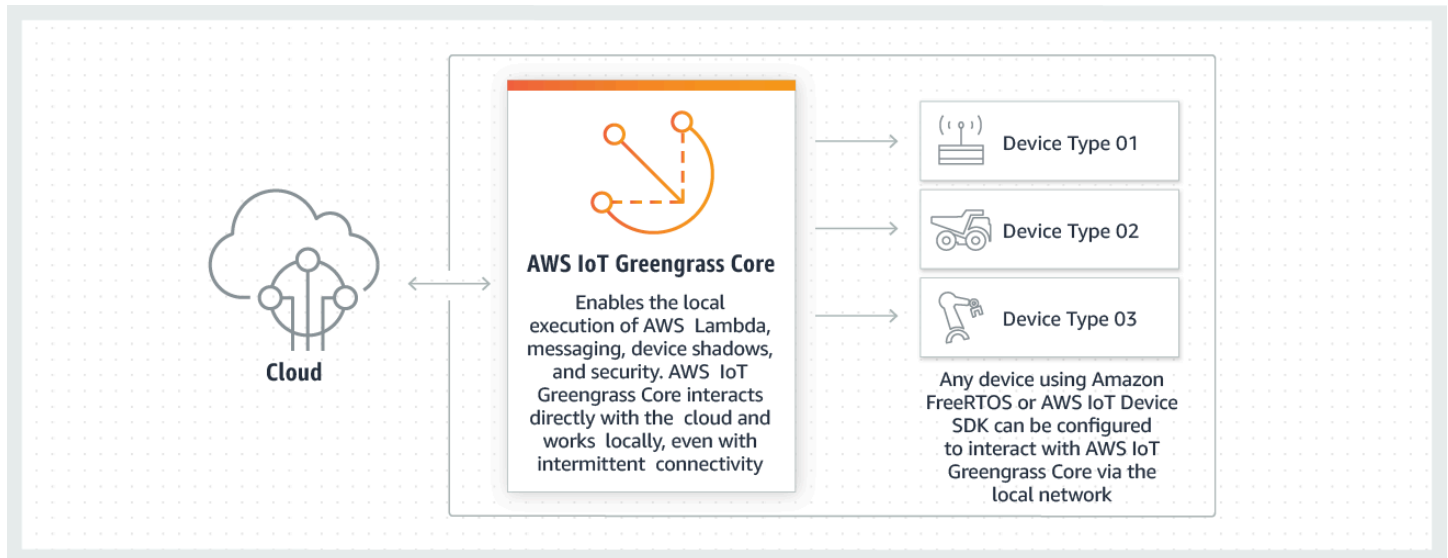
支援終止通知：2026 年 10 月 7 日 AWS 將停止 的支援 AWS IoT Greengrass Version 1。2026 年 10 月 7 日之後，您將無法再存取 AWS IoT Greengrass V1 資源。如需詳細資訊，請造訪[從 遷移 AWS IoT Greengrass Version 1](#)。

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。

什麼是 AWS IoT Greengrass ？

AWS IoT Greengrass 是將雲端功能擴展到本機裝置的軟體。這能讓裝置收集與分析更接近資訊來源的資料、自主回應本機裝置，在本機網路上安全地互相通訊。本機裝置也可以安全地與 通訊 AWS IoT Core，並將 IoT 資料匯出至 AWS 雲端。AWS IoT Greengrass 開發人員可以使用 AWS Lambda 函數和預先建置的[連接器](#)來建立部署至裝置以進行本機執行的無伺服器應用程式。

下圖顯示 的基本架構 AWS IoT Greengrass。



AWS IoT Greengrass 可讓客戶建置 IoT 裝置和應用程式邏輯。具體而言，AWS IoT Greengrass 提供在裝置上執行之應用程式邏輯的雲端型管理。本機部署的 Lambda 函數和連接器是由本機事件、來自雲端的訊息或其他來源觸發。

在 中 AWS IoT Greengrass，裝置可在本機網路上安全地通訊，並互相交換訊息，而不必連線到雲端。AWS IoT Greengrass 提供本機 pub/sub 訊息管理員，可在連線中斷時智慧緩衝訊息，以便保留傳入和傳出至雲端的訊息。

AWS IoT Greengrass 保護使用者資料：

- 透過裝置的安全身分驗證和授權。
- 在本機網路內安全的連線能力。
- 本機裝置和雲端之間。

裝置安全登入資料在群組內作用，直到撤銷，即使當連接至雲端的連線中斷，裝置還可以繼續安全地在本機進行通訊。

AWS IoT Greengrass 提供 Lambda over-the-air 更新。

AWS IoT Greengrass 包含：

- 軟體分發
 - AWS IoT Greengrass 核心軟體
 - AWS IoT Greengrass 核心 SDK
- 雲端服務
 - AWS IoT Greengrass API
- 功能
 - Lambda 執行時間
 - 陰影實作
 - 訊息管理員
 - 群組管理
 - 探索服務
 - 無線更新代理程式
 - 串流管理員
 - 本機資源存取
 - 本機機器學習推論
 - 本機秘密管理員
 - 內建整合服務、通訊協定和軟體的連接器

主題

- [AWS IoT Greengrass 核心軟體](#)
- [AWS IoT Greengrass 群組](#)
- [中的裝置 AWS IoT Greengrass](#)
- [軟體開發套件](#)
- [支援平台和需求](#)
- [AWS IoT Greengrass 下載](#)
- [我們希望傾聽您的意見](#)
- [安裝 AWS IoT Greengrass 核心軟體](#)
- [設定 AWS IoT Greengrass 核心](#)

AWS IoT Greengrass 核心軟體

Core AWS IoT Greengrass 軟體提供下列功能：

- 部署和本機執行連接器和 Lambda 函數。
- 在本機使用自動匯出至 來處理資料串流 AWS 雲端。
- 使用 受管訂閱，透過裝置、連接器和 Lambda 函數之間的本機網路傳送 MQTT 訊息。
- 使用受管訂閱的 AWS IoT 和 裝置、連接器和 Lambda 函數之間的 MQTT 訊息。
- AWS 雲端 使用裝置身分驗證和授權保護裝置與 之間的連線。
- 裝置的本機陰影同步。陰影可設定為與 同步 AWS 雲端。
- 對本機裝置和磁碟區資源控制的存取。
- 用於執行本機推論的雲端訓練機器學習模型部署。
- 可讓裝置探索 Greengrass 核心裝置的自動 IP 地址偵測。
- 新的或更新的群組組態的集中部署。下載組態資料之後，核心裝置會自動重新啟動。
- 使用者定義 Lambda 函數的安全over-the-air (OTA) 軟體更新。
- 本機秘密的安全加密儲存，以及連接器和 Lambda 函數控制的存取。

AWS IoT Greengrass 核心執行個體是透過 AWS IoT Greengrass APIs設定，以建立和更新存放在雲端中的 AWS IoT Greengrass 群組定義。

AWS IoT Greengrass 核心軟體版本

AWS IoT Greengrass 提供多種安裝 AWS IoT Greengrass Core 軟體的選項，包括 tar.gz 下載檔案、快速入門指令碼，以及在支援的 Debian 平台上apt安裝。如需詳細資訊，請參閱[the section called “安裝 AWS IoT Greengrass 核心軟體”](#)。

下列索引標籤說明 AWS IoT Greengrass Core 軟體版本中的新增和變更項目。

GGC v1.11

1.11.6

錯誤修正與效能改進：

- 改善在部署期間突然發生電源中斷時的彈性。
- 修正串流管理員資料損毀可能使 AWS IoT Greengrass Core 軟體無法啟動的問題。

- 已修正在某些情況下，新用戶端裝置無法連線至核心的問題。
- 修正串流管理員串流名稱不能包含 `.log` 的問題。

1.11.5

錯誤修正與效能改進：

- 一般效能改進與錯誤修正。

1.11.4

錯誤修正與效能改進：

- 修正串流管理員無法升級至 AWS IoT Greengrass Core 軟體 v1.11.3 的問題。如果您使用串流管理員將資料匯出至雲端，您現在可以使用 OTA 更新，將舊版 AWS IoT Greengrass 的核心軟體 v1.x 升級至 v1.11.4。
- 一般效能改進與錯誤修正。

1.11.3

錯誤修正與效能改進：

- 修正在 Ubuntu 裝置上快速執行 AWS IoT Greengrass 的 Core 軟體在裝置突然停電後停止回應的問題。
- 修正導致 MQTT 訊息延遲交付至長期 Lambda 函數的問題。
- 修正當 `maxWorkItemCount` 值設為大於 1024 的值時，導致 MQTT 訊息無法正確傳送的問題。
- 修正導致 OTA 更新代理程式忽略 `keepAlive` 屬性中指定的 MQTT KeepAlive 期間的問題 [config.json](#)。
- 一般效能改進與錯誤修正。

Important

如果您使用串流管理員將資料匯出至雲端，請勿從舊版 v1.x 升級到 AWS IoT Greengrass Core 軟體 v1.11.3。如果您是第一次啟用串流管理員，強烈建議您先安裝最新版本的 AWS IoT Greengrass Core 軟體。

1.11.1

錯誤修正與效能改進：

- 已修正造成串流管理員記憶體使用量增加的問題。
- 修正如果 Greengrass 核心裝置關閉的時間超過串流資料的指定time-to-live (TTL) 期間，導致串流管理員將串流的序號重設為的問題。
- 修正串流管理員無法正確停止重試嘗試將資料匯出至的問題 AWS 雲端。

1.11.0

新功能：

- Greengrass 核心上的遙測代理程式會收集本機遙測資料並將其發佈至 AWS 雲端。若要擷取遙測資料以供進一步處理，客戶可以建立 Amazon EventBridge 規則並訂閱目標。如需詳細資訊，請參閱[從 AWS IoT Greengrass 核心裝置收集系統運作狀態遙測資料](#)。
- 本機 HTTP API 會傳回由啟動之本機工作者程序目前狀態的快照 AWS IoT Greengrass。如需詳細資訊，請參閱[呼叫本機運作狀態檢查 API](#)。
- [串流管理員](#)會自動將資料匯出至 Amazon S3 和 AWS IoT SiteWise。

新的[串流管理員參數](#)可讓您更新現有的串流，並暫停或繼續資料匯出。

- 支援在核心上執行 Python 3.8.x Lambda 函數。
- 中的新ggDaemonPort屬性[config.json](#)，用於設定 Greengrass 核心 IPC 連接埠號碼。預設連接埠號碼為 8000。

在中[config.json](#)用來設定 Greengrass 核心 IPC 身分驗證逾時的
新systemComponentAuthTimeout屬性。預設逾時為 5000 毫秒。

- 將每個 AWS IoT Greengrass 群組 AWS IoT 的裝置數量上限從 200 個增加到 2500 個。

將每個群組的訂閱數量上限從 1000 個增加到 10000 個。

如需詳細資訊，請參閱 [AWS IoT Greengrass 端點和配額](#)。

錯誤修正與效能改進：

- 一般最佳化，可減少 Greengrass 服務程序的記憶體使用率。
- 新的執行時間組態參數 (mountAllBlockDevices) 可讓 Greengrass 在設定 OverlayFS 之後，使用繫結掛載將所有區塊裝置掛載到容器中。如果 /usr 不在/階層下，此功能解決了導致 Greengrass 部署失敗的問題。
- 修正如果 /tmp是符號連結，則會導致 AWS IoT Greengrass 核心故障的問題。
- 修正讓 Greengrass 部署代理程式從 mlmodel_public 資料夾移除未使用機器學習模型成品的問題。

- 一般效能改進與錯誤修正。

Extended life versions

1.10.5

錯誤修正與效能改進：

- 一般效能改進與錯誤修正。

1.10.4

錯誤修正與效能改進：

- 修正在 Ubuntu 裝置上快速執行 AWS IoT Greengrass 的 Core 軟體在裝置突然停電後停止回應的問題。
- 修正導致 MQTT 訊息延遲交付至長期 Lambda 函數的問題。
- 修正當maxWorkItemCount值設為大於 1024 的值時，導致 MQTT 訊息無法正確傳送的問題1024。
- 修正導致 OTA 更新代理程式忽略中 keepAlive 屬性中指定的 MQTT KeepAlive期間的問題[config.json](#)。
- 一般效能改進與錯誤修正。

1.10.3

錯誤修正與效能改進：

- 在 [config.json](#) 用來設定 Greengrass 核心 IPC 身分驗證逾時的 newSystemComponentAuthTimeout屬性。預設逾時為 5000 毫秒。
- 已修正造成串流管理員記憶體使用量增加的問題。

1.10.2

錯誤修正與效能改進：

- [config.json](#) 中的新mqttOperationTimeout屬性，用於設定 MQTT 連線中發佈、訂閱和取消訂閱操作的逾時 AWS IoT Core。
- 一般效能改進與錯誤修正。

1.10.1

錯誤修正與效能改進：

- [串流管理員](#)對檔案資料損毀更有彈性。
- 使用 Linux 核心 5.1 及更新版本，修正在裝置上造成 sysfs 掛載失敗的問題。
- 一般效能改進與錯誤修正。

1.10.0

新功能：

- 串流管理員，可在本機處理資料串流，並將其 AWS 雲端 自動匯出至 。此功能需使用 Greengrass 核心裝置上的 Java 8。如需詳細資訊，請參閱[管理資料串流](#)。
- 在核心裝置上執行 Docker 應用程式的全新 Greengrass Docker 應用程式部署連接器。如需詳細資訊，請參閱[the section called “Docker 應用程式部署”](#)。
- 新的 IoT SiteWise 連接器，可將工業裝置資料從 OPC-UA 伺服器傳送至其中的資產屬性 AWS IoT SiteWise。如需詳細資訊，請參閱[the section called “IoT SiteWise”](#)。
- 在沒有容器化的情況下執行的 Lambda 函數可以存取 Greengrass 群組中的機器學習資源。如需詳細資訊，請參閱[the section called “存取機器學習資源”](#)。
- 支援搭配 的 MQTT 持久性工作階段 AWS IoT。如需詳細資訊，請參閱[the section called “使用的 MQTT 持久性工作階段 AWS IoT Core”](#)。
- 本機 MQTT 流量可以透過預設連接埠 8883 以外的連接埠傳輸。如需詳細資訊，請參閱[the section called “用於本機訊息的 MQTT 連接埠”](#)。
- [AWS IoT Greengrass 核心 SDK](#) 中的新queueFullPolicy選項，用於從 Lambda 函數進行可靠的訊息發佈。
- 支援在核心上執行 Node.js 12.x Lambda 函數。
- 具備硬體安全性整合的無線 (OTA) 更新可透過 OpenSSL 1.1 進行設定。
- 一般效能改進與錯誤修正。

1.9.4

錯誤修正與效能改進：

- 一般效能改進與錯誤修正。

1.9.3

新功能：

- Armv6l. AWS IoT Greengrass Core 軟體 v1.9.3 或更新版本的支援可以安裝在 Armv6l 架構的 Raspbian 分佈上（例如 Raspberry Pi Zero 裝置）。

- 帶有 ALPN 的連接埠 443 OTA 更新。使用連接埠 443 進行 MQTT 流量的 Greengrass 核心現在支援over-the-air(OTA) 軟體更新。AWS IoT Greengrass 使用應用程式層通訊協定網路 (ALPN) TLS 延伸來啟用這些連線。如需詳細資訊，請參閱 [AWS IoT Greengrass Core 軟體的 OTA 更新](#)及 [the section called “連線至連接埠 443 或透過網路代理”](#)。

錯誤修正與效能改進：

- 修正 v1.9.0 中引入的錯誤，防止 Python 2.7 Lambda 函數將二進位承載傳送至其他 Lambda 函數。
- 一般效能改進與錯誤修正。

1.9.2

新功能：

- 支援 [OpenWrt](#). AWS IoT Greengrass Core 軟體 v1.9.2 或更新版本，可以安裝在具有 Armv8 (AArch64) 和 Armv7l 架構的 OpenWrt 分發上。目前，OpenWrt 不支援 ML 推論。

1.9.1

錯誤修正與效能改進：

- 修正存在於 1.9.0 版，從 cloud 寫出，標題中包含萬用字元之訊息的錯誤。

1.9.0

新功能：

- 支援 Python 3.7 和 Node.js 8.10 Lambda 執行時間。使用 Python 3.7 和 Node.js 8.10 執行期的 Lambda 函數現在可以在 AWS IoT Greengrass 核心上執行。(AWS IoT Greengrass 繼續支援 Python 2.7 和 Node.js 6.10 執行期。)
- 最佳化的 MQTT 連線。Greengrass 核心建立更少與 AWS IoT Core 的連線。這個變更可對根據連線數目的費用降低操作成本。
- 本機 MQTT 伺服器的 Elliptic Curve (EC) 金鑰。除了 RSA 金鑰之外，本機 MQTT 伺服器還支援 EC 金鑰。(無論金鑰類型為何，MQTT 伺服器憑證都有一個 SHA-256 RSA 簽章。) 如需詳細資訊，請參閱 [the section called “安全性主體”](#)。

錯誤修正與效能改進：

- 一般效能改進與錯誤修正。

1.8.4

已修正陰影同步和裝置憑證管理員重新連線的問題。

一般效能改進與錯誤修正。

1.8.3

一般效能改進與錯誤修正。

1.8.2

一般效能改進與錯誤修正。

1.8.1

一般效能改進與錯誤修正。

1.8.0

新功能：

- 群組中 Lambda 函數的可設定預設存取身分。此群組層級設定會決定用來執行 Lambda 函數的預設許可。您可以設定使用者 ID、群組 ID 或兩者。個別 Lambda 函數可以覆寫其群組的預設存取身分。如需詳細資訊，請參閱[the section called “為群組中的 Lambda 函數設定預設存取身分”](#)。
- 透過連接埠 443 的 HTTPS 流量。HTTPS 通訊可設定為透過連接埠 443 傳送，而非預設的連接埠 8443。這補充了對應用程式層通訊協定網路 (ALPN) TLS 延伸的 AWS IoT Greengrass 支援，並允許 MQTT 和 HTTPS 的所有 Greengrass 訊息流量都使用連接埠 443。如需詳細資訊，請參閱[the section called “連線至連接埠 443 或透過網路代理”](#)。
- AWS IoT 連線的可預測名稱用戶端 IDs。此變更可支援 AWS IoT Device Defender 和 [AWS IoT 生命週期事件](#)，因此您會收到連線、中斷連線、訂閱和取消訂閱事件的通知。可預測命名也能讓您更輕易建立關於連線 ID 的邏輯 (例如根據憑證屬性建立[訂閱政策範本](#))。如需詳細資訊，請參閱[the section called “與進行 MQTT 連線的用戶端 IDs AWS IoT”](#)。

錯誤修正與效能改進：

- 已修正陰影同步和裝置憑證管理員重新連線的問題。
- 一般效能改進與錯誤修正。

1.7.1

新功能：

- Greengrass 連接器提供與本機基礎設施、裝置通訊協定 AWS 和其他雲端服務的內建整合。如需詳細資訊，請參閱[使用連接器整合服務和通訊協定](#)。
- AWS IoT Greengrass 延伸 AWS Secrets Manager 到核心裝置，讓您的密碼、字符和其他秘密可供連接器和 Lambda 函數使用。私密是在傳輸和靜態時加密。如需詳細資訊，請參閱[將私密部署至核心](#)。

- 支援信任安全選項的硬體根。如需詳細資訊，請參閱[the section called “硬體安全整合”](#)。
- 允許 Lambda 函數在沒有 Greengrass 容器的情況下執行，並使用指定使用者和群組的許可的隔離和許可設定。如需詳細資訊，請參閱[the section called “控制 Greengrass Lambda 函數執行”](#)。
- 您可以將 Greengrass 群組設定為在沒有容器化的情況下執行，以在 Docker 容器（在 Windows、macOS 或 Linux 上）AWS IoT Greengrass 中執行。如需詳細資訊，請參閱[the section called “AWS IoT Greengrass 在 Docker 容器中執行”](#)。
- 運用應用程式層通訊協定交涉 (ALPN) 或透過網路代理的連線，在連接埠 443 上使用 MQTT 簡訊。如需詳細資訊，請參閱[the section called “連線至連接埠 443 或透過網路代理”](#)。
- SageMaker AI Neo 深度學習執行期，支援由 SageMaker AI Neo 深度學習編譯器最佳化的機器學習模型。如需 Neo 深度學習執行時間的資訊，請參閱 [the section called “適用於 ML 推論的執行時間和程式庫”](#)。
- 支援 Raspberry Pi 核心裝置上的 Raspbian Stretch (2018-06-27)。

錯誤修正與效能改進：

- 一般效能改進與錯誤修正。

此外，這個版本還提供以下功能：

- AWS IoT Device Tester for AWS IoT Greengrass，您可以使用它來驗證 CPU 架構、核心組態和驅動程式是否可運作 AWS IoT Greengrass。如需詳細資訊，請參閱[使用 AWS IoT Device Tester for AWS IoT Greengrass V1](#)。
- AWS IoT Greengrass Core 軟體、AWS IoT Greengrass Core SDK 和 AWS IoT Greengrass Machine Learning SDK 套件可透過 Amazon CloudFront 下載。如需詳細資訊，請參閱[the section called “AWS IoT Greengrass 下載”](#)。

1.6.1

新功能：

- 在 Greengrass 核心上執行二進位程式碼的 Lambda 可執行檔。使用適用於 C 的新 AWS IoT Greengrass 核心 SDK，以 C 和 C++ 撰寫 Lambda 可執行檔。如需詳細資訊，請參閱[the section called “Lambda 可執行檔”](#)。
- 選用本機儲存訊息快取可以在重新啟動持續。您可以設定佇列待處理的 MQTT 訊息之儲存設定。如需詳細資訊，請參閱[the section called “MQTT 訊息佇列”](#)。
- 當核心中斷連線時，可設定重新連線重試時間間隔的最大值。如需詳細資訊，請參閱 [the section called “AWS IoT Greengrass 核心組態檔案”](#) 中的 `mqttMaxConnectionRetryInterval` 屬性。

- 本機資源存取 host/proc 目錄。如需詳細資訊，請參閱[存取本機資源](#)。
- 可設定的寫入目錄。Core AWS IoT Greengrass 軟體可以部署到唯讀和讀寫位置。如需詳細資訊，請參閱[the section called “設定寫入目錄”](#)。

錯誤修正與效能改進：

- 在 Greengrass 核心中以及裝置與核心間執行發佈訊息的改善。
- 減少處理使用者定義 Lambda 函數所產生的日誌所需的運算資源。

1.5.0

新功能：

- AWS IoT Greengrass Machine Learning (ML) 推論已全面推出。您可以在本機 AWS IoT Greengrass 裝置上使用在雲端建構和訓練的模型來執行 ML 推導。如需詳細資訊，請參閱[執行機器學習推論](#)。
- 除了 JSON 之外，Greengrass Lambda 函數現在還支援二進位資料作為輸入承載。若要使用此功能，您必須升級至 AWS IoT Greengrass Core SDK 1.1.0 版，您可以從 [AWS IoT Greengrass Core SDK](#) 下載頁面下載。

錯誤修正與效能改進：

- 降低整體的記憶體足跡。
- 傳送訊息至雲端的效能提升。
- 下載代理程式的穩定性改善、裝置憑證管理員、和 OTA 更新代理程式。
- 次要錯誤修正。

1.3.0

新功能：

- 無線 (OTA) 更新代理程式能夠處理雲端部署的 Greengrass 更新任務。在新 /greengrass/ota 目錄中找到的代理程式。如需詳細資訊，請參閱[AWS IoT Greengrass Core 軟體的 OTA 更新](#)。
- 資源存取功能可讓 Greengrass Lambda 函數存取本機資源，例如週邊裝置和磁碟區。如需詳細資訊，請參閱[使用 Lambda 函數和連接器存取本機資源](#)。

1.1.0

新功能：

- 您可以透過刪除 Lambda 函數、訂閱和組態來重設部署的 AWS IoT Greengrass 群組。如需詳細資訊，請參閱[the section called “重設部署”](#)。

- 除了 Python 2.7 之外，還支援 Node.js 6.10 和 Java 8 Lambda 執行時間。

若要從舊版 AWS IoT Greengrass 的核心遷移：

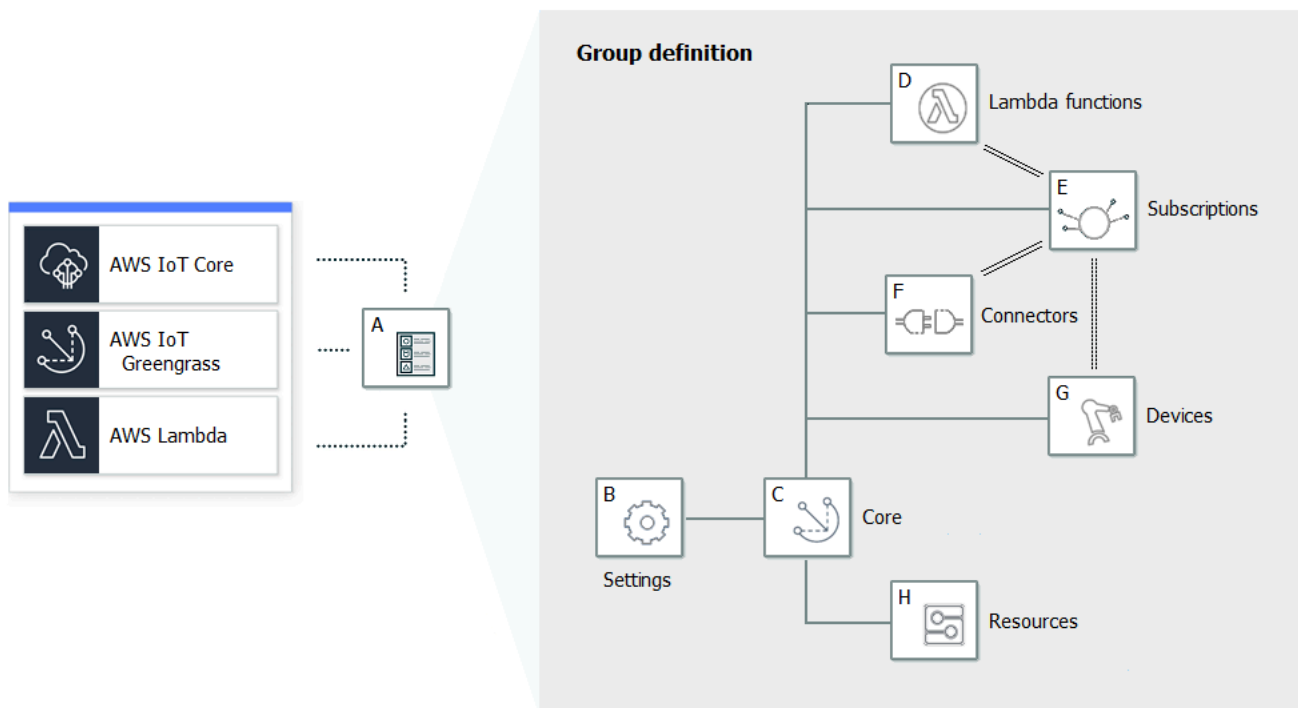
- 從 `/greengrass/configuration/certs` 資料夾複製憑證至 `/greengrass/certs`。
- 將 `/greengrass/configuration/config.json` 複製至 `/greengrass/config/config.json`。
- 執行 `/greengrass/ggc/core/greengrassd`，而不是 `/greengrass/greengrassd`。
- 部署群組到新的核心。

1.0.0

初始版本

AWS IoT Greengrass 群組

Greengrass 群組是設定和元件的集合，例如 Greengrass 核心、裝置和訂閱。群組可用來定義互動的範圍。例如，群組可能代表大樓的其中一個樓層、一輛卡車，或整個礦場。下圖顯示可組成 Greengrass 群組的元件。



在上圖中：

A：Greengrass 群組定義

群組設定和元件的相關資訊。

B：Greengrass 群組設定

其中包含：

- Greengrass 群組角色。
- 憑證授權機構和本機連線組態。
- Greengrass 核心連線資訊。
- 預設 Lambda 執行時間環境。如需詳細資訊，請參閱[the section called “為群組中的 Lambda 函數設定預設容器化”](#)。
- CloudWatch 和本機日誌組態。如需詳細資訊，請參閱[the section called “使用 AWS IoT Greengrass 日誌進行監控”](#)。

C：Greengrass 核心

代表 Greengrass 核心的 AWS IoT 物件（裝置）。如需詳細資訊，請參閱[the section called “設定 AWS IoT Greengrass 核心”](#)。

D：Lambda 函數定義

在核心本機執行的 Lambda 函數清單，以及相關聯的組態資料。如需詳細資訊，請參閱[執行本機 Lambda 函數](#)。

E：訂閱定義

可讓通訊使用 MQTT 訊息的訂閱清單。訂閱定義：

- 訊息來源和訊息目標。這些可以是用戶端裝置、Lambda 函數 AWS IoT Core、連接器和本機影子服務。
- 用來篩選訊息的主題 (或主旨)。

如需詳細資訊，請參閱[the section called “MQTT 簡訊工作流程中的受管訂閱”](#)。

F：連接器定義

在本機核心上執行的連接器清單，以及相關聯的組態資料。如需詳細資訊，請參閱[使用連接器整合服務和通訊協定](#)。

G：裝置定義

屬於 Greengrass 群組成員的 AWS IoT 物件清單（稱為用戶端裝置），以及相關聯的組態資料。如需詳細資訊，請參閱[the section called “中的裝置 AWS IoT Greengrass”](#)。

H：資源定義

Greengrass 核心上本機資源、機器學習資源和私密資源的清單，以及相關聯的組態資料。如需詳細資訊，請參閱[存取本機資源](#)、[執行機器學習推論](#)及[將私密部署至核心](#)。

部署時，Greengrass 群組定義、Lambda 函數、連接器、資源和訂閱資料表會複製到核心裝置。如需詳細資訊，請參閱[部署 AWS IoT Greengrass 群組](#)。

中的裝置 AWS IoT Greengrass

Greengrass 群組可以包含兩種類型的 AWS IoT 裝置：

Greengrass 核心

Greengrass 核心是執行 AWS IoT Greengrass Core 軟體的裝置，可讓它直接與 AWS IoT Core 和服務通訊 AWS IoT Greengrass。核心有自己的裝置憑證，可用來驗證 AWS IoT Core。它在 AWS IoT Core 登錄檔中具有裝置影子和項目。Greengrass 核心會執行本機 Lambda 執行期、部署代理程式和 IP 地址追蹤器，將 IP 地址資訊傳送至 AWS IoT Greengrass 服務，以允許用戶端裝置自動探索其群組和核心連線資訊。如需詳細資訊，請參閱[the section called “設定 AWS IoT Greengrass 核心”](#)。

Note

Greengrass 群組必須包含恰好一個核心。

用戶端裝置

用戶端裝置（也稱為連線裝置、Greengrass 裝置或裝置）是透過 MQTT 連接到 Greengrass 核心的裝置。他們在 AWS IoT Core 登錄檔中擁有自己的裝置憑證以進行 AWS IoT Core 身分驗證、裝置影子和項目。用戶端裝置可以執行 [FreeRTOS](#) 或使用 [AWS IoT 裝置 SDK](#) 或 [AWS IoT Greengrass 探索 API](#)，取得用來與相同 Greengrass 群組中核心進行連線和驗證的探索資訊。若要了解如何使用 AWS IoT 主控台為建立和設定用戶端裝置 AWS IoT Greengrass，請參閱 [the section called “單元 4：與 AWS IoT Greengrass 群組中的用戶端裝置互動”](#)。或者，如需示範如何使用 AWS CLI 建立和設定用戶端裝置的範例 AWS IoT Greengrass，請參閱《AWS CLI 命令參考》中的 [create-device-definition](#)。

在 Greengrass 群組中，您可以建立訂閱，允許用戶端裝置透過 MQTT 與群組中的 Lambda 函數、連接器和其他用戶端裝置，以及與 AWS IoT Core 或本機影子服務進行通訊。MQTT 訊息透過核心

路由。如果核心裝置失去與雲端的連線，用戶端裝置可以繼續透過本機網路進行通訊。用戶端裝置的大小可能有所不同，從較小的微型控制器型裝置到大型設備。目前，Greengrass 群組最多可包含 2,500 個用戶端裝置。用戶端裝置最多可為 10 個群組的成員。

Note

OPC-UA 是一個用於工業通訊的資訊交換標準。若要在 Greengrass 核心上實作對 OPC-UA 的支援，您可以使用 [IoT SiteWise 連接器](#)。連接器會將工業裝置資料從 OPC-UA 伺服器傳送至其中的資產屬性 AWS IoT SiteWise。

下表顯示這些裝置所相關的類型。

	Core	Device
Certificate	✓	✓
IoT Policy	✓	✓
IoT Thing	✓	✓
Device use	Gateway	Sensor and/or Actuator
Software	AWS IoT Greengrass Core Software	Amazon FreeRTOS / AWS IoT Device SDK
Group membership	✓	✓
Functions outside a Greengrass Group	✗	✓

AWS IoT Greengrass 核心裝置會將憑證存放在兩個位置：

- `/greengrass-root/certs` 中的核心裝置憑證。核心裝置憑證通常命名為 `hash.cert.pem` (例如，`86c84488a5.cert.pem`)。當核心連線至和 AWS IoT Core AWS IoT Greengrass 服務時，AWS IoT 用戶端會使用此憑證進行交互身分驗證。
- `/greengrass-root/ggc/var/state/server` 中的 MQTT 伺服器憑證。MQTT 伺服器憑證的名稱為 `server.crt`。此憑證用於本機 MQTT 伺服器 (位於 Greengrass 核心) 和 Greengrass 裝置之間的相互驗證。

Note

`greengrass-root` 代表在裝置上安裝 AWS IoT Greengrass Core 軟體的路徑。通常，這是 `/greengrass` 目錄。

軟體開發套件

下列 AWS 提供的 SDKs 可用於 AWS IoT Greengrass：

AWS 開發套件

使用 AWS SDK 建置與任何 AWS 服務互動的應用程式，包括 Amazon S3、Amazon DynamoDB AWS IoT AWS IoT Greengrass 等。在的內容中 AWS IoT Greengrass，您可以在部署的 Lambda 函數中使用 AWS SDK，直接呼叫任何 AWS 服務。如需詳細資訊，請參閱 [AWS SDKs](#)。

Note

也可在 [AWS IoT Greengrass API](#) 和 中使用 AWS SDKs 的 Greengrass 特定操作 [AWS CLI](#)。

AWS IoT 裝置 SDK

AWS IoT 裝置 SDK 可協助裝置連線至 AWS IoT Core 和 AWS IoT Greengrass。如需詳細資訊，請參閱《AWS IoT 開發人員指南 [AWS IoT](#)》中的 [裝置 SDKs](#)。

用戶端裝置可以使用任何 AWS IoT Device SDK v2 平台來探索 Greengrass 核心的連線資訊。連線資訊包括：

- 用戶端裝置所屬的 Greengrass 群組 IDs。
- 每個群組中 Greengrass 核心的 IP 地址。這些也稱為核心端點。

- 群組 CA 憑證，哪些裝置使用與核心進行交互身分驗證。如需詳細資訊，請參閱[the section called “裝置連線工作流程”](#)。

Note

在 AWS IoT 裝置 SDKs 的 v1 中，只有 C++ 和 Python 平台提供內建探索支援。

AWS IoT Greengrass 核心 SDK

AWS IoT Greengrass 核心 SDK 可讓 Lambda 函數與 Greengrass 核心互動、發佈訊息給 AWS IoT、與本機影子服務互動、叫用其他部署的 Lambda 函數，以及存取秘密資源。此 SDK 由在 AWS IoT Greengrass 核心上執行的 Lambda 函數使用。如需詳細資訊，請參閱[AWS IoT Greengrass 核心 SDK](#)。

AWS IoT Greengrass Machine Learning SDK

AWS IoT Greengrass Machine Learning SDK 可讓 Lambda 函數使用部署到 Greengrass 核心的機器學習模型作為機器學習資源。此 SDK 供在 AWS IoT Greengrass 核心上執行並與本機推論服務互動的 Lambda 函數使用。如需詳細資訊，請參閱[AWS IoT Greengrass Machine Learning SDK](#)。

支援平台和需求

下列索引標籤列出 AWS IoT Greengrass Core 軟體支援的平台和需求。

Note

您可以從 AWS IoT Greengrass 核心軟體下載下載 [AWS IoT Greengrass 核心軟體](#)。

GGC v1.11

支援的平台：

- 架構：Armv7l
 - 作業系統：Linux
 - 作業系統：Linux ([OpenWrt](#))
- 架構：Armv8 (AArch64)

- 作業系統：Linux
- 作業系統：Linux ([OpenWrt](#))
- 架構：Armv6l
 - 作業系統：Linux
- 架構：x86_64
 - 作業系統：Linux
- Windows、macOS 和 Linux 平台可以在 Docker AWS IoT Greengrass 容器中執行。如需詳細資訊，請參閱[the section called “AWS IoT Greengrass 在 Docker 容器中執行”](#)。

使用要求：

- AWS IoT Greengrass 核心軟體可用的磁碟空間至少為 128 MB。如果您使用 [OTA 更新代理程式](#)，則最小值為 400 MB。
- 配置給 AWS IoT Greengrass Core 軟體的最小 128 MB RAM。啟用[串流管理員](#)時，至少要用 198 MB RAM。

Note

如果您使用 AWS IoT 主控台上的預設群組建立選項來建立 Greengrass 群組，則預設會啟用串流管理員。

- Linux 核心版本：
 - 需要 Linux 核心 4.4 版或更新版本，才能支援 AWS IoT Greengrass 使用[容器](#)執行。
 - 需要 Linux 核心 3.17 版或更新版本，才能支援 AWS IoT Greengrass 在沒有容器的情況下執行。在此組態中，Greengrass 群組的預設 Lambda 函數容器化必須設定為無容器。如需說明，請參閱[the section called “為群組中的 Lambda 函數設定預設容器化”](#)。
- [GNU C 程式庫](#) (glibc) 2.14 版或更新版本。OpenWrt 發行套件需要 [musl C 程式庫](#) 1.1.16 版或更新版本。
- 必須在此裝置上呈現 /var/run 目錄。
- /dev/stdin、/dev/stdout 和 /dev/stderr 檔案必須可用。
- 必須在裝置上啟用硬連結和軟連結保護。否則，AWS IoT Greengrass 只能使用 -i 旗標在不安全模式下執行。
- 必須在此裝置啟用下列的 Linux 核心組態：
 - 命名空間：

- CONFIG_IPC_NS
- CONFIG_UTS_NS
- CONFIG_USER_NS
- CONFIG_PID_NS
- Cgroups :
 - CONFIG_CGROUP_DEVICE
 - CONFIG_CGROUPS
 - CONFIG_MEMCG

核心必須支援 [cgroups](#)。AWS IoT Greengrass 使用 [容器](#) 執行時，適用下列要求：

- 必須啟用並掛載記憶體 cgroup，以允許 AWS IoT Greengrass 設定 Lambda 函數的記憶體限制。
- 如果使用具有 [本機資源存取權](#) 的 Lambda 函數來開啟 AWS IoT Greengrass 核心裝置上的檔案，則必須啟用並掛載裝置 cgroup。
- 其他：
 - CONFIG_POSIX_MQUEUE
 - CONFIG_OVERLAY_FS
 - CONFIG_HAVE_ARCH_SECCOMP_FILTER
 - CONFIG_SECCOMP_FILTER
 - CONFIG_KEYS
 - CONFIG_SECCOMP
 - CONFIG_SHMEM
- Amazon S3 和 的根憑證 AWS IoT 必須存在於系統信任存放區中。
- 除了基本 AWS IoT Greengrass 核心軟體記憶體需求之外，[串流管理員](#) 還需要 Java 8 執行期和至少 70 MB RAM。當您在 AWS IoT 主控台上使用預設群組建立選項時，預設會啟用串流管理員。OpenWrt 發行版本不支援串流管理員。
- 支援您要在本機執行之 Lambda 函數所需執行 [AWS Lambda 時間](#) 的程式庫。必要的程式庫必須安裝在核心上，並新增到 PATH 環境變數。多個程式庫可以安裝在同一個核心上。
 - [Python](#) 3.8 版適用於使用 Python 3.8 執行時間的函數。
 - [Python](#) 3.7 版 (若為使用 Python 3.7 執行時間的函數)。
 - [Python](#) 2.7 版 (若為使用 Python 2.7 執行時間的函數)。

- [Java](#) 第 8 版或更新版本 (若為使用 Java 執行時間的函數)。

 Note

不正式支援在 OpenWrt 分發執行 Java。不過，如果您的 OpenWrt 組建有 Java 支援，您可能可以在 OpenWrt 裝置上執行以 Java 撰寫的 Lambda 函數。

如需 Lambda 執行時間 AWS IoT Greengrass 支援的詳細資訊，請參閱 [執行本機 Lambda 函數](#)。

- [無線 \(OTA\) 更新代理程式](#) 需要以下 shell 命令 (非 BusyBox 變體)：

- wget
- realpath
- tar
- readlink
- basename
- dirname
- pidof
- df
- grep
- umount
- mv
- gzip
- mkdir
- rm
- ln
- cut
- cat
- /bin/bash

GGC v1.10

支援的平台：

支援平台和需求

- 架構：Armv7l
 - 作業系統：Linux
 - 作業系統：Linux ([OpenWrt](#))
- 架構：Armv8 (AArch64)
 - 作業系統：Linux
 - 作業系統：Linux ([OpenWrt](#))
- 架構：Armv6l
 - 作業系統：Linux
- 架構：x86_64
 - 作業系統：Linux
- Windows、macOS 和 Linux 平台可以在 Docker AWS IoT Greengrass 容器中執行。如需詳細資訊，請參閱[the section called “AWS IoT Greengrass 在 Docker 容器中執行”](#)。

使用要求：

- AWS IoT Greengrass 核心軟體可用的磁碟空間至少為 128 MB。如果您使用 [OTA 更新代理程式](#)，則最小值為 400 MB。
- 配置給 AWS IoT Greengrass Core 軟體的最小 128 MB RAM。啟用[串流管理員](#)時，至少要用 198 MB RAM。

 Note

如果您使用 AWS IoT 主控台上的預設群組建立選項來建立 Greengrass 群組，則預設會啟用串流管理員。

- Linux 核心版本：
 - 需要 Linux 核心 4.4 版或更新版本，才能支援 AWS IoT Greengrass 使用[容器](#)執行。
 - 需要 Linux 核心 3.17 版或更新版本，才能支援 AWS IoT Greengrass 在沒有容器的情況下執行。在此組態中，Greengrass 群組的預設 Lambda 函數容器化必須設定為無容器。如需說明，請參閱[the section called “為群組中的 Lambda 函數設定預設容器化”](#)。
- [GNU C 程式庫](#) (glibc) 2.14 版或更新版本。OpenWrt 發行套件需要 [musl C 程式庫](#) 1.1.16 版或更新版本。
- 必須在此裝置上呈現 /var/run 目錄。
- /dev/stdin、/dev/stdout 和 /dev/stderr 檔案必須可用。

- 必須在裝置上啟用硬連結和軟連結保護。否則，AWS IoT Greengrass 只能使用 `-i` 旗標在不安全模式下執行。
- 必須在此裝置啟用下列的 Linux 核心組態：
 - 命名空間：
 - CONFIG_IPC_NS
 - CONFIG_UTS_NS
 - CONFIG_USER_NS
 - CONFIG_PID_NS
 - Cgroups：
 - CONFIG_CGROUP_DEVICE
 - CONFIG_CGROUPS
 - CONFIG_MEMCG

核心必須支援 [cgroups](#)。AWS IoT Greengrass 使用 [容器](#) 執行時，適用下列要求：

- 必須啟用並掛載記憶體 cgroup，以允許 AWS IoT Greengrass 設定 Lambda 函數的記憶體限制。
- 如果使用具有 [本機資源存取權](#) 的 Lambda 函數來開啟 AWS IoT Greengrass 核心裝置上的檔案，則必須啟用和掛載裝置 cgroup。
- 其他：
 - CONFIG_POSIX_MQUEUE
 - CONFIG_OVERLAY_FS
 - CONFIG_HAVE_ARCH_SECCOMP_FILTER
 - CONFIG_SECCOMP_FILTER
 - CONFIG_KEYS
 - CONFIG_SECCOMP
 - CONFIG_SHMEM
- Amazon S3 和 的根憑證 AWS IoT 必須存在於系統信任存放區中。
- 除了基本 AWS IoT Greengrass 核心軟體記憶體需求之外，[串流管理員](#)還需要 Java 8 執行期和至少 70 MB RAM。當您在 AWS IoT 主控台上使用預設群組建立選項時，預設會啟用串流管理員。OpenWrt 發行版本不支援串流管理員。
- ~~支援您要在本機執行之 Lambda 函數所需執行 [AWS Lambda 時間](#) 的程式庫。必要的程式庫必須安裝在核心上，並新增到 PATH 環境變數。多個程式庫可以安裝在同一個核心上。~~

- [Python](#) 3.7 版 (若為使用 Python 3.7 執行時間的函數)。
- [Python](#) 2.7 版 (若為使用 Python 2.7 執行時間的函數)。
- [Node.js](#) 12.x 版 (若為使用 Node.js 12.x 執行時間的函數)。
- [Java](#) 第 8 版或更新版本 (若為使用 Java 執行時間的函數)。

 Note

不正式支援在 OpenWrt 分發執行 Java。不過，如果您的 OpenWrt 組建有 Java 支援，您可能可以在 OpenWrt 裝置上執行以 Java 撰寫的 Lambda 函數。

如需 Lambda 執行時間 AWS IoT Greengrass 支援的詳細資訊，請參閱 [執行本機 Lambda 函數](#)。

- [無線 \(OTA\) 更新代理程式](#) 需要以下 shell 命令 (非 BusyBox 變體)：

- wget
- realpath
- tar
- readlink
- basename
- dirname
- pidof
- df
- grep
- umount
- mv
- gzip
- mkdir
- rm
- ln
- cut
- cat

- /bin/bash

GGC v1.9

支援的平台：

- 架構：Armv7l
 - 作業系統：Linux
 - 作業系統：Linux ([OpenWrt](#))
- 架構：Armv8 (AArch64)
 - 作業系統：Linux
 - 作業系統：Linux ([OpenWrt](#))
- 架構：Armv6l
 - 作業系統：Linux
- 架構：x86_64
 - 作業系統：Linux
- Windows、macOS 和 Linux 平台可以在 Docker AWS IoT Greengrass 容器中執行。如需詳細資訊，請參閱[the section called “AWS IoT Greengrass 在 Docker 容器中執行”](#)。

使用要求：

- AWS IoT Greengrass 核心軟體可用的磁碟空間至少為 128 MB。如果您使用 [OTA 更新代理程式](#)，則最小值為 400 MB。
- 配置給 AWS IoT Greengrass Core 軟體的最小 128 MB RAM。
- Linux 核心版本：
 - 需要 Linux 核心 4.4 版或更新版本，才能支援 AWS IoT Greengrass 使用[容器](#)執行。
 - 需要 Linux 核心 3.17 版或更新版本，才能支援 AWS IoT Greengrass 在沒有容器的情況下執行。在此組態中，Greengrass 群組的預設 Lambda 函數容器化必須設定為無容器。如需說明，請參閱[the section called “為群組中的 Lambda 函數設定預設容器化”](#)。
- [GNU C 程式庫](#) (glibc) 2.14 版或更新版本。OpenWrt 發行套件需要 [musl C 程式庫](#) 1.1.16 版或更新版本。
- 必須在此裝置上呈現 /var/run 目錄。
- /dev/stdin、/dev/stdout 和 /dev/stderr 檔案必須可用。
- 必須在裝置上啟用硬連結和軟連結保護。否則，AWS IoT Greengrass 只能使用 -i 旗標在不安全模式下執行。

- 必須在此裝置啟用下列的 Linux 核心組態：

- 命名空間：

- CONFIG_IPC_NS
- CONFIG_UTS_NS
- CONFIG_USER_NS
- CONFIG_PID_NS

- Cgroups：

- CONFIG_CGROUP_DEVICE
- CONFIG_CGROUPS
- CONFIG_MEMCG

核心必須支援 [cgroups](#)。AWS IoT Greengrass 使用 [容器](#) 執行時，適用下列要求：

- 必須啟用並掛載記憶體 cgroup，以允許 AWS IoT Greengrass 設定 Lambda 函數的記憶體限制。
- 如果使用具有 [本機資源存取權](#) 的 Lambda 函數來開啟 AWS IoT Greengrass 核心裝置上的檔案，則必須啟用並掛載裝置 cgroup。

- 其他：

- CONFIG_POSIX_MQUEUE
- CONFIG_OVERLAY_FS
- CONFIG_HAVE_ARCH_SECCOMP_FILTER
- CONFIG_SECCOMP_FILTER
- CONFIG_KEYS
- CONFIG_SECCOMP
- CONFIG_SHMEM

- Amazon S3 和 的根憑證 AWS IoT 必須存在於系統信任存放區中。
- 支援您要在本機執行之 Lambda 函數所需執行 [AWS Lambda 時間](#) 的程式庫。必要的程式庫必須安裝在核心上，並新增到 PATH 環境變數。多個程式庫可以安裝在同一個核心上。
 - [Python](#) 2.7 版 (若為使用 Python 2.7 執行時間的函數)。
 - [Python](#) 3.7 版 (若為使用 Python 3.7 執行時間的函數)。
 - [Node.js](#) 6.10 版或更新版本 (若為使用 Node.js 6.10 執行時間的函數)。
 - [Node.js](#) 8.10 版或更新版本 (若為使用 Node.js 8.10 執行時間的函數)。
 - [Java](#) 第 8 版或更新版本 (若為使用 Java 執行時間的函數)。

 Note

不正式支援在 OpenWrt 分發執行 Java。不過，如果您的 OpenWrt 組建有 Java 支援，您可能可以在 OpenWrt 裝置上執行以 Java 撰寫的 Lambda 函數。

如需 Lambda 執行時間 AWS IoT Greengrass 支援的詳細資訊，請參閱 [執行本機 Lambda 函數](#)。

- [無線 \(OTA\) 更新代理程式](#) 需要以下 shell 命令 (非 BusyBox 變體)：

- wget
- realpath
- tar
- readlink
- basename
- dirname
- pidof
- df
- grep
- umount
- mv
- gzip
- mkdir
- rm
- ln
- cut
- cat

GGC v1.8

- 支援的平台：
 - 架構：Armv7l；作業系統：Linux

- 架構：Armv8 (AArch64)；作業系統：Linux
- Windows、macOS 和 Linux 平台可以在 Docker AWS IoT Greengrass 容器中執行。如需詳細資訊，請參閱[the section called “AWS IoT Greengrass 在 Docker 容器中執行”](#)。
- Linux 平台可以使用 Greengrass 快照執行功能 AWS IoT Greengrass 有限的版本，該快照可透過 [Snapcraft](#) 取得。如需詳細資訊，請參閱[the section called “AWS IoT Greengrass snap 軟體”](#)。
- 下列是必要項目：
 - AWS IoT Greengrass 核心軟體可用的磁碟空間至少為 128 MB。如果您使用 [OTA 更新代理程式](#)，則最小值為 400 MB。
 - 配置給 AWS IoT Greengrass Core 軟體的最小 128 MB RAM。
 - Linux 核心版本：
 - 需要 Linux 核心 4.4 版或更新版本，才能支援 AWS IoT Greengrass 使用[容器](#)執行。
 - 需要 Linux 核心 3.17 版或更新版本，才能支援 AWS IoT Greengrass 在沒有容器的情況下執行。在此組態中，Greengrass 群組的預設 Lambda 函數容器化必須設定為無容器。如需說明，請參閱[the section called “為群組中的 Lambda 函數設定預設容器化”](#)。
 - [GNU C 程式庫](#) (glibc) 2.14 版或更新版本。
 - 必須在此裝置上呈現 `/var/run` 目錄。
 - `/dev/stdin`、`/dev/stdout` 和 `/dev/stderr` 檔案必須可用。
 - 必須在裝置上啟用硬連結和軟連結保護。否則，AWS IoT Greengrass 只能使用 `-i` 旗標在不安全模式下執行。
 - 必須在此裝置啟用下列的 Linux 核心組態：
 - 命名空間：
 - `CONFIG_IPC_NS`
 - `CONFIG_UTS_NS`
 - `CONFIG_USER_NS`
 - `CONFIG_PID_NS`
 - Cgroups：
 - `CONFIG_CGROUP_DEVICE`
 - `CONFIG_CGROUPS`
 - `CONFIG_MEMCG`

- 必須啟用並掛載記憶體 cgroup，以允許 AWS IoT Greengrass 設定 Lambda 函數的記憶體限制。
- 如果使用具有[本機資源存取權](#)的 Lambda 函數來開啟 AWS IoT Greengrass 核心裝置上的檔案，則必須啟用並掛載裝置 cgroup。
- 其他：
 - CONFIG_POSIX_MQUEUE
 - CONFIG_OVERLAY_FS
 - CONFIG_HAVE_ARCH_SECCOMP_FILTER
 - CONFIG_SECCOMP_FILTER
 - CONFIG_KEYS
 - CONFIG_SECCOMP
 - CONFIG_SHMEM
- Amazon S3 和 的根憑證 AWS IoT 必須存在於系統信任存放區中。
- 下列項目是必要條件：
 - 支援您要在本機[AWS Lambda 執行之 Lambda 函數所需執行時間](#)的程式庫。必要的程式庫必須安裝在核心上，並新增到 PATH 環境變數。多個程式庫可以安裝在同一個核心上。
 - [Python](#) 2.7 版 (若為使用 Python 2.7 執行時間的函數)。
 - [Node.js](#) 6.10 版或更新版本 (若為使用 Node.js 6.10 執行時間的函數)。
 - [Java](#) 第 8 版或更新版本 (若為使用 Java 執行時間的函數)。
 - [無線 \(OTA\) 更新代理程式](#)需要以下 shell 命令 (非 BusyBox 變體)：
 - wget
 - realpath
 - tar
 - readlink
 - basename
 - dirname
 - pidof
 - df
 - grep
 - mv

- `gzip`
- `mkdir`
- `rm`
- `ln`
- `cut`
- `cat`

如需配額 AWS IoT Greengrass（限制）的資訊，請參閱《》中的 [Service Quotas](#) Amazon Web Services 一般參考。

如需定價資訊，請參閱 [AWS IoT Greengrass 定價](#) 和 [AWS IoT Core 定價](#)。

AWS IoT Greengrass 下載

您可以使用以下資訊來尋找及下載與 AWS IoT Greengrass 搭配使用的軟體。

主題

- [AWS IoT Greengrass 核心軟體](#)
- [AWS IoT Greengrass snap 軟體](#)
- [AWS IoT Greengrass Docker 軟體](#)
- [AWS IoT Greengrass 核心 SDK](#)
- [支援的 Machine Learning 執行時間和程式庫](#)
- [AWS IoT Greengrass ML 開發套件軟體](#)

AWS IoT Greengrass 核心軟體

Core AWS IoT Greengrass 軟體將 AWS 功能延伸到 AWS IoT Greengrass 核心裝置，讓本機裝置可以在本機根據其產生的資料採取行動。

v1.11

1.11.6

錯誤修正與效能改進：

- 改善在部署期間突然發生電源中斷時的彈性。

- 修正串流管理員資料損毀可能使 AWS IoT Greengrass Core 軟體無法啟動的問題。
- 已修正在某些情況下，新用戶端裝置無法連線至核心的問題。
- 修正串流管理員串流名稱不能包含 `.log` 的問題。

1.11.5

錯誤修正與效能改進：

- 一般效能改進與錯誤修正。

1.11.4

錯誤修正與效能改進：

- 修正串流管理員無法升級至 AWS IoT Greengrass Core 軟體 v1.11.3 的問題。如果您使用串流管理員將資料匯出至雲端，您現在可以使用 OTA 更新，將舊版 AWS IoT Greengrass 的核心軟體 v1.x 升級至 v1.11.4。
- 一般效能改進與錯誤修正。

1.11.3

錯誤修正與效能改進：

- 修正在 Ubuntu 裝置上快速執行 AWS IoT Greengrass 的 Core 軟體在裝置突然停電後停止回應的問題。
- 修正導致 MQTT 訊息延遲交付至長期 Lambda 函數的問題。
- 修正當 `maxWorkItemCount` 值設為大於 1024 的值時，導致 MQTT 訊息無法正確傳送的問題。
- 修正導致 OTA 更新代理程式忽略 `keepAlive` 屬性中指定的 MQTT KeepAlive 期間的問題 [config.json](#)。
- 一般效能改進與錯誤修正。

Important

如果您使用串流管理員將資料匯出至雲端，請勿從舊版 v1.x 升級至 AWS IoT Greengrass Core 軟體 v1.11.3。如果您是第一次啟用串流管理員，強烈建議您先安裝最新版本的 AWS IoT Greengrass Core 軟體。

1.11.1

錯誤修正與效能改進：

- 已修正造成串流管理員記憶體使用量增加的問題。
- 修正如果 Greengrass 核心裝置關閉的時間超過串流資料的指定time-to-live (TTL) 期間，導致串流管理員將串流的序號重設為的問題。
- 修正串流管理員無法正確停止重試嘗試將資料匯出至的問題 AWS 雲端。

1.11.0

新功能：

- Greengrass 核心上的遙測代理程式會收集本機遙測資料並將其發佈至 AWS 雲端。若要擷取遙測資料以供進一步處理，客戶可以建立 Amazon EventBridge 規則並訂閱目標。如需詳細資訊，請參閱[從 AWS IoT Greengrass 核心裝置收集系統運作狀態遙測資料](#)。
- 本機 HTTP API 會傳回由啟動之本機工作者程序目前狀態的快照 AWS IoT Greengrass。如需詳細資訊，請參閱[呼叫本機運作狀態檢查 API](#)。
- [串流管理員](#)會自動將資料匯出至 Amazon S3 和 AWS IoT SiteWise。

新的[串流管理員參數](#)可讓您更新現有的串流，並暫停或繼續資料匯出。

- 支援在核心上執行 Python 3.8.x Lambda 函數。
- 中的新ggDaemonPort屬性[config.json](#)，用於設定 Greengrass 核心 IPC 連接埠號碼。預設連接埠號碼為 8000。

在 [config.json](#)用來設定 Greengrass 核心 IPC 身分驗證逾時的
新systemComponentAuthTimeout屬性。預設逾時為 5000 毫秒。

- 將每個 AWS IoT Greengrass 群組 AWS IoT 的裝置數量上限從 200 個增加到 2500 個。

將每個群組的訂閱數量上限從 1000 個增加到 10000 個。

如需詳細資訊，請參閱 [AWS IoT Greengrass 端點和配額](#)。

錯誤修正與效能改進：

- 一般最佳化，可減少 Greengrass 服務程序的記憶體使用率。
- 新的執行時間組態參數 (mountAllBlockDevices) 可讓 Greengrass 在設定 OverlayFS 之後，使用繫結掛載將所有區塊裝置掛載到容器中。如果 /usr 不在/階層下，此功能解決了導致 Greengrass 部署失敗的問題。
- 修正如果 /tmp是符號連結，則會導致 AWS IoT Greengrass 核心故障的問題。
- 修正讓 Greengrass 部署代理程式從 mlmodel_public 資料夾移除未使用機器學習模型成品的問題。

- 一般效能改進與錯誤修正。

若要在核心裝置上安裝 AWS IoT Greengrass 核心軟體，請為您的架構和作業系統 (OS) 下載 套件，然後遵循 [入門指南](#) 中的步驟。

 Tip

AWS IoT Greengrass 也提供安裝 AWS IoT Greengrass Core 軟體的其他選項。例如，您可以使用 [Greengrass 裝置設定](#) 來設定您的環境，並安裝最新版本的 AWS IoT Greengrass Core 軟體。或者，在支援的 Debian 平台上，您可以使用 [APT 套件管理員](#) 來安裝或升級 AWS IoT Greengrass 核心軟體。如需詳細資訊，請參閱 [the section called “安裝 AWS IoT Greengrass 核心軟體”](#)。

Architecture	作業系統	連結
Armv8 (AArch64)	Linux	下載
Armv8 (AArch64)	Linux (OpenWrt)	下載
Armv7l	Linux	下載
Armv7l	Linux (OpenWrt)	下載
Armv6l	Linux	下載
x86_64	Linux	下載

Extended life versions

1.10.5

v1.10 中的新功能：

- 串流管理員，可在本機處理資料串流，並將其 AWS 雲端 自動匯出至 。此功能需使用 Greengrass 核心裝置上的 Java 8。如需詳細資訊，請參閱 [管理資料串流](#)。
- 在核心裝置上執行 Docker 應用程式的全新 Greengrass Docker 應用程式部署連接器。如需詳細資訊，請參閱 [the section called “Docker 應用程式部署”](#)。

- 新的 IoT SiteWise 連接器，可將工業裝置資料從 OPC-UA 伺服器傳送至其中的資產屬性 AWS IoT SiteWise。如需詳細資訊，請參閱[the section called “IoT SiteWise”](#)。
- 在沒有容器化的情況下執行的 Lambda 函數可以存取 Greengrass 群組中的機器學習資源。如需詳細資訊，請參閱[the section called “存取機器學習資源”](#)。
- 支援搭配 MQTT 持久性工作階段 AWS IoT。如需詳細資訊，請參閱[the section called “使用的 MQTT 持久性工作階段 AWS IoT Core”](#)。
- 本機 MQTT 流量可以透過預設連接埠 8883 以外的連接埠傳輸。如需詳細資訊，請參閱[the section called “用於本機訊息的 MQTT 連接埠”](#)。
- [AWS IoT Greengrass 核心 SDK](#) 中的新queueFullPolicy選項，用於從 Lambda 函數進行可靠的訊息發佈。
- 支援在核心上執行 Node.js 12.x Lambda 函數。

錯誤修正與效能改進：

- 具備硬體安全性整合的無線 (OTA) 更新可透過 OpenSSL 1.1 進行設定。
- [串流管理員](#)對檔案資料損毀更有彈性。
- 使用 Linux 核心 5.1 及更新版本，修正在裝置上造成 sysfs 掛載失敗的問題。
- [config.json](#) 中的新mqttOperationTimeout屬性，用於設定 MQTT 連線中發佈、訂閱和取消訂閱操作的逾時 AWS IoT Core。
- 已修正造成串流管理員記憶體使用量增加的問題。
- 在中[config.json](#)用來設定 Greengrass 核心 IPC 身分驗證逾時的新systemComponentAuthTimeout屬性。預設逾時為 5000 毫秒。
- 修正導致 OTA 更新代理程式忽略中 keepAlive 屬性中指定的 MQTT KeepAlive期間的問題[config.json](#)。
- 修正當maxWorkItemCount值設為大於 1024 的值時，導致 MQTT 訊息無法正確傳送的問題1024。
- 修正導致 MQTT 訊息延遲交付至長期 Lambda 函數的問題。
- 修正在 Ubuntu 裝置上快速執行 AWS IoT Greengrass 的 Core 軟體在裝置突然停電後停止回應的問題。
- 一般效能改進與錯誤修正。

若要在核心裝置上安裝 AWS IoT Greengrass 核心軟體，請為您的架構和作業系統 (OS) 下載套件，然後遵循 [入門指南](#) 中的步驟。

Architecture	作業系統	連結
Armv8 (AArch64)	Linux	下載
Armv8 (AArch64)	Linux (OpenWrt)	下載
Armv7l	Linux	下載
Armv7l	Linux (OpenWrt)	下載
Armv6l	Linux	下載
x86_64	Linux	下載

1.9.4

v1.9 中的新功能：

- 支援 Python 3.7 和 Node.js 8.10 Lambda 執行時間。使用 Python 3.7 和 Node.js 8.10 執行期的 Lambda 函數現在可以在 AWS IoT Greengrass 核心上執行。(AWS IoT Greengrass 繼續支援 Python 2.7 和 Node.js 6.10 執行期。)
- 最佳化的 MQTT 連線。Greengrass 核心建立更少與 AWS IoT Core 的連線。這個變更可對根據連線數目的費用降低操作成本。
- 本機 MQTT 伺服器的 Elliptic Curve (EC) 金鑰。除了 RSA 金鑰之外，本機 MQTT 伺服器還支援 EC 金鑰。(無論金鑰類型為何，MQTT 伺服器憑證都有一個 SHA-256 RSA 簽章。) 如需詳細資訊，請參閱[the section called “安全性主體”](#)。
- 支援 [OpenWrt](#)。AWS IoT Greengrass Core 軟體 v1.9.2 或更新版本，可以安裝在具有 Armv8 (AArch64) 和 Armv7l 架構的 OpenWrt 分發上。目前，OpenWrt 不支援 ML 推論。
- Armv6l。AWS IoT Greengrass Core 軟體 v1.9.3 或更新版本的支援可以安裝在 Armv6l 架構的 Raspbian 分佈上（例如 Raspberry Pi Zero 裝置）。
- 帶有 ALPN 的連接埠 443 OTA 更新。使用連接埠 443 進行 MQTT 流量的 Greengrass 核心現在支援 over-the-air (OTA) 軟體更新。AWS IoT Greengrass 使用應用程式層通訊協定網路 (ALPN) TLS 延伸來啟用這些連線。如需詳細資訊，請參閱 [AWS IoT Greengrass Core 軟體的 OTA 更新](#) 及 [the section called “連線至連接埠 443 或透過網路代理”](#)。

若要在核心裝置上安裝 AWS IoT Greengrass 核心軟體，請為您的架構和作業系統 (OS) 下載套件，然後遵循 [入門指南](#) 中的步驟。

Architecture	作業系統	連結
Armv8 (AArch64)	Linux	下載
Armv8 (AArch64)	Linux (OpenWrt)	下載
Armv7l	Linux	下載
Armv7l	Linux (OpenWrt)	下載
Armv6l	Linux	下載
x86_64	Linux	下載

1.8.4

- 新功能：
 - 群組中 Lambda 函數的可設定預設存取身分。此群組層級設定會決定用來執行 Lambda 函數的預設許可。您可以設定使用者 ID、群組 ID 或兩者。個別 Lambda 函數可以覆寫其群組的預設存取身分。如需詳細資訊，請參閱[the section called “為群組中的 Lambda 函數設定預設存取身分”](#)。
 - 透過連接埠 443 的 HTTPS 流量。HTTPS 通訊可設定為透過連接埠 443 傳送，而非預設的連接埠 8443。這補充了對應用程式層通訊協定網路 (ALPN) TLS 延伸的 AWS IoT Greengrass 支援，並允許 MQTT 和 HTTPS 的所有 Greengrass 訊息流量都使用連接埠 443。如需詳細資訊，請參閱[the section called “連線至連接埠 443 或透過網路代理”](#)。
 - AWS IoT 連線的可預測名稱用戶端 IDs。此變更可支援 AWS IoT Device Defender 和 [AWS IoT 生命週期事件](#)，因此您會收到連線、中斷連線、訂閱和取消訂閱事件的通知。可預測命名也能讓您更輕易建立關於連線 ID 的邏輯 (例如根據憑證屬性建立[訂閱政策範本](#))。如需詳細資訊，請參閱[the section called “與進行 MQTT 連線的用戶端 IDs AWS IoT”](#)。

錯誤修正與效能改進：

- 已修正陰影同步和裝置憑證管理員重新連線的問題。
- 一般效能改進與錯誤修正。

若要在核心裝置上安裝 AWS IoT Greengrass 核心軟體，請下載架構和作業系統 (OS) 的套件，然後遵循 [入門指南](#) 中的步驟。

Architecture	作業系統	連結
Armv8 (AArch64)	Linux	下載
Armv7l	Linux	下載
x86_64	Linux	下載

下載此軟體，即表示您同意 [Greengrass 核心軟體授權合約](#) 之規定。

如需在裝置上安裝 AWS IoT Greengrass Core 軟體之其他選項的相關資訊，請參閱 [the section called “安裝 AWS IoT Greengrass 核心軟體”](#)。

AWS IoT Greengrass snap 軟體

AWS IoT Greengrass snap 1.11.x 可讓您 AWS IoT Greengrass 透過方便的軟體套件以及所有必要的相依性，在容器化環境中執行有限的 版本。

Note

此 AWS IoT Greengrass snap 適用於 AWS IoT Greengrass Core 軟體 v1.11.x. AWS IoT Greengrass doesn't，不提供 v1.10.x 的 snap。不支援的版本不會收到錯誤修正或更新。AWS IoT Greengrass snap 不支援連接器和機器學習 (ML) 推論。

如需詳細資訊，請參閱 [the section called “AWS IoT Greengrass 快速執行”](#)。

AWS IoT Greengrass Docker 軟體

AWS 提供 Dockerfile 和 Docker 映像，讓您更輕鬆地 AWS IoT Greengrass 在 Docker 容器中執行。

Dockerfile

Dockerfiles 包含用於建置自訂 AWS IoT Greengrass 容器映像的原始碼。您可以修改映像，以在不同平台架構上執行或減少映像大小。如需指示，請參閱 README 檔案。

下載您的目標 AWS IoT Greengrass Core 軟體版本。

v1.11

- 適用於 [v1 AWS IoT Greengrass .11.6 的 Dockerfile](#)。

Extended life versions

1.10 版

適用於 [v1.10 AWS IoT Greengrass .5 的 Dockerfile](#)。

1.9 版

適用於 [v1.9 AWS IoT Greengrass .4 的 Dockerfile](#)。

1.8 版

適用於 [v1.8 AWS IoT Greengrass .1 的 Dockerfile](#)。

Docker 映像檔

Docker 映像具有安裝在 Amazon Linux 2 (x86_64) 和 Alpine Linux (x86_64、Armv7l 或 AArch64) 基礎映像上的 AWS IoT Greengrass Core 軟體和相依性。您可以使用預先建立的映像來開始試驗 AWS IoT Greengrass。

Important

2022 年 6 月 30 日，發佈至 Amazon Elastic Container Registry (Amazon ECR) 和 Docker Hub 的 AWS IoT Greengrass Core 軟體 v1.x Docker 映像的 AWS IoT Greengrass 結束維護。您可以繼續從 Amazon ECR 和 Docker Hub 下載這些 Docker 映像，直到 2023 年 6 月 30 日，也就是維護結束後 1 年。不過，在 2022 年 6 月 30 日結束維護之後，AWS IoT Greengrass Core 軟體 v1.x Docker 映像將不再收到安全修補程式或錯誤修正。如果您執行依賴這些 Docker 映像的生產工作負載，建議您使用 AWS IoT Greengrass 提供的 Dockerfile 建置自己的 Docker 映像。如需詳細資訊，請參閱[AWS IoT Greengrass Version 1 維護政策](#)。

從 [Docker Hub](#) 或 Amazon Elastic Container Registry (Amazon ECR) 下載預先建置的映像。

- 對於 Docker Hub，使用 `##` 標籤下載特定版本的 Greengrass Docker 映像。若要尋找所有可用的映像，請檢查 Docker Hub 上的 Tags (標籤) 頁面。
- 對於 Amazon ECR，請使用 `latest` 標籤下載 Greengrass Docker 映像的最新可用版本。如需列出可用映像版本並從 Amazon ECR 下載映像的詳細資訊，請參閱 [AWS IoT Greengrass 在 Docker 容器中執行](#)。

 Warning

從 AWS IoT Greengrass 核心軟體的 1.11.6 版開始，Greengrass Docker 映像不再包含 Python 2.7，因為 Python 2.7 在 2020 年達到 end-of-life，而且不再收到安全性更新。如果您選擇更新這些 Docker 映像，建議您在將更新部署到生產裝置之前，先驗證您的應用程式是否使用新的 Docker 映像。如果您需要使用 Greengrass Docker 映像的應用程式使用 Python 2.7，您可以修改 Greengrass Dockerfile 以包含應用程式的 Python 2.7。

AWS IoT Greengrass 不提供 AWS IoT Greengrass Core 軟體 v1.11.1 的 Docker 映像。

 Note

依預設，alpine-aarch64 和 alpine-armv7l 映像只能在以 Arm 為基礎的主機上執行。若要在 x86 主機上執行這些映像，您可以安裝 [QEMU](#) 並在主機上安裝 QEMU 程式庫。例如：

```
docker run --rm --privileged multiarch/qemu-user-static --reset -p yes
```

AWS IoT Greengrass 核心 SDK

Lambda 函數使用 AWS IoT Greengrass 核心 SDK 在本機與 AWS IoT Greengrass 核心互動。這可讓部署的 Lambda 函數：

- 與交換 MQTT 訊息 AWS IoT Core。
- 使用 Greengrass 群組中的連接器、用戶端裝置和其他 Lambda 函數交換 MQTT 訊息。
- 與本機陰影服務互動。
- 叫用其他本機 Lambda 函數。
- 存取[私密資源](#)。
- 與[串流管理員](#)交動。

從 GitHub 下載適用於您的語言或平台 AWS IoT Greengrass 的核心 SDK。

- [AWS IoT Greengrass 適用於 Java 的核心 SDK](#)

- [AWS IoT Greengrass 適用於 Node.js 的核心 SDK](#)
- [AWS IoT Greengrass 適用於 Python 的核心 SDK](#)
- [AWS IoT Greengrass 適用於 C 的核心 SDK](#)

如需詳細資訊，請參閱[AWS IoT Greengrass 核心 SDK](#)。

支援的 Machine Learning 執行時間和程式庫

若要在 Greengrass 核心上[執行推論](#)，您必須為 ML 模型類型安裝機器學習執行階段或程式庫。

AWS IoT Greengrass 支援下列 ML 模型類型。請使用這些連結來尋找有關如何為您的模型類型和裝置平台安裝執行階段或程式庫的資訊。

- [深度學習執行時間 \(DLR\)](#)
- [MXNet](#)
- [TensorFlow](#)

機器學習範例

AWS IoT Greengrass 提供範例，您可以搭配支援的 ML 執行期 and 程式庫使用。這些範例是依據[Greengrass 核心軟體授權合約](#)所發行的。

Deep learning runtime (DLR)

下載適用於您裝置平台的範例：

- [Raspberry Pi](#) 的 DLR 範例
- [NVIDIA Jetson TX2](#) 的 DLR 範例
- [Intel Atom](#) 的 DLR 範例

如需使用 DLR 範例的教學課程，請參閱 [the section called “如何設定最佳化的機器學習推論”](#)。

MXNet

下載適用於您裝置平台的範例：

- [Raspberry Pi](#) 的 MXNet 範例

- [NVIDIA Jetson TX2](#) 的 MXNet 範例
- [Intel Atom](#) 的 MXNet 範例

如需使用 MXNet 範例的教學課程，請參閱 [the section called “如何設定機器學習推論”](#)。

TensorFlow

下載適用於您裝置平台的 [Tensorflow 範例](#)。此範例適用於 Raspberry Pi、NVIDIA Jetson TX2 與 Intel Atom。

AWS IoT Greengrass ML 開發套件軟體

[AWS IoT Greengrass Machine Learning SDK](#) 可讓您撰寫的 Lambda 函數使用本機機器學習模型，並將資料傳送至 [ML Feedback](#) 連接器以進行上傳和發佈。

v1.1.0

- [Python 3.7](#)。

v1.0.0

- [Python 2.7](#)。

我們希望傾聽您的意見

我們誠摯歡迎您提供意見回饋。若要聯絡我們，請造訪 [AWS re : Post](#) 並使用 [AWS IoT Greengrass 標籤](#)。

安裝 AWS IoT Greengrass 核心軟體

Core AWS IoT Greengrass 軟體將 AWS 功能延伸到 AWS IoT Greengrass 核心裝置，讓本機裝置可以在本機根據其產生的資料採取行動。

AWS IoT Greengrass 提供數種安裝 AWS IoT Greengrass Core 軟體的選項：

- [下載並解壓縮 tar.gz 檔案](#)。
- [執行 Greengrass 裝置設定指令碼](#)。

- [從 APT 儲存庫安裝](#)。

AWS IoT Greengrass 也提供執行 AWS IoT Greengrass Core 軟體的容器化環境。

- [AWS IoT Greengrass 在 Docker 容器中執行](#)。
- [AWS IoT Greengrass 快速執行](#)。

下載並擷取 AWS IoT Greengrass Core 軟體套件

為您的平台選擇要下載為 tar.gz 檔案並在裝置上擷取 AWS IoT Greengrass 的核心軟體。您可以下載最新版本的軟體。如需詳細資訊，請參閱[the section called “AWS IoT Greengrass 核心軟體”](#)。

執行 Greengrass 裝置設定指定碼

執行 Greengrass 裝置設定以設定您的裝置、安裝最新的 AWS IoT Greengrass Core 軟體版本，並在幾分鐘內部署 Hello World Lambda 函數。如需詳細資訊，請參閱[the section called “快速入門：Greengrass 裝置安裝”](#)。

從 APT 儲存庫安裝 AWS IoT Greengrass 核心軟體

Important

自 2022 年 2 月 11 日起，您無法再從 APT 儲存庫安裝或更新 AWS IoT Greengrass Core 軟體。在您新增 AWS IoT Greengrass 儲存庫的裝置上，您必須[從來源清單中移除儲存庫](#)。從 APT 儲存庫執行軟體的裝置將繼續正常運作。建議您使用 [tar 檔案](#) 更新 AWS IoT Greengrass Core 軟體。

提供的 APT 儲存庫 AWS IoT Greengrass 包含下列套件：

- aws-iot-greengrass-core。安裝 AWS IoT Greengrass 核心軟體。
- aws-iot-greengrass-keyring。安裝用來簽署 AWS IoT Greengrass 套件儲存庫的 GnuPG (GPG) 金鑰。

下載此軟體，即表示您同意 [Greengrass 核心軟體授權合約](#) 之規定。

主題

- [使用 systemd 指令碼以管理 Greengrass 協助程式生命週期](#)
- [使用 APT 儲存庫解除安裝 AWS IoT Greengrass 核心軟體](#)
- [移除 AWS IoT Greengrass 核心軟體儲存庫來源](#)

使用 systemd 指令碼以管理 Greengrass 協助程式生命週期

aws-iot-greengrass-core 套件也會安裝可用來管理 AWS IoT Greengrass Core 軟體（協助程式）生命週期的systemd指令碼。

- 在開機期間啟動 Greengrass 協助程式：

```
systemctl enable greengrass.service
```

- 啟動 Greengrass 協助程式：

```
systemctl start greengrass.service
```

- 停止 Greengrass 協助程式：

```
systemctl stop greengrass.service
```

- 檢查 Greengrass 協助程式的狀態：

```
systemctl status greengrass.service
```

使用 APT 儲存庫解除安裝 AWS IoT Greengrass 核心軟體

當您解除安裝 AWS IoT Greengrass 核心軟體時，您可以選擇是否要保留或移除 AWS IoT Greengrass 核心軟體的組態資訊，例如裝置憑證、群組資訊和日誌檔案。

解除安裝 AWS IoT Greengrass 核心軟體並保留組態資訊

- 執行下列命令以移除 AWS IoT Greengrass 核心軟體套件，並將組態資訊保留在 /greengrass 資料夾中。

```
sudo apt remove aws-iot-greengrass-core aws-iot-greengrass-keyring
```

解除安裝 AWS IoT Greengrass 核心軟體並移除組態資訊

1. 執行下列命令以移除 AWS IoT Greengrass 核心軟體套件，並從 移除組態資訊/greengrass folder。

```
sudo apt purge aws-iot-greengrass-core aws-iot-greengrass-keyring
```

2. 從來源清單中移除 AWS IoT Greengrass 核心軟體儲存庫。如需詳細資訊，請參閱[移除 AWS IoT Greengrass 核心軟體儲存庫來源](#)。

移除 AWS IoT Greengrass 核心軟體儲存庫來源

當您不再需要從 APT 儲存庫安裝或更新 AWS IoT Greengrass 核心軟體時，您可以移除 AWS IoT Greengrass 核心軟體儲存庫來源。2022 年 2 月 11 日之後，您必須從來源清單中移除儲存庫，以避免執行時發生錯誤 `apt update`。

從來源清單中移除 APT 儲存庫

- 執行下列命令，從來源清單中移除 AWS IoT Greengrass 核心軟體儲存庫。

```
sudo rm /etc/apt/sources.list.d/greengrass.list  
sudo apt update
```

AWS IoT Greengrass 在 Docker 容器中執行

AWS IoT Greengrass 提供 Dockerfile 和 Docker 映像，讓您更輕鬆地在 Docker 容器中執行 AWS IoT Greengrass Core 軟體。如需詳細資訊，請參閱[the section called “AWS IoT Greengrass Docker 軟體”](#)。

Note

您也可以在 Greengrass 核心裝置上執行 Docker 應用程式。為此，使用 [Greengrass Docker 應用程式部署連接器](#)。

AWS IoT Greengrass 快速執行

AWS IoT Greengrass snap 1.11.x 可讓您 AWS IoT Greengrass 透過方便的軟體套件以及所有必要的相依性，在容器化環境中執行有限的 版本。

在 2023 年 12 月 31 日，AWS IoT Greengrass 將結束在 <https://> 上發佈 AWS IoT Greengrass 的核心軟體版本 1.11.x Snap 的維護。 snapcraft.io 目前執行 Snap 的裝置將繼續運作，直到進一步通知為止。不過，AWS IoT Greengrass 核心 Snap 在維護結束後將不再收到安全修補程式或錯誤修正。

快照概念

以下是基本的貼齊概念，可協助您了解如何使用貼 AWS IoT Greengrass 齊：

[Channel](#)

快照元件，定義要安裝和追蹤哪個版本的快照以進行更新。快照會自動更新至目前頻道的最新版本。

[介面](#)

授予資源存取權的快照元件，例如網路和使用者檔案。

若要執行 AWS IoT Greengrass Snap，必須連接下列介面。請注意，greengrass-support-no-container 必須先連線，且絕不會中斷連線。

- **greengrass-support-no-container**
- hardware-observe
- home-for-hooks
- hugepages-control
- log-observe
- mount-observe
- network
- network-bind
- network-control
- process-control
- system-observe

其他介面是選用的。如果您的 Lambda 函數需要存取特定資源，您可能需要連線到適當的介面。

重新整理

快照會自動更新。snapd 協助程式是 snap 套件管理員，預設會每天檢查更新四次。每個更新檢查稱為重新整理。重新整理發生時，協助程式會停止、快照會更新，然後協助程式會重新啟動。

如需詳細資訊，請參閱 [Snapcraft](#) 網站。

snap v1 AWS IoT Greengrass .11.x 的新功能

以下說明 1.11.x 版 AWS IoT Greengrass Snap 的新功能和變更。

- 此版本僅支援 snap_daemon 以使用者 ID (UID) 和群組 (GID) 公開的使用者 584788。
- 此版本僅支援非容器化 Lambda 函數。

Important

由於非容器化 Lambda 函數必須共用相同的使用者 (snap_daemon)，因此 Lambda 函數沒有彼此隔離。如需詳細資訊，請參閱 [使用群組特定組態控制 Greengrass Lambda 函數的執行](#)。

- 此版本支援 C、C++、Java 8、Node.js 12.x、Python 2.7、Python 3.7 和 Python 3.8 執行時間。

Note

為了避免多餘的 Python 執行期，Python 3.7 Lambda 函數實際上會執行 Python 3.8 執行期。

開始使用 AWS IoT Greengrass snap

下列程序可協助您在裝置上安裝和設定 AWS IoT Greengrass Snap。

要求

若要執行 AWS IoT Greengrass Snap，您必須執行下列動作：

- 在支援的 Linux 發行版本上執行 AWS IoT Greengrass Snap，例如 Ubuntu、Linux Mint、Debian 和 Fedora。
- 在您的裝置上安裝 snapd 協助程式。包含 snapd snap 工具的協助程式會管理裝置上的快照環境。

如需支援的 Linux 發行版本和安裝說明清單，請參閱 Snap 文件中的[安裝快照](#)。

安裝和設定 AWS IoT Greengrass Snap

下列教學課程說明如何在裝置上安裝和設定 AWS IoT Greengrass Snap。

Note

- 雖然本教學課程使用 Amazon EC2 執行個體 (x86 t2.micro Ubuntu 20.04)，但您可以使用實體硬體執行 AWS IoT Greengrass 快照，例如 Raspberry Pi。
- snapd 協助程式預先安裝在 Ubuntu 上。

1. 在裝置的終端機中執行下列命令來安裝 core18 Snap：

```
sudo snap install core18
```

core18 快照是[基本快照](#)，提供具有常用程式庫的執行時間環境。此快照是從 [Ubuntu 18.04 LTS 建置而成](#)。

2. 執行下列命令snapd進行升級：

```
sudo snap install --channel=edge snapd; sudo snap refresh --channel=edge snapd
```

3. 執行 snap list 命令來檢查是否已安裝 AWS IoT Greengrass Snap。

下列範例回應顯示 snapd 已安裝，但未安裝aws-iot-greengrass。

Name	Version	Rev	Tracking	Publisher	Notes
amazon-ssm-agent	3.0.161.0	2996	latest/stable/...	aws#	classic
core	16-2.48	10444	latest/stable	canonical#	core
core18	20200929	1932	latest/stable	canonical#	base
lxd	4.0.4	18150	4.0/stable/...	canonical#	-
snapd	2.48+git548.g929ccfb	10526	latest/edge	canonical#	snapd

4. 選擇下列其中一個選項來安裝 AWS IoT Greengrass snap 1.11.x。

- 若要安裝 AWS IoT Greengrass snap，請執行下列命令：

```
sudo snap install aws-iot-greengrass
```

回應範例：

```
aws-iot-greengrass 1.11.5 from Amazon Web Services (aws) installed
```

- 若要從舊版遷移至 v1.11.x 或更新至最新的可用修補程式版本，請執行下列命令：

```
sudo snap refresh --channel=1.11.x aws-iot-greengrass
```

如同其他快照，該 AWS IoT Greengrass 快照會使用頻道來管理次要版本。快照會自動更新為目前頻道的最新可用版本。例如，如果您指定 `--channel=1.11.x`，您的 AWS IoT Greengrass snap 會更新為 v1.11.5。

您可以執行 `snap info aws-iot-greengrass` 命令以取得可用頻道的清單 AWS IoT Greengrass。

回應範例：

```
name:      aws-iot-greengrass
summary:   AWS supported software that extends cloud capabilities to local devices.
publisher: Amazon Web Services (aws#)
store-url: https://snapcraft.io/aws-iot-greengrass
contact:   https://repost.aws/tags/TA4ckIed1sR4enZBey29rKTg/aws-io-t-greengrass
license:   Proprietary
description: |
  AWS IoT Greengrass seamlessly extends AWS onto edge devices so they can act
  locally on the data
  they generate, while still using the cloud for management, analytics, and durable
  storage.
  AWS IoT Greengrass snap v1.11.0 enables you to run a limited version of AWS IoT
  Greengrass with
  all necessary dependencies in a containerized environment.
  The AWS IoT Greengrass snap doesn't support connectors and machine learning (ML)
  inference.
  By downloading this software you agree to the Greengrass Core Software License
  Agreement
  (https://s3-us-west-2.amazonaws.com/greengrass-release-license/greengrass-
  license-v1.pdf).
  For more information, see Run AWS IoT Greengrass in a snap
  (https://docs.aws.amazon.com/greengrass/latest/developerguide/install-
  gg.html#gg-snap-support) in
```

```

the AWS IoT Greengrass Developer.
If you need help, try the AWS IoT Greengrass tag on AWS re:Post
(https://repost.aws/tags/TA4ckIed1sR4enZBey29rKTg/aws-io-t-greengrass) or connect
with an AWS IQ expert
(https://iq.aws.amazon.com/services/aws/greengrass).
snap-id: SRDuhPJGj4XPxFNNZQK0TvURAp0wxKnd
channels:
  latest/stable: 1.11.3 2021-06-15 (59) 111MB -
  latest/candidate: 1.11.3 2021-06-14 (59) 111MB -
  latest/beta: 1.11.3 2021-06-14 (59) 111MB -
  latest/edge: 1.11.3 2021-06-14 (59) 111MB -
  1.11.x/stable: 1.11.3 2021-06-15 (59) 111MB -
  1.11.x/candidate: 1.11.3 2021-06-15 (59) 111MB -
  1.11.x/beta: 1.11.3 2021-06-15 (59) 111MB -
  1.11.x/edge: 1.11.3 2021-06-15 (59) 111MB -

```

5. 若要存取 Lambda 函數所需的特定資源，您可以連線至其他介面。

執行下列命令以取得 AWS IoT Greengrass 快照支援的介面清單：

```
snap connections aws-iot-greengrass
```

回應範例：

Interface	Notes	Plug	Slot
camera	-	aws-iot-greengrass:camera	-
dvb	-	aws-iot-greengrass:dvb	-
gpio	-	aws-iot-greengrass:gpio	-
gpio-memory-control	-	aws-iot-greengrass:gpio-memory-control	-
greengrass-support	-	aws-iot-greengrass:greengrass-support-no-container	-
:greengrass-support	-	aws-iot-greengrass:greengrass-support-no-container	-
hardware-observe	-	aws-iot-greengrass:hardware-observe	-
:hardware-observe	-	aws-iot-greengrass:hardware-observe	-
hardware-random-control	-	aws-iot-greengrass:hardware-random-control	-
home	-	aws-iot-greengrass:home-for-greengrassd	-

home		aws-iot-greengrass:home-for-hooks	:home
	manual		
hugepages-control		aws-iot-greengrass:hugepages-control	
:hugepages-control	manual		
i2c		aws-iot-greengrass:i2c	-
	-		
iio		aws-iot-greengrass:iio	-
	-		
joystick		aws-iot-greengrass:joystick	-
	-		
log-observe		aws-iot-greengrass:log-observe	:log-
observe	manual		
mount-observe		aws-iot-greengrass:mount-observe	
:mount-observe	manual		
network		aws-iot-greengrass:network	
:network	-		
network-bind		aws-iot-greengrass:network-bind	
:network-bind	-		
network-control		aws-iot-greengrass:network-control	
:network-control	-		
opengl		aws-iot-greengrass:opengl	
:opengl	-		
optical-drive		aws-iot-greengrass:optical-drive	
:optical-drive	-		
process-control		aws-iot-greengrass:process-control	
:process-control	-		
raw-usb		aws-iot-greengrass:raw-usb	-
	-		
removable-media		aws-iot-greengrass:removable-media	-
	-		
serial-port		aws-iot-greengrass:serial-port	-
	-		
spi		aws-iot-greengrass:spi	-
	-		
system-observe		aws-iot-greengrass:system-observe	
:system-observe	-		

如果您在插槽欄中看到連字號 (-)，則不會連接對應的界面。

6. 遵循[安裝 AWS IoT Greengrass Core 軟體](#)來建立 AWS IoT 物件、Greengrass 群組、啟用安全通訊的安全資源 AWS IoT，以及 AWS IoT Greengrass Core 軟體組態檔案。組態檔案 config.json 包含 Greengrass 核心特定的組態，例如憑證檔案和 AWS IoT 裝置資料端點的位置。

Note

如果您將檔案下載到不同的裝置，請依照[此步驟](#)將檔案傳輸至 AWS IoT Greengrass 核心裝置。

7. 對於 AWS IoT Greengrass Snap，請確定您更新 [config.json](#) 檔案，如下所示：

- 將 *certificateId* 的每個執行個體取代為憑證和金鑰檔案名稱中的憑證 ID。
- 如果您下載的 Amazon 根 CA 憑證與 Amazon 根 CA 1 不同，請將 *AmazonRootCA1.pem* 的每個執行個體取代為 Amazon 根 CA 檔案的名稱。

```
{
  ...
  "crypto" : {
    "principals" : {
      "SecretsManager" : {
        "privateKeyPath" : "file:///snap/aws-iot-greengrass/current/greengrass/
certs/certificateId-private.pem.key"
      },
      "IoTCertificate" : {
        "privateKeyPath" : "file:///snap/aws-iot-greengrass/current/greengrass/
certs/certificateId-private.pem.key",
        "certificatePath" : "file:///snap/aws-iot-greengrass/current/greengrass/
certs/certificateId-certificate.pem.crt"
      }
    },
    "caPath" : "file:///snap/aws-iot-greengrass/current/greengrass/
certs/AmazonRootCA1.pem"
  },
  "writeDirectory": "/var/snap/aws-iot-greengrass/current/ggc-write-directory",
  "pidFileDirectory": "/var/snap/aws-iot-greengrass/current/pidFileDirectory"
}
```

8. 執行下列命令來新增您的 AWS IoT Greengrass 憑證和組態檔案：

```
sudo snap set aws-iot-greengrass gg-certs=/home/ubuntu/my-certs
```

部署 Lambda 函數

本節說明如何在 AWS IoT Greengrass Snap 上部署客戶受管 Lambda 函數。

Important

AWS IoT Greengrass snap v1.11 僅支援非容器化 Lambda 函數。

1. 執行下列命令來啟動 AWS IoT Greengrass 協助程式：

```
sudo snap start aws-iot-greengrass
```

回應範例：

```
Started.
```

Note

如果發生錯誤，您可以使用 `snap run` 命令取得詳細的錯誤訊息。如需更多故障診斷資訊，請參閱 [錯誤：無法執行下列任務：- 執行 `snap "aws-iot-greengrass" \(【start snap.aws-iot-greengrass.greengrassd.service】的服務【"greengrassd" 啟動服務命令 "start"，結束狀態為 1 的【start snap.aws-iot-greengrass.greengrassd.service】失敗：由於控制程序以錯誤碼結束。如需詳細資訊，請參閱「systemctl status snap.aws-iot-greengrass.greengrassd.service」和「journalctl -xe」。`】\)](#)。

2. 執行下列命令以確認協助程式正在執行：

```
snap services aws-iot-greengrass.greengrassd
```

回應範例：

Service	Startup	Current	Notes
aws-iot-greengrass.greengrassd	disabled	active	-

3. 遵循[模組 3（第 1 部分）：上的 Lambda 函數 AWS IoT Greengrass](#)來建立和部署 Hello World Lambda 函數。不過，在您部署 Lambda 函數之前，請先完成下一個步驟。

4. 請確定您的 Lambda 函數以 snap_daemon 使用者身分執行，且處於無容器模式。若要更新 Greengrass 群組的設定，請在 AWS IoT Greengrass 主控台中執行下列動作：
 - a. 登入 AWS IoT Greengrass 主控台。
 - b. 在 AWS IoT 主控台導覽窗格的管理下，展開 Greengrass 裝置，然後選擇群組 (V1)。
 - c. 在 Greengrass 群組下，選擇目標群組。
 - d. 在群組組態頁面的導覽窗格中，選擇 Lambda 函數索引標籤。
 - e. 在預設 Lambda 函數執行期環境中，選擇編輯，然後執行下列動作：
 - i. 對於預設系統使用者和群組，選擇另一個使用者 ID/群組 ID，然後 **584788** 針對系統使用者 ID（數字）和系統群組 ID（數字）輸入。
 - ii. 針對預設 Lambda 函數容器化，選擇無容器。
 - iii. 選擇儲存。

停止 AWS IoT Greengrass 協助程式

您可以使用 `snap stop` 命令來停止服務。

若要停止 AWS IoT Greengrass 協助程式，請執行下列命令：

```
sudo snap stop aws-iot-greengrass
```

命令應傳回 `Stopped.`。

若要檢查您是否已成功停止 snap，請執行下列命令：

```
snap services aws-iot-greengrass.greengrassd
```

回應範例：

Service	Startup	Current	Notes
aws-iot-greengrass.greengrassd	disabled	inactive	-

解除安裝 AWS IoT Greengrass Snap

若要解除安裝 AWS IoT Greengrass snap，請執行下列命令：

```
sudo snap remove aws-iot-greengrass
```

回應範例：

```
aws-iot-greengrass removed
```

故障診斷 AWS IoT Greengrass Snap

使用以下資訊來協助故障診斷 AWS IoT Greengrass Snap 的問題。

取得許可拒絕錯誤。

解決方案：許可遭拒錯誤通常是因為缺少界面。如需缺少界面的清單和詳細的故障診斷資訊，您可以使用 `snappy-debug` 工具。

執行下列命令來安裝 工具。

```
sudo snap install snappy-debug
```

回應範例：

```
snappy-debug 0.36-snapd2.45.1 from Canonical# installed
```

在個別終端機工作階段中執行 `sudo snappy-debug` 命令。操作會持續進行，直到發生許可遭拒錯誤為止。

例如，如果您的 Lambda 函數嘗試讀取 `$HOME` 目錄中的檔案，您可能會收到下列回應：

```
INFO: Following '/var/log/syslog'. If have dropped messages, use:
INFO: $ sudo journalctl --output=short --follow --all | sudo snappy-debug
kernel.printk_ratelimit = 0
= AppArmor =
Time: Dec 6 04:48:26
Log: apparmor="DENIED" operation="mknod" profile="snap.aws-iot-greengrass.greengrassd"
     name="/home/ubuntu/my-file.txt" pid=12345 comm="touch" requested_mask="c"
     denied_mask="c" fsuid=0 ouid=0
File: /home/ubuntu/my-file.txt (write)
Suggestion:
* add 'home' to 'plugins'
```

此範例顯示建立 `/home/ubuntu/my-file.txt` 檔案會導致許可錯誤。它也建議您將 `home` 新增至 `plugins`。不過，此語句不適用。`home-for-greengrassd` 和 `home-for-hooks` 插頭只會獲得唯讀存取權。

如需詳細資訊，請參閱 [Snap 文件中的 Snappy-debug](#) snap。

錯誤：無法執行下列任務：- 執行 snap "aws-iot-greengrass" (【start snap.aws-iot-greengrass.greengrassd.service】的服務【"greengrassd" 啟動服務命令 "start"，結束狀態為 1 的【start snap.aws-iot-greengrass.greengrassd.service】失敗：由於控制程序以錯誤碼結束。如需詳細資訊，請參閱「systemctl status snap.aws-iot-greengrass.greengrassd.service」和「journalctl -xe」。))

解決方案：當 snap start aws-iot-greengrass 命令無法啟動 AWS IoT Greengrass Core 軟體時，您可能會看到此錯誤。

如需更多疑難排解資訊，請執行下列命令：

```
sudo snap run aws-iot-greengrass.greengrassd
```

回應範例：

```
Couldn't find /snap/aws-iot-greengrass/44/greengrass/config/config.json.
```

此範例顯示 AWS IoT Greengrass 找不到 config.json 檔案。您可以檢查組態和憑證檔案。

/var/snap/aws-iot-greengrass/current/ggc-write-directory/packages/1.11.5/rootfs/merged 不是絕對路徑或是符號連結。

解決方案：AWS IoT Greengrass Snap 僅支援非容器化 Lambda 函數。請確定您在無容器模式下執行 Lambda 函數。如需詳細資訊，請參閱《AWS IoT Greengrass Version 1 開發人員指南》中的[選擇 Lambda 函數容器化時的考量](#)事項。

在您執行 sudo snap refresh snapd 命令後，Snapd 協助程式無法重新啟動。

解決方案：遵循 中的步驟 6 到 8 [安裝和設定 AWS IoT Greengrass Snap](#)，將 AWS IoT Greengrass 憑證和組態檔案新增至 AWS IoT Greengrass 快照。

封存 AWS IoT Greengrass Core 軟體安裝

升級到新版本的 AWS IoT Greengrass Core 軟體時，您可以封存目前安裝的版本。這會保留您目前的安裝環境，讓您可以在相同硬體上測試新的軟體版本。這也可讓您隨時輕鬆轉返至封存版本。

封存目前安裝並安裝新版本

1. 下載您想要升級至的 [AWS IoT Greengrass 核心軟體](#) 安裝套件。

2. 將套件複製到目的地核心裝置。如需示範傳輸檔案的說明，請參閱此[步驟](#)。

 Note

稍後請將您目前的憑證、金鑰和組態檔案複製到新的安裝。

在您的核心裝置終端機以下列步驟執行命令。

3. 請確認 Greengrass 協助程式在核心裝置上已停止。

- a. 檢查精靈是否有在運作：

```
ps aux | grep -E 'greengrass.*daemon'
```

若輸出的 root 含有 `/greengrass/ggc/packages/ggc-version/bin/daemon` 項目，則精靈有在運作。

 Note

編寫此程序時假設 AWS IoT Greengrass Core 軟體已安裝在 `/greengrass` 目錄中。

- b. 停止 協助程式：

```
cd /greengrass/ggc/core/  
sudo ./greengrassd stop
```

4. 將目前的 Greengrass 根目錄移到不同的目錄。

```
sudo mv /greengrass /greengrass_backup
```

5. 在核心裝置上將新的軟體解壓縮。取代命令中的 `os-architecture` 和 `version` 預留位置。

```
sudo tar -zxvf greengrass-os-architecture-version.tar.gz -C /
```

6. 將封存的憑證、金鑰和組態檔案複製到新的安裝。

```
sudo cp /greengrass_backup/certs/* /greengrass/certs  
sudo cp /greengrass_backup/config/* /greengrass/config
```

7. 啟動協助程式：

```
cd /greengrass/ggc/core/  
sudo ./greengrassd start
```

現在，您可以執行群組部署以測試新的安裝。如果發生故障，您可以還原封存的安裝。

還原封存的安裝

1. 停止協助程式。
2. 刪除新的 /greengrass 目錄。
3. 將 /greengrass_backup 目錄移回 /greengrass。
4. 啟動協助程式。

設定 AWS IoT Greengrass 核心

AWS IoT Greengrass 核心是做為節點環境中中樞或閘道的 AWS IoT 物件（裝置）。如同其他 AWS IoT 裝置，核心存在於登錄檔中，具有裝置影子，並使用裝置憑證向 AWS IoT Core 和進行身分驗證 AWS IoT Greengrass。核心裝置執行 AWS IoT Greengrass 核心軟體，利其管理 Greengrass 群組的本機步驟，例如，通訊、陰影同步和字符交換。

Core AWS IoT Greengrass 軟體提供下列功能：

- 部署和本機執行連接器和 Lambda 函數。
- 在本機使用自動匯出至來處理資料串流 AWS 雲端。
- 使用受管訂閱，透過裝置、連接器和 Lambda 函數之間的本機網路傳送 MQTT 訊息。
- 使用受管訂閱的 AWS IoT 和裝置、連接器和 Lambda 函數之間的 MQTT 訊息。
- AWS 雲端使用裝置身分驗證和授權保護裝置與之間的連線。
- 裝置的本機陰影同步。陰影可設定為與同步 AWS 雲端。
- 對本機裝置和磁碟區資源控制的存取。
- 用於執行本機推論的雲端訓練機器學習模型部署。
- 可讓裝置探索 Greengrass 核心裝置的自動 IP 地址偵測。
- 新的或更新的群組組態的集中部署。下載組態資料之後，核心裝置會自動重新啟動。
- 使用者定義 Lambda 函數的安全 over-the-air (OTA) 軟體更新。
- 本機秘密的安全加密儲存，以及連接器和 Lambda 函數控制的存取。

AWS IoT Greengrass 核心組態檔案

Core AWS IoT Greengrass 軟體的組態檔案為 `config.json`。它會置放於 `/greengrass-root/config` 目錄中。

Note

`greengrass-root` 代表 Core AWS IoT Greengrass 軟體安裝在您裝置上的路徑。通常，這是 `/greengrass` 目錄。

如果您從 AWS IoT Greengrass 主控台使用預設群組建立選項，則 `config.json` 檔案會部署到運作狀態的核心裝置。

您可以執行下列命令檢閱此檔案內容：

```
cat /greengrass-root/config/config.json
```

以下是範例 `config.json` 檔案。這是從 AWS IoT Greengrass 主控台建立核心時產生的版本。

GGC v1.11

```
{
  "coreThing": {
    "caPath": "root.ca.pem",
    "certPath": "hash.cert.pem",
    "keyPath": "hash.private.key",
    "thingArn": "arn:partition:iot:region:account-id:thing/core-thing-name",
    "iotHost": "host-prefix-ats.iot.region.amazonaws.com",
    "ggHost": "greengrass-ats.iot.region.amazonaws.com",
    "keepAlive": 600,
    "ggDaemonPort": 8000,
    "systemComponentAuthTimeout": 5000
  },
  "runtime": {
    "maxWorkItemCount": 1024,
    "maxConcurrentLimit": 25,
    "lruSize": 25,
    "mountAllBlockDevices": "no",
    "cgroup": {
      "useSystemd": "yes"
    }
  }
}
```

```
{,
  "managedRespawn": false,
  "crypto": {
    "principals": {
      "SecretsManager": {
        "privateKeyPath": "file:///greengrass/certs/hash.private.key"
      },
      "IoTCertificate": {
        "privateKeyPath": "file:///greengrass/certs/hash.private.key",
        "certificatePath": "file:///greengrass/certs/hash.cert.pem"
      }
    },
    "caPath": "file:///greengrass/certs/root.ca.pem"
  },
  "writeDirectory": "/var/snap/aws-iot-greengrass/current/ggc-write-directory",
  "pidFileDirectory": "/var/snap/aws-iot-greengrass/current/pidFileDirectory"
}
```

config.json 檔案支援以下屬性：

coreThing

欄位	Description	備註
caPath	AWS IoT 根 CA 相對於 <i>/greengrass-root</i> / certs 目錄的路徑。	用於回溯相容於 1.7.0 之前的版本。當crypto物件存在時，會忽略此屬性。  Note 請確定您的端點對應至您的憑證類型。
certPath	相對於 <i>/greengrass-root</i> /certs 目錄的核心裝置憑證路徑。	用於回溯相容於 1.7.0 之前的版本。當crypto物件存在時，會忽略此屬性。
keyPath	相對於 <i>/greengrass-root</i> /certs 目錄的核心私有金鑰路徑。	用於回溯相容於 1.7.0 之前的版本。當crypto物件存在時，會忽略此屬性。

欄位	Description	備註
thingArn	代表 AWS IoT Greengrass 核心裝置的 AWS IoT 物件的 Amazon Resource Name (ARN)。	在 AWS IoT Greengrass 主控台的 核心下尋找核心的 ARN，或執行 CLI aws greengrass get-core-definition-version 命令。
iotHost	您的 AWS IoT 端點。	在主控台的設定 AWS IoT 下或透過執行 CLI aws iot describe-endpoint --endpoint-type iot:Data-ATS 命令來尋找端點。 此命令傳回 Amazon Trust Services (ATS) 端點。如需詳細資訊，請參閱伺服器身分驗證文件。  Note 請確定您的端點對應至您的憑證類型。 請確定您的端點與您的對應 AWS 區域。

欄位	Description	備註
ggHost	您的 AWS IoT Greengrass 端點。	這是您的 iotHost 端點，其具有的主機字首會由 greengrass 取代 (例如 greengrass-ats.iot. <i>region</i>.amazonaws.com)。使用與 AWS 區域相同的 iotHost。  Note 請確定您的端點對應至您的憑證類型。 請確定您的端點對應至您的 AWS 區域。
iotMqttPort	選用。用於 MQTT 通訊的連接埠號碼 AWS IoT。	有效值為 8883 或 443。預設值為 8883。如需詳細資訊，請參閱 連線至連接埠 443 或透過網路代理 。
iotHttpPort	選用。用於建立到 AWS IoT 之 HTTPS 連線的連接埠號碼。	有效值為 8443 或 443。預設值為 8443。如需詳細資訊，請參閱 連線至連接埠 443 或透過網路代理 。
ggMqttPort	選用。用於透過區域網路進行 MQTT 通訊的連接埠號碼。	有效值為 1024 到 65535。預設值為 8883。如需詳細資訊，請參閱 the section called “用於本機訊息的 MQTT 連接埠” 。
ggHttpPort	選用。用於建立到 AWS IoT Greengrass 服務之 HTTPS 連線的連接埠號碼。	有效值為 8443 或 443。預設值為 8443。如需詳細資訊，請參閱 連線至連接埠 443 或透過網路代理 。

欄位	Description	備註
keepAlive	選用。MQTT KeepAlive 期間 (以秒為單位)。	有效範圍介於 30 與 1200 秒之間。預設值為 600。
networkProxy	選用。定義要連線代理伺服器的物件。	代理伺服器可以是 HTTP 或 HTTPS。如需詳細資訊，請參閱 連線至連接埠 443 或透過網路代理 。
mqttOperationTimeout	選用。在與 AWS IoT Core 的 MQTT 連線中，允許 Greengrass 核心完成發佈、訂閱或取消訂閱操作的時間量 (以秒為單位)。	預設值為 5。最小值為 5。
ggDaemonPort	選用。Greengrass 核心 IPC 連接埠號碼。	此屬性可在 AWS IoT Greengrass v1.11.0 或更新版本中使用。 有效值介於 1024 和 65535 之間。預設值為 8000。
systemComponentAuthTimeout	選用。允許 Greengrass 核心 IPC 完成身分驗證的時間 (以毫秒為單位)。	此屬性可在 AWS IoT Greengrass v1.11.0 或更新版本中使用。 有效值介於 500 到 5000 之間。預設值為 5000。

runtime

欄位	Description	備註
maxWorkItemCount	選用。Greengrass 協助程式一次可以處理的工作項目數量上限。超過此限制的工作項目會被忽略。	預設值為 1024。該最大值受裝置硬體的限制。

欄位	Description	備註
	工作項目佇列由系統元件、使用者定義的 Lambda 函數和連接器共用。	增加此值會增加 AWS IoT Greengrass 使用的記憶體。如果您預期核心會接收大量的 MQTT 訊息流量，則可以提高此值。
<code>maxConcurrentLimit</code>	選用。Greengrass 協助程式可以擁有的並行未鎖定 Lambda 工作者數量上限。您可以指定不同的整數來覆寫此參數。	預設值為 25。最小值由定義 <code>lruSize</code> 。
<code>lruSize</code>	Optional. Defines the minimum value for <code>maxConcurrentLimit</code> .	The default value is 25.
<code>mountAllBlockDevices</code>	Optional. Enables AWS IoT Greengrass to use bind mounts to mount all block devices into a container after setting up the OverlayFS.	此屬性可在 AWS IoT Greengrass v1.11.0 或更新版本中使用。 有效值為 <code>yes</code> 和 <code>no</code> 。預設值為 <code>no</code> 。 <code>yes</code> 如果您的 <code>/usr</code> 目錄不在 <code>/階層</code> 下，請將此值設定為 <code>yes</code> 。
<code>postStartHealthCheckTimeout</code>	Optional. The time (in milliseconds) after starting that the Greengrass daemon waits for the health check to finish.	The default timeout is 30 seconds (30000 ms).
<code>cgroup</code>		

欄位	Description	備註
useSystemd	Indicates whether your device uses systemd .	Valid values are # or #. Run the <code>check_ggc_dependencies</code> script in 模組 1 to see if your device uses systemd.

加密

crypto 包含支援透過 PKCS#11 和本機秘密儲存在硬體安全模組 (HSM) 上儲存私密金鑰的屬性。如需詳細資訊，請參閱[the section called “安全性主體”](#)、[the section called “硬體安全整合”](#)及[將私密部署至核心](#)。支援 HSM 上或檔案系統中私有金鑰儲存的組態。

欄位	Description	備註
caPath	AWS IoT 根 CA 的絕對路徑。	必須為此格式的檔案 URI： <code>file:///absolute/path/to/file</code> 。

Note

請確定您的端點對應至您的憑證類型。

PKCS11

OpenSSLEngine	選用。OpenSSL 引擎 .so 檔案的絕對路徑，此檔案可在 OpenSSL 上啟用 PKCS#11 支援。	必須是檔案系統上的檔案路徑。 如果您使用具有硬體安全性的 Greengrass OTA 更新代理程式，則需要此屬性。如需詳細資訊，請參閱 the section called “設定 OTA 更新” 。
P11Provider	PKCS#11 實作的 libdl-loadable 程式庫的絕對路徑。	必須是檔案系統上的檔案路徑。

欄位	Description	備註
slotLabel	用於識別硬體模組的插槽標籤。	必須符合 PKCS#11 標籤規格。
slotUserPin	用來向模組驗證 Greengrass 核心的使用者 PIN 碼。	必須具有足夠的許可，才能使用設定的私有金鑰執行 C_Sign。
principals		
IoTCertificate	The certificate and private key that the core uses to make requests to AWS IoT.	
IoTCertificate .privateKeyPath	核心私有金鑰的路徑。	如為檔案系統儲存，必須為此格式的檔案 URI : file:///absolute/path/to/file 。 如為 HSM 儲存，必須為指定物件標籤的 RFC 7512 PKCS # 11 路徑。
IoTCertificate .certificatePath	核心裝置憑證的絕對路徑。	必須為此格式的檔案 URI : file:///absolute/path/to/file 。
MQTTServerCertificate	選用。核心用於結合憑證做為 MQTT 伺服器或閘道的私有金鑰。	

欄位	Description	備註
MQTTServerCertificate.privateKeyPath	本機 MQTT 伺服器私有金鑰的路徑。	使用此值，為本機 MQTT 伺服器指定您自己的私有金鑰。 如為檔案系統儲存，必須為此格式的檔案 URI : file:///absolute/path/to/file 。 如為 HSM 儲存，必須為指定物件標籤的 RFC 7512 PKCS # 11 路徑。 如果省略此屬性，會根據輪換設定 AWS IoT Greengrass 輪換金鑰。若有指定，則由客戶負責輪換金鑰。
SecretsManager	The private key that secures the data key used for encryption. For more information, see 將私密部署至核心 .	
SecretsManager.privateKeyPath	本機 Secrets Manager 私有金鑰的路徑。	僅支援 RSA 金鑰。 如為檔案系統儲存，必須為此格式的檔案 URI : file:///absolute/path/to/file 。 如為 HSM 儲存，必須為指定物件標籤的 RFC 7512 PKCS # 11 路徑。必須使用 PKCS#1 v1.5 填補機制產生私密金鑰。

也支援下列組態屬性：

欄位	Description	備註
mqttMaxConnectionRetryInterval	選用。連線中斷時，MQTT 連線重試的間隔上限 (以秒為單位)。	請指定此數值做為未簽署的整數。預設值為 60。
managedRespawn	選用。指出 OTA 代理程式在更新之前需要執行自訂程式碼。	有效值為 true 或 false。如需詳細資訊，請參閱 AWS IoT Greengrass Core 軟體的 OTA 更新 。
writeDirectory	選用。其中 AWS IoT Greengrass 建立所有讀取/寫入資源的寫入目錄。	如需詳細資訊，請參閱 設定的寫入目錄 AWS IoT Greengrass 。
pidFileDirectory	Optional. 會將其程序 ID (PID) AWS IoT Greengrass 存放在此目錄下。	預設值為 /var/run。

Extended life versions

下列版本的 AWS IoT Greengrass Core 軟體處於[延長生命週期階段](#)。此資訊僅供參考之用。

GGC 1.10 版

```
{
  "coreThing" : {
    "caPath" : "root.ca.pem",
    "certPath" : "hash.cert.pem",
    "keyPath" : "hash.private.key",
    "thingArn" : "arn:partition:iot:region:account-id:thing/core-thing-name",
    "iotHost" : "host-prefix-ats.iot.region.amazonaws.com",
    "ggHost" : "greengrass-ats.iot.region.amazonaws.com",
    "keepAlive" : 600,
    "systemComponentAuthTimeout": 5000
  },
  "runtime" : {
    "maxWorkItemCount" : 1024,
    "maxConcurrentLimit" : 25,
    "lruSize": 25,
  }
}
```

```
"cgroup" : {
  "useSystemd" : "yes"
},
"managedRespawn" : false,
"crypto" : {
  "principals" : {
    "SecretsManager" : {
      "privateKeyPath" : "file:///greengrass/certs/hash.private.key"
    },
    "IoTCertificate" : {
      "privateKeyPath" : "file:///greengrass/certs/hash.private.key",
      "certificatePath" : "file:///greengrass/certs/hash.cert.pem"
    }
  },
  "caPath" : "file:///greengrass/certs/root.ca.pem"
}
```

config.json 檔案支援以下屬性：

coreThing

欄位	Description	備註
caPath	AWS IoT 根 CA 相對於 <i>/greengrass-root</i> / certs 目錄的路徑。	用於回溯相容於 1.7.0 之前的版本。當crypto物件存在時，會忽略此屬性。  Note 請確定您的端點對應至您的憑證類型。
certPath	相對於 <i>/greengrass-root</i> /certs 目錄的核心裝置憑證路徑。	用於回溯相容於 1.7.0 之前的版本。當crypto物件存在時，會忽略此屬性。

欄位	Description	備註
keyPath	相對於 <i>/greengrass-root/certs</i> 目錄的核心私有金鑰路徑。	用於回溯相容於 1.7.0 之前的版本。當crypto物件存在時，會忽略此屬性。
thingArn	代表 AWS IoT Greengrass 核心裝置的 AWS IoT 物件的 Amazon Resource Name (ARN)。	在 AWS IoT Greengrass 主控台的 核心下尋找核心的 ARN，或執行 CLI aws greengrass get-core-definition-version 命令。
iotHost	您的 AWS IoT 端點。	在 AWS IoT 主控台的設定下或透過執行 CLI aws iot describe-endpoint --endpoint-type iot:Data-ATS 命令來尋找端點。 此命令傳回 Amazon Trust Services (ATS) 端點。如需詳細資訊，請參閱伺服器身分驗證文件。  Note 請確定您的端點對應至您的憑證類型。 請確定您的端點對應至您的 AWS 區域。

欄位	Description	備註
ggHost	您的 AWS IoT Greengrass 端點。	這是您的 iotHost 端點，其具有的主機字首會由 greengrass 取代 (例如 greengrass-ats.iot . <i>region</i>.amazonaws.com)。使用與 AWS 區域 相同的 iotHost。  Note 請確定您的端點對應至您的憑證類型。 請確定您的端點對應至您的 AWS 區域。
iotMqttPort	選用。用於 MQTT 通訊的連接埠號碼 AWS IoT。	有效值為 8883 或 443。預設值為 8883。如需詳細資訊，請參閱 連線至連接埠 443 或透過網路代理 。
iotHttpPort	選用。用於建立到 AWS IoT 之 HTTPS 連線的連接埠號碼。	有效值為 8443 或 443。預設值為 8443。如需詳細資訊，請參閱 連線至連接埠 443 或透過網路代理 。
ggMqttPort	選用。用於透過區域網路進行 MQTT 通訊的連接埠號碼。	有效值為 1024 到 65535。預設值為 8883。如需詳細資訊，請參閱 the section called “用於本機訊息的 MQTT 連接埠” 。
ggHttpPort	選用。用於建立到 AWS IoT Greengrass 服務之 HTTPS 連線的連接埠號碼。	有效值為 8443 或 443。預設值為 8443。如需詳細資訊，請參閱 連線至連接埠 443 或透過網路代理 。

欄位	Description	備註
keepAlive	選用。MQTT KeepAlive 期間 (以秒為單位)。	有效範圍介於 30 與 1200 秒之間。預設值為 600。
networkProxy	選用。定義要連線代理伺服器的物件。	代理伺服器可以是 HTTP 或 HTTPS。如需詳細資訊，請參閱 連線至連接埠 443 或透過網路代理 。
mqttOperationTimeout	選用。在與 AWS IoT Core 的 MQTT 連線中，允許 Greengrass 核心完成發佈、訂閱或取消訂閱操作的時間量 (以秒為單位)。	此屬性從 AWS IoT Greengrass v1.10.2 開始可用。 預設值為 5。最小值為 5。

runtime

欄位	Description	備註
maxWorkItemCount	選用。Greengrass 協助程式一次可以處理的工作項目數量上限。超過此限制的工作項目會被忽略。 工作項目佇列由系統元件、使用者定義的 Lambda 函數和連接器共用。	預設值為 1024。該最大值受裝置硬體的限制。 增加此值會增加 AWS IoT Greengrass 使用的記憶體。如果您預期核心會接收大量的 MQTT 訊息流量，則可以提高此值。
maxConcurrentLimit	選用。Greengrass 協助程式可以擁有的並行未鎖定 Lambda 工作者數量上限。您可以指定不同的整數來覆寫此參數。	預設值為 25。最小值由定義lruSize。

欄位	Description	備註
lruSize	Optional. Defines the minimum value for maxConcurrentLimit .	The default value is 25.
postStartHealthCheckTimeout	Optional. The time (in milliseconds) after starting that the Greengrass daemon waits for the health check to finish.	The default timeout is 30 seconds (30000 ms).
cgroup		
useSystemd	Indicates whether your device uses systemd .	Valid values are # or #. Run the check_ggc_dependencies script in 模組 1 to see if your device uses systemd.

加密

crypto 包含支援透過 PKCS#11 和本機秘密儲存在硬體安全模組 (HSM) 上儲存私密金鑰的屬性。如需詳細資訊，請參閱[the section called “安全性主體”](#)、[the section called “硬體安全整合”](#)及[將私密部署至核心](#)。支援 HSM 上或檔案系統中私有金鑰儲存的組態。

欄位	Description	備註
caPath	AWS IoT 根 CA 的絕對路徑。	必須為此格式的檔案 URI : file:///absolute/path/to/file 。

Note

請確定您的[端點對應至您的憑證類型](#)。

欄位	Description	備註
PKCS11		
OpenSSL Engine	選用。OpenSSL 引擎 .so 檔案的絕對路徑，此檔案可在 OpenSSL 上啟用 PKCS#11 支援。	必須是檔案系統上的檔案路徑。 如果您使用具有硬體安全性的 Greengrass OTA 更新代理程式，則需要此屬性。如需詳細資訊，請參閱 the section called “設定 OTA 更新” 。
P11Provider	PKCS#11 實作的 libdl-loadable 程式庫的絕對路徑。	必須是檔案系統上的檔案路徑。
slotLabel	用於識別硬體模組的插槽標籤。	必須符合 PKCS#11 標籤規格。
slotUserPin	用來向模組驗證 Greengrass 核心的使用者 PIN 碼。	必須具有足夠的許可，才能使用設定的私有金鑰執行 C_Sign。
principals		
IoTCertificate	The certificate and private key that the core uses to make requests to AWS IoT.	
IoTCertificate .privateKeyPath	核心私有金鑰的路徑。	如為檔案系統儲存，必須為此格式的檔案 URI : file:///absolute/path/to/file 。 如為 HSM 儲存，必須為指定物件標籤的 RFC 7512 PKCS # 11 路徑。

欄位	Description	備註
IoTCertificate .certificatePath	核心裝置憑證的絕對路徑。	必須為此格式的檔案 URI : file:///absolute/ path/to/file 。
MQTTServerCertificate	選用。核心用於結合憑證做為 MQTT 伺服器或閘道的私有金鑰。	
MQTTServerCertificate.privateKeyPath	本機 MQTT 伺服器私有金鑰的路徑。	使用此值，為本機 MQTT 伺服器指定您自己的私有金鑰。 如為檔案系統儲存，必須為此格式的檔案 URI : file:///absolute/ path/to/file 。 如為 HSM 儲存，必須為指定物件標籤的 RFC 7512 PKCS # 11 路徑。 如果省略此屬性，會根據輪換設定 AWS IoT Greengrass 輪換金鑰。若有指定，則由客戶負責輪換金鑰。
SecretsManager	The private key that secures the data key used for encryption. For more information, see 將私密部署至核心 .	

欄位	Description	備註
SecretsManager .privateKeyPath	本機 Secrets Manager 私有金鑰的路徑。	僅支援 RSA 金鑰。 如為檔案系統儲存，必須為此格式的檔案 URI : file:///absolute/path/to/file 。 如為 HSM 儲存，必須為指定物件標籤的 RFC 7512 PKCS # 11 路徑。必須使用 PKCS#1 v1.5 填補機制產生私密金鑰。

也支援下列組態屬性：

欄位	Description	備註
mqttMaxConnectionRetryInterval	選用。連線中斷時，MQTT 連線重試的間隔上限 (以秒為單位)。	請指定此數值做為未簽署的整數。預設值為 60。
managedRespawn	選用。指出 OTA 代理程式在更新之前需要執行自訂程式碼。	有效值為 true 或 false。如需詳細資訊，請參閱 AWS IoT Greengrass Core 軟體的 OTA 更新 。
writeDirectory	選用。其中 AWS IoT Greengrass 建立所有讀取/寫入資源的寫入目錄。	如需詳細資訊，請參閱 設定的寫入目錄 AWS IoT Greengrass 。

GGC 1.9 版

```
{
  "coreThing" : {
    "caPath" : "root.ca.pem",
```

```

    "certPath" : "hash.cert.pem",
    "keyPath" : "hash.private.key",
    "thingArn" : "arn:partition:iot:region:account-id:thing/core-thing-name",
    "iotHost" : "host-prefix-ats.iot.region.amazonaws.com",
    "ggHost" : "greengrass-ats.iot.region.amazonaws.com",
    "keepAlive" : 600
  },
  "runtime" : {
    "cgroup" : {
      "useSystemd" : "yes"
    }
  },
  "managedRespawn" : false,
  "crypto" : {
    "principals" : {
      "SecretsManager" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key"
      },
      "IoTCertificate" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key",
        "certificatePath" : "file:///greengrass/certs/hash.cert.pem"
      }
    }
  },
  "caPath" : "file:///greengrass/certs/root.ca.pem"
}

```

config.json 檔案支援以下屬性：

coreThing

欄位	Description	備註
caPath	AWS IoT 根 CA 相對於 <i>greengrass-root</i> / certs 目錄的路徑。	用於回溯相容於 1.7.0 之前的版本。當 crypto 物件存在時，會忽略此屬性。

 **Note**
 請確定您的端點對應至您的憑證類型。

欄位	Description	備註
certPath	相對於 <code>/greengrass-root/certs</code> 目錄的核心裝置憑證路徑。	用於回溯相容於 1.7.0 之前的版本。當crypto物件存在時，會忽略此屬性。
keyPath	相對於 <code>/greengrass-root/certs</code> 目錄的核心私有金鑰路徑。	用於回溯相容於 1.7.0 之前的版本。當crypto物件存在時，會忽略此屬性。
thingArn	代表 AWS IoT Greengrass 核心裝置的 AWS IoT 物件的 Amazon Resource Name (ARN)。	在 AWS IoT Greengrass 主控台的 核心下尋找核心的 ARN，或執行 CLI aws greengrass get-core-definition-version 命令。
iotHost	您的 AWS IoT 端點。	在主控台的設定 AWS IoT 下或透過執行 CLI aws iot describe-endpoint --endpoint-type iot:Data-ATS 命令來尋找端點。 此命令傳回 Amazon Trust Services (ATS) 端點。如需詳細資訊，請參閱伺服器身分驗證文件。  Note 請確定您的端點對應至您的憑證類型。 請確定您的端點與您的對應 AWS 區域。

欄位	Description	備註
ggHost	您的 AWS IoT Greengrass 端點。	這是您的 iotHost 端點，其具有的主機字首會由 greengrass 取代 (例如 greengrass-ats.iot . <i>region</i>.amazonaws.com)。使用與 AWS 區域 相同的 iotHost。  Note 請確定您的端點對應至您的憑證類型。 請確定您的端點對應至您的 AWS 區域。
iotMqttPort	選用。用於 MQTT 通訊的連接埠號碼 AWS IoT。	有效值為 8883 或 443。預設值為 8883。如需詳細資訊，請參閱 連線至連接埠 443 或透過網路代理 。
iotHttpPort	選用。用於建立到 AWS IoT 之 HTTPS 連線的連接埠號碼。	有效值為 8443 或 443。預設值為 8443。如需詳細資訊，請參閱 連線至連接埠 443 或透過網路代理 。
ggHttpPort	選用。用於建立到 AWS IoT Greengrass 服務之 HTTPS 連線的連接埠號碼。	有效值為 8443 或 443。預設值為 8443。如需詳細資訊，請參閱 連線至連接埠 443 或透過網路代理 。
keepAlive	選用。MQTT KeepAlive 期間 (以秒為單位)。	有效範圍介於 30 與 1200 秒之間。預設值為 600。

欄位	Description	備註
networkProxy	選用。定義要連線代理伺服器的物件。	代理伺服器可以是 HTTP 或 HTTPS。如需詳細資訊，請參閱 連線至連接埠 443 或透過網路代理 。

runtime

欄位	Description	備註
maxConcurrentLimit	選用。Greengrass 協助程式可以擁有的並行未鎖定 Lambda 工作者數量上限。您可以指定不同的整數來覆寫此參數。	預設值為 25。最小值由定義lruSize。
lruSize	Optional. Defines the minimum value for maxConcurrentLimit .	The default value is 25.
postStartHealthCheckTimeout	Optional. The time (in milliseconds) after starting that the Greengrass daemon waits for the health check to finish.	The default timeout is 30 seconds (30000 ms).
cgroup		
useSystemd	Indicates whether your device uses systemd .	Valid values are # or #. Run the check_ggc_dependencies script in 模組 1 to see if your device uses systemd.

加密

已在 v1.7.0 中新增 crypto 物件。其中推出的屬性可透過 PKCS#11 和本機私密儲存支援硬體安全模組 (HSM) 上的私有金鑰儲存。如需詳細資訊，請參閱[the section called “安全性主體”](#)、[the section called “硬體安全整合”](#)及[將私密部署至核心](#)。支援 HSM 上或檔案系統中私有金鑰儲存的組態。

欄位	Description	備註
caPath	AWS IoT 根 CA 的絕對路徑。	必須為此格式的檔案 URI : file:///absolute/path/to/file 。
 Note 請確定您的端點對應至您的憑證類型。		
PKCS11		
OpenSSLEngine	選用。OpenSSL 引擎 .so 檔案的絕對路徑，此檔案可在 OpenSSL 上啟用 PKCS#11 支援。	必須是檔案系統上的檔案路徑。 如果您使用具有硬體安全性的 Greengrass OTA 更新代理程式，則需要此屬性。如需詳細資訊，請參閱 the section called “設定 OTA 更新” 。
P11Provider	PKCS#11 實作的 libdl-loadable 程式庫的絕對路徑。	必須是檔案系統上的檔案路徑。
slotLabel	用於識別硬體模組的插槽標籤。	必須符合 PKCS#11 標籤規格。
slotUserPin	用來向模組驗證 Greengrass 核心的使用者 PIN 碼。	必須具有足夠的許可，才能使用設定的私有金鑰執行 C_Sign。

欄位	Description	備註
principals		
IoTCertificate	The certificate and private key that the core uses to make requests to AWS IoT.	
IoTCertificate .privateKeyPath	核心私有金鑰的路徑。	如為檔案系統儲存，必須為此格式的檔案 URI : file:///absolute/path/to/file 。 如為 HSM 儲存，必須為指定物件標籤的 RFC 7512 PKCS # 11 路徑。
IoTCertificate .certificatePath	核心裝置憑證的絕對路徑。	必須為此格式的檔案 URI : file:///absolute/path/to/file 。
MQTTServerCertificate	選用。核心用於結合憑證做為 MQTT 伺服器或閘道的私有金鑰。	

欄位	Description	備註
MQTTServerCertificate.privateKeyPath	本機 MQTT 伺服器私有金鑰的路徑。	使用此值，為本機 MQTT 伺服器指定您自己的私有金鑰。 如為檔案系統儲存，必須為此格式的檔案 URI : file:///absolute/path/to/file 。 如為 HSM 儲存，必須為指定物件標籤的 RFC 7512 PKCS # 11 路徑。 如果省略此屬性，會根據輪換設定 AWS IoT Greengrass 輪換金鑰。若有指定，則由客戶負責輪換金鑰。
SecretsManager	The private key that secures the data key used for encryption. For more information, see 將私密部署至核心 .	
SecretsManager.privateKeyPath	本機 Secrets Manager 私有金鑰的路徑。	僅支援 RSA 金鑰。 如為檔案系統儲存，必須為此格式的檔案 URI : file:///absolute/path/to/file 。 如為 HSM 儲存，必須為指定物件標籤的 RFC 7512 PKCS # 11 路徑。必須使用 PKCS#1 v1.5 填補機制產生私密金鑰。

也支援下列組態屬性。

欄位	Description	備註
mqttMaxConnectionRetryInterval	選用。連線中斷時，MQTT 連線重試的間隔上限 (以秒為單位)。	請指定此數值做為未簽署的整數。預設值為 60。
managedRespawn	選用。指出 OTA 代理程式在更新之前需要執行自訂程式碼。	有效值為 true 或 false。如需詳細資訊，請參閱 AWS IoT Greengrass Core 軟體的 OTA 更新 。
writeDirectory	選用。其中 AWS IoT Greengrass 建立所有讀取/寫入資源的寫入目錄。	如需詳細資訊，請參閱 設定的寫入目錄 AWS IoT Greengrass 。

GGC 1.8 版

```
{
  "coreThing" : {
    "caPath" : "root.ca.pem",
    "certPath" : "hash.cert.pem",
    "keyPath" : "hash.private.key",
    "thingArn" : "arn:aws:iot:region:account-id:thing/core-thing-name",
    "iotHost" : "host-prefix-ats.iot.region.amazonaws.com",
    "ggHost" : "greengrass-ats.iot.region.amazonaws.com",
    "keepAlive" : 600
  },
  "runtime" : {
    "cgroup" : {
      "useSystemd" : "yes"
    }
  },
  "managedRespawn" : false,
  "crypto" : {
    "principals" : {
      "SecretsManager" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key"
      },
      "IoTCertificate" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key",

```

```

    "certificatePath" : "file:///greengrass/certs/hash.cert.pem"
  },
  "caPath" : "file:///greengrass/certs/root.ca.pem"
}

```

config.json 檔案支援下列屬性。

coreThing

欄位	Description	備註
caPath	AWS IoT 根 CA 相對於 <i>/greengrass-root</i> / certs 目錄的路徑。	用於回溯相容於 1.7.0 之前的版本。當 crypto 物件存在時，會忽略此屬性。  Note 請確定您的 端點對應至您的憑證類型。
certPath	相對於 <i>/greengrass-root</i> /certs 目錄的核心裝置憑證路徑。	用於回溯相容於 1.7.0 之前的版本。當 crypto 物件存在時，會忽略此屬性。
keyPath	相對於 <i>/greengrass-root</i> /certs 目錄的核心私有金鑰路徑。	用於回溯相容於 1.7.0 之前的版本。當 crypto 物件存在時，會忽略此屬性。
thingArn	代表 AWS IoT Greengrass 核心裝置的 AWS IoT 物件的 Amazon Resource Name (ARN)。	在 AWS IoT Greengrass 主控台的 核心 下尋找核心的 ARN，或執行 CLI aws greengrass get-core-definition-version 命令。
iotHost	您的 AWS IoT 端點。	在 AWS IoT 主控台的設定下或透過執行 CLI aws iot

欄位	Description	備註
		describe-endpoint --endpoint-type iot:Data-ATS 命令來尋找端點。 此命令傳回 Amazon Trust Services (ATS) 端點。如需詳細資訊，請參閱伺服器身分驗證文件。  Note 請確定您的端點對應至您的憑證類型。確保您的端點對應至您的 AWS 區域。
ggHost	您的 AWS IoT Greengrass 端點。	這是您的 iotHost 端點，其具有的主機字首會由 greengrass 取代 (例如 greengrass-ats.iot . <i>region</i>.amazonaws.com)。使用與 AWS 區域相同的 iotHost。  Note 請確定您的端點對應至您的憑證類型。請確定您的端點對應至您的 AWS 區域。

欄位	Description	備註
iotMqttPort	選用。用於 MQTT 通訊的連接埠號碼 AWS IoT。	有效值為 8883 或 443。 預設值為 8883。如需詳細資訊，請參閱 連線至連接埠 443 或透過網路代理 。
iotHttpPort	選用。用於建立到 AWS IoT 之 HTTPS 連線的連接埠號碼。	有效值為 8443 或 443。 預設值為 8443。如需詳細資訊，請參閱 連線至連接埠 443 或透過網路代理 。
ggHttpPort	選用。用於建立到 AWS IoT Greengrass 服務之 HTTPS 連線的連接埠號碼。	有效值為 8443 或 443。 預設值為 8443。如需詳細資訊，請參閱 連線至連接埠 443 或透過網路代理 。
keepAlive	選用。MQTT KeepAlive 期間 (以秒為單位)。	有效範圍介於 30 與 1200 秒之間。預設值為 600。
networkProxy	選用。定義要連線代理伺服器的物件。	代理伺服器可以是 HTTP 或 HTTPS。如需詳細資訊，請參閱 連線至連接埠 443 或透過網路代理 。

runtime

欄位	Description	備註
cgroup		
useSystemd	Indicates whether your device uses systemd .	Valid values are # or #. Run the check_ggc_dependencies script in 模組 1 to see if your device uses systemd.

加密

已在 v1.7.0 中新增 crypto 物件。其中推出的屬性可透過 PKCS#11 和本機私密儲存支援硬體安全模組 (HSM) 上的私有金鑰儲存。如需詳細資訊，請參閱[the section called “安全性主體”](#)、[the section called “硬體安全整合”](#)及[將私密部署至核心](#)。支援 HSM 上或檔案系統中私有金鑰儲存的組態。

欄位	Description	備註
caPath	AWS IoT 根 CA 的絕對路徑。	必須為此格式的檔案 URI : file:///absolute/path/to/file 。
 Note 請確定您的端點對應至您的憑證類型。		
PKCS11		
OpenSSL Engine	選用。OpenSSL 引擎 .so 檔案的絕對路徑，此檔案可在 OpenSSL 上啟用 PKCS#11 支援。	必須是檔案系統上的檔案路徑。 如果您使用具有硬體安全性的 Greengrass OTA 更新代理程式，則需要此屬性。如需詳細資訊，請參閱the section called “設定 OTA 更新”。
P11Provider	PKCS#11 實作的 libdl-loadable 程式庫的絕對路徑。	必須是檔案系統上的檔案路徑。
slotLabel	用於識別硬體模組的插槽標籤。	必須符合 PKCS#11 標籤規格。

欄位	Description	備註
slotUserPin	用來向模組驗證 Greengrass 核心的使用者 PIN 碼。	必須具有足夠的許可，才能使用設定的私有金鑰執行 C_Sign。
principals		
IoTCertificate	The certificate and private key that the core uses to make requests to AWS IoT.	
IoTCertificate .privateKeyPath	核心私有金鑰的路徑。	如為檔案系統儲存，必須為此格式的檔案 URI : file:///absolute/path/to/file 。 如為 HSM 儲存，必須為指定物件標籤的 RFC 7512 PKCS # 11 路徑。
IoTCertificate .certificatePath	核心裝置憑證的絕對路徑。	必須為此格式的檔案 URI : file:///absolute/path/to/file 。
MQTTServerCertificate	選用。核心用於結合憑證做為 MQTT 伺服器或閘道的私有金鑰。	

欄位	Description	備註
MQTTServerCertificate.privateKeyPath	本機 MQTT 伺服器私有金鑰的路徑。	使用此值，為本機 MQTT 伺服器指定您自己的私有金鑰。 如為檔案系統儲存，必須為此格式的檔案 URI : file:///absolute/path/to/file 。 如為 HSM 儲存，必須為指定物件標籤的 RFC 7512 PKCS # 11 路徑。 如果省略此屬性，會根據輪換設定 AWS IoT Greengrass 輪換金鑰。若有指定，則由客戶負責輪換金鑰。
SecretsManager	The private key that secures the data key used for encryption. For more information, see 將私密部署至核心 .	
SecretsManager.privateKeyPath	本機 Secrets Manager 私有金鑰的路徑。	僅支援 RSA 金鑰。 如為檔案系統儲存，必須為此格式的檔案 URI : file:///absolute/path/to/file 。 如為 HSM 儲存，必須為指定物件標籤的 RFC 7512 PKCS # 11 路徑。必須使用 PKCS#1 v1.5 填補機制產生私密金鑰。

也支援下列組態屬性：

欄位	Description	備註
mqttMaxConnectionRetryInterval	選用。連線中斷時，MQTT 連線重試的間隔上限 (以秒為單位)。	請指定此數值做為未簽署的整數。預設值為 60。
managedRespawn	選用。指出 OTA 代理程式在更新之前需要執行自訂程式碼。	有效值為 true 或 false。如需詳細資訊，請參閱 AWS IoT Greengrass Core 軟體的 OTA 更新 。
writeDirectory	選用。其中 AWS IoT Greengrass 建立所有讀取/寫入資源的寫入目錄。	如需詳細資訊，請參閱 設定的寫入目錄 AWS IoT Greengrass 。

GGC v1.7

```
{
  "coreThing" : {
    "caPath" : "root.ca.pem",
    "certPath" : "hash.cert.pem",
    "keyPath" : "hash.private.key",
    "thingArn" : "arn:aws:iot:region:account-id:thing/core-thing-name",
    "iotHost" : "host-prefix-ats.iot.region.amazonaws.com",
    "ggHost" : "greengrass-ats.iot.region.amazonaws.com",
    "keepAlive" : 600
  },
  "runtime" : {
    "cgroup" : {
      "useSystemd" : "yes"
    }
  },
  "managedRespawn" : false,
  "crypto" : {
    "principals" : {
      "SecretsManager" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key"
      },
      "IoTCertificate" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key",

```

```

    "certificatePath" : "file:///greengrass/certs/hash.cert.pem"
  },
  "caPath" : "file:///greengrass/certs/root.ca.pem"
}

```

config.json 檔案支援以下屬性：

coreThing

欄位	Description	備註
caPath	AWS IoT 根 CA 相對於 <i>/greengrass-root</i> / certs 目錄的路徑。	用於回溯相容於 1.7.0 之前的版本。當 crypto 物件存在時，會忽略此屬性。  Note 請確定您的 端點對應至您的憑證類型。
certPath	相對於 <i>/greengrass-root</i> /certs 目錄的核心裝置憑證路徑。	用於回溯相容於 1.7.0 之前的版本。當 crypto 物件存在時，會忽略此屬性。
keyPath	相對於 <i>/greengrass-root</i> /certs 目錄的核心私有金鑰路徑。	用於回溯相容於 1.7.0 之前的版本。當 crypto 物件存在時，會忽略此屬性。
thingArn	代表 AWS IoT Greengrass 核心裝置的 AWS IoT 物件的 Amazon Resource Name (ARN)。	在 AWS IoT Greengrass 主控台的 核心下尋找核心的 ARN，或執行 CLI aws greengrass get-core-definition-version 命令。
iotHost	您的 AWS IoT 端點。	在 AWS IoT 主控台的設定下或透過執行 CLI aws iot

欄位	Description	備註
		describe-endpoint --endpoint-type iot:Data-ATS 命令來尋找端點。 此命令傳回 Amazon Trust Services (ATS) 端點。如需詳細資訊，請參閱伺服器身分驗證文件。  Note 請確定您的端點對應至您的憑證類型。確保您的端點對應至您的 AWS 區域。
ggHost	您的 AWS IoT Greengrass 端點。	這是您的 iotHost 端點，其具有的主機字首會由 greengrass 取代 (例如 greengrass-ats.iot . <i>region</i>.amazonaws.com)。使用與 AWS 區域相同的 iotHost。  Note 請確定您的端點對應至您的憑證類型。確保您的端點對應至您的 AWS 區域。

欄位	Description	備註
iotMqttPort	選用。用於 MQTT 通訊的連接埠號碼 AWS IoT。	有效值為 8883 或 443。 預設值為 8883。如需詳細資訊，請參閱 連線至連接埠 443 或透過網路代理 。
keepAlive	選用。MQTT KeepAlive 期間 (以秒為單位)。	有效範圍介於 30 與 1200 秒之間。預設值為 600。
networkProxy	選用。定義要連線代理伺服器的物件。	代理伺服器可以是 HTTP 或 HTTPS。如需詳細資訊，請參閱 連線至連接埠 443 或透過網路代理 。

runtime

欄位	Description	備註
cgroup		
useSystemd	Indicates whether your device uses systemd .	Valid values are # or #. Run the check_ggc_dependencies script in 模組 1 to see if your device uses systemd.

加密

於 v1.7.0 新增的 crypto 物件所引進的屬性，可透過 PKCS#11 和本機私密儲存支援硬體安全模組 (HSM) 上的私有金鑰儲存。如需詳細資訊，請參閱[the section called “硬體安全整合”](#)及[將私密部署至核心](#)。支援 HSM 上或檔案系統中私有金鑰儲存的組態。

欄位	Description	備註
caPath	AWS IoT 根 CA 的絕對路徑。	必須為此格式的檔案 URI : file:///absolute/path/to/file 。
 Note 請確定您的端點對應至您的憑證類型。		
PKCS11		
OpenSSLEngine	選用。OpenSSL 引擎 .so 檔案的絕對路徑，此檔案可在 OpenSSL 上啟用 PKCS#11 支援。	必須是檔案系統上的檔案路徑。 如果您使用具有硬體安全性的 Greengrass OTA 更新代理程式，則需要此屬性。如需詳細資訊，請參閱 the section called “設定 OTA 更新” 。
P11Provider	PKCS#11 實作的 libdl-loadable 程式庫的絕對路徑。	必須是檔案系統上的檔案路徑。
slotLabel	用於識別硬體模組的插槽標籤。	必須符合 PKCS#11 標籤規格。
slotUserPin	用來向模組驗證 Greengrass 核心的使用者 PIN 碼。	必須具有足夠的許可，才能使用設定的私有金鑰執行 C_Sign。
principals		
IoTCertificate	The certificate and private key that the core uses to make requests to AWS IoT.	

欄位	Description	備註
IoTCertificate .privateKeyPath	核心私有金鑰的路徑。	如為檔案系統儲存，必須為此格式的檔案 URI : file:///absolute/path/to/file 。 如為 HSM 儲存，必須為指定物件標籤的 RFC 7512 PKCS # 11 路徑。
IoTCertificate .certificatePath	核心裝置憑證的絕對路徑。	必須為此格式的檔案 URI : file:///absolute/path/to/file 。
MQTTServerCertificate	選用。核心用於結合憑證做為 MQTT 伺服器或閘道的私有金鑰。	
MQTTServerCertificate .privateKeyPath	本機 MQTT 伺服器私有金鑰的路徑。	使用此值，為本機 MQTT 伺服器指定您自己的私有金鑰。 如為檔案系統儲存，必須為此格式的檔案 URI : file:///absolute/path/to/file 。 如為 HSM 儲存，必須為指定物件標籤的 RFC 7512 PKCS # 11 路徑。 如果省略此屬性，會根據輪換設定 AWS IoT Greengrass 輪換金鑰。若有指定，則由客戶負責輪換金鑰。
SecretsManager	The private key that secures the data key used for encryption. For more information, see 將私密部署至核心 .	

欄位	Description	備註
SecretsManager .privateKeyPath	本機 Secrets Manager 私有金鑰的路徑。	僅支援 RSA 金鑰。 如為檔案系統儲存，必須為此格式的檔案 URI：<code>file:///absolute/path/to/file</code>。 如為 HSM 儲存，必須為指定物件標籤的 RFC 7512 PKCS # 11 路徑。必須使用 PKCS#1 v1.5 填補機制產生私密金鑰。

也支援下列組態屬性：

欄位	Description	備註
mqttMaxConnectionRetryInterval	選用。連線中斷時，MQTT 連線重試的間隔上限 (以秒為單位)。	請指定此數值做為未簽署的整數。預設值為 60。
managedRespawn	選用。指出 OTA 代理程式在更新之前需要執行自訂程式碼。	有效值為 true 或 false。如需詳細資訊，請參閱 AWS IoT Greengrass Core 軟體的 OTA 更新 。
writeDirectory	選用。其中 AWS IoT Greengrass 建立所有讀取/寫入資源的寫入目錄。	如需詳細資訊，請參閱 設定的寫入目錄 AWS IoT Greengrass 。

GGC v1.6

```
{
  "coreThing": {
    "caPath": "root-ca-pem",
```

```

    "certPath": "cloud-pem-crt",
    "keyPath": "cloud-pem-key",
    "thingArn": "arn:aws:iot:region:account-id:thing/core-thing-name",
    "iotHost": "host-prefix.iot.region.amazonaws.com",
    "ggHost": "greengrass.iot.region.amazonaws.com",
    "keepAlive": 600,
    "mqttMaxConnectionRetryInterval": 60
  },
  "runtime": {
    "cgroup": {
      "useSystemd": "yes/no"
    }
  },
  "managedRespawn": true,
  "writeDirectory": "/write-directory"
}

```

Note

如果您從 AWS IoT Greengrass 主控台使用預設群組建立選項，則 config.json 檔案會部署到工作狀態的核心裝置，以指定預設組態。

config.json 檔案支援以下屬性：

欄位	Description	備註
caPath	AWS IoT 根 CA 相對於 <i>/greengrass-root / certs</i> 目錄的路徑。	將檔案儲存於 <i>/greengrass-root / certs</i> 下。
certPath	AWS IoT Greengrass 核心憑證相對於 <i>/greengrass-root / certs</i> 目錄的路徑。	將檔案儲存於 <i>/greengrass-root / certs</i> 下。
keyPath	AWS IoT Greengrass 核心私有金鑰相對於 <i>/greengrass-root / certs</i> 目錄的路徑。	將檔案儲存於 <i>/greengrass-root / certs</i> 下。

欄位	Description	備註
thingArn	代表 AWS IoT Greengrass 核心裝置的 AWS IoT 物件的 Amazon Resource Name (ARN)。	在 AWS IoT Greengrass 主控台的 核心下尋找核心的 ARN，或執行 CLI aws greengrass get-core-definition-version 命令。
iotHost	您的 AWS IoT 端點。	在 AWS IoT 主控台的設定下或透過執行 CLI aws iot describe-endpoint 命令來尋找此項目。
ggHost	您的 AWS IoT Greengrass 端點。	此值使用的格式為 greengrass.iot. <i>region</i> .amazonaws.com。使用與 iotHost 相同的區域。
keepAlive	MQTT KeepAlive 期間 (以秒為單位)。	此為選用值。預設值為 600。
mqttMaxConnectionRetryInterval	連線中斷時，MQTT 連線重試的間隔上限 (以秒為單位)。	請指定此數值做為未簽署的整數。此為選用值。預設值為 60。
useSystemd	指出您的裝置是否使用 systemd 。	有效值為 yes 或 no。執行 模組 1 中的 check_ggc_dependencies 指令碼，以查看您的裝置是否使用 systemd。
managedRespawn	選用的無線 (OTA) 更新功能，這表示 OTA 代理程式在更新之前需執行自訂程式碼。	有效值為 true 或 false。如需詳細資訊，請參閱 AWS IoT Greengrass Core 軟體的 OTA 更新 。

欄位	Description	備註
writeDirectory	其中 AWS IoT Greengrass 建立所有讀取/寫入資源的寫入目錄。	此為選用值。如需詳細資訊，請參閱 設定的寫入目錄 AWS IoT Greengrass 。

GGC v1.5

```
{
  "coreThing": {
    "caPath": "root-ca-pem",
    "certPath": "cloud-pem-crt",
    "keyPath": "cloud-pem-key",
    "thingArn": "arn:aws:iot:region:account-id:thing/core-thing-name",
    "iotHost": "host-prefix.iot.region.amazonaws.com",
    "ggHost": "greengrass.iot.region.amazonaws.com",
    "keepAlive": 600
  },
  "runtime": {
    "cgroup": {
      "useSystemd": "yes/no"
    }
  },
  "managedRespawn": true
}
```

config.json 檔案存在於 */greengrass-root*/config 中並包含以下參數：

欄位	Description	備註
caPath	AWS IoT 根 CA 相對於 <i>/greengrass-root</i> /certs 資料夾的路徑。	將檔案儲存在 <i>/greengrass-root</i> /certs 資料夾底下。
certPath	AWS IoT Greengrass 核心憑證相對於 <i>/greengrass-root</i> /certs 資料夾的路徑。	將檔案儲存在 <i>/greengrass-root</i> /certs 資料夾底下。

欄位	Description	備註
keyPath	AWS IoT Greengrass 核心私有金鑰相對於 <code>/greengrass-root /certs</code> 資料夾的路徑。	將檔案儲存在 <code>/greengrass-root /certs</code> 資料夾底下。
thingArn	代表 AWS IoT Greengrass 核心裝置的 AWS IoT 物件的 Amazon Resource Name (ARN)。	在 AWS IoT Greengrass 主控台的 核心下尋找核心的 ARN，或執行 CLI aws greengrass get-core-definition-version 命令。
iotHost	您的 AWS IoT 端點。	在 AWS IoT 主控台的設定下或透過執行 aws iot describe-endpoint 命令來尋找此項目。
ggHost	您的 AWS IoT Greengrass 端點。	此值使用的格式為 <code>greengrass.iot.region.amazonaws.com</code> 。使用與 <code>iotHost</code> 相同的區域。
keepAlive	MQTT KeepAlive 期間 (以秒為單位)。	此為選用值。預設值為 600 秒。
useSystemd	指出您的裝置是否使用 systemd 。	有效值為 <code>yes</code> 或 <code>no</code> 。執行 模組 1 中的 <code>check_ggc_dependencies</code> 指令碼，以查看您的裝置是否使用 <code>systemd</code> 。
managedRespawn	選用的無線 (OTA) 更新功能，這表示 OTA 代理程式在更新之前需執行自訂程式碼。	如需詳細資訊，請參閱 AWS IoT Greengrass Core 軟體的 OTA 更新 。

GGC v1.3

```
{
  "coreThing": {
    "caPath": "root-ca-pem",
    "certPath": "cloud-pem-crt",
    "keyPath": "cloud-pem-key",
    "thingArn": "arn:aws:iot:region:account-id:thing/core-thing-name",
    "iotHost": "host-prefix.iot.region.amazonaws.com",
    "ggHost": "greengrass.iot.region.amazonaws.com",
    "keepAlive": 600
  },
  "runtime": {
    "cgroup": {
      "useSystemd": "yes/no"
    }
  },
  "managedRespawn": true
}
```

config.json 檔案存在於 */greengrass-root*/config 中並包含以下參數：

欄位	Description	備註
caPath	AWS IoT 根 CA 相對於 <i>/greengrass-root</i> /certs 資料夾的路徑。	將檔案儲存在 <i>/greengrass-root</i> /certs 資料夾底下。
certPath	AWS IoT Greengrass 核心憑證相對於 <i>/greengrass-root</i> /certs 資料夾的路徑。	將檔案儲存在 <i>/greengrass-root</i> /certs 資料夾底下。
keyPath	AWS IoT Greengrass 核心私有金鑰相對於 <i>/greengrass-root</i> /certs 資料夾的路徑。	將檔案儲存在 <i>/greengrass-root</i> /certs 資料夾底下。

欄位	Description	備註
thingArn	代表 AWS IoT Greengrass 核心之 AWS IoT 物件的 Amazon Resource Name (ARN)。	您可以在 AWS IoT Greengrass 主控台的 AWS IoT 物件定義下找到此值。
iotHost	您的 AWS IoT 端點。	您可以在 AWS IoT 主控台的設定下找到此值。
ggHost	您的 AWS IoT Greengrass 端點。	您可以在設定下的 AWS IoT 主控台中找到此值，greengrass. 前面加上。
keepAlive	MQTT KeepAlive 期間 (以秒為單位)。	此為選用值。預設值為 600 秒。
useSystemd	二進位標記，如果您的裝置使用 systemd 。	值為 yes 或 no。請使用 單元 1 中的相依性指令碼查看您的裝置是否使用 systemd。
managedRespawn	選用的無線 (OTA) 更新功能，這表示 OTA 代理程式在更新之前需執行自訂程式碼。	如需詳細資訊，請參閱 AWS IoT Greengrass Core 軟體的 OTA 更新 。

GGC v1.1

```
{
  "coreThing": {
    "caPath": "root-ca-pem",
    "certPath": "cloud-pem-crt",
    "keyPath": "cloud-pem-key",
    "thingArn": "arn:aws:iot:region:account-id:thing/core-thing-name",
    "iotHost": "host-prefix.iot.region.amazonaws.com",
    "ggHost": "greengrass.iot.region.amazonaws.com",
    "keepAlive": 600
  },
}
```

```

    "runtime": {
      "cgroup": {
        "useSystemd": "yes/no"
      }
    }
  }
}

```

config.json 檔案存在於 */greengrass-root/config* 中並包含以下參數：

欄位	Description	備註
caPath	AWS IoT 根 CA 相對於 <i>/greengrass-root/certs</i> 資料夾的路徑。	將檔案儲存在 <i>/greengrass-root/certs</i> 資料夾底下。
certPath	AWS IoT Greengrass 核心憑證相對於 <i>/greengrass-root/certs</i> 資料夾的路徑。	將檔案儲存在 <i>/greengrass-root/certs</i> 資料夾底下。
keyPath	AWS IoT Greengrass 核心私有金鑰相對於 <i>/greengrass-root/certs</i> 資料夾的路徑。	將檔案儲存在 <i>/greengrass-root/certs</i> 資料夾底下。
thingArn	代表 AWS IoT Greengrass 核心之 AWS IoT 物件的 Amazon Resource Name (ARN)。	您可以在 AWS IoT Greengrass 主控台的 AWS IoT 物件定義下找到此值。
iotHost	您的 AWS IoT 端點。	您可以在 AWS IoT 主控台的設定下找到此值。
ggHost	您的 AWS IoT Greengrass 端點。	您可以在設定下的 AWS IoT 主控台中找到此值，greengrass. 前面加上。
keepAlive	MQTT KeepAlive 期間 (以秒為單位)。	此為選用值。預設值為 600 秒。

欄位	Description	備註
useSystemd	二進位標記，如果您的裝置使用 systemd 。	值為 yes 或 no。請使用 單元 1 中的相依性指令碼查看您的裝置是否使用 systemd。

GGC v1.0

在 AWS IoT Greengrass Core v1.0 中，config.json 會部署到 *greengrass-root/configuration*。

```
{
  "coreThing": {
    "caPath": "root-ca-pem",
    "certPath": "cloud-pem-crt",
    "keyPath": "cloud-pem-key",
    "thingArn": "arn:aws:iot:region:account-id:thing/core-thing-name",
    "iotHost": "host-prefix.iot.region.amazonaws.com",
    "ggHost": "greengrass.iot.region.amazonaws.com",
    "keepAlive": 600
  },
  "runtime": {
    "cgroup": {
      "useSystemd": "yes/no"
    }
  }
}
```

config.json 檔案存在於 */greengrass-root/configuration* 中並包含以下參數：

欄位	Description	備註
caPath	AWS IoT 根 CA 相對於 <i>/greengrass-root / configuration/certs</i> 資料夾的路徑。	將檔案儲存在 <i>/greengrass-root / configuration/certs</i> 資料夾底下。

欄位	Description	備註
certPath	AWS IoT Greengrass 核心憑證相對於 <code>/greengrass-root/configuration/certs</code> 資料夾的路徑。	將檔案儲存在 <code>/greengrass-root/configuration/certs</code> 資料夾底下。
keyPath	AWS IoT Greengrass 核心私有金鑰相對於 <code>/greengrass-root/configuration/certs</code> 資料夾的路徑。	將檔案儲存在 <code>/greengrass-root/configuration/certs</code> 資料夾底下。
thingArn	代表 AWS IoT Greengrass 核心之 AWS IoT 物件的 Amazon Resource Name (ARN)。	您可以在 AWS IoT Greengrass 主控台的 thing AWS IoT 定義下找到此值。
iotHost	您的 AWS IoT 端點。	您可以在 AWS IoT 主控台的設定下找到此值。
ggHost	您的 AWS IoT Greengrass 端點。	您可以在設定下的 AWS IoT 主控台中找到此值， <code>greengrass.</code> 前面加上。
keepAlive	MQTT KeepAlive 期間 (以秒為單位)。	此為選用值。預設值為 600 秒。
useSystemd	二進位標記，如果您的裝置使用 systemd 。	值為 yes 或 no。請使用 單元 1 中的相依性指令碼查看您的裝置是否使用 systemd。

服務端點必須符合根 CA 憑證類型

您的 AWS IoT Core 和 AWS IoT Greengrass 端點必須對應至裝置上的根 CA 憑證的憑證類型。如果端點和憑證類型不相符，身分驗證嘗試會在裝置和 AWS IoT Core 或 之間失敗 AWS IoT Greengrass。如需詳細資訊，請參閱《AWS IoT 開發人員指南》中的[伺服器身分驗證](#)。

如果您的裝置使用 Amazon Trust Services (ATS) 根 CA 憑證，這是偏好的方法，它也必須使用 ATS 端點進行裝置管理和探索資料平面操作。ATS 端點包含 `ats` 區段，如下列 AWS IoT Core 端點語法所示。

```
prefix-ats.iot.region.amazonaws.com
```

Note

為了回溯相容性，AWS IoT Greengrass 目前支援某些 中的舊版 VeriSign 根 CA AWS 區域憑證和端點。如果您使用的是舊式 VeriSign 根 CA 憑證，我們建議您建立 ATS 端點，並改用 ATS 根 CA 憑證。否則，請務必使用對應的舊式端點。如需詳細資訊，請參閱 [支援的舊版端點](#) Amazon Web Services 一般參考。

config.json 中的端點

在 Greengrass 核心裝置上，端點會在 [config.json](#) 檔案的 `coreThing` 物件中指定。`iotHost` 屬性代表 AWS IoT Core 端點。`ggHost` 屬性代表 AWS IoT Greengrass 端點。在下列範例程式碼片段中，這些屬性指定 ATS 端點。

```
{
  "coreThing" : {
    ...
    "iotHost" : "abcde1234uvwxyz-ats.iot.us-west-2.amazonaws.com",
    "ggHost" : "greengrass-ats.iot.us-west-2.amazonaws.com",
    ...
  },
}
```

AWS IoT Core 端點

您可以使用適當的 `--endpoint-type` 參數執行 [aws iot describe-endpoint](#) CLI 命令來取得 AWS IoT Core 端點。

- 若要傳回 ATS 簽署的端點，請執行：

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

- 若要傳回舊式 VeriSign 簽署的端點，請執行：

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

AWS IoT Greengrass 端點

您的 AWS IoT Greengrass 端點是您的 `iotHost` 端點，主機字首會取代為 `greengrass`。例如，ATS 簽署的端點是 `greengrass-ats.iot.region.amazonaws.com`。這使用與 AWS IoT Core 端點相同的區域。

連線至連接埠 443 或透過網路代理

此功能適用於 AWS IoT Greengrass Core v1.7 和更新版本。

Greengrass 核心 AWS IoT Core 使用 MQTT 訊息通訊協定搭配 TLS 用戶端身分驗證與通訊。根據慣例，透過 TLS 的 MQTT 使用連接埠 8883。不過，做為安全措施之用，受限環境可能會將傳入和傳出流量限制於小範圍的 TCP 連接埠。例如，企業防火牆可能會針對 HTTPS 流量開放連接埠 443，但關閉較不常用之通訊協定的其他連接埠，例如 MQTT 流量的連接埠 8883。其他受限環境可能會要求所有流量先透過 HTTP 連線，再連線至網際網路。

若要在這些案例中啟用通訊，AWS IoT Greengrass 允許下列組態：

- 透過連接埠 443 使用 TLS 用戶端身分驗證的 MQTT。如果您的網路允許對連接埠 443 的連線，您可以將核心設定為針對 MQTT 流量使用連接埠 443，而不使用預設的連接埠 8883。這可以直接連線至連接埠 443，或透過網路代理伺服器連線。

AWS IoT Greengrass 使用 [Application Layer Protocol Network \(ALPN\)](#) TLS 延伸來啟用此連線。針對預設組態，連接埠 443 上透過 TLS 的 MQTT 使用憑證型的用戶端身分驗證。

當設定為使用連接埠 443 的直接連線時，核心支援軟體 [over-the-air\(OTA\) 更新](#)。AWS IoT Greengrass 此支援需要 AWS IoT Greengrass Core v1.9.3 或更新版本。

- 根據預設，透過連接埠 443 的 HTTPS 通訊會透過連接埠 8443 AWS IoT Greengrass 傳送 HTTPS 流量，但您可以將其設定為使用連接埠 443。
- 透過網路代理的連線。您可以設定網路代理伺服器，做為連線至 Greengrass 核心的中介。僅支援基本身分驗證和 HTTP 與 HTTPS 代理。

代理組態會透過 `http_proxy`、`https_proxy` 和 `no_proxy` 環境變數傳遞給使用者定義的 Lambda 函數。使用者定義的 Lambda 函數必須使用這些傳入設定來透過代理連線。根據預設，Lambda 函數用來進行連線的常見程式庫（例如 `boto3` 或 `cURL` 和 `python requests` 套件）通常會使用這些環境變數。如果 Lambda 函數也指定這些相同的環境變數，AWS IoT Greengrass 則不會覆寫它們。

Important

設定為使用網路代理的 Greengrass 核心不支援 [OTA 更新](#)。

透過連接埠 443 設定 MQTT

此功能需要 AWS IoT Greengrass Core v1.7 或更新版本。

此程序可讓 Greengrass 核心將連接埠 443 用於與 AWS IoT Core 傳遞 MQTT 訊息。

1. 執行下列命令來停止 Greengrass 協助程式：

```
cd /greengrass-root/ggc/core/  
sudo ./greengrassd stop
```

2. 開放 `greengrass-root/config/config.json` 由 `su` 使用者編輯。
3. 在 `coreThing` 物件中，新增 `iotMqttPort` 屬性，並將值設定為 **443**，如以下範例所示。

```
{  
  "coreThing" : {  
    "caPath" : "root.ca.pem",  
    "certPath" : "12345abcde.cert.pem",  
    "keyPath" : "12345abcde.private.key",  
    "thingArn" : "arn:aws:iot:us-west-2:123456789012:thing/core-thing-name",  
    "iotHost" : "abcd123456wxyz-ats.iot.us-west-2.amazonaws.com",  
    "iotMqttPort" : 443,  
    "ggHost" : "greengrass-ats.iot.us-west-2.amazonaws.com",  
    "keepAlive" : 600  
  },  
  ...  
}
```

4. 啟動協助程式。

```
cd /greengrass-root/ggc/core/  
sudo ./greengrassd start
```

透過連接埠 443 設定 HTTPS

此功能需要 AWS IoT Greengrass Core v1.8 或更新版本。

此程序會設定核心使用連接埠 443 進行 HTTPS 通訊。

1. 執行下列命令來停止 Greengrass 協助程式：

```
cd /greengrass-root/ggc/core/  
sudo ./greengrassd stop
```

2. 開放 *greengrass-root*/config/config.json 由 su 使用者編輯。
3. 在 coreThing 物件中，新增 iotHttpPort 和 ggHttpPort 屬性，如以下範例所示。

```
{  
  "coreThing" : {  
    "caPath" : "root.ca.pem",  
    "certPath" : "12345abcde.cert.pem",  
    "keyPath" : "12345abcde.private.key",  
    "thingArn" : "arn:aws:iot:us-west-2:123456789012:thing/core-thing-name",  
    "iotHost" : "abcd123456wxyz-ats.iot.us-west-2.amazonaws.com",  
    "iotHttpPort" : 443,  
    "ggHost" : "greengrass-ats.iot.us-west-2.amazonaws.com",  
    "ggHttpPort" : 443,  
    "keepAlive" : 600  
  },  
  ...  
}
```

4. 啟動協助程式。

```
cd /greengrass-root/ggc/core/  
sudo ./greengrassd start
```

設定網路代理

此功能需要 AWS IoT Greengrass Core v1.7 或更新版本。

此程序允許 透過 HTTP 或 HTTPS 網路代理 AWS IoT Greengrass 連線到網際網路。

1. 執行下列命令來停止 Greengrass 協助程式：

```
cd /greengrass-root/ggc/core/  
sudo ./greengrassd stop
```

2. 開放 `greengrass-root/config/config.json` 由 su 使用者編輯。
3. 使用 `coreThing` 物件來新增 [networkProxy](#) 物件，如以下範例所示。

```
{  
  "coreThing" : {  
    "caPath" : "root.ca.pem",  
    "certPath" : "12345abcde.cert.pem",  
    "keyPath" : "12345abcde.private.key",  
    "thingArn" : "arn:aws:iot:us-west-2:123456789012:thing/core-thing-name",  
    "iotHost" : "abcd123456wxyz-ats.iot.us-west-2.amazonaws.com",  
    "ggHost" : "greengrass-ats.iot.us-west-2.amazonaws.com",  
    "keepAlive" : 600,  
    "networkProxy": {  
      "noProxyAddresses" : "http://128.12.34.56,www.mywebsite.com",  
      "proxy" : {  
        "url" : "https://my-proxy-server:1100",  
        "username" : "Mary_Major",  
        "password" : "pass@word1357"  
      }  
    }  
  },  
  ...  
}
```

4. 啟動協助程式。

```
cd /greengrass-root/ggc/core/  
sudo ./greengrassd start
```

networkProxy 物件

使用 networkProxy 物件，指定網路代理的相關資訊。此物件具有下列屬性。

欄位	Description (描述)
noProxyAddresses	選用。從代理中免除的 IP 地址或主機名稱清單 (以逗號分隔)。
proxy	要連線至的代理。代理具有以下屬性。 • url。代理伺服器的 URL，格式為 <code>scheme://userinfo@host:port</code>。 • scheme。結構描述。必須是 http 或 https。 • userinfo。選用。使用者名稱和密碼資訊。如指定，則會略過 username 和 password 欄位。 • host。代理伺服器的主機名稱或 IP 地址。 • port。選用。連接埠號碼。若未指定，則會使用以下預設值： • http : 80 • https : 443 • username。選用。要用來向代理伺服器進行身分驗證的使用者名稱。 • password。選用。要用來向代理伺服器進行身分驗證的密碼。

允許端點

Greengrass 裝置和 AWS IoT Core 或之間的通訊 AWS IoT Greengrass 必須經過身分驗證。此身分驗證是根據已註冊的 X.509 裝置憑證和加密金鑰。若要允許經驗證的請求通過代理而不需其他加密，請允許以下端點。

Endpoint	站點	Description
greengrass. <i>region</i> .amazonaws.com	443	用於群組管理的控制平面操作。
<i>prefix</i> -ats.iot. <i>region</i> .amazonaws.com 或 <i>prefix</i> .iot. <i>region</i> .amazonaws.com	MQTT : 8883 或 443 HTTPS : 8443 或 443	用於裝置管理的資料平面操作，例如陰影同步。 根據您的核心和用戶端裝置是否使用 Amazon Trust Services (偏好) 根 CA 憑證、舊版根 CA 憑證或兩者，允許使用一個或兩個端點。如需詳細資訊，請參閱 the section called “服務端點必須符合憑證類型” 。
greengrass-ats.iot . <i>region</i> .amazonaws.com 或 greengrass.iot. <i>region</i> .amazonaws.com	8443 或 443	用於裝置探索操作。 根據您的核心和用戶端裝置是否使用 Amazon Trust

Endpoint	站點	Description
		Services（偏好）根 CA 憑證、舊版根 CA 憑證或兩者，允許使用一個或兩個端點。如需詳細資訊，請參閱the section called “服務端點必須符合憑證類型”。 Note 在連接埠 443 上連線的用戶端必須實作 Application Layer Protocol Negotiation (ALPN) TLS 延伸模組，

Endpoint	站點	Description
		並在 ProtocolName 中 x-amzn-http-ca 做為傳遞 ProtocolNameList 。如需詳細資訊，請參閱《AWS IoT 開發人員指南》中的 通訊協定 。
*.s3.amazonaws.com	443	用於部署操作和無線更新。此格式包括 * 字元，因為端點字首是由內部控制，並可能隨時變更。

Endpoint	站點	Description
logs. <i>region</i> .amazonaws.com	443	如果 Greengrass 群組設定為將日誌寫入 CloudWatch，則為必要項目。

設定 的寫入目錄 AWS IoT Greengrass

此功能適用於 AWS IoT Greengrass Core v1.6 和更新版本。

根據預設，AWS IoT Greengrass Core 軟體會部署在單一根目錄下，其中會 AWS IoT Greengrass 執行所有讀取和寫入操作。不過，您可以將設定為對所有寫入操作 AWS IoT Greengrass 使用單獨的目錄，包括建立目錄和檔案。在此情況下，AWS IoT Greengrass 會使用兩個最上層目錄：

- *greengrass-root* 目錄，可將其保留為可讀寫或選擇為唯讀。這包含 AWS IoT Greengrass 核心軟體和其他重要元件，這些元件在執行期間應保持不變，例如憑證和 config.json。
- 指定的寫入目錄。這包含可寫入的內容，例如日誌、狀態資訊和部署的使用者定義 Lambda 函數。

此組態結果為下列目錄結構。

Greengrass 根目錄

```
greengrass-root/
|-- certs/
|   |-- root.ca.pem
|   |-- hash.cert.pem
|   |-- hash.private.key
|   |-- hash.public.key
|-- config/
|   |-- config.json
|-- ggc/
|   |-- packages/
|       |-- package-version/
|           |-- bin/
|           |-- daemon
```

```
|
|         |-- greengrassd
|         |-- lambda/
|         |-- LICENSE/
|         |-- release_notes_package-version.html
|         |-- runtime/
|             |-- java8/
|             |-- nodejs8.10/
|             |-- python3.8/
| |-- core/
```

寫入目錄

```
write-directory/
|-- packages/
|   |-- package-version/
|       |-- ggc_root/
|       |-- rootfs_nosys/
|       |-- rootfs_sys/
|       |-- var/
|-- deployment/
|   |-- group/
|       |-- group.json
|   |-- lambda/
|   |-- mlmodel/
|-- var/
|   |-- log/
|   |-- state/
```

設定寫入目錄

1. 執行下列命令來停止 AWS IoT Greengrass 協助程式：

```
cd /greengrass-root/ggc/core/
sudo ./greengrassd stop
```

2. 開放 *greengrass-root*/config/config.json 由 su 使用者編輯。
3. 新增 writeDirectory 為參數並指定目標目錄路徑，如下所示。

```
{
```

```
"coreThing": {
  "caPath": "root-CA.pem",
  "certPath": "hash.pem.crt",
  ...
},
...
"writeDirectory" : "/write-directory"
}
```

Note

您隨時可以更新 writeDirectory 設定。更新設定後，會在下次開始時 AWS IoT Greengrass 使用新指定的寫入目錄，但不會從先前的寫入目錄遷移內容。

- 現在，您的寫入目錄已設定，您可以選擇設定 *greengrass-root* 目錄為唯讀。如需詳細資訊，請參閱[設定 Greengrass 根目錄成為唯讀](#)。

否則，請啟動 AWS IoT Greengrass 協助程式：

```
cd /greengrass-root/ggc/core/
sudo ./greengrassd start
```

將 Greengrass 根目錄設為唯讀

只有在您想要將 Greengrass 根目錄設為唯讀狀態時，才需依照下列步驟執行。開始之前，必須先設定寫入目錄。

- 授與必要目錄的存取許可：

- 將讀取和寫入許可提供給 config.json 擁有者。

```
sudo chmod 0600 /greengrass-root/config/config.json
```

- 使 ggc_user 成為憑證和系統 Lambda 目錄的擁有者。

```
sudo chown -R ggc_user:ggc_group /greengrass-root/certs/
sudo chown -R ggc_user:ggc_group /greengrass-root/ggc/packages/1.11.6/lambda/
```

Note

預設會使用 `ggc_user` 和 `ggc_group` 帳戶來執行系統 Lambda 函數。如果您設定群組層級的[預設存取身分](#)使用不同的帳戶，您應該改將許可提供給該使用者 (UID) 和群組 (GID)。

2. 使用您偏好的機制將 *greengrass-root* 目錄設為唯讀。

Note

其中一種設定 *greengrass-root* 目錄為唯讀的方法是掛載目錄為唯讀。不過，若要将無線 (OTA) 更新套用至掛載目錄中的 AWS IoT Greengrass Core 軟體，必須先移除掛載目錄，待更新完成後再次掛載目錄。您可以新增這些 `umount` 和 `mount` 操作到 `ota_pre_update` 和 `ota_post_update` 指令碼中。如需 OTA 更新的詳細資訊，請參閱 [the section called “Greengrass OTA 更新代理程式”](#) 和 [the section called “透過 OTA 更新受管的 respawn”](#)。

3. 啟動協助程式。

```
cd /greengrass-root/ggc/core/  
sudo ./greengrassd start
```

如果未正確設定步驟 1 的許可，則協助程式不會啟動。

配置 MQTT 設定

在 AWS IoT Greengrass 環境中，本機用戶端裝置、Lambda 函數、連接器和系統元件可以互相通訊並與之通訊 AWS IoT Core。所有通訊都經過核心，負責管理授權實體與 MQTT 通訊之間的[訂閱](#)。

如需可設定 MQTT 設定的相關資訊 AWS IoT Greengrass，請參閱下列各節：

- [the section called “訊息服務品質”](#)
- [the section called “MQTT 訊息佇列”](#)
- [the section called “使用的 MQTT 持久性工作階段 AWS IoT Core”](#)
- [the section called “與進行 MQTT 連線的用戶端 IDs AWS IoT”](#)
- [用於本機訊息的 MQTT 連接埠](#)

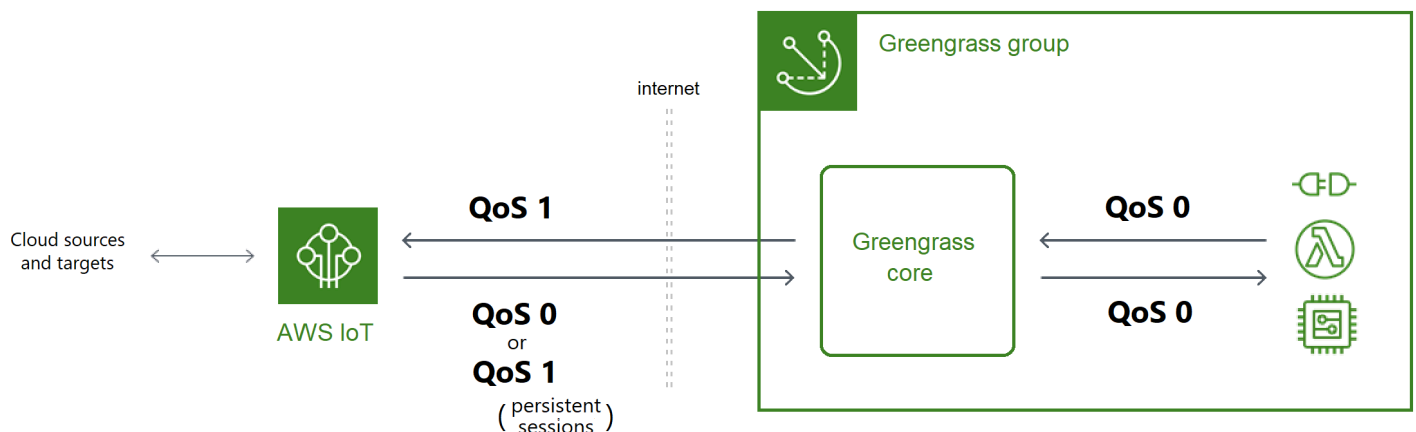
- [the section called “使用 發佈、訂閱、取消訂閱 MQTT 連線操作的逾時 AWS 雲端”](#)

Note

OPC-UA 是一個用於工業通訊的資訊交換標準。若要在 Greengrass 核心上實作對 OPC-UA 的支援，您可以使用 [IoT SiteWise 連接器](#)。連接器會將工業裝置資料從 OPC-UA 伺服器傳送至其中的資產屬性 AWS IoT SiteWise。

訊息服務品質

AWS IoT Greengrass 根據您的組態和通訊的目標和方向，支援服務品質 (QoS) 層級 0 或 1。Greengrass 核心做為與通訊的用戶端，AWS IoT Core 以及用於在本機網路上通訊的訊息中介裝置。



如需 MQTT 和 QoS 的詳細資訊，請參閱 MQTT 網站上的 [入門](#)。

與通訊 AWS 雲端

- 使用 QoS 1 的傳出訊息

核心會使用 QoS 1 傳送目標 AWS 雲端 的訊息。AWS IoT Greengrass 會使用 MQTT 訊息佇列來處理這些訊息。如果未由 確認訊息傳遞 AWS IoT，訊息會經過多工緩衝處理，以便稍後重試。如果佇列已滿，則無法重試訊息。訊息傳遞確認有助於將間歇性連線造成的資料遺失降到最低。

由於傳出訊息 AWS IoT 要使用 QoS 1，Greengrass 核心可以傳送訊息的最大速率取決於核心與 之間的延遲 AWS IoT。每次核心傳送訊息時，都會等待 AWS IoT 確認訊息後再傳送下一個訊

息。例如，如果核心與其之間的往返時間為 AWS 區域 50 毫秒，則核心每秒最多可傳送 20 則訊息。當您選擇核心連線 AWS 區域 的 時，請考慮此行為。若要將大量 IoT 資料擷取至 AWS 雲端，您可以使用[串流管理員](#)。

如需 MQTT 訊息佇列的詳細資訊，包括如何設定本機儲存快取，以保留目的地為 AWS 雲端 目標的訊息，請參閱 [the section called “MQTT 訊息佇列”](#)。

- 使用 QoS 0 (預設值) 或 QoS 1 的傳入訊息

根據預設，核心會使用 QoS 0 訂閱 AWS 雲端 來源的訊息。如果您啟用持久性工作階段，該核心會使用 QoS 1 進行訂閱。這有助於將間歇連線所造成的資料遺失降至最低。若要管理這些訂閱的 QoS，請在本機多工緩衝處理器系統元件上設定持久性設定。

如需詳細資訊，包括如何啟用核心以建立具有 AWS 雲端 目標的持久性工作階段，請參閱 [the section called “使用的 MQTT 持久性工作階段 AWS IoT Core”](#)。

與本機目標的通訊

所有本機通訊都使用 QoS 0。核心會嘗試傳送訊息至本機目標，該目標可以是 Greengrass Lambda 函數、連接器或[用戶端裝置](#)。核心不會存放訊息或確認交付。您可以在元件之間任意刪除訊息。

Note

雖然 Lambda 函數之間的直接通訊不使用 MQTT 訊息，但行為是相同的。

雲端目標的 MQTT 訊息佇列

目標的 MQTT 訊息 AWS 雲端 會排入佇列等待處理。佇列的訊息依先進先出 (FIFO) 順序處理。處理訊息並發佈至 後 AWS IoT Core，訊息會從佇列中移除。

根據預設，Greengrass 核心會存放在以 AWS 雲端 目標為目標的記憶體未處理訊息中。您可以設定該核心，以將未處理的訊息改存放於本機儲存快取中。與記憶體內儲存不同，本機儲存快取能夠在核心重新啟動（例如，在群組部署或裝置重新啟動之後）之間保留，因此 AWS IoT Greengrass 可以繼續處理訊息。您也可以設定儲存大小。

Warning

Greengrass 核心可能會在失去連線時將重複的 MQTT 訊息排入佇列，因為它會在 MQTT 用戶端偵測到離線之前重試發佈操作。若要避免雲端目標重複的 MQTT 訊息，請將核

心 `keepAlive` 的值設定為小於其 `mqttOperationTimeout` 值的一半。如需詳細資訊，請參閱 [AWS IoT Greengrass 核心組態檔案](#)。

AWS IoT Greengrass 使用多工緩衝處理系統元件 (`GGCloudSpoolerLambda` 函數) 來管理訊息佇列。您可以使用下列 `GGCloudSpooler` 環境變數來設定儲存設定。

- `GG_CONFIG_STORAGE_TYPE`。訊息佇列的位置。以下為有效值：
 - `FileSystem`。將未處理的訊息儲存在實體核心裝置的磁碟本機儲存快取中。核心重新啟動時，會保留佇列中的訊息以待處理。訊息處理後會移除。
 - `Memory` (預設)。將未處理的訊息存放於記憶體。核心重新啟動時，佇列的訊息會遺失。

此選項針對具有有限硬體功能的裝置進行最佳化。若使用此組態，建議您在服務中斷最低時部署群組或重新啟動裝置。

- `GG_CONFIG_MAX_SIZE_BYTES`。儲存大小，以位元組計。此值可以是小於或等於 262144 (256 KB) 的任何非負整數；較小的值可防止 AWS IoT Greengrass Core 軟體啟動。預設大小為 2.5 MB。超過大小限制時，舊佇列訊息會由新訊息取代。

Note

此功能適用於 AWS IoT Greengrass Core v1.6 和更新版本。舊版使用佇列大小為 2.5 MB 的記憶體內儲存。您不能設定舊版的儲存設定。

在本機儲存快取訊息

您可以設定 AWS IoT Greengrass 將訊息快取至檔案系統，使其在核心重新啟動期間持續存在。若要執行此操作，請部署函數定義版本，其中 `GGCloudSpooler` 函數會將儲存類型設定為 `FileSystem`。您必須使用 AWS IoT Greengrass API 來設定本機儲存快取。您無法在主控制台執行這項操作。

下列程序使用 [create-function-definition-version](#) CLI 命令來設定多工緩衝處理程式，將排入佇列的訊息儲存至檔案系統。這還會設定一個 2.6 MB 大小的佇列。

1. 取得目標 Greengrass 群組 ID 和目標群組版本 ID。此程序假設這是最新的群組和群組版本。下列查詢會傳回最近建立的群組。


```
aws greengrass list-groups --query "reverse(sort_by(Groups, &CreationTimestamp))
[0]"
```

或者，您可以依名稱查詢。群組名稱不需要是唯一名稱，因此可能會傳回多個群組。

```
aws greengrass list-groups --query "Groups[?Name=='MyGroup']"
```

 Note

您也可以在 AWS IoT 主控台中找到這些值。群組 ID 會顯示在群組的 Settings (設定) 頁面上。群組版本 IDs 會顯示在群組的部署索引標籤上。

2. 從輸出中的目標群組複製 Id 和 LatestVersion 值。
3. 取得最新的群組版本。
 - 將 *group-id* 取代為您複製 Id 的。
 - 將 *latest-group-version-id* 取代為您複製 LatestVersion 的。

```
aws greengrass get-group-version \
--group-id group-id \
--group-version-id latest-group-version-id
```

4. 從輸出中的 Definition 物件，複製 CoreDefinitionVersionArn 以及所有其他群組元件的 ARN，但 FunctionDefinitionVersionArn 除外。建立新群組版本時，您會需要使用這些數值。
5. 從輸出中的 FunctionDefinitionVersionArn，複製函數定義的 ID。該 ID 是 ARN 中跟在 functions 區段後面的 GUID，如下列範例所示。

```
arn:aws:greengrass:us-west-2:123456789012:/greengrass/
definition/functions/bcfc6b49-beb0-4396-b703-6dEXAMPLEcu5/
versions/0f7337b4-922b-45c5-856f-1aEXAMPLEsf6
```

 Note

或者，您可以執行 [create-function-definition](#) 命令，然後從輸出複製 ID 來建立函數定義。

6. 新增函數定義版本至函數定義。

- 將 *function-definition-id* 取代為您為函數定義複製的 Id。
- 將 *arbitrary-function-id* 取代為函數的名稱，例如 **spooler-function**。
- 將您想要在此版本中包含的任何 Lambda 函數新增至 functions 陣列。您可以使用 [get-function-definition-version](#) 命令，從現有的函數定義版本取得 Greengrass Lambda 函數。

 Warning

務必為 GG_CONFIG_MAX_SIZE_BYTES 指定大於或等於 262144 的值。較小的大小可防止 AWS IoT Greengrass Core 軟體啟動。

```
aws greengrass create-function-definition-version \  
--function-definition-id function-definition-id \  
--functions '[{"FunctionArn":  
  "arn:aws:lambda:::function:GGCloudSpooler:1", "FunctionConfiguration":  
  {"Environment": {"Variables":  
    {"GG_CONFIG_MAX_SIZE_BYTES": "2621440", "GG_CONFIG_STORAGE_TYPE": "FileSystem"}}, "Executable":  
    "spooler", "MemorySize": 32768, "Pinned": true, "Timeout": 3}, "Id": "arbitrary-  
function-id"}]'
```

 Note

如果您先前已將 GG_CONFIG_SUBSCRIPTION_QUALITY 環境變數設定為 [支援的持久性工作階段 AWS IoT Core](#)，請將其包含在此函數執行個體中。

7. 從輸出複製函數定義版本的 Arn。
8. 建立包含系統 Lambda 函數的群組版本。

- 為該群組將 *group-id* 取代為 Id。
- 將 *core-definition-version-arn* 取代為您從最新群組版本所複製的 CoreDefinitionVersionArn。
- 將 *function-definition-version-arn* 取代為您為功能定義版本所複製的 Arn。
- 取代您已從最新群組版本複製之其他群組元件 (例如 SubscriptionDefinitionVersionArn 或 DeviceDefinitionVersionArn) 的 ARN。
- 移除任何未使用的參數。比如說若您的群組版本不包含任何資源，則請移除 `--resource-definition-version-arn`。

```
aws greengrass create-group-version \  
--group-id group-id \  
--core-definition-version-arn core-definition-version-arn \  
--function-definition-version-arn function-definition-version-arn \  
--device-definition-version-arn device-definition-version-arn \  
--logger-definition-version-arn logger-definition-version-arn \  
--resource-definition-version-arn resource-definition-version-arn \  
--subscription-definition-version-arn subscription-definition-version-arn
```

9. 從輸出複製 Version。這是新群組版本的 ID。

10. 部署具有新群組版本的群組。

- 將 *group-id* 取代為您為群組所複製的 Id。
- 將 *group-version-id* 取代為您為新群組版本所複製的 Version。

```
aws greengrass create-deployment \  
--group-id group-id \  
--group-version-id group-version-id \  
--deployment-type NewDeployment
```

若要更新儲存設定，您可以使用 AWS IoT Greengrass API 來建立新的函數定義版本，其中包含具有更新組態的 GGCloudSpooler 函數。然後，請將函數定義版本新增至新群組版本 (以及其他群組元件) 和部署群組的版本中。如果想要還原預設組態，您可以部署一個不包含 GGCloudSpooler 函數的函數定義版本。

此系統 Lambda 函數不會顯示在主控台中。然而，在函數新增至最新的群組版本後，除非您使用 API 執行取代或移除作業，否則其將包含在從主控台進行的部署中。

使用的 MQTT 持久性工作階段 AWS IoT Core

此功能適用於 AWS IoT Greengrass Core v1.10 和更新版本。

Greengrass 核心可以與 AWS IoT 訊息中介裝置建立持久性工作階段。持久性工作階段是持續的連線，允許核心在離線時接收傳送的訊息。該核心是連線中的用戶端。

在持久性工作階段中，AWS IoT 訊息中介裝置會儲存核心在連線期間所做的所有訂閱。如果核心中斷連線，AWS IoT 訊息中介裝置會儲存未確認和發佈為 QoS 1 的新訊息，並以本機目標為目標，例如 Lambda 函數和[用戶端裝置](#)。當核心重新連線時，持久性工作階段會繼續，且 AWS IoT 訊息中介裝置會以每秒 10 則訊息的最大速率將儲存的訊息傳送至核心。持久性工作階段的預設到期時間為 1 小時，並在訊息代理程式偵測到核心中斷連線時開始。如需詳細資訊，請參閱《AWS IoT 開發人員指南》中的[MQTT 持久性工作階段](#)。

AWS IoT Greengrass 使用多工緩衝處理系統元件 (GGCloudSpoolerLambda 函數) 來建立具有 AWS IoT 作為來源的訂閱。您可以使用下列 GGCloudSpooler 環境變數來設定持久性工作階段。

- GG_CONFIG_SUBSCRIPTION_QUALITY。以 AWS IoT 做為來源的訂閱品質。以下為有效值：
 - AtMostOnce (預設)。停用持久性工作階段。使用 QoS 0 的訂閱。
 - AtLeastOncePersistent。啟用持久性工作階段。在 CONNECT 訊息中設定 cleanSession 旗標為 0，並以 QoS 1 訂閱。

核心接收到的 QoS 1 發佈訊息，會保證達到 Greengrass 協助程式的記憶體內工作佇列。核心會在訊息新增至佇列後確認訊息。從佇列到本機目標（例如 Greengrass Lambda 函數、連接器或裝置）的後續通訊會以 QoS 0 傳送。AWS IoT Greengrass 不保證可交付至本機目標。

Note

您可以使用 [maxWorkItemCount](#) 組態屬性來控制工作項目佇列的大小。例如，如果您的工作負載需要大量的 MQTT 流量，您可以提升佇列大小。

啟用持久性工作階段時，核心會開啟至少一個額外的連線，以便與 MQTT 訊息交換 AWS IoT。如需詳細資訊，請參閱[the section called “與進行 MQTT 連線的用戶端 IDs AWS IoT”](#)。

設定 MQTT 持久性工作階段

您可以將設定為搭配 AWS IoT Greengrass 使用持久性工作階段 AWS IoT Core。若要執行此操作，請部署函數定義版本，其中 GGCloudSpooler 函數會將訂閱品質設定為 `AtLeastOncePersistent`。此設定適用於您所有以 AWS IoT Core (cloud) 做為來源的訂閱。您必須使用 AWS IoT Greengrass API 來設定持久性工作階段。您無法在主控台執行這項操作。

下列程序使用 [create-function-definition-version](#) CLI 命令，將多工緩衝處理程式設定為使用持久性工作階段。在此程序中，我們假設您正在更新現有群組的最新群組版本設定。

1. 取得目標 Greengrass 群組 ID 和目標群組版本 ID。此程序假設這是最新的群組和群組版本。下列查詢會傳回最近建立的群組。

```
aws greengrass list-groups --query "reverse(sort_by(Groups, &CreationTimestamp))
[0]"
```

或者，您可以依名稱查詢。群組名稱不需要是唯一名稱，因此可能會傳回多個群組。

```
aws greengrass list-groups --query "Groups[?Name=='MyGroup']"
```

Note

您也可以在 AWS IoT 主控台中找到這些值。群組 ID 會顯示在群組的 Settings (設定) 頁面上。群組版本 IDs 會顯示在群組的部署索引標籤上。

2. 從輸出中的目標群組複製 Id 和 LatestVersion 值。
3. 取得最新的群組版本。
 - 將 *group-id* 取代為您複製 Id 的。
 - 將 *latest-group-version-id* 取代為您複製 LatestVersion 的。

```
aws greengrass get-group-version \
--group-id group-id \
--group-version-id latest-group-version-id
```

4. 從輸出中的 Definition 物件，複製 CoreDefinitionVersionArn 以及所有其他群組元件的 ARN，但 FunctionDefinitionVersionArn 除外。建立新群組版本時，您會需要使用這些數值。

5. 從輸出中的 `FunctionDefinitionVersionArn`，複製函數定義的 ID。該 ID 是 ARN 中跟在 `functions` 區段後面的 GUID，如下列範例所示。

```
arn:aws:greengrass:us-west-2:123456789012:/greengrass/
definition/functions/bcfc6b49-beb0-4396-b703-6dEXAMPLEcu5/
versions/0f7337b4-922b-45c5-856f-1aEXAMPLEsf6
```

 Note

或者，您可以執行 [create-function-definition](#) 命令來建立函數定義，然後從輸出複製 ID。

6. 新增函數定義版本至函數定義。
 - 將 *function-definition-id* 取代為您為函數定義複製的 Id。
 - 將 *arbitrary-function-id* 取代為函數的名稱，例如 **spooler-function**。
 - 將您想要在此版本中包含的任何 Lambda 函數新增至 `functions` 陣列。您可以使用 [get-function-definition-version](#) 命令，從現有的函數定義版本取得 Greengrass Lambda 函數。

```
aws greengrass create-function-definition-version \
--function-definition-id function-definition-id \
--functions '[{"FunctionArn":
  "arn:aws:lambda:::function:GGCloudSpooler:1","FunctionConfiguration":
  {"Environment": {"Variables":
{"GG_CONFIG_SUBSCRIPTION_QUALITY":"AtLeastOncePersistent"}},"Executable":
  "spooler","MemorySize": 32768,"Pinned": true,"Timeout": 3},"Id": "arbitrary-
function-id"}]'
```

 Note

如果您先前設定 `GG_CONFIG_STORAGE_TYPE` 或 `GG_CONFIG_MAX_SIZE_BYTES` 環境變數來[定義儲存設定](#)，請將它們包含在此函數執行個體中。

7. 從輸出複製函數定義版本的 Arn。
8. 建立包含系統 Lambda 函數的群組版本。

- 為該群組將 *group-id* 取代為 Id。
- 將 *core-definition-version-arn* 取代為您從最新群組版本所複製的 CoreDefinitionVersionArn。
- 將 *function-definition-version-arn* 取代為您為功能定義版本所複製的 Arn。
- 取代您已從最新群組版本複製之其他群組元件 (例如 SubscriptionDefinitionVersionArn 或 DeviceDefinitionVersionArn) 的 ARN。
- 移除任何未使用的參數。比如說若您的群組版本不包含任何資源，則請移除 --resource-definition-version-arn。

```
aws greengrass create-group-version \  
--group-id group-id \  
--core-definition-version-arn core-definition-version-arn \  
--function-definition-version-arn function-definition-version-arn \  
--device-definition-version-arn device-definition-version-arn \  
--logger-definition-version-arn logger-definition-version-arn \  
--resource-definition-version-arn resource-definition-version-arn \  
--subscription-definition-version-arn subscription-definition-version-arn
```

9. 從輸出複製 Version。這是新群組版本的 ID。

10. 部署具有新群組版本的群組。

- 將 *group-id* 取代為您為群組所複製的 Id。
- 將 *group-version-id* 取代為您為新群組版本所複製的 Version。

```
aws greengrass create-deployment \  
--group-id group-id \  
--group-version-id group-version-id \  
--deployment-type NewDeployment
```

11. (選用) 提高核心組態檔案中的 [maxWorkItemCount](#) 屬性。這有助核心處理增加的 MQTT 流量以及與本機目標之間的通訊。

若要使用這些組態變更更新核心，您可以使用 AWS IoT Greengrass API 來建立新的函數定義版本，其中包含具有更新組態的 GGCloudSpooler 函數。然後，請將函數定義版本新增至新群組版本 (以及其他群組元件) 和部署群組的版本中。如果想要還原預設組態，您可以建立一個不包含 GGCloudSpooler 函數的函數定義版本。

此系統 Lambda 函數不會顯示在主控台中。然而，在函數新增至最新的群組版本後，除非您使用 API 執行取代或移除作業，否則其將包含在從主控台進行的部署中。

與進行 MQTT 連線的用戶端 IDs AWS IoT

此功能適用於 AWS IoT Greengrass Core v1.8 和更新版本。

Greengrass 核心會開啟與的 MQTT 連線 AWS IoT Core，以進行陰影同步和憑證管理等操作。對於這些連線，核心會根據核心物件名稱產生可預測的用戶端 ID。可預測 IDs 可與監控、稽核和定價功能搭配使用，包括 AWS IoT Device Defender 和 [AWS IoT 生命週期事件](#)。您也可以建立關於可預測用戶端 ID 的邏輯 (例如根據憑證屬性建立[訂閱政策範本](#))。

GGC v1.9 and later

兩個 Greengrass 系統元件會開啟 MQTT 連線 AWS IoT Core。這些元件使用下列模式來產生連線的用戶端 ID。

作業	用戶端 ID 模式
部署	<i>core-thing-name</i> 範例：MyCoreThing 此用戶端 ID 可用來連線、中斷連線、訂閱和取消訂閱生命週期事件通知。
Subscriptions	<i>core-thing-name -cn</i> 範例：MyCoreThing-c01 <i>n</i> 是一個整數，從 00 開始，並隨著每個新連線遞增，最大數量為 250。連線數量取決於與 AWS IoT Core 影子狀態同步的裝置數量（每個群組最多 2,500 個），以及將 cloud 作為其來源的訂閱數量（每個群組最多 10,000 個）。 多工緩衝處理系統元件會與連線 AWS IoT Core，以與雲端來源或目標交換訂閱的訊息。多工緩衝處理也會做為 AWS IoT Core 與本機

作業	用戶端 ID 模式
	影子服務和裝置憑證管理器之間訊息交換的代理。

若要計算每個群組的 MQTT 連線數，請使用下列公式：

number of MQTT connections per group = number of connections for Deployment Agent + number of connections for Subscriptions

其中：

- 部署代理程式的連線數 = 1。
- 訂閱的連線數 = (2 subscriptions for supporting certificate generation + number of MQTT topics in AWS IoT Core + number of device shadows synced) / 50。
- 其中，50 = 每個 AWS IoT Core 連線可支援的訂閱數量上限。

Note

如果您為訂閱啟用[持久性工作階段](#) AWS IoT Core，核心會開啟至少一個要在持久性工作階段中使用的額外連線。系統組件不支援持久性工作階段，因此它們不能共用該連線。

若要減少 MQTT 連線數量並協助降低成本，您可以使用本機 Lambda 函數彙總邊緣的資料。然後，您將彙總的資料傳送至 AWS 雲端。因此，您在中使用較少的 MQTT 主題 AWS IoT Core。如需詳細資訊，請參閱[AWS IoT Greengrass 定價](#)。

GGC v1.8

有數個 Greengrass 系統元件會開啟與 AWS IoT Core 的 MQTT 連線。這些元件使用下列模式來產生連線的用戶端 ID。

作業	用戶端 ID 模式
部署	<i>core-thing-name</i> 範例：MyCoreThing

作業	用戶端 ID 模式
	此用戶端 ID 可用來連線、中斷連線、訂閱和取消訂閱生命週期事件通知。
與 交換 MQTT 訊息 AWS IoT Core	<i>core-thing-name</i> -spr 範例：MyCoreThing-spr
影子同步	<i>core-thing-name</i> -snn 範例：MyCoreThing-s01 <i>nn</i> 是從 00 起始的整數，隨著每個新連線遞增，最大到 03。連線數目取決於與影子狀態同步的裝置數目（每個群組最多 200 個裝置）AWS IoT Core（每個連線最多 50 個訂閱）。
裝置憑證管理	<i>core-thing-name</i> -dcm 範例：MyCoreThing-dcm

Note

同時連線中使用重複的用戶端 ID，可能會導致連線-中斷連線的無限迴圈。如果另一個裝置已硬式編碼為在連線中使用核心裝置名稱做為用戶端 ID，就會發生這種情形。如需詳細資訊，請參閱此[故障診斷步驟](#)。

Greengrass 裝置也與 AWS IoT Device Management 的機群索引服務完全整合。這可讓您根據裝置屬性、影子狀態和雲端連線狀態，索引和搜尋裝置。例如，Greengrass 裝置使用實物名稱當做用戶端 ID 建立至少一個連線，因此您可以使用裝置連線索引，探索哪些 Greengrass 裝置目前與 AWS IoT Core 連線或中斷連線。如需詳細資訊，請參閱 [《開發人員指南》中的機群索引服務](#)。AWS IoT

設定本機簡訊的 MQTT 連接埠

此功能需要 AWS IoT Greengrass Core v1.10 或更新版本。

Greengrass 核心做為本機 Lambda 函數、連接器和[用戶端裝置](#)之間 MQTT 訊息的本機訊息中介裝置。根據預設，核心會使用連接埠 8883 以供區域網路上的 MQTT 流量使用。您可能會想變更連接埠，以避免與在連接埠 8883 上執行的其他軟體發生衝突。

設定核心用於本機 MQTT 流量的連接埠號碼

1. 執行下列命令來停止 Greengrass 協助程式：

```
cd /greengrass-root/ggc/core/  
sudo ./greengrassd stop
```

2. 開放 `greengrass-root/config/config.json` 由 su 使用者編輯。
3. 在 `coreThing` 物件中，新增 `ggMqttPort` 屬性並將值設為您要使用的連接埠號碼。有效值為 1024 到 65535。下列範例會將連接埠號碼設定為 9000。

```
{  
  "coreThing" : {  
    "caPath" : "root.ca.pem",  
    "certPath" : "12345abcde.cert.pem",  
    "keyPath" : "12345abcde.private.key",  
    "thingArn" : "arn:aws:iot:us-west-2:123456789012:thing/core-thing-name",  
    "iotHost" : "abcd123456wxyz-ats.iot.us-west-2.amazonaws.com",  
    "ggHost" : "greengrass-ats.iot.us-west-2.amazonaws.com",  
    "ggMqttPort" : 9000,  
    "keepAlive" : 600  
  },  
  ...  
}
```

4. 啟動協助程式。

```
cd /greengrass-root/ggc/core/  
sudo ./greengrassd start
```

5. 如果該核心已啟用[自動 IP 偵測](#)，則組態即完成。

如果自動 IP 偵測未啟用，您必須更新核心的連線資訊。這可讓用戶端裝置在探索操作期間接收正確的連接埠號碼，以取得核心連線資訊。您可以使用 AWS IoT 主控台或 AWS IoT Greengrass API 來更新核心連線資訊。在此程序中，您只會更新連接埠號碼。核心的本機 IP 地址會保持不變。

更新核心 (主控台) 的連線資訊

1. 在群組組態頁面上，選擇 Greengrass 核心。
2. 在核心詳細資訊頁面上，選擇 MQTT 代理程式端點索引標籤。
3. 選擇管理端點，然後選擇新增端點
4. 輸入您目前的本機 IP 地址和新的連接埠號碼。下列範例會設定 IP 地址 192.168.1.8 的連接埠號碼 9000。
5. 移除已淘汰的端點，然後選擇 Update (更新)

更新核心 (API) 的連線資訊

- 使用 [UpdateConnectivityInfo](#) 動作。下列範例使用 AWS CLI 中的 update-connectivity-info 來設定 IP 地址 192.168.1.8 的連接埠號碼 9000。

```
aws greengrass update-connectivity-info \
  --thing-name "MyGroup_Core" \
  --connectivity-info "[{\"Metadata\":\"\",\"PortNumber\":9000,
  \"HostAddress\":\"192.168.1.8\",\"Id\":\"localIP_192.168.1.8\"},{\"Metadata
  \":\"\",\"PortNumber\":8883,\"HostAddress\":\"127.0.0.1\",\"Id\":
  \"localhost_127.0.0.1_0\"}]\""
```

Note

您也可以設定核心用於 MQTT 訊息的連接埠 AWS IoT Core。如需詳細資訊，請參閱[the section called “連線至連接埠 443 或透過網路代理”](#)。

使用 發佈、訂閱、取消訂閱 MQTT 連線操作的逾時 AWS 雲端

此功能可在 AWS IoT Greengrass v1.10.2 或更新版本中使用。

您可以設定允許 Greengrass 核心在與 AWS IoT Core 的 MQTT 連線中完成發佈、訂閱或取消訂閱操作的時間量 (以秒為單位)。如果因為頻寬限制或高延遲而造成操作逾時，您可能會想要調整此設定。若要在 [config.json](#) 檔案中設定此設定，請新增或變更 coreThing 物件中的 mqttOperationTimeout 屬性。例如：

```
{
```

```
"coreThing": {  
  "mqttOperationTimeout": 10,  
  "caPath": "root-ca.pem",  
  "certPath": "hash.cert.pem",  
  "keyPath": "hash.private.key",  
  ...  
},  
...  
}
```

預設逾時時間為 5 秒。最小逾時為 5 秒。

啟動自動 IP 偵測

您可以設定 AWS IoT Greengrass 讓 Greengrass 群組中的用戶端裝置自動探索 Greengrass 核心。啟用時，核心會監控其 IP 地址的變更。如果地址變更，核心會發佈更新的地址清單。這些地址可供與核心位於相同 Greengrass 群組中的用戶端裝置使用。

Note

用戶端裝置 AWS IoT 的政策必須授予 `greengrass:Discover` 許可，允許裝置擷取核心的連線資訊。如需有關此政策陳述式的詳細資訊，請參閱 [the section called “探索授權”](#)。

若要從 AWS IoT Greengrass 主控台啟用此功能，請在第一次部署 Greengrass 群組時選擇自動偵測。您也可以選擇 Lambda 函數索引標籤並選擇 IP 偵測器，在群組組態頁面上啟用或停用此功能。如果選取自動偵測和覆寫 MQTT 代理程式端點，則會啟用自動 IP 偵測。

若要使用 AWS IoT Greengrass API 管理自動探索，您必須設定 IPDetector 系統 Lambda 函數。下列程序說明如何使用 [create-function-definition-version](#) CLI 命令來設定 Greengrass 核心的自動探索。

1. 取得目標 Greengrass 群組 ID 和目標群組版本 ID。此程序假設這是最新的群組和群組版本。下列查詢會傳回最近建立的群組。

```
aws greengrass list-groups --query "reverse(sort_by(Groups, &CreationTimestamp))  
[0]"
```

或者，您可以依名稱查詢。群組名稱不需要是唯一名稱，因此可能會傳回多個群組。

```
aws greengrass list-groups --query "Groups[?Name=='MyGroup']"
```

Note

您也可以在此 AWS IoT 主控台找到這些值。群組 ID 會顯示在群組的 Settings (設定) 頁面上。群組版本 IDs 會顯示在群組的部署索引標籤上。

2. 從輸出中的目標群組複製 Id 和 LatestVersion 值。

3. 取得最新的群組版本。

- 將 *group-id* 取代為您複製 Id 的。
- 將 *latest-group-version-id* 取代為您複製 LatestVersion 的。

```
aws greengrass get-group-version \  
--group-id group-id \  
--group-version-id latest-group-version-id
```

4. 從輸出中的 Definition 物件，複製 CoreDefinitionVersionArn 以及所有其他群組元件的 ARN，但 FunctionDefinitionVersionArn 除外。建立新群組版本時，您會需要使用這些數值。

5. 從輸出中的 FunctionDefinitionVersionArn，複製函數定義的 ID 和函數定義版本：

```
arn:aws:greengrass:region:account-id:/greengrass/groups/function-definition-id/  
versions/function-definition-version-id
```

Note

您可以選擇性地執行 [create-function-definition](#) 命令來建立函數定義，然後從輸出複製 ID。

6. 使用 [get-function-definition-version](#) 命令來取得目前定義狀態。使用您為函數定義複製的 *function-definition-id*。例如，*4d941bc7-92a1-4f45-8d64-EXAMPLEf76c3*。

```
aws greengrass get-function-definition-version  
--function-definition-id function-definition-id  
--function-definition-version-id function-definition-version-id
```

請記下列出的函數組態。您需要在建立新的函數定義版本中包含這些組態，以防止目前的定義設定遺失。

7. 新增函數定義版本至函數定義。

- 將 *function-definition-id* 取代為您為函數定義複製的 Id。例如，*4d941bc7-92a1-4f45-8d64-EXAMPLEf76c3*。
- 將 *arbitrary-function-id* 取代為函數的名稱，例如 **auto-detection-function**。
- 將您想要在此版本中包含的所有 Lambda 函數新增至 `functions` 陣列，例如上一個步驟中列出的任何。

```
aws greengrass create-function-definition-version \  
--function-definition-id function-definition-id \  
--functions  
  '[{"FunctionArn":"arn:aws:lambda::function:GGIPDetector:1","Id": "arbitrary-function-id", "FunctionConfiguration":  
  {"Pinned":true,"MemorySize":32768,"Timeout":3}}]' \  
--region us-west-2
```

8. 從輸出複製函數定義版本的 Arn。

9. 建立包含系統 Lambda 函數的群組版本。

- 為該群組將 *group-id* 取代為 Id。
- 將 *core-definition-version-arn* 取代為您從最新群組版本所複製的 `CoreDefinitionVersionArn`。
- 將 *function-definition-version-arn* 取代為您為功能定義版本所複製的 Arn。
- 取代您已從最新群組版本複製之其他群組元件 (例如 `SubscriptionDefinitionVersionArn` 或 `DeviceDefinitionVersionArn`) 的 ARN。
- 移除任何未使用的參數。比如說若您的群組版本不包含任何資源，則請移除 `--resource-definition-version-arn`。

```
aws greengrass create-group-version \  
--group-id group-id \  
--core-definition-version-arn core-definition-version-arn \  
--function-definition-version-arn function-definition-version-arn \  
--device-definition-version-arn device-definition-version-arn \  
--logger-definition-version-arn logger-definition-version-arn \  
--region us-west-2
```

```
--resource-definition-version-arn resource-definition-version-arn \  
--subscription-definition-version-arn subscription-definition-version-arn
```

10. 從輸出複製 Version。這是新群組版本的 ID。

11. 部署具有新群組版本的群組。

- 將 *group-id* 取代為您為群組所複製的 Id。
- 將 *group-version-id* 取代為您為新群組版本所複製的 Version。

```
aws greengrass create-deployment \  
--group-id group-id \  
--group-version-id group-version-id \  
--deployment-type NewDeployment
```

如果想要手動輸入 Greengrass 核心的 IP 地址，您可以使用不包含 IPDetector 函數的不同函數定義來完成此教學課程。這將防止偵測函數尋找並自動輸入您的 Greengrass 核心 IP 地址。

Lambda 主控台中看不到此系統 Lambda 函數。將函數新增至最新的群組版本後，它會包含在您從主控台執行的部署中，除非您使用 API 將其取代或移除。

設定初始化系統，啟動 Greengrass 協助程式

設定您的初始化系統在開機時啟動 Greengrass 協助程式是很好的實務練習，尤其是在管理大型叢集裝置時。

Note

如果您使用 apt 安裝 AWS IoT Greengrass Core 軟體，您可以使用系統化指令碼在開機時啟用啟動。如需詳細資訊，請參閱[the section called “使用 systemd 指令碼以管理 Greengrass 協助程式生命週期”](#)。

有不同類型的初始化系統，例如 initd、systemd、SystemV，以及其所使用類似的組態參數。以下範例為 systemd 的服務檔案：此 Type 參數設為 forking 是因為 greengrassd (用於啟動 Greengrass) 分岔 Greengrass 協助程式程序，而 Restart 參數設為 on-failure，若 Greengrass 故障時將導引 systemd 重新啟動 Greengrass。

Note

若要查看您的裝置是否使用 systemd，請依照[模組 1](#) 所述執行 `check_ggc_dependencies` 指令碼。然後，若要使用 systemd，請確定 中的 `useSystemd` 參數 [config.json](#) 設定為 `yes`。

```
[Unit]
Description=Greengrass Daemon

[Service]
Type=forking
PIDFile=/var/run/greengrassd.pid
Restart=on-failure
ExecStart=/greengrass/ggc/core/greengrassd start
ExecReload=/greengrass/ggc/core/greengrassd restart
ExecStop=/greengrass/ggc/core/greengrassd stop

[Install]
WantedBy=multi-user.target
```

另請參閱

- [什麼是 AWS IoT Greengrass？](#)
- [the section called “支援平台和需求”](#)
- [入門 AWS IoT Greengrass](#)
- [the section called “群組物件模型概觀”](#)
- [the section called “硬體安全整合”](#)

AWS IoT Greengrass Version 1 維護政策

使用此 AWS IoT Greengrass V1 維護政策來了解 AWS IoT Greengrass V1 服務和 AWS IoT Greengrass Core 軟體 v1.x 的不同維護和更新層級。

主題

- [AWS IoT Greengrass 版本控制方案](#)
- [Core AWS IoT Greengrass 軟體主要版本的生命週期階段](#)
- [AWS IoT Greengrass Core 軟體的維護政策](#)
- [棄用排程](#)
- [Greengrass 核心裝置上的 AWS Lambda 函數支援政策](#)
- [AWS IoT Device Tester for 的支援政策 AWS IoT Greengrass V1](#)
- [結束維護排程](#)

AWS IoT Greengrass 版本控制方案

AWS IoT Greengrass 使用 Core 軟體的 AWS IoT Greengrass [語意版本控制](#)。語意版本遵循 major.minor.patch 號碼系統。功能和 API 變更的主要版本會遞增，這些變更與先前的主要版本不回溯相容。新增回溯相容功能之版本的次要版本會遞增。安全性修補程式或錯誤修正的修補程式版本增量。自第一個主要版本 v1.0.0 以來，AWS IoT Greengrass 已發行 11 個次要版本的 AWS IoT Greengrass Core 軟體 v1.x，其中 v1.11.6 是最新版本。我們建議您將 AWS IoT Greengrass Core 軟體更新為最新的可用版本，以利用新的功能、增強功能和錯誤修正。

2020 年 12 月，AWS IoT Greengrass 發佈了第一個主要版本更新。此更新包含 AWS IoT Greengrass V2 服務和 AWS IoT Greengrass 2.0.3 版的核心軟體。對於新應用程式，強烈建議您使用 AWS IoT Greengrass Version 2 和 AWS IoT Greengrass Core 軟體 v2.x。第 2 版接收新功能，包括所有主要 V1 功能，並支援額外平台和持續部署至大型裝置機群。如需詳細資訊，請參閱[什麼是 AWS IoT Greengrass V2 ?](#)。

Core AWS IoT Greengrass 軟體主要版本的生命週期階段

每個 AWS IoT Greengrass Core 軟體的主要版本都有下列三個循序生命週期階段。每個生命週期階段都會在初始發行日期後的一段時間內提供不同層級的維護。

- 發行階段 – AWS IoT Greengrass 可能會發行下列更新：

- 次要版本更新，可為現有功能提供新功能或增強功能
- 提供安全性修補程式和錯誤修正的修補程式版本更新
- 維護階段 – AWS IoT Greengrass 可能會發行修補程式版本更新，以提供安全性修補程式和錯誤修正。AWS IoT Greengrass 在維護階段期間，不會發行新功能或增強現有功能。
- 延長生命週期 – AWS IoT Greengrass 不會發佈提供功能、增強現有功能、安全修補程式或錯誤修正的更新。不過，AWS 雲端端點和 API 操作將保持可用，並根據[AWS IoT Greengrass 服務水準協議](#)運作。執行 AWS IoT Greengrass Core 軟體 v1.x 的裝置可以繼續連線至 AWS 雲端並操作。

在主要版本的延長生命週期結束後 AWS IoT Greengrass，端點 AWS 雲端和 API 操作將被棄用且不再可用。執行 AWS IoT Greengrass Core 軟體 v1.x 的裝置將無法連線至 AWS 雲端服務以運作。

AWS IoT Greengrass Core 軟體的維護政策

Core AWS IoT Greengrass 軟體 v1.x 於 2023 年 6 月 30 日進入延長生命週期階段。在此日期之後，AWS IoT Greengrass Core 軟體 v1.x 將維持在延長生命週期階段，直到進一步通知為止。

Core AWS IoT Greengrass 軟體 v2.x 目前處於發行階段，並會維持在發行階段，直到進一步通知為止。會 AWS IoT Greengrass 繼續將新功能和增強功能新增至 AWS IoT Greengrass Core 軟體 v2.x。例如，AWS IoT Greengrass Core 軟體 v2.5.0 中 AWS IoT Greengrass 發行的 Windows 支援。會在發行日期後至少 1 年，針對所有次要版本的 AWS IoT Greengrass Core v2.x AWS IoT Greengrass 發行安全性修補程式和錯誤修正。如需詳細資訊，請參閱 [中的新功能 AWS IoT Greengrass V2](#)。

維護階段排程

2023 年 6 月 30 日，AWS IoT Greengrass Core 軟體 v1.11.x 的維護階段結束。2022 年 3 月 31 日，AWS IoT Greengrass Core 軟體 v1.10.x 的維護階段結束。特定 AWS IoT Greengrass Core 軟體 v1.x 成品和功能的維護階段會在這些日期之前結束。如需詳細資訊，請參閱[結束維護排程](#)。

如果您有 AWS 支援計劃，則 AWS IoT Greengrass Core 軟體 v1.x 的維護階段不會影響您的 AWS 支援計劃。即使維護階段結束，您也可以繼續開啟 AWS 支援票證。如果您有問題或疑慮，請聯絡您的 AWS 支援聯絡人，或使用 AWS IoT Greengrass 標籤在 [AWS re : Post](#) 上提出問題。

棄用排程

目前，沒有計畫停止支援 AWS IoT Greengrass Core 軟體 v1.x。端點和 API AWS IoT Greengrass V1 操作將保持可用狀態，直到進一步通知為止。Core AWS IoT Greengrass 軟體 1.11.6 版於 2023 年 6

月 30 日進入延長生命週期階段。在此階段，執行 AWS IoT Greengrass Core 軟體 v1.x 的裝置可以繼續連線至 AWS IoT Greengrass V1 服務以運作，直到進一步通知為止。

如果 在未來 AWS IoT Greengrass V1 停止支援，AWS IoT Greengrass 將在發生這種情況前 12 個月提供通知。這將協助您規劃要使用的應用程式更新 AWS IoT Greengrass V2 和 AWS IoT Greengrass Core 軟體 v2.x。如需如何將應用程式更新至 V2 的詳細資訊，請參閱[從 移至 AWS IoT Greengrass V1 V2](#)。

Greengrass 核心裝置上的 AWS Lambda 函數支援政策

AWS IoT Greengrass 可讓您在 IoT 裝置上執行 AWS Lambda 函數。AWS Lambda 提供可判斷 Lambda 執行時間支援的支援政策和時間表 AWS IoT Greengrass。Lambda 執行時間到達支援階段後，AWS IoT Greengrass 也會結束對該執行時間的支援。如需詳細資訊，請參閱《AWS Lambda 開發人員指南》中的[執行期支援政策](#)。

當 Lambda 執行時間結束支援時，您無法建立或更新使用該執行時間的 Lambda 函數。不過，您可以繼續將這些 Lambda 函數部署到 Greengrass 核心裝置，並叫用已部署的 Lambda 函數。此政策也適用於 AWS IoT Greengrass V2。

AWS IoT Device Tester for 的支援政策 AWS IoT Greengrass V1

AWS IoT Device Tester (IDT) for AWS IoT Greengrass V1 可讓您驗證 AWS IoT Greengrass 裝置，並[符合資格](#)納入 [AWS Partner Device Catalog](#)。截至 2022 年 4 月 4 日，適用於的 AWS IoT Device Tester (IDT) AWS IoT Greengrass V1 不再產生已簽章的資格報告。您無法再透過 AWS IoT Greengrass V1 裝置資格計劃，讓新裝置符合[AWS Partner 在裝置目錄中](#)列出的資格。[AWS](#)雖然您無法限定 Greengrass V1 裝置的資格，但您可以繼續使用 IDT for AWS IoT Greengrass V1 來測試 Greengrass V1 裝置。我們建議您使用 [IDT for AWS IoT Greengrass V2](#) 在 Device [AWS Partner Catalog](#) 中[限定和列出 Greengrass 裝置](#)。如需詳細資訊，請參閱[AWS IoT Device Tester for 的支援政策 AWS IoT Greengrass V1](#)。

結束維護排程

下表列出 AWS IoT Greengrass Core v1.x 成品和功能的維護結束日期。如果您對維護排程或政策有任何疑問，請聯絡 [AWS Support](#)。

成品或功能	維護結束日期
Greengrass APT 儲存庫安裝	2022 年 2 月 11 日

成品或功能	維護結束日期
ML 影像分類連接器	2022 年 3 月 31 日
ML 物件偵測連接器	2022 年 3 月 31 日
ML 回饋連接器	2022 年 3 月 31 日
AWS IoT Analytics 連接器	2022 年 3 月 31 日
Twilio 通知連接器	2022 年 3 月 31 日
Splunk 整合連接器	2022 年 3 月 31 日
序列串流連接器	2022 年 3 月 31 日
ServiceNow MetricBase 整合連接器	2022 年 3 月 31 日
Raspberry Pi GPIO 連接器	2022 年 3 月 31 日
AWS IoT Greengrass 核心軟體 v1.10.x	2022 年 3 月 31 日
AWS IoT Greengrass 核心軟體 v1.x Docker 映像	2022 年 6 月 30 日
AWS IoT Greengrass 核心軟體 v1.11.x	2023 年 6 月 30 日
AWS IoT Greengrass 核心軟體 v1.11.x Snap	2023 年 12 月 31 日

Core AWS IoT Greengrass 軟體 v1.x Docker 映像的維護結束

2022 年 6 月 30 日，發佈至 Amazon Elastic Container Registry (Amazon ECR) 和 Docker Hub 的 AWS IoT Greengrass Core 軟體 v1.x Docker 映像的 AWS IoT Greengrass 結束維護。您可以繼續從 Amazon ECR 和 Docker Hub 下載這些 Docker 映像，直到 2023 年 6 月 30 日，也就是維護結束後 1 年。不過，在 2022 年 6 月 30 日結束維護之後，AWS IoT Greengrass Core 軟體 v1.x Docker 映像將不再收到安全修補程式或錯誤修正。如果您執行依賴這些 Docker 映像的生產工作負載，建議您使用 AWS IoT Greengrass 提供的 Dockerfile 建置自己的 Docker 映像。如需詳細資訊，請參閱[AWS IoT Greengrass Docker 軟體](#)。

Core AWS IoT Greengrass 軟體 v1.x APT 儲存庫的維護結束

在 2022 年 2 月 11 日，AWS IoT Greengrass 結束了從 [APT 儲存庫安裝 AWS IoT Greengrass Core 軟體 v1.x](#) 的選項維護。APT 儲存庫已在此日期移除，因此您無法再使用 APT 儲存庫來更新 AWS IoT Greengrass Core 軟體，或在新裝置上安裝 AWS IoT Greengrass Core 軟體。在您新增 AWS IoT Greengrass 儲存庫的裝置上，您必須[從來源清單中移除儲存庫](#)。建議您使用 [tar 檔案](#)更新 AWS IoT Greengrass Core 軟體 v1.x。

Core AWS IoT Greengrass 軟體 v1.11.x Snap 的維護結束

在 2023 年 12 月 31 日，AWS IoT Greengrass 將結束在 <https://> 上發佈 AWS IoT Greengrass 的核心軟體版本 1.11.x Snap 的維護。snapcraft.io 目前執行 Snap 的裝置將繼續運作，直到進一步通知為止。不過，AWS IoT Greengrass 核心 Snap 在維護結束後將不再收到安全修補程式或錯誤修正。

入門 AWS IoT Greengrass

本入門教學課程包含數個模組，旨在向您展示 AWS IoT Greengrass 基本概念並協助您開始使用 AWS IoT Greengrass。本教學涵蓋了基本概念，例如：

- 設定 AWS IoT Greengrass 核心和群組。
- 在邊緣執行 AWS Lambda 函數的部署程序。
- 將稱為用戶端 AWS IoT 裝置的裝置連接到 AWS IoT Greengrass 核心。
- 建立訂閱以允許本機 Lambda 函數、用戶端裝置和 之間的 MQTT 通訊 AWS IoT。

選擇如何開始使用 AWS IoT Greengrass

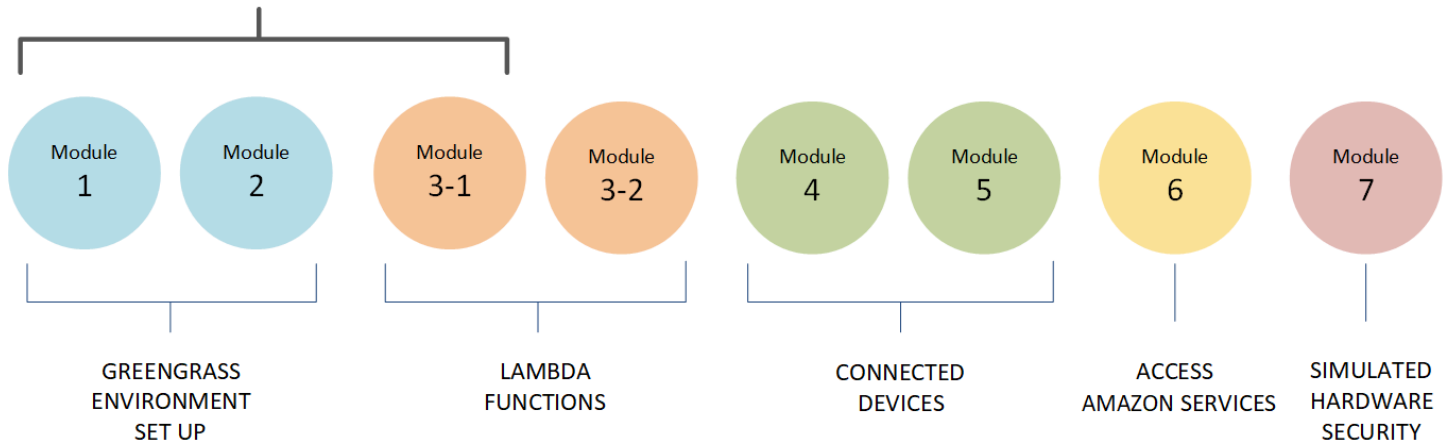
您可以選擇如何開始使用本教學來設定您的核心裝置：

- 在核心裝置上執行 [Greengrass 裝置設定](#)，這可讓您在幾分鐘內安裝 AWS IoT Greengrass 相依性以測試 Hello World Lambda 函數。這個指令碼會重現單元 1 到單元 3-1 的步驟。

- 或 -

- 逐步進行單元 1 到單元 3-1 的步驟，更仔細地檢查 Greengrass 的需求及程序。這些步驟會設定您的核心裝置、建立和設定包含 Hello World Lambda 函數的 Greengrass 群組，以及部署您的 Greengrass 群組。通常，這需要一或兩個小時來完成。

Quick Start: Greengrass Device Setup



Quick Start

[Greengrass 裝置安裝](#)會設定您的核心裝置和 Greengrass 資源。指令碼：

- 安裝 AWS IoT Greengrass 相依性。
- 下載根 CA 憑證及核心裝置憑證和金鑰。
- 在您的裝置上下載、安裝和設定 AWS IoT Greengrass 核心軟體。
- 在核心裝置上啟動 Greengrass 精靈程序。
- 視需要建立或更新 [Greengrass 服務角色](#)。
- 建立 Greengrass 群組和 Greengrass 核心。
- (選用) 建立 Hello World Lambda 函數、訂閱和本機記錄組態。
- (選用) 部署 Greengrass 群組。

單元 1 和 2

[單元 1](#) 和 [單元 2](#) 會說明如何設定您的環境。(或者，使用 [Greengrass 裝置安裝](#)來為您執行這些單元。)

- 為 Greengrass 設定您的核心裝置。
- 執行依存項目檢查程式指令碼。
- 建立 Greengrass 群組和 Greengrass 核心。
- 從 tar.gz 檔案下載並安裝最新的 AWS IoT Greengrass Core 軟體。
- 在核心上啟動 Greengrass 精靈程序。

 Note

AWS IoT Greengrass 也提供安裝 AWS IoT Greengrass Core 軟體的其他選項，包括在支援的 Debian 平台上的 apt 安裝。如需詳細資訊，請參閱[the section called “安裝 AWS IoT Greengrass 核心軟體”](#)。

單元 3-1 和 3-2

[模組 3-1](#) 和 [模組 3-2](#) 說明如何使用本機 Lambda 函數。(或者，使用 [Greengrass 裝置安裝](#) 來為您執行單元 3-1。)

- 在 中建立 Hello World Lambda 函數 AWS Lambda。
- 將 Lambda 函數新增至 Greengrass 群組。
- 建立允許 Lambda 函數與 之間進行 MQTT 通訊的訂閱 AWS IoT。
- 設定 Greengrass 系統元件和 Lambda 函數的本機記錄。
- 部署包含 Lambda 函數和訂閱的 Greengrass 群組。
- 從本機 Lambda 函數傳送訊息至 AWS IoT。
- 從 叫用本機 Lambda 函數 AWS IoT。
- 測試隨需及長時間的函數。

單元 4 和 5

[模組 4](#) 顯示用戶端裝置如何連線至核心並互相通訊。

[模組 5](#) 顯示用戶端裝置如何使用陰影來控制狀態。

- 註冊和佈建 AWS IoT 裝置（由命令列終端機表示）。
- 安裝 AWS IoT Device SDK 適用於 Python 的。這可供用戶端裝置用來探索 Greengrass 核心。
- 將用戶端裝置新增至 Greengrass 群組。
- 建立允許 MQTT 通訊的訂閱。
- 部署包含用戶端裝置的 Greengrass 群組。
- 測試device-to-device通訊。
- 測試影子狀態更新。

單元 6

[模組 6](#) 顯示 Lambda 函數如何存取 AWS 雲端。

- 建立允許存取 Amazon DynamoDB 資源的 Greengrass 群組角色。

- 將 Lambda 函數新增至 Greengrass 群組。此函數使用適用於 Python 的 AWS SDK 與 DynamoDB 互動。
- 建立允許 MQTT 通訊的訂閱。
- 測試與 DynamoDB 的互動。

單元 7

[單元 7](#) 會示範如何設定模擬的硬體安全模組 (HSM) 以搭配 Greengrass 核心使用。

Important

提供這個進階單元的目的僅是為了實驗和初始測試。其不適合用於任何生產用途。

- 安裝及設定軟體類型的 HSM 和私有金鑰。
- 設定 Greengrass 核心使用硬體安全。
- 測試硬體安全組態。

要求

為了完成本教學，您需要以下項目：

- Mac 或 Windows PC 或 UNIX-like 系統。
- AWS 帳戶。如果您沒有帳戶，請參閱 [the section called “建立 AWS 帳戶”](#)。
- 使用支援的 an AWS [Region](#) AWS IoT Greengrass。如需支援的區域清單 AWS IoT Greengrass，請參閱《》中的[AWS 端點和配額](#)AWS 一般參考。

Note

請記下您的 [AWS 區域](#) 並確保在本教學課程中一致地使用它。如果您在教學 AWS 區域課程中切換 [AWS 區域](#)，則完成步驟時可能會遇到問題。

- Raspberry Pi 4 Model B 或 Raspberry Pi 3 Model B/B+，搭配 8 GB microSD 卡或 Amazon EC2 執行個體。由於 AWS IoT Greengrass 最好與實體硬體搭配使用，所以建議您使用 Raspberry Pi。

Note

執行以下命令來取得您的 Raspberry Pi 型號：

```
cat /proc/cpuinfo
```

靠近清單的底部，記下 Revision 屬性的值，並查閱 [Which Pi have I got?](#) 表格。例如，若 Revision 的值為 a02082，表格會顯示 Pi 為 3 Model B。

執行以下命令來判斷您的 Raspberry Pi 的架構：

```
uname -m
```

在本教學課程中，結果應大於或等於 armv7l。

- 對 Python 的基本認識。

雖然本教學課程旨在於 Raspberry Pi AWS IoT Greengrass 上執行，但 AWS IoT Greengrass 也支援其他平台。如需詳細資訊，請參閱[the section called “支援平台和需求”](#)。

建立 AWS 帳戶

如果您沒有 AWS 帳戶，請依照下列步驟建立和啟用 AWS 帳戶：

註冊 AWS 帳戶

如果您沒有 AWS 帳戶，請完成下列步驟來建立一個。

註冊 AWS 帳戶

1. 開啟 <https://portal.aws.amazon.com/billing/signup>。
2. 請遵循線上指示進行。

部分註冊程序需接收來電或簡訊，並在電話鍵盤輸入驗證碼。

當您註冊時 AWS 帳戶，AWS 帳戶根使用者會建立。根使用者有權存取該帳戶中的所有 AWS 服務和資源。作為安全最佳實務，請將管理存取權指派給使用者，並且僅使用根使用者來執行[需要根使用者存取權的任務](#)。

AWS 會在註冊程序完成後傳送確認電子郵件給您。您可以隨時登錄 <https://aws.amazon.com/> 並選擇我的帳戶，以檢視您目前的帳戶活動並管理帳戶。

建立具有管理存取權的使用者

註冊後 AWS 帳戶，請保護 AWS 帳戶根使用者、啟用 AWS IAM Identity Center 和建立管理使用者，以免將根使用者用於日常任務。

保護您的 AWS 帳戶根使用者

1. 選擇根使用者並輸入 AWS 帳戶 您的電子郵件地址，以帳戶擁有者 [AWS 管理主控台](#) 身分登入。在下一頁中，輸入您的密碼。

如需使用根使用者登入的說明，請參閱 AWS 登入 使用者指南中的 [以根使用者身分登入](#)。

2. 若要在您的根使用者帳戶上啟用多重要素驗證 (MFA)。

如需說明，請參閱《IAM 使用者指南》中的 [為您的 AWS 帳戶 根使用者（主控台）啟用虛擬 MFA 裝置](#)。

建立具有管理存取權的使用者

1. 啟用 IAM Identity Center。

如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的 [啟用 AWS IAM Identity Center](#)。

2. 在 IAM Identity Center 中，將管理存取權授予使用者。

如需使用 IAM Identity Center 目錄 做為身分來源的教學課程，請參閱 AWS IAM Identity Center 《使用者指南》中的 [使用預設值設定使用者存取 IAM Identity Center 目錄](#)。

以具有管理存取權的使用者身分登入

- 若要使用您的 IAM Identity Center 使用者簽署，請使用建立 IAM Identity Center 使用者時傳送至您電子郵件地址的簽署 URL。

如需使用 IAM Identity Center 使用者登入的說明，請參閱 AWS 登入 《使用者指南》中的 [登入 AWS 存取入口網站](#)。

指派存取權給其他使用者

1. 在 IAM Identity Center 中，建立一個許可集來遵循套用最低權限的最佳實務。

如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的 [建立許可集](#)。

2. 將使用者指派至群組，然後對該群組指派單一登入存取權。

如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的[新增群組](#)。

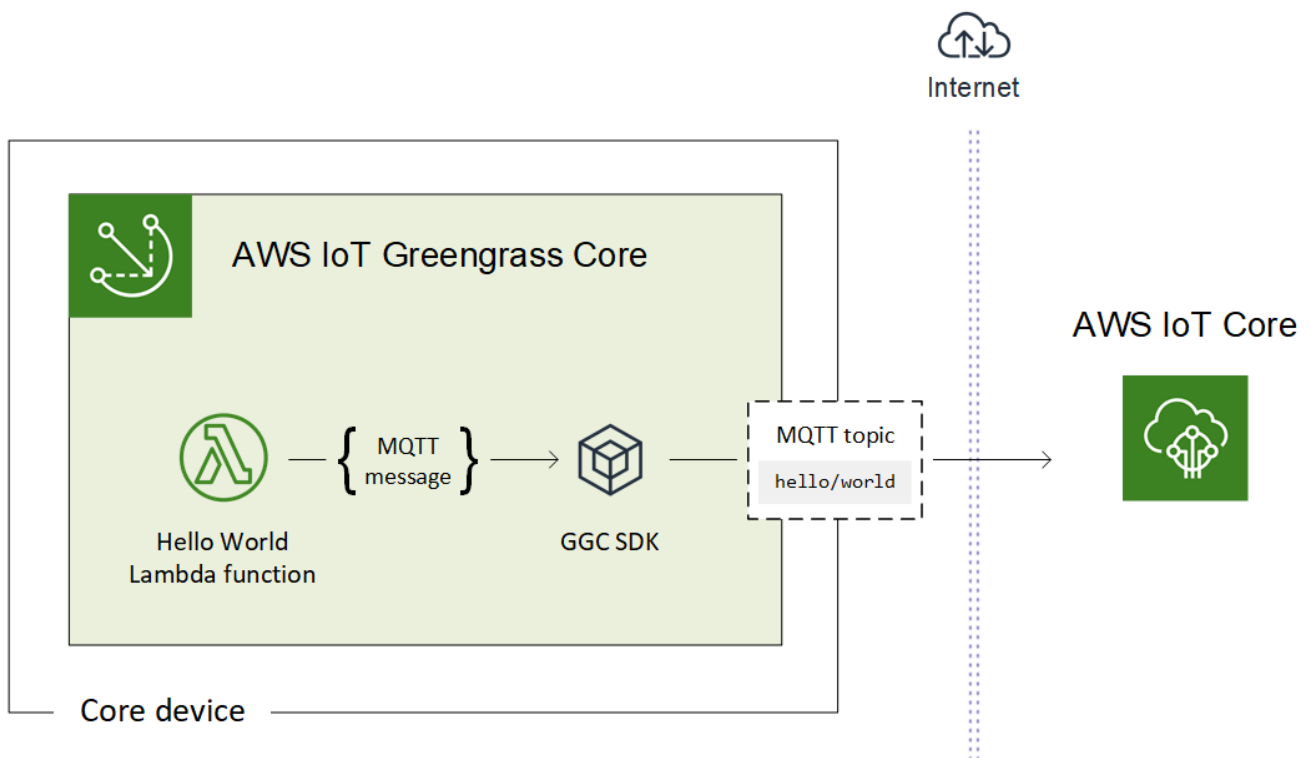
⚠ Important

在本教學課程中，我們假設您的 IAM 使用者帳戶具有管理員存取許可。

快速入門：Greengrass 裝置安裝

Greengrass 裝置設定是一種指令碼，可在幾分鐘內設定您的核心裝置，讓您可以開始使用 AWS IoT Greengrass。使用此指令碼來：

1. 設定您的裝置並安裝 AWS IoT Greengrass 核心軟體。
2. 設定雲端型資源。
3. 選擇性地部署具有 Hello World Lambda 函數的 Greengrass 群組，該函數 AWS IoT 會從 AWS IoT Greengrass 核心傳送 MQTT 訊息至。這會設定下圖所示的 Greengrass 環境。



要求

Greengrass 裝置安裝具有下列需求：

- 您的核心裝置必須使用[支援的平台](#)。裝置必須安裝適當的套件管理員：apt、yum 或 opkg。
- 執行指令碼的 Linux 使用者必須具備以 sudo 執行的許可。
- 您必須提供您的 AWS 帳戶 登入資料。如需詳細資訊，請參閱[the section called “提供 AWS 帳戶 登入資料”](#)。

Note

Greengrass 裝置設定會在裝置上安裝[最新版本](#)的 AWS IoT Greengrass Core 軟體。安裝 AWS IoT Greengrass 核心軟體，即表示您同意 [Greengrass 核心軟體授權合約](#)。

執行 Greengrass 裝置安裝

您只需要幾個步驟，便能執行 Greengrass 裝置安裝。在您提供 AWS 帳戶 登入資料後，指令碼會佈建您的 Greengrass 核心裝置，並在幾分鐘內部署 Greengrass 群組。請在目標裝置的終端機視窗中執行下列命令。

Note

下列步驟說明如何在互動模式下執行指令碼，這會提示您輸入或接受每個輸入值。如需如何以無提示的方式執行指令碼的資訊，請參閱[the section called “在無提示模式下執行 Greengrass 裝置設定”](#)。

1. [提供您的登入資料](#)。在此程序中，我們會假設您已提供暫時安全憑證做為環境變數。

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o50ytwEXAMPLE=
```

Note

如果您正在 Raspbian 或 OpenWrt 平台上執行 Greengrass 裝置安裝，請記下這些命令。將裝置重新開機後，您必須再次提供這些命令。

2. 下載及啟動指令碼。您可以使用 `wget` 或 `curl` 下載指令碼。

`wget`:

```
wget -q -O ./gg-device-setup-latest.sh https://d1onfpft10uf5o.cloudfront.net/greengrass-device-setup/downloads/gg-device-setup-latest.sh && chmod +x ./gg-device-setup-latest.sh && sudo -E ./gg-device-setup-latest.sh bootstrap-greengrass-interactive
```

`curl`:

```
curl https://d1onfpft10uf5o.cloudfront.net/greengrass-device-setup/downloads/gg-device-setup-latest.sh > gg-device-setup-latest.sh && chmod +x ./gg-device-setup-latest.sh && sudo -E ./gg-device-setup-latest.sh bootstrap-greengrass-interactive
```

3. 繼續完成輸入值的命令提示。您可以按 Enter 鍵來使用預設值，或是輸入自訂值並按 Enter。

指令碼會將狀態訊息寫入與以下相似的終端機。

```
##### Greengrass Device Setup v1.0.0 #####
[GreengrassDeviceSetup] The Greengrass Device Setup bootstrap log is available at: /tmp/greengrass-device-setup-bootstrap-1575933831.log
[GreengrassDeviceSetup] Using package management tool: yum...
[GreengrassDeviceSetup] Using runtime: python3.7...
[GreengrassDeviceSetup] Installing a dedicated pip for Greengrass Device Setup...
[GreengrassDeviceSetup] Validating and installing required dependencies...
[GreengrassDeviceSetup] The Greengrass Device Setup configuration is complete. Starting the Greengrass environment setup...
[GreengrassDeviceSetup] Forwarding command-line parameters: bootstrap-greengrass-interactive

[GreengrassDeviceSetup] Validating the device environment...
[GreengrassDeviceSetup] Validation of the device environment is complete.

[GreengrassDeviceSetup] Running the Greengrass environment setup...
[GreengrassDeviceSetup] The Greengrass environment setup is complete.

[GreengrassDeviceSetup] Configuring cloud-based Greengrass group management...
[GreengrassDeviceSetup] The Greengrass group configuration is complete.

[GreengrassDeviceSetup] Preparing the Greengrass core software...
[GreengrassDeviceSetup] The Greengrass core software is running.

[GreengrassDeviceSetup] Configuring the group deployment...
[GreengrassDeviceSetup] The group deployment is complete.
```

4. 如果您的核心裝置正在 Raspbian 或 OpenWrt 上執行，請在出現提示時重新開機裝置，提供您的登入資料，然後重新啟動指令碼。

- a. 當出現提示要求重新開機裝置時，執行以下其中一個命令。

針對 Raspbian 平台：

```
sudo reboot
```

針對 OpenWrt 平台：

```
reboot
```

- b. 在裝置重新開機後，請開啟終端機並提供您的登入資料做為環境變數。

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE  
export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY  
export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o50ytwEXAMPLE=
```

- c. 重新啟動指令碼。

```
sudo -E ./gg-device-setup-latest.sh bootstrap-greengrass-interactive
```

- d. 當出現提示，詢問您是要使用先前工作階段的輸入值，還是啟動新的安裝時，請輸入 yes 來重新使用您的輸入值。

 Note

在需要重新開機的平台，您先前工作階段的輸入值 (登入資料除外) 會暫時存放在 GreengrassDeviceSetup.config.info 檔案中。

安裝完成後，終端機會顯示予以下相似的成功狀態訊息。


```

=====
Your device is running the Greengrass core software.
Your Greengrass group and Hello World Lambda function were deployed to the core device.

Setup information:

Device info: Linux-4.14.152-127.182.amzn2.x86_64-x86_64-with-glibc2.2.5
Greengrass core software location: /
Installed Greengrass core software version: 1.10.0
Greengrass core: arn:aws:iot:us-west-2:012345678910:thing/GreengrassDeviceSetup_Core_d46a0ea4-18ae-4376-8f44-4a504cdea608
Greengrass core IoT certificate: arn:aws:iot:us-west-2:012345678910:cert/23fbf0f4b6a5ea369f2b97f1a1b558180a240faa8e059ce19dc58f4a4c0d3b77
Greengrass core IoT certificate location: /greengrass/certs/23fbf0f4b6.cert.pem
Greengrass core IoT key location: /greengrass/certs/23fbf0f4b6.private.key
Deployed Greengrass group name: GreengrassDeviceSetup_Group_ee70f777-9af0-43b6-8612-a18b418e8b4a
Deployed Greengrass group ID: 6f5c8410-f3a6-43a2-acf3-33158e10fb8e
Deployed Greengrass group version: arn:aws:greengrass:us-west-2:012345678910:/greengrass/groups/6f5c8410-f3a6-43a2-acf3-33158e10fb8e/vers
Greengrass service role: arn:aws:iam::012345678910:role/GreengrassServiceRole_mui1v
GreengrassDeviceSetup log location: GreengrassDeviceSetup-20191209-232356.log
Deployed hello-world Lambda function: arn:aws:lambda:us-west-2:012345678910:function:Greengrass_HelloWorld_uNTf2:1
Hello-world subscriber topic: hello/world

You can now use the AWS IoT Console to subscribe
to the 'hello/world' topic to receive messages published from your
Greengrass core.
=====

```

5. 檢閱指令碼使用您提供的輸入值設定的新 Greengrass 群組。

- a. 在您的 [AWS 管理主控台](#) 電腦上登入，並開啟 AWS IoT 主控台。

Note

請確定在主控台 AWS 區域 中選取的 與您用來設定 Greengrass 環境的 相同。根據預設，區域為美國西部（奧勒岡）。

- b. 在導覽窗格中，展開 Greengrass 裝置，然後選擇群組 (V1) 以尋找新建立的群組。

6. 如果您包含 Hello World Lambda 函數，Greengrass 裝置設定會將 Greengrass 群組部署到您的核心裝置。若要測試 Lambda 函數，或取得如何從群組中移除 Lambda 函數的相關資訊，請繼續前往入門教學 [the section called “驗證 Lambda 函數是否在核心裝置上執行”](#) 課程的單元 3-1。

Note

請確定在主控台 AWS 區域 中選取的 與您用來設定 Greengrass 環境的 相同。根據預設，區域為美國西部（奧勒岡）。

如果您沒有包含 Hello World Lambda 函數，您可以 [建立自己的 Lambda 函數](#) 或嘗試其他 Greengrass 功能。例如，您可以將 [Docker 應用程式部署](#) 連接器新增至您的群組，並用該連接器來將 Docker 容器部署到您的核心裝置。

疑難排解 問題

您可以使用下列資訊，對 AWS IoT Greengrass 裝置設定的問題進行疑難排解。

錯誤：找不到 Python (python3.7)。正在嘗試安裝...

解決方案：使用 Amazon EC2 執行個體時，您可能會看到此錯誤。/usr/bin/python3.7 資料夾中未安裝 Python 時，會發生此錯誤。若要解決此錯誤，請在安裝 Python 之後，在正確的目錄中移動 Python：

```
sudo ln -s /usr/local/bin/python3.7 /usr/bin/python3.7
```

其他疑難排解

若要針對 AWS IoT Greengrass 裝置設定的其他問題進行疑難排解，您可以在日誌檔案中尋找偵錯資訊：

- 針對 Greengrass 裝置安裝組態的問題，請檢查 /tmp/greengrass-device-setup-bootstrap-*epoch-timestamp*.log 檔案。
- 針對 Greengrass 群組或核心環境安裝的問題，請檢查與 gg-device-setup-latest.sh 相同目錄，或是在您指定位置的 GreengrassDeviceSetup-*date-time*.log 檔案。

如需更多故障診斷說明，請參閱[疑難排解](#)或檢查 [AWS IoT Greengrass AWS re : Post 上的標籤](#)。

Greengrass 裝置安裝組態選項

您可以設定 Greengrass 裝置設定以存取您的 AWS 資源，並設定您的 Greengrass 環境。

提供 AWS 帳戶 登入資料

Greengrass 裝置設定使用您的 AWS 帳戶 登入資料來存取您的 AWS 資源。它支援 IAM 使用者的長期登入資料或來自 IAM 角色的臨時安全登入資料。

首先，請取得您的登入資料。

- 若要使用長期憑證，請提供 IAM 使用者的存取金鑰 ID 和私密存取金鑰。如需為長期憑證建立存取金鑰的相關資訊，請參閱 [《IAM 使用者指南》中的管理 IAM 使用者的存取金鑰](#)。

- 若要使用臨時安全登入資料（建議），請提供來自擔任 IAM 角色的存取金鑰 ID、私密存取金鑰和工作階段字符。如需從 AWS STS `assume-role` 命令擷取臨時安全登入資料的資訊，請參閱《IAM 使用者指南》中的[使用臨時安全登入資料搭配 AWS CLI](#)。

Note

基於本教學的目的，我們假設 IAM 使用者或 IAM 角色具有管理員存取許可。

接著，透過下列兩種方式的其中一種，向 Greengrass 裝置設定提供您的登入資料：

- 環境變數。在您啟動指令碼前設定 `AWS_ACCESS_KEY_ID`、`AWS_SECRET_ACCESS_KEY` 和 `AWS_SESSION_TOKEN` (如需要的話) 環境變數，如 [the section called “執行 Greengrass 裝置安裝”](#) 的步驟 1 所示。
- 輸入值。啟動指令碼後，直接在終端機中輸入您的存取金鑰 ID、私密存取金鑰和工作階段字符 (如需要) 值。

Greengrass 裝置安裝不會儲存或存放您的登入資料。

提供輸入值

在互動模式下，Greengrass 裝置設定會提示您提供輸入值。您可以按 Enter 鍵來使用預設值，或是輸入自訂值並按 Enter。在無提示模式下，您需要在啟動指令碼後提供輸入值。

輸入值

AWS 存取金鑰 ID

長期或暫時安全憑證的存取金鑰 ID。只有在您沒有提供登入資料做為環境變數時，才指定此選項做為輸入值。如需詳細資訊，請參閱[the section called “提供 AWS 帳戶 登入資料”](#)。

無提示模式的選項名稱：`--aws-access-key-id`

AWS 私密存取金鑰

長期或暫時安全憑證的私密存取金鑰。只有在您沒有提供登入資料做為環境變數時，才指定此選項做為輸入值。如需詳細資訊，請參閱[the section called “提供 AWS 帳戶 登入資料”](#)。

無提示模式的選項名稱：`--aws-secret-access-key`

AWS 工作階段字符

暫時安全憑證的工作階段字符。只有在您沒有提供登入資料做為環境變數時，才指定此選項做為輸入值。如需詳細資訊，請參閱[the section called “提供 AWS 帳戶 登入資料”](#)。

無提示模式的選項名稱：`--aws-session-token`

AWS 區域

您要建立 Greengrass 群組的 AWS 區域。如需支援的清單，請參閱 AWS 區域 [《AWS IoT Greengrass》](#) 中的 Amazon Web Services 一般參考。

預設值：`us-west-2`

無提示模式的選項名稱：`--region`

Group name (群組名稱)

Greengrass 群組名稱。

預設值：`GreengrassDeviceSetup_Group_`*guid*

無提示模式的選項名稱：`--group-name`

核心名稱

Greengrass 核心的名稱。核心是執行 AWS IoT Greengrass Core 軟體 AWS IoT 的裝置（物件）。核心會新增至 AWS IoT 登錄檔和 Greengrass 群組。如果您提供名稱，它在 AWS 帳戶和中必須是唯一的 AWS 區域。

預設值：`GreengrassDeviceSetup_Core_`*guid*

無提示模式的選項名稱：`--core-name`

AWS IoT Greengrass 核心軟體安裝路徑

您要安裝 AWS IoT Greengrass Core 軟體的裝置檔案系統中的位置。

預設值：`/`

無提示模式的選項名稱：`--ggc-root-path`

Hello World Lambda 函數

指出是否要在 Greengrass 群組中包含 Hello World Lambda 函數。函數每五秒會向 `hello/world` 主題發佈 MQTT 訊息。

指令碼會在 中建立並發佈此使用者定義的 Lambda 函數，AWS Lambda 並將其新增至您的 Greengrass 群組。指令碼也會在群組中建立訂閱，允許函數傳送 MQTT 訊息至 AWS IoT。

Note

這是 Python 3.7 Lambda 函數。如果並未在裝置上安裝 Python 3.7，且指令碼無法進行安裝，指令碼便會在終端機中印出錯誤訊息。若要在群組中包含 Lambda 函數，您必須手動安裝 Python 3.7 並重新啟動指令碼。若要在沒有 Lambda 函數的情況下建立 Greengrass 群組，請重新啟動指令碼，並在出現提示 `no` 時輸入 `y` 以包含函數。

預設值：`no`

無提示模式的選項名稱：`--hello-world-lambda` - 此選項不採用數值。若要建立函數，請將它加入您的命令中。

部署逾時

Greengrass 裝置安裝停止檢查 [Greengrass 群組部署](#) 狀態前的秒數。這只會在群組包含 Hello World Lambda 函數時使用。否則便不會部署群組。

部署時間取決於您的網路速度。針對緩慢的網路速度，您可以增加這個值。

預設值：`180`

無提示模式的選項名稱：`--deployment-timeout`

日誌路徑

日誌檔案的位置，其中包含 Greengrass 群組和核心安裝操作的相關資訊。請使用此日誌來針對 Greengrass 群組和核心安裝的部署及其他問題進行故障診斷。

預設值：`./`

無提示模式的選項名稱：`--log-path`

Verbosity (詳細資訊)

指出指令碼執行時，是否在終端機中列印詳細日誌資訊。您可以使用此資訊來對裝置設定進行故障診斷。

預設值：no

無提示模式的選項名稱：--verbose - 此選項不採用數值。若要列印詳細日誌資訊，請將它加入您的命令中。

在無提示模式下執行 Greengrass 裝置設定

您可以在無提示模式下執行 Greengrass 裝置設定，這樣指令碼就不會提示您輸入任何值。若要在無提示模式下執行，請在啟動指令碼後指定 bootstrap-greengrass 模式和[輸入值](#)。若要使用其預設值，則可以忽略輸入值。

此程序取決於您在啟動指令碼之前提供 AWS 帳戶 登入資料做為環境變數，還是在啟動指令碼之後提供做為輸入值。

提供登入資料做為環境變數

1. [提供登入資料](#)做為環境變數 下列範例會匯出臨時登入資料，其中包括工作階段字符。

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxrFiCYEXAMPLEKEY
export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o50ytwEXAMPLE=
```

Note

如果您正在 Raspbian 或 OpenWrt 平台上執行 Greengrass 裝置安裝，請記下這些命令。將裝置重新開機後，您必須再次提供這些命令。

2. 下載及啟動指令碼。視需要提供輸入值。例如：

- 若要使用所有預設值：

```
wget -q -O ./gg-device-setup-latest.sh https://d1onfpft10uf5o.cloudfront.net/greengrass-device-setup/downloads/gg-device-setup-latest.sh && chmod +x ./
```

```
gg-device-setup-latest.sh && sudo -E ./gg-device-setup-latest.sh bootstrap-  
greengrass
```

- 若要指定自訂值：

```
wget -q -O ./gg-device-setup-latest.sh https://d1onfpft10uf5o.cloudfront.net/  
greengrass-device-setup/downloads/gg-device-setup-latest.sh && chmod +x ./  
gg-device-setup-latest.sh && sudo -E ./gg-device-setup-latest.sh bootstrap-  
greengrass  
--region us-east-1  
--group-name Custom_Group_Name  
--core-name Custom_Core_Name  
--ggc-root-path /custom/ggc/root/path  
--deployment-timeout 300  
--log-path /customized/log/path  
--hello-world-lambda  
--verbose
```

Note

若要使用 `curl` 下載指令碼，請將命令中的 `wget -q -O` 換為 `curl`。

- 如果您的核心裝置正在 Raspbian 或 OpenWrt 上執行，請在出現提示時重新開機裝置，提供您的登入資料，然後重新啟動指令碼。

- 當出現提示要求重新開機裝置時，執行以下其中一個命令。

針對 Raspbian 平台：

```
sudo reboot
```

針對 OpenWrt 平台：

```
reboot
```

- 在裝置重新開機後，請開啟終端機並提供您的登入資料做為環境變數。

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE  
export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
```



```
export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o50ytwEXAMPLE=
```

- c. 重新啟動指令碼。

```
sudo -E ./gg-device-setup-latest.sh bootstrap-greengrass
```

- d. 當出現提示，詢問您是要使用先前工作階段的輸入值，還是啟動新的安裝時，請輸入 yes 來重新使用您的輸入值。

Note

在需要重新開機的平台，您先前工作階段的輸入值 (登入資料除外) 會暫時存放在 GreengrassDeviceSetup.config.info 檔案中。

安裝完成後，終端機會顯示予以下相似的成功狀態訊息。

```
=====
Your device is running the Greengrass core software.
Your Greengrass group and Hello World Lambda function were deployed to the core device.

Setup information:

Device info: Linux-4.14.152-127.182.amzn2.x86_64-x86_64-with-glibc2.2.5
Greengrass core software location: /
Installed Greengrass core software version: 1.10.0
Greengrass core: arn:aws:iot:us-west-2:012345678910:thing/GreengrassDeviceSetup_Core_d46a0ea4-18ae-4376-8f44-4a504cdea608
Greengrass core IoT certificate: arn:aws:iot:us-west-2:012345678910:cert/23fbf0f4b6a5ea369f2b97f1a1b558180a240faa8e059ce19dc58f4a4c0d3b77
Greengrass core IoT certificate location: /greengrass/certs/23fbf0f4b6.cert.pem
Greengrass core IoT key location: /greengrass/certs/23fbf0f4b6.private.key
Deployed Greengrass group name: GreengrassDeviceSetup_Group_ee70f777-9af0-43b6-8612-a18b418e8b4a
Deployed Greengrass group ID: 6f5c8410-f3a6-43a2-acf3-33158e10fb8e
Deployed Greengrass group version: arn:aws:greengrass:us-west-2:012345678910:/greengrass/groups/6f5c8410-f3a6-43a2-acf3-33158e10fb8e/vers
Greengrass service role: arn:aws:iam::012345678910:role/GreengrassServiceRole_mu1lv
GreengrassDeviceSetup log location: GreengrassDeviceSetup-20191209-232356.log
Deployed hello-world Lambda function: arn:aws:lambda:us-west-2:012345678910:function:Greengrass_HelloWorld_uNTf2:1
Hello-world subscriber topic: hello/world

You can now use the AWS IoT Console to subscribe
to the 'hello/world' topic to receive messages published from your
Greengrass core.
=====
```

4. 如果您包含 Hello World Lambda 函數，Greengrass 裝置設定會將 Greengrass 群組部署到您的核心裝置。若要測試 Lambda 函數，或取得如何從群組中移除 Lambda 函數的相關資訊，請繼續前往入門教學[the section called “驗證 Lambda 函數是否在核心裝置上執行”](#)課程的單元 3-1。

Note

請確定在主控台 AWS 區域 中選取的 與您用來設定 Greengrass 環境的 相同。根據預設，區域為美國西部（奧勒岡）。

如果您沒有包含 Hello World Lambda 函數，您可以[建立自己的 Lambda 函數](#)或嘗試其他 Greengrass 功能。例如，您可以將 [Docker 應用程式部署](#) 連接器新增至您的群組，並用該連接器來將 Docker 容器部署到您的核心裝置。

提供登入資料做為輸入值

1. 下載及啟動指令碼。[提供您的登入資料](#)和您要指定的任何其他輸入值。下列範例說明如何提供臨時登入資料，其中包括工作階段字符。

- 若要使用所有預設值：

```
wget -q -O ./gg-device-setup-latest.sh https://d1onfpft10uf5o.cloudfront.net/greengrass-device-setup/downloads/gg-device-setup-latest.sh && chmod +x ./gg-device-setup-latest.sh && sudo -E ./gg-device-setup-latest.sh bootstrap-greengrass
--aws-access-key-id AKIAIOSFODNN7EXAMPLE
--aws-secret-access-key wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
--aws-session-token AQoDYXdzEJr1K...o50ytwEXAMPLE=
```

- 若要指定自訂值：

```
wget -q -O ./gg-device-setup-latest.sh https://d1onfpft10uf5o.cloudfront.net/greengrass-device-setup/downloads/gg-device-setup-latest.sh && chmod +x ./gg-device-setup-latest.sh && sudo -E ./gg-device-setup-latest.sh bootstrap-greengrass
--aws-access-key-id AKIAIOSFODNN7EXAMPLE
--aws-secret-access-key wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
--aws-session-token AQoDYXdzEJr1K...o50ytwEXAMPLE=
--region us-east-1
--group-name Custom_Group_Name
--core-name Custom_Core_Name
--ggc-root-path /custom/ggc/root/path
--deployment-timeout 300
--log-path /customized/log/path
--hello-world-lambda
--verbose
```

Note

如果您正在 Raspbian 或 OpenWrt 平台上執行 Greengrass 裝置安裝，請記下您的登入資料。將裝置重新開機後，您必須再次提供這些命令。

若要使用 curl 下載指令碼，請將命令中的 wget -q -O 換為 curl。

2. 如果您的核心裝置正在 Raspbian 或 OpenWrt 上執行，請在出現提示時重新開機裝置，提供您的登入資料，然後重新啟動指令碼。

- a. 當出現提示要求重新開機裝置時，執行以下其中一個命令。

針對 Raspbian 平台：

```
sudo reboot
```

針對 OpenWrt 平台：

```
reboot
```

- b. 重新啟動指令碼。您必須在命令中加入您的登入資料，而不是其他輸入值。例如：

```
sudo -E ./gg-device-setup-latest.sh bootstrap-greengrass
--aws-access-key-id AKIAIOSFODNN7EXAMPLE
--aws-secret-access-key wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
--aws-session-token AQoDYXdzEJr1K...o50ytwEXAMPLE=
```

- c. 當出現提示，詢問您是要使用先前工作階段的輸入值，還是啟動新的安裝時，請輸入 yes 來重新使用您的輸入值。

Note

在需要重新開機的平台，您先前工作階段的輸入值 (登入資料除外) 會暫時存放在 GreengrassDeviceSetup.config.info 檔案中。

安裝完成後，終端機會顯示予以下相似的成功狀態訊息。

```

=====
Your device is running the Greengrass core software.
Your Greengrass group and Hello World Lambda function were deployed to the core device.

Setup information:

Device info: Linux-4.14.152-127.182.amzn2.x86_64-x86_64-with-glibc2.2.5
Greengrass core software location: /
Installed Greengrass core software version: 1.10.0
Greengrass core: arn:aws:iot:us-west-2:012345678910:thing/GreengrassDeviceSetup_Core_d46a0ea4-18ae-4376-8f44-4a504cdea608
Greengrass core IoT certificate: arn:aws:iot:us-west-2:012345678910:cert/23fbf0f4b6a5ea369f2b97f1a1b558180a240faa8e059ce19dc58f4a4c0d3b77
Greengrass core IoT certificate location: /greengrass/certs/23fbf0f4b6.cert.pem
Greengrass core IoT key location: /greengrass/certs/23fbf0f4b6.private.key
Deployed Greengrass group name: GreengrassDeviceSetup_Group_ee70f777-9af0-43b6-8612-a18b418e8b4a
Deployed Greengrass group ID: 6f5c8410-f3a6-43a2-acf3-33158e10fb8e
Deployed Greengrass group version: arn:aws:greengrass:us-west-2:012345678910:/greengrass/groups/6f5c8410-f3a6-43a2-acf3-33158e10fb8e/vers
Greengrass service role: arn:aws:iam::012345678910:role/GreengrassServiceRole_mui1v
GreengrassDeviceSetup log location: GreengrassDeviceSetup-20191209-232356.log
Deployed hello-world Lambda function: arn:aws:lambda:us-west-2:012345678910:function:Greengrass_HelloWorld_uNTf2:1
Hello-world subscriber topic: hello/world

You can now use the AWS IoT Console to subscribe
to the 'hello/world' topic to receive messages published from your
Greengrass core.
=====

```

- 如果您包含 Hello World Lambda 函數，Greengrass 裝置設定會將 Greengrass 群組部署到您的核心裝置。若要測試 Lambda 函數，或取得如何從群組中移除 Lambda 函數的相關資訊，請繼續前往入門教學[the section called “驗證 Lambda 函數是否在核心裝置上執行”](#)課程的單元 3-1。

Note

請確定在主控台 AWS 區域 中選取的 與您用來設定 Greengrass 環境的 相同。根據預設，區域為美國西部（奧勒岡）。

如果您沒有包含 Hello World Lambda 函數，您可以[建立自己的 Lambda 函數](#)或嘗試其他 Greengrass 功能。例如，您可以將 [Docker 應用程式部署](#) 連接器新增至您的群組，並用該連接器來將 Docker 容器部署到您的核心裝置。

單元 1：Greengrass 的環境設定

本單元說明如何取得out-of-the-box Raspberry Pi、Amazon EC2 執行個體或其他裝置，以供 AWS IoT Greengrass 做為您的 AWS IoT Greengrass 核心裝置使用。

i Tip

或者，若要使用指令碼為您設定核心裝置，請參閱[the section called “快速入門：Greengrass 裝置安裝”](#)。

本單元應可在 30 分鐘內完成。

開始之前，請閱讀本教學課程的[需求](#)。然後，遵循以下其中一個主題的安裝說明。請只選擇適用您核心裝置類型的主題。

主題

- [設定 Raspberry Pi](#)
- [設定 Amazon EC2 執行個體](#)
- [設定其他裝置](#)

i Note

若要了解如何在預先建置的 Docker 容器中使用 AWS IoT Greengrass 執行，請參閱 [the section called “AWS IoT Greengrass 在 Docker 容器中執行”](#)。

設定 Raspberry Pi

遵循本主題中的步驟，設定 Raspberry Pi 以用作 AWS IoT Greengrass 核心。

i Tip

AWS IoT Greengrass 也提供安裝 AWS IoT Greengrass 核心軟體的其他選項。例如，您可以使用 [Greengrass 裝置設定](#) 來設定您的環境，並安裝最新版本的 AWS IoT Greengrass Core 軟體。或者，在支援的 Debian 平台上，您可以使用 [APT 套件管理員](#) 來安裝或升級 AWS IoT Greengrass 核心軟體。如需詳細資訊，請參閱[the section called “安裝 AWS IoT Greengrass 核心軟體”](#)。

如果您是第一次設定 Raspberry Pi，請您必須遵循這些步驟。否則，您可以跳到[步驟 9](#)。但是，我們建議您在步驟 2 使用作業系統重新映像您的 Raspberry Pi。

1. 下載並安裝 SD 卡格式器，例如 [SD 記憶卡格式器](#)。請將 SD 卡片插入電腦。啟動程式，選擇您插入 SD 卡片的磁碟。您可以執行 SD 卡片的快速格式化。
2. 下載 [Raspbian Buster](#) 作業系統做為 zip 檔案。
3. 使用 SD 卡撰寫工具 (如 [Etcher](#))，請按照此工具指示快閃將下載的 zip 檔案傳到 SD 卡中。由於作業系統映像較大，此步驟需要時間才能完成。退出您的 SD 卡，將 microSD 卡插入 Raspberry Pi。
4. 第一次啟動時，建議您將 Raspberry Pi 連接到監視器 (透過 HDMI)、鍵盤和滑鼠。接下來，將 Pi 連接到微型 USB 電源，而 Raspbian 作業系統應該會啟動。
5. 建議您先設定 Pi 的鍵盤配置再繼續。若要執行此作業，請選擇右上角的 Raspberry 圖示，選擇 Preferences (喜好設定) 然後選擇 Mouse and Keyboard Settings (滑鼠與鍵盤設定)。接著，在 Keyboard (鍵盤) 標籤上，選擇 Keyboard Layout (鍵盤配置)，然後選擇適當的鍵盤變化。
6. 接著，[使用 Wi-Fi 網路將 Raspberry Pi 連接到網際網路](#)，或使用乙太網路纜線。

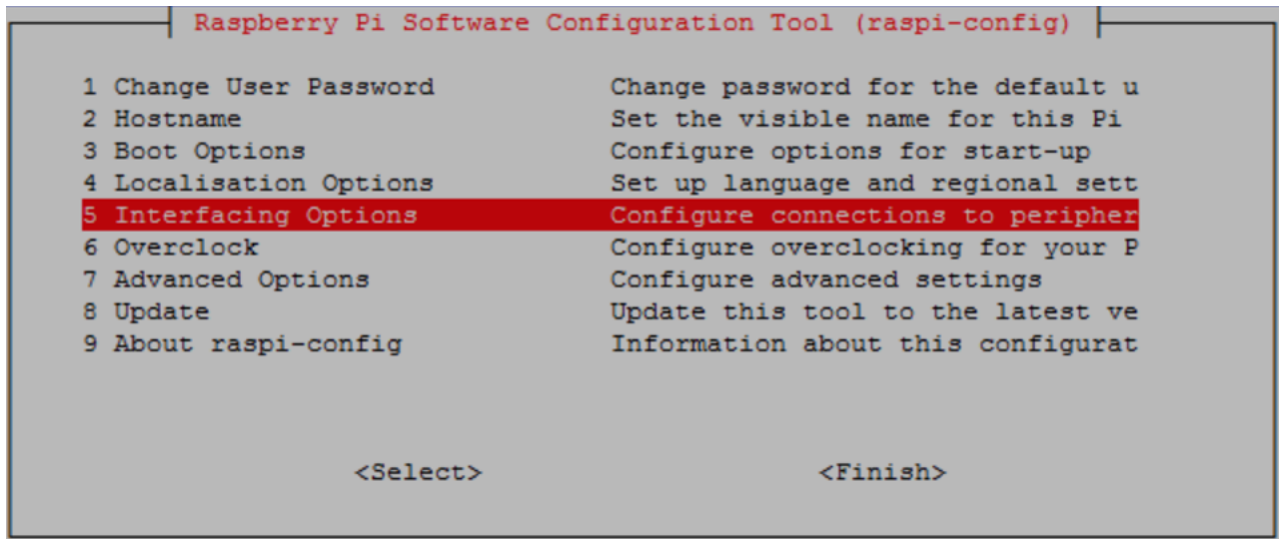
 Note

將您的 Raspberry Pi 連線到與您電腦連線相同的網路，並確認您的電腦和 Raspberry Pi 都具備網際網路存取再繼續。如果您位於工作環境內或防火牆內，您可能需要將 Pi 和您的電腦連線到訪客網路，才能讓兩者享有同樣的網際網路。但是這種方法可能會中斷您電腦與本機網路資源 (例如您的內部網路) 的連線。其中一個解決方案是將 Pi 連線到訪客 Wi-Fi 網路，並將您的電腦連線到訪客 Wi-Fi 網路和使用乙太網路纜線將其連線到您的本機網路。在此組態中，您的電腦應該能夠透過訪客 Wi-Fi 網路連線到 Raspberry Pi，並透過乙太網路纜線連線到本機網路資源。

7. 您必須在您的 Pi 上設定 [SSH](#) 遠端連接。在您的 Raspberry Pi 上開啟[終端機視窗](#)並執行以下命令：

```
sudo raspi-config
```

請查看下列事項：



向下捲動並選擇 Interfacing Options (連接選項)，然後選擇 P2 SSH。出現提示時，請選擇 Yes (是)。(使用 Tab 鍵加上 Enter)。SSH 現在應已啟用。選擇確定。使用 Tab 鍵並選擇 Finish (完成)，然後按 Enter。如果 Raspberry Pi 不會自動重新啟動，請執行下列命令：

```
sudo reboot
```

8. 在您的 Raspberry Pi 上，從終端機執行以下命令：

```
hostname -I
```

這會傳回您的 Raspberry Pi 的 IP 地址。

Note

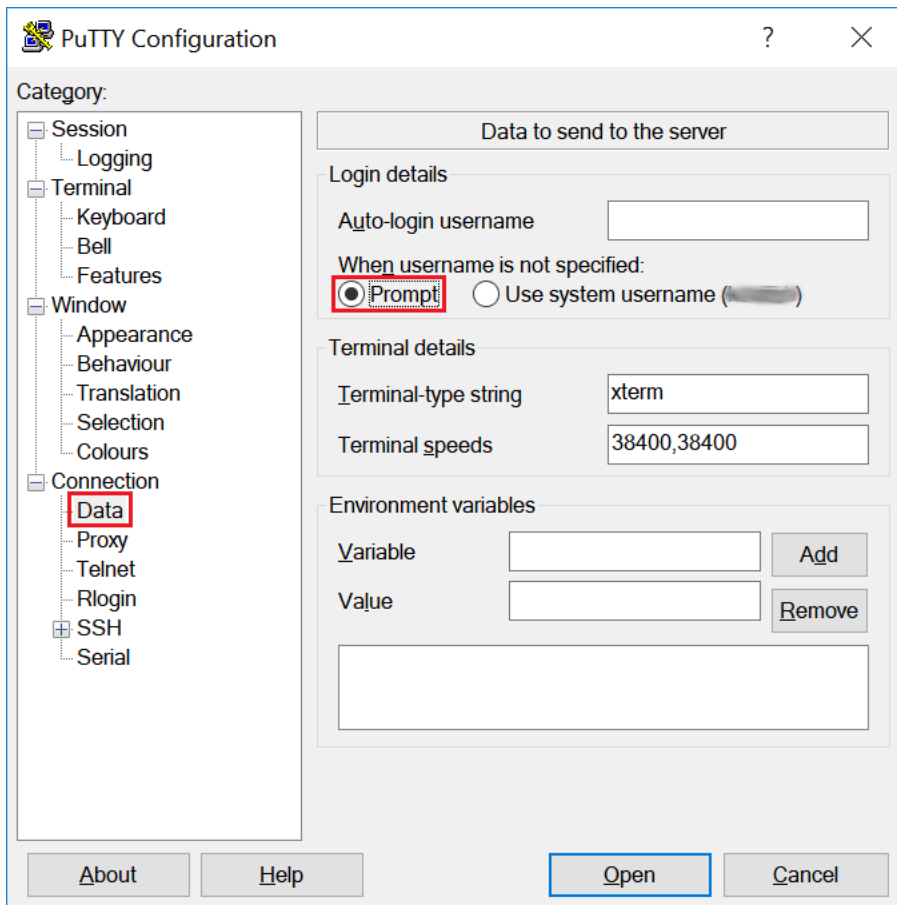
針對以下內容，若您收到 ECDSA 金鑰指紋訊息 (Are you sure you want to continue connecting (yes/no)?)，請輸入 yes。Raspberry Pi 的預設密碼為 **raspberry**。

如果您使用的是 macOS，請開啟終端機視窗並輸入以下資訊：

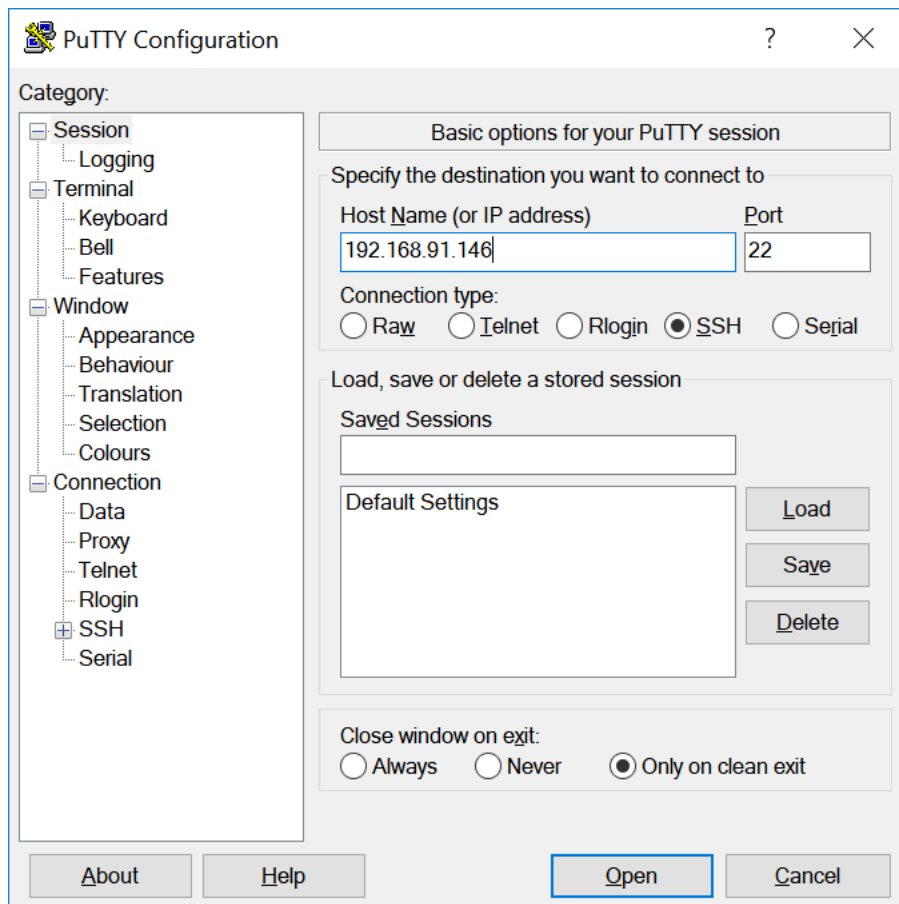
```
ssh pi@IP-address
```

IP-address 是您透過使用 hostname -I 命令取得的 Raspberry Pi IP 地址。

如果您使用 Windows，您需要安裝和設定 [PuTTY](#)。展開 Connection (連線)，選擇 Data (資料)，並確認已選取 Prompt (提示)：

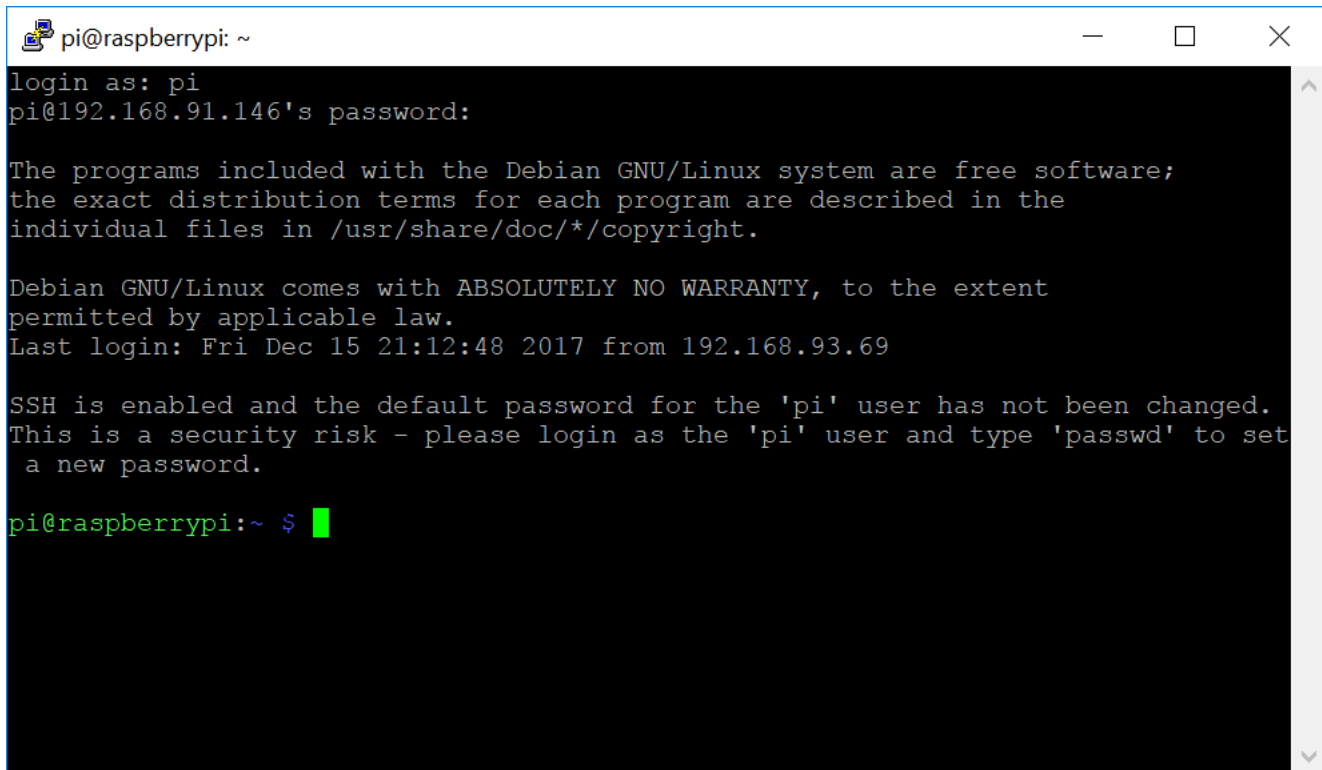


接著，選擇 Session (工作階段)，輸入 Raspberry Pi 的 IP 地址，然後選擇 Open (開啟)。



若顯示 PuTTY 安全提醒，請選擇 Yes (是)。

Raspberry Pi 的預設登入和預設密碼為 **pi** 與 **raspberry**。



```
pi@raspberrypi: ~  
login as: pi  
pi@192.168.91.146's password:  
  
The programs included with the Debian GNU/Linux system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/*/copyright.  
  
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent  
permitted by applicable law.  
Last login: Fri Dec 15 21:12:48 2017 from 192.168.93.69  
  
SSH is enabled and the default password for the 'pi' user has not been changed.  
This is a security risk - please login as the 'pi' user and type 'passwd' to set  
a new password.  
  
pi@raspberrypi:~ $
```

Note

如果您的電腦使用 VPN 連線到遠端網路，您可能會在使用 SSH 從電腦連線到 Raspberry Pi 時遭遇困難。

9. 您現在可以為設定 Raspberry Pi AWS IoT Greengrass。首先，從本機 Raspberry Pi 終端機視窗執行以下命令，或從 SSH 終端機視窗：

Tip

AWS IoT Greengrass 也提供安裝 AWS IoT Greengrass 核心軟體的其他選項。例如，您可以使用 [Greengrass 裝置設定](#) 來設定您的環境，並安裝最新版本的 AWS IoT Greengrass Core 軟體。或者，在支援的 Debian 平台上，您可以使用 [APT 套件管理員](#) 來安裝或升級 AWS IoT Greengrass 核心軟體。如需詳細資訊，請參閱 [the section called “安裝 AWS IoT Greengrass 核心軟體”](#)。

```
sudo adduser --system ggc_user  
sudo addgroup --system ggc_group
```

10. 為了改善 Pi 裝置的安全性，請在啟動時在作業系統上啟用 hardlink 和 softlink (symlink) 保護。

- a. 導覽至 98-rpi.conf 檔案。

```
cd /etc/sysctl.d
ls
```

 Note

如果您未看見 98-rpi.conf 檔案，請依照 README.sysctl 檔案中的指示進行。

- b. 使用文字編輯器 (例如 Leafpad、GNU nano 或 vi) 將以下兩行新增到檔案結尾。您可能需要使用 sudo 命令以根編輯 (例如，sudo nano 98-rpi.conf)。

```
fs.protected_hardlinks = 1
fs.protected_symlinks = 1
```

- c. 重新啟動 Pi。

```
sudo reboot
```

約一分鐘後，使用 SSH 連線至 Pi，然後執行以下命令來確認變更：

```
sudo sysctl -a 2> /dev/null | grep fs.protected
```

您應該會看到 fs.protected_hardlinks = 1 和 fs.protected_symlinks = 1。

11. 編輯您的命令列開機檔案來啟用及掛載記憶體 cgroups。這可讓 AWS IoT Greengrass 設定 Lambda 函數的記憶體限制。在 AWS IoT Greengrass 預設[容器化](#)模式中執行時，也需要 Cgroup。

- a. 導覽至您的 boot 目錄。

```
cd /boot/
```

- b. 使用文字編輯器開啟 cmdline.txt。將下列內容附加到現有的一行，不要另起新的一行。您可能需要使用 sudo 命令以根編輯 (例如，sudo nano cmdline.txt)。

```
cgroup_enable=memory cgroup_memory=1
```

c. 現在重新啟動 Pi。

```
sudo reboot
```

您的 Pi Raspberry 現在應已為 AWS IoT Greengrass 做好準備。

12. 選用。安裝[串流管理員](#)需要的 Java 8 執行時間。本教學課程不會使用串流管理員，但是會使用預設啟用串流管理員的 Default Group creation (預設群組建立) 工作流程。部署群組之前，先使用下列命令在核心裝置上安裝 Java 8 執行時間，或停用串流管理員。單元 3 中提供了停用串流管理員的指示。

```
sudo apt install openjdk-8-jdk
```

13. 為了確保您擁有所有必要的相依性，請從 GitHub 上的[AWS IoT Greengrass 範例](#)儲存庫下載並執行 Greengrass 相依性檢查程式。這些命令會解壓縮並在 Downloads 目錄中執行相依性檢查程式指令碼。

 Note

如果您執行的是 Raspbian 核心的第 5.4.51 版，相依性檢查程式可能會失敗。此版本未正確掛載記憶體 cgroup。這可能會導致在容器模式下執行的 Lambda 函數失敗。如需更新核心的詳細資訊，請參閱 Raspberry Pi 論壇中[核心升級後未載入的 Cgroups](#)。

```
cd /home/pi/Downloads
mkdir greengrass-dependency-checker-GGCv1.11.x
cd greengrass-dependency-checker-GGCv1.11.x
wget https://github.com/aws-samples/aws-greengrass-samples/raw/master/greengrass-dependency-checker-GGCv1.11.x.zip
unzip greengrass-dependency-checker-GGCv1.11.x.zip
cd greengrass-dependency-checker-GGCv1.11.x
sudo modprobe configs
sudo ./check_ggc_dependencies | more
```

在 more 出現的位置，按 Spacebar 鍵來顯示另一個文字畫面。

Important

本教學課程需要 Python 3.7 執行時間來執行本機 Lambda 函數。啟用串流管理員時，串流管理員需要 Java 8 執行時間。如果 `check_ggc_dependencies` 指令碼產生了與這些遺漏執行時間事前準備有關的警告，請確認在您繼續前已安裝了這些項目。您可以忽略與其他遺漏選用執行時間事前準備有關的警告。

如需 `modprobe` 命令的資訊，請在終端機中執行 `man modprobe`。

您的 Raspberry Pi 組態已完成。繼續進行[the section called “單元 2：安裝 AWS IoT Greengrass 核心軟體”](#)。

設定 Amazon EC2 執行個體

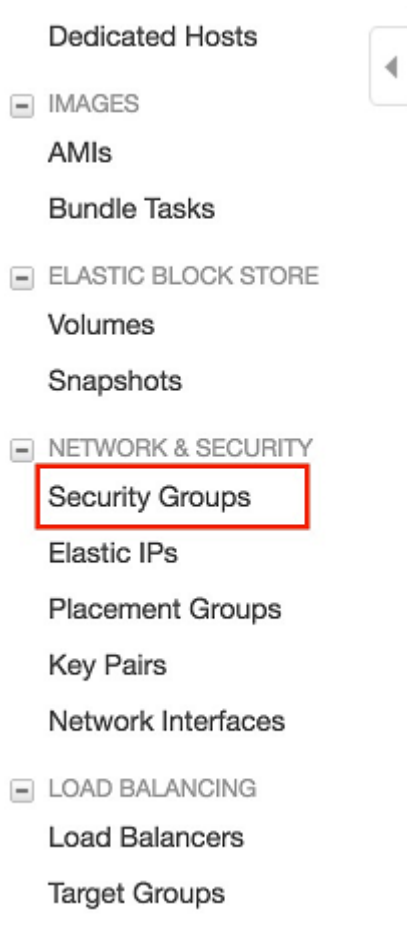
請遵循本主題中的步驟，設定 Amazon EC2 執行個體以用作您的 AWS IoT Greengrass 核心。

Tip

或者，若要使用指令碼來設定您的環境並為您安裝 AWS IoT Greengrass 核心軟體，請參閱[the section called “快速入門：Greengrass 裝置安裝”](#)。

雖然您可以使用 Amazon EC2 執行個體完成本教學課程，AWS IoT Greengrass 但最好搭配實體硬體使用。我們建議您[盡可能設定 Raspberry Pi](#)，而不是使用 Amazon EC2 執行個體。如果您正在使用 Raspberry Pi，您不需要遵循本主題中的步驟。

1. 登入 [AWS 管理主控台](#) 並使用 Amazon Linux AMI 啟動 Amazon EC2 執行個體。如需 Amazon EC2 執行個體的相關資訊，請參閱 [《Amazon EC2 入門指南》](#)。
2. Amazon EC2 執行個體執行後，請啟用連接埠 8883 以允許傳入的 MQTT 通訊，讓其他裝置可以與 AWS IoT Greengrass 核心連線。
 - a. 在 Amazon EC2 主控台的導覽窗格中，選擇安全群組。



- b. 選取您剛啟動之執行個體的安全群組，然後選擇傳入規則索引標籤。
- c. 選擇 Edit inbound Rules (編輯傳入規則)。

若要啟用連接埠 8883，您需新增自訂的 TCP 規則至安全群組。如需詳細資訊，請參閱《Amazon EC2 使用者指南》中的[將規則新增至安全群組](#)。

- d. 在編輯傳入規則頁面上，選擇新增規則，輸入下列設定，然後選擇儲存。
 - 針對 Type (類型)，選擇 Custom TCP Rule (自訂 TCP 規則)。
 - 針對連接埠範圍，輸入 **8883**。
 - 針對 Source (來源)，選擇 Anywhere (隨處)。
 - 對於 Description (說明)，輸入 **MQTT Communications**。

3. 連線到您的 Amazon EC2 執行個體。

- a. 在導覽窗格中，選擇 Instances (執行個體)，選擇您的執行個體，然後選擇 Connect (連線)。

- b. 依照 [Connect To Your Instance](#) (連線至您的執行個體) 頁面中的說明，[使用 SSH](#) 和您的私密金鑰檔案連線至您的執行個體。

您可以使用適用於 Windows 的 [PuTTY](#) 或適用於 macOS 的終端機。如需詳細資訊，請參閱《Amazon EC2 使用者指南》中的[連線至 Linux 執行個體](#)。

您現在已準備好設定 Amazon EC2 執行個體 AWS IoT Greengrass。

4. 連線至 Amazon EC2 執行個體後，請建立 `ggc_user` 和 `ggc_group` 帳戶：

```
sudo adduser --system ggc_user
sudo groupadd --system ggc_group
```

 Note

如果在您系統中沒有可用的 `adduser` 命令，請使用以下命令。

```
sudo useradd --system ggc_user
```

5. 为了提高安全性，請確定在啟動時已在 Amazon EC2 執行個體的作業系統上啟用硬連結和軟連結（符號連結）保護。

 Note

啟用 `hardlink` 和 `softlink` 保護的步驟會因作業系統而有所不同。請參閱文件來了解您的發行版本。

- a. 執行以下命令來檢查是否已啟用 `hardlink` 和 `softlink` 保護：

```
sudo sysctl -a | grep fs.protected
```

如果 `hardlinks` 和 `softlinks` 已設定為 1，您的保護就會正確啟用。繼續進行步驟 6。

 Note

Softlink 會以 `fs.protected_symlinks` 顯示。

- b. 如果 `hardlinks` 和 `softlinks` 未設定為 1，則啟用這些保護。導覽至您的系統組態檔案。

```
cd /etc/sysctl.d
ls
```

- c. 使用您喜愛的文字編輯器 (例如 Leafpad、GNU nano 或 vi)，將以下兩行新增至系統組態檔案結尾。在 Amazon Linux 1 上，這是 `00-defaults.conf` 檔案。在 Amazon Linux 2 上，這是 `99-amazon.conf` 檔案。您可能需要變更許可 (使用 `chmod` 命令)，才能寫入檔案，或使用 `sudo` 命令以根編輯 (例如，`sudo nano 00-defaults.conf`)。

```
fs.protected_hardlinks = 1
fs.protected_symlinks = 1
```

- d. 重新啟動 Amazon EC2 執行個體。

```
sudo reboot
```

在幾分鐘後，透過 SSH 連線到您的執行個體，然後執行以下命令來確認變更。

```
sudo sysctl -a | grep fs.protected
```

您應該會看到 `hardlinks` 和 `softlinks` 設為 1。

6. 擷取並執行下列指令碼來掛載 [Linux 控制群組](#) (群組)。這可讓 AWS IoT Greengrass 設定 Lambda 函數的記憶體限制。在 AWS IoT Greengrass 預設[容器化](#)模式中執行時，也需要 Cgroup。

```
curl https://raw.githubusercontent.com/tianon/cgroupfs-mount/951c38ee8d802330454bdede20d85ec1c0f8d312/cgroupfs-mount > cgroupfs-mount.sh
chmod +x cgroupfs-mount.sh
sudo bash ./cgroupfs-mount.sh
```

您的 Amazon EC2 執行個體現在應已準備就緒 AWS IoT Greengrass。

7. 選用。安裝[串流管理員](#)需要的 Java 8 執行時間。本教學課程不會使用串流管理員，但是會使用預設啟用串流管理員的 Default Group creation (預設群組建立) 工作流程。部署群組之前，先使用下列命令在核心裝置上安裝 Java 8 執行時間，或停用串流管理員。單元 3 中提供了停用串流管理員的指示。

- 針對 Debian 為基礎的發行版本：

```
sudo apt install openjdk-8-jdk
```

- 針對 Red Hat 為基礎的發行版本：

```
sudo yum install java-1.8.0-openjdk
```

8. 為了確保您擁有所有必要的相依性，請從 GitHub 上的[AWS IoT Greengrass 範例](#)儲存庫下載並執行 Greengrass 相依性檢查程式。這些命令會在您的 Amazon EC2 執行個體中下載、解壓縮和執行相依性檢查程式指令碼。

```
mkdir greengrass-dependency-checker-GGCv1.11.x
cd greengrass-dependency-checker-GGCv1.11.x
wget https://github.com/aws-samples/aws-greengrass-samples/raw/master/greengrass-dependency-checker-GGCv1.11.x.zip
unzip greengrass-dependency-checker-GGCv1.11.x.zip
cd greengrass-dependency-checker-GGCv1.11.x
sudo ./check_ggc_dependencies | more
```

Important

本教學課程需要 Python 3.7 執行時間來執行本機 Lambda 函數。啟用串流管理員時，串流管理員需要 Java 8 執行時間。如果 `check_ggc_dependencies` 指令碼產生了與這些遺漏執行時間事前準備有關的警告，請確認在您繼續前已安裝了這些項目。您可以忽略與其他遺漏選用執行時間事前準備有關的警告。

您的 Amazon EC2 執行個體組態已完成。繼續進行[the section called “單元 2：安裝 AWS IoT Greengrass 核心軟體”](#)。

設定其他裝置

遵循本主題中的步驟，設定裝置 (Raspberry Pi 除外) 以做為您的 AWS IoT Greengrass 核心。

Tip

或者，若要使用指令碼來設定您的環境並為您安裝 AWS IoT Greengrass 核心軟體，請參閱[the section called “快速入門：Greengrass 裝置安裝”](#)。

如果您是初次使用 AWS IoT Greengrass，我們建議您使用 Raspberry Pi 或 Amazon EC2 執行個體做為核心裝置，並遵循適合您裝置的[設定步驟](#)。

如果您計劃使用 Yocto 專案建置自訂 Linux 型系統，您可以使用 meta-aws 專案中的 AWS IoT Greengrass Bitbake Recipe。此配方也可協助您開發支援內嵌應用程式的 AWS 邊緣軟體的軟體平台。Bitbake 建置會在您的裝置上安裝、設定和自動執行 AWS IoT Greengrass 核心軟體。

Yocto 專案

開放原始碼協同合作專案，可協助您為嵌入式應用程式建置自訂 Linux 型系統，無論硬體架構為何。如需詳細資訊，請參閱 [Yocto 專案](#)。

meta-aws

提供 Yocto 配方的 AWS 受管專案。您可以使用配方，在以 [OpenEmbedded](#) 和 Yocto Project 建置的 Linux 型系統中開發 AWS 邊緣軟體。如需此社群支援功能的詳細資訊，請參閱 GitHub 上的 [meta-aws](#) 專案。

meta-aws-demos

包含專案示範的 AWS 受管 meta-aws 專案。如需整合程序的更多範例，請參閱 GitHub 上的 [meta-aws-demos](#) 專案。

若要使用不同的裝置或[支援的平台](#)，請遵循本主題中的步驟。

1. 如果您的核心裝置是 NVIDIA Jetson 裝置，您必須先使用 JetPack 4.3 安裝程式刷新韌體。如果您是設定不同的裝置，請跳到步驟 2。

Note

您所使用的 JetPack 安裝程式版本是根據您的目標 CUDA 工具組版本而定。下列指示使用 JetPack 4.3 和 CUDA Toolkit 10.0。如需如何為裝置使用適合版本的相關資訊，請參閱 NVIDIA 文件中的[如何安裝 Jetpack](#)。

- a. 在執行 Ubuntu 16.04 或更新版本的實體桌面上，使用 JetPack 4.3 安裝程式刷新韌體，如 NVIDIA 文件中的[下載和安裝 JetPack](#) (4.3) 所述。

依照安裝程式的指示在 Jetson 面板上安裝所有的套件和相依性，而 Jetson 面板必須使用 Micro-B 纜線連接到桌面。

- b. 在一般模式中重新啟動您的面板，並將顯示器連接到面板。

 Note

當您使用 SSH 連線到 Jetson 面板時，請使用預設使用者名稱 (**nvidia**) 及預設密碼 (**nvidia**)。

2. 執行以下命令建立使用者 `ggc_user` 和群組 `ggc_group`。您執行的命令可能不盡相同，取決於您核心裝置上安裝的分發。

- 如果您的核心裝置執行 OpenWrt，請執行下列命令：

```
opkg install shadow-useradd
opkg install shadow-groupadd
useradd --system ggc_user
groupadd --system ggc_group
```

- 否則，請執行下列命令：

```
sudo adduser --system ggc_user
sudo addgroup --system ggc_group
```

 Note

如果在您系統中沒有可用的 `addgroup` 命令，請使用以下命令。

```
sudo groupadd --system ggc_group
```

3. 選用。安裝[串流管理員](#)需要的 Java 8 執行時間。本教學課程不會使用串流管理員，但是會使用預設啟用串流管理員的 Default Group creation (預設群組建立) 工作流程。部署群組之前，先使用下列命令在核心裝置上安裝 Java 8 執行時間，或停用串流管理員。單元 3 中提供了停用串流管理員的指示。

- 針對 Debian 為基礎或 Ubuntu 為基礎的發行版本：

```
sudo apt install openjdk-8-jdk
```

- 針對 Red Hat 為基礎的發行版本：

```
sudo yum install java-1.8.0-openjdk
```

- 為了確保您擁有所有必要的相依性，請從 GitHub 上的[AWS IoT Greengrass 範例](#)儲存庫下載並執行 Greengrass 相依性檢查程式。這些命令會解壓縮並執行相依性檢查程式指令碼。

```
mkdir greengrass-dependency-checker-GGCv1.11.x
cd greengrass-dependency-checker-GGCv1.11.x
wget https://github.com/aws-samples/aws-greengrass-samples/raw/master/greengrass-dependency-checker-GGCv1.11.x.zip
unzip greengrass-dependency-checker-GGCv1.11.x.zip
cd greengrass-dependency-checker-GGCv1.11.x
sudo ./check_ggc_dependencies | more
```

Note

check_ggc_dependencies 指令碼在 AWS IoT Greengrass 支援的平台上執行，且需要特定的 Linux 系統命令。如需詳細資訊，請參閱相依性檢查程式的[讀我檔案](#)。

- 依照相依性檢查程式輸出的指示，在您的裝置上安裝所有必要的相依性。對於遺失核心層級的相依性，您可能需要重新編譯您的核心。針對掛載 Linux 控制群組 (cgroups)，您可以執行 [cgroupfs-mount](#) 指令碼。這可讓 AWS IoT Greengrass 設定 Lambda 函數的記憶體限制。在 AWS IoT Greengrass 預設[容器化](#)模式中執行時，也需要 Cgroup。

如果輸出中沒有出現錯誤，AWS IoT Greengrass 則應該能夠在裝置上成功執行。

Important

本教學課程需要 Python 3.7 執行時間來執行本機 Lambda 函數。啟用串流管理員時，串流管理員需要 Java 8 執行時間。如果 check_ggc_dependencies 指令碼產生了與這些遺漏執行時間事前準備有關的警告，請確認在您繼續前已安裝了這些項目。您可以忽略與其他遺漏選用執行時間事前準備有關的警告。

如需 AWS IoT Greengrass 需求和相依性的清單，請參閱 [the section called “支援平台和需求”](#)。

單元 2：安裝 AWS IoT Greengrass 核心軟體

本單元說明如何在您選擇的裝置上安裝 AWS IoT Greengrass 核心軟體。在本單元中，您先建立 Greengrass 群組和核心。然後，在您的核心裝置上下載、設定和啟動軟體。如需有關 AWS IoT Greengrass 核心軟體功能的詳細資訊，請參閱 [the section called “設定 AWS IoT Greengrass 核心”](#)。

開始之前，請確定已為所選的裝置完成[單元 1](#) 中的設定步驟。

i Tip

AWS IoT Greengrass 也提供安裝 AWS IoT Greengrass 核心軟體的其他選項。例如，您可以使用 [Greengrass 裝置設定](#) 來設定您的環境，並安裝最新版本的 AWS IoT Greengrass Core 軟體。或者，在支援的 Debian 平台上，您可以使用 [APT 套件管理員](#) 來安裝或升級 AWS IoT Greengrass 核心軟體。如需詳細資訊，請參閱[the section called “安裝 AWS IoT Greengrass 核心軟體”](#)。

本單元應可在 30 分鐘內完成。

主題

- [佈建要用作 Greengrass 核心的 AWS IoT 物件](#)
- [建立核心的 AWS IoT Greengrass 群組](#)
- [AWS IoT Greengrass 在核心裝置上安裝和執行](#)

佈建要用作 Greengrass 核心的 AWS IoT 物件

Greengrass 核心是執行 AWS IoT Greengrass Core 軟體以管理本機 IoT 程序的裝置。若要設定 Greengrass 核心，您可以建立 AWS IoT 實體，代表連線的裝置或邏輯實體 AWS IoT。當您將裝置註冊為 AWS IoT 物件時，該裝置可以使用允許其存取的數位憑證和金鑰 AWS IoT。您可以使用 [AWS IoT 政策](#) 來允許裝置與 AWS IoT 和 AWS IoT Greengrass 服務通訊。

在本節中，您將裝置註冊為 AWS IoT 物件，以將其用作 Greengrass 核心。

建立 AWS IoT 物件

1. 導覽至 [AWS IoT 主控台](#)。
2. 在管理下，展開所有裝置，然後選擇物件。
3. 在 Things (物件) 頁面上，選擇 Create things (建立物件)。
4. 在建立物件頁面上，選擇建立單一物件，然後選擇下一步。
5. 在指定物件屬性頁面上，執行下列動作：
 - a. 針對物件名稱，輸入代表您的裝置的名稱，例如 **MyGreengrassV1Core**。
 - b. 選擇下一步。

6. 在設定裝置憑證頁面上，選擇下一步。
7. 在將政策連接至憑證頁面上，執行下列其中一項操作：
 - 選取授予核心所需許可的現有政策，然後選擇建立物件。

此時會開啟模式，您可以在其中下載裝置用來連線至 的憑證和金鑰 AWS 雲端。

- 建立連接授予核心裝置許可的新政策。請執行下列操作：
 - a. 選擇建立政策。

Create policy (建立政策) 頁面隨即在新標籤中開啟。

- b. 在 Create policy (建立政策) 頁面上，執行下列動作：
 - i. 針對政策名稱，輸入描述政策的名稱，例如 **GreengrassV1CorePolicy**。
 - ii. 在政策陳述式索引標籤的政策文件下，選擇 JSON。
 - iii. 輸入下列政策文件。此政策允許核心與 AWS IoT Core 服務通訊、與裝置陰影互動，以及與 AWS IoT Greengrass 服務通訊。如需如何根據您的使用案例限制此政策存取的資訊，請參閱 [AWS IoT Greengrass 核心裝置的最低 AWS IoT 政策](#)。

JSON

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

```
        "iot:DeleteThingShadow"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "greengrass:*"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

iv. 選擇 Create (建立) 以建立政策。

c. 返回瀏覽器索引標籤，開啟附加政策至憑證頁面。請執行下列操作：

i. 在政策清單中，選取您建立的政策，例如 GreengrassV1CorePolicy。

如果您沒有看到政策，請選擇重新整理按鈕。

ii. 選擇 Create thing (建立物件)。

模式開啟，您可以在其中下載核心用來連線的憑證和金鑰 AWS IoT。

8. 返回瀏覽器索引標籤，開啟附加政策至憑證頁面。請執行下列操作：

a. 在政策清單中，選取您建立的政策，例如 GreengrassV1CorePolicy。

如果您沒有看到政策，請選擇重新整理按鈕。

b. 選擇 Create thing (建立物件)。

模式開啟，您可以在其中下載核心用來連線的憑證和金鑰 AWS IoT。

9. 在下載憑證和金鑰模式中，下載裝置的憑證。

Important

選擇完成之前，請下載安全資源。

請執行下列操作：

- a. 針對裝置憑證，選擇下載以下載裝置憑證。
- b. 針對公有金鑰檔案，選擇下載以下載憑證的公有金鑰。
- c. 針對私有金鑰檔案，選擇下載以下載憑證的私有金鑰檔案。
- d. 檢閱《AWS IoT 開發人員指南》中的[伺服器身分驗證](#)，然後選擇適當的根 CA 憑證。我們建議您使用 Amazon Trust Services (ATS) 端點和 ATS 根 CA 憑證。在根 CA 憑證下，選擇根 CA 憑證的下載。
- e. 選擇完成。

請記下裝置憑證和金鑰檔案名稱中常見的憑證 ID。供稍後使用。

建立核心的 AWS IoT Greengrass 群組

AWS IoT Greengrass 群組包含與其元件相關的設定和其他資訊，例如用戶端裝置、Lambda 函數和連接器。群組定義核心的組態，包括其元件如何互相互動。

在本節中，您會為核心建立 群組。

Tip

如需使用 AWS IoT Greengrass API 建立和部署群組的範例，請參閱 GitHub 上的 [gg_group_setup](#) 儲存庫。

建立核心的群組

1. 導覽至 [AWS IoT 主控台](#)。
2. 在管理下，展開 Greengrass 裝置，然後選擇群組 (V1)。

Note

如果您沒有看到 Greengrass 裝置選單，請變更為 AWS 區域支援的 AWS IoT Greengrass V1。如需支援區域的清單，請參閱《AWS 一般參考》中的 [AWS IoT](#)

[Greengrass V1 端點和配額](#)。您必須在 AWS IoT Greengrass V1 可使用的區域中[為核心建立 AWS IoT 物件](#)。

3. 在 Greengrass 群組頁面上，選擇建立群組。
4. 在建立 Greengrass 群組頁面上，執行下列動作：
 - a. 針對 Greengrass 群組名稱，輸入描述群組的名稱，例如 **MyGreengrassGroup**。
 - b. 針對 Greengrass 核心，選擇您先前建立的 AWS IoT 物件，例如 MyGreengrassV1Core。

主控台會自動為您選取物件的裝置憑證。

- c. 選擇建立群組。

AWS IoT Greengrass 在核心裝置上安裝和執行

Note

本教學課程提供在 Raspberry Pi 上執行 AWS IoT Greengrass 核心軟體的指示，但您可以使用任何支援的裝置。

在本節中，您會在核心裝置上設定、安裝和執行 AWS IoT Greengrass 核心軟體。

安裝和執行 AWS IoT Greengrass

1. 從本指南的核心[AWS IoT Greengrass 軟體](#)區段中，下載 AWS IoT Greengrass 核心軟體安裝套件。選擇最適合您核心裝置 CPU 架構、發行版和作業系統的套件。
 - 對於 Raspberry Pi，請下載 Armv7l 架構和 Linux 作業系統的 套件。
 - 對於 Amazon EC2 執行個體，請下載 x86_64 架構和 Linux 作業系統的 套件。
 - 對於 NVIDIA Jetson TX2，請下載 Armv8 (AArch64) 架構和 Linux 作業系統的 套件。
 - 對於 Intel Atom，請下載 x86_64 架構和 Linux 作業系統的 套件。
2. 在先前的步驟中，您下載了五個檔案到您的電腦：
 - `greengrass-0S-architecture-1.11.6.tar.gz` – 此壓縮檔案包含 AWS IoT Greengrass 核心裝置上執行的 核心軟體。
 - `certificateId-certificate.pem.crt` – 裝置憑證檔案。
 - `certificateId-public.pem.key` – 裝置憑證的公有金鑰檔案。

- *certificateId*-private.pem.key – 裝置憑證的私有金鑰檔案。
- AmazonRootCA1.pem – Amazon 根憑證授權機構 (CA) 檔案。

在此步驟中，您會將這些檔案從電腦傳輸到核心裝置。請執行下列操作：

- a. 如果您不知道 Greengrass 核心裝置的 IP 地址，請在核心裝置上開啟終端機並執行下列命令。

 Note

對於某些裝置，此命令可能不會傳回正確的 IP 地址。請參閱您的裝置文件，以擷取裝置 IP 地址。

```
hostname -I
```

- b. 將這些檔案從您的電腦傳輸到核心裝置。檔案傳輸步驟會根據電腦的作業系統而有所不同。選擇您的作業系統，以取得展示如何將檔案傳輸到 Raspberry Pi 裝置的步驟。

 Note

Raspberry Pi 的預設使用者名稱為 **pi**、預設密碼為 **raspberry**。
NVIDIA Jetson TX2 的預設使用者名稱為 **nvidia**、預設密碼為 **nvidia**。

Windows

若要將壓縮檔案從您的電腦傳輸到 Raspberry Pi 裝置，請使用工具 (像是 [WinSCP](#)) 或 [PuTTY](#) pscp 命令。若要使用 pscp 命令，請在電腦開啟 [命令提示] 視窗，並執行下列動作：

```
cd path-to-downloaded-files  
pscp -pw Pi-password greengrass-OS-architecture-1.11.6.tar.gz pi@IP-address:/home/pi  
pscp -pw Pi-password certificateId-certificate.pem.crt pi@IP-address:/home/pi  
pscp -pw Pi-password certificateId-public.pem.key pi@IP-address:/home/pi  
pscp -pw Pi-password certificateId-private.pem.key pi@IP-address:/home/pi
```

```
pscp -pw Pi-password AmazonRootCA1.pem pi@IP-address:/home/pi
```

 Note

此命令中的版本編號必須符合您的 AWS IoT Greengrass Core 軟體套件版本。

macOS

若要將壓縮檔案從您的 Mac 傳輸到 Raspberry Pi 核心裝置，請在您的電腦上開啟「終端機」視窗，並執行下列命令。*path-to-downloaded-files* 通常是 ~/Downloads。

 Note

您可能會收到提示，要求您輸入兩組密碼。若是跳出提示請您輸入兩組密碼，則第一組密碼為 Mac 的 sudo 命令密碼，第二組則為 Raspberry Pi 的密碼。

```
cd path-to-downloaded-files  
scp greengrass-OS-architecture-1.11.6.tar.gz pi@IP-address:/home/pi  
scp certificateId-certificate.pem.crt pi@IP-address:/home/pi  
scp certificateId-public.pem.key pi@IP-address:/home/pi  
scp certificateId-private.pem.key pi@IP-address:/home/pi  
scp AmazonRootCA1.pem pi@IP-address:/home/pi
```

 Note

此命令中的版本編號必須符合您的 AWS IoT Greengrass Core 軟體套件版本。

UNIX-like system

若要將壓縮檔案從您的電腦傳輸到 Raspberry Pi 核心裝置，請在您的電腦開啟 [終端機] 視窗，並執行下列命令：

```
cd path-to-downloaded-files  
scp greengrass-OS-architecture-1.11.6.tar.gz pi@IP-address:/home/pi  
scp certificateId-certificate.pem.crt pi@IP-address:/home/pi  
scp certificateId-public.pem.key pi@IP-address:/home/pi
```

```
scp certificateId-private.pem.key pi@IP-address:/home/pi  
scp AmazonRootCA1.pem pi@IP-address:/home/pi
```

 Note

此命令中的版本編號必須符合您的 AWS IoT Greengrass Core 軟體套件版本。

Raspberry Pi web browser

如果您使用 Raspberry Pi 的 Web 瀏覽器下載壓縮檔案，檔案應該位於 Pi 的~/Downloads資料夾中，例如 /home/pi/Downloads。否則，壓縮的檔案應該位於 Pi 的~資料夾中，例如 /home/pi。

3. 在 Greengrass 核心裝置上，開啟終端機，然後導覽至包含 AWS IoT Greengrass Core 軟體和憑證的資料夾。將 *path-to-transferred-files* 取代為您在核心裝置上傳檔案的路徑。例如，在 Raspberry Pi 上執行 `cd /home/pi`。

```
cd path-to-transferred-files
```

4. 解壓縮 AWS IoT Greengrass 核心裝置上的 核心軟體。執行下列命令來解壓縮您傳輸至核心裝置的軟體封存。此命令使用 `-C /` 引數，在核心裝置的根/greengrass資料夾中建立 資料夾。

```
sudo tar -xzf greengrass-OS-architecture-1.11.6.tar.gz -C /
```

 Note

此命令中的版本編號必須符合您的 AWS IoT Greengrass Core 軟體套件版本。

5. 將憑證和金鑰移至 AWS IoT Greengrass Core 軟體資料夾。執行下列命令來建立憑證的資料夾，並將憑證和金鑰移至其中。將 *path-to-transferred-files* 取代為您在核心裝置上傳檔案的路徑，並將 *certificateId* 取代為檔案名稱中的憑證 ID。例如，在 Raspberry Pi 上，將 *path-to-transferred-files* 取代為 `/home/pi`

```
sudo mv path-to-transferred-files/certificateId-certificate.pem.crt /greengrass/  
certs  
sudo mv path-to-transferred-files/certificateId-public.pem.key /greengrass/certs  
sudo mv path-to-transferred-files/certificateId-private.pem.key /greengrass/certs  
sudo mv path-to-transferred-files/AmazonRootCA1.pem /greengrass/certs
```

6. Core AWS IoT Greengrass 軟體使用指定軟體參數的組態檔案。此組態檔案會指定憑證檔案的檔案路徑，以及要使用的 AWS 雲端端點。在此步驟中，您會為核心建立 AWS IoT Greengrass 核心軟體組態檔案。請執行下列操作：
 - a. 取得核心 AWS IoT 物件的 Amazon Resource Name (ARN)。請執行下列操作：
 - i. 在[AWS IoT 主控台](#)的管理下，於 Greengrass 裝置下，選擇群組 (V1)。
 - ii. 在 Greengrass 群組頁面上，選擇您先前建立的群組。
 - iii. 在概觀下，選擇 Greengrass 核心。
 - iv. 在核心詳細資訊頁面上，複製 AWS IoT 物件 ARN，並儲存它以在 AWS IoT Greengrass 核心組態檔案中使用。
 - b. 取得 AWS 帳戶目前區域中的裝置 AWS IoT 資料端點。裝置使用此端點做為 AWS AWS IoT 實物連線至。請執行下列操作：
 - i. 在 [AWS IoT 主控台](#) 中，選擇設定。
 - ii. 在裝置資料端點下，複製端點並儲存以用於 AWS IoT Greengrass 核心組態檔案。
 - c. 建立 AWS IoT Greengrass 核心軟體組態檔案。例如，您可以執行下列命令，使用 GNU nano 建立檔案。

```
sudo nano /greengrass/config/config.json
```

使用下列 JSON 文件取代檔案的內容。

```
{
  "coreThing" : {
    "caPath": "AmazonRootCA1.pem",
    "certPath": "certificateId-certificate.pem.crt",
    "keyPath": "certificateId-private.pem.key",
    "thingArn": "arn:aws:iot:region:account-id:thing/MyGreengrassV1Core",
    "iotHost": "device-data-prefix-ats.iot.region.amazonaws.com",
    "ggHost": "greengrass-ats.iot.region.amazonaws.com",
    "keepAlive": 600
  },
  "runtime": {
    "cgroup": {
      "useSystemd": "yes"
    }
  },
  "managedRespawn": false,
```

```
"crypto": {
  "caPath": "file:///greengrass/certs/AmazonRootCA1.pem",
  "principals": {
    "SecretsManager": {
      "privateKeyPath": "file:///greengrass/certs/certificateId-private.pem.key"
    },
    "IoTCertificate": {
      "privateKeyPath": "file:///greengrass/certs/certificateId-private.pem.key",
      "certificatePath": "file:///greengrass/certs/certificateId-certificate.pem.crt"
    }
  }
}
```

然後，執行下列動作：

- 如果您下載的 Amazon 根 CA 憑證與 Amazon 根 CA 1 不同，請將 *AmazonRootCA1.pem* 的每個執行個體取代為 Amazon 根 CA 檔案的名稱。
- 將 *certificateId* 的每個執行個體取代為憑證和金鑰檔案名稱中的憑證 ID。
- 將 *arn#aws#iot#region#account-id#thing/MyGreengrassV1Core* 取代為您先前儲存的核心物件的 ARN。
- 以核心物件的名稱取代 *MyGreengrassV1core*。
- 以您稍早儲存 AWS IoT 的裝置資料端點取代 *device-data-prefix-ats.iot.region.amazonaws.com*。
- 將 *##* 取代為您的 AWS 區域。

如需您可以在此組態檔案中指定之組態選項的詳細資訊，請參閱 [AWS IoT Greengrass 核心組態檔案](#)。

7. 確認您的核心裝置已連線到網際網路。然後，AWS IoT Greengrass 在您的核心裝置上啟動。

```
cd /greengrass/ggc/core/
sudo ./greengrassd start
```

您應該會看到 `Greengrass successfully started` 訊息。請記下 PID。

Note

若要設定核心裝置在系統開機 AWS IoT Greengrass 時啟動，請參閱 [the section called “在系統開機啟動 Greengrass”](#)。

您可以執行下列命令，以確認 AWS IoT Greengrass Core 軟體 (Greengrass 協助程式) 正常運作。將 *PID-number* 取代為您的 PID：

```
ps aux | grep PID-number
```

您應該會看到一個 PID 項目，包含指向執行中 Greengrass 協助程式的路徑 (例如 /greengrass/ggc/packages/1.11.6/bin/daemon)。如果您遇到啟動問題 AWS IoT Greengrass，請參閱 [疑難排解](#)。

單元 3 (第 1 部分)：上的 Lambda 函數 AWS IoT Greengrass

本單元說明如何建立和部署從 AWS IoT Greengrass 核心裝置傳送 MQTT 訊息的 Lambda 函數。模組說明 Lambda 函數組態、用於允許 MQTT 訊息傳送的訂閱，以及核心裝置的部署。

[模組 3 \(第 2 部分\)](#) 涵蓋 AWS IoT Greengrass 核心上執行的隨需和長期 Lambda 函數之間的差異。

開始之前，請確定您已完成[單元 1](#) 和[單元 2](#)，並擁有執行中 AWS IoT Greengrass 的核心裝置。

Tip

或者，若要使用指令碼為您設定核心裝置，請參閱[the section called “快速入門：Greengrass 裝置安裝”](#)。指令碼也可以建立和部署此模組中使用的 Lambda 函數。

此單元需約 30 分鐘完成。

主題

- [建立和封裝 Lambda 函數](#)
- [設定的 Lambda 函數 AWS IoT Greengrass](#)
- [部署雲端組態到 Greengrass 核心裝置](#)
- [驗證 Lambda 函數是否在核心裝置上執行](#)

建立和封裝 Lambda 函數

本模組中的範例 Python Lambda 函數使用適用於 Python [AWS IoT Greengrass 的核心 SDK](#) 來發佈 MQTT 訊息。

在此步驟中，您：

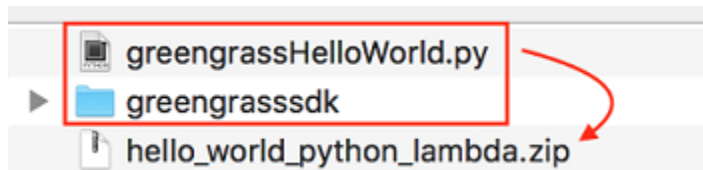
- 將適用於 Python AWS IoT Greengrass 的核心 SDK 下載到您的電腦（而非 AWS IoT Greengrass 核心裝置）。
- 建立包含函數程式碼和相依性的 Lambda 函數部署套件。
- 使用 Lambda 主控台建立 Lambda 函數並上傳部署套件。
- 發佈 Lambda 函數的版本，並建立指向該版本的別名。

您必須將 Python 3.7 安裝在核心裝置上，才能完成此單元。

1. 從[AWS IoT Greengrass 核心開發套件](#)下載頁面，將適用於 Python 的 AWS IoT Greengrass 核心開發套件下載至您的電腦。
2. 解壓縮下載的套件，以取得 Lambda 函數程式碼和 SDK。

此模組中的 Lambda 函數使用：

- examples\HelloWorld 中的 greengrassHelloWorld.py 檔案。這是您的 Lambda 函數程式碼。每隔五秒鐘，函數便會從可能發佈的訊息中二選一發佈給 hello/world 主題。
 - greengrasssdk 資料夾。此為軟體開發套件。
3. 將 greengrasssdk 資料夾複製到包含 greengrassHelloWorld.py 的 HelloWorld 資料夾中。
 4. 若要建立 Lambda 函數部署套件，請將 greengrassHelloWorld.py 和 greengrasssdk 資料夾儲存至名為的壓縮zip檔案hello_world_python_lambda.zip。py 檔案和 greengrasssdk 資料夾必須為在目錄的根內。



在 UNIX 式系統 (包括 Mac 終端機) 上，可使用以下命令來封裝檔案和資料夾：

```
zip -r hello_world_python_lambda.zip greengrasssdk greengrassHelloWorld.py
```

Note

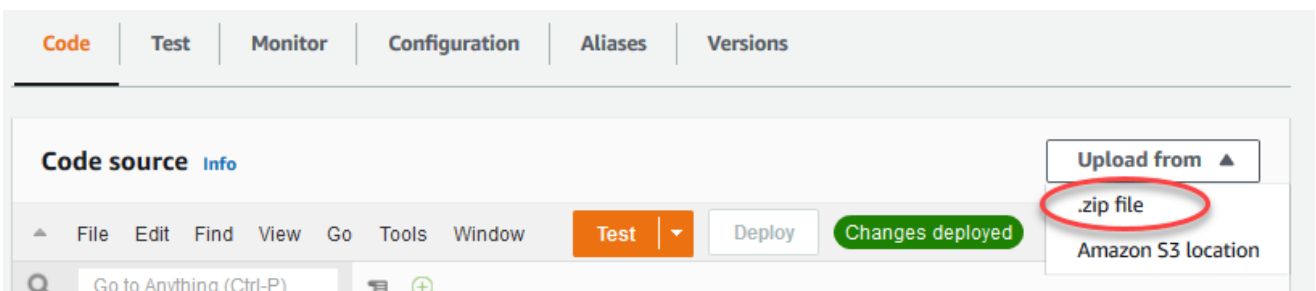
取決於您的發行版本，您可能需要先安裝 zip (例如透過執行 `sudo apt-get install zip`)。適用於您發行版本的安裝命令可能會有所不同。

現在您已準備好建立 Lambda 函數並上傳部署套件。

5. 開啟 Lambda 主控台，然後選擇建立函數。
6. 選擇從頭開始撰寫。
7. 將您的函數命名為 **Greengrass_HelloWorld**，並將其餘的欄位設定如下：
 - 針對執行期，選擇 Python 3.7。
 - 對於許可，請保留預設設定。這會建立授予基本 Lambda 許可的執行角色。不會使用此角色 AWS IoT Greengrass。

選擇 Create function (建立函數)。

8. 上傳您的 Lambda 函數部署套件：
 - a. 在程式碼索引標籤的程式碼來源下，選擇從中上傳。從下拉式清單中選擇 .zip 檔案。



- b. 選擇上傳，然後選擇您的 `hello_world_python_lambda.zip` 部署套件。然後選擇 Save (儲存)。
 - c. 在函數的程式碼索引標籤的執行時間設定下，選擇編輯，然後輸入下列值。
 - 針對執行期，選擇 Python 3.7。
 - 對於 Handler (處理常式)，輸入 **greengrassHelloWorld.function_handler**。

Runtime settings [Info](#)

Runtime

Python 3.7 ▼

Handler [Info](#)

greengrassHelloWorld.function_handler

- d. 選擇 Save (儲存)。

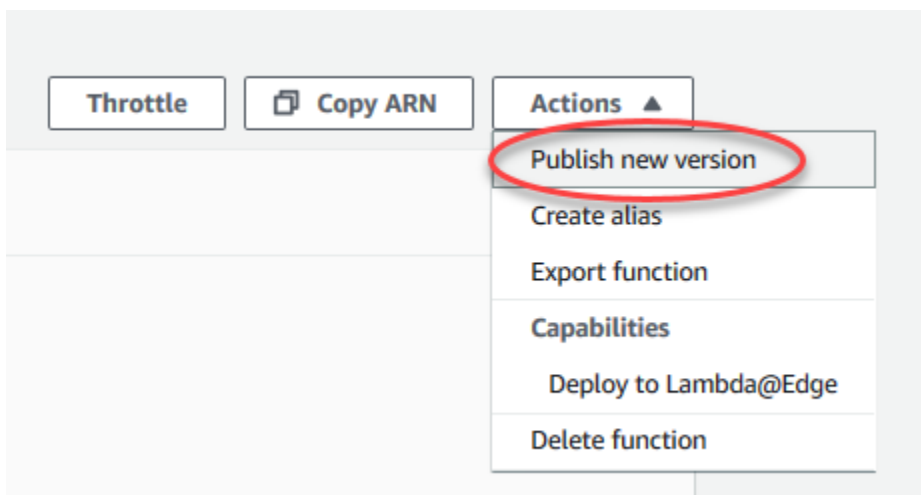
Note

AWS Lambda 主控台上的測試按鈕不適用於此函數。AWS IoT Greengrass 核心 SDK 不包含在 AWS Lambda 主控台中獨立執行 Greengrass Lambda 函數所需的模組。這些模組 (例如 greengrass_common) 會在部署到您的 Greengrass 核心之後提供給函數。

9.

發佈 Lambda 函數：

- a. 從頁面頂端的動作選單中，選擇發佈新版本。



- b. 針對 Version description (版本描述)，輸入 **First version**，然後選擇 Publish (發佈)。

Publish new version from \$LATEST

Publishing a new version will save a "snapshot" of the code and configuration of the \$LATEST version. You will be unable to edit the new version's code. Please click to confirm.

Version description

First version

Cancel

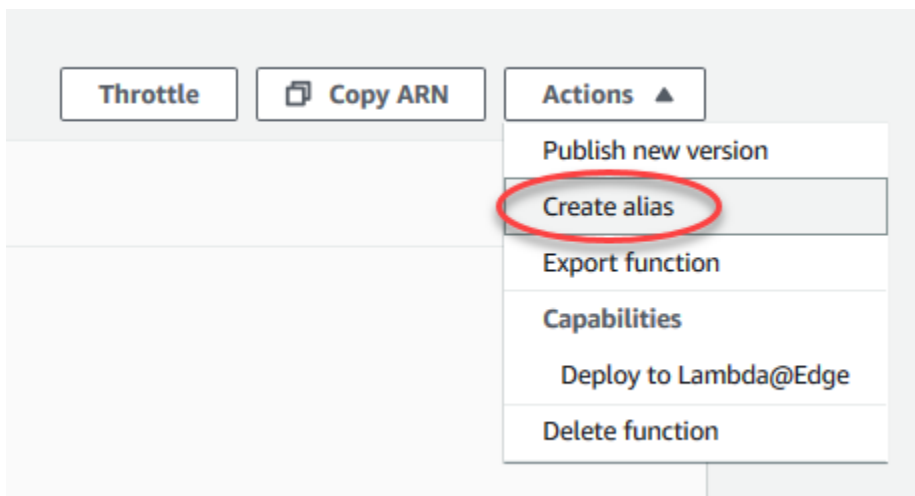
Publish

10. 建立 Lambda 函數 版本的別名：

Note

Greengrass 群組可以依別名（建議）或版本參考 Lambda 函數。使用別名可讓您更輕鬆地管理程式碼更新，因為您不必在更新函數程式碼時變更訂閱資料表或群組定義。反之，您只需將別名指向新的函數版本。

- a. 從頁面頂端的動作選單中，選擇建立別名。



- b. 命名別名 **GG_HelloWorld**，將版本設定為 **1**（對應至您剛發佈的版本），然後選擇儲存。

Note

AWS IoT Greengrass 不支援 \$LATEST 版本的 Lambda 別名。

Create alias

Alias configuration

An alias is a pointer to one or two versions. Choose each version that you want the alias to point to.

Name

Description - optional

Version

1 ▼

► Weighted alias

CancelSave

設定 的 Lambda 函數 AWS IoT Greengrass

您現在已準備好設定 Lambda 函數 AWS IoT Greengrass。

在此步驟中，您：

- 使用 AWS IoT 主控台將 Lambda 函數新增至 Greengrass 群組。
- 設定 Lambda 函數的群組特定設定。
- 將訂閱新增至允許 Lambda 函數發佈 MQTT 訊息的群組 AWS IoT。
- 設定群組的本機日誌設定。

1. 在 AWS IoT 主控台導覽窗格的管理下，展開 Greengrass 裝置，然後選擇群組 (V1)。
2. 在 Greengrass 群組下，選擇您在[單元 2](#)中建立的群組。

3. 在群組組態頁面上，選擇 Lambda 函數索引標籤，然後向下捲動至我的 Lambda 函數區段，然後選擇新增 Lambda 函數。
4. 選取您在上一個步驟中建立的 Lambda 函數名稱 (Greengrass_HelloWorld，而非別名名稱)。
5. 針對版本，選擇別名：GG_HelloWorld。
6. 在 Lambda 函數組態區段中，進行下列變更：
 - 將系統使用者和群組設定為使用群組預設值。
 - 將 Lambda 函數容器化設定為使用群組預設值。
 - 將 Timeout (逾時) 設為 25 秒。此 Lambda 函數會在每次調用前休眠 5 秒。
 - 針對固定，選擇 True。

 Note

長期（或固定）Lambda 函數會在啟動後自動 AWS IoT Greengrass 啟動，並在自己的容器中繼續執行。這與隨需 Lambda 函數相反，該函數會在叫用時啟動，並在沒有任務剩餘執行時停止。如需詳細資訊，請參閱[the section called “生命週期組態”](#)。

7. 選擇新增 Lambda 函數以儲存變更。如需 Lambda 函數屬性的詳細資訊，請參閱 [the section called “控制 Greengrass Lambda 函數執行”](#)。

接著，建立允許 Lambda 函數傳送 [MQTT](#) 訊息的訂閱 AWS IoT Core。

Greengrass Lambda 函數可以與下列項目交換 MQTT 訊息：

- Greengrass 群組中的[裝置](#)。
- 該群組中的[連接器](#)。
- 群組中的其他 Lambda 函數。
- AWS IoT Core。
- 本機陰影服務。如需詳細資訊，請參閱[the section called “第五單元：與裝置陰影互動”](#)。

群組使用訂閱控制這些實體互相通訊的方式。訂閱會提供可預測的互動以及多一層安全性。

訂閱包含來源、目標和主題。來源是訊息的起源。目標是訊息的目的地。主題可讓您篩選從來源傳送到目標的資料。來源或目標可以是 Greengrass 裝置、Lambda 函數、連接器、裝置影子或 AWS IoT Core。

 Note

訂閱是指訊息以特定方向流動：從來源至目標。若要允許雙向通訊，您必須設定兩個訂閱。

 Note

目前，訂閱主題篩選條件不允許在主題中使用多個+字元。主題篩選條件僅允許在主題結尾使用單一#字元。

Greengrass_HelloWorld Lambda 函數只會傳送訊息至 中的hello/world主題 AWS IoT Core，因此您只需要從 Lambda 函數建立一個訂閱 AWS IoT Core。您將在下一個步驟中建立此項目。

8. 在群組組態頁面上，選擇訂閱索引標籤，然後選擇新增訂閱。

如需示範如何使用 建立訂閱的範例 AWS CLI，請參閱《AWS CLI 命令參考》中的 [create-subscription-definition](#)。

9. 在來源類型中，選擇 Lambda 函數，然後在來源中選擇 Greengrass_HelloWorld。
10. 針對目標類型，選擇服務，並針對目標選取 IoT 雲端。
11. 針對主題篩選條件，輸入 **hello/world**，然後選擇建立訂閱。
12. 設定群組的記錄設定。在本教學課程中，您會設定 AWS IoT Greengrass 系統元件和使用使用者定義的 Lambda 函數，將日誌寫入核心裝置的檔案系統。
 - a. 在群組組態頁面上，選擇日誌索引標籤。
 - b. 在本機日誌組態區段中，選擇編輯。
 - c. 在編輯本機日誌組態對話方塊中，保留日誌層級和儲存體大小的預設值，然後選擇儲存。

您可以使用日誌來針對您在執行本教學時遇到的任何問題進行故障診斷。問題故障診斷時，您可以暫時將記錄層級變更為 Debug (偵錯)。如需詳細資訊，請參閱[the section called “存取檔案系統日誌”](#)。

13. 如果您的核心裝置上未安裝 Java 8 執行時間，您必須進行安裝或停用串流管理員。

 Note

本教學課程不會使用串流管理員，但是會使用預設啟用串流管理員的 Default Group creation (預設群組建立) 工作流程。如果啟用串流管理員，但未安裝 Java 8，則群組部署會失敗。如需詳細資訊，請參閱[串流管理員需求](#)。

若要停用串流管理員：

- a. 在群組設定頁面上，選擇 Lambda 函數索引標籤。
- b. 在 System Lambda 函數區段下，選取串流管理員，然後選擇編輯。
- c. 選擇 Disable (停用)，然後選擇 Save (儲存)。

部署雲端組態到 Greengrass 核心裝置

1. 確認您的 Greengrass 核心裝置已連線到網際網路。例如，嘗試成功導覽至網頁。
2. 確認 Greengrass 協助程式正在您的核心裝置上執行。在核心裝置終端機中，執行下列命令來檢查協助程式是否正在執行，並視需要啟動它。
 - a. 檢查精靈是否有在運作：

```
ps aux | grep -E 'greengrass.*daemon'
```

若輸出的 root 含有 /greengrass/ggc/packages/1.11.6/bin/daemon 項目，則精靈有在運作。

- b. 若要啟動協助程式：

```
cd /greengrass/ggc/core/  
sudo ./greengrassd start
```

現在您已準備好將 Lambda 函數和訂閱組態部署到您的 Greengrass 核心裝置。

3. 在 AWS IoT 主控台導覽窗格的管理下，展開 Greengrass 裝置，然後選擇群組 (V1)。
4. 在 Greengrass 群組下，選擇您在[單元 2](#)中建立的群組。
5. 在群組組態頁面上，選擇部署。
6. 在 Lambda 函數索引標籤的系統 Lambda 函數區段中，選擇 IP 偵測器。
7. 選擇編輯，然後選取自動偵測和覆寫 MQTT 代理程式端點。這可讓裝置自動取得核心的連接資訊，例如 IP 位址、DNS、連接埠編號。建議使用自動偵測，但是 AWS IoT Greengrass 也支援手動指定端點。只會在第一次部署群組時收到復原方法的提示。

初次部署需要花幾分鐘。部署完成時，您應該會在部署頁面上的狀態欄中看到成功完成：

 Note

部署狀態也會顯示在頁面標頭的群組名稱底下。

如需故障診斷協助，請參閱[疑難排解](#)。

驗證 Lambda 函數是否在核心裝置上執行

1. 在[AWS IoT 主控台](#)的導覽窗格中，於測試下，選擇 MQTT 測試用戶端。
2. 選擇訂閱主題索引標籤。
3. 在主題篩選條件**hello/world**中輸入，然後展開其他組態。
4. 輸入下列每個欄位中列出的資訊：
 - 針對 Quality of Service (服務品質)，選擇 0。
 - 針對 MQTT payload display (MQTT 承載顯示)，選擇 Display payloads as strings (將承載顯示為字串)。
5. 選擇 Subscribe (訂閱)。

假設 Lambda 函數正在您的裝置上執行，它會發佈類似下列hello/world主題的訊息：

Subscriptions

hello/world  

hello/world Pause Clear Export Edit

 hello/world

April 29, 2021, 17:35:40 (UTC-0400)

```
{
  "message": "Hello world! Sent from Greengrass Core running on platform: Linux-4.9.30-
v7+-armv7l-with-debian-8.0"
}
```

雖然 Lambda AWS IoT Greengrass 函數會持續將 MQTT 訊息傳送至hello/world主題，但請勿停止協助程式。剩餘的模組會在假設其仍在執行中的情況下寫入。

您可以從群組刪除函數和訂閱：

- 在群組組態頁面的 Lambda 函數索引標籤下，選取您要移除的 Lambda 函數，然後選擇移除。
- 在群組組態頁面的訂閱索引標籤下，選擇訂閱，然後選擇刪除。

函數和訂閱會在接下來的部署群組期間從核心中移除。

單元 3（第 2 部分）：Lambda 函數開啟 AWS IoT Greengrass

本單元探索在 AWS IoT Greengrass 核心上執行的隨需和長期 Lambda 函數之間的差異。

開始之前，請先執行 [Greengrass 裝置安裝](#) 指令碼，或確定您已完成[單元 1](#)、[單元 2](#) 和 [單元 3 \(第 1 部分\)](#)。

此單元需約 30 分鐘完成。

主題

- [建立和封裝 Lambda 函數](#)
- [為 設定長期 Lambda 函數 AWS IoT Greengrass](#)
- [測試長期 Lambda 函數](#)
- [測試隨需 Lambda 函數](#)

建立和封裝 Lambda 函數

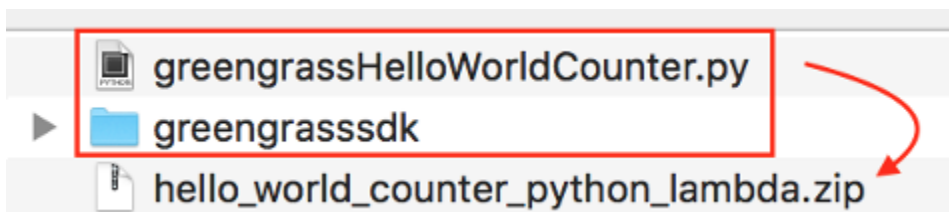
在此步驟中，您：

- 建立包含函數程式碼和相依性的 Lambda 函數部署套件。
- 使用 Lambda 主控台建立 Lambda 函數並上傳部署套件。
- 發佈 Lambda 函數的版本，並建立指向該版本的別名。

1. 在電腦上，前往您在單元 3-1 [the section called “建立和封裝 Lambda 函數”](#) 中下載並擷取的適用於 Python AWS IoT Greengrass 的核心 SDK。

此模組中的 Lambda 函數使用：

- examples\HelloWorldCounter 中的 greengrassHelloWorldCounter.py 檔案。這是您的 Lambda 函數程式碼。
 - greengrasssdk 資料夾。此為軟體開發套件。
2. 建立 Lambda 函數部署套件：
 - a. 將 greengrasssdk 資料夾複製到包含 greengrassHelloWorldCounter.py 的 HelloWorldCounter 資料夾中。
 - b. 將 greengrassHelloWorldCounter.py 和 greengrasssdk 資料夾儲存至名為 hello_world_counter_python_lambda.zip 的 zip 檔案。py 檔案和 greengrasssdk 資料夾必須為在目錄的根內。



在安裝 zip 的 UNIX 式系統 (包括 Mac 終端機) 上，可使用以下命令來封裝檔案和資料夾：

```
zip -r hello_world_counter_python_lambda.zip greengrasssdk
greengrassHelloWorldCounter.py
```

現在您已準備好建立 Lambda 函數並上傳部署套件。

3. 開啟 Lambda 主控台，然後選擇建立函數。
4. 選擇從頭開始撰寫。
5. 將您的函數命名為 **Greengrass_HelloWorld_Counter**，並將其餘的欄位設定如下：
 - 針對執行期，選擇 Python 3.7。
 - 對於許可，請保留預設設定。這會建立授予基本 Lambda 許可的執行角色。不會使用此角色 AWS IoT Greengrass。或者，您可以重複使用您在單元 3-1 中建立的角色。

選擇 Create function (建立函數)。

The screenshot shows the 'Basic information' tab of the AWS Lambda 'Create function' wizard. It includes a 'Function name' field with the value 'Greengrass_HelloWorld_Counter', a 'Runtime' dropdown menu set to 'Python 3.7', and a 'Permissions' section with a link to 'Change default execution role'. At the bottom, there are 'Cancel' and 'Create function' buttons.

Basic information

Function name
Enter a name that describes the purpose of your function.

Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime [Info](#)
Choose the language to use to write your function.

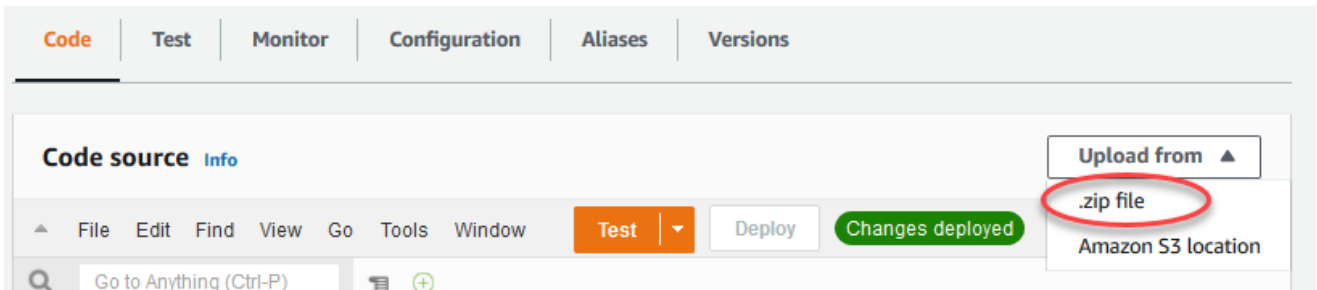
Permissions [Info](#)
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

► [Change default execution role](#)

► **Advanced settings**

[Cancel](#) [Create function](#)

6. 上傳您的 Lambda 函數部署套件。
 - a. 在程式碼索引標籤的程式碼來源下，選擇從中上傳。從下拉式清單中選擇 .zip 檔案。

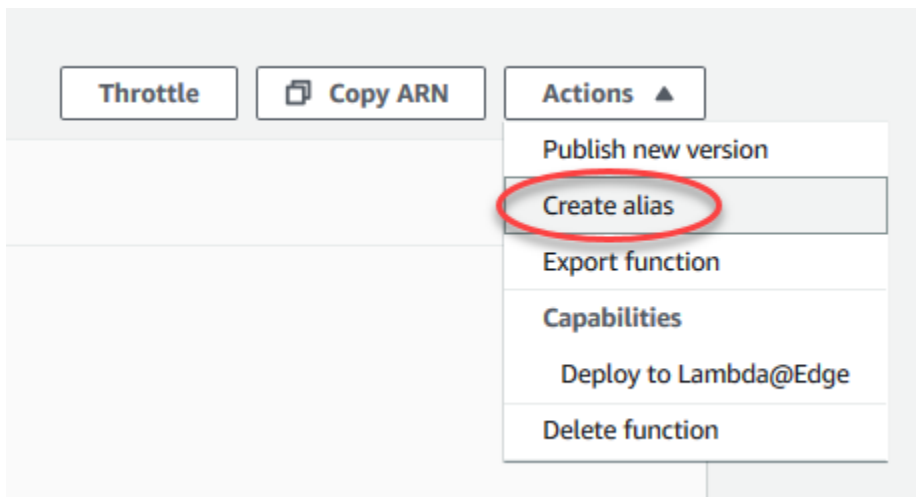


- b. 選擇上傳，然後選擇您的hello_world_counter_python_lambda.zip部署套件。然後選擇 Save (儲存)。
- c. 在函數的程式碼索引標籤的執行時間設定下，選擇編輯，然後輸入下列值。
 - 針對執行期，選擇 Python 3.7。
 - 對於 Handler (處理常式)，輸入 **greengrassHelloWorldCounter.function_handler**。
- d. 選擇 Save (儲存)。

Note

AWS Lambda 主控台上的測試按鈕不適用於此函數。AWS IoT Greengrass 核心 SDK 不包含在 AWS Lambda 主控台中獨立執行 Greengrass Lambda 函數所需的模組。這些模組 (例如 greengrass_common) 會在部署到您的 Greengrass 核心之後提供給函數。

7. 發佈函數的第一個版本。
 - a. 從頁面頂端的動作選單中，選擇發佈新版本。針對 Version description (版本描述)，輸入 **First version**。
 - b. 選擇 Publish (發布)。
8. 建立函數版本的別名。
 - a. 從頁面頂端的動作選單中，選擇建立別名。



- b. 對於名稱，輸入 **GG_HW_Counter**。
- c. 對於版本，選擇 1。
- d. 選擇 Save (儲存)。

Create alias

Alias configuration

An alias is a pointer to one or two versions. Choose each version that you want the alias to point to.

Name

Description - optional

Version

1 ▼

► **Weighted alias**

CancelSave

別名會為您的 Lambda 函數建立單一實體，供 Greengrass 裝置訂閱。如此一來，您就不必在每次修改函數時，使用新的 Lambda 函數版本編號更新訂閱。

為 設定長期 Lambda 函數 AWS IoT Greengrass

您現在已準備好設定 Lambda 函數 AWS IoT Greengrass。

1. 在 AWS IoT 主控台導覽窗格的管理下，展開 Greengrass 裝置，然後選擇群組 (V1)。
2. 在 Greengrass 群組下，選擇您在[單元 2](#) 中建立的群組。
3. 在群組組態頁面上，選擇 Lambda 函數索引標籤，然後在我的 Lambda 函數下，選擇新增。
4. 針對 Lambda 函數，選擇 Greengrass_HelloWorld_Counter。
5. 針對 Lambda 函數版本，選擇您發佈的版本別名。
6. 針對逾時（秒），輸入 **25**。此 Lambda 函數會在每次調用前休眠 20 秒。
7. 針對固定，選擇 True。
8. 保留所有其他欄位的預設值，然後選擇新增 Lambda 函數。

測試長期 Lambda 函數

當 AWS IoT Greengrass 核心在單一容器（或沙盒）中啟動和執行時，[長期](#) Lambda 函數會自動啟動。在函數處理常式外定義的任何變數或預先處理的邏輯，每次呼叫函數處理常式時都會保留下來。多個函數處理常式的呼叫已排入佇列，直到已執行舊的呼叫。

在此單元中使用的 greengrassHelloWorldCounter.py 程式碼可在函數處理常式外定義 my_counter 變數。

Note

您可以在 AWS Lambda 主控台或 GitHub 上[AWS IoT Greengrass 適用於 Python 的 Core SDK](#) 中檢視程式碼。

在此步驟中，您會建立允許 Lambda 函數和 AWS IoT 交換 MQTT 訊息的訂閱。然後，您部署群組和測試函數。

1. 在群組組態頁面上，選擇訂閱，然後選擇新增。
2. 在來源類型下，選擇 Lambda 函數，然後選擇 Greengrass_HelloWorld_Counter。
3. 在目標類型下，選擇服務，選擇 IoT 雲端。
4. 針對 Topic filter (主題篩選條件)，輸入 **hello/world/counter**。

5. 選擇 Create subscription (建立訂閱)。

此單一訂閱僅朝一個方向進行：從 Greengrass_HelloWorld_Counter Lambda 函數到 AWS IoT。若要從雲端叫用（或觸發）此 Lambda 函數，您必須以相反方向建立訂閱。

6. 請依照步驟 1 - 5 新增另一個使用下列值的訂閱。此訂閱允許 Lambda 函數接收來自的訊息 AWS IoT。當您從叫用函數的 AWS IoT 主控台傳送訊息時，您會使用此訂閱。

- 針對來源，選擇服務，然後選擇 IoT 雲端。
- 針對目標，選擇 Lambda 函數，然後選擇 Greengrass_HelloWorld_Counter。
- 針對主題篩選條件，輸入 **hello/world/counter/trigger**。

此主題篩選條件中使用 /trigger 延伸部分，因為您已建立兩個訂閱，並且不希望它們互相影響。

7. 請確定 Greengrass 協助程式正在執行，如中所述[部署雲端組態到核心裝置](#)。

8. 在群組組態頁面上，選擇部署。

9. 部署完成後，返回 AWS IoT 主控台首頁，然後選擇測試。

10. 設定下列欄位：

- 針對 Subscription topic (訂閱主題)，輸入 **hello/world/counter**。
- 針對 Quality of Service (服務品質)，選擇 0。
- 針對 MQTT payload display (MQTT 承載顯示)，選擇 Display payloads as strings (將承載顯示為字串)。

11. 選擇 Subscribe (訂閱)。

與此模組的[第 1 部分](#)不同，您應該不會在訂閱 hello/world/counter 之後看到任何訊息。這是因為發佈到該 hello/world/counter 主題的 greengrassHelloWorldCounter.py 程式碼是在函數處理常式中，該處理常式僅會在叫用該函數時執行。

在此模組中，您設定 Greengrass_HelloWorld_Counter Lambda 函數在接收hello/world/counter/trigger有關主題的 MQTT 訊息時被叫用。

Greengrass_HelloWorld_Counter 對 IoT Cloud (IoT 雲端) 的訂閱允許函數將訊息傳送至 hello/world/counter 主題的 AWS IoT。IoT 雲端至 Greengrass_HelloWorld_Counter 訂閱允許 AWS IoT 傳送訊息至hello/world/counter/trigger主題上的函數。

12. 若要測試長期生命週期，請發佈訊息至 hello/world/counter/trigger 主題來叫用 Lambda 函數。您可以使用預設的訊息。

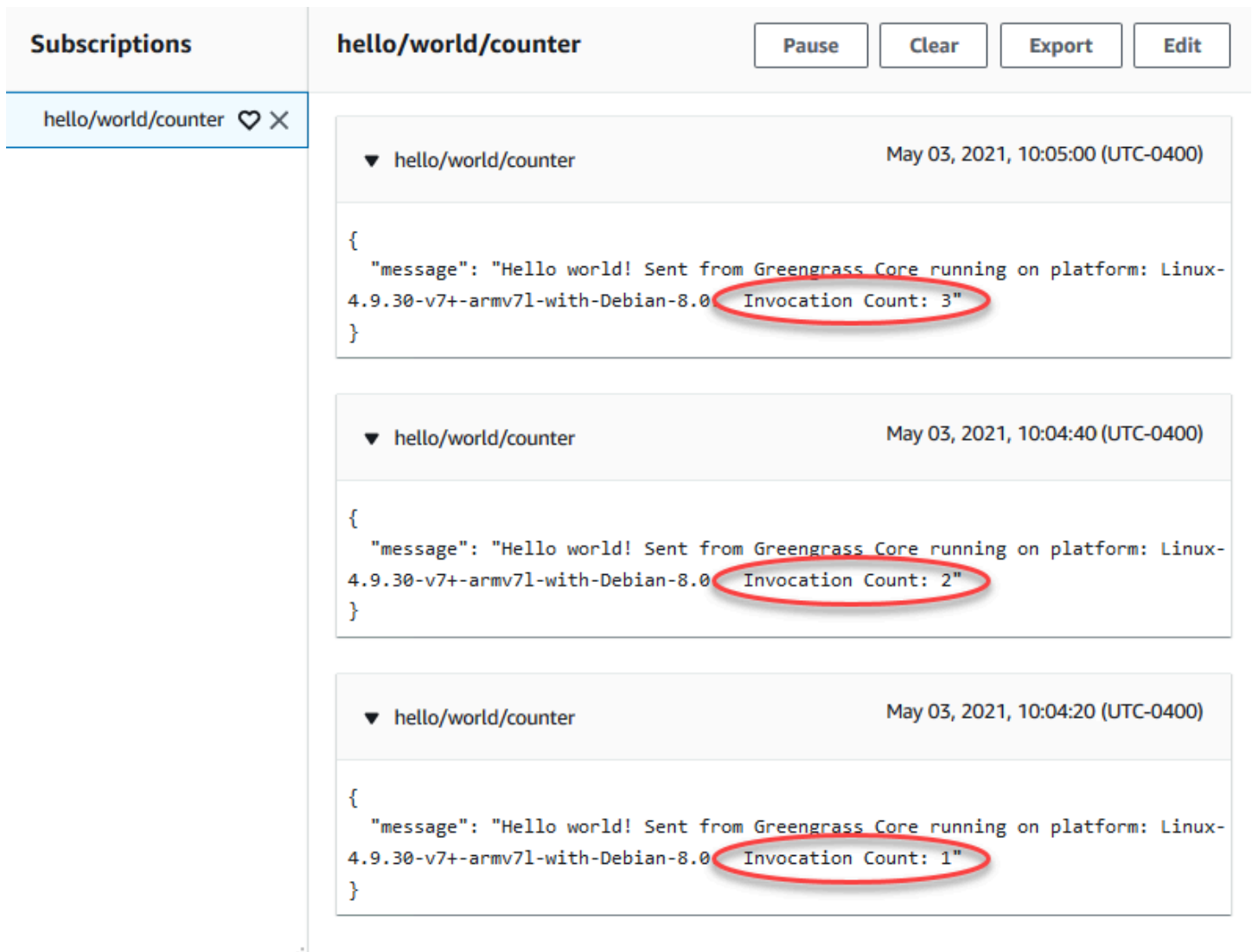
Subscribe to a topic**Publish to a topic****Topic name**

The topic name identifies the message. The message payload will be published to this topic with a Quality of Service (QoS) of 0.

Message payload**► Additional configuration****Publish****Note**

Greengrass_HelloWorld_Counter 函數會忽略接收訊息的內容。它只需在 `function_handler` 中執行程式碼，這會將訊息傳送到 `hello/world/counter` 主題。您可以從 GitHub 上的 [AWS IoT Greengrass 適用於 Python 的核心 SDK](#) 檢閱此程式碼。

每次將訊息發佈給 `hello/world/counter/trigger` 主題時，`my_counter` 變數會增加。此調用計數會顯示在從 Lambda 函數傳送的訊息中。由於函數處理常式包含 20 秒的休眠週期 (`time.sleep(20)`)，因此重複觸發處理常式會將 AWS IoT Greengrass 核心的回應排入佇列。



The screenshot displays the 'Subscriptions' section of the AWS IoT Greengrass console. A subscription named 'hello/world/counter' is selected. The page shows three invocation logs, each with a timestamp and a message. The 'Invocation Count' is highlighted in red in each log.

Subscription	Timestamp	Message	Invocation Count
hello/world/counter	May 03, 2021, 10:05:00 (UTC-0400)	{ "message": "Hello world! Sent from Greengrass Core running on platform: Linux-4.9.30-v7+-armv7l-with-Debian-8.0" }	3
hello/world/counter	May 03, 2021, 10:04:40 (UTC-0400)	{ "message": "Hello world! Sent from Greengrass Core running on platform: Linux-4.9.30-v7+-armv7l-with-Debian-8.0" }	2
hello/world/counter	May 03, 2021, 10:04:20 (UTC-0400)	{ "message": "Hello world! Sent from Greengrass Core running on platform: Linux-4.9.30-v7+-armv7l-with-Debian-8.0" }	1

測試隨需 Lambda 函數

隨需 Lambda 函數的功能與雲端 AWS Lambda 函數類似。隨需 Lambda 函數的多個調用可以平行執行。如果資源允許，Lambda 函數的調用會建立個別的容器來處理調用或重複使用現有容器。任何在函數處理常式外部定義的變數或預先處理都不會在建立容器時保留。

1. 在群組組態頁面上，選擇 Lambda 函數索引標籤。
2. 在我的 Lambda 函數下，選擇 Greengrass_HelloWorld_Counter Lambda 函數。
3. 在 Greengrass_HelloWorld_Counter 詳細資訊頁面上，選擇編輯。
4. 針對固定，選擇 False，然後選擇儲存。
5. 在群組組態頁面上，選擇部署。
6. 部署完成後，請返回 AWS IoT 主控台首頁，然後選擇測試。

7. 設定下列欄位：

- 針對 Subscription topic (訂閱主題)，輸入 **hello/world/counter**。
- 針對 Quality of Service (服務品質)，選擇 0。
- 針對 MQTT payload display (MQTT 承載顯示)，選擇 Display payloads as strings (將承載顯示為字串)。

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.

hello/world/counter

▼ Additional configuration

Number of messages to keep
The MQTT test client keeps this many of the most recent messages published to a topic that matches this topic filter.

100

Quality of service
When subscribing to a topic, quality of service 0 will be chosen by default.

☒ Quality of Service 0 - Message will be delivered at most once
☐ Quality of Service 1 - Message will be delivered at least once

MQTT payload display

☒ Display payloads as strings (more accurate)
☐ Auto-format JSON payloads (improves readability)
☐ Display raw payloads (displays binary data as hexadecimal values)

Subscribe

8. 選擇 Subscribe (訂閱)。

Note

您應該不會在訂閱之後看到任何訊息。

9. 若要測試隨需生命週期，請透過將訊息發佈到 `hello/world/counter/trigger` 主題來叫用函數。您可以使用預設的訊息。
 - a. 選擇快速發佈三次，每次按下按鈕後五秒內。

Subscribe to a topic | **Publish to a topic**

Topic name
The topic name identifies the message. The message payload will be published to this topic with a Quality of Service (QoS) of 0.

Message payload

► **Additional configuration**

Publish x3

每個發佈都會呼叫函數處理常式，並為每次呼叫建立容器。呼叫計數不會在您觸發函數的三次中遞增，因為每個隨需 Lambda 函數都有自己的容器/沙盒。

The screenshot shows the AWS IoT Greengrass Subscriptions console. On the left, a sidebar titled 'Subscriptions' contains a link for 'hello/world/counter' with a heart and close icon. The main panel is titled 'hello/world/counter' and includes 'Pause', 'Clear', 'Export', and 'Edit' buttons. It displays a list of three messages for the 'hello/world/counter' topic, each with a timestamp and a JSON payload. In each payload, the 'Invocation Count: 1' is circled in red.

Topic	Timestamp	Message
hello/world/counter	May 03, 2021, 10:14:13 (UTC-0400)	<pre>{ "message": "Hello world! Sent from Greengrass Core running on platform: Linux-4.9.30-v7+-armv7l-with-Debian-8.0. Invocation Count: 1" }</pre>
hello/world/counter	May 03, 2021, 10:14:12 (UTC-0400)	<pre>{ "message": "Hello world! Sent from Greengrass Core running on platform: Linux-4.9.30-v7+-armv7l-with-Debian-8.0. Invocation Count: 1" }</pre>
hello/world/counter	May 03, 2021, 10:14:11 (UTC-0400)	<pre>{ "message": "Hello world! Sent from Greengrass Core running on platform: Linux-4.9.30-v7+-armv7l-with-Debian-8.0. Invocation Count: 1" }</pre>

- b. 約 30 秒後，選擇 Publish to topic (發佈到主題)。呼叫計數應遞增到 2。這表示從之前呼叫中建立的容器目前正在重新使用，並且已存放在函數處理常式之外的預先處理變數。

Subscriptions

hello/world/counter

hello/world/counter Pause Clear Export Edit

▼ hello/world/counter May 03, 2021, 10:14:40 (UTC-0400)

```
{
  "message": "Hello world! Sent from Greengrass Core running on platform: Linux-4.9.30-v7+-armv7l-with-Debian-8.0. Invocation Count: 2"
}
```

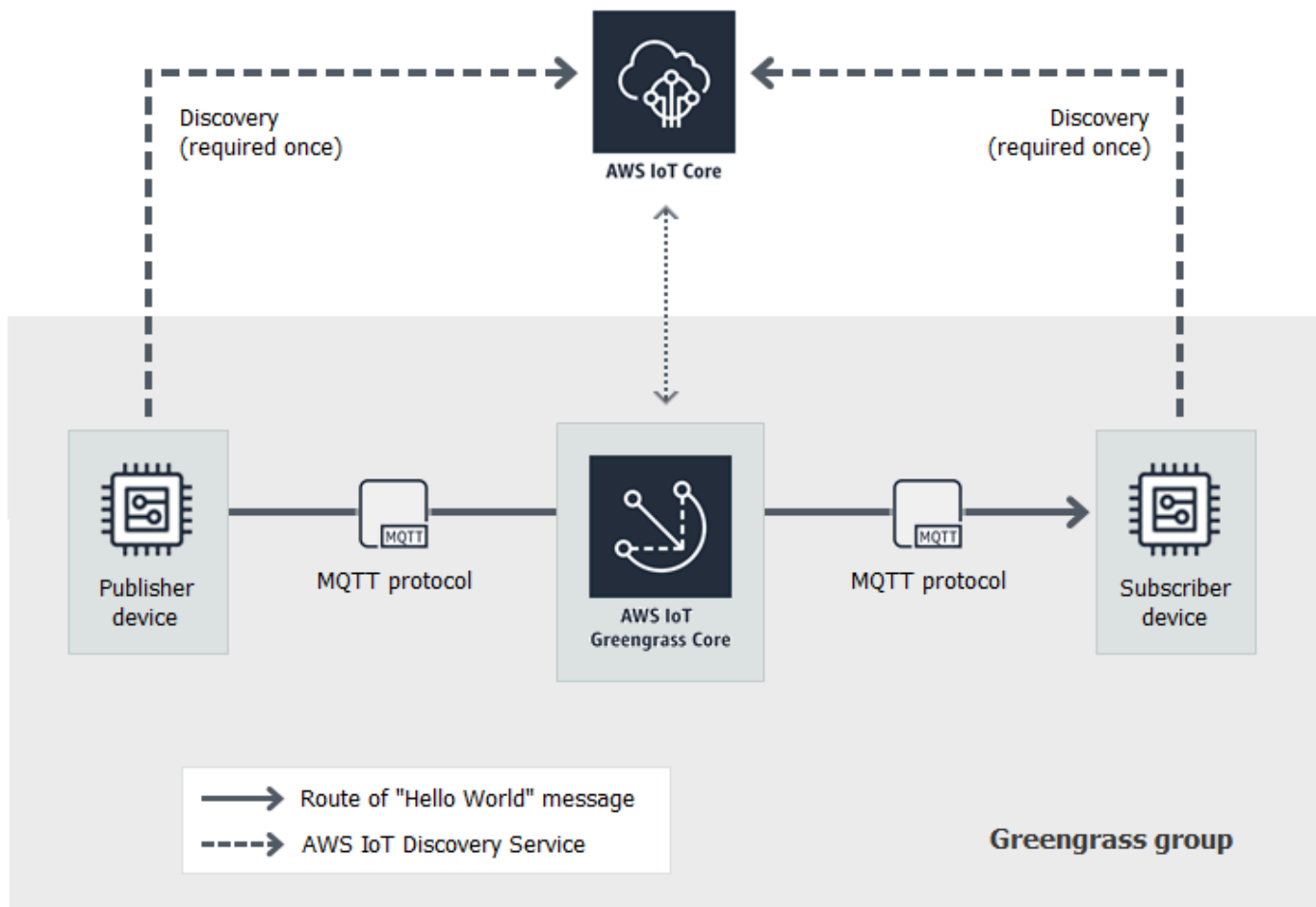
▼ hello/world/counter May 03, 2021, 10:14:11 (UTC-0400)

```
{
  "message": "Hello world! Sent from Greengrass Core running on platform: Linux-4.9.30-v7+-armv7l-with-Debian-8.0. Invocation Count: 1"
}
```

您現在應該了解可在 AWS IoT Greengrass 核心上執行的兩種 Lambda 函數類型。下一個模組，[模組 4](#)，向您展示本機 IoT 裝置如何在 AWS IoT Greengrass 群組中互動。

單元 4：與 AWS IoT Greengrass 群組中的用戶端裝置互動

本單元會示範稱為用戶端裝置或裝置的本機 IoT 裝置如何連線至 AWS IoT Greengrass 核心裝置並與之通訊。連接到 AWS IoT Greengrass 核心的用戶端裝置是 AWS IoT Greengrass 群組的一部分，可以參與 AWS IoT Greengrass 程式設計範例。在此模組中，一個用戶端裝置會將 Hello World 訊息傳送至 Greengrass 群組中的另一個用戶端裝置。



在您開始之前，請執行 [Greengrass 裝置安裝](#) 指令碼，或完成[單元 1](#) 及 [單元 2](#)。此模組會建立兩個模擬用戶端裝置。您不需要其他的元件或裝置。

本單元應可在 30 分鐘內完成。

主題

- [在 AWS IoT Greengrass 群組中建立用戶端裝置](#)
- [設定訂閱](#)
- [安裝 AWS IoT Device SDK for Python](#)
- [測試通訊](#)

在 AWS IoT Greengrass 群組中建立用戶端裝置

在此步驟中，您會將兩個用戶端裝置新增至 Greengrass 群組。此程序包括將裝置註冊為 AWS IoT 實體，以及設定憑證和金鑰以允許裝置連線 AWS IoT Greengrass。

1. 在 AWS IoT 主控台導覽窗格的管理下，展開 Greengrass 裝置，然後選擇群組 (V1)。
2. 選擇目標群組。
3. 在群組組態頁面上，選擇用戶端裝置，然後選擇關聯。
4. 在將用戶端裝置與此群組模態建立關聯中，選擇建立新 AWS IoT 物件。

建立物件頁面會在新標籤中開啟。

5. 在建立物件頁面上，選擇建立單一物件，然後選擇下一步。
6. 在指定物件屬性頁面上，將此用戶端裝置註冊為 **HelloWorld_Publisher**，然後選擇下一步。
7. 在設定裝置憑證頁面上，選擇下一步。
8. 在將政策連接至憑證頁面上，執行下列其中一項操作：
 - 選取授予用戶端裝置所需許可的現有政策，然後選擇建立物件。

模態開啟，您可以在其中下載裝置用來連線至和核心的憑證 AWS 雲端和金鑰。

- 建立並連接授予用戶端裝置許可的新政策。請執行下列操作：
 - a. 選擇建立政策。

Create policy (建立政策) 頁面隨即在新標籤中開啟。

- b. 在 Create policy (建立政策) 頁面上，執行下列動作：
 - i. 針對政策名稱，輸入描述政策的名稱，例如 **GreengrassV1ClientDevicePolicy**。
 - ii. 在政策陳述式索引標籤的政策文件下，選擇 JSON。
 - iii. 輸入下列政策文件。此政策可讓用戶端裝置探索 Greengrass 核心，並在所有 MQTT 主題上進行通訊。如需如何限制此政策存取的資訊，請參閱 [的裝置身分驗證和授權 AWS IoT Greengrass](#)。

JSON

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

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

iv. 選擇 Create (建立) 以建立政策。

c. 返回瀏覽器索引標籤，開啟附加政策至憑證頁面。請執行下列操作：

i. 在政策清單中，選取您建立的政策，例如 GreengrassV1ClientDevicePolicy。

如果您沒有看到政策，請選擇重新整理按鈕。

ii. 選擇 Create thing (建立物件)。

模式開啟，您可以在其中下載裝置用來連線至 和 核心的憑證 AWS 雲端 和金鑰。

9. 在下載憑證和金鑰模式中，下載裝置的憑證。

Important

選擇完成之前，請下載安全資源。

請執行下列操作：

- a. 針對裝置憑證，選擇下載以下載裝置憑證。
- b. 針對公有金鑰檔案，選擇下載以下載憑證的公有金鑰。
- c. 針對私有金鑰檔案，選擇下載以下載憑證的私有金鑰檔案。
- d. 檢閱《AWS IoT 開發人員指南》中的[伺服器身分驗證](#)，然後選擇適當的根 CA 憑證。我們建議您使用 Amazon Trust Services (ATS) 端點和 ATS 根 CA 憑證。在根 CA 憑證下，選擇根 CA 憑證的下載。
- e. 選擇完成。

請記下裝置憑證和金鑰檔案名稱中常見的憑證 ID。供稍後使用。

10. 返回瀏覽器索引標籤，並將用戶端裝置與此群組模式開啟的關聯。請執行下列操作：

- a. 針對AWS IoT 物件名稱，選擇您建立的HelloWorld_Publisher物件。

如果您沒有看到物件，請選擇重新整理按鈕。

- b. 選擇關聯。

11. 重複步驟 3 - 10，將第二個用戶端裝置新增至群組。

將此用戶端裝置命名為 **HelloWorld_Subscriber**。將此用戶端裝置的憑證和金鑰下載到您的電腦。同樣地，請記下 HelloWorld_Subscriber 裝置檔案名稱中常見的憑證 ID。

您現在應該在 Greengrass 群組中有兩個用戶端裝置：

- HelloWorld_Publisher
- HelloWorld_Subscriber

12. 在電腦上為這些用戶端裝置的安全登入資料建立資料夾。將憑證和金鑰複製到此資料夾。

設定訂閱

在此步驟中，您會啟用 HelloWorld_Publisher 用戶端裝置，將 MQTT 訊息傳送至 HelloWorld_Subscriber 用戶端裝置。

1. 在群組組態頁面上，選擇訂閱索引標籤，然後選擇新增。
2. 在建立訂閱頁面上，執行下列動作來設定訂閱：
 - a. 針對來源類型，選擇用戶端裝置，然後選擇 HelloWorld_Publisher。

- b. 在目標類型下，選擇用戶端裝置，然後選擇 HelloWorld_Subscriber。
- c. 針對 Topic filter (主題篩選條件)，輸入 **hello/world/pubsub**。

 Note

您可以將訂閱從先前模組中刪除。在群組的訂閱頁面上，選取要刪除的訂閱，然後選擇刪除。

- d. 選擇 Create subscription (建立訂閱)。
3. 確定已啟用自動偵測，這樣 Greengrass 核心就能發佈其 IP 地址清單。用戶端裝置會使用此資訊來探索核心。請執行下列操作：
 - a. 在群組組態頁面上，選擇 Lambda 函數索引標籤。
 - b. 在 System Lambda 函數下，選擇 IP 偵測器，然後選擇編輯。
 - c. 在編輯 IP 偵測器設定中，選擇自動偵測和覆寫 MQTT 代理程式端點，然後選擇儲存。
 4. 請確定 Greengrass 協助程式正在執行，如中所述[部署雲端組態到核心裝置](#)。
 5. 在群組組態頁面上，選擇部署。

部署狀態會顯示在頁面標頭的群組名稱底下。若要查看部署詳細資訊，請選擇部署索引標籤。

安裝 AWS IoT Device SDK for Python

用戶端裝置可以使用 AWS IoT Device SDK 適用於 Python 的與 AWS IoT 和 AWS IoT Greengrass 核心裝置通訊（使用 Python 程式設計語言）。如需詳細資訊，包括需求，請參閱 GitHub 上的 AWS IoT Device SDK 適用於 Python [Readme](#) 的。

在此步驟中，您會安裝 SDK，並取得模擬用戶端裝置在電腦上使用 `basicDiscovery.py` 的範例函數。

1. 若要將軟體開發套件與所有需要的元件安裝到您的電腦，請選擇您的作業系統：

Windows

1. 請開啟[提高命令提示](#)，並執行下列命令：

```
python --version
```

若沒有傳回任何版本資訊，或是版本號碼小於 2.7 (Python 2) 或小於 3.3 (Python 3)，請遵循 [Downloading Python](#) 中的說明來安裝 Python 2.7+ 或 Python 3.3+。如需詳細資訊，請參閱 [Using Python on Windows](#)。

2. 將 [AWS IoT Device SDK for Python](#) 下載為 zip 檔案，並將其解壓縮到電腦上的適當位置。

記下已解壓縮之 `aws-iot-device-sdk-python-master` 資料夾的檔案路徑，該資料夾包含 `setup.py` 檔案。在下一個步驟中，這檔案路徑會以 *path-to-SDK-folder* 表示。

3. 自提高命令提示執行下列命令：

```
cd path-to-SDK-folder
python setup.py install
```

macOS

1. 請開啟終端機視窗並執行下列命令：

```
python --version
```

若沒有傳回任何版本資訊，或是版本號碼小於 2.7 (Python 2) 或小於 3.3 (Python 3)，請遵循 [Downloading Python](#) 中的說明來安裝 Python 2.7+ 或 Python 3.3+。如需詳細資訊，請參閱 [Using Python on a Macintosh](#)。

2. 在終端機視窗中執行下列命令以判斷 OpenSSL 版本：

```
python
>>>import ssl
>>>print ssl.OPENSSL_VERSION
```

記下 OpenSSL 的版本值。

Note

若您執行的是 Python 3，請使用 `print(ssl.OPENSSL_VERSION)`。

若要關閉 Python shell，請執行下列命令：

```
>>>exit()
```

如果 OpenSSL 版本為 1.0.1 或更新，請跳到[步驟 c](#)。否則，請遵循這些步驟：

- 請從終端機視窗中執行下列命令來判斷電腦是否使用 Simple Python 版本管理：

```
which pyenv
```

如果傳回檔案路徑，請選擇使用 **pyenv** 標籤。如果未傳回，請選擇不使用 **pyenv** 標籤。

Using pyenv

- 請參閱 [Python Releases for Mac OS X](#) (或類似) 以判斷最新又穩定的 Python 版本。在以下範例中，此數值會以 *latest-Python-version* 表示。
- 從終端機視窗中執行下列命令：

```
pyenv install latest-Python-version  
pyenv global latest-Python-version
```

例如，若 Python 2 的最新版本是 2.7.14，則這些命令為：

```
pyenv install 2.7.14  
pyenv global 2.7.14
```

- 關閉然後重新開啟終端機視窗，執行以下命令：

```
python  
>>>import ssl  
>>>print ssl.OPENSSL_VERSION
```

OpenSSL 版本至少應為 1.0.1。若版本小於 1.0.1，則更新會失敗。檢查 `pyenv install` 和 `pyenv global` 命令中所使用的 Python 版本值，然後再試一次。

- 執行下列命令退出 Python shell：

```
exit()
```

Not using pyenv

1. 從終端機視窗執行下列命令來判斷 [brew](#) 是否已安裝：

```
which brew
```

如果沒有傳回檔案的路徑，請安裝如下所示的 brew：

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

Note

遵循安裝提示。下載 Xcode 命令列工具可能需要一些時間。

2. 執行下列命令：

```
brew update  
brew install openssl  
brew install python@2
```

AWS IoT Device SDK 適用於 Python 的 需要使用 Python 可執行檔編譯的 OpenSSL 1.0.1 版（或更新版本）。brew install python 命令會安裝 python2 可執行檔，以符合此需求。python2 可執行檔安裝於 /usr/local/bin 目錄中，應屬於 PATH 環境變數的一部分。若要確認，請執行下列命令：

```
python2 --version
```

如果已提供 python2 版本資訊，請跳到下一個步驟。若否，請新增 /usr/local/bin 路徑至您的 PATH 環境變數，此需為永久持續，透過將下行附加至您的 shell 描述檔：

```
export PATH="/usr/local/bin:$PATH"
```

例如，如果您使用的是 `.bash_profile` 或沒有 shell 設定檔從終端視窗，請執行下列命令：

```
echo 'export PATH="/usr/local/bin:$PATH"' >> ~/.bash_profile
```

接著，[來源](#)您的 shell 描述檔，並確認 `python2 --version` 提供版本資訊。例如，如果您是使用 `.bash_profile`，請執行下列命令：

```
source ~/.bash_profile
python2 --version
```

應傳回 `python2` 版本資訊。

3. 附加下行至您的 shell 描述檔中：

```
alias python="python2"
```

例如，如果您是使用 `.bash_profile` 或尚未有 shell 描述檔，則請執行下列命令：

```
echo 'alias python="python2"' >> ~/.bash_profile
```

4. 接著，[來源](#)您的 shell 描述檔。例如，如果您是使用 `.bash_profile`，請執行下列命令：

```
source ~/.bash_profile
```

呼叫 `python` 命令會執行 Python 可執行檔，其中包含所需的 OpenSSL 版本 (`python2`)。

5. 執行下列命令：

```
python
import ssl
print ssl.OPENSSL_VERSION
```

OpenSSL 版本至少應為 1.0.1 或更新版。

6. 若要退出 Python shell，請執行下列命令：

```
exit()
```

3. 執行下列命令來安裝 AWS IoT Device SDK for Python：

```
cd ~  
git clone https://github.com/aws/aws-iot-device-sdk-python.git  
cd aws-iot-device-sdk-python  
sudo python setup.py install
```

UNIX-like system

1. 從終端機視窗中，執行下列命令：

```
python --version
```

若沒有傳回任何版本資訊，或是版本號碼小於 2.7 (Python 2) 或小於 3.3 (Python 3)，請遵循 [Downloading Python](#) 中的說明來安裝 Python 2.7+ 或 Python 3.3+。如需詳細資訊，請參閱 [Using Python on Unix platforms](#)。

2. 在終端機中執行下列命令以判斷 OpenSSL 版本：

```
python  
>>>import ssl  
>>>print ssl.OPENSSL_VERSION
```

記下 OpenSSL 的版本值。

Note

若您執行的是 Python 3，請使用 `print(ssl.OPENSSL_VERSION)`。

若要關閉 Python shell，請執行下列命令：

```
exit()
```

如果 OpenSSL 版本為 1.0.1 或更新，請跳到下一個步驟。否則，請執行命令來更新您發行版本的 OpenSSL (例如 `sudo yum update openssl`、`sudo apt-get update` 等)。

透過執行下列命令確認 OpenSSL 版本為 1.0.1 或更新版：

```
python
>>>import ssl
>>>print ssl.OPENSSL_VERSION
>>>exit()
```

3. 執行下列命令來安裝 AWS IoT Device SDK for Python：

```
cd ~
git clone https://github.com/aws/aws-iot-device-sdk-python.git
cd aws-iot-device-sdk-python
sudo python setup.py install
```

2. 安裝 AWS IoT Device SDK 適用於 Python 的之後，導覽至 `samples` 資料夾並開啟 `greengrass` 資料夾。

在本教學課程中，您將複製 `basicDiscovery.py` 範例函數，該函數會使用您在 [the section called “在 AWS IoT Greengrass 群組中建立用戶端裝置”](#) 中下載的憑證和金鑰。

3. `basicDiscovery.py` 複製到包含 `HelloWorld_Publisher` 和 `HelloWorld_Subscriber` 裝置憑證和金鑰的資料夾。

測試通訊

1. 請確定您的電腦和 AWS IoT Greengrass 核心裝置已使用相同的網路連線至網際網路。
 - a. 在 AWS IoT Greengrass 核心裝置上，執行下列命令來尋找其 IP 地址。

```
hostname -I
```

- b. 在您的電腦，使用核心的 IP 地址執行以下命令。您可以使用 `Ctrl + C` 來停止 `ping` 命令。

```
ping IP-address
```

類似以下的輸出表示電腦與 AWS IoT Greengrass 核心裝置之間的通訊成功（封包遺失 0%）：

```
$ping 176.32.103.205
PING 176.32.103.205 (176.32.103.205) 56(84) bytes of data.
64 bytes from 176.32.103.205: icmp_seq=1 ttl=230 time=77.2 ms
64 bytes from 176.32.103.205: icmp_seq=2 ttl=230 time=77.1 ms
64 bytes from 176.32.103.205: icmp_seq=3 ttl=230 time=77.1 ms
64 bytes from 176.32.103.205: icmp_seq=4 ttl=230 time=77.1 ms
64 bytes from 176.32.103.205: icmp_seq=5 ttl=230 time=77.1 ms
64 bytes from 176.32.103.205: icmp_seq=6 ttl=230 time=77.1 ms
^C
--- 176.32.103.205 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5549ms
rtt min/avg/max/mdev = 77.107/77.172/77.256/0.361 ms
```

Note

如果您無法 ping 執行中的 EC2 執行個體 AWS IoT Greengrass，請確定執行個體的傳入安全群組規則允許 [Echo 請求](#) 訊息的 ICMP 流量。如需詳細資訊，請參閱《Amazon EC2 使用者指南》中的 [將規則新增至安全群組](#)。

在 Windows 主機電腦的 Windows 防火牆和進階安全應用程式中，您可能需要啟用允許傳入回聲請求的內送規則（例如，File and Printer Sharing (Echo Request - ICMPv4-In)（檔案和印表機共用（回聲請求 - ICMPv4-In））或建立規則。

2. 取得您的 AWS IoT 端點。

- a. 從 [AWS IoT 主控台](#) 導覽窗格中，選擇設定。
- b. 在裝置資料端點下，記下端點的值。您可以使用這個值，在以下步驟的命令中取代 ***AWS_IOT_ENDPOINT*** 預留位置。

Note

請確定您的 [端點對應至您的憑證類型](#)。

3. 在您的電腦（而非 AWS IoT Greengrass 核心裝置）上，開啟兩個 [命令列](#)（終端或命令提示）視窗。一個視窗代表 HelloWorld_Publisher 用戶端裝置，另一個視窗代表 HelloWorld_Subscriber 用戶端裝置。

執行時，會 `basicDiscovery.py` 嘗試收集 AWS IoT Greengrass 核心端點位置的相關資訊。此資訊會在用戶端裝置發現並成功連接到核心後儲存。這讓未來的簡訊和操作可在本機執行 (而不需要網際網路連線)。

 Note

用於 MQTT 連線的用戶端 IDs 必須符合用戶端裝置的物件名稱。`basicDiscovery.py` 指令碼會將 MQTT 連線的用戶端 ID 設定為您在執行指令碼時指定的物件名稱。從包含 `basicDiscovery.py` 檔案的資料夾執行下列命令，以取得詳細的指令碼使用資訊：

```
python basicDiscovery.py --help
```

4. 從 HelloWorld_Publisher 用戶端裝置視窗中，執行下列命令。

- 將 *path-to-certs-folder* 替換為包含憑證、金鑰和 `basicDiscovery.py` 的資料夾路徑。
- 將 *AWS_IOT_ENDPOINT* 取代為您的端點。
- 將兩個 *publisherCertId* 執行個體取代為 HelloWorld_Publisher 用戶端裝置檔案名稱中的憑證 ID。

```
cd path-to-certs-folder  
python basicDiscovery.py --endpoint AWS_IOT_ENDPOINT --rootCA AmazonRootCA1.pem  
--cert publisherCertId-certificate.pem.crt --key publisherCertId-private.pem.key  
--thingName HelloWorld_Publisher --topic 'hello/world/pubsub' --mode publish --  
message 'Hello, World! Sent from HelloWorld_Publisher'
```

您應該會看到類似以下的輸出，其中包含像是 Published topic 'hello/world/pubsub': {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 1} 的項目。

 Note

如果指令碼傳回 error: unrecognized arguments 訊息，會將 `--topic` 和 `--message` 參數的單引號變更為雙引號並再次執行命令。

若要排除連線問題，您可以嘗試使用[手動 IP 偵測](#)。

```
Published topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 0}
2017-11-13 21:12:26,296 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Produced [puback] event
2017-11-13 21:12:26,297 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Dispatching [puback] event
2017-11-13 21:12:27,301 - AWSIoTPythonSDK.core.protocol.mqtt_core - INFO - Performing sync publish...
Published topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 1}
2017-11-13 21:12:27,302 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Produced [puback] event
2017-11-13 21:12:27,303 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Dispatching [puback] event
2017-11-13 21:12:28,305 - AWSIoTPythonSDK.core.protocol.mqtt_core - INFO - Performing sync publish...
Published topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 2}
2017-11-13 21:12:28,306 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Produced [puback] event
2017-11-13 21:12:28,307 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Dispatching [puback] event
2017-11-13 21:12:29,310 - AWSIoTPythonSDK.core.protocol.mqtt_core - INFO - Performing sync publish...
Published topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 3}
```

5. 從 HelloWorld_Subscriber 用戶端裝置視窗中，執行下列命令。

- 將 *path-to-certs-folder* 替換為包含憑證、金鑰和 basicDiscovery.py 的資料夾路徑。
- 將 *AWS_IOT_ENDPOINT* 取代為您的端點。
- 將兩個 *subscriberCertId* 執行個體取代為 HelloWorld_Subscriber 用戶端裝置檔案名稱中的憑證 ID。

```
cd path-to-certs-folder
python basicDiscovery.py --endpoint AWS_IOT_ENDPOINT --rootCA AmazonRootCA1.pem --
cert subscriberCertId-certificate.pem.crt --key subscriberCertId-private.pem.key --
thingName HelloWorld_Subscriber --topic 'hello/world/pubsub' --mode subscribe
```

您應該會看到類似以下的輸出，其中包含像是 Received message on topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 1} 的項目。

```
Received message on topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 0}
2017-11-13 21:12:27,435 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Produced [message] event
2017-11-13 21:12:27,435 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Dispatching [message] event
2017-11-13 21:12:27,436 - AWSIoTPythonSDK.core.protocol.internal.clients - DEBUG - Invoking custom event callback...
Received message on topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 1}
2017-11-13 21:12:28,320 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Produced [message] event
2017-11-13 21:12:28,324 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Dispatching [message] event
2017-11-13 21:12:28,324 - AWSIoTPythonSDK.core.protocol.internal.clients - DEBUG - Invoking custom event callback...
Received message on topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 2}
2017-11-13 21:12:29,547 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Produced [message] event
2017-11-13 21:12:29,552 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Dispatching [message] event
2017-11-13 21:12:29,552 - AWSIoTPythonSDK.core.protocol.internal.clients - DEBUG - Invoking custom event callback...
```

關閉HelloWorld_Publisher視窗以停止訊息在HelloWorld_Subscriber視窗中累積。

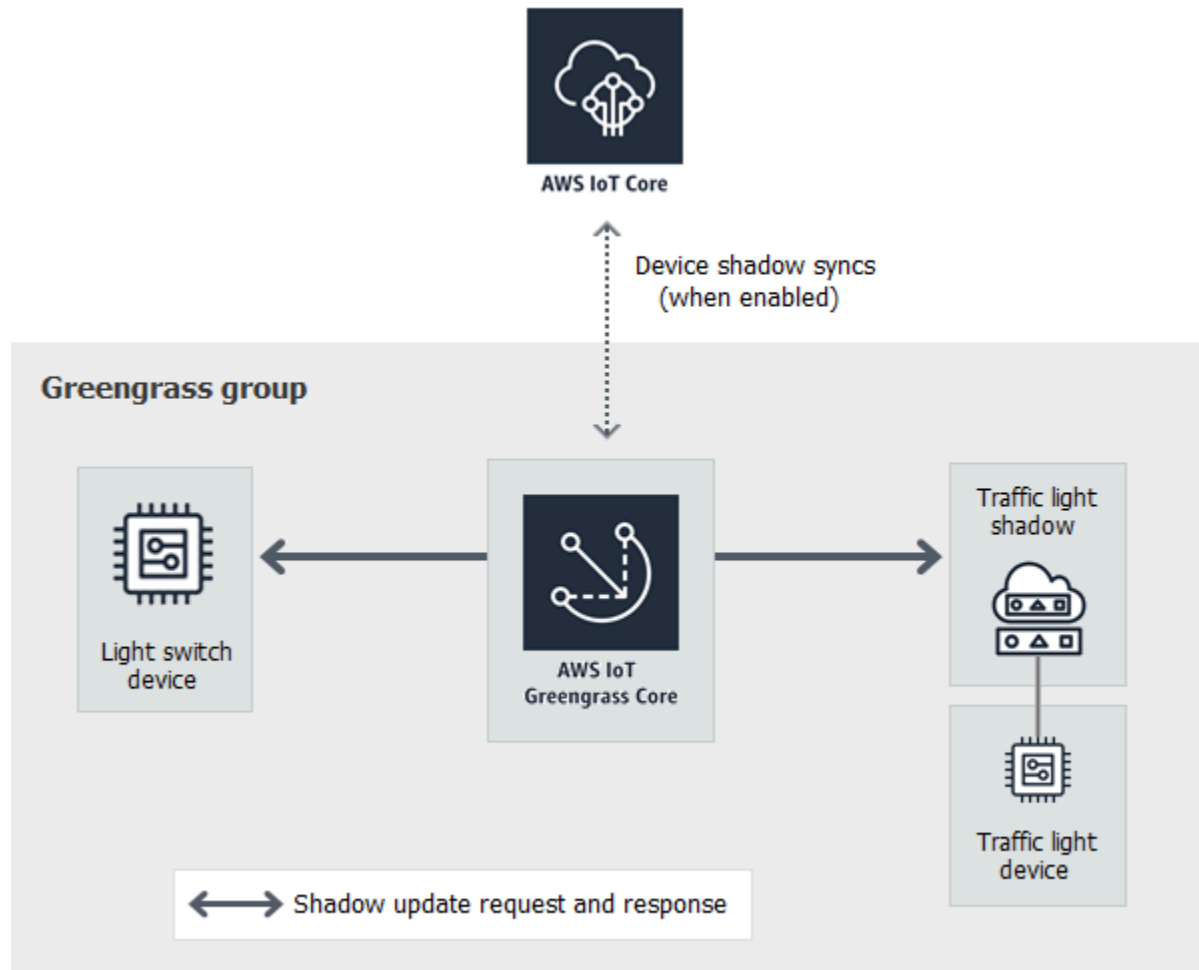
在企業網路上測試可能會影響對核心的連線。若要解決這項問題，您可以手動輸入端點。這可確保basicDiscovery.py指令碼連接到 AWS IoT Greengrass 核心裝置的正確 IP 地址。

手動輸入端點

1. 在 AWS IoT 主控台導覽窗格的管理下，展開 Greengrass 裝置，然後選擇群組 (V1)。
2. 在 Greengrass 群組下，選擇您的群組。
3. 設定核心以手動管理 MQTT 代理程式端點。請執行下列操作：
 - a. 在群組組態頁面上，選擇 Lambda 函數索引標籤。
 - b. 在 System Lambda 函數下，選擇 IP 偵測器，然後選擇編輯。
 - c. 在編輯 IP 偵測器設定中，選擇手動管理 MQTT 代理程式端點，然後選擇儲存。
4. 輸入核心的 MQTT 代理程式端點。請執行下列操作：
 - a. 在概觀下，選擇 Greengrass 核心。
 - b. 在 MQTT 代理程式端點下，選擇管理端點。
 - c. 選擇新增端點，並確定您只有一個端點值。此值必須是 AWS IoT Greengrass 核心裝置連接埠 8883 的 IP 地址端點（例如 192.168.1.4）。
 - d. 選擇更新。

第五單元：與裝置陰影互動

此進階模組會示範用戶端裝置如何與 AWS IoT Greengrass 群組中的[AWS IoT 裝置影子](#)互動。陰影為 JSON 文件用於儲存事物的目前或想要的狀態資訊。在此模組中，您將探索一個用戶端裝置 (GG_Switch) 如何修改另一個用戶端裝置 (GG_TrafficLight) 的狀態，以及如何將這些 AWS IoT Greengrass 狀態同步至雲端：



開始之前，請先執行 [Greengrass 裝置安裝](#) 指令碼，或確定您已完成[單元 1](#) 和[單元 2](#)。您也應該了解如何將用戶端裝置連接至 AWS IoT Greengrass 核心 ([單元 4](#))。您不需要其他的元件或裝置。

此單元需約 30 分鐘完成。

主題

- [設定裝置和訂閱](#)
- [下載必要的檔案](#)
- [測試通訊 \(停用裝置同步\)](#)
- [測試通訊 \(啟用裝置同步\)](#)

設定裝置和訂閱

當 AWS IoT Greengrass 核心連線到網際網路 AWS IoT 時，陰影可以同步到。您在此模組中會先使用本機陰影 (但不同步到雲端)。然後，您啟用雲端同步。

每個用戶端裝置都有自己的影子。如需詳細資訊，請參閱《AWS IoT 開發人員指南》中的[適用於的裝置陰影服務 AWS IoT](#)。

1. 在群組組態頁面上，選擇用戶端裝置索引標籤。
2. 從用戶端裝置索引標籤中，在您的 AWS IoT Greengrass 群組中新增兩個新的用戶端裝置。如需此程序的詳細步驟，請參閱 [the section called “在 AWS IoT Greengrass 群組中建立用戶端裝置”](#)。
 - 命名用戶端裝置 **GG_Switch** 和 **GG_TrafficLight**。
 - 產生並下載兩個用戶端裝置的安全資源。
 - 記下用戶端裝置安全資源檔案名稱中的憑證 ID。您稍後會用到這些值。
3. 在電腦上為這些用戶端裝置的安全登入資料建立資料夾。將憑證和金鑰複製到此資料夾。
4. 確定用戶端裝置設定為使用本機影子，而不是與 同步 AWS 雲端。如果沒有，請選取用戶端裝置，選擇同步陰影，然後選擇停用與雲端的陰影同步。
5. 將下表中的訂閱新增到群組。例如，若要建立第一個訂閱：
 - a. 在群組組態頁面上，選擇訂閱索引標籤，然後選擇新增。
 - b. 針對來源類型，選擇用戶端裝置，然後選擇 GG_Switch。
 - c. 針對目標類型，選擇服務，然後選擇 Local Shadow Service。
 - d. 針對 Topic filter (主題篩選條件)，輸入 **\$aws/things/GG_TrafficLight/shadow/update**。
 - e. 選擇 Create subscription (建立訂閱)。

主題必須完全照表格所示輸入。雖然您可以使用萬用字元來合併部分訂閱，但我們並不建議這麼做。如需詳細資訊，請參閱《AWS IoT 開發人員指南》中的 [Shadow MQTT 主題](#)。

來源	目標	主題	備註
GG_Switch	本機陰影服務	\$aws/things/GG_TrafficLight/shadow/update	GG_Switch 傳送更新請求以更新主題。

來源	目標	主題	備註
本機陰影服務	GG_Switch	\$aws/things/GG_TrafficLight/shadow/update/accepted	GG_Switch 需要知曉更新請求是否被接受。
本機陰影服務	GG_Switch	\$aws/things/GG_TrafficLight/shadow/update/rejected	GG_Switch 需要知曉更新請求是否被拒絕。
GG_TrafficLight	本機陰影服務	\$aws/things/GG_TrafficLight/shadow/update	GG_TrafficLight 傳送其狀態的更新給更新主題。
本機陰影服務	GG_TrafficLight	\$aws/things/GG_TrafficLight/shadow/update/delta	本機陰影服務透過 delta 主題傳送已接收的更新給 GG_TrafficLight。
本機陰影服務	GG_TrafficLight	\$aws/things/GG_TrafficLight/shadow/update/accepted	GG_TrafficLight 需要知曉其狀態更新是否被接受。
本機陰影服務	GG_TrafficLight	\$aws/things/GG_TrafficLight/shadow/update/rejected	GG_TrafficLight 需要知曉其狀態更新是否被拒絕。

新的訂閱會顯示在訂閱索引標籤上。

Note

如需 \$ 字元的相關資訊，請參閱[預留主題](#)。

6. 確定已啟用自動偵測，這樣 Greengrass 核心就能發佈其 IP 地址清單。用戶端裝置會使用此資訊來探索核心。請執行下列操作：
 - a. 在群組組態頁面上，選擇 Lambda 函數索引標籤。
 - b. 在 System Lambda 函數下，選擇 IP 偵測器，然後選擇編輯。

- c. 在編輯 IP 偵測器設定中，選擇自動偵測和覆寫 MQTT 代理程式端點，然後選擇儲存。
7. 請確定 Greengrass 協助程式正在執行，如中所述[部署雲端組態到核心裝置](#)。
8. 在群組組態頁面上，選擇部署。

下載必要的檔案

1. 如果您尚未這麼做，請安裝 AWS IoT Device SDK 適用於 Python 的。如需說明，請參閱[the section called “安裝 AWS IoT Device SDK for Python”](#)中的步驟 1。

此 SDK 供用戶端裝置用來與 AWS IoT 和 AWS IoT Greengrass 核心裝置通訊。

2. 從 GitHub 上的 [TrafficLight](#) 範例資料夾中，將 `lightController.py` 和 `trafficLight.py` 檔案下載到您的電腦。將它們儲存在包含 GG_Switch 和 GG_TrafficLight 用戶端裝置憑證和金鑰的資料夾中。

`lightController.py` 指令碼對應至 GG_Switch 用戶端裝置，而 `trafficLight.py` 指令碼對應至 GG_TrafficLight 用戶端裝置。



- 7aa87aa1cf.cert.pem
- 7aa87aa1cf.private.key
- 7aa87aa1cf.public.key
- a27b261ea9.cert.pem
- a27b261ea9.private.key
- a27b261ea9.public.key
- lightController.py
- root-ca-cert.pem
- trafficLight.py

Note

為了方便起見，Python 檔案範例會存放在適用於 Python 的 AWS IoT Greengrass Core SDK 儲存庫中，但不會使用 AWS IoT Greengrass Core SDK。

測試通訊 (停用裝置同步)

1. 請確定您的電腦和 AWS IoT Greengrass 核心裝置已使用相同的網路連線至網際網路。
 - a. 在 AWS IoT Greengrass 核心裝置上，執行下列命令來尋找其 IP 地址。


```
hostname -I
```

- b. 在您的電腦，使用核心的 IP 地址執行以下命令。您可以使用 Ctrl + C 來停止 ping 命令。

```
ping IP-address
```

類似以下的輸出表示電腦與 AWS IoT Greengrass 核心裝置之間的通訊成功（封包遺失 0%）：

```
$ping 176.32.103.205
PING 176.32.103.205 (176.32.103.205) 56(84) bytes of data.
64 bytes from 176.32.103.205: icmp_seq=1 ttl=230 time=77.2 ms
64 bytes from 176.32.103.205: icmp_seq=2 ttl=230 time=77.1 ms
64 bytes from 176.32.103.205: icmp_seq=3 ttl=230 time=77.1 ms
64 bytes from 176.32.103.205: icmp_seq=4 ttl=230 time=77.1 ms
64 bytes from 176.32.103.205: icmp_seq=5 ttl=230 time=77.1 ms
64 bytes from 176.32.103.205: icmp_seq=6 ttl=230 time=77.1 ms
^C
--- 176.32.103.205 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5549ms
rtt min/avg/max/mdev = 77.107/77.172/77.256/0.361 ms
```

 Note

如果您無法 ping 執行中的 EC2 執行個體 AWS IoT Greengrass，請確定執行個體的傳入安全群組規則允許 [Echo 請求](#) 訊息的 ICMP 流量。如需詳細資訊，請參閱《Amazon EC2 使用者指南》中的 [將規則新增至安全群組](#)。

在 Windows 主機電腦的 Windows 防火牆和進階安全應用程式中，您可能需要啟用允許傳入回聲請求的內送規則（例如，File and Printer Sharing (Echo Request - ICMPv4-In)（檔案和印表機共用（回聲請求 - ICMPv4-In））或建立規則。

2. 取得您的 AWS IoT 端點。

- a. 從 [AWS IoT 主控台](#) 導覽窗格中，選擇設定。
- b. 在裝置資料端點下，記下端點的值。您可以使用這個值，在以下步驟的命令中取代 *AWS_IOT_ENDPOINT* 預留位置。

Note

請確定您的[端點對應至您的憑證類型](#)。

3. 在您的電腦（而非 AWS IoT Greengrass 核心裝置）上，開啟兩個[命令列](#)（終端或命令提示）視窗。一個視窗代表 GG_Switch 用戶端裝置，另一個視窗代表 GG_TrafficLight 用戶端裝置。
 - a. 從 GG_Switch 用戶端裝置視窗中，執行下列命令。
 - 將 *path-to-certs-folder* 替換為包含憑證、金鑰和 Python 檔案的資料夾路徑。
 - 將 *AWS_IOT_ENDPOINT* 取代為您的端點。
 - 將兩個 *switchCertId* 執行個體取代為 GG_Switch 用戶端裝置的檔案名稱中的憑證 ID。

```
cd path-to-certs-folder
python lightController.py --endpoint AWS_IOT_ENDPOINT --rootCA
  AmazonRootCA1.pem --cert switchCertId-certificate.pem.crt --key switchCertId-
  private.pem.key --thingName GG_TrafficLight --clientId GG_Switch
```

- b. 從 GG_TrafficLight 用戶端裝置視窗中，執行下列命令。
 - 將 *path-to-certs-folder* 替換為包含憑證、金鑰和 Python 檔案的資料夾路徑。
 - 將 *AWS_IOT_ENDPOINT* 取代為您的端點。
 - 將兩個 *lightCertId* 執行個體取代為 GG_TrafficLight 用戶端裝置的檔案名稱中的憑證 ID。

```
cd path-to-certs-folder
python trafficLight.py --endpoint AWS_IOT_ENDPOINT --rootCA AmazonRootCA1.pem
  --cert lightCertId-certificate.pem.crt --key lightCertId-private.pem.key --
  thingName GG_TrafficLight --clientId GG_TrafficLight
```

每 20 秒會切換更新陰影狀態為 G、Y 和 R，光源會顯示其新的狀態，如下所示。

GG_Switch 輸出：

```
{
  "state": {
    "desired": {
      "property": "R"
    }
  }
}
2018-12-20 12:23:01,446 - AWSIoTPythonSDK.core.protocol.mqtt_core - INFO - Performing sync publish...
~~~~~Shadow Update Accepted~~~~~
Update request with token: 3b22e27c-930d-4c6a-8562-9f86088249f4 accepted!
property: R
~~~~~
```

GG_TrafficLight 輸出：

```
+++++++ Received Shadow Delta ++++++++
{
  "state": {
    "reported": {
      "property": "R"
    }
  },
  "metadata": {
    "timestamp": 1545337381,
    "version": 33,
    "clientToken": "3b22e27c-930d-4c6a-8562-9f86088249f4"
  }
}
property: R
version: 33
+++++++

Light changed to: R
{
  "state": {
    "reported": {
      "property": "R"
    }
  }
}
2018-12-20 12:23:01,539 - AWSIoTPythonSDK.core.protocol.mqtt_core - INFO - Performing sync publish...
~~~~~Shadow Update Accepted~~~~~
Update request with token: f552109f-c1c2-4ae6-a841-8443506eefcb accepted!
property: R
~~~~~
```

第一次執行時，每個用戶端裝置指令碼都會 AWS IoT Greengrass 執行探索服務以連線到 AWS IoT Greengrass 核心（透過網際網路）。在用戶端裝置發現並成功連接到 AWS IoT Greengrass 核心之後，就可以在本機執行未來的操作。

Note

lightController.py 和 trafficLight.py 指令碼會將連線資訊存放在 groupCA 資料夾，系統會在與指令碼相同的資料夾建立這個資料夾。如果您收到連線錯誤，請確定ggc-host檔案中的 IP 地址符合核心的 IP 地址端點。

4. 在 AWS IoT 主控台中，選擇您的 AWS IoT Greengrass 群組，選擇用戶端裝置索引標籤，然後選擇 GG_TrafficLight 以開啟用戶端裝置的 AWS IoT 物件詳細資訊頁面。
5. 選擇裝置影子索引標籤。在 GG_Switch 變更狀態之後，此影子不應有任何更新。這是因為 GG_TrafficLight 設定為停用與雲端的陰影同步。
6. C 在 GG_Switch (lightController.py) 用戶端裝置視窗中按下 Ctrl +。您應該會看到 GG_TrafficLight (trafficLight.py) 視窗停止接收狀態變更訊息。

將這些視窗保持開啟，讓您可以在下一個區段中執行命令。

測試通訊 (啟用裝置同步)

對於此測試，您將 GG_TrafficLight 裝置陰影設定為與 AWS IoT 同步。您執行的命令與在之前測試中的命令相同，但此次雲端中的陰影狀態會在 GG_Switch 傳送更新請求時更新。

1. 在 AWS IoT 主控台中，選擇您的 AWS IoT Greengrass 群組，然後選擇用戶端裝置索引標籤。
2. 選取 GG_TrafficLight 裝置，選擇同步陰影，然後選擇啟用與雲端的陰影同步。

您應該會收到裝置影子同步狀態已更新的通知。

3. 在群組組態頁面上，選擇部署。
4. 在兩個命令列視窗中，針對 [GG_Switch](#) 和 [GG_TrafficLight](#) 用戶端裝置執行先前測試的命令。
5. 現在，在 AWS IoT 主控台中檢查陰影狀態。選擇您的 AWS IoT Greengrass 群組，選擇用戶端裝置索引標籤，選擇 GG_TrafficLight，選擇裝置影子索引標籤，然後選擇 Classic Shadow。

由於您已啟用 GG_TrafficLight 陰影的同步 AWS IoT，因此每當 GG_Switch 傳送更新時，應該更新雲端中的陰影狀態。此功能可用來公開用戶端裝置的狀態 AWS IoT。

Note

如有必要，您可以透過檢視 AWS IoT Greengrass 核心日誌來疑難排解問題，特別是 runtime.log：

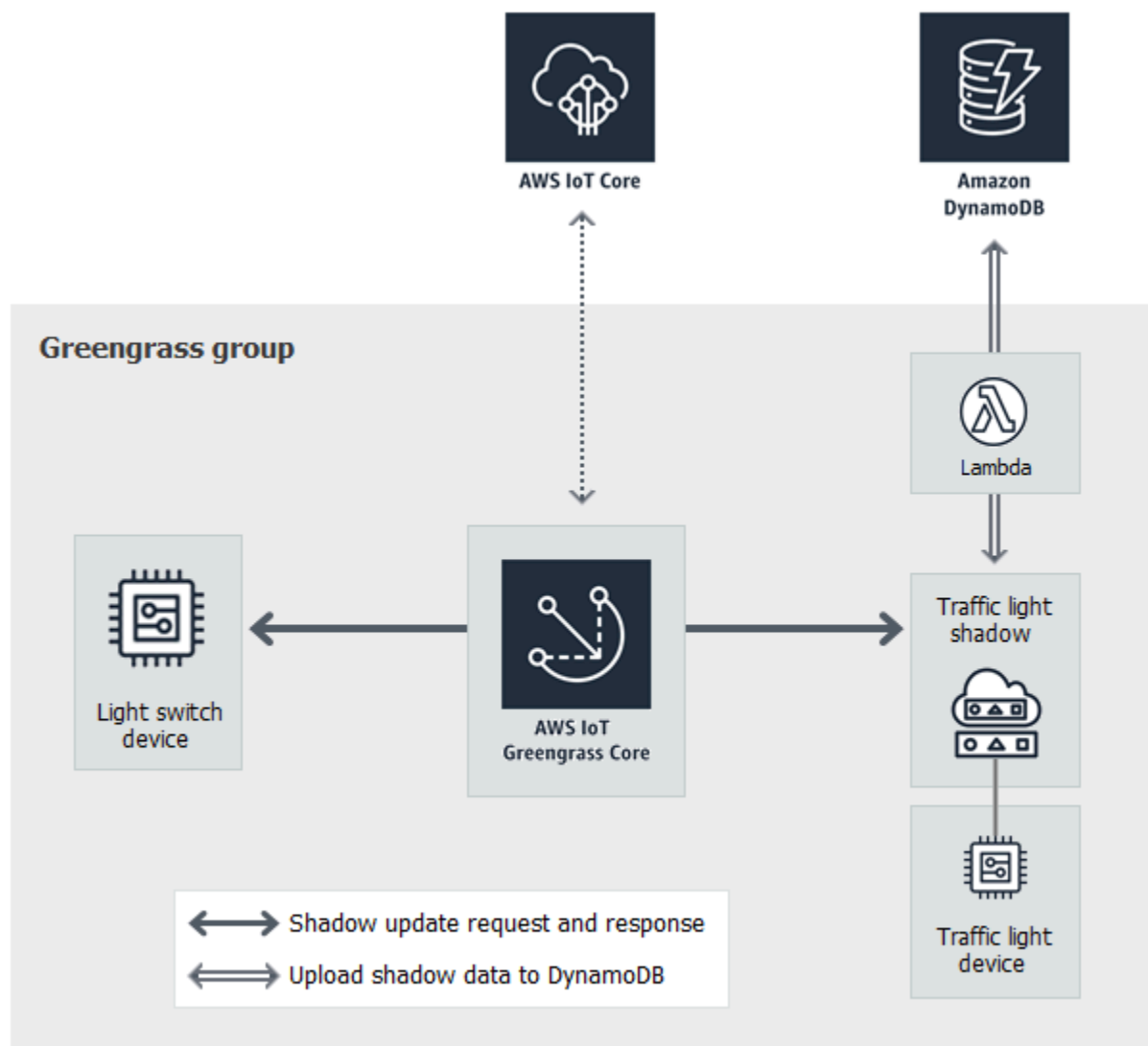
```
cd /greengrass/ggc/var/log
sudo cat system/runtime.log | more
```

您也可以檢視 GGShadowSyncManager.log 和 GGShadowService.log。如需詳細資訊，請參閱[疑難排解](#)。

保持設定用戶端裝置和訂閱。您會在下一個模組中用到它們。您也可以執行相同的命令。

單元 6：存取其他 AWS 服務

此進階模組會示範 AWS IoT Greengrass 核心如何與雲端中的其他 AWS 服務互動。它以[模組 5](#) 的流量光源範例為基礎，並新增 Lambda 函數來處理陰影狀態，並將摘要上傳至 Amazon DynamoDB 資料表。



開始之前，請先執行 [Greengrass 裝置安裝](#) 指令碼，或確定您已完成[單元 1](#) 和[單元 2](#)。你也應該完成[單元 5](#)。您不需要其他的元件或裝置。

此單元需約 30 分鐘完成。

Note

此模組會在 DynamoDB 中建立和更新資料表。雖然大部分操作都很小，且屬於 Amazon Web Services 免費方案，但執行本模組中的一些步驟可能會導致您的帳戶產生費用。如需定價的資訊，請參閱 [DynamoDB 定價文件](#)。

主題

- [設定群組角色](#)
- [建立並設定 Lambda 函式](#)
- [設定訂閱](#)
- [測試通訊](#)

設定群組角色

群組角色是您建立並連接至 Greengrass 群組的 [IAM 角色](#)。此角色包含部署 Lambda 函數（和其他 AWS IoT Greengrass 功能）用於存取 AWS 服務的許可。如需詳細資訊，請參閱[the section called “Greengrass 群組角色”](#)。

您可以使用下列高階步驟，在 IAM 主控台中建立群組角色。

1. 建立允許或拒絕對一或多個資源執行動作的政策。
2. 建立使用 Greengrass 服務做為信任的實體的角色。
3. 將您的政策附加至角色。

然後，在 AWS IoT 主控台中，您將角色新增至 Greengrass 群組。

Note

Greengrass 群組有一個群組角色。如果您要新增權限，可以編輯附加的政策或附加更多政策。

在本教學課程中，您會建立許可政策，允許在 Amazon DynamoDB 資料表上描述、建立和更新動作。接著，您要附加此政策至新的角色，並將該角色與您的 Greengrass 群組相關聯。

首先，建立客戶受管政策，在此模組中授予 Lambda 函數所需的許可。

1. 在 IAM 主控台的導覽窗格中，選擇政策，然後選擇建立政策。
2. 在 JSON 標籤上，用以下政策取代預留位置內容。此模組中的 Lambda 函數使用這些許可來建立和更新名為的 DynamoDB 資料表CarStats。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PermissionsForModule6",
      "Effect": "Allow",
      "Action": [
        "dynamodb:DescribeTable",
        "dynamodb:CreateTable",
        "dynamodb:PutItem"
      ],
      "Resource": "arn:aws:dynamodb:*:*:table/CarStats"
    }
  ]
}
```

3. 選擇 Next: Tags (下一步：標籤)，然後選擇 Next: Review (下一步：檢閱)。本教學課程並未使用任何標籤。
4. 在 Name (名稱) 中輸入 **greengrass_CarStats_Table**，然後選擇 Create policy (建立政策)。

接著，建立一個使用此新政策的角色。

5. 在導覽窗格中，選擇角色，然後選擇建立角色。
6. 在受信任的實體類型下，選擇 AWS 服務。
7. 在使用案例下，其他服務 AWS 的使用案例選擇 Greengrass，選擇 Greengrass，然後選擇下一步。
8. 在許可政策下，選取新**greengrass_CarStats_Table**政策，然後選擇下一步。
9. 在角色名稱中，輸入 **Greengrass_Group_Role**。
10. 對於 Description (說明)，輸入 **Greengrass group role for connectors and user-defined Lambda functions**。
11. 選擇建立角色。

現在，將該角色新增至您的 Greengrass 群組。

12. 在 AWS IoT 主控台導覽窗格的管理下，展開 Greengrass 裝置，然後選擇群組 (V1)。

13. 在 Greengrass 群組下，選擇您的群組。
14. 選擇設定，然後選擇關聯角色。
15. Greengrass_Group_Role 從角色清單中選擇，然後選擇關聯角色。

建立並設定 Lambda 函式

在此步驟中，您會建立 Lambda 函數來追蹤通過交通燈號的汽車數量。每次 GG_TrafficLight 陰影狀態變更為時 G，Lambda 函數都會模擬隨機傳遞的汽車數量（從 1 到 20）。在每第三個 G 光線變更時，Lambda 函數會將基本統計資料傳送至 DynamoDB 資料表，例如最小值和最大值。

1. 在您的電腦上建立名為 car_aggregator 的資料夾。
2. 從 GitHub 上的 [TrafficLight](#) 範例資料夾中，將 carAggregator.py 檔案下載到 car_aggregator 資料夾。這是您的 Lambda 函數程式碼。

Note

為了方便起見，此範例 Python 檔案會存放在 AWS IoT Greengrass 核心 SDK 儲存庫中，但不會使用 AWS IoT Greengrass 核心 SDK。

3. 如果您不是在美國東部（維吉尼亞北部）區域工作，請在下列行 region_name AWS 區域中開啟 carAggregator.py 並變更為目前在 AWS IoT 主控台中選取的。如需支援的清單 AWS 區域，請參閱 [AWS IoT Greengrass](#) 中的 Amazon Web Services 一般參考。

```
dynamodb = boto3.resource('dynamodb', region_name='us-east-1')
```

4. 在 [命令列](#) 視窗中執行下列命令，以在 car_aggregator 資料夾中安裝 [適用於 Python \(Boto3\) 的 AWS SDK](#) 套件及其相依性。Greengrass Lambda 函數使用 AWS SDK 存取其他 AWS 服務。(Windows 平台請使用 [提升權限的命令提示](#))。

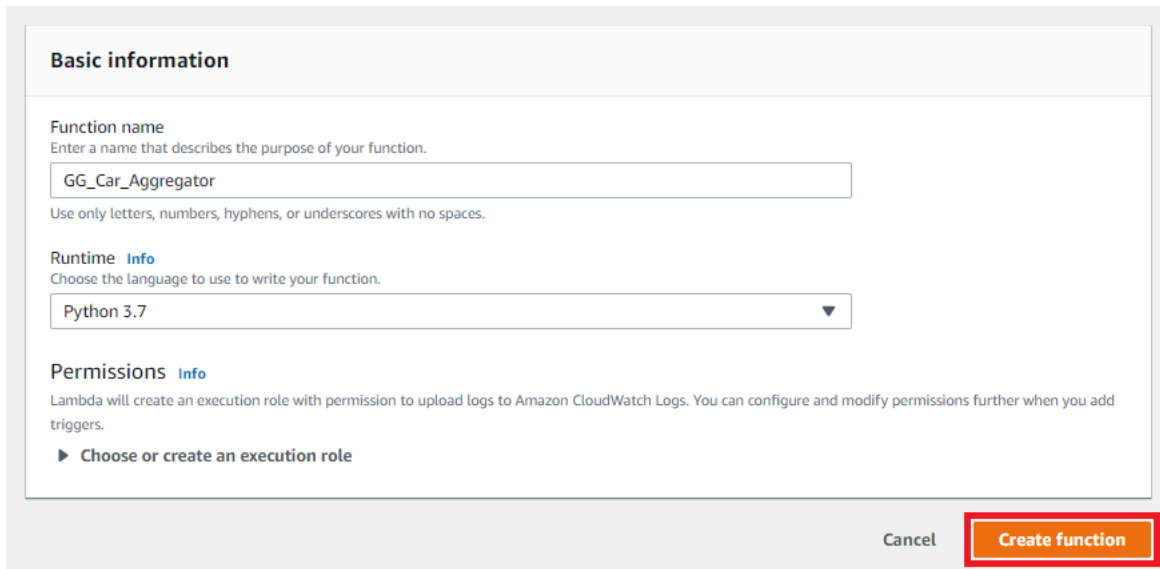
```
pip install boto3 -t path-to-car_aggregator-folder
```

如此會產生類似如下的目錄清單：

Name	Date modified	Type
 bin	12/31/2018 2:27 PM	File folder
 boto3	12/31/2018 2:27 PM	File folder
 boto3-1.9.71.dist-info	12/31/2018 2:27 PM	File folder
 botocore	12/31/2018 2:27 PM	File folder
 botocore-1.12.71.dist-info	12/31/2018 2:27 PM	File folder
 concurrent	12/31/2018 2:27 PM	File folder
 dateutil	12/31/2018 2:27 PM	File folder
 docutils	12/31/2018 2:27 PM	File folder
 docutils-0.14.dist-info	12/31/2018 2:27 PM	File folder
 futures-3.2.0.dist-info	12/31/2018 2:27 PM	File folder
 jmespath	12/31/2018 2:27 PM	File folder
 jmespath-0.9.3.dist-info	12/31/2018 2:27 PM	File folder
 python_dateutil-2.7.5.dist-info	12/31/2018 2:27 PM	File folder
 s3transfer	12/31/2018 2:27 PM	File folder
 s3transfer-0.1.13.dist-info	12/31/2018 2:27 PM	File folder
 six-1.12.0.dist-info	12/31/2018 2:27 PM	File folder
 urllib3	12/31/2018 2:27 PM	File folder
 urllib3-1.24.1.dist-info	12/31/2018 2:27 PM	File folder
 carAggregator.py	12/31/2018 2:25 PM	PY File
 six.py	12/31/2018 2:27 PM	PY File
 six.pyc	12/31/2018 2:27 PM	Compiled Python ...

5. 將 car_aggregator 資料夾的內容壓縮成 .zip 檔案，其名為 car_aggregator.zip。(壓縮資料夾的內容，而非資料夾。) 這是您的 Lambda 函數部署套件。
6. 在 Lambda 主控台中，建立名為 的函數 **GG_Car_Aggregator**，並設定其餘欄位，如下所示：
 - 針對執行期，選擇 Python 3.7。
 - 對於許可，請保留預設設定。這會建立授予基本 Lambda 許可的執行角色。不會使用此角色 AWS IoT Greengrass。

選擇建立函數。



Basic information

Function name
Enter a name that describes the purpose of your function.
GG_Car_Aggregator
Use only letters, numbers, hyphens, or underscores with no spaces.

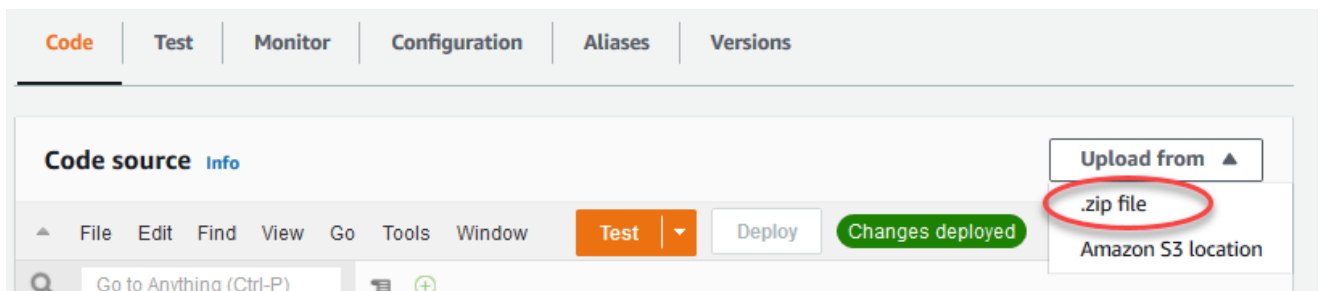
Runtime [Info](#)
Choose the language to use to write your function.
Python 3.7

Permissions [Info](#)
Lambda will create an execution role with permission to upload logs to Amazon CloudWatch Logs. You can configure and modify permissions further when you add triggers.
▶ **Choose or create an execution role**

Cancel **Create function**

7. 上傳您的 Lambda 函數部署套件：

- a. 在程式碼索引標籤的程式碼來源下，選擇上傳來源。從下拉式清單中，選擇 .zip 檔案。



- b. 選擇上傳，然後選擇您的 `car_aggregator.zip` 部署套件。然後選擇 Save (儲存)。
- c. 在函數的程式碼索引標籤的執行時間設定下，選擇編輯，然後輸入下列值。
- 針對執行期，選擇 Python 3.7。
 - 對於 Handler (處理常式)，輸入 `carAggregator.function_handler`。
- d. 選擇儲存。
8. 發佈 Lambda 函數，然後建立名為 `GG_CarAggregator` 的別名。如需 step-by-step 說明，請參閱 [發佈 Lambda 函數](#) 和在模組 3 (第 1 部分) 中 [建立別名](#) 的步驟。
9. 在 AWS IoT 主控台中，將您剛建立的 Lambda 函數新增至您的 AWS IoT Greengrass 群組：
- a. 在群組組態頁面上，選擇 Lambda 函數，然後在我的 Lambda 函數下，選擇新增。
- b. 針對 Lambda 函數，選擇 GG_Car_Aggregator。
- c. 對於 Lambda 函數版本，選擇您發佈的版本別名。
- d. 針對 Memory limit (記憶體限制)，輸入 **64 MB**。

- e. 針對固定，選擇 True。
- f. 選擇新增 Lambda 函數。

Note

您可以從舊版模組中移除其他 Lambda 函數。

設定訂閱

在此步驟中，您會建立訂閱，讓 GG_TrafficLight 影子將更新的狀態資訊傳送至 GG_Car_Aggregator Lambda 函數。此訂閱是您在[模組 5](#) 中建立訂閱的補充，在此模組中為必須。

1. 在群組組態頁面上，選擇訂閱索引標籤，然後選擇新增。
2. 在建立訂閱頁面上，執行下列動作：
 - a. 針對來源類型，選擇服務，然後選擇本機影子服務。
 - b. 針對目標類型，選擇 Lambda 函數，然後選擇 GG_Car_Aggregator。
 - c. 針對 Topic filter (主題篩選條件)，輸入 `$aws/things/GG_TrafficLight/shadow/update/documents`。
 - d. 選擇 Create subscription (建立訂閱)。

此模組需要新訂閱和您在單元 5 中建立的[訂閱](#)。

3. 請確定 Greengrass 協助程式正在執行，如中所述[部署雲端組態到核心裝置](#)。
4. 在群組組態頁面上，選擇部署。

測試通訊

1. 請在電腦上開啟兩個[命令列](#)視窗。如同[模組 5](#) 所示，一個視窗用於 GG_Switch 用戶端裝置，另一個視窗用於 GG_TrafficLight 用戶端裝置。您會使用這些來執行在模組 5 中執行的相同命令。

針對 GG_Switch 用戶端裝置執行下列命令：

```
cd path-to-certs-folder
```

```
python lightController.py --endpoint AWS_IOT_ENDPOINT --rootCA AmazonRootCA1.pem  
--cert switchCertId-certificate.pem.crt --key switchCertId-private.pem.key --  
thingName GG_TrafficLight --clientId GG_Switch
```

針對 GG_TrafficLight 用戶端裝置執行下列命令：

```
cd path-to-certs-folder  
python trafficLight.py --endpoint AWS_IOT_ENDPOINT --rootCA AmazonRootCA1.pem --  
cert lightCertId-certificate.pem.crt --key lightCertId-private.pem.key --thingName  
GG_TrafficLight --clientId GG_TrafficLight
```

每 20 秒會切換更新陰影狀態為 G、Y 和 R，光源會顯示其新的狀態。

2. Lambda 函數的函數處理常式會在每第三個綠燈（每三分鐘）觸發，並建立新的 DynamoDB 記錄。在 trafficLight.py lightController.py 和執行三分鐘後，前往 AWS 管理主控台，然後開啟 DynamoDB 主控台。
3. 在 AWS 區域功能表中選擇美國東部（維吉尼亞北部）。這是 GG_Car_Aggregator 函數建立表格所在的區域。
4. 在導覽窗格中，選擇 Tables (資料表)，然後選擇 CarStats 資料表。
5. 選擇檢視項目以檢視資料表中的項目。

您應該會看到一個具有基本統計資料的項目，在 cars 通過上 (每隔 3 分鐘)。您可能需要選擇重新整理按鈕來檢視資料表更新。

6. 如果測試不成功，您可以在 Greengrass 日誌中尋找故障診斷資訊。
 - a. 切換到根使用者和導覽至 log 目錄。存取 AWS IoT Greengrass 日誌需要根許可。

```
sudo su  
cd /greengrass/ggc/var/log
```

- b. 檢查 runtime.log 是否有錯誤。

```
cat system/runtime.log | grep 'ERROR'
```

- c. 檢查 Lambda 函數產生的日誌。

```
cat user/region/account-id/GG_Car_Aggregator.log
```

lightController.py 和 trafficLight.py 指令碼會將連線資訊存放在 groupCA 資料夾，系統會在與指令碼相同的資料夾建立這個資料夾。如果您收到連線錯誤，請確定ggc-host檔案中的 IP 地址符合核心的 IP 地址端點。

如需詳細資訊，請參閱[疑難排解](#)。

基本教學課程至此已到尾聲。您現在應該了解 AWS IoT Greengrass 程式設計模型及其基本概念，包括 AWS IoT Greengrass 核心、群組、訂閱、用戶端裝置，以及在邊緣執行之 Lambda 函數的部署程序。

您可以刪除 DynamoDB 資料表和 Greengrass Lambda 函數和訂閱。若要停止 AWS IoT Greengrass 核心裝置與 AWS IoT 雲端之間的通訊，請在核心裝置上開啟終端機，並執行下列其中一個命令：

- 若要關閉 AWS IoT Greengrass 核心裝置：

```
sudo halt
```

- 若要停止 AWS IoT Greengrass 協助程式：

```
cd /greengrass/ggc/core/  
sudo ./greengrassd stop
```

單元 7：模擬硬體安全整合

此功能適用於 AWS IoT Greengrass Core v1.7 和更新版本。

這個進階單元會示範如何設定模擬的硬體安全模組 (HSM)，以與 Greengrass 核心搭配使用。組態會使用 SoftHSM，即為使用 [PKCS#11](#) 應用程式設計界面 (API) 的純軟體實作。本單元的目的是讓您設定環境，以在其中學習，並對 PKCS#11 API 的純軟體實作進行初始測試。僅供學習和初始測試之用，不可用於任何類型的正式生產。

您可以使用此組態，以實驗使用 PKCS#11 相容服務來存放您的私有金鑰。如需純軟體實作的詳細資訊，請參閱 [SoftHSM](#)。如需在 AWS IoT Greengrass 核心上整合硬體安全性的詳細資訊，包括一般需求，請參閱 [the section called “硬體安全整合”](#)。

Important

本單元僅供實驗之用。我們強烈建議您不要在生產環境中使用 SoftHSM，因為這可能會產生額外安全的錯覺。產生的組態並不會提供任何實際的安全優勢。在 SoftHSM 存放金鑰此一方法，不會比 Greengrass 環境中私密儲存的任何其他方法更安全。

本單元的目的是要讓您了解 PKCS#11 規格，並在您計劃未來使用真正的硬體型 HSM 時，執行軟體的初始測試。

您必須在正式生產前，對您未來的硬體實作進行個別、完整的測試，因為 SoftHSM 中提供的 PKCS#11 實作與硬體型實作之間可能有差異。

如果您需要協助加入[支援的硬體安全模組](#)，請聯絡您的 AWS 企業支援代表。

開始之前，請先執行 [Greengrass 裝置安裝](#) 指令碼，或確定您已完成入門教學課程的[單元 1](#) 和[單元 2](#)。在本單元中，我們假設您的核心已佈建並與之通訊 AWS。此單元需約 30 分鐘完成。

安裝 SoftHSM 軟體

在此步驟中，您會安裝 SoftHSM 和 pkcs11 工具，以用來管理您的 SoftHSM 執行個體。

- 在 AWS IoT Greengrass 核心裝置上的終端機中，執行下列命令：

```
sudo apt-get install softhsm2 libsofthsm2-dev pkcs11-dump
```

如需這些套件的詳細資訊，請參閱[安裝 softhsm2](#)、[安裝 libsofthsm2-dev](#) 和[安裝 pkcs11-dump](#)。

Note

在系統上使用此命令時，如果遇到問題，請參閱 GitHub 上的 [SoftHSM 第 2 版](#)。此網站提供更多安裝資訊，包括如何從來源建置。

設定 SoftHSM

在此步驟中，您要[設定 SoftHSM](#)。

1. 切換至根使用者。

```
sudo su
```

2. 使用手冊頁面尋找全系統的 `softhsm2.conf` 位置。常見的位置為 `/etc/softhsm/softhsm2.conf`，但在某些系統上的位置可能會不同。

```
man softhsm2.conf
```

3. 在全系統位置中為 `softhsm2` 組態檔案建立目錄。此範例假設位置為 `/etc/softhsm/softhsm2.conf`。

```
mkdir -p /etc/softhsm
```

4. 在 `/greengrass` 目錄中建立字符目錄。

 Note

如果略過此步驟，`softhsm2-util` 會報告 `ERROR: Could not initialize the library`。

```
mkdir -p /greengrass/softhsm2/tokens
```

5. 設定字符目錄。

```
echo "directories.tokenidir = /greengrass/softhsm2/tokens" > /etc/softhsm/softhsm2.conf
```

6. 設定檔案型後端。

```
echo "objectstore.backend = file" >> /etc/softhsm/softhsm2.conf
```

 Note

這些組態設定僅供實驗用。若要查看所有組態選項，請參閱組態檔的手冊頁面。

```
man softhsm2.conf
```

將私有金鑰匯入至 SoftHSM

在此步驟中，您要初始化 SoftHSM 字符、轉換私有金鑰格式，然後匯入私有金鑰。

1. 初始化 SoftHSM 字符。

```
softhsm2-util --init-token --slot 0 --label greengrass --so-pin 12345 --pin 1234
```

Note

出現提示時，請輸入 12345 的 SO PIN 和 1234 的使用者 PIN。AWS IoT Greengrass 不會使用 SO (監督員) PIN，所以您可以使用任何值。

如果您收到錯誤訊息 CKR_SLOT_ID_INVALID: Slot 0 does not exist，請改用下列命令：

```
softhsm2-util --init-token --free --label greengrass --so-pin 12345 --pin 1234
```

2. 將私有金鑰轉換為 SoftHSM 匯入工具可以使用的格式。在本教學課程中，您會轉換從入門教學[課程單元 2](#) 中的預設群組建立選項取得的私有金鑰。

```
openssl pkcs8 -topk8 -inform PEM -outform PEM -nocrypt -in hash.private.key -out hash.private.pem
```

3. 將私有金鑰匯入至 SoftHSM。根據您的 softhsm2-util 版本，僅執行以下其中一個命令。

Raspbian softhsm2-util v2.2.0 語法

```
softhsm2-util --import hash.private.pem --token greengrass --label iotkey --id 0000 --pin 12340
```

Ubuntu softhsm2-util v2.0.0 語法

```
softhsm2-util --import hash.private.pem --slot 0 --label iotkey --id 0000 --pin 1234
```

此命令會將插槽識別為 0，並將金鑰標籤定義為 iotkey。您會在下一節中使用這些值。

匯入私有金鑰之後，您可以選擇將其從 `/greengrass/certs` 目錄中移除。務必將根 CA 和裝置憑證保留在目錄中。

設定 Greengrass 核心以使用 SoftHSM

在此步驟中，您會修改 Greengrass 核心組態檔以使用 SoftHSM。

1. 在您的系統上尋找 SoftHSM 供應商程式庫 (`libsofthsm2.so`) 的路徑：

- a. 取得程式庫的已安裝套件清單。

```
sudo dpkg -L libsofthsm2
```

`libsofthsm2.so` 檔案位於 `softhsm` 目錄中。

- b. 複製檔案的完整路徑 (例如 `/usr/lib/x86_64-linux-gnu/softhsm/libsofthsm2.so`)。您稍後會使用此值。

2. 停止 Greengrass 協助程式。

```
cd /greengrass/ggc/core/  
sudo ./greengrassd stop
```

3. 開啟 Greengrass 組態檔。這是 `/greengrass/config` 目錄中 [config.json](#) 的檔案。

Note

此程序中的範例編寫時假設 `config.json` 檔案使用從入門教學 [課程單元 2](#) 中的預設群組建立選項產生的格式。

4. 在 `crypto.principals` 物件中，插入下列 MQTT 伺服器憑證物件。視需要插入逗號以建立有效的 JSON 檔案。

```
"MQTTServerCertificate": {  
  "privateKeyPath": "path-to-private-key"  
}
```

5. 在 `crypto` 物件中，插入下列 PKCS11 物件。視需要插入逗號以建立有效的 JSON 檔案。

```
"PKCS11": {  
  "P11Provider": "/path-to-pkcs11-provider-so",  
  "slotLabel": "crypto-token-name",
```



```
"slotUserPin": "crypto-token-user-pin"
}
```

您的檔案應會如下所示：

```
{
  "coreThing" : {
    "caPath" : "root.ca.pem",
    "certPath" : "hash.cert.pem",
    "keyPath" : "hash.private.key",
    "thingArn" : "arn:partition:iot:region:account-id:thing/core-thing-name",
    "iotHost" : "host-prefix.iot.region.amazonaws.com",
    "ggHost" : "greengrass.iot.region.amazonaws.com",
    "keepAlive" : 600
  },
  "runtime" : {
    "cgroup" : {
      "useSystemd" : "yes"
    }
  },
  "managedRespawn" : false,
  "crypto": {
    "PKCS11": {
      "P11Provider": "/path-to-pkcs11-provider-so",
      "slotLabel": "crypto-token-name",
      "slotUserPin": "crypto-token-user-pin"
    },
    "principals" : {
      "MQTTServerCertificate": {
        "privateKeyPath": "path-to-private-key"
      },
      "IoTCertificate" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key",
        "certificatePath" : "file:///greengrass/certs/hash.cert.pem"
      },
      "SecretsManager" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key"
      }
    },
    "caPath" : "file:///greengrass/certs/root.ca.pem"
  }
}
```

Note

若要使用具硬體安全的無線 (OTA) 更新，PKCS11 物件必須包含 OpenSSLEngine 屬性。如需詳細資訊，請參閱[the section called “設定 OTA 更新”](#)。

6. 編輯 crypto 物件：**a. 設定 PKCS11 物件。**

- 針對 P11Provider，輸入 libsofthsm2.so 的完整路徑。
- 針對 slotLabel，請輸入 greengrass。
- 針對 slotUserPin，請輸入 1234。

b. 在 principals 物件中設定私有金鑰路徑。請勿編輯 certificatePath 屬性。

- 針對 privateKeyPath 屬性，請輸入下列 RFC 7512 PKCS#11 路徑 (指定金鑰的標籤)。對 IoTCertificate、SecretsManager 和 MQTTServerCertificate 委託人執行此動作。

```
pkcs11:object=iotkey;type=private
```

c. 檢查 crypto 物件。其看起來與下列類似：

```
"crypto": {
  "PKCS11": {
    "P11Provider": "/usr/lib/x86_64-linux-gnu/softhsm/libsofthsm2.so",
    "slotLabel": "greengrass",
    "slotUserPin": "1234"
  },
  "principals": {
    "MQTTServerCertificate": {
      "privateKeyPath": "pkcs11:object=iotkey;type=private"
    },
    "SecretsManager": {
      "privateKeyPath": "pkcs11:object=iotkey;type=private"
    },
    "IoTCertificate": {
      "certificatePath": "file://certs/core.crt",
      "privateKeyPath": "pkcs11:object=iotkey;type=private"
    }
  }
},
```

```
"caPath": "file://certs/root.ca.pem"
}
```

7. 從 `coreThing` 物件移除 `caPath`、`certPath` 和 `keyPath` 值。其看起來與下列類似：

```
"coreThing" : {
  "thingArn" : "arn:partition:iot:region:account-id:thing/core-thing-name",
  "iotHost" : "host-prefix-ats.iot.region.amazonaws.com",
  "ggHost" : "greengrass-ats.iot.region.amazonaws.com",
  "keepAlive" : 600
}
```

Note

在本教學課程中，您會對所有委託人指定相同的私有金鑰。如需選擇本機 MQTT 伺服器之私有金鑰的詳細資訊，請參閱[效能](#)。如需本機 Secrets Manager 的詳細資訊，請參閱[將私密部署至核心](#)。

測試組態

- 啟動 Greengrass 協助程式。

```
cd /greengrass/ggc/core/
sudo ./greengrassd start
```

如果協助程式啟動成功，則表示您的核心已正確設定。

您現在可以開始了解 PKCS#11 規格，並使用 SoftHSM 實作所提供的 PKCS#11 API 執行初始測試。

Important

再次提醒您，本單元僅供學習和測試之用，這點非常重要。這其實不會提高 Greengrass 環境的安全狀態。

反之，單元的目的是讓您可以開始學習和測試，以準備將來使用真正的硬體型 HSM。屆時，在正式生產之前，您必須針對硬體型 HSM 個別且完整地測試您的軟體，因為相較於硬體型實作，SoftHSM 中提供的 PKCS#11 實作之間可能有差異。

另請參閱

- PKCS #11 Cryptographic Token Interface Usage Guide 2.40 版。編輯者為 John Leiseboer 和 Robert Griffin。2014 年 11 月 16 日。OASIS Committee Note 02。 <http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/cn02/pkcs11-ug-v2.40-cn02.html>。最新版本： <http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/pkcs11-ug-v2.40.html>。
- [RFC 7512](#)

AWS IoT Greengrass Core 軟體的 OTA 更新

Core AWS IoT Greengrass 軟體套件包含更新代理程式，可執行 AWS IoT Greengrass 軟體的 over-the-air(OTA) 更新。您可以使用 OTA 更新，在一或多個核心上安裝最新版本的核心 AWS IoT Greengrass 軟體或 OTA 更新代理程式軟體。透過 OTA 更新，您的核心裝置就不需要實際存在。

我們建議您盡可能使用 OTA 更新。它們提供了一種機制，讓您用來追蹤更新狀態和更新歷程記錄。如果發生更新失敗，OTA 更新代理程式會轉返至先前的軟體版本。

Note

當您使用 安裝 AWS IoT Greengrass Core 軟體時，不支援 OTA apt 更新。針對這些安裝，我們建議您使用 apt 以升級軟體。如需詳細資訊，請參閱[the section called “從 APT 儲存庫安裝”](#)。

OTA 更新可讓您更有效率地執行下列作業：

- 修復安全漏洞。
- 解決軟體穩定性問題。
- 部署新功能和改良的功能。

此功能與 [AWS IoT 任務整合](#)。

要求

下列需求適用於 AWS IoT Greengrass 軟體的 OTA 更新。

- Greengrass 核心在本機儲存中必須至少有 400 MB 的可用磁碟空間。OTA 更新代理程式需要大約 AWS IoT Greengrass Core 軟體執行時間用量需求的三倍。如需詳細資訊，請參閱《》中的 Greengrass 核心的[服務配額](#)Amazon Web Services 一般參考。
- Greengrass 核心必須具有與的連線 AWS 雲端。
- 必須使用憑證和金鑰正確設定和佈建 Greengrass 核心，以便使用 AWS IoT Core 和 進行身分驗證 AWS IoT Greengrass。如需詳細資訊，請參閱[the section called “X.509 憑證”](#)。
- Greengrass 核心無法設定為使用網路代理。

Note

從 v1 AWS IoT Greengrass .9.3 開始，將 MQTT 流量設定為使用連接埠 443 而非預設連接埠 8883 的核心支援 OTA 更新。不過，OTA 更新代理程式不支援透過網路代理進行更新。如需詳細資訊，請參閱[the section called “連線至連接埠 443 或透過網路代理”](#)。

- 無法在包含 AWS IoT Greengrass Core 軟體的分割區中啟用信任的開機。

Note

您可以在已啟用信任開機的分割區上安裝和執行 AWS IoT Greengrass Core 軟體，但不支援 OTA 更新。

- AWS IoT Greengrass 必須在包含 Core 軟體的 AWS IoT Greengrass 分割區上具有讀取/寫入許可。
- 如果您使用初始化系統來管理您的 Greengrass 核心，則必須設定 OTA 更新以與初始化系統整合。如需詳細資訊，請參閱[the section called “與初始化系統整合”](#)。
- 您必須建立用來預先簽署 Amazon S3 URLs 以 AWS IoT Greengrass 軟體更新成品的角色。此簽署者角色允許代表您 AWS IoT Core 存取存放在 Amazon S3 中的軟體更新成品。如需詳細資訊，請參閱[the section called “OTA 更新的 IAM 許可”](#)。

OTA 更新的 IAM 許可

當 AWS IoT Greengrass 發行新版本的 AWS IoT Greengrass 核心軟體時，會 AWS IoT Greengrass 更新存放在 Amazon S3 中用於 OTA 更新的軟體成品。

您的 AWS 帳戶 必須包含可用於存取這些成品的 Amazon S3 URL 簽署者角色。角色必須具有許可政策，允許對目標 中的儲存貯體執行 AWS 區域s3:GetObject動作。該角色也必須具有允許 `iot.amazonaws.com` 將角色視為信任實體的信任政策。

許可政策

對於角色許可，您可以使用 AWS 受管政策或建立自訂政策。

- 使用 AWS 受管政策

[GreengrassOTAUpdateArtifactAccess](#) 受管政策是由 AWS IoT Greengrass提供。如果您想要允許在目前和未來支援的所有 Amazon Web Services 區域中存取 AWS IoT Greengrass，請使用此政策。

- 建立自訂政策

如果您想要明確指定核心部署所在的 Amazon Web Services 區域，您應該建立自訂政策。下列範例政策允許存取六個區域中的 AWS IoT Greengrass 軟體更新。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowAccessToGreengrassOTAUpdateArtifacts",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3:::us-east-1-greengrass-updates/*",
        "arn:aws:s3:::us-west-2-greengrass-updates/*",
        "arn:aws:s3:::ap-northeast-1-greengrass-updates/*",
        "arn:aws:s3:::ap-southeast-2-greengrass-updates/*",
        "arn:aws:s3:::eu-central-1-greengrass-updates/*",
        "arn:aws:s3:::eu-west-1-greengrass-updates/*"
      ]
    }
  ]
}
```

信任政策

連接至角色的信任政策必須允許 `sts:AssumeRole` 動作並定義 `iot.amazonaws.com` 為委託人。這允許 AWS IoT Core 將角色擔任信任的實體。以下是範例政策文件：

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowIotToAssumeRole",
      "Action": "sts:AssumeRole",
      "Principal": {
        "Service": "iot.amazonaws.com"
      },
    }
  ]
}
```

```

        "Effect": "Allow"
      }
    ]
  }

```

此外，起始 OTA 更新的使用者必須具有 `greengrass:CreateSoftwareUpdateJob` 和 `iot:CreateJob` 的使用許可，以及使用 `iam:PassRole` 傳遞簽署者角色的許可。以下是 IAM 政策範例：

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "greengrass:CreateSoftwareUpdateJob"
      ],
      "Effect": "Allow",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:CreateJob"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": "arn:aws:iam::123456789012:role/name-of-s3-url-signer-  
role"
    }
  ]
}

```


考量事項

啟動 Greengrass 核心軟體的 OTA 更新前，請注意更新會對 Greengrass 群組中的裝置造成的影響，不論群組是在核心裝置上還是在本機連線至該核心的用戶端裝置上：

- 更新期間會關閉核心。
- 會關閉在核心執行的任何 Lambda 函數。如果這些函數寫入到本機資源，這些函數可能會讓這些資源處於不正確的狀態，除非有正確地關閉。
- 在核心停機時間期間，其與的所有連線 AWS 雲端 都會遺失。由用戶端裝置透過核心路由傳送的訊息將會遺失。
- 登入資料快取會遺失。
- 佇列為 Lambda 函數所保留待處理的工作會遺失。
- 長期 Lambda 函數會遺失其動態狀態資訊，並捨棄所有待定工作。

在 OTA 更新期間，以下狀態資訊皆會保留：

- 核心組態
- Greengrass 群組組態
- 本機陰影
- Greengrass 日誌
- OTA 更新代理程式日誌

Greengrass OTA 更新代理程式

Greengrass OTA 更新代理程式是裝置上的軟體元件，可處理在雲端中建立和部署的更新任務。OTA 更新代理程式會分佈在與 AWS IoT Greengrass Core 軟體相同的軟體套件中。代理程式位於 `/greengrass-root/ota/ota_agent/ggc-ota` 中。它會將日誌寫入 `/var/log/greengrass/ota/ggc_ota.txt`。

Note

`greengrass-root` 代表在裝置上安裝 AWS IoT Greengrass Core 軟體的路徑。通常，這是 `/greengrass` 目錄。

您可以透過手動執行二進位檔，或將其整合為初始化指令碼的一部分，例如系統化服務檔案，來啟動 OTA 更新代理程式。如果您手動執行二進位檔，它應該做為根執行。啟動時，OTA 更新代理程式會接聽來自的 AWS IoT Greengrass 軟體更新任務，AWS IoT Core 並依序執行這些任務。OTA 更新代理程式會忽略所有其他 AWS IoT 任務類型。

以下摘錄顯示啟動、停止和重新啟動 OTA 更新代理程式的系統化服務檔案範例：

```
[Unit]
Description=Greengrass OTA Daemon

[Service]
Type=forking
Restart=on-failure
ExecStart=/greengrass/ota/ota_agent/ggc-ota

[Install]
WantedBy=multi-user.target
```

作為更新目標的核心不得執行兩個 OTA 更新代理程式執行個體。這麼做會讓兩個代理程式處理相同的任務，而造成衝突。

與初始化系統整合

在 OTA 更新期間，OTA 更新代理程式會在核心裝置上重新啟動二進位檔。如果二進位檔正在執行，當初始化系統在更新期間監控 AWS IoT Greengrass 核心軟體或代理程式的狀態時，這可能會導致衝突。為了協助整合 OTA 更新機制與監控策略，您可以撰寫在更新前後執行的 shell 指令碼。例如，您可以使用 `ggc_pre_update.sh` 指令碼在裝置關閉之前備份資料或停止程序。

若要告知 OTA 更新代理程式執行這些指令碼，您必須在 [config.json](#) 檔案中包含 `"managedRespawn" : true` 旗標。此設定會顯示在下列摘錄中：

```
{
  "coreThing": {
    ...
  },
  "runtime": {
    ...
  },
  "managedRespawn": true
  ...
}
```

透過 OTA 更新受管的 respawn

下列需求適用於將 `managedRespawn` 設定為 `true` 的 OTA 更新：

- `/greengrass-root/usr/scripts` 目錄中必須存在下列 shell 指令碼：
 - `ggc_pre_update.sh`
 - `ggc_post_update.sh`
 - `ota_pre_update.sh`
 - `ota_post_update.sh`
- 指令碼必須傳回成功的傳回碼。
- 這些指令碼必須由根目錄擁有，而且只能由根目錄執行。
- `ggc_pre_update.sh` 指令碼必須停止 Greengrass 協助程式。
- `ggc_post_update.sh` 指令碼必須啟動 Greengrass 協助程式。

Note

由於 OTA 更新代理程式會管理自己的程序，因此 `ota_pre_update.sh` 和 `ota_post_update.sh` 指令碼不需要停止或啟動 OTA 服務。

OTA 更新代理程式會從 執行指令碼 `/greengrass-root/usr/scripts`。目錄樹狀圖看起來應該如下所示：

```
<greengrass_root>
|-- certs
|-- config
|   |-- config.json
|-- ggc
|-- usr/scripts
|   |-- ggc_pre_update.sh
|   |-- ggc_post_update.sh
|   |-- ota_pre_update.sh
|   |-- ota_post_update.sh
|-- ota
```

當 `managedRespawn` 設定為 `true` 時，OTA 更新代理程式會在軟體更新前後檢查這些指令碼的 `/greengrass-root/usr/scripts` 目錄。如果指令碼不存在，更新會失敗。AWS IoT

Greengrass 不會驗證這些指令碼的內容。最佳實務是驗證您的指令碼是否正常運作，並針對錯誤發出適當的結束代碼。

針對 AWS IoT Greengrass Core 軟體的 OTA 更新：

- 在開始更新之前，代理程式會執行 `ggc_pre_update.sh` 指令碼。將此指令碼用於在 OTA 更新代理程式啟動 AWS IoT Greengrass 核心軟體更新之前需要執行的命令，例如備份資料或停止任何執行中的程序。下列範例顯示停止 Greengrass 協助程式的簡單指令碼。

```
#!/bin/bash
set -euo pipefail
systemctl stop greengrass
```

- 完成更新後，代理程式會執行 `ggc_post_update.sh` 指令碼。將此指令碼用於在 OTA 更新代理程式啟動 AWS IoT Greengrass 核心軟體更新後需要執行的命令，例如重新啟動程序。下列範例顯示啟動 Greengrass 協助程式的簡單指令碼。

```
#!/bin/bash
set -euo pipefail
systemctl start greengrass
```

對於 OTA 更新代理程式的 OTA 更新：

- 在開始更新之前，代理程式會執行 `ota_pre_update.sh` 指令碼。將此指令碼用於需要在 OTA 更新代理程式自行更新之前執行的命令，例如備份資料或停止任何執行中的程序。
- 完成更新後，代理程式會執行 `ota_post_update.sh` 指令碼。將此指令碼用於在 OTA 更新代理程式更新之後需要執行的命令，例如重新啟動程序。

Note

如果 `managedRespawn` 設定為 `false`，OTA 更新代理程式不會執行指令碼。

建立 OTA 更新

請依照下列步驟，在一或多個核心上執行 AWS IoT Greengrass 軟體的 OTA 更新：

1. 確定您的核心符合 OTA 更新的[要求](#)。

Note

如果您已設定初始化系統來管理 AWS IoT Greengrass 核心軟體或 OTA 更新代理程式，請在您的核心上驗證下列項目：

- [config.json](#) 檔案會指定 "managedRespawn" : true。
- `/greengrass-root/usr/scripts` 目錄包含下列指令碼：
 - `ggc_pre_update.sh`
 - `ggc_post_update.sh`
 - `ota_pre_update.sh`
 - `ota_post_update.sh`

如需詳細資訊，請參閱[the section called “與初始化系統整合”](#)。

2. 在核心裝置終端機中，啟動 OTA 更新代理程式。

```
cd /greengrass-root/ota/ota_agent  
sudo ./ggc-ota
```

Note

`greengrass-root` 代表在裝置上安裝 AWS IoT Greengrass Core 軟體的路徑。通常，這是 `/greengrass` 目錄。

請勿在核心上啟動多個 OTA 更新代理程式執行個體，因為它可能會導致衝突。

3. 使用 AWS IoT Greengrass API 建立軟體更新任務。

- a. 呼叫 [CreateSoftwareUpdateJob](#) API。在此範例程序中，我們使用 AWS CLI 命令。

下列命令會建立任務，以更新一個核心上的 AWS IoT Greengrass Core 軟體。取代範例值，然後執行命令。

Linux or macOS terminal

```
aws greengrass create-software-update-job \
```

```
--update-targets-architecture x86_64 \  
--update-targets ["arn:aws:iot:region:123456789012:thing/myCoreDevice\""] \  
--update-targets-operating-system ubuntu \  
--software-to-update core \  
--s3-url-signer-role arn:aws:iam::123456789012:role/myS3UrlSignerRole \  
--update-agent-log-level WARN \  
--amzn-client-token myClientToken1
```

Windows command prompt

```
aws greengrass create-software-update-job ^  
--update-targets-architecture x86_64 ^  
--update-targets ["arn:aws:iot:region:123456789012:thing/myCoreDevice\""] ^  
--update-targets-operating-system ubuntu ^  
--software-to-update core ^  
--s3-url-signer-role arn:aws:iam::123456789012:role/myS3UrlSignerRole ^  
--update-agent-log-level WARN ^  
--amzn-client-token myClientToken1
```

該命令會傳回下列回應。

```
{  
  "IotJobId": "GreengrassUpdateJob_c3bd7f36-ee80-4d42-8321-a1da0EXAMPLE",  
  "IotJobArn": "arn:aws:iot:region:123456789012:job/  
GreengrassUpdateJob_c3bd7f36-ee80-4d42-8321-a1da0EXAMPLE",  
  "PlatformSoftwareVersion": "1.10.1"  
}
```

- b. 從回應複製 IoTJobId。
- c. 在 AWS IoT Core API 中呼叫 [DescribeJob](#) 以查看任務狀態。以您的任務 ID 取代範例值，然後執行命令。

```
aws iot describe-job --job-id GreengrassUpdateJob_c3bd7f36-ee80-4d42-8321-a1da0EXAMPLE
```

此命令會傳回包含任務相關資訊的回應物件，包括 status 和 jobProcessDetails。

```
{  
  "job": {
```

```

    "jobArn": "arn:aws:iot:region:123456789012:job/
GreengrassUpdateJob_c3bd7f36-ee80-4d42-8321-a1da0EXAMPLE",
    "jobId": "GreengrassUpdateJob_c3bd7f36-ee80-4d42-8321-a1da0EXAMPLE",
    "targetSelection": "SNAPSHOT",
    "status": "IN_PROGRESS",
    "targets": [
        "arn:aws:iot:region:123456789012:thing/myCoreDevice"
    ],
    "description": "This job was created by Greengrass to update the
Greengrass Cores in the targets with version 1.10.1 of the core software
running on x86_64 architecture.",
    "presignedUrlConfig": {
        "roleArn": "arn:aws::iam::123456789012:role/myS3UrlSignerRole",
        "expiresInSec": 3600
    },
    "jobExecutionsRolloutConfig": {},
    "createdAt": 1588718249.079,
    "lastUpdatedAt": 1588718253.419,
    "jobProcessDetails": {
        "numberOfCanceledThings": 0,
        "numberOfSucceededThings": 0,
        "numberOfFailedThings": 0,
        "numberOfRejectedThings": 0,
        "numberOfQueuedThings": 1,
        "numberOfInProgressThings": 0,
        "numberOfRemovedThings": 0,
        "numberOfTimedOutThings": 0
    },
    "timeoutConfig": {}
}
}

```

如需故障診斷協助，請參閱[疑難排解](#)。

CreateSoftwareUpdateJob API

您可以使用 CreateSoftwareUpdateJob API 來更新核心裝置上的 AWS IoT Greengrass 核心軟體或 OTA 更新代理程式軟體。此 API 會建立 AWS IoT 快照任務，在有可用更新時通知裝置。呼叫之後 CreateSoftwareUpdateJob，您可以使用其他 AWS IoT 任務命令來追蹤軟體更新。如需詳細資訊，請參閱《AWS IoT 開發人員指南》中的[任務](#)。

下列範例示範如何使用 AWS CLI 來建立任務，以更新 AWS IoT Greengrass 核心裝置上的 Core 軟體：

```
aws greengrass create-software-update-job \
--update-targets-architecture x86_64 \
--update-targets ["arn:aws:iot:region:123456789012:thing/myCoreDevice\""] \
--update-targets-operating-system ubuntu \
--software-to-update core \
--s3-url-signer-role arn:aws:iam::123456789012:role/myS3UrlSignerRole \
--update-agent-log-level WARN \
--amzn-client-token myClientToken1
```

create-software-update-job 命令會傳回一個 JSON 回應，其中包含任務 ID、任務 ARN，以及更新所安裝的軟體版本：

```
{
  "IotJobId": "GreengrassUpdateJob_c3bd7f36-ee80-4d42-8321-a1da0EXAMPLE",
  "IotJobArn": "arn:aws:iot:region:123456789012:job/GreengrassUpdateJob_c3bd7f36-ee80-4d42-8321-a1da0EXAMPLE",
  "PlatformSoftwareVersion": "1.9.2"
}
```

如需說明如何使用 create-software-update-job 更新核心裝置的步驟，請參閱 [the section called “建立 OTA 更新”](#)。

create-software-update-job 命令具有下列參數：

--update-targets-architecture

核心裝置的架構。

有效值：armv7l、armv6l、x86_64 或 aarch64

--update-targets

要更新的核心。此清單可包含個別核心的 ARN，以及其成員為核心之物件群組的 ARN。如需物件群組的詳細資訊，請參閱《AWS IoT 開發人員指南》中的[靜態物件群組](#)。

--update-targets-operating-system

核心裝置的作業系統。

有效值：ubuntu、amazon_linux、raspbian 或 openwrt

--software-to-update

指定核心的軟體或 OTA 更新代理程式軟體是否應更新。

有效值：core 或 ota_agent

--s3-url-signer-role

用於預先簽署連結至 AWS IoT Greengrass 軟體更新成品之 Amazon S3 URL 的 IAM 角色 ARN。角色連接的許可政策必須允許對目標中的儲存貯體執行 s3:GetObject 動作 AWS 區域。角色也必須允許 `iot.amazonaws.com` 將角色擔任為信任的實體。如需詳細資訊，請參閱[the section called “OTA 更新的 IAM 許可”](#)。

--amzn-client-token

(選用) 使用用戶端字串發出等冪請求。提供一組唯一的字串，避免因內部重新嘗試而產生重複更新。

--update-agent-log-level

(選用) OTA 更新代理程式所產生日誌陳述式的記錄層級。預設值為 ERROR。

有效值：NONE、TRACE、DEBUG、VERBOSE、INFO、WARN、ERROR 或 FATAL

Note

CreateSoftwareUpdateJob 僅接受下列支援架構和作業系統組合的要求：

- ubuntu/x86_64
- ubuntu/aarch64
- amazon_linux/x86_64
- raspbian/armv7l
- raspbian/armv6l
- openwrt/aarch64
- openwrt/armv7l

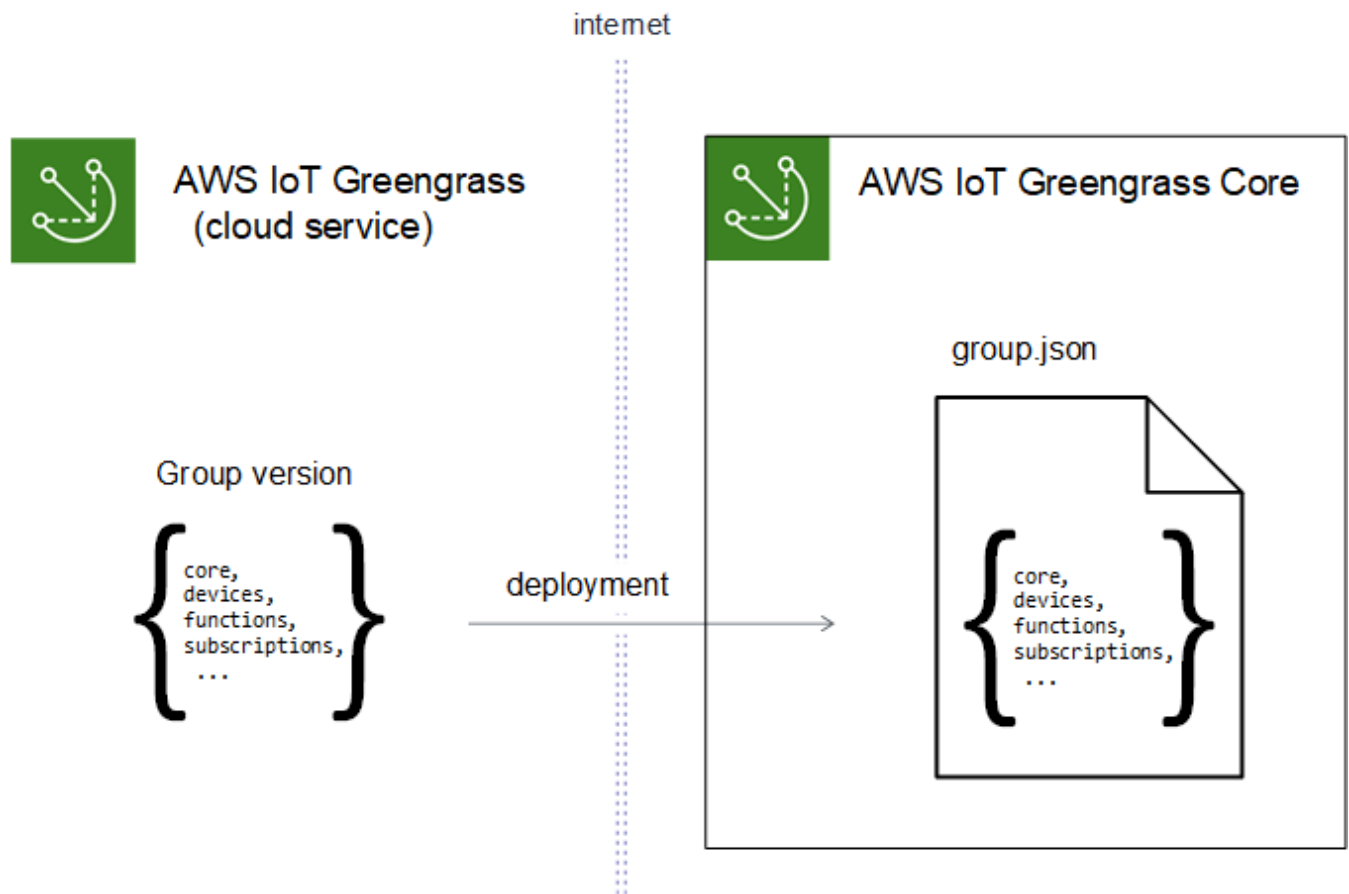
將 AWS IoT Greengrass 群組部署至 AWS IoT Greengrass 核心

使用 AWS IoT Greengrass 群組在邊緣環境中組織實體。您也可以使用 群組來控制群組中的實體如何彼此和 互動 AWS 雲端。例如，只有 群組中的 Lambda 函數會部署在本機執行，而且只有群組中的裝置可以使用本機 MQTT 伺服器進行通訊。

群組必須包含[核心](#)，這是執行 AWS IoT Greengrass Core 軟體 AWS IoT 的裝置。核心可做為邊緣閘道，並在邊緣環境中提供 AWS IoT Core 功能。您也可以根據不同的商務需求，將下列實體新增至群組中：

- 用戶端裝置。已表示為 AWS IoT 登錄檔中的物件。這些裝置必須執行 [FreeRTOS](#) 或使用[AWS IoT 裝置 SDK](#) 或 [AWS IoT Greengrass Discovery API](#) 來取得核心的連線資訊。只有屬於 群組成員的用戶端裝置才能連線到核心。
- Lambda 函數。在核心上執行程式碼的使用者定義無伺服器應用程式。Lambda 函數是在 中編寫 AWS Lambda 並從 Greengrass 群組參考。如需詳細資訊，請參閱[執行本機 Lambda 函數](#)。
- 連接器。在核心上執行程式碼的預先定義無伺服器應用程式。連接器可以提供與本機基礎設施、裝置通訊協定 AWS 和其他雲端服務的內建整合。如需詳細資訊，請參閱[使用連接器整合服務和通訊協定](#)。
- 訂閱。定義專為 MQTT 通訊授權的發佈者、訂閱者和 MQTT 主題 (或主體)。
- 資源。本機[裝置和磁碟區](#)、[機器學習模型](#)和[秘密](#)的參考，用於 Greengrass Lambda 函數和連接器的存取控制。
- 日誌。記錄 AWS IoT Greengrass 系統元件和 Lambda 函數的組態。如需詳細資訊，請參閱[the section called “使用 AWS IoT Greengrass 日誌進行監控”](#)。

您可以在 中管理 Greengrass 群組，AWS 雲端 然後將其部署到核心。部署會將群組組態複製到核心裝置上的 `group.json` 檔案。此檔案位於 `greengrass-root/ggc/deployments/group`：

**Note**

在部署期間，核心裝置上的 Greengrass 協助程式程序會停止並重新啟動。

從 AWS IoT 主控台部署群組

您可以從 AWS IoT 主控台內的群組組態頁面部署群組和管理其部署。

Note

若要在主控台中開啟此頁面，請選擇 Greengrass 裝置，然後選擇群組 (V1)，然後在 Greengrass 群組下選擇您的群組。

部署群組的目前版本

- 在群組組態頁面中，選擇部署。

檢視群組的部署歷史記錄

群組的部署歷史記錄包括日期和時間、群組版本，以及每個部署嘗試狀態。

1. 從群組組態頁面，選擇部署索引標籤。
2. 若要查看部署的詳細資訊，包括錯誤訊息，請在 Greengrass 裝置下從 AWS IoT 主控台選擇部署。

重新部署群組部署

如果目前的部署失敗或還原成不同的群組版本，您可能會想要重新部署部署。

1. 從 AWS IoT 主控台選擇 Greengrass 裝置，然後選擇群組 (V1)。
2. 選擇 Deployment (部署) 索引標籤。
3. 選擇您要重新部署的部署，然後選擇重新部署。

重設群組部署

您可能想要重設群組部署以移動或刪除群組，或是移除部署資訊。如需詳細資訊，請參閱[the section called “重設部署”](#)。

1. 從 AWS IoT 主控台選擇 Greengrass 裝置，然後選擇群組 (V1)。
2. 選擇 Deployment (部署) 索引標籤。
3. 選擇您要重設的部署，然後選擇重設部署。

使用 AWS IoT Greengrass API 部署群組

AWS IoT Greengrass API 提供下列動作來部署 AWS IoT Greengrass 群組和管理群組部署。您可以從、AWS CLI AWS IoT Greengrass API 或 AWS SDK 呼叫這些動作。

動作	描述
CreateDeployment	建立 NewDeployment 或 Redeployment 部署。

動作	描述
	如果目前的部署失敗，您可能想要重新部署部署。或者，您可能想要重新部署，將其還原至不同的群組版本。
GetDeploymentStatus	傳回部署的狀態：Building、InProgress、Success 或 Failure。 您可以設定 Amazon EventBridge 事件來接收部署通知。如需詳細資訊，請參閱 the section called “取得部署通知” 。
ListDeployments	傳回群組的部署歷史記錄。
ResetDeployments	重設群組的部署。 您可能想要重設群組部署以移動或刪除群組，或是移除部署資訊。如需詳細資訊，請參閱 the section called “重設部署” 。

Note

如需大量部署操作的詳細資訊，請參閱[the section called “建立大量部署”](#)。

取得群組 ID

群組 ID 通常用於 API 動作。您可以使用 [ListGroups](#) 動作，從群組清單中尋找目標群組的 ID。例如，在中 AWS CLI，使用 `list-groups` 命令。

```
aws greengrass list-groups
```

您也可以包含 `query` 選項來篩選結果。例如：

- 取得最近建立的群組：

```
aws greengrass list-groups --query "reverse(sort_by(Groups, &CreationTimestamp))[0]"
```

- 按名稱取得群組：

```
aws greengrass list-groups --query "Groups[?Name=='MyGroup']"
```

群組名稱不需要是唯一名稱，因此可能會傳回多個群組。

以下是 `list-groups` 回應範例。每個群組的資訊包括群組的 ID (在 `Id` 屬性中) 和最新群組版本的 ID (在 `LatestVersion` 屬性中)。若要取得其他群組版本 ID，請使用具有 [ListGroupVersions](#) 的群組 ID。

Note

您也可以在 AWS IoT 主控台中找到這些值。群組 ID 會顯示在群組的 Settings (設定) 頁面上。群組版本 IDs 會顯示在群組的部署索引標籤上。

```
{
  "Groups": [
    {
      "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/groups/00dedaaa-ac16-484d-ad77-c3eedEXAMPLE/versions/4cbc3f07-fc5e-48c4-a50e-7d356EXAMPLE",
      "Name": "MyFirstGroup",
      "LastUpdatedTimestamp": "2019-11-11T05:47:31.435Z",
      "LatestVersion": "4cbc3f07-fc5e-48c4-a50e-7d356EXAMPLE",
      "CreationTimestamp": "2019-11-11T05:47:31.435Z",
      "Id": "00dedaaa-ac16-484d-ad77-c3eedEXAMPLE",
      "Arn": "arn:aws:us-west-2:123456789012:/greengrass/groups/00dedaaa-ac16-484d-ad77-c3eedEXAMPLE"
    },
    {
      "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE/versions/8fe9e8ec-64d1-4647-b0b0-01dc8EXAMPLE",
      "Name": "GreenhouseSensors",
      "LastUpdatedTimestamp": "2020-01-07T19:58:36.774Z",
      "LatestVersion": "8fe9e8ec-64d1-4647-b0b0-01dc8EXAMPLE",
      "CreationTimestamp": "2020-01-07T19:58:36.774Z",
      "Id": "036ceaf9-9319-4716-ba2a-237f9EXAMPLE",
      "Arn": "arn:aws:us-west-2:123456789012:/greengrass/groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE"
    }
  ]
}
```

```
    "Arn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE"
  },
  ...
]
}
```

如果您未指定 AWS 區域，AWS CLI 命令會使用設定檔中的預設區域。若要傳回不同區域中的群組，請包含 `##` 選項。例如：

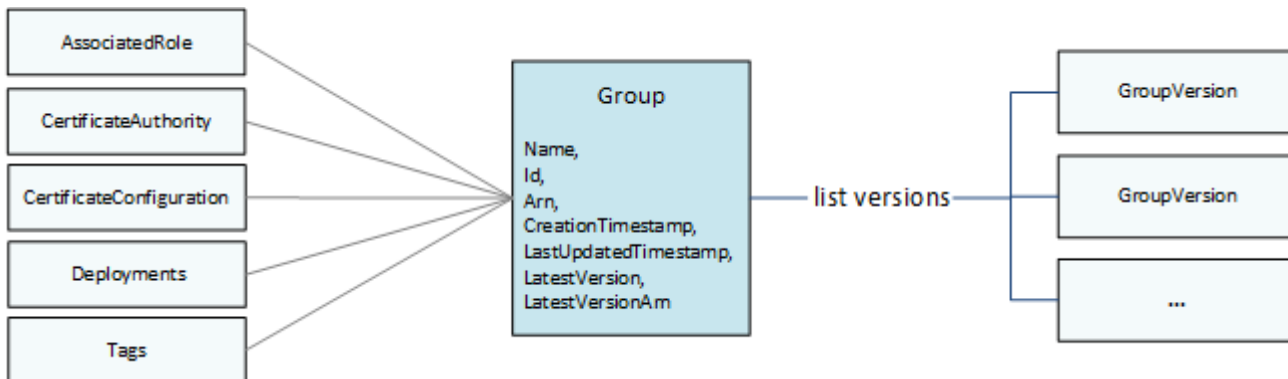
```
aws greengrass list-groups --region us-east-1
```

AWS IoT Greengrass 群組物件模型概觀

使用 AWS IoT Greengrass API 進程式設計時，了解 Greengrass 群組物件模型會很有幫助。

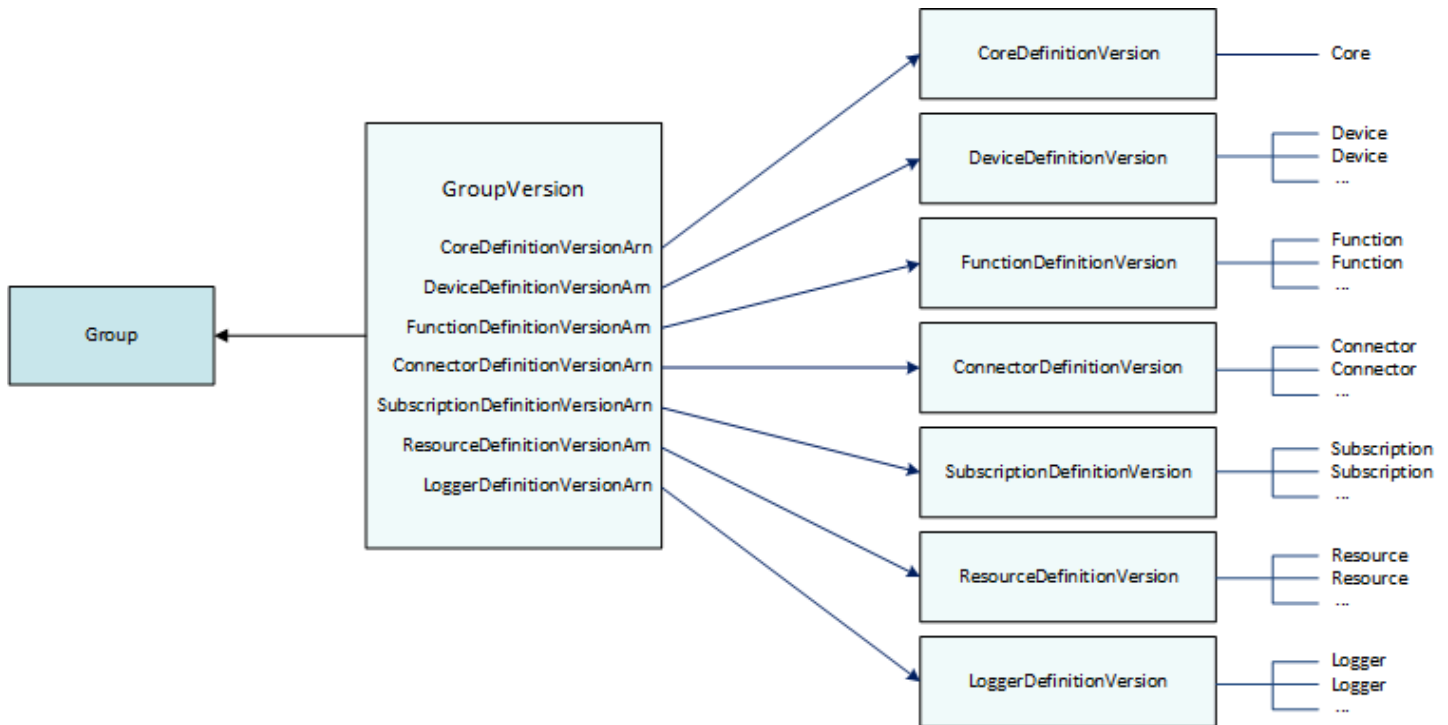
群組

在 AWS IoT Greengrass API 中，最上層 Group 物件包含中繼資料和 GroupVersion 物件清單。GroupVersion 物件會 Group 依 ID 與 建立關聯。



群組版本

GroupVersion 物件定義了群組成員資格。每個 GroupVersion 均參考 CoreDefinitionVersion 及其他依 ARN 的元件版本。這些參考決定了要包含在群組中的實體。



例如，若要在群組中包含三個 Lambda 函數、一個裝置和兩個訂閱，這些 GroupVersion 參考：

- CoreDefinitionVersion，其中包含必要核心。
- FunctionDefinitionVersion，其中包含三個函數。
- DeviceDefinitionVersion 包含用戶端裝置的。
- SubscriptionDefinitionVersion，其中包含兩個訂閱。

核心裝置上的已部署 GroupVersion 則決定了本機環境中的可用實體，及其互動方式。

群組元件

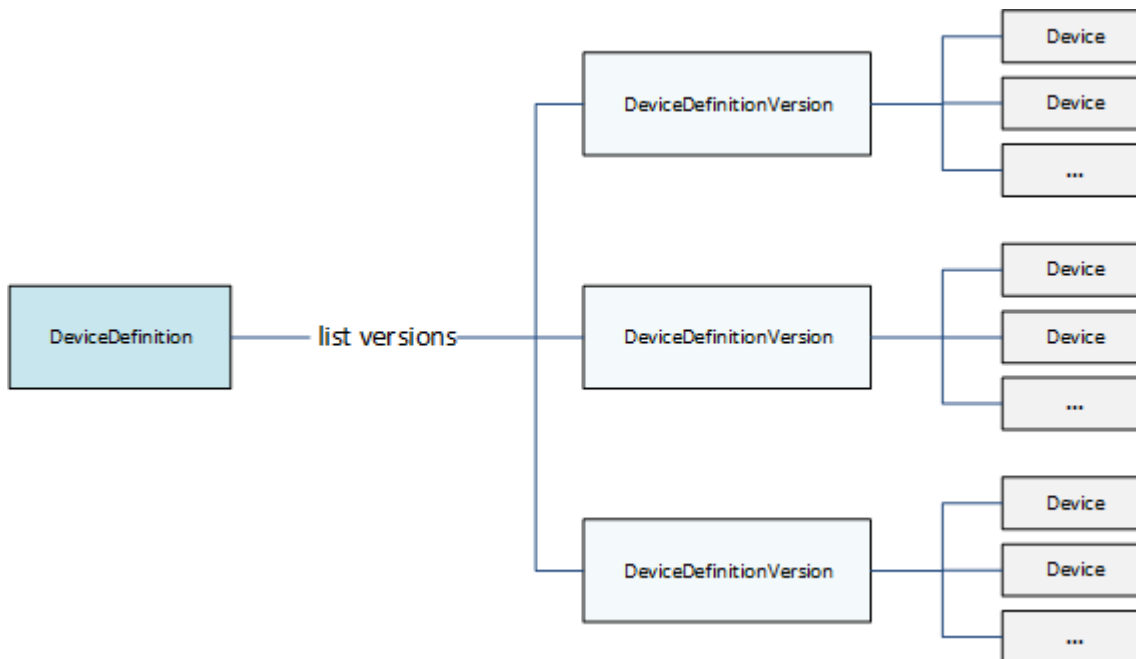
您新增至群組的元件分為三層：

- Definition (定義)，其會參考指定類型之 DefinitionVersion 物件清單。例如，DeviceDefinition 會參考 DeviceDefinitionVersion 物件的清單。
- DefinitionVersion，其中包含一組指定類型的實體。例如，DeviceDefinitionVersion 包含了 Device 物件的清單。
- 定義其屬性和行為的個別實體。例如，會 Device 定義 AWS IoT 登錄檔中對應用戶端裝置的 ARN、其裝置憑證的 ARN，以及其本機影子是否自動與雲端同步。

您可以將下列類型的實體新增至群組：

- [連接器](#)
- [核心](#)
- [Device](#)
- [函數](#)
- [Logger](#)
- [Resource](#)
- [訂閱](#)

下面範例 DeviceDefinition 參考了三個 DeviceDefinitionVersion 物件，其中每個都會包含多個 Device 物件。一次只會在群組中使用一個 DeviceDefinitionVersion。



更新群組

在 AWS IoT Greengrass API 中，您可以使用版本來更新群組的組態。版本是不可變的，因此如要新增、移除或變更群組元件，您必須建立包含有全新或更新實體的 DefinitionVersion 物件。

您可以將新的 DefinitionVersions 物件關聯至新的或現有的 Definition 物件。例如，您可以使用 CreateFunctionDefinition 動作來建立包含以 FunctionDefinitionVersion 做為初始版本的 FunctionDefinition，或者您可以使用 CreateFunctionDefinitionVersion 動作，同時參考現有的 FunctionDefinition。

建立群組元件之後，您可以建立 `GroupVersion`，其中包含您要包含在群組中的所有 `DefinitionVersion` 物件。然後，您要部署 `GroupVersion`。

若要部署 `GroupVersion`，其必須參考其中僅包含一個 Core 的 `CoreDefinitionVersion`。所有參考的實體都必須是該群組的成員。此外，[Greengrass 服務角色](#) 必須與您要部署 AWS 區域的 AWS 帳戶中的相關聯 `GroupVersion`。

 Note

API 中的 `Update` 動作是用於變更 `Group` 或元件 `Definition` (定義) 物件的名稱。

更新參考 AWS 資源的實體

`Greengrass Lambda` 函數和[秘密資源](#)會定義 `Greengrass` 特定的屬性，並參考對應的 AWS 資源。若要更新這些實體，您可以變更對應的 AWS 資源，而不是 `Greengrass` 物件。例如，`Lambda` 函數參考中的函數，`AWS Lambda` 並定義 `Greengrass` 群組特定的生命週期和其他屬性。

- 若要更新 `Lambda` 函數程式碼或封裝相依性，請在 中進行變更 `AWS Lambda`。在下一個群組部署期間，這些變更會從 擷取 `AWS Lambda` 並複製到您的本機環境。
- 若要更新 [Greengrass 特定屬性](#)，您可以建立其中包含已更新 `Function` 屬性的 `FunctionDefinitionVersion`。

 Note

`Greengrass Lambda` 函數可以透過別名 ARN 或版本 ARN 來參考 `Lambda` 函數。如果是參考別名 ARN (建議)，則當您在 `AWS Lambda` 中發佈新的函數版本時，這時不需要更新 `FunctionDefinitionVersion` (或 `SubscriptionDefinitionVersion`)。如需詳細資訊，請參閱[the section called “依別名或版本參考函數”](#)。

另請參閱

- [the section called “取得部署通知”](#)
- [the section called “重設部署”](#)
- [the section called “建立大量部署”](#)
- [疑難排解部署問題](#)

- [AWS IoT Greengrass Version 1 API 參考](#)
- [AWS IoT Greengrass 命令](#) AWS CLI 參考中的 命令

取得部署通知

Amazon EventBridge 事件規則可為您提供 Greengrass 群組部署狀態變更的通知。EventBridge 提供近乎即時的系統事件串流，說明 AWS 資源的變更。至少會一次將這些事件 AWS IoT Greengrass 傳送至 EventBridge。這表示 AWS IoT Greengrass 可能會傳送指定事件的多個副本，以確保交付。此外，您的事件接聽程式可能不會依事件發生順序接收事件。

Note

Amazon EventBridge 是一種事件匯流排服務，可用來將應用程式與各種來源的資料連線，例如 [Greengrass 核心裝置](#) 和部署通知。如需詳細資訊，請參閱《Amazon EventBridge 使用者指南》中的 [什麼是 Amazon EventBridge？](#)

AWS IoT Greengrass 當群組部署變更狀態時，會發出事件。您可以建立 EventBridge 規則，針對所有狀態轉換或轉換至您指定的狀態執行該規則。當部署進入啟動規則的狀態時，EventBridge 會叫用規則中定義的目標動作。這可讓您傳送通知、擷取事件資訊、採取修正動作，或啟動其他事件來回應狀態變更。例如，您可以為下列使用案例建立規則：

- 啟動部署後操作，例如下載資產和通知人員。
- 部署成功或失敗後傳送通知。
- 發佈關於部署事件的自訂指標。

AWS IoT Greengrass 當部署進入下列狀態時，會發出事件：Building、Success、InProgress 和 Failure。

Note

目前尚不支援監控 [大量部署](#) 操作的狀態。不過，會針對屬於大量部署一部分的個別群組部署 AWS IoT Greengrass 發出狀態變更事件。

群組部署狀態變更事件

部署狀態變更的[事件](#)會使用下列格式：

```
{
  "version": "0",
  "id": " cd4d811e-ab12-322b-8255-EXAMPLEb1bc8",
  "detail-type": "Greengrass Deployment Status Change",
  "source": "aws.greengrass",
  "account": "123456789012",
  "time": "2018-03-22T00:38:11Z",
  "region": "us-west-2",
  "resources": [],
  "detail": {
    "group-id": "284dcd4e-24bc-4c8c-a770-EXAMPLEf03b8",
    "deployment-id": "4f38f1a7-3dd0-42a1-af48-EXAMPLE09681",
    "deployment-type": "NewDeployment|Redeployment|ResetDeployment|ForceResetDeployment",
    "status": "Building|InProgress|Success|Failure"
  }
}
```

您可以建立套用至一或多個群組的規則。您可以依下列一或多個部署類型和部署狀態來篩選規則：

部署類型

- NewDeployment。群組版本的第一個部署。
- ReDeployment。群組版本的重新部署。
- ResetDeployment。刪除存放在 和 AWS IoT Greengrass 核心 AWS 雲端 上的部署資訊。如需詳細資訊，請參閱[the section called “重設部署”](#)。
- ForceResetDeployment。刪除存放在 中的部署資訊，AWS 雲端 並報告成功，無需等待核心回應。核心如已連線或將於下次連線，存放在核心內的部署資訊也會予以刪除。

部署狀態

- Building。AWS IoT Greengrass 正在驗證群組組態和建置部署成品。
- InProgress。AWS IoT Greengrass 核心上的部署正在進行中。
- Success。部署成功。
- Failure。部署失敗。

事件可能會重複或不按順序。若要判斷事件的順序，請使用 time 屬性。

 Note

AWS IoT Greengrass 不會使用 `resources` 屬性，因此一律為空白。

建立 EventBridge 規則的先決條件

為建立 EventBridge 規則之前 AWS IoT Greengrass，請執行下列動作：

- 熟悉 Eventbridge 中的事件、規則和目標。
- 建立和設定將由您的 Eventbridge 規則叫用的目標。規則可以叫用許多類型的目標，包括：
 - Amazon Simple Notification Service (Amazon SNS)
 - AWS Lambda 函數
 - Amazon Kinesis Video Streams
 - Amazon Simple Queue Service (Amazon SQS) 佇列

如需詳細資訊，請參閱《Amazon EventBridge 使用者指南》中的[什麼是 Amazon EventBridge？](#)和[Amazon EventBridge 入門](#)。

設定部署通知 (主控台)

使用下列步驟建立 EventBridge 規則，在群組的部署狀態變更時發佈 Amazon SNS 主題。藉由這個規則，Web 伺服器、電子郵件地址和其他主題訂閱者將能回應事件。如需詳細資訊，請參閱《Amazon EventBridge 使用者指南》中的[從 AWS 資源建立事件觸發的 EventBridge 規則](#)。EventBridge

1. 開啟 [Amazon EventBridge 主控台](#)。
2. 在導覽窗格中，選擇規則。
3. 選擇建立規則。
4. 輸入規則的名稱和描述。

在同一個區域和同一個事件匯流排上，規則不能與另一個規則同名。

5. 針對事件匯流排，選擇要與此規則建立關聯的事件匯流排。如果您想要此規則匹配來自您的帳戶的事件，請選取 AWS 預設事件匯流排。當您帳戶中的 AWS 服務發出事件時，一律會前往您帳戶的預設事件匯流排。
6. 針對規則類型，選擇具有事件模式的規則。

7. 選擇下一步。
8. 在事件來源欄位中，選擇 AWS 服務。
9. 針對事件模式，選擇 AWS 服務。
10. 針對AWS 服務，選擇 Greengrass。
11. 在 Event type (事件類型)，選擇 Greengrass Deployment Status Change (Greengrass 部署狀態變更)。

 Note

AWS 透過 CloudTrail 的 API 呼叫事件類型是以與的 AWS IoT Greengrass 整合為基礎 AWS CloudTrail。您可以使用此選項來建立由讀取或寫入呼叫 AWS IoT Greengrass API 所起始的規則。如需詳細資訊，請參閱[the section called “使用 記錄 AWS IoT Greengrass API 呼叫 AWS CloudTrail”](#)。

12. 選擇啟動通知的部署狀態。
 - 若要接收所有狀態變更事件的通知，請選擇 Any state (任何狀態)。
 - 若只要接收某些狀態變更事件的通知，請選擇 Specific state(s) (特定狀態)，然後選擇目標狀態。
13. 選擇啟動通知的部署類型。
 - 若要接收所有部署類型的通知，請選擇 Any state (任何狀態)。
 - 若只要接收某些部署類型的通知，請選擇 Specific state(s) (特定狀態)，然後選擇目標部署類型。
14. 選擇下一步。
15. 在目標類型欄位中，選擇 AWS 服務。
16. 針對選取目標，設定您的目標。此範例使用 Amazon SNS 主題，但您可以設定其他目標類型來傳送通知。
 - a. 在 Target (目標)，選擇 SNS topic (SNS 主題)。
 - b. 在 Topic (主題)，選擇您的目標主題。
 - c. 選擇下一步。
17. 在標籤下，定義規則的標籤或將欄位保留空白。
18. 選擇下一步。
19. 檢閱規則的詳細資訊，然後選擇建立規則。

設定部署通知 (CLI)

使用下列步驟建立 EventBridge 規則，在群組的部署狀態變更時發佈 Amazon SNS 主題。藉由這個規則，Web 伺服器、電子郵件地址和其他主題訂閱者將能回應事件。

1. 建立 規則。

- 使用 `### ID ## group-id`。AWS IoT Greengrass

```
aws events put-rule \  
  --name TestRule \  
  --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail\": {\"group-id\":  
    [\"group-id\"]}}"
```

模式省略的屬性會遭到忽略。

2. 新增主題作為規則目標。

- 將 `topic-arn` 取代為 Amazon SNS 主題的 ARN。

```
aws events put-targets \  
  --rule TestRule \  
  --targets "Id"="1", "Arn"="topic-arn"
```

Note

若要讓 Amazon EventBridge 呼叫您的目標主題，您必須將以資源為基礎的政策新增到您的主題。如需詳細資訊，請參閱《Amazon EventBridge 使用者指南》中的 [Amazon SNS 許可](#)。

如需詳細資訊，請參閱《Amazon EventBridge 使用者指南》中的 [EventBridge 中的事件和事件模式](#)。

設定部署通知 (CloudFormation)

使用 CloudFormation 範本建立 EventBridge 規則，傳送 Greengrass 群組部署狀態變更的通知。如需詳細資訊，請參閱《AWS CloudFormation 使用者指南》中的 [Amazon EventBridge 資源類型參考](#)。

另請參閱

- [部署 AWS IoT Greengrass 群組](#)
- [《Amazon EventBridge 使用者指南》中的什麼是 Amazon EventBridge？](#) EventBridge

重設部署

此功能適用於 AWS IoT Greengrass Core v1.1 和更新版本。

您可能想要重設群組的部署，以執行下列作業：

- 刪除群組，例如您想要將群組的核心移至另一個群組，或群組的核心已重新影複。刪除群組之前，您必須重設群組的部署，以將核心與另一個 Greengrass 群組搭配使用。
- 將該群組的核心移至不同的群組。
- 將群組回復成任何部署之前的狀態。
- 移除核心裝置的部署組態。
- 刪除核心裝置或雲端的敏感資料。
- 部署新群組組態至核心，且在目前的群組中無須以另一個核心取代此核心。

Note

AWS IoT Greengrass 核心軟體 v1.0.0 中無法使用重設部署功能。您無法刪除已使用 v1.0.0 部署的群組。

重設部署操作會先清除雲端中針對特定群組所儲存的所有部署資訊。然後，它會指示群組的核心裝置清除所有部署相關資訊 (Lambda 函數、使用者日誌、影子資料庫和伺服器憑證，但不是使用者定義 config.json 或 Greengrass 核心憑證)。群組目前的部署狀態如果是 In Progress 或 Building，群組部署重設作業就無法啟動。

從 AWS IoT 主控台重設部署

您可以從 AWS IoT 主控台的群組組態頁面重設群組部署。

1. 在 AWS IoT 主控台導覽窗格的管理下，展開 Greengrass 裝置，然後選擇群組 (V1)。
2. 選擇目標群組。

3. 從部署索引標籤中，選擇重設部署。
4. 在此 Greengrass 群組重設部署對話方塊中，輸入 **confirm** 以同意，然後選擇重設部署。

使用 AWS IoT Greengrass API 重設部署

您可以使用 AWS CLI、AWS IoT Greengrass API 或 AWS SDK 中的 `ResetDeployments` 動作來重設部署。本主題中的範例使用 CLI。

```
aws greengrass reset-deployments --group-id GroupId [--force]
```

reset-deployments CLI 命令的引數：

--group-id

群組的 ID。使用 `list-groups` 命令來取得這個值。

--force

選用。如果此群組的裝置發生遺失、遭竊或損毀，請使用此參數。此選項使重設部署程序在雲端中所有的部署資訊均已清除後回報成功，而無需等待核心裝置回應。不過，如果核心裝置為作用中或將為作用中，這時它也會執行清除作業。

`reset-deployments` CLI 命令的輸出如下所示：

```
{
  "DeploymentId": "4db95ef8-9309-4774-95a4-eea580b6ceef",
  "DeploymentArn": "arn:aws:greengrass:us-west-2:106511594199:/greengrass/groups/
b744ed45-a7df-4227-860a-8d4492caa412/deployments/4db95ef8-9309-4774-95a4-eea580b6ceef"
}
```

您可以使用 `get-deployment-status` 命令檢查重設政策的狀態。

```
aws greengrass get-deployment-status --deployment-id DeploymentId --group-id GroupId
```

get-deployment-status CLI 命令的引數：

--deployment-id

部署 ID。

--group-id

群組的 ID。

get-deployment-status CLI 命令的輸出如下所示：

```
{
  "DeploymentStatus": "Success",
  "UpdatedAt": "2017-04-04T00:00:00.000Z"
}
```

當準備重設部署時，DeploymentStatus 會設為 Building。當重設部署準備就緒，但 AWS IoT Greengrass 核心尚未取得重設部署時，DeploymentStatus 為 InProgress。

如果重設操作失敗，在回應中會傳回錯誤資訊。

另請參閱

- [部署 AWS IoT Greengrass 群組](#)
- AWS IoT Greengrass Version 1 API 參考中的 [ResetDeployments](#)
- API AWS IoT Greengrass Version 1 參考中的 [GetDeploymentStatus](#)

建立群組的大量部署

您可以使用簡單的 API 呼叫，一次部署大量 Greengrass 群組。這些部署由固定上限的自適性速率所觸發。

本教學說明如何使用 AWS CLI 在中建立和監控大量群組部署 AWS IoT Greengrass。本教學課程中的大量部署範例包含多個群組。您可以在實作中使用此範例來新增所需的群組數量。

本教學課程所述以下高階執行步驟：

1. [建立和上傳大量部署輸入檔](#)
2. [為大量部署建立和設定 IAM 執行角色](#)
3. [允許執行角色存取 S3 儲存貯體](#)
4. [部署群組](#)
5. [測試部署](#)

先決條件

為完成此教學課程您需要：

- 一或多個可部署的 Greengrass 群組。如需建立 AWS IoT Greengrass 群組和核心的詳細資訊，請參閱 [入門 AWS IoT Greengrass](#)。
- 在機器上安裝 AWS CLI 和設定的。如需更多詳細資訊，請參閱 [AWS CLI 使用者指南](#) 相關文章。
- 在相同中建立的 S3 儲存貯體 AWS 區域 AWS IoT Greengrass。如需詳細資訊，請參閱《Amazon Simple Storage Service 使用者指南》中的 [建立和設定 S3 儲存貯體](#)。

Note

目前，不支援已啟用 SSE KMS 的儲存貯體。

步驟 1：建立和上傳大量部署輸入檔

在此步驟中，您會建立部署輸入檔案，並將其上傳至 Amazon S3 儲存貯體。此檔案是序列化、以行分隔的 JSON 檔案，其中包含大量部署中每個群組的相關資訊。當您初始化大量群組部署時，AWS IoT Greengrass 會使用此資訊代表您部署每個群組。

1. 執行以下命令為您要部署的每個群組取得 groupId。您需要將 groupId 輸入大量部署輸入檔中，讓 AWS IoT Greengrass 可以識別每個要部署的群組。

Note

您也可以在 AWS IoT 主控台中找到這些值。群組 ID 會顯示在群組的 Settings (設定) 頁面上。群組版本 IDs 會顯示在群組的部署索引標籤上。

```
aws greengrass list-groups
```

回應包含您 AWS IoT Greengrass 帳戶中每個群組的相關資訊：

```
{
```

```
"Groups": [  
  {  
    "Name": "string",  
    "Id": "string",  
    "Arn": "string",  
    "LastUpdatedTimestamp": "string",  
    "CreationTimestamp": "string",  
    "LatestVersion": "string",  
    "LatestVersionArn": "string"  
  },  
],  
"NextToken": "string"  
}
```

執行以下命令為您要部署的每個群組取得 groupVersionId。

```
list-group-versions --group-id groupId
```

回應包含群組中所有版本的相關資訊。請記下您要使用的群組版本Version值。

```
{  
  "Versions": [  
    {  
      "Arn": "string",  
      "Id": "string",  
      "Version": "string",  
      "CreationTimestamp": "string"  
    },  
  ],  
  "NextToken": "string"  
}
```

2. 在您的電腦終端機或您選擇的編輯器中，從以下範例建立檔案 *MyBulkDeploymentInputFile*。此檔案包含要包含在大量部署中的每個 AWS IoT Greengrass 群組的相關資訊。雖然這個範例中定義多個群組，但對於此教學課程，您的檔案可以只包含一個群組。

Note

此檔案的大小必須小於 100 MB。

```
{"GroupId": "groupId1", "GroupVersionId": "groupVersionId1",  
  "DeploymentType": "NewDeployment"}  
{"GroupId": "groupId2", "GroupVersionId": "groupVersionId2",  
  "DeploymentType": "NewDeployment"}  
{"GroupId": "groupId3", "GroupVersionId": "groupVersionId3",  
  "DeploymentType": "NewDeployment"}  
...
```

每個記錄 (或列) 包含一個群組物件。每個群組物件包含其對應的 GroupId 和 GroupVersionId 及 DeploymentType。目前，僅 AWS IoT Greengrass 支援 NewDeployment 大量部署類型。

儲存並關閉檔案。請記下檔案的位置。

3. 在終端機中使用下列命令，將輸入檔案上傳至 Amazon S3 儲存貯體。以您的檔案位置與名稱取代檔案路徑。如需相關資訊，請參閱[將物件新增至儲存貯體](#)。

```
aws s3 cp path/MyBulkDeploymentInputFile s3://amzn-s3-demo-bucket/
```

步驟 2：建立和設定 IAM 執行角色

在此步驟中，您會使用 IAM 主控台來建立獨立執行角色。然後，您可以在角色與之間建立信任關係，AWS IoT Greengrass 並確保您的 IAM 使用者具有執行角色 PassRole 的權限。這可讓 AWS IoT Greengrass 擔任您的執行角色，並代表您建立部署。

1. 使用下列政策來建立執行角色。此政策文件允許 AWS IoT Greengrass 在替您建立每個部署時存取您的大量部署輸入檔。

如需建立 IAM 角色和委派許可的詳細資訊，請參閱[建立 IAM 角色](#)。

JSON

```
{
```

```

"Version": "2012-10-17",
"Statement": [
  {
    "Sid": "VisualEditor0",
    "Effect": "Allow",
    "Action": "greengrass:CreateDeployment",
    "Resource": [
      "arn:aws:greengrass:us-east-1:123456789012:/greengrass/
groups/groupId1",
      "arn:aws:greengrass:us-east-1:123456789012:/greengrass/groups/groupId2",
      "arn:aws:greengrass:us-east-1:123456789012:/greengrass/groups/groupId3"
    ]
  }
]
}

```

Note

對於大量部署輸入檔中要由 AWS IoT Greengrass 部署的每個群組或群組版本，此政策必須具有資源。若要允許存取所有群組，請為 Resource 指定星號：

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

2. 將執行角色的信任關係修改為包含 AWS IoT Greengrass。這可讓 AWS IoT Greengrass 使用您的執行角色及其連接的許可。如需相關資訊，請參閱[編輯現有角色的信任關係](#)。

我們建議您也在信任政策中包含 `aws:SourceArn` 和 `aws:SourceAccount` 全域條件內容金鑰，以協助防止混淆代理人安全問題。條件內容索引鍵會限制存取，只允許來自指定帳戶和 Greengrass 工作區的這些請求。如需有關混淆代理人問題的詳細資訊，請參閱[預防跨服務混淆代理人](#)。

JSON

```

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

```

```

    "Effect": "Allow",
    "Principal": {
      "Service": "greengrass.amazonaws.com"
    },
    "Action": "sts:AssumeRole",
    "Condition": {
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      },
      "ArnLike": {
        "aws:SourceArn": "arn:aws:greengrass:us-east-1:123456789012:*"
      }
    }
  }
]
}

```

3. 將執行角色的 IAM PassRole 許可授予您的 IAM 使用者。此 IAM 使用者是用來啟動大量部署的使用者。PassRole 許可可讓您的 IAM 使用者將您的執行角色傳遞給 AWS IoT Greengrass 以供使用。如需詳細資訊，請參閱[授予使用者將角色傳遞至 AWS 服務的許可](#)。

使用下列範例來更新連接至執行角色的 IAM 政策。視需要修改此範例。

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1508193814000",
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": [
        "arn:aws:iam::123456789012:user/executionRoleArn"
      ],
      "Condition": {
        "StringEquals": {
          "iam:PassedToService": "greengrass.amazonaws.com"
        }
      }
    }
  ]
}

```

```
    }  
  ]  
}
```

步驟 3：允許執行角色存取 S3 儲存貯體

若要開始大量部署，您的執行角色必須能夠從 Amazon S3 儲存貯體讀取大量部署輸入檔案。將下列範例政策連接至 Amazon S3 儲存貯體，以便您的執行角色可存取其GetObject許可。

如需詳細資訊，請參閱[如何新增 S3 儲存貯體政策？](#)

JSON

```
{  
  "Version": "2012-10-17",  
  "Id": "examplePolicy",  
  "Statement": [  
    {  
      "Sid": "Stmt1535408982966",  
      "Effect": "Allow",  
      "Principal": {  
        "AWS": [  
          "executionRoleArn"  
        ]  
      },  
      "Action": "s3:GetObject",  
      "Resource":  
        "arn:aws:s3:::amzn-s3-demo-bucket/objectKey"  
    }  
  ]  
}
```

您可以在終端機中使用下列命令，以檢查儲存貯體的政策：

```
aws s3api get-bucket-policy --bucket amzn-s3-demo-bucket
```


Note

您可以直接修改執行角色，將許可授予 Amazon S3 儲存貯體的GetObject許可。若要這麼做，請將以下範例政策連接到執行角色。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/objectKey"
    }
  ]
}
```

步驟 4：部署群組

在此步驟中，您將為大量部署輸入檔中設定的所有群組版本開始大量部署操作。每個群組版本的部署動作都是 NewDeploymentType 類型。

Note

當相同帳戶仍在執行其他大量部署時，您不能呼叫 StartBulkDeployment。請求會遭到拒絕。

1. 使用下列命令來開始大量部署。

建議您在每個 StartBulkDeployment 請求中包含 X-Amzn-Client-Token 字符。關於字符和請求參數，這些請求是等冪。此字符可以是任何唯一的、區分大小寫的字串，最多 64 個 ASCII 字元。

```
aws greengrass start-bulk-deployment --cli-input-json "{
  "InputFileUri": "URI of file in S3 bucket",
```

```
"ExecutionRoleArn": "ARN of execution role",  
"AmznClientToken": "your Amazon client token"  
}"
```

此命令應該會產生成功狀態碼 200 及以下的回應：

```
{  
  "bulkDeploymentId": UUID  
}
```

請記下大量部署 ID。它可用於檢查大量部署的狀態。

Note

雖然目前不支援大量部署操作，但您可以建立 Amazon EventBridge 事件規則，以取得個別群組部署狀態變更的通知。如需詳細資訊，請參閱[the section called “取得部署通知”](#)。

2. 使用下列命令來檢查大量部署的狀態。

```
aws greengrass get-bulk-deployment-status --bulk-deployment-id 1234567
```

除了 JSON 承載資訊，此命令還應該會傳回成功狀態碼 200：

```
{  
  "BulkDeploymentStatus": Running,  
  "Statistics": {  
    "RecordsProcessed": integer,  
    "InvalidInputRecords": integer,  
    "RetryAttempts": integer  
  },  
  "CreatedAt": "string",  
  "ErrorMessage": "string",  
  "ErrorDetails": [  
    {  
      "DetailedErrorCode": "string",  
      "DetailedErrorMessage": "string"  
    }  
  ]  
}
```

```
]
}
```

`BulkDeploymentStatus` 包含大量執行的目前狀態。執行可以有六種不同狀態：

- `Initializing`。已收到大量部署請求，且執行正在準備開始。
- `Running`。大量部署執行已開始。
- `Completed`。大量部署執行已完成處理所有記錄。
- `Stopping`。大量部署執行已收到要停止的命令，並將很快終止。當先前部署是 `Stopping` 狀態時，您無法開始新的大量部署。
- `Stopped`。大量部署執行已手動停止。
- `Failed`。大量部署執行發生錯誤並已終止。您可以在 `ErrorDetails` 欄位中找到錯誤詳細資訊。

JSON 承載還包括大量部署進度的統計資訊。您可以使用此資訊判斷已處理多少群組，以及多少群組已失敗。統計資訊包括：

- `RecordsProcessed`：已嘗試的群組記錄數目。
- `InvalidInputRecords`：已傳回不可重試錯誤的記錄總數。例如，如果輸入檔中的群組記錄使用無效格式或指定不存在的群組版本，或者，如果執行未授權來部署群組或群組版本，就可能發生這種情況。
- `RetryAttempts`：已傳回不可重試錯誤的部署嘗試次數。例如，如果嘗試部署群組傳回調節錯誤，則會觸發重試。群組部署最多可重試五次。

在大量部署執行失敗的情況下，這個承載還包含 `ErrorDetails` 區段，可用於故障排除。它包含執行失敗原因的相關資訊。

您可以定期檢查大量部署的狀態，以確認其是否如預期進行。部署完成後，`RecordsProcessed` 應該等於大量部署輸入檔中的部署群組數目。這表示已處理每個記錄。

步驟 5：測試部署

使用 `ListBulkDeployments` 命令來尋找大量部署的 ID。

```
aws greengrass list-bulk-deployments
```

此命令會傳回所有大量部署的清單 (從最新到最舊)，包括您的 BulkDeploymentId。

```
{
  "BulkDeployments": [
    {
      "BulkDeploymentId": 1234567,
      "BulkDeploymentArn": "string",
      "CreatedAt": "string"
    }
  ],
  "NextToken": "string"
}
```

現在呼叫 ListBulkDeploymentDetailedReports 命令，收集每個部署的詳細資訊。

```
aws greengrass list-bulk-deployment-detailed-reports --bulk-deployment-id 1234567
```

除了 JSON 承載資訊，此命令還應該會傳回成功狀態碼 200：

```
{
  "BulkDeploymentResults": [
    {
      "DeploymentId": "string",
      "GroupVersionedArn": "string",
      "CreatedAt": "string",
      "DeploymentStatus": "string",
      "ErrorMessage": "string",
      "ErrorDetails": [
        {
          "DetailedErrorCode": "string",
          "DetailedErrorMessage": "string"
        }
      ]
    }
  ]
}
```

```
],  
  "NextToken": "string"  
}
```

這個承載通常包含每個部署及其部署狀態的分頁清單 (從最新到最舊)。它還包含萬一大量部署執行失敗時的詳細資訊。同樣，所列的部署總數應該等於大量部署輸入檔中識別的群組總數。

在部署變成終止狀態 (成功或失敗) 之前，傳回的資訊可能變更。在此之前，您可以定期呼叫此命令。

大量部署故障診斷

如果大量部署不成功，您可以嘗試以下故障診斷步驟。在您的終端機中執行命令。

輸入檔錯誤故障診斷

如果大量部署輸入檔的語法錯誤，則大量部署可能失敗。這會傳回大量部署狀態 Failed，錯誤訊息會指出第一個驗證錯誤的行號。有四個可能的錯誤：

- InvalidInputFile: Missing *GroupId* at line number: *line number*

此錯誤表示指定的輸入檔列無法註冊指定的參數。可能遺漏的參數是 GroupId 和 GroupVersionId。

- InvalidInputFile: Invalid deployment type at line number : *line number*. Only valid type is 'NewDeployment'.

此錯誤表示指定的輸入檔列列出的是無效的部署類型。目前，唯一支援的部署類型是 NewDeployment。

- Line *%s* is too long in S3 File. Valid line is less than 256 chars.

此錯誤表示指定的輸入檔列太長，必須縮短。

- Failed to parse input file at line number: *line number*

此錯誤表示指定的輸入檔列不是有效的 json。

檢查並行大量部署

當另一個大量部署仍在執行或處於未終止狀態時，您不能啟動新的大量部署。這可能導致 `Concurrent Deployment Error`。您可以使用 `ListBulkDeployments` 命令來驗證目前並未進行大量部署。此命令從最新到最舊列出大量部署。

```
{
  "BulkDeployments": [
    {
      "BulkDeploymentId": BulkDeploymentId,
      "BulkDeploymentArn": "string",
      "CreatedAt": "string"
    }
  ],
  "NextToken": "string"
}
```

使用第一個列出的大量部署的 `BulkDeploymentId` 以執行 `GetBulkDeploymentStatus` 命令。如果您的最新大量部署處於執行狀態 (`Initializing` 或 `Running`)，請使用下列命令來停止大量部署。

```
aws greengrass stop-bulk-deployment --bulk-deployment-id BulkDeploymentId
```

這個動作會導致狀態 `Stopping`，直到部署 `Stopped` 為止。在部署已達到 `Stopped` 狀態後，您就可以開始新的大量部署。

檢查 ErrorDetails

執行 `GetBulkDeploymentStatus` 命令以傳回 JSON 承載，其中包含任何大量部署執行失敗的相關資訊。

```
"Message": "string",
"ErrorDetails": [
  {
    "DetailedErrorCode": "string",
    "DetailedErrorMessage": "string"
  }
]
```

如果結束時發生錯誤，此呼叫傳回的 `ErrorDetails` JSON 承載會包含大量部署執行失敗的詳細資訊。例如，400 系列中的錯誤狀態碼表示輸入參數或呼叫者相依性中的輸入錯誤。

檢查 AWS IoT Greengrass 核心日誌

您可以檢視 AWS IoT Greengrass 核心日誌來疑難排解問題。使用下列命令以檢視 `runtime.log`：

```
cd /greengrass/ggc/var/log
sudo cat system/runtime.log | more
```

如需 AWS IoT Greengrass 記錄的詳細資訊，請參閱 [使用 AWS IoT Greengrass 日誌監控](#)。

另請參閱

如需詳細資訊，請參閱下列資源：

- [部署 AWS IoT Greengrass 群組](#)
- 《[命令參考](#)》中的 [Amazon S3 API](#) 命令 AWS CLI
- [AWS IoT Greengrass 命令](#) AWS CLI 參考中的 命令

在 AWS IoT Greengrass 核心上執行 Lambda 函數

AWS IoT Greengrass 為您撰寫的使用者定義程式碼提供容器化 Lambda 執行期環境 AWS Lambda。部署至 AWS IoT Greengrass 核心執行的 Lambda 函數會在核心的本機 Lambda 執行時間中執行。本機 Lambda 函數可由本機事件、來自雲端的訊息和其他來源觸發，將本機運算功能帶入用戶端裝置。例如，您可以在將資料傳輸至雲端之前，使用 Greengrass Lambda 函數來篩選裝置資料。

若要將 Lambda 函數部署至核心，請將函數新增至 Greengrass 群組（透過參考現有的 Lambda 函數）、設定函數的群組特定設定，然後部署群組。如果函數存取 AWS 服務，您也必須將任何必要的許可新增至 [Greengrass 群組角色](#)。

您可以設定參數來決定 Lambda 函數的執行方式，包括許可、隔離、記憶體限制等。如需詳細資訊，請參閱[the section called “控制 Greengrass Lambda 函數執行”](#)。

Note

這些設定也可讓您 AWS IoT Greengrass 在 Docker 容器中執行。如需詳細資訊，請參閱[the section called “AWS IoT Greengrass 在 Docker 容器中執行”](#)。

下表列出支援的[AWS Lambda 執行時間](#)及其可執行的 AWS IoT Greengrass Core 軟體版本。

語言或平台	GGC 版本
Python 3.8	1.11
Python 3.7	1.9 或更新版本
Python 2.7 *	1.0 或更新版本
Java 8	1.1 或更新版本
Node.js 12.x *	1.10 或更新版本
Node.js 8.10 *	1.9 或更新版本
Node.js 6.10 *	1.1 或更新版本
C、C++	1.6 或更新版本

* 您可以在支援的 版本上執行使用這些執行時間的 Lambda 函數 AWS IoT Greengrass，但您無法在其中建立這些函數 AWS Lambda。如果裝置上的執行時間與該函數指定的 AWS Lambda 執行時間不同，您可以在 `FunctionRuntimeOverride` 中使用 選擇您自己的執行時間 `FunctionDefinitionVersion`。如需詳細資訊，請參閱 [CreateFunctionDefinition](#)。如需支援執行時間的詳細資訊，請參閱《AWS Lambda 開發人員指南》中的[執行時間支援政策](#)。

適用於 Greengrass Lambda 函數 SDKs

AWS 提供三個軟體 SDKs，可供在 AWS IoT Greengrass 核心上執行的 Greengrass Lambda 函數使用。這些軟體開發套件包含在不同的套裝服務中，因此函數可以同時使用它們。若要在 Greengrass Lambda 函數中使用 SDK，請在您上傳到的 Lambda 函數部署套件中包含它 AWS Lambda。

AWS IoT Greengrass 核心 SDK

讓本機 Lambda 函數與核心互動，以便：

- 與 交換 MQTT 訊息 AWS IoT Core。
- 使用 Greengrass 群組中的連接器、用戶端裝置和其他 Lambda 函數交換 MQTT 訊息。
- 與本機陰影服務互動。
- 叫用其他本機 Lambda 函數。
- 存取[私密資源](#)。
- 與[串流管理員](#)交動。

AWS IoT Greengrass 在 GitHub 上以下列語言和平台提供 AWS IoT Greengrass 核心 SDK。

- [AWS IoT Greengrass 適用於 Java 的核心 SDK](#)
- [AWS IoT Greengrass 適用於 Node.js 的核心 SDK](#)
- [AWS IoT Greengrass 適用於 Python 的核心 SDK](#)
- [AWS IoT Greengrass 適用於 C 的核心 SDK](#)

若要在 Lambda 函數部署套件中包含 AWS IoT Greengrass 核心 SDK 相依性：

1. 下載符合 Lambda 函數執行時間之 AWS IoT Greengrass Core SDK 套件的語言或平台。
2. 解壓縮下載的封裝，以取得軟體開發套件。SDK 為 `greengrasssdk` 資料夾。
3. 在包含函數程式碼的 Lambda 函數部署套件 `greengrasssdk` 中包含。這是您在建立 Lambda 函數 AWS Lambda 時上傳至 的套件。

StreamManagerClient

只有下列 AWS IoT Greengrass 核心 SDKs 可用於[串流管理員](#)操作：

- Java 開發套件 (1.4.0 版或更新版本)
- Python SDK (1.5.0 版或更新版本)
- Node.js SDK (1.6.0 版或更新版本)

若要使用適用於 Python AWS IoT Greengrass 的核心 SDK 與串流管理員互動，您必須安裝 Python 3.7 或更新版本。您還必須安裝相依性，以包含在 Python Lambda 函數部署套件中：

1. 前往包含 `requirements.txt` 檔案的軟體開發套件目錄。這個檔案會列出相依性。
2. 安裝軟體開發套件相依性。例如，執行下列 `pip` 命令，將它們安裝在目前的目錄中：

```
pip install --target . -r requirements.txt
```

在 AWS IoT Greengrass 核心裝置上安裝適用於 Python 的核心 SDK

如果您正在執行 Python Lambda 函數，您也可以使用在 AWS IoT Greengrass 核心裝置上[pip](#)安裝適用於 Python 的核心 SDK。然後，您可以部署函數，而無需在 Lambda 函數部署套件中包含 SDK。如需詳細資訊，請參閱[greengrasssdk](#)。

此支援適用於具有大小限制的核心。我們建議您盡可能在 Lambda 函數部署套件中包含開發套件。

AWS IoT Greengrass Machine Learning SDK

讓本機 Lambda 函數使用部署到 Greengrass 核心作為 ML 資源的機器學習 (ML) 模型。Lambda 函數可以使用 SDK 來叫用本機推論服務，並與部署到核心做為連接器的本機推論服務互動。Lambda 函數和 ML 連接器也可以使用 SDK 將資料傳送至 ML Feedback 連接器，以進行上傳和發佈。如需詳細資訊，包括使用開發套件的程式碼範例，請參閱[the section called “ML 影像分類”](#)、[the section called “ML 物件偵測”](#)和[the section called “ML 意見回饋”](#)。

下表列出 SDK 版本支援的語言或平台，以及他們可以執行 AWS IoT Greengrass 的核心軟體版本。

SDK 版本	語言或平台	所需的 GGC 版本	Changelog
1.1.0	Python 3.7 或 2.7	1.9.3 或更新版本	新增 Python 3.7 支援和新的 feedback 用戶端。
1.0.0	Python 2.7	1.7 或更新版本	初始版本。

如需下載資訊，請參閱 [the section called “AWS IoT Greengrass ML 開發套件軟體”](#)。

AWS SDKs

讓本機 Lambda 函數直接呼叫 AWS 服務，例如 Amazon S3、AWS IoT、DynamoDB 和 AWS IoT Greengrass。若要在 Greengrass Lambda 函數中使用 AWS SDK，您必須在部署套件中包含它。當您在與 AWS IoT Greengrass Core AWS SDK 相同的套件中使用 SDK 時，請確定您的 Lambda 函數使用正確的命名空間。當核心離線時，Greengrass Lambda 函數無法與雲端服務通訊。

從[入門資源中心](#)下載 AWS SDKs。

如需建立部署套件的詳細資訊，請參閱《入門教學》[the section called “建立和封裝 Lambda 函數”](#)中的或《AWS Lambda 開發人員指南》中的[建立部署套件](#)。

遷移雲端型 Lambda 函數

AWS IoT Greengrass 核心 SDK 遵循 AWS SDK 程式設計模型，可讓您輕鬆地將為雲端開發的 Lambda 函數移植到在 AWS IoT Greengrass 核心上執行的 Lambda 函數。

例如，下列 Python Lambda 函數使用適用於 Python (Boto3) 的 AWS SDK 將訊息發佈至 some/topic 雲端中的主題：

```
import boto3

iot_client = boto3.client("iot-data")
response = iot_client.publish(
```

```
topic="some/topic", qos=0, payload="Some payload".encode()
)
```

若要移植 AWS IoT Greengrass 核心的 函數，請在 `import` 陳述式和 `client` 初始化中，將 `boto3` 模組名稱變更為 `greengrasssdk`，如下列範例所示：

```
import greengrasssdk

iot_client = greengrasssdk.client("iot-data")
iot_client.publish(topic="some/topic", qos=0, payload="Some payload".encode())
```

Note

AWS IoT Greengrass 核心 SDK 僅支援傳送 QoS = 0 的 MQTT 訊息。如需詳細資訊，請參閱 [the section called “訊息服務品質”](#)。

程式設計模型之間的相似性也可讓您在雲端開發 Lambda 函數，然後 AWS IoT Greengrass 輕鬆遷移到。 [Lambda 可執行檔](#) 不會在雲端執行，因此您無法在部署之前使用 AWS SDK 在雲端中進行開發。

依別名或版本參考 Lambda 函數

Greengrass 群組可以依別名（建議）或版本參考 Lambda 函數。使用別名可讓您更輕鬆地管理程式碼更新，因為您不必在更新函數程式碼時變更訂閱資料表或群組定義。反之，您只需將別名指向新的函數版本。別名會在群組部署期間解析為版本號碼。當您使用別名會更新解析的版本別名，此別名會指向部署的時間。

AWS IoT Greengrass 不支援 `$LATEST` 版本的 Lambda 別名。`$LATEST` 版本不受限於不可變、已發佈的函數版本，並且可以隨時變更，這與 AWS IoT Greengrass 版本不可變性原則相反。

透過程式碼變更來保持 Greengrass Lambda 函數更新的常見做法是使用 Greengrass 群組和訂閱 **PRODUCTION** 中名為 的別名。當您將 Lambda 函數的新版本提升為生產環境時，請將別名指向最新的穩定版本，然後重新部署群組。您也可以使用此方法轉返至舊版本。

使用群組特定的組態控制 Greengrass Lambda 函數的執行

AWS IoT Greengrass 提供 Greengrass Lambda 函數的雲端型管理。雖然 Lambda 函數的程式碼和相依性是使用 管理 AWS Lambda，但您可以設定 Lambda 函數在 Greengrass 群組中執行時的行為。

群組專屬組態設定

AWS IoT Greengrass 為 Greengrass Lambda 函數提供下列群組特定的組態設定。

系統使用者和群組

用來執行 Lambda 函數的存取身分。根據預設，Lambda 函數會以群組的[預設存取身分](#)執行。一般而言，這是標準的 AWS IoT Greengrass 系統帳戶 (ggc_user 和 ggc_group)。您可以變更設定，並選擇具有執行 Lambda 函數所需許可的使用者 ID 和群組 ID。您可以同時覆寫 UID 和 GID，或如果另一個欄位留白，可以只覆寫一個。此設定可讓您更精細地控制對裝置資源的存取。建議您使用適當的資源限制、檔案許可和磁碟配額來設定 Greengrass 硬體，以用於執行 Lambda 函數的使用者和群組。

此功能適用於 AWS IoT Greengrass Core v1.7 和更新版本。

Important

除非絕對必要，否則建議您避免將 Lambda 函數作為根函數執行。以根目錄執行會增加下列風險：

- 意外變更的風險，例如意外刪除關鍵檔案。
- 惡意個人對您的資料和裝置造成的風險。
- 當 Docker 容器搭配 `--net=host` 和執行時，容器逸出的風險 `UID=EUID=0`。

如果您確實需要以根身分執行，則必須更新 AWS IoT Greengrass 組態以啟用它。如需詳細資訊，請參閱[the section called “執行 Lambda 函數做為根”](#)。

系統使用者 ID（數字）

具有執行 Lambda 函數所需許可之使用者的使用者 ID。只有在您選擇以另一個使用者 ID/群組 ID 執行時，才能使用此設定。您可以在 AWS IoT Greengrass 核心裝置上使用 `getent passwd` 命令來查詢要用來執行 Lambda 函數的使用者 ID。

如果您使用相同的 UID 在 Greengrass 核心裝置上執行程序和 Lambda 函數，您的 Greengrass 群組角色可以授予程序臨時登入資料。這些程序可以使用 Greengrass 核心部署的臨時登入資料。

系統群組 ID (數字)

具有執行 Lambda 函數所需許可的群組的群組 ID。只有在您選擇以另一個使用者 ID/群組 ID 執行時，才能使用此設定。您可以在 AWS IoT Greengrass 核心裝置上使用 `getent group` 命令來查詢要用來執行 Lambda 函數的群組 ID。

Lambda 函數容器化

選擇 Lambda 函數是否使用群組的預設容器化執行，或指定應一律用於此 Lambda 函數的容器化。

Lambda 函數的容器化模式決定其隔離層級。

- 容器化 Lambda 函數會以 Greengrass 容器模式執行。Lambda 函數會在 AWS IoT Greengrass 容器內的隔離執行時間環境（或命名空間）中執行。
- 非容器化 Lambda 函數會以無容器模式執行。Lambda 函數會以一般 Linux 程序執行，而沒有任何隔離。

此功能適用於 AWS IoT Greengrass Core v1.7 和更新版本。

建議您在 Greengrass 容器中執行 Lambda 函數，除非您的使用案例要求它們在沒有容器化的情況下執行。當您的 Lambda 函數在 Greengrass 容器中執行時，您可以使用連接的本機和裝置資源，並取得隔離和提高安全性的優勢。請先參閱[the section called “選擇 Lambda 函數容器化時的考量事項”](#)，再變更 containerization (容器化)。

Note

若要在不啟用裝置核心命名空間和 cgroup 的情況下執行，所有 Lambda 函數都必須在沒有容器化的情況下執行。只要設定群組的預設容器化，即可輕鬆達成此目標。如需相關資訊，請參閱[the section called “為群組中的 Lambda 函數設定預設容器化”](#)。

Memory limit (記憶體限制)

函數所需記憶體的分配。預設為 16 MB。

Note

當您將 Lambda 函數變更為在沒有容器化的情況下執行時，記憶體限制設定會變成無法使用。在沒有容器化的情況下執行的 Lambda 函數沒有記憶體限制。當您將 Lambda 函數或群組預設容器化設定變更為在沒有容器化的情況下執行時，會捨棄記憶體限制設定。

Timeout (逾時)

在函數或要求終止前的時間值。預設為 3 秒。

固定

Lambda 函數生命週期可以是隨需或長期。預設為隨需。

呼叫時，隨需 Lambda 函數會在新的或重複使用的容器中啟動。請求任何可以處理的函數。Lambda 函數會在 AWS IoT Greengrass 啟動後自動啟動，並在自己的容器（或沙盒）中持續執行。所有請求函數的要求皆由相同的容器處理。如需詳細資訊，請參閱[the section called “生命週期組態”](#)。

Read access to /sys directory (對 /sys 目錄的讀取存取)

不管該函數可以存取主機的 /sys 資料夾。當函數必須讀取 /sys 的裝置資訊時，請使用此函數。預設值為 false。

Note

當您在沒有容器化的情況下執行 Lambda 函數時，無法使用此設定。當您將 Lambda 函數變更為在沒有容器化的情況下執行時，會捨棄此設定的值。

編碼類型

函數輸入承載預期的編碼類型，可能是 JSON 或二進位。預設值為 JSON。

從 AWS IoT Greengrass Core Software v1.5.0 和 AWS IoT Greengrass Core SDK v1.1.0 開始，即可支援二進位編碼類型。接受二進位輸入資料可幫助函數與裝置資料互動，因為裝置硬體的限制讓他們自身難以或無法建構 JSON 資料類型。

Note

[Lambda 可執行檔](#)僅支援二進位編碼類型，不支援 JSON。

程序引數

命令列引數會在執行時傳遞至 Lambda 函數。

環境變數

金鑰值組可動態地將設定傳給函數程式碼和二進位程式庫。本機環境變數會執行的方式與 [AWS Lambda 函數環境變數](#) 一樣，但是，不同在於可在核心環境取得。

資源存取政策

允許 Lambda 函數存取的最多 10 個[本機資源](#)、[秘密資源](#)和[機器學習資源](#)的清單，以及對應的 read-only 或 read-write 許可。在主控台中，這些附屬資源會列在資源索引標籤的群組組態頁面上。

[容器化模式](#)會影響 Lambda 函數如何存取本機裝置和磁碟區資源，以及機器學習資源。

- 非容器化 Lambda 函數必須透過核心裝置上的檔案系統直接存取本機裝置和磁碟區資源。
- 若要允許非容器化 Lambda 函數存取 Greengrass 群組中的機器學習資源，您必須在機器學習資源上設定資源擁有者和存取許可屬性。如需詳細資訊，請參閱[the section called “存取機器學習資源”](#)。

如需有關使用 AWS IoT Greengrass API 為使用者定義的 Lambda 函數設定群組特定組態設定的資訊，請參閱 AWS IoT Greengrass Version 1 《API 參考》中的 [CreateFunctionDefinition](#) 或《AWS CLI 命令參考》中的 [create-function-definition](#)。若要將 Lambda 函數部署至 Greengrass 核心，請建立包含函數的函數定義版本，建立參考函數定義版本和其他群組元件的群組版本，然後[部署群組](#)。

執行 Lambda 函數做為根

此功能適用於 AWS IoT Greengrass Core v1.7 和更新版本。

您必須先更新 AWS IoT Greengrass 組態以啟用支援，才能執行一或多個 Lambda 函數做為根目錄。支援在根目錄關閉時執行 Lambda 函數。如果您嘗試部署 Lambda 函數並以根 (UID 和 GID 為 0) 執行，且尚未更新 AWS IoT Greengrass 組態，則部署會失敗。類似以下的錯誤會出現在執行時間日誌 ([greengrass_root/ggc/var/log/system/runtime.log](#)) 中：

```
lambda(s)
[list of function arns] are configured to run as root while Greengrass is not
configured to run lambdas with root permissions
```

Important

除非絕對必要，否則建議您避免將 Lambda 函數作為根函數執行。以根目錄執行會增加下列風險：

- 意外變更的風險，例如意外刪除關鍵檔案。
- 惡意個人對您的資料和裝置造成的風險。
- 當 Docker 容器搭配 `--net=host` 和執行時，容器逸出的風險 `UID=EUID=0`。

允許 Lambda 函數以根目錄執行

1. 在您的裝置上 AWS IoT Greengrass，導覽至 *greengrass-root*/config 資料夾。

 Note

根據預設，*greengrass-root* 為 /greengrass 目錄。

2. 編輯 config.json 檔案，以將 "allowFunctionsToRunAsRoot" : "yes" 新增到 runtime 欄位。例如：

```
{
  "coreThing" : {
    ...
  },
  "runtime" : {
    ...
    "allowFunctionsToRunAsRoot" : "yes"
  },
  ...
}
```

3. 使用以下命令重新啟動 AWS IoT Greengrass：

```
cd /greengrass/ggc/core
sudo ./greengrassd restart
```

現在，您可以將 Lambda 函數的使用者 ID 和群組 ID (UID/GID) 設定為 0，以將該 Lambda 函數執行為根。

如果您想要不允許 Lambda 函數做為根函數執行 AWS IoT Greengrass，您可以將 的值變更為 "allowFunctionsToRunAsRoot""no"並重新啟動。

選擇 Lambda 函數容器化時的考量事項

此功能適用於 AWS IoT Greengrass Core v1.7 和更新版本。

根據預設，Lambda 函數會在 AWS IoT Greengrass 容器內執行。該容器會隔離您的函數和主機，為容器內的主機和函數提供更多安全性。

建議您在 Greengrass 容器中執行 Lambda 函數，除非您的使用案例要求它們在沒有容器化的情況下執行。透過在 Greengrass 容器中執行 Lambda 函數，您可以更好地控制限制對資源的存取。

以下是一些在無容器情況下執行的範例使用案例：

- 您想要 AWS IoT Greengrass 在不支援容器模式的裝置上執行（例如，因為您使用的是特殊 Linux 發行版本或核心版本太舊）。
- 您想要在另一個容器環境中使用自己的 OverlayFS 執行 Lambda 函數，但在 Greengrass 容器中執行時遇到 OverlayFS 衝突。
- 您需要存取無法在部署期間確定路徑的本機資源，或其路徑會在部署後變更，例如插入式裝置。
- 您有一個已寫入程序的舊版應用程式，且在以容器化 Lambda 函數執行它時遇到問題。

容器化差異

容器化	備註
Greengrass 容器	• 當您在 Greengrass 容器中執行 Lambda 函數時，所有 AWS IoT Greengrass 功能都可使用。 • 在 Greengrass 容器中執行的 Lambda 函數無法存取其他 Lambda 函數的部署程式碼，即使它們使用相同的群組 ID 執行。換句話說，您的 Lambda 函數會以更大的隔離彼此執行。 • 由於在 AWS IoT Greengrass 容器中執行的 Lambda 函數的所有子程序都會在與 Lambda 函數相同的容器中執行，因此當 Lambda 函數終止時，子程序就會終止。
沒有容器	• 下列功能不適用於非容器化 Lambda 函數： • Lambda 函數記憶體限制。

容器化	備註
	• 本機裝置和磁碟區資源。您必須在核心裝置上直接存取這些資源，而不是以 Greengrass 群組的成員身分存取。 • 如果您的非容器化 Lambda 函數存取機器學習資源，您必須識別資源擁有者，並在資源上設定存取許可，而不是在 Lambda 函數上。這需要 AWS IoT Greengrass Core 軟體 1.10 版或更新版本。如需詳細資訊，請參閱the section called “存取機器學習資源”。 • Lambda 函數對使用相同群組 ID 執行之其他 Lambda 函數的部署程式碼具有唯讀存取權。 • 在不同的程序工作階段中或使用覆寫的 SIGHUP（訊號掛斷）處理常式產生子程序的 Lambda 函數，例如使用 nohup 公用程式，不會在父 Lambda 函數終止 AWS IoT Greengrass 時由自動終止。

 Note

Greengrass 群組的預設容器化設定不適用於[連接器](#)。

變更 Lambda 函數的容器化可能會在您部署時造成問題。如果您已將本機資源指派給新的容器化設定不再可用的 Lambda 函數，則部署會失敗。

- 當您將 Lambda 函數從 Greengrass 容器中執行變更為在沒有容器化的情況下執行時，會捨棄函數的記憶體限制。您必須直接存取檔案系統，而不是使用連接的本機資源。您必須在部署前移除所有連接的資源。
- 當您將 Lambda 函數從在沒有容器化的情況下執行變更為在容器中執行時，您的 Lambda 函數會失去檔案系統的直接存取權。您必須定義各函數的記憶體上限，或接受預設值 16 MB。您可以在部署之前為每個 Lambda 函數設定這些設定。

變更 Lambda 函數的容器化設定

1. 在 AWS IoT 主控台導覽窗格的管理下，展開 Greengrass 裝置，然後選擇群組 (V1)。
2. 選擇包含您要變更其設定之 Lambda 函數的群組。
3. 選擇 Lambda 函數索引標籤。
4. 在您要變更的 Lambda 函數上，選擇省略符號 (...)，然後選擇編輯組態。
5. 變更容器化設定。如果您將 Lambda 函數設定為在 Greengrass 容器中執行，您還必須設定記憶體限制和對 /sys 目錄的讀取存取權。
6. 選擇儲存，然後選擇確認以儲存對 Lambda 函數的變更。

變更會在群組部署後生效。

您也可以使用 AWS IoT Greengrass API 參考中的 [CreateFunctionDefinition](#) 和 [CreateFunctionDefinitionVersion](#)。如果您要變更容器化設定，請務必也要更新其他參數。例如，如果您要從在 Greengrass 容器中執行 Lambda 函數變更為在沒有容器化的情況下執行，請務必清除 MemorySize 參數。

判斷您 Greengrass 裝置支援的隔離模式

您可以使用 AWS IoT Greengrass 相依性檢查程式來判斷 Greengrass 裝置支援哪些隔離模式 (Greengrass 容器/無容器)。

執行 AWS IoT Greengrass 相依性檢查程式

1. 從 [GitHub 儲存庫](#) 下載並執行 AWS IoT Greengrass 相依性檢查程式。

```
wget https://github.com/aws-samples/aws-greengrass-samples/raw/master/greengrass-dependency-checker-GGCv1.11.x.zip
unzip greengrass-dependency-checker-GGCv1.11.x.zip
cd greengrass-dependency-checker-GGCv1.11.x
sudo modprobe configs
sudo ./check_ggc_dependencies | more
```

2. 在 more 出現的位置，按 Spacebar 鍵可顯示另一頁文字。

如需 modprobe 命令的資訊，請在終端機中執行 man modprobe。

為群組中的 Lambda 函數設定預設存取身分

此功能適用於 AWS IoT Greengrass Core v1.8 和更新版本。

如需進一步控制裝置資源的存取，您可以設定用於在群組中執行 Lambda 函數的預設存取身分。此設定會決定在核心裝置上執行 Lambda 函數時授予的預設許可。若要覆寫群組中的個別函數設定，您可以使用函數的 Run as (執行身分) 屬性。如需詳細資訊，請參閱[執行身分](#)。

此群組層級設定也用於執行基礎 AWS IoT Greengrass Core 軟體。這包含管理操作的系統 Lambda 函數，例如訊息路由、本機陰影同步和自動 IP 地址偵測。

預設存取身分可設定為做為標準 AWS IoT Greengrass 系統帳戶 (ggc_user 和 ggc_group) 執行，或使用其他使用者或群組的許可。建議您針對其許可用於執行使用者定義或系統 Lambda 函數的任何使用者和群組，使用適當的資源限制、檔案許可和磁碟配額來設定 Greengrass 硬體。

修改 AWS IoT Greengrass 群組的預設存取身分

1. 在 AWS IoT 主控台導覽窗格的管理下，展開 Greengrass 裝置，然後選擇群組 (V1)。
2. 選擇您希望變更設定的群組。
3. 選擇 Lambda 函數索引標籤，然後在預設 Lambda 函數執行期環境區段下，選擇編輯。
4. 在編輯預設 Lambda 函數執行期環境頁面的預設系統使用者和群組下，選擇另一個使用者 ID/群組 ID。

當您選擇此選項時，系統使用者 ID (數字) 和系統群組 ID (數字) 欄位會顯示。

5. 輸入使用者 ID、群組 ID 或兩者。如果您將欄位留白，則會使用個別的 Greengrass 系統帳戶 (ggc_user 或 ggc_group)。
 - 在系統使用者 ID (數字) 中，輸入具有預設要在群組中執行 Lambda 函數之許可的使用者 ID。您可以使用 AWS IoT Greengrass 裝置上的 `getent passwd` 命令查看使用者 ID。
 - 針對系統群組 ID (數字)，輸入群組的群組 ID，該群組具有預設要在群組中執行 Lambda 函數的許可。您可以使用 AWS IoT Greengrass 裝置上的 `getent group` 命令查看群組 ID。

Important

執行身分為根使用者會增加資料和裝置的風險。除非您的商業案例要求，否則請勿以根執行 (UID/GID = 0)。如需詳細資訊，請參閱[the section called “執行 Lambda 函數做為根”](#)。

變更會在群組部署後生效。

為群組中的 Lambda 函數設定預設容器化

此功能適用於 AWS IoT Greengrass Core v1.7 和更新版本。

Greengrass 群組的容器化設定會決定群組中 Lambda 函數的預設容器化。

- 在 Greengrass 容器模式中，Lambda 函數預設會在 AWS IoT Greengrass 容器內的隔離執行時間環境中執行。
- 在無容器模式中，Lambda 函數預設會以一般 Linux 程序執行。

您可以修改群組設定，以指定群組中 Lambda 函數的預設容器化。如果您想要 Lambda 函數以與群組預設值不同的容器化執行，您可以覆寫群組中一或多個 Lambda 函數的此設定。請先參閱[the section called “選擇 Lambda 函數容器化時的考量事項”](#)，再變更容器化設定。

Important

如果您想要變更群組的預設容器化，但有一或多個函數使用不同的容器化，請在變更群組設定之前變更 Lambda 函數的設定。如果您先變更群組容器化設定，就會捨棄 Memory limit (記憶體限制) 和 Read access to /sys directory (對 /sys 目錄的讀取存取) 設定。

修改 AWS IoT Greengrass 群組的容器化設定

1. 在 AWS IoT 主控台導覽窗格的管理下，展開 Greengrass 裝置，然後選擇群組 (V1)。
2. 選擇您希望變更設定的群組。
3. 選擇 Lambda 函數索引標籤。
4. 在預設 Lambda 函數執行期環境中，選擇編輯。
5. 在編輯預設 Lambda 函數執行期環境頁面的預設 Lambda 函數容器化下，變更容器化設定。
6. 選擇儲存。

變更會在群組部署後生效。

Greengrass Lambda 函數的通訊流程

Greengrass Lambda 函數支援數種與 AWS IoT Greengrass 群組的其他成員、本機服務和雲端服務（包括 AWS 服務）通訊的方法。

使用 MQTT 訊息進行通訊

Lambda 函數可以使用由訂閱控制的發佈訂閱模式來傳送和接收 MQTT 訊息。

此通訊流程可讓 Lambda 函數與下列實體交換訊息：

- 群組中的用戶端裝置。
- 該群組中的連接器。
- 群組中的其他 Lambda 函數。
- AWS IoT。
- 本機裝置陰影服務。

訂閱會定義將訊息從來源路由到目標時所用的訊息來源、訊息目標和主題 (或主體)。發佈至 Lambda 函數的訊息會傳遞至函數的已註冊處理常式。訂閱啟用更多安全性，和提供可預測的互動。如需詳細資訊，請參閱[the section called “MQTT 簡訊工作流程中的受管訂閱”](#)。

Note

當核心離線時，Greengrass Lambda 函數可以與用戶端裝置、連接器、其他函數和本機影子交換訊息，但訊息 AWS IoT 會排入佇列。如需詳細資訊，請參閱[the section called “MQTT 訊息佇列”](#)。

其他通訊流程

- 為了在核心裝置上與本機裝置和磁碟區資源和機器學習模型互動，Greengrass Lambda 函數會使用平台特定的作業系統介面。例如，您可以在 Python 函數的 [os](#) 模組中使用 open 方法。若要允許函數存取資源，該函數必須隸屬於資源並授與 read-only 或 read-write 許可。如需詳細資訊，包括 AWS IoT Greengrass 核心版本可用性，請參閱 [存取本機資源](#) 和 [the section called “從 Lambda 函數程式碼存取機器學習資源”](#)。

Note

如果您在沒有容器化的情況下執行 Lambda 函數，則無法使用連接的本機裝置和磁碟區資源，而且必須直接存取這些資源。

- Lambda 函數可以使用 AWS IoT Greengrass 核心 SDK 中的 Lambda 用戶端來叫用 Greengrass 群組中的其他 Lambda 函數。
- Lambda 函數可以使用 AWS SDK 與 AWS 服務通訊。如需詳細資訊，請參閱 [AWS SDK](#)。
- Lambda 函數可以使用第三方界面與外部雲端服務通訊，類似於雲端型 Lambda 函數。

Note

當核心離線時，Greengrass Lambda 函數無法與 AWS 或其他雲端服務通訊。

擷取輸入 MQTT 主題 (或主旨)

AWS IoT Greengrass 使用訂閱來控制用戶端裝置、Lambda 函數和群組中的連接器，以及 AWS IoT 或本機影子服務之間的 MQTT 訊息交換。訂閱會定義訊息來源、訊息目標，以及用於路由訊息的 MQTT 主題。當目標為 Lambda 函數時，當來源發佈訊息時，會叫用函數的處理常式。如需詳細資訊，請參閱 [the section called “使用 MQTT 訊息進行通訊”](#)。

下列範例顯示 Lambda 函數如何從傳遞給處理常式 context 的取得輸入主題。方式是從內容階層 (context.client_context.custom['subject']) 存取 subject 金鑰。範例也會剖析輸入 JSON 訊息，並發佈剖析的主題和訊息。

Note

在 AWS IoT Greengrass API 中，[訂閱](#)的主題由 subject 屬性表示。

```
import greengrasssdk
import logging

client = greengrasssdk.client('iot-data')
```



```
OUTPUT_TOPIC = 'test/topic_results'

def get_input_topic(context):
    try:
        topic = context.client_context.custom['subject']
    except Exception as e:
        logging.error('Topic could not be parsed. ' + repr(e))
    return topic

def get_input_message(event):
    try:
        message = event['test-key']
    except Exception as e:
        logging.error('Message could not be parsed. ' + repr(e))
    return message

def function_handler(event, context):
    try:
        input_topic = get_input_topic(context)
        input_message = get_input_message(event)
        response = 'Invoked on topic "%s" with message "%s"' % (input_topic,
input_message)
        logging.info(response)
    except Exception as e:
        logging.error(e)

    client.publish(topic=OUTPUT_TOPIC, payload=response)

    return
```

若要測試函數，請使用預設的組態設定將其新增至群組。接著，新增以下訂閱並部署群組。如需說明，請參閱[the section called “單元 3 \(第 1 部分\)：上的 Lambda 函數 AWS IoT Greengrass”](#)。

target

灑

篩

選

條

件

test/

topic

input_message

雲

端)

test/

topic

results

雲

端)

部署完成後，呼叫該函數。

1. 在 AWS IoT 主控台中，開啟 MQTT 測試用戶端頁面。
2. 選取訂閱test/topic_results主題索引標籤來訂閱主題。
3. 選取發佈至test/input_message主題索引標籤，將訊息發佈至主題。在此範例中，您必須於JSON 訊息包含 test-key 屬性。

```
{
  "test-key": "Some string value"
}
```

如果成功，函數會將輸入主題和訊息字串發佈至 test/topic_results 主題。

Greengrass Lambda 函數的生命週期組態

Greengrass Lambda 函數生命週期會決定函數何時啟動，以及它如何建立和使用容器。生命週期也可決定在函數處理常式外所保留的變數和預先處理邏輯。

AWS IoT Greengrass 支援隨需（預設）或長期生命週期：

- 隨需函數，在沒有需要執行工作時，其被呼叫和停用會讓此隨需函數啟動。叫用函數會建立分離的容器（或沙盒）以處理呼叫，除非已有可重複使用的現有容器。傳送到函數的資料可能會透過任何容器拉出。

隨需函數的多個呼叫可以平行執行。

每當建立新容器時，在函數處理常式外定義的變數或預先處理的邏輯將不將予以保留。

- 長期（或固定）函數會在 AWS IoT Greengrass 核心啟動並在單一容器中執行時自動啟動。傳送到函數的所有資料可能會透過一樣的容器拉出。

會佇列多個呼叫，直到先前的呼叫已執行。

在函數處理常式外定義的變數或預先處理的邏輯，每次呼叫時會保留於函數處理常式。

當您需要在沒有任何初始輸入的情況下開始執行工作時，長期 Lambda 函數很有用。例如，當函數開始接收裝置資料時，長期函數可以載入和啟動就緒的機器學習模型。

Note

請記住長期函數有與其相關的呼叫處理常式之逾時。如果您想要無限期地執行程式碼，則您必須在處理常式外啟動此程式碼。請確定在處理常式外沒有封鎖程式碼，可以預防完成其初始化的函數。

除非核心停止（例如，在群組部署或裝置重新啟動期間）或函數進入錯誤狀態（例如處理常式逾時、未攔截的例外狀況，或超過其記憶體限制），否則這些函數會執行。

如需容器重複使用的詳細資訊，請參閱 AWS 運算部落格中的[了解中的容器重複使用 AWS Lambda](#)。

Lambda 可執行檔

此功能適用於 AWS IoT Greengrass Core v1.6 和更新版本。

Lambda 可執行檔是一種 Greengrass Lambda 函數，可用於在核心環境中執行二進位程式碼。它可讓您原生執行裝置特定的功能，並受益於編譯程式碼的較小足跡。事件可以叫用 Lambda 可執行檔、叫用其他函數，以及存取本機資源。

Lambda 可執行檔僅支援二進位編碼類型（不支援 JSON），否則您可以在 Greengrass 群組中管理它們，並像其他 Greengrass Lambda 函數一樣進行部署。不過，建立 Lambda 可執行檔的程序與建立 Python、Java 和 Node.js Lambda 函數不同：

- 您無法使用 AWS Lambda 主控台建立（或管理）Lambda 可執行檔。您只能使用 AWS Lambda API 建立 Lambda 可執行檔。
- 您可以將函數程式碼上傳到 AWS Lambda 做為編譯的可執行檔，其中包含[AWS IoT Greengrass 適用於 C 的核心 SDK](#)。
- 您可以指定可執行檔的名稱為函數處理常式。

Lambda 可執行檔必須在其函數程式碼中實作特定呼叫和程式設計模式。例如，main 方法必須：

- 呼叫 `gg_global_init` 以初始化 Greengrass 內部全域變數。在建立任何執行緒之前，以及在呼叫任何其他 AWS IoT Greengrass Core SDK 函數之前，必須呼叫此函數。
- 呼叫 `gg_runtime_start` 以向 Greengrass Lambda 執行時間註冊函數處理常式。此函數必須在初始化期間呼叫。呼叫此函數會讓執行時間使用目前的執行緒。此選擇性使用的 `GG_RT_OPT_ASYNC` 參數通知該函數不要進行封鎖，反而請它為執行時間建立新的執行緒。此函數使用 `SIGTERM` 處理常式。

下列程式碼片段是 GitHub 上 [simple_handler.c](#) 程式碼範例的 main 方法。

```
int main() {
    gg_error err = GGE_SUCCESS;

    err = gg_global_init(0);
    if(err) {
        gg_log(GG_LOG_ERROR, "gg_global_init failed %d", err);
        goto cleanup;
    }

    gg_runtime_start(handler, 0);

cleanup:
    return -1;
}
```

如需求、限制條件和其他實作詳細資訊的詳細資訊，請參閱[AWS IoT Greengrass 適用於 C 的核心 SDK](#)。

建立 Lambda 可執行檔

編譯程式碼與 SDK 之後，請使用 AWS Lambda API 來建立 Lambda 函數並上傳已編譯的可執行檔。

Note

必須使用 C89 相容編譯器編譯您的函數。

下列範例使用 [create-function](#) CLI 命令來建立 Lambda 可執行檔。此命令指定：

- 處理常式的可執行檔名稱。這必須為您編譯的可執行檔之確切名稱。
- .zip 檔案路徑含有編譯的可執行檔。
- 執行階段的 `arn:aws:greengrass:::runtime/function/executable`。這是所有 Lambda 可執行檔的執行時間。

Note

對於 `role`，您可以指定任何 Lambda 執行角色的 ARN。AWS IoT Greengrass 不使用此角色，但需要參數才能建立函數。如需 Lambda 執行角色的詳細資訊，請參閱《AWS Lambda 開發人員指南》中的[AWS Lambda 許可模型](#)。

```
aws lambda create-function \  
--region aws-region \  
--function-name function-name \  
--handler executable-name \  
--role role-arn \  
--zip-file fileb://file-name.zip \  
--runtime arn:aws:greengrass:::runtime/function/executable
```

接著，使用 AWS Lambda API 發佈版本並建立別名。

- 使用 [publish-version](#) 發佈函數版本。

```
aws lambda publish-version \  
--function-name function-name \  
--region aws-region
```

- 使用 [create-alias](#) 來建立別名，以指向您剛發佈的版本。建議您在將 Lambda 函數新增至 Greengrass 群組時，依別名參考它們。

```
aws lambda create-alias \  
--function-name function-name \  
--name alias-name \  
--function-version version-number \  
--region aws-region
```

Note

AWS Lambda 主控台不會顯示 Lambda 可執行檔。若要更新函數程式碼，您必須使用 AWS Lambda API。

然後，將 Lambda 可執行檔新增至 Greengrass 群組，將其設定為在其群組特定設定中接受二進位輸入資料，然後部署群組。您可以在 AWS IoT Greengrass 主控台或使用 AWS IoT Greengrass API 執行此操作。

AWS IoT Greengrass 在 Docker 容器中執行

AWS IoT Greengrass 可以設定為在 [Docker](#) 容器中執行。

您可以透過已安裝 AWS IoT Greengrass Core 軟體和相依性的 [Amazon CloudFront](#) 下載 Dockerfile。若要修改 Docker 映像檔以在不同平台架構上執行或減少 Docker 映像檔的大小，請參閱 Docker 套件下載中的 README 檔案。

為了協助您開始試驗 AWS IoT Greengrass，AWS 也提供已安裝 AWS IoT Greengrass Core 軟體和相依性的預先建置 Docker 映像。您可以從 [Docker Hub](#) 或 [Amazon Elastic Container Registry](#) (Amazon ECR) 下載映像。這些預先建置映像使用 Amazon Linux 2 (x86_64) 和 Alpine Linux (x86_64、Arm v7l 貨 AArch64) 的基本映像。

Important

2022 年 6 月 30 日，發佈至 Amazon Elastic Container Registry (Amazon ECR) 和 Docker Hub 的 AWS IoT Greengrass Core 軟體 v1.x Docker 映像的 AWS IoT Greengrass 結束維護。您可以繼續從 Amazon ECR 和 Docker Hub 下載這些 Docker 映像，直到 2023 年 6 月 30 日，也就是維護結束後 1 年。不過，在 2022 年 6 月 30 日結束維護之後，AWS IoT

Greengrass Core 軟體 v1.x Docker 映像將不再收到安全修補程式或錯誤修正。如果您執行依賴這些 Docker 映像的生產工作負載，建議您使用 AWS IoT Greengrass 提供的 Dockerfile 建置自己的 Docker 映像。如需詳細資訊，請參閱[AWS IoT Greengrass Docker 軟體](#)。

本主題說明如何從 Amazon ECR 下載 AWS IoT Greengrass Docker 映像，並在 Windows、macOS 或 Linux (x86_64) 平台上執行。本主題包含以下步驟：

1. [從 Amazon ECR 取得 AWS IoT Greengrass 容器映像](#)
2. [建立及設定 Greengrass 群組和核心](#)
3. [在 AWS IoT Greengrass 本機執行](#)
4. [為群組的容器化設定「No container」\(無容器\)](#)
5. [將 Lambda 函數部署至 Docker 容器](#)
6. [\(選用\) 在 Docker 容器中部署與 Greengrass 互動的用戶端裝置](#)

當您 AWS IoT Greengrass 在 Docker 容器中執行時，不支援下列功能：

- 在 Greengrass 容器模式下執行[的連接器](#)。若要在 Docker 容器中執行連接器，連接器必須以 No container (無容器) 模式執行。若要尋找支援 No container (無容器) 模式的連接器，請參閱 [the section called “AWS提供的 Greengrass 連接器”](#)。其中一些連接器具有隔離模式參數，您必須設定為 No container (無容器)。
- [本機裝置和磁碟區資源](#)。您在 Docker 容器中執行的使用者定義 Lambda 函數必須直接存取核心上的裝置和磁碟區。

當 Greengrass 群組的 Lambda 執行期環境設定為[無容器](#)時，不支援這些功能，這是 AWS IoT Greengrass 在 Docker 容器中執行的必要項目。

先決條件

開始本教學課程之前，您必須執行下列動作。

- 您必須根據您選擇的 AWS Command Line Interface (AWS CLI) 版本，在主機電腦上安裝下列軟體和版本。

AWS CLI version 2

- [Docker](#) 18.09 版或更新版本。舊版也可以運作，但我們建議使用 18.09 或更新版本。

- AWS CLI 2.0.0 版或更新版本。
- 若要安裝第 2 AWS CLI 版，請參閱[安裝第 2 AWS CLI 版](#)。
- 若要設定 AWS CLI，請參閱[設定 AWS CLI](#)。

 Note

若要在 Windows 電腦上升級至更新 AWS CLI 版本 2，您必須重複 [MSI 安裝](#) 程序。

AWS CLI version 1

- [Docker](#) 18.09 版或更新版本。舊版也可以運作，但我們建議使用 18.09 或更新版本。
- [Python](#) 3.6 版或更新版本。
- [pip](#) 版本 18.1 或更新版本。
- AWS CLI 1.17.10 版或更新版本
- 若要安裝第 1 AWS CLI 版，請參閱[安裝第 1 AWS CLI 版](#)。
- 若要設定 AWS CLI，請參閱[設定 AWS CLI](#)。
- 若要升級至最新版本的第 1 AWS CLI 版，請執行下列命令。

```
pip install awscli --upgrade --user
```

 Note

如果您在 Windows 上使用 AWS CLI 版本 1 的 [MSI 安裝](#)，請注意下列事項：

- 如果第 1 AWS CLI 版安裝無法安裝 botocore，請嘗試使用 [Python 和 pip 安裝](#)。
- 若要升級至更新 AWS CLI 版本 1，您必須重複 MSI 安裝程序。

- 若要存取 Amazon Elastic Container Registry (Amazon ECR) 資源，您必須授予下列許可。
- Amazon ECR 要求使用者透過 AWS Identity and Access Management (IAM) 政策授予 `ecr:GetAuthorizationToken` 許可，才能向登錄檔進行身分驗證，並從 Amazon ECR 儲存庫推送或提取映像。如需詳細資訊，請參閱《[Amazon Elastic Container Registry 使用者指南](#)》中的 [Amazon ECR 儲存庫政策範例](#)和[存取一個 Amazon ECR 儲存庫](#)。

步驟 1：從 Amazon ECR 取得 AWS IoT Greengrass 容器映像

AWS 提供已安裝 AWS IoT Greengrass Core 軟體的 Docker 映像。

從 Amazon ECR 取得 AWS IoT Greengrass 容器映像

⚠ Warning

從 AWS IoT Greengrass 核心軟體的 1.11.6 版開始，Greengrass Docker 映像不再包含 Python 2.7，因為 Python 2.7 在 2020 年達到end-of-life，且不再收到安全性更新。如果您選擇更新這些 Docker 映像，建議您先驗證應用程式是否使用新的 Docker 映像，再將更新部署到生產裝置。如果您為使用 Greengrass Docker 映像的應用程式需要 Python 2.7，您可以修改 Greengrass Dockerfile 以包含應用程式的 Python 2.7。

如需示範如何從 Amazon ECR 提取latest映像的步驟，請選擇您的作業系統：

提取容器映像 (Linux)

在您的電腦終端機上執行下列命令。

1. 登入 Amazon ECR 中的 AWS IoT Greengrass 登錄檔。

```
aws ecr get-login-password --region us-west-2 | docker login --username AWS --password-stdin https://216483018798.dkr.ecr.us-west-2.amazonaws.com
```

如果成功，輸出會顯示 Login Succeeded。

2. 擷取 AWS IoT Greengrass 容器映像。

```
docker pull 216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

i Note

latest 映像包含安裝在 Amazon Linux 2 基礎映像上最新穩定版本的 AWS IoT Greengrass Core 軟體。您也可以從儲存庫提取其他映像。若要尋找所有可用的映像，請檢查 [Docker Hub](#) 上的標籤頁面或使用 `aws ecr list-images` 命令。例如：

```
aws ecr list-images --region us-west-2 --registry-id 216483018798 --repository-name aws-iot-greengrass
```

3. 啟用 symlink 和 hardlink 保護。如果您正在實驗 AWS IoT Greengrass 容器中執行，您只能啟用目前開機的設定。

Note

您可能需要使用 `sudo` 來執行這些命令。

- 僅針對目前的開機裝置啟用設定：

```
echo 1 > /proc/sys/fs/protected_hardlinks
echo 1 > /proc/sys/fs/protected_symlinks
```

- 使設定在重新啟動時保留：

```
echo '# AWS IoT Greengrass' >> /etc/sysctl.conf
echo 'fs.protected_hardlinks = 1' >> /etc/sysctl.conf
echo 'fs.protected_symlinks = 1' >> /etc/sysctl.conf

sysctl -p
```

4. 啟用 IPv4 網路轉送，這是 AWS IoT Greengrass 雲端部署和 MQTT 通訊在 Linux 上運作的必要條件。在 `/etc/sysctl.conf` 檔案中，將 `net.ipv4.ip_forward` 設為 1，然後重新載入 `sysctls`。

```
sudo nano /etc/sysctl.conf
# set this net.ipv4.ip_forward = 1
sudo sysctl -p
```

Note

您可以使用偏好的編輯器來取代 `nano`。

提取容器映像 (macOS)

在您的電腦終端機上執行下列命令。

1. 登入 Amazon ECR 中的 AWS IoT Greengrass 登錄檔。

```
aws ecr get-login-password --region us-west-2 | docker login --username AWS --password-stdin https://216483018798.dkr.ecr.us-west-2.amazonaws.com
```

如果成功，輸出會顯示 Login Succeeded。

2. 擷取 AWS IoT Greengrass 容器映像。

```
docker pull 216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

Note

latest 映像包含安裝在 Amazon Linux 2 基礎映像上最新穩定版本的 AWS IoT Greengrass Core 軟體。您也可以從儲存庫提取其他映像。若要尋找所有可用的映像，請檢查 [Docker Hub](https://hub.docker.com/r/aws-iot-greengrass) 上的標籤頁面或使用 `aws ecr list-images` 命令。例如：

```
aws ecr list-images --region us-west-2 --registry-id 216483018798 --  
repository-name aws-iot-greengrass
```

提取容器映像 (Windows)

在命令提示中執行下列命令：要在 Windows 上使用 Docker 命令，必須先執行 Docker 桌面。

1. 登入 Amazon ECR 中的 AWS IoT Greengrass 登錄檔。

```
aws ecr get-login-password --region us-west-2 | docker login --username AWS --  
password-stdin https://216483018798.dkr.ecr.us-west-2.amazonaws.com
```

如果成功，輸出會顯示 Login Succeeded。

2. 擷取 AWS IoT Greengrass 容器映像。

```
docker pull 216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

Note

latest 映像包含安裝在 Amazon Linux 2 基礎映像上最新穩定版本的 AWS IoT Greengrass Core 軟體。您也可以從儲存庫提取其他映像。若要尋找所有可用的映像，請檢查 [Docker Hub](https://hub.docker.com/r/aws-iot-greengrass) 上的標籤頁面或使用 `aws ecr list-images` 命令。例如：

```
aws ecr list-images --region us-west-2 --registry-id 216483018798 --  
repository-name aws-iot-greengrass
```

步驟 2：建立及設定 Greengrass 群組和核心

Docker 映像已安裝 AWS IoT Greengrass Core 軟體，但您必須建立 Greengrass 群組和核心。這包括下載憑證和核心的組態檔案。

- 請遵循 [the section called “單元 2：安裝 AWS IoT Greengrass 核心軟體”](#) 中的步驟。略過下載和執行 AWS IoT Greengrass 核心軟體的步驟。此軟體及其執行時間相依性已在 Docker 映像檔中設定。

步驟 3：AWS IoT Greengrass 在本機執行

設定完群組之後，您便能設定及啟動核心。如需示範執行方式的步驟，請選擇您的作業系統：

在本機執行 Greengrass (Linux)

在您的電腦終端機上執行下列命令。

- 為裝置的安全資源建立資料夾，並將憑證和金鑰移至該資料夾。執行下列命令。以安全資源的路徑取代 *path-to-security-files*，並以檔案名稱中的憑證 ID 取代 *certificateId*。

```
mkdir /tmp/certs  
mv path-to-security-files/certificateId-certificate.pem.crt /tmp/certs  
mv path-to-security-files/certificateId-public.pem.key /tmp/certs  
mv path-to-security-files/certificateId-private.pem.key /tmp/certs  
mv path-to-security-files/AmazonRootCA1.pem /tmp/certs
```

- 為裝置的組態建立資料夾，並將 AWS IoT Greengrass 核心組態檔案移至該資料夾。執行下列命令。將 *path-to-config-file* 取代為組態檔案的路徑。

```
mkdir /tmp/config  
mv path-to-config-file/config.json /tmp/config
```

- 在 Docker 容器中啟動 AWS IoT Greengrass 並繫結掛載憑證和組態檔案。

以解壓縮憑證和組態檔案的路徑取代 /tmp。

```
docker run --rm --init -it --name aws-iot-greengrass \
--entrypoint /greengrass-entrypoint.sh \
-v /tmp/certs:/greengrass/certs \
-v /tmp/config:/greengrass/config \
-p 8883:8883 \
216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

輸出應類似以下範例：

```
Setting up greengrass daemon
Validating hardlink/softlink protection
Waiting for up to 30s for Daemon to start

Greengrass successfully started with PID: 10
```

在本機執行 Greengrass (macOS)

在您的電腦終端機上執行下列命令。

1. 為裝置的安全資源建立資料夾，並將憑證和金鑰移至該資料夾。執行下列命令。以安全資源的路徑取代 *path-to-security-files*，並以檔案名稱中的憑證 ID 取代 *certificateId*。

```
mkdir /tmp/certs
mv path-to-security-files/certificateId-certificate.pem.crt /tmp/certs
mv path-to-security-files/certificateId-public.pem.key /tmp/certs
mv path-to-security-files/certificateId-private.pem.key /tmp/certs
mv path-to-security-files/AmazonRootCA1.pem /tmp/certs
```

2. 為裝置的組態建立資料夾，並將 AWS IoT Greengrass 核心組態檔案移至該資料夾。執行下列命令。將 *path-to-config-file* 取代為組態檔案的路徑。

```
mkdir /tmp/config
mv path-to-config-file/config.json /tmp/config
```

3. 在 Docker 容器中啟動 AWS IoT Greengrass 並繫結掛載憑證和組態檔案。

以解壓縮憑證和組態檔案的路徑取代 /tmp。

```
docker run --rm --init -it --name aws-iot-greengrass \
--entrypoint /greengrass-entrypoint.sh \
```

```
-v /tmp/certs:/greengrass/certs \  
-v /tmp/config:/greengrass/config \  
-p 8883:8883 \  
216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

輸出應類似以下範例：

```
Setting up greengrass daemon  
Validating hardlink/softlink protection  
Waiting for up to 30s for Daemon to start  
  
Greengrass successfully started with PID: 10
```

在本機執行 Greengrass (Windows)

1. 為裝置的安全資源建立資料夾，並將憑證和金鑰移至該資料夾。在命令提示中執行下列命令：以安全資源的路徑取代 *path-to-security-files*，並以檔案名稱中的憑證 ID 取代 *certificateId*。

```
mkdir C:\Users\%USERNAME%\Downloads\certs  
move path-to-security-files\certificateId-certificate.pem.crt C:\Users\%USERNAME%\Downloads\certs  
move path-to-security-files\certificateId-public.pem.key C:\Users\%USERNAME%\Downloads\certs  
move path-to-security-files\certificateId-private.pem.key C:\Users\%USERNAME%\Downloads\certs  
move path-to-security-files\AmazonRootCA1.pem C:\Users\%USERNAME%\Downloads\certs
```

2. 為裝置的組態建立資料夾，並將 AWS IoT Greengrass 核心組態檔案移至該資料夾。在命令提示中執行下列命令：將 *path-to-config-file* 取代為組態檔案的路徑。

```
mkdir C:\Users\%USERNAME%\Downloads\config  
move path-to-config-file\config.json C:\Users\%USERNAME%\Downloads\config
```

3. 在 Docker 容器中啟動 AWS IoT Greengrass 並繫結掛載憑證和組態檔案。在命令提示中執行下列命令。

```
docker run --rm --init -it --name aws-iot-greengrass --entrypoint /greengrass-entrypoint.sh -v c:/Users/%USERNAME%/Downloads/certs:/greengrass/certs
```

```
-v c:/Users/%USERNAME%/Downloads/config:/greengrass/config -p 8883:8883  
216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

Docker 提示您與 Docker 協助程式共用 C:\ 磁碟機時，允許其在 Docker 容器內綁定掛載 C:\ 目錄。如需詳細資訊，請參閱 Docker 文件中的[共用磁碟機](#)。

輸出應類似以下範例：

```
Setting up greengrass daemon  
Validating hardlink/softlink protection  
Waiting for up to 30s for Daemon to start  
  
Greengrass successfully started with PID: 10
```

Note

如果容器不開啟 shell 和立即結束，您可以透過在您啟動映像時連結掛載 Greengrass 執行時間日誌來偵錯問題。如需詳細資訊，請參閱[the section called “將 Greengrass 執行時間日誌保留在 Docker 容器之外”](#)。

步驟 4：為 Greengrass 群組的容器化設定「No container」（無容器）

當您 AWS IoT Greengrass 在 Docker 容器中執行時，所有 Lambda 函數都必須在沒有容器化的情況下執行。在本步驟中，您會將群組的預設容器化設為 No container（無容器）。第一次部署群組之前，必須先執行此動作。

1. 在 AWS IoT 主控台導覽窗格的管理下，展開 Greengrass 裝置，然後選擇群組 (V1)。
2. 選擇您希望變更設定的群組。
3. 選擇 Lambda 函數索引標籤。
4. 在預設 Lambda 函數執行期環境中，選擇編輯。
5. 在編輯預設 Lambda 函數執行期環境中，在預設 Lambda 函數容器化下，變更容器化設定。
6. 選擇儲存。

變更會在群組部署後生效。

如需詳細資訊，請參閱[the section called “為群組中的 Lambda 函數設定預設容器化”](#)。

Note

根據預設，Lambda 函數會使用群組容器化設定。如果您在 Docker 容器中執行時，覆寫任何 Lambda 函數的無容器設定，則部署會失敗。AWS IoT Greengrass

步驟 5：將 Lambda 函數部署至 AWS IoT Greengrass Docker 容器

您可以將長期 Lambda 函數部署到 Greengrass Docker 容器。

- 請依照中的步驟[the section called “單元 3（第 1 部分）：上的 Lambda 函數 AWS IoT Greengrass”](#)，將長期 Hello World Lambda 函數部署至容器。

步驟 6：（選用）部署與 Docker 容器中執行之 Greengrass 互動的用戶端裝置

您也可以部署在 Docker 容器中執行 AWS IoT Greengrass 時與互動的用戶端裝置。

- 依照 中的步驟[the section called “單元 4：與 AWS IoT Greengrass 群組中的用戶端裝置互動”](#)部署連接到核心的用戶端裝置，並傳送 MQTT 訊息。

停止 AWS IoT Greengrass Docker 容器

若要停止 AWS IoT Greengrass Docker 容器，請在終端機或命令提示中按 Ctrl+C。此動作會SIGTERM傳送至 Greengrass 協助程式程序，以銷毀 Greengrass 協助程式程序，以及由協助程式程序啟動的所有 Lambda 程序。Docker 容器是以 /dev/init 程序初始化為 PID 1，這有助於移除任何剩餘的 zombie 程序。如需詳細資訊，請參閱 [Docker 執行參考](#)。

在 Docker AWS IoT Greengrass 容器中進行故障診斷

使用以下資訊來協助疑難排解在 Docker 容器中執行 AWS IoT Greengrass 的問題。

錯誤：無法從非 TTY 裝置執行互動式登入。

解決方案：執行 `aws ecr get-login-password` 命令時，可能會發生此錯誤。請確定您已安裝最新版本 AWS CLI 2 或版本 1。我們建議您使用第 2 AWS CLI 版。如需詳細資訊，請參閱「AWS Command Line Interface 使用者指南」中的[安裝 AWS CLI](#)。

錯誤：不明選項：-no-include-email。

解決方案：執行 `aws ecr get-login` 命令時，可能會發生此錯誤。請確定您已安裝 AWS CLI 最新版本（例如，執行：`pip install awscli --upgrade --user`）。如果您使用 Windows 並已使用 MSI 安裝程式安裝 CLI，您必須重新啟動安裝程序。如需詳細資訊，請參閱 [AWS Command Line Interface 《使用者指南》](#) 中的在 [Microsoft Windows AWS Command Line Interface 上安裝](#)。

警告：IPv4 已停用。網路將無法運作。

解決方案：在 Linux AWS IoT Greengrass 電腦上執行時，您可能會收到此警告或類似訊息。如[此步驟](#)所述啟用 IPv4 網路轉送。未啟用 IPv4 轉送時 AWS IoT Greengrass，雲端部署和 MQTT 通訊無法運作。如需詳細資訊，請參閱 Docker 文件中的[在執行時間設定命名空間核心參數 \(sysctls\)](#)。

錯誤：防火牆封鎖了 Windows 和容器之間的檔案共用。

解決方案：在 Windows 電腦上執行 Docker 時，您可能收到此錯誤或 Firewall Detected 訊息。如果您登入虛擬私有網路 (VPN)，而您的網路設定防止掛載共用磁碟機，也可能會發生這個錯誤。在這種情況下，請關閉 VPN 並重新執行 Docker 容器。

錯誤：呼叫 `GetAuthorizationToken` 操作時發生錯誤 (`AccessDeniedException`)：
User: arn:aws:iam::<account-id>:user/<user-name> 未獲授權在資源上執行：
`ecr:GetAuthorizationToken`：*

如果您沒有足夠的許可來存取 Amazon ECR 儲存庫，則執行 `aws ecr get-login-password` 命令時可能會收到此錯誤。如需詳細資訊，請參閱 [《Amazon ECR 使用者指南》](#) 中的 [Amazon ECR 儲存庫政策範例和存取一個 Amazon ECR 儲存庫](#)。

如需一般 AWS IoT Greengrass 故障診斷說明，請參閱 [疑難排解](#)。

在 Docker AWS IoT Greengrass 容器中偵錯

為了對 Docker 容器的問題進行偵錯，您可以保留 Greengrass 執行時間日誌或將互動式 shell 連接到 Docker 容器。

將 Greengrass 執行時間日誌保留在 Docker 容器之外

您可以在繫結掛載/`greengrass/ggc/var/log`目錄之後執行 AWS IoT Greengrass Docker 容器。容器結束或移除之後，日誌仍會保留。

在 Linux 或 macOS 上

[停止任何 Greengrass Docker 容器](#)在主機上執行，接著在終端機中執行以下命令。此會連結掛載 Greengrass log 目錄並啟動 Docker 映像檔。

以解壓縮憑證和組態檔案的路徑取代 /tmp。

```
docker run --rm --init -it --name aws-iot-greengrass \
  --entrypoint /greengrass-entrypoint.sh \
  -v /tmp/certs:/greengrass/certs \
  -v /tmp/config:/greengrass/config \
  -v /tmp/log:/greengrass/ggc/var/log \
  -p 8883:8883 \
  216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

接著您可以在主機上的 /tmp/log 檢查日誌，以查看 Greengrass 在 Docker 容器內部執行的同時發生了什麼。

在 Windows 上

[停止任何 Greengrass Docker 容器](#)在主機上執行，接著在命令提示中執行以下命令。此會連結掛載 Greengrass log 目錄並啟動 Docker 映像檔。

```
cd C:\Users\%USERNAME%\Downloads
mkdir log
docker run --rm --init -it --name aws-iot-greengrass --entrypoint /greengrass-
entrypoint.sh -v c:/Users/%USERNAME%/Downloads/certs:/greengrass/certs -v c:/
Users/%USERNAME%/Downloads/config:/greengrass/config -v c:/Users/%USERNAME%/
Downloads/log:/greengrass/ggc/var/log -p 8883:8883 216483018798.dkr.ecr.us-
west-2.amazonaws.com/aws-iot-greengrass:latest
```

接著您可以在主機上的 C:/Users/%USERNAME%/Downloads/log 檢查日誌，以查看 Greengrass 在 Docker 容器內部執行的同時發生了什麼。

將互動式 Shell 連接到 Docker 容器

您可以將互動式 shell 連接到執行中的 AWS IoT Greengrass Docker 容器。這可協助您調查 Greengrass Docker 容器的狀態。

在 Linux 或 macOS 上

Greengrass Docker 容器執行時，在獨立的終端機中執行以下命令。

```
docker exec -it $(docker ps -a -q -f "name=aws-iot-greengrass") /bin/bash
```

在 Windows 上

Greengrass Docker 容器執行時，在獨立的命令提示中執行以下命令。

```
docker ps -a -q -f "name=aws-iot-greengrass"
```

從先前的命令中將 *gg-container-id* 取代為 container_id 結果。

```
docker exec -it gg-container-id /bin/bash
```

使用 Lambda 函數和連接器存取本機資源

此功能適用於 AWS IoT Greengrass Core v1.3 和更新版本。

使用 AWS IoT Greengrass，您可以編寫 AWS Lambda 函數並在雲端中設定[連接器](#)，並將其部署到核心裝置以進行本機執行。在執行 Linux 的 Greengrass 核心上，這些本機部署的 Lambda 函數和連接器可以存取 Greengrass 核心裝置上實際存在的本機資源。例如，若要與透過 Modbus 或 CANbus 連線的裝置通訊，您可以讓 Lambda 函數存取核心裝置上的序列連接埠。若要設定安全地存取本機資源，您必須保證實體硬體和 Greengrass 核心裝置作業系統這兩者的安全性。

若要開始存取本機資源，請參閱以下教學課程：

- [如何使用 AWS 命令列界面設定本機資源存取](#)
- [如何使用 設定本機資源存取 AWS 管理主控台](#)

支援的資源類型

您可以存取兩種類型的本機資源：磁碟區資源和裝置資源。

磁碟區資源

在根檔案系統中的檔案或目錄 (除了在 /sys、/dev 或 /var 之下)。其中包含：

- 用於跨 Greengrass Lambda 函數讀取或寫入資訊的資料夾或檔案 (例如 /usr/lib/python2.x/site-packages/local)。
- 在主機的 /proc 檔案系統下的資料夾或檔案 (例如，/proc/net 或 /proc/stat)。支援 v1.6 或更新版本。如需其他需求，請參閱 [the section called “在 /proc 目錄下的磁碟區資源”](#)。

Tip

若要設定 /var、/var/run 和 /var/lib 目錄為磁碟區資源，首先，請掛載目錄於不同的資料夾中，然後設定資料夾為磁碟區資源。

當您設定磁碟區資源，您指定來源路徑和目的地路徑。來源路徑是主機上資源的絕對路徑。目的地路徑是 Lambda 命名空間環境中資源的絕對路徑。這是 Greengrass Lambda 函數或連接器執行所在的容器。目標路徑的所有變更都會反映在主機檔案系統的來源路徑上。

 Note

目的地路徑中的檔案僅會顯示在 Lambda 命名空間中。您無法在一般 Linux 命名空間中看到這些檔案。

裝置資源

檔案位於 `/dev` 之下。只有位於 `/dev` 底下的字元裝置或區塊型儲存設備才允許使用裝置資源。其中包含：

- 序列埠可用於與透過序列埠 (例如 `/dev/ttyS0`、`/dev/ttyS1`) 連線的裝置通訊。
- USB 可用於連接 USB 週邊設備 (例如 `/dev/ttyUSB0` 或 `/dev/bus/usb`)。
- GPIO，使用 GPIO (例如 `/dev/gpiomem`) 的感測器和傳動器。
- GPU，使用內建的 GPU (例如，`/dev/nvidia0`) 加速機器學習速度。
- 相機用於捕捉影像和影片 (例如，`/dev/video0`)。

 Note

`/dev/shm` 為例外。它僅能設為磁碟區資源。在 `/dev/shm` 底下的資源必須授與 `rw` 權限。

AWS IoT Greengrass 也支援用於執行機器學習推論的資源類型。如需詳細資訊，請參閱[執行機器學習推論](#)。

需求

以下要求適用於設定安全地存取本機資源：

- 您必須使用 AWS IoT Greengrass 核心軟體 1.3 版或更新版本。若要為主機的 `/proc` 目錄建立資源，您必須使用 v1.6 或更新版本。
- 本機資源 (包括任何必要的驅動程式和程式庫) 必須正確安裝在 Greengrass 核心裝置，並在使用時保持可用狀態。
- 所需操作的資源和存取資源的權限，並不需要根權限。
- 僅提供 `read` 或 `read and write` 許可。Lambda 函數無法在資源上執行授權的操作。
- 在 Greengrass 核心裝置的作業系統中，您必須提供完整的本機資源路徑。

- 資源名稱或 ID 字元達最大限制 128 個字元時，必須使用模式 `[a-zA-Z0-9:_-]+`。

在 /proc 目錄下的磁碟區資源

以下考量適用於主機的 /proc 目錄下的磁碟區資源。

- 您必須使用 AWS IoT Greengrass 核心軟體 1.6 版或更新版本。
- 您可以允許 Lambda 函數的唯讀存取，但不允許讀寫存取。此存取層級由 管理 AWS IoT Greengrass。
- 您可能也需要授予許可給作業系統群組，才能在檔案系統中啟用讀取權限。例如，假設您的來源目錄或檔案擁有 660 個檔案許可，這表示只有群組中的擁有者或使用者有讀取 (和寫入) 存取權限。在此情況下，您必須將作業系統群組擁有者的許可新增至資源中。如需詳細資訊，請參閱[the section called “群組擁有者檔案存取許可”](#)。
- 主機環境和 Lambda 命名空間都包含 /proc 目錄，因此當您指定目的地路徑時，請務必避免命名衝突。例如，如果 /proc 是來源路徑，您可以指定 /host-proc 做為目的地路徑 (或除了「/proc」以外的任何路徑)。

群組擁有者檔案存取許可

AWS IoT Greengrass Lambda 函數程序通常會以 `ggc_user` 和 執行 `ggc_group`。不過，您可以在本機資源定義中為 Lambda 函數程序提供額外的檔案存取許可，如下所示：

- 若要新增擁有資源的 Linux 群組許可，請使用 `GroupOwnerSetting#AutoAddGroupOwner` 參數或自動新增擁有資源主控台選項之系統群組的檔案系統許可。
- 若要新增不同 Linux 群組的許可，請使用 `GroupOwnerSetting#GroupOwner` 參數或指定另一個系統群組來新增檔案系統許可主控台選項。若 `GroupOwnerSetting#AutoAddGroupOwner` 為 `true`，則 `GroupOwner` 會被忽略。

AWS IoT Greengrass Lambda 函數程序會繼承 `ggc_user`、`ggc_group` 和 Linux 群組 (如果新增) 的所有檔案系統許可。若要讓 Lambda 函數存取資源，Lambda 函數程序必須具有資源的必要許可。您可以使用 `chmod(1)` 命令變更資源許可，如果需要的話。

另請參閱

- 中的資源 [Service Quotas](#) Amazon Web Services 一般參考

如何使用 AWS 命令列界面設定本機資源存取

此功能適用於 AWS IoT Greengrass Core v1.3 和更新版本。

若要使用本機資源，您必須新增資源定義至部署到 Greengrass 核心裝置的群組定義中。群組定義必須在授與 Lambda 函數存取本機資源許可的地方含有 Lambda 函數定義。如需包括要求和限制的詳細資訊，請參閱[使用 Lambda 函數和連接器存取本機資源](#)。

本教學課程說明使用 AWS Command Line Interface (CLI) 建立本機資源和設定其存取權的程序。為了執行教學課程中的步驟，您必須已經建立 [入門 AWS IoT Greengrass](#) 中所述的 Greengrass 群組。

如需使用的教學課程 AWS 管理主控台，請參閱 [如何使用 設定本機資源存取 AWS 管理主控台](#)。

建立本機資源

首先，請您使用 [CreateResourceDefinition](#) 命令建立資源定義，以指定要存取的資源。在此範例中，我們將建立兩個資源，TestDirectory 和 TestCamera：

```
aws greengrass create-resource-definition --cli-input-json '{
  "Name": "MyLocalVolumeResource",
  "InitialVersion": {
    "Resources": [
      {
        "Id": "data-volume",
        "Name": "TestDirectory",
        "ResourceDataContainer": {
          "LocalVolumeResourceData": {
            "SourcePath": "/src/LRAtest",
            "DestinationPath": "/dest/LRAtest",
            "GroupOwnerSetting": {
              "AutoAddGroupOwner": true,
              "GroupOwner": ""
            }
          }
        }
      },
      {
        "Id": "data-device",
        "Name": "TestCamera",
        "ResourceDataContainer": {
          "LocalDeviceResourceData": {
```

```

        "SourcePath": "/dev/video0",
        "GroupOwnerSetting": {
            "AutoAddGroupOwner": true,
            "GroupOwner": ""
        }
    }
}
]
}
}'

```

資源：Greengrass 群組中的 Resource 物件清單。一個 Greengrass 群組最多可有 50 個資源。

Resource#Id：資源的唯一識別符。此 ID 用於查閱 Lambda 函數組態裡的資源。最大長度 128 個字元。模式：`[a-zA-Z0-9:_-]+`。

Resource#Name：資源的名稱。在 Greengrass 主控台中顯示的資源名稱。最大長度 128 個字元。模式：`[a-zA-Z0-9:_-]+`。

LocalDeviceResourceData#SourcePath：裝置資源確實的本機路徑。裝置資源的來源路徑可以只參閱在 `/dev` 下的字元裝置或區塊型儲存設備。

LocalVolumeResourceData#SourcePath：在 Greengrass 核心裝置上，磁碟區資源確實的本機路徑。此位置位於函數執行所在的[容器](#)外。磁碟區資源類型的來源路徑不能以 `/sys` 為開頭。

LocalVolumeResourceData#DestinationPath：Lambda 環境內磁碟區資源的確實路徑。此位置位於函數執行所在的容器內。

GroupOwnerSetting：可讓您為 Lambda 程序設定其他的群組權限。此欄位為選用欄位。如需詳細資訊，請參閱[群組擁有者檔案存取許可](#)。

GroupOwnerSetting#AutoAddGroupOwner：如果為 `true`，則 Greengrass 會自動將資源的指定 Linux 作業系統群組擁有者新增至 Lambda 程序權限。因此，Lambda 程序對新增的 Linux 群組具有檔案存取許可。

GroupOwnerSetting#GroupOwner：指定已新增權限至 Lambda 程序的 Linux 作業系統群組名稱。此欄位為選用欄位。

資源定義版本 ARN 會由 [CreateResourceDefinition](#) 傳回。應使用 ARN 更新群組定義。

```
{
```



```

    "LatestVersionArn": "arn:aws:greengrass:us-west-2:012345678901:/greengrass/
definition/resources/ab14d0b5-116e-4951-a322-9cde24a30373/versions/a4d9b882-
d025-4760-9cfe-9d4fada5390d",
    "Name": "MyLocalVolumeResource",
    "LastUpdatedTimestamp": "2017-11-15T01:18:42.153Z",
    "LatestVersion": "a4d9b882-d025-4760-9cfe-9d4fada5390d",
    "CreationTimestamp": "2017-11-15T01:18:42.153Z",
    "Id": "ab14d0b5-116e-4951-a322-9cde24a30373",
    "Arn": "arn:aws:greengrass:us-west-2:123456789012:/greengrass/definition/resources/
ab14d0b5-116e-4951-a322-9cde24a30373"
  }

```

建立 Greengrass 函數

建立資源後，請使用 [CreateFunctionDefinition](#) 命令建立 Greengrass 函數並授與該函數存取資源的權限：

```

aws greengrass create-function-definition --cli-input-json '{
  "Name": "MyFunctionDefinition",
  "InitialVersion": {
    "Functions": [
      {
        "Id": "greengrassLraTest",
        "FunctionArn": "arn:aws:lambda:us-
west-2:012345678901:function:lraTest:1",
        "FunctionConfiguration": {
          "Pinned": false,
          "MemorySize": 16384,
          "Timeout": 30,
          "Environment": {
            "ResourceAccessPolicies": [
              {
                "ResourceId": "data-volume",
                "Permission": "rw"
              },
              {
                "ResourceId": "data-device",
                "Permission": "ro"
              }
            ],
            "AccessSysfs": true
          }
        }
      }
    ]
  }
}

```

```

    }
  ]
}
}'

```

ResourceAccessPolicies：包含 **permission** 授予 Lambda 函數資源存取權的 **resourceId** 和 **resource**。Lambda 函數最多可以存取 20 個資源。

ResourceAccessPolicy#Permission：指定 Lambda 函數在資源上擁有的許可。可行的選項為 **rw** (讀取/寫入) 或 **ro** (唯讀)。

AccessSysfs：如果為 **true**，則 Lambda 程序可在 Greengrass 核心裝置上讀取 **/sys** 資料夾。在 Greengrass Lambda 函數需要從 讀取裝置資訊時使用 **/sys**。

同樣地，[CreateFunctionDefinition](#) 會傳回一個函數定義版本 ARN。此 ARN 應於您的群組定義版本中使用。

```

{
  "LatestVersionArn": "arn:aws:greengrass:us-west-2:012345678901:/greengrass/
definition/functions/3c9b1685-634f-4592-8dfd-7ae1183c28ad/versions/37f0d50e-ef50-4faf-
b125-ade8ed12336e",
  "Name": "MyFunctionDefinition",
  "LastUpdatedTimestamp": "2017-11-22T02:28:02.325Z",
  "LatestVersion": "37f0d50e-ef50-4faf-b125-ade8ed12336e",
  "CreationTimestamp": "2017-11-22T02:28:02.325Z",
  "Id": "3c9b1685-634f-4592-8dfd-7ae1183c28ad",
  "Arn": "arn:aws:greengrass:us-west-2:123456789012:/greengrass/definition/
functions/3c9b1685-634f-4592-8dfd-7ae1183c28ad"
}

```

將 Lambda 函數新增至群組

最後，請使用 [CreateGroupVersion](#) 新增函數至群組。例如：

```

aws greengrass create-group-version --group-id "b36a3aeb-3243-47ff-9fa4-7e8d98cd3cf5" \
--resource-definition-version-arn "arn:aws:greengrass:us-west-2:123456789012:/
greengrass/definition/resources/db6bf40b-29d3-4c4e-9574-21ab7d74316c/versions/31d0010f-
e19a-4c4c-8098-68b79906fb87" \
--core-definition-version-arn "arn:aws:greengrass:us-west-2:123456789012:/
greengrass/definition/cores/adbf3475-f6f3-48e1-84d6-502f02729067/
versions/297c419a-9deb-46dd-8ccc-341fc670138b" \

```

```
--function-definition-version-arn "arn:aws:greengrass:us-west-2:123456789012:/greengrass/definition/functions/d1123830-da38-4c4c-a4b7-e92eec7b6d3e/versions/a2e90400-caae-4ffd-b23a-db1892a33c78" \
--subscription-definition-version-arn "arn:aws:greengrass:us-west-2:123456789012:/greengrass/definition/subscriptions/7a8ef3d8-1de3-426c-9554-5b55a32fbc66/versions/470c858c-7eb3-4abd-9d48-230236bfbf6a"
```

Note

若要了解如何取得群組 ID 來與此命令搭配使用，請參閱 [the section called “取得群組 ID”](#)。

會傳回一個新的版本：

```
{
  "Arn": "arn:aws:greengrass:us-west-2:012345678901:/greengrass/groups/b36a3aeb-3243-47ff-9fa4-7e8d98cd3cf5/versions/291917fb-ec54-4895-823e-27b52da25481",
  "Version": "291917fb-ec54-4895-823e-27b52da25481",
  "CreationTimestamp": "2017-11-22T01:47:22.487Z",
  "Id": "b36a3aeb-3243-47ff-9fa4-7e8d98cd3cf5"
}
```

您的 Greengrass 群組現在有 `lraTest` Lambda 函數，其擁有兩個資源的存取權：`TestDirectory` 和 `TestCamera`。

此範例 Lambda 函數「`lraTest.py`」，以 Python 寫入本機磁碟區資源：

```
# Demonstrates a simple use case of local resource access.
# This Lambda function writes a file test to a volume mounted inside
# the Lambda environment under destLRAtest. Then it reads the file and
# publishes the content to the AWS IoT LRAtest topic.

import sys
import greengrasssdk
import platform
import os
import logging

# Setup logging to stdout
logger = logging.getLogger(__name__)
logging.basicConfig(stream=sys.stdout, level=logging.DEBUG)
```

```
# Create a Greengrass Core SDK client.
client = greengrasssdk.client('iot-data')
volumePath = '/dest/LRAtest'

def function_handler(event, context):
    try:
        client.publish(topic='LRA/test', payload='Sent from AWS IoT Greengrass Core.')
        volumeInfo = os.stat(volumePath)
        client.publish(topic='LRA/test', payload=str(volumeInfo))
        with open(volumePath + '/test', 'a') as output:
            output.write('Successfully write to a file.')
        with open(volumePath + '/test', 'r') as myfile:
            data = myfile.read()
        client.publish(topic='LRA/test', payload=data)
    except Exception as e:
        logger.error('Failed to publish message: ' + repr(e))
    return
```

這些命令由 Greengrass API 提供，建立與管理資源定義和資源定義版本：

- [CreateResourceDefinition](#)
- [CreateResourceDefinitionVersion](#)
- [DeleteResourceDefinition](#)
- [GetResourceDefinition](#)
- [GetResourceDefinitionVersion](#)
- [ListResourceDefinitions](#)
- [ListResourceDefinitionVersions](#)
- [UpdateResourceDefinition](#)

疑難排解

- 問：為什麼我的 Greengrass 群組部署失敗，發生與以下類似的狀況：

```
group config is invalid:
  ggc_user or [ggc_group root tty] don't have ro permission on the file: /dev/tty0
```

答：此失敗狀況表示 Lambda 程序並沒有存取指定資源的許可。解決方法為變更資源的檔案許可權限，讓 Lambda 可以存取即可。(如需詳細資訊，請參閱[群組擁有者檔案存取許可](#))。

- 問：當我將 `/var/run` 設為磁碟區資源時，為什麼 Lambda 函數無法啟動，並在 `runtime.log` 出現錯誤訊息：

```
[ERROR]-container_process.go:39,Runtime execution error: unable to start lambda container.
container_linux.go:259: starting container process caused "process_linux.go:345: container init caused \"rootfs_linux.go:62: mounting \"/var/run\" to rootfs \"/greengrass/ggc/packages/1.3.0/rootfs_sys\" at \"/greengrass/ggc/packages/1.3.0/rootfs_sys/run\" caused \"invalid argument\""
```

A：AWS IoT Greengrass core 目前不支援將 `/var`、`/var/run` 和組態 `/var/lib` 為磁碟區資源。一種解決方法是首先在另一個文件中掛載 `/var`、`/var/run` 或 `/var/lib`，然後再設定此資料夾做為磁碟區資源。

- 問：當我將 `/dev/shm` 設為唯讀許可的磁碟區資源時，為什麼 Lambda 函數無法啟動，並在 `runtime.log` 出現錯誤訊息：

```
[ERROR]-container_process.go:39,Runtime execution error: unable to start lambda container.
container_linux.go:259: starting container process caused "process_linux.go:345: container init caused \"rootfs_linux.go:62: mounting \"/dev/shm\" to rootfs \"/greengrass/ggc/packages/1.3.0/rootfs_sys\" at \"/greengrass/ggc/packages/1.3.0/rootfs_sys/dev/shm\" caused \"operation not permitted\""
```

答：`/dev/shm` 因為只能設定成 [讀取/寫入]。變更資源許可為 `rw` 以解決此問題。

如何使用 設定本機資源存取 AWS 管理主控台

此功能適用於 AWS IoT Greengrass Core v1.3 和更新版本。

您可以設定 Lambda 函數，以安全地存取主機 Greengrass 核心裝置上的本機資源。本機資源指實際存在於主機上的匯流排及週邊設備，或主機作業系統上的檔案系統磁碟區。如需包括要求和限制的詳細資訊，請參閱[使用 Lambda 函數和連接器存取本機資源](#)。

本教學說明如何使用 AWS 管理主控台 設定對 AWS IoT Greengrass 核心裝置上存在的本機資源的存取。包含以下高階執行步驟：

1. [建立 Lambda 函數部署套件](#)

2. [建立和發佈 Lambda 函數](#)
3. [將 Lambda 函數新增至群組](#)
4. [新增本機資源至群組](#)
5. [新增訂閱到群組](#)
6. [部署群組](#)

如需使用的教學課程 AWS Command Line Interface，請參閱 [如何使用 AWS 命令列界面設定本機資源存取](#)。

先決條件

為完成此教學課程您需要：

- Greengrass 群組和 Greengrass 核心 (v1.3 或更新版本)。若要建立 Greengrass 群組或核心，請參閱 [入門 AWS IoT Greengrass](#)。
- 以下目錄位於 Greengrass 核心裝置：
 - /src/LRAtest
 - /dest/LRAtest

這些目錄的擁有者群組必須擁有讀取和寫入存取目錄的權限。您可以使用以下命令來授與存取權限：

```
sudo chmod 0775 /src/LRAtest
```

步驟 1：建立 Lambda 函數部署套件

在此步驟中，您會建立 Lambda 函數部署套件，這是一個包含函數程式碼和相依性的 ZIP 檔案。您也可以下載 AWS IoT Greengrass 核心 SDK，以做為相依性包含在套件中。

1. 在您的電腦中複製以下 Python 指令碼到本機檔案 `lraTest.py`。這是 Lambda 函數的應用程式邏輯。

```
# Demonstrates a simple use case of local resource access.
# This Lambda function writes a file test to a volume mounted inside
# the Lambda environment under destLRAtest. Then it reads the file and
# publishes the content to the AWS IoT LRAtest topic.
```

```
import sys
import greengrasssdk
import platform
import os
import logging

# Setup logging to stdout
logger = logging.getLogger(__name__)
logging.basicConfig(stream=sys.stdout, level=logging.DEBUG)

# Create a Greengrass Core SDK client.
client = greengrasssdk.client('iot-data')
volumePath = '/dest/LRAtest'

def function_handler(event, context):
    try:
        client.publish(topic='LRA/test', payload='Sent from AWS IoT Greengrass Core.')
        volumeInfo = os.stat(volumePath)
        client.publish(topic='LRA/test', payload=str(volumeInfo))
        with open(volumePath + '/test', 'a') as output:
            output.write('Successfully write to a file.')
        with open(volumePath + '/test', 'r') as myfile:
            data = myfile.read()
            client.publish(topic='LRA/test', payload=data)
    except Exception as e:
        logger.error('Failed to publish message: ' + repr(e))
    return
```

2. 從[AWS IoT Greengrass 核心開發套件](#)下載頁面，將適用於 Python AWS IoT Greengrass 的核心開發套件下載至您的電腦。
3. 解壓縮下載的封裝，以取得軟體開發套件。SDK 為 greengrasssdk 資料夾。
4. 請將下列項目壓縮成一個檔案，並命名為 lraTestLambda.zip：
 - lraTest.py。應用程式邏輯。
 - greengrasssdk。所有 Python Lambda 函數所需的程式庫。

lraTestLambda.zip 檔案是您的 Lambda 函數部署套件。現在您已準備好建立 Lambda 函數並上傳部署套件。

步驟 2：建立和發佈 Lambda 函數

在此步驟中，您會使用 AWS Lambda 主控台來建立 Lambda 函數，並將其設定為使用您的部署套件。然後，您會發佈函數版本和建立別名。

首先，建立 Lambda 函數。

1. 在 AWS 管理主控台，選擇服務，然後開啟 AWS Lambda 主控台。
2. 選擇 函數。
3. 選擇建立函數，然後選擇從頭開始撰寫。
4. 在 Basic information (基本資訊) 區段中，使用下列值。
 - a. 針對函數名稱，請輸入 **TestLRA**。
 - b. 針對執行期，選擇 Python 3.7。
 - c. 對於許可，請保留預設設定。這會建立授予基本 Lambda 許可的執行角色。不會使用此角色 AWS IoT Greengrass。
5. 選擇建立函數。

Basic information

Function name
Enter a name that describes the purpose of your function.

TestLRA

Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime [Info](#)
Choose the language to use to write your function.

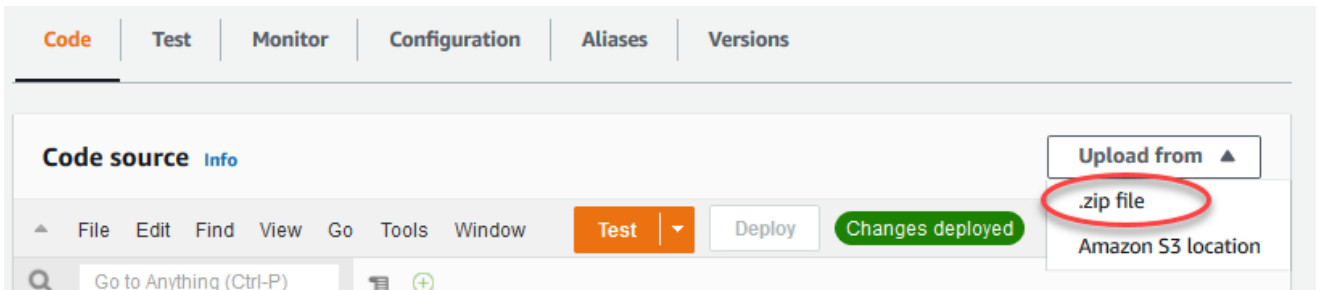
Python 3.7 ▼

Permissions [Info](#)
Lambda will create an execution role with permission to upload logs to Amazon CloudWatch Logs. You can configure and modify permissions further when you add triggers.

► Choose or create an execution role

Cancel Create function

6. 上傳您的 Lambda 函數部署套件並註冊處理常式。
 - a. 在程式碼索引標籤的程式碼來源下，選擇上傳來源。從下拉式清單中，選擇 .zip 檔案。



- b. 選擇上傳，然後選擇您的lraTestLambda.zip部署套件。然後選擇 Save (儲存)。
- c. 在函數的程式碼索引標籤的執行時間設定下，選擇編輯，然後輸入下列值。
 - 針對執行期，選擇 Python 3.7。
 - 針對 Handler (處理常式)，輸入 lraTest.function_handler。
- d. 選擇儲存。

Note

AWS Lambda 主控台上的測試按鈕不適用於此函數。AWS IoT Greengrass 核心 SDK 不包含在 AWS Lambda 主控台中獨立執行 Greengrass Lambda 函數所需的模組。這些模組 (例如 greengrass_common) 會在部署到您的 Greengrass 核心之後提供給函數。

接著，發佈 Lambda 函數的第一個版本。然後，建立[版本的別名](#)。

Greengrass 群組可以依別名 (建議) 或版本參考 Lambda 函數。使用別名可讓您更輕鬆地管理程式碼更新，因為您不必在更新函數程式碼時變更訂閱資料表或群組定義。反之，您只需將別名指向新的函數版本。

7. 從 Actions (動作)，選擇 Publish new version (發佈新版本)。
8. 針對 Version description (版本描述)，輸入 **First version**，然後選擇 Publish (發佈)。
9. 在 TestLRA: 1 組態頁面，從Actions (動作) 中選擇 Create alias (建立別名)。
10. 在建立別名頁面上，針對名稱輸入 **test**。針對 Version (版本) 輸入 1。

Note

AWS IoT Greengrass 不支援 \$LATEST 版本的 Lambda 別名。

11. 選擇建立。

An alias is a pointer to one or two versions. Choose each version that you want the alias to point to.

Name*

test

Description

Version*

1

You can shift traffic between two versions, based on weights (%) that you assign. Click [here](#) to learn more.

Additional version

Cancel

Create

您現在可以將 Lambda 函數新增至 Greengrass 群組。

步驟 3：將 Lambda 函數新增至 Greengrass 群組

在此步驟中，您將會將此函數新增到您的群組，以及設定函數的生命週期。

首先，將 Lambda 函數新增至 Greengrass 群組。

1. 在 AWS IoT 主控台導覽窗格的管理下，展開 Greengrass 裝置，然後選擇群組 (V1)。
2. 選擇您要新增 Lambda 函數的 Greengrass 群組。
3. 在群組組態頁面上，選擇 Lambda 函數索引標籤。
4. 在我的 Lambda 函數區段下，選擇新增。
5. 在新增 Lambda 函數頁面上，選擇 Lambda 函數。選取 **TestLRA**。
6. 選擇 Lambda 函數版本。
7. 在 Lambda 函數組態區段中，選取系統使用者和群組以及 Lambda 函數容器化。

接著，設定 Lambda 函數的生命週期。

8. 針對 Timeout (逾時)，選擇 30 seconds (30 秒)。

⚠ Important

使用本機資源（如本程序所述）的 Lambda 函數必須在 Greengrass 容器中執行。否則，部署會在您嘗試部署函數時失敗。如需詳細資訊，請參閱[容器化](#)。

9. 在頁面底部，選擇新增 Lambda 函數。

步驟 4：新增本機資源至 Greengrass 群組

在此步驟中，您將新增本機磁碟區資源至 Greengrass 群組，並授予函數對該項資源的讀取及寫入存取權。本機資源擁有群組等級範圍。您可以授予群組中任何 Lambda 函數存取資源的許可。

1. 在群組組態頁面上，選擇資源索引標籤。
2. 在本機資源區段下，選擇新增。
3. 在新增本機資源頁面上，使用下列值。
 - a. 針對資源名稱，輸入 **testDirectory**。
 - b. 針對 Resource type (資源類型)，選擇 Volume (磁碟區)。
 - c. 針對本機裝置路徑，輸入 **/src/LRAtest**。此路徑必須存在於主機作業系統。

本機裝置路徑是核心裝置檔案系統上資源的本機絕對路徑。此位置位於函數執行所在的[容器](#)外。此路徑不能以 **/sys** 做為開頭。

- d. 針對 Destination path (目的地路徑)，輸入 **/dest/LRAtest**。此路徑必須存在於主機作業系統。

目的地路徑是 Lambda 命名空間中資源的絕對路徑。此位置位於函數執行所在的容器內。

- e. 在系統群組擁有者和檔案存取許可下，選取自動新增擁有資源之系統群組的檔案系統許可。

系統群組擁有者和檔案存取許可選項可讓您將其他檔案存取許可授予 Lambda 程序。如需詳細資訊，請參閱[群組擁有者檔案存取許可](#)。

4. 選擇 Add resource (新增資源)。資源頁面會顯示新的 testDirectory 資源。

步驟 5：將訂閱新增到 Greengrass 群組

在此步驟中，您將新增兩個訂閱到 Greengrass 群組。這些訂閱可啟用 Lambda 函數與之間的雙向通訊 AWS IoT。

首先，為 Lambda 函數建立要傳送訊息的訂閱 AWS IoT。

1. 在群組組態頁面上，選擇訂閱索引標籤。
2. 選擇新增。
3. 在建立訂閱頁面上，設定來源和目標，如下所示：
 - a. 針對來源類型，選擇 Lambda 函數，然後選擇 TestLRA。
 - b. 針對目標類型，選擇服務，然後選擇 IoT 雲端。
 - c. 針對主題篩選條件，輸入 **LRA/test**，然後選擇建立訂閱。
4. 訂閱頁面會顯示新的訂閱。

接著，設定從中叫用函數的訂閱 AWS IoT。

5. 在訂閱頁面上，選擇新增訂閱。
6. 在 Select your source and target (選擇您的來源和目標) 頁面設定來源和目標，如下所示：
 - a. 針對來源類型，選擇 Lambda 函數，然後選擇 IoT Cloud。
 - b. 針對目標類型，選擇服務，然後選擇 TestLRA。
 - c. 選擇下一步。
7. 在 Filter your data with a topic (使用主題篩選您的資料) 頁面上，對於 Topic filter (主題篩選條件)，輸入 **invoke/LRAFunction**，然後選擇 Next (下一步)。
8. 選擇完成。訂閱頁面會顯示這兩個的訂閱。

步驟 6：部署 AWS IoT Greengrass 群組

在此步驟中，您將部署目前群組定義的版本。

1. 確定 AWS IoT Greengrass 核心正在執行。如果需要，請在您的 Raspberry Pi 終端機執行以下命令。
 - a. 檢查精靈是否有在運作：

```
ps aux | grep -E 'greengrass.*daemon'
```

若輸出的 root 含有 /greengrass/ggc/packages/1.11.6/bin/daemon 項目，則精靈有在運作。

 Note

路徑中的版本取決於核心裝置上安裝的核心 AWS IoT Greengrass 軟體版本。

- b. 若要啟動協助程式：

```
cd /greengrass/ggc/core/  
sudo ./greengrassd start
```

2. 在群組組態頁面上，選擇部署。

 Note

如果您在沒有容器化的情況下執行 Lambda 函數，並嘗試存取連接的本機資源，則部署會失敗。

3. 如果出現提示，請在 Lambda 函數索引標籤的系統 Lambda 函數下，選取 IP 偵測器，然後選取編輯，然後自動偵測。

這可讓裝置自動取得核心的連接資訊，例如 IP 位址、DNS、連接埠編號。建議自動偵測，但 AWS IoT Greengrass 也支援手動指定的端點。只會在第一次部署群組時收到復原方法的提示。

 Note

如果出現提示，請授予許可來建立 [Greengrass 服務角色](#)，並將其與目前 AWS 帳戶中的建立關聯 AWS 區域。此角色允許 AWS IoT Greengrass 存取 AWS 服務中的資源。

此部署頁面會顯示部署時間戳記、版本 ID 和狀態。完成時，部署狀態為已完成。

如需故障診斷協助，請參閱[疑難排解](#)。

測試本機資源存取

現在您可以驗證本機資源存取是否已正確設定。若要測試，請訂閱 `LRA/test` 主題並發佈到 `invoke/LRAFunction` 主題。如果 Lambda 函數將預期的承載傳送至 `invoke/LRAFunction`，則測試成功 AWS IoT。

1. 從 AWS IoT 主控台導覽功能表的測試下，選擇 MQTT 測試用戶端。
2. 在訂閱主題下，針對主題篩選條件，輸入 **LRA/test**。
3. 在其他資訊下，針對 MQTT 承載顯示，選取將承載顯示為字串。
4. 選擇 Subscribe (訂閱)。您的 Lambda 函數會發佈至 LRA/測試主題。

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.

lra/test

▼ Additional configuration

Number of messages to keep
The MQTT test client keeps this many of the most recent messages published to a topic that matches this topic filter.

100

Quality of service
When subscribing to a topic, quality of service 0 will be chosen by default.

☒ Quality of Service 0 - Message will be delivered at most once

☐ Quality of Service 1 - Message will be delivered at least once

MQTT payload display

☐ Auto-format JSON payloads (improves readability)

☒ Display payloads as strings (more accurate)

☐ Display raw payloads (displays binary data as hexadecimal values)

Subscribe

5. 在發佈至主題下，在主題名稱中輸入 **invoke/LRAFunction**，然後選擇發佈以叫用您的 Lambda 函數。如果頁面會顯示該函數的三個承載訊息，則表示測試成功。

Subscribe to a topic

Publish to a topic

Topic name

The topic name identifies the message. The message payload will be published to this topic with a Quality of Service (QoS) of 0.

Q

invoke/LRAFunction

X

Message payload

```
{  
  "message": "Hello from AWS IoT console"  
}
```

► Additional configuration

Publish

Subscriptions

lra/test

▼

X

lra/test

▼

May 03, 2021, 12:09:18 (UTC-0400)

Successfully write to a file.

lra/test

▼

May 03, 2021, 12:09:06 (UTC-0400)

```
posix.stat_result(st_mode=16893, st_ino=171142L, st_dev=45831L, st_nlink=2, st_uid=0, st  
_gid=119, st_size=4096L, st_atime=1620054520, st_mtime=1620058120, st_ctime=1620058120)
```

lra/test

▼

May 03, 2021, 12:09:04 (UTC-0400)

Sent from Greengrass Core.

Pause

Clear

Export

Edit

Lambda 函數建立的測試檔案位於 Greengrass 核心裝置上的 `/src/LRAtest` 目錄中。雖然 Lambda 函數會寫入 `/dest/LRAtest` 目錄中的檔案，但該檔案只會顯示在 Lambda 命名空間中。您無法在一般 Linux 命名空間中看到它。目的地路徑的所有變更都會反映在檔案系統的來源路徑上。

如需故障診斷協助，請參閱[疑難排解](#)。

執行機器學習推論

此功能適用於 AWS IoT Greengrass Core v1.6 或更新版本。

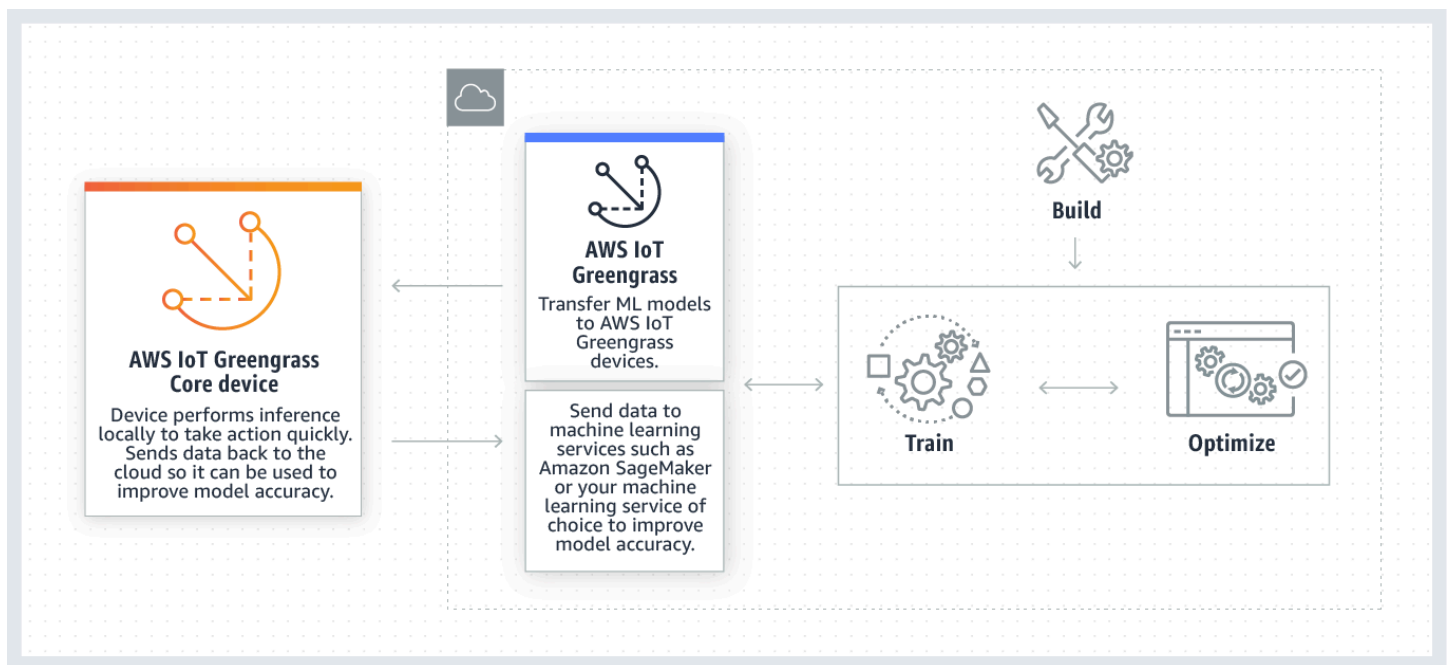
透過 AWS IoT Greengrass，您可以使用雲端訓練模型，在本機產生的資料上執行邊緣機器學習 (ML) 推論。您可以從低延遲時間和節省執行本機推論成本中獲益，但仍同時充分利用雲端運算能力進行訓練模型和處理複雜地處理。

若要取得已執行的本機推論，請參閱 [the section called “如何設定機器學習推論”](#)。

AWS IoT Greengrass ML 推論的運作方式

您可以在任何地方訓練推論模型，將它們部署為 Greengrass 群組中的機器學習資源，然後從 Greengrass Lambda 函數存取它們。例如，您可以在 [SageMaker AI](#) 中建置和訓練深度學習模型，並將其部署到您的 Greengrass 核心。然後，您的 Lambda 函數可以使用本機模型在連線裝置上執行推論，並將新的訓練資料傳回雲端。

下圖顯示 AWS IoT Greengrass ML 推論工作流程。



AWS IoT Greengrass ML 推論可簡化 ML 工作流程的每個步驟，包括：

- 建立和部署 Machine Learning 架構的原型。
- 存取雲端訓練模型和將其部署到 Greengrass 核心裝置。
- 建立可以存取硬體 Accelerator (例如 GPU 和 FPGA) 做為 [本機資源](#) 的推論應用程式。

機器學習資源

機器學習資源代表部署到 AWS IoT Greengrass 核心的雲端訓練推論模型。若要部署機器學習資源，請先將資源新增至 Greengrass 群組，然後定義群組中的 Lambda 函數如何存取這些資源。在群組部署期間，從雲端 AWS IoT Greengrass 擷取來源模型套件，並將其擷取至 Lambda 執行時間命名空間內的目錄。然後，Greengrass Lambda 函數會使用本機部署的模型來執行推論。

若要更新本機部署模型，首先，請更新對應到 machine learning 資源的來源模型 (在雲端中)，然後部署群組。在部署期間，AWS IoT Greengrass 會檢查來源是否有變更。如果偵測到變更，則會 AWS IoT Greengrass 更新本機模型。

支援的模型來源

AWS IoT Greengrass 支援機器學習資源的 SageMaker AI 和 Amazon S3 模型來源。

以下要求適用於模型來源：

- 存放 SageMaker AI 和 Amazon S3 模型來源的 S3 儲存貯體不得使用 SSE-C 加密。Amazon S3 對於使用伺服器端加密的儲存貯體，AWS IoT Greengrass ML 推論目前僅支援 SSE-S3 或 SSE-KMS 加密選項。如需伺服器端加密選項的詳細資訊，請參閱《Amazon Simple Storage Service 使用者指南》中的[使用伺服器端加密保護資料](#)。
- 存放 SageMaker AI 和 Amazon S3 模型來源的 S3 儲存貯體名稱不得包含句點 (.)。Amazon S3 如需詳細資訊，請參閱《Amazon Simple Storage Service 使用者指南》中的使用虛擬託管式儲存貯體搭配 SSL 的[儲存貯體命名規則](#)。
- 服務層級 AWS 區域 支援必須同時適用於 [AWS IoT Greengrass](#) 和 [SageMaker AI](#)。目前，AWS IoT Greengrass 在下列區域中支援 SageMaker AI 模型：
 - 美國東部 (俄亥俄)
 - 美國東部 (維吉尼亞北部)
 - 美國西部 (奧勒岡)
 - 亞太區域 (孟買)
 - 亞太區域 (首爾)
 - 亞太區域 (新加坡)
 - 亞太地區 (雪梨)
 - 亞太地區 (東京)
 - 歐洲 (法蘭克福)
 - 歐洲 (愛爾蘭)

- 歐洲 (倫敦)
- AWS IoT Greengrass 必須具有模型來源的 read 許可，如以下各節所述。

SageMaker AI

AWS IoT Greengrass 支援儲存為 SageMaker AI 訓練任務的模型。SageMaker AI 是一項全受管 ML 服務，可讓您使用內建或自訂演算法來建置和訓練模型。如需詳細資訊，請參閱 [《SageMaker AI 開發人員指南》中的什麼是 SageMaker AI？](#)。

如果您透過[建立名稱包含的儲存貯體](#)來設定 SageMaker AI 環境 sagemaker，則 AWS IoT Greengrass 具有足夠的許可來存取 SageMaker AI 訓練任務。由政策受管的 AWSGreengrassResourceAccessRolePolicy 允許存取名稱含有 sagemaker 字串的儲存貯體。此政策附屬於 [Greengrass 服務角色](#)。

否則，您必須將許可授予 AWS IoT Greengrass read 存放訓練任務的 儲存貯體。若要這樣做，請在服務角色中內嵌下列政策。您可以列出多個儲存貯體 ARN。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket"
      ]
    }
  ]
}
```

Amazon S3

AWS IoT Greengrass 支援以 tar.gz 或 .zip 檔案儲存在 Amazon S3 中的模型。

若要讓 AWS IoT Greengrass 存取存放在 Amazon S3 儲存貯體中的模型，您必須執行下列其中一項動作，授予存取儲存貯體的 AWS IoT Greengrass read 許可：

- 存放模型於名稱含有 greengrass 的儲存貯體中。

由政策受管的 `AWSGreengrassResourceAccessRolePolicy` 允許存取名稱含有 `greengrass` 字串的儲存貯體。此政策附屬於 [Greengrass 服務角色](#)。

- 在 Greengrass 服務角色中內嵌政策。

如果您的儲存貯體名稱不包含 `greengrass`，請將以下內嵌政策新增到服務角色中。您可以列出多個儲存貯體 ARN。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket"
      ]
    }
  ]
}
```

如需詳細資訊，請參閱《IAM 使用者指南》中的 [內嵌內嵌政策](#)。

要求

以下要求適用於建立和使用 machine learning 資源：

- 您必須使用 AWS IoT Greengrass Core v1.6 或更新版本。
- 使用者定義的 Lambda 函數可以對資源執行 `read` 或 `read and write` 操作。不提供其他操作的權限。附屬 Lambda 函數的容器化模式會決定您如何設定存取許可。如需詳細資訊，請參閱 [the section called “存取機器學習資源”](#)。
- 在 Greengrass 核心裝置的作業系統中，您必須提供完整的資源路徑。
- 資源名稱或 ID 字元達最大限制 128 個字元時，必須使用模式 `[a-zA-Z0-9:_-]+`。

適用於 ML 推論的執行時間和程式庫

您可以使用下列 ML 執行時間和程式庫 AWS IoT Greengrass。

- [Amazon SageMaker Neo 深度學習執行時間](#)
- Apache MXNet
- TensorFlow

這些執行時間和程式庫可以安裝在 NVIDIA Jetson TX2、Intel Atom 和 Raspberry Pi 平台上。如需下載資訊，請參閱 [the section called “支援的 Machine Learning 執行時間和程式庫”](#)。您可以將其直接安裝在核心裝置上。

請務必閱讀以下相容性和限制的內容。

SageMaker AI Neo 深度學習執行期

您可以使用 SageMaker AI Neo 深度學習執行期，在 AWS IoT Greengrass 裝置上透過最佳化的機器學習模型執行推論。這些模型使用 SageMaker AI Neo 深度學習編譯器進行最佳化，以改善機器學習推論預測速度。如需 SageMaker AI 中模型最佳化的詳細資訊，請參閱 [SageMaker AI Neo 文件](#)。

Note

目前，您只能在特定 Amazon Web Services 區域中使用 Neo 深度學習編譯器來最佳化機器學習模型。不過，您可以在支援 AWS 區域 AWS IoT Greengrass 核心的每個中使用 Neo 深度學習執行期與最佳化模型。如需詳細資訊，請參閱[如何設定最佳化的機器學習推論](#)。

MXNet 版本控制

Apache MXNet 目前不保證支援正向相容性，因此您使用較新的架構版本訓練模型時，可能會比使用舊版不順。為了避免模型訓練與模型服務階段兩者間的衝突，以及提供一致的端對端體驗，請在這兩者階段使用相同的 MXNet 架構版本。

在 Raspberry Pi 中的 MXNet

存取本機 MXNet 模型的 Greengrass Lambda 函數必須設定下列環境變數：

```
MXNET_ENGINE_TYPE=NativeEngine
```

您可以在函數程式碼中設定環境變數，或將其新增至特定群組函數的組態中。如需將其新增為組態設定的範例，請參閱[步驟](#)。

Note

對於一般使用 MXNet 架構來說，如執行第三方程式碼範例，其環境變數必須設定在 Raspberry Pi 中。

在 Raspberry Pi 上的 TensorFlow 模型服務限制

以下是根據我們在 Raspberry Pi 平台上測試 TensorFlow 32 位元 Arm 程式庫所得結果而推論出來的改善建議。這些建議僅供進階使用者參考，不具任何保證。

- 使用 [檢查點](#) 格式訓練的模型應在服務之前「凍結」為協定緩衝區格式。如需範例，請參閱 [TensorFlow-Slim 映像分類模型程式庫](#)。
- 請勿在訓練或推論程式碼中使用 TF-Estimator 和 TF-Slim 程式庫。反之，請使用 .pb 檔案模型載入模式，如下列範例所示。

```
graph = tf.Graph()
graph_def = tf.GraphDef()
graph_def.ParseFromString(pb_file.read())
with graph.as_default():
    tf.import_graph_def(graph_def)
```

Note

如需 TensorFlow 支援平台的詳細資訊，請參閱 TensorFlow 文件中的[安裝 TensorFlow](#)。

從 Lambda 函數存取機器學習資源

使用者定義的 Lambda 函數可以存取機器學習資源，以在 AWS IoT Greengrass 核心上執行本機推論。機器學習資源包含培訓過的模型，以及下載至核心裝置的其他成品。

若要允許 Lambda 函數存取核心上的機器學習資源，您必須將資源連接至 Lambda 函數並定義存取許可。附屬（或連接的）Lambda 函數的[容器化模式](#)會決定您的執行方式。

機器學習資源的存取權限

從 AWS IoT Greengrass Core v1.10.0 開始，您可以定義機器學習資源的資源擁有者。資源擁有者代表 AWS IoT Greengrass 用來下載資源成品的作業系統群組和許可。如果未定義資源擁有者，則僅限根存取的下載資源成品。

- 如果非容器化 Lambda 函數存取機器學習資源，您必須定義資源擁有者，因為容器沒有許可控制。非容器化 Lambda 函數可以繼承資源擁有者許可，並使用它們來存取資源。
- 如果只有容器化 Lambda 函數存取資源，建議您使用函數層級許可，而不是定義資源擁有者。

資源擁有者屬性

資源擁有者指定群組擁有者和群組擁有者權限。

群組擁有者。核心裝置上現有 Linux 作業系統群組的 ID (GID)。群組許可會新增至 Lambda 程序。具體而言，GID 會新增至 Lambda 函數的補充群組 IDs。

如果 Greengrass 群組中的 Lambda 函數設定為[執行為](#)與機器學習資源的資源擁有者相同的作業系統群組，則該資源必須連接到 Lambda 函數。否則，部署會失敗，因為此組態會提供 Lambda 函數可在 AWS IoT Greengrass 未經授權的情況下用來存取資源的隱含許可。如果 Lambda 函數以根 (UID=0) 執行，則會略過部署驗證檢查。

我們建議您使用其他資源、Lambda 函數或 Greengrass 核心上的檔案未使用的作業系統群組。使用共用作業系統群組可讓連接的 Lambda 函數獲得比所需更多的存取許可。如果您使用共用作業系統群組，則連接的 Lambda 函數也必須連接至使用共用作業系統群組的所有機器學習資源。否則，部署會失敗。

群組擁有者權限。新增至 Lambda 程序的唯讀或讀取和寫入許可。

非容器化 Lambda 函數必須繼承資源的這些存取許可。容器化 Lambda 函數可以繼承這些資源層級許可或定義函數層級許可。如果他們定義了函數層級權限，則權限必須與資源層級權限相同或更加嚴格。

下表顯示支援的存取權限組態。

GGC v1.10 or later

屬性	如果只有容器化 Lambda 函數存取資源	如果任何非容器化 Lambda 函數存取資源
函數層級屬性		
權限 (讀取/寫入)	除非資源定義資源擁有者，否則為必要項目。如果已定義資源擁有者，則函數層級權限必須與資源擁有者權限相同或更加嚴格。 如果只有容器化 Lambda 函數存取資源，建議您不要定義資源擁有者。	非容器化 Lambda 函數： 不支援。非容器化 Lambda 函數必須繼承資源層級許可。 容器化 Lambda 函數： 可選，但必須與資源層級權限相同或更加嚴格。
資源層級屬性		
資源擁有者	可選 (不建議使用)。	必要。
權限 (讀取/寫入)	可選 (不建議使用)。	必要。

GGC v1.9 or earlier

屬性	如果只有容器化 Lambda 函數存取資源	如果任何非容器化 Lambda 函數存取資源
函數層級屬性		
權限 (讀取/寫入)	必要。	不支援。
資源層級屬性		
資源擁有者	不支援。	不支援。
權限 (讀取/寫入)	不支援。	不支援。

Note

當您使用 AWS IoT Greengrass API 設定 Lambda 函數和資源時，也需要函數層級 `ResourceId` 屬性。`ResourceId` 屬性會將機器學習資源連接至 Lambda 函數。

定義 Lambda 函數的存取許可（主控台）

在 AWS IoT 主控台中，您可以在設定機器學習資源或將機器學習資源附加至 Lambda 函數時定義存取許可。

容器化 Lambda 函數

如果只有容器化 Lambda 函數連接到機器學習資源：

- 選擇無系統群組做為機器學習資源的資源擁有者。當只有容器化 Lambda 函數存取機器學習資源時，建議使用此設定。否則，您可以為連接的 Lambda 函數提供更多的存取許可。

非容器化 Lambda 函數（需要 GGC 1.10 版或更新版本）

如果任何非容器化 Lambda 函數連接至機器學習資源：

- 指定系統群組 ID (GID) 做為機器學習資源的資源擁有者。選擇指定系統群組和許可，然後輸入 GID。您可以在核心裝置上使用 `getent group` 命令來查詢系統群組的 ID。
- 選擇系統群組許可的唯讀存取權或讀取和寫入存取權。

定義 Lambda 函數 (API) 的存取許可

在 AWS IoT Greengrass API 中，您可以在 Lambda 函數的 `ResourceAccessPolicy` 屬性或資源的 `OwnerSetting` 屬性中定義機器學習資源的許可。

容器化 Lambda 函數

如果只有容器化 Lambda 函數連接到機器學習資源：

- 對於容器化 Lambda 函數，請在 `ResourceAccessPolicies` 屬性的 `Permission` 屬性中定義存取許可。例如：


```

"Functions": [
  {
    "Id": "my-containerized-function",
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:function-name:alias-or-version",
    "FunctionConfiguration": {
      "Environment": {
        "ResourceAccessPolicies": [
          {
            "ResourceId": "my-resource-id",
            "Permission": "ro-or-rw"
          }
        ]
      },
      "MemorySize": 512,
      "Pinned": true,
      "Timeout": 5
    }
  }
]

```

- 對於機器學習資源，請省略 OwnerSetting 屬性。例如：

```

"Resources": [
  {
    "Id": "my-resource-id",
    "Name": "my-resource-name",
    "ResourceDataContainer": {
      "S3MachineLearningModelResourceData": {
        "DestinationPath": "/local-destination-path",
        "S3Uri": "s3://uri-to-resource-package"
      }
    }
  }
]

```

當只有容器化 Lambda 函數存取機器學習資源時，建議使用此組態。否則，您可以為連接的 Lambda 函數提供更多的存取許可。

非容器化 Lambda 函數（需要 GGC 1.10 版或更新版本）

如果任何非容器化 Lambda 函數連接至機器學習資源：

- 對於非容器化 Lambda 函數，請省略 中的 `Permission` 屬性 `ResourceAccessPolicies`。此設定是必要的，並允許函數繼承資源層級權限。例如：

```
"Functions": [
  {
    "Id": "my-non-containerized-function",
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:function-name:alias-or-version",
    "FunctionConfiguration": {
      "Environment": {
        "Execution": {
          "IsolationMode": "NoContainer",
        },
        "ResourceAccessPolicies": [
          {
            "ResourceId": "my-resource-id"
          }
        ]
      },
      "Pinned": true,
      "Timeout": 5
    }
  }
]
```

- 對於也存取機器學習資源的容器化 Lambda 函數，請省略 中的 `Permission` 屬性，`ResourceAccessPolicies`或定義與資源層級許可相同或更嚴格的許可。例如：

```
"Functions": [
  {
    "Id": "my-containerized-function",
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:function-name:alias-or-version",
    "FunctionConfiguration": {
      "Environment": {
        "ResourceAccessPolicies": [
          {
            "ResourceId": "my-resource-id",
            "Permission": "ro-or-rw" // Optional, but cannot exceed
            the GroupPermission defined for the resource.
          }
        ]
      }
    }
  }
]
```

```

        }
      ]
    },
    "MemorySize": 512,
    "Pinned": true,
    "Timeout": 5
  }
}
]

```

- 對於機器學習資源，定義 OwnerSetting 屬性，包括子項 GroupOwner 和 GroupPermission 屬性。例如：

```

"Resources": [
  {
    "Id": "my-resource-id",
    "Name": "my-resource-name",
    "ResourceDataContainer": {
      "S3MachineLearningModelResourceData": {
        "DestinationPath": "/local-destination-path",
        "S3Uri": "s3://uri-to-resource-package",
        "OwnerSetting": {
          "GroupOwner": "os-group-id",
          "GroupPermission": "ro-or-rw"
        }
      }
    }
  }
]

```

從 Lambda 函數程式碼存取機器學習資源

使用者定義的 Lambda 函數使用平台特定的作業系統介面來存取核心裝置上的機器學習資源。

GGC v1.10 or later

對於容器化 Lambda 函數，資源會掛載在 Greengrass 容器內，並可在為資源定義的本機目的地路徑中使用。對於非容器化 Lambda 函數，資源會符號連結到 Lambda 特定的工作目錄，並在 Lambda 程序中傳遞至 AWS_GG_RESOURCE_PREFIX 環境變數。

為了取得機器學習資源下載成品的路徑，Lambda 函數會將 AWS_GG_RESOURCE_PREFIX 環境變數附加至為資源定義的本機目的地路徑。對於容器化 Lambda 函數，傳回的值是單一正斜線 (/)。

```
resourcePath = os.getenv("AWS_GG_RESOURCE_PREFIX") + "/destination-path"
with open(resourcePath, 'r') as f:
    # load_model(f)
```

GGC v1.9 or earlier

機器學習資源的下載成品位於為資源定義的本機目的地路徑中。只有容器化 Lambda 函數可以存取 Core v1.9 和更早版本的 AWS IoT Greengrass 機器學習資源。

```
resourcePath = "/local-destination-path"
with open(resourcePath, 'r') as f:
    # load_model(f)
```

模型載入實作取決於您的 ML 程式庫。

疑難排解

使用下列資訊，協助疑難排解存取機器學習資源的問題。

主題

- [InvalidMLModelOwner - GroupOwnerSetting 在 ML 模型資源中提供，但是 GroupOwner 或 GroupPermission 不存在](#)
- [附加 Machine Learning 資源時，NoContainer 函數無法設定權限。<function-arn> 係指在資源存取政策中，具有權限 <ro/rw> 的 Machine Learning 資源 <resource-id>。](#)
- [函數 <function-arn> 係指在資源 <resource-id> ResourceAccessPolicy 和資源 OwnerSetting 中具有遺失權限的 Machine Learning 資源。](#)
- [函數 <function-arn> 係指具有 \"rw\" 權限的 Machine Learning 資源 <resource-id>，而資源擁有者設定 GroupPermission 只允許 \"ro\"。](#)
- [NoContainer 函數 <function-arn> 係指巢狀目的地路徑的資源。](#)
- [Lambda <function-arn> 透過共用相同群組擁有者 ID 來獲得資源 <resource-id> 的存取權](#)

InvalidMLModelOwner - GroupOwnerSetting 在 ML 模型資源中提供，但是 GroupOwner 或 GroupPermission 不存在

解決方案：如果機器學習資源包含 [ResourceDownloadOwnerSetting](#) 物件，但未定義必要的 GroupOwner 或 GroupPermission 屬性，便會收到此錯誤訊息。若要解決此問題，請定義遺失的屬性。

附加 Machine Learning 資源時，NoContainer 函數無法設定權限。<function-arn> 係指在資源存取政策中，具有權限 <ro/rw> 的 Machine Learning 資源 <resource-id>。

解決方案：如果非容器化 Lambda 函數指定機器學習資源的函數層級許可，您會收到此錯誤。非容器化函數必須從機器學習資源上定義的資源擁有者權限繼承權限。若要解決此問題，請選擇[繼承資源擁有者許可](#)（主控台），或從 Lambda 函數的資源存取政策 (API) 移除許可。

函數 <function-arn> 係指在資源 <resource-id> ResourceAccessPolicy 和資源 OwnerSetting 中具有遺失權限的 Machine Learning 資源。

解決方案：如果未針對連接的 Lambda 函數或資源設定機器學習資源的許可，您會收到此錯誤。若要解決此問題，請在 Lambda 函數的 [ResourceAccessPolicy](#) 屬性或資源的 [OwnerSetting](#) 屬性中設定許可。

函數 <function-arn> 係指具有 \"rw\" 權限的 Machine Learning 資源 <resource-id>，而資源擁有者設定 GroupPermission 只允許 \"ro\"。

解決方案：如果為連接的 Lambda 函數定義的存取許可超過為機器學習資源定義的資源擁有者許可，您會收到此錯誤。若要解決此問題，請為 Lambda 函數設定更嚴格的許可，或為資源擁有者設定較不嚴格的許可。

NoContainer 函數 <function-arn> 係指巢狀目的地路徑的資源。

解決方案：如果連接到非容器化 Lambda 函數的多個機器學習資源使用相同的目的地路徑或巢狀目的地路徑，則會收到此錯誤。若要解決此問題，請為資源指定個別目的地路徑。

Lambda <function-arn> 透過共用相同群組擁有者 ID 來獲得資源 <resource-id> 的存取權

解決方案：runtime.log 如果將相同的作業系統群組指定為 Lambda 函數的 [Run as identity](#) 和機器學習資源的資源擁有者，但資源未連接至 Lambda 函數，您會在中收到此錯誤。此組態提供 Lambda 函數隱含許可，其可在未經授權的情況下用來存取資源 AWS IoT Greengrass。

若要解決此問題，請針對其中一個屬性使用不同的作業系統群組，或將機器學習資源連接至 Lambda 函數。

另請參閱

- [執行機器學習推論](#)
- [the section called “如何設定機器學習推論”](#)
- [the section called “如何設定最佳化的機器學習推論”](#)
- [AWS IoT Greengrass Version 1 API 參考](#)

如何使用 設定機器學習推論 AWS 管理主控台

若要遵循本教學課程中的步驟，您需要 AWS IoT Greengrass Core v1.10 或更新版本。

您可以在 Greengrass 核心裝置本機上使用本機產生的資料來執行機器學習 (ML) 推論。如需要要求和限制的詳細資訊，請參閱 [執行機器學習推論](#)。

本教學課程說明如何使用 AWS 管理主控台 來設定 Greengrass 群組，以執行 Lambda 推論應用程式，在本機辨識攝影機的影像，而無需將資料傳送至雲端。推論應用程式在 Raspberry Pi 存取攝影機模組，並使用開放來源 [SqueezeNet](#) 模型推論。

本教學課程所述以下高階執行步驟：

1. [設定 Raspberry Pi](#)
2. [安裝 MXNet 架構](#)
3. [建立模型套件](#)
4. [建立和發佈 Lambda 函數](#)

5. [將 Lambda 函數新增至群組](#)
6. [新增資源到群組](#)
7. [新增訂閱到群組](#)
8. [部署群組](#)
9. [測試應用程式](#)

先決條件

為完成此教學課程您需要：

- Raspberry Pi 4 模型 B 或 Raspberry Pi 3 模型 B/B+，設定與 搭配使用 AWS IoT Greengrass。若要使用 設定 Raspberry Pi AWS IoT Greengrass，請執行 [Greengrass 裝置設定](#) 指令碼，或確認您已完成的 [單元 1](#) 和 [單元 2 入門 AWS IoT Greengrass](#)。

Note

Raspberry Pi 可能需要 2.5A [電源供應器](#) 來執行通常用於影像分類的深度學習架構。額定值較低的電源供應器可能會導致裝置重新啟動。

- [Raspberry Pi 攝影機模組 V2 - 8 百萬像素、1080p](#)。如需如何設定攝影機的資訊，請參閱 Raspberry Pi 文件中的 [連接攝影機](#)。
- Greengrass 群組和 Greengrass 核心。如需如何建立 Greengrass 群組或核心的詳細資訊，請參閱 [入門 AWS IoT Greengrass](#)。

Note

本教學課程使用 Raspberry Pi，但 AWS IoT Greengrass 支援其他平台，例如 [Intel Atom](#) 和 [NVIDIA Jetson TX2](#)。在 Jetson TX2 的範例中，您可以使用靜態影像，而非相機的串流影像。如果使用 Jetson TX2 範例，您可能需要安裝 Python 3.6 而不是 Python 3.7。如需有關設定裝置以便您可以安裝 AWS IoT Greengrass 核心軟體的資訊，請參閱 [the section called “設定其他裝置”](#)。

對於 AWS IoT Greengrass 不支援的第三方平台，您必須在非容器化模式下執行 Lambda 函數。若要在非容器化模式下執行，您必須以根身分執行 Lambda 函數。如需詳細資訊，請參閱 [the section called “選擇 Lambda 函數容器化時的考量事項”](#) 及 [the section called “為群組中的 Lambda 函數設定預設存取身分”](#)。

步驟一：設定 Raspberry Pi

在此步驟中，請安裝 Raspbian 作業系統的更新、安裝相機模組軟體和 Python 相依性，以及啟用相機界面。

在您的 Raspberry Pi 終端機執行以下命令。

1. 安裝 Raspbian 的更新。

```
sudo apt-get update
sudo apt-get dist-upgrade
```

2. 安裝攝影機模組的 picamera 界面和本單元其他所需的 Python 程式庫。

```
sudo apt-get install -y python3-dev python3-setuptools python3-pip python3-picamera
```

驗證安裝：

- 確保您的 Python 3.7 安裝包含 pip。

```
python3 -m pip
```

如果未安裝 pip，請從 [pip 網站](#) 下載它，然後執行以下命令。

```
python3 get-pip.py
```

- 請確保您的 Python 版本是 3.7 或更高版本。

```
python3 --version
```

如果輸出中列出較早的版本，請執行下列命令。

```
sudo apt-get install -y python3.7-dev
```

- 請確定 Setuptools 和 Picamera 已安裝成功。

```
sudo -u ggc_user bash -c 'python3 -c "import setuptools"'
sudo -u ggc_user bash -c 'python3 -c "import picamera"'
```

如果輸出中未包含錯誤，則表示驗證成功。

Note

如果裝置上安裝的 Python 可執行檔是 python3.7，請針對本教學課程中的命令使用 python3.7 而不是 python3。確保您的 pip 安裝對應到正確的 python3.7 或 python3 版本，以避免相依性錯誤。

3. 重新啟動 Raspberry Pi。

```
sudo reboot
```

4. 請開啟 Raspberry Pi 組態工具。

```
sudo raspi-config
```

5. 請使用箭頭鍵開啟 Interfacing Options (連接選項) 並啟用攝影機界面。如果出現提示，請允許重新啟動裝置。**6. 請使用以下命令測試攝影機建立。**

```
raspistill -v -o test.jpg
```

這會在 Raspberry Pi 上開啟預覽視窗、將名為 test.jpg 的圖片儲存至現行目錄，並在 Raspberry Pi 終端機中顯示相機的相關資訊。

步驟 2：安裝 MXNet 架構

在此步驟中，請於 Raspberry Pi 上安裝 MXNet 程式庫。

1. 遠端登入到 Raspberry Pi。

```
ssh pi@your-device-ip-address
```

2. 開啟 MXNet 文件、開啟[安裝 MXNet](#)，然後依照指示在裝置上安裝 MXNet。**Note**

我們建議您安裝 1.5.0 版，並從本教學課程的來源建置 MXNet，以避免裝置衝突。

3. 安裝 MXNet 之後，請驗證下列組態：

- 確定 `ggc_user` 系統帳戶可以使用 MXNet 架構。

```
sudo -u ggc_user bash -c 'python3 -c "import mxnet"'
```

- 確保已安裝 NumPy。

```
sudo -u ggc_user bash -c 'python3 -c "import numpy"'
```

步驟 3：建立 MXNet 模型套件

在此步驟中，建立模型套件，其中包含要上傳至 Amazon Simple Storage Service (Amazon S3) 的範例預先訓練 MXNet 模型。AWS IoT Greengrass 可以使用來自 Amazon S3 的模型套件，前提是您使用 `tar.gz` 或 `zip` 格式。

1. 在您的電腦上，從 [the section called “機器學習範例”](#) 下載 Raspberry Pi 的 MXNet 範例。
2. 解壓縮所下載的 `mxnet-py3-armv7l.tar.gz` 檔案。
3. 導覽至 `squeezenet` 目錄。

```
cd path-to-downloaded-sample/mxnet-py3-armv7l/models/squeezenet
```

此目錄中的 `squeezenet.zip` 檔案是您的模型套件。其包含影像分類模型的 SqueezeNet 開放原始碼模型成品。稍後，您將此模型套件上傳到 Amazon S3。

步驟 4：建立和發佈 Lambda 函數

在此步驟中，建立 Lambda 函數部署套件和 Lambda 函數。然後，發佈函數版本和建立別名。

首先，建立 Lambda 函數部署套件。

1. 在您的電腦上，瀏覽至您在 [the section called “建立模型套件”](#) 中解壓縮的範例套件內的 `examples` 目錄。

```
cd path-to-downloaded-sample/mxnet-py3-armv7l/examples
```

`examples` 目錄包含函數程式碼和相依性。

- `greengrassObjectClassification.py` 是本教學課程中使用的推論程式碼。您可以使用此程式碼作為範本來建立您自己的推論函數。
- `greengrasssdk` 是適用於 Python 的核心 AWS IoT Greengrass SDK 的 1.5.0 版。

 Note

如果有新版本可用，您可以下載並升級部署套件中的 SDK 版本。如需詳細資訊，請參閱 GitHub 上的 [AWS IoT Greengrass 適用於 Python 的核心 SDK](#)。

2. 將 `examples` 目錄的內容壓縮成名為 `greengrassObjectClassification.zip` 的檔案。這是您的部署套件。

```
zip -r greengrassObjectClassification.zip .
```

 Note

確保 `.py` 檔案和相依性皆位於目錄的根中。

接著，建立 Lambda 函數。

3. 從 AWS IoT 主控台中，選擇函數和建立函數。
4. 選擇從頭開始撰寫，並使用下列值來建立函數：
 - 針對函數名稱，請輸入 **`greengrassObjectClassification`**。
 - 針對執行期，選擇 Python 3.7。

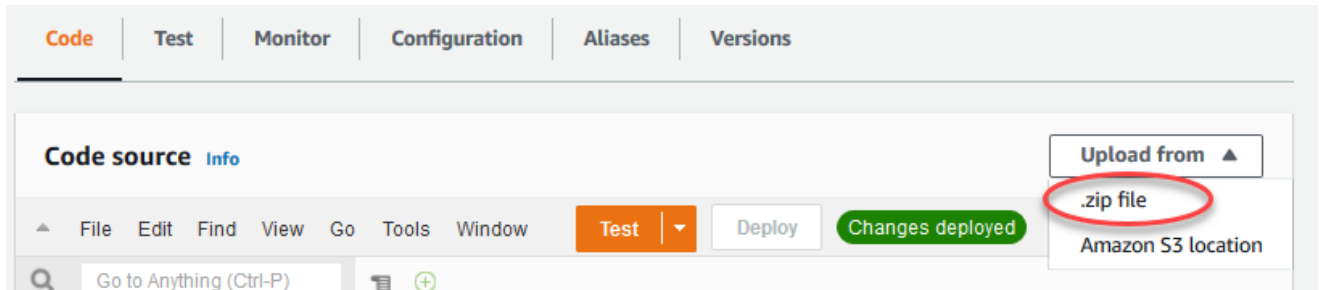
對於許可，請保留預設設定。這會建立授予基本 Lambda 許可的執行角色。不會使用此角色 AWS IoT Greengrass。

5. 選擇建立函數。

現在，上傳您的 Lambda 函數部署套件並註冊處理常式。

6. 選擇您的 Lambda 函數並上傳 Lambda 函數部署套件。

- a. 在程式碼索引標籤的程式碼來源下，選擇上傳來源。從下拉式清單中，選擇 .zip 檔案。



- b. 選擇上傳，然後選擇您的 `greengrassObjectClassification.zip` 部署套件。然後選擇 Save (儲存)。
- c. 在函數的程式碼索引標籤的執行時間設定下，選擇編輯，然後輸入下列值。
- 針對執行期，選擇 Python 3.7。
 - 對於 Handler (處理常式)，輸入 **`greengrassObjectClassification.function_handler`**。

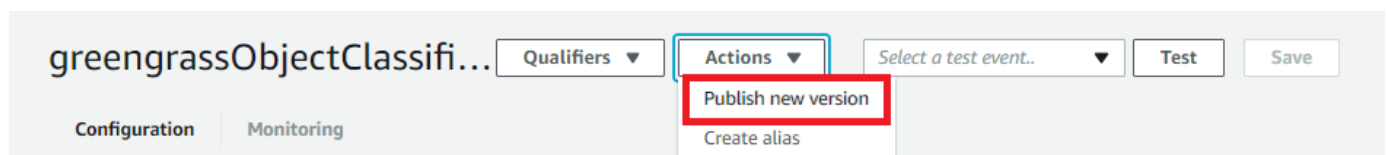
選擇儲存。

接著，發佈 Lambda 函數的第一個版本。然後，建立 [版本的別名](#)。

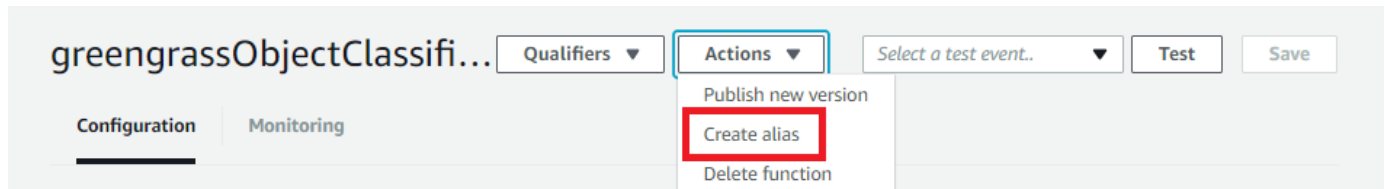
Note

Greengrass 群組可以依別名（建議）或版本參考 Lambda 函數。使用別名可讓您更輕鬆地管理程式碼更新，因為您不必在更新函數程式碼時變更訂閱資料表或群組定義。反之，您只需將別名指向新的函數版本。

7. 請從操作功能表中選擇發行新版本。



8. 針對 Version description (版本描述)，輸入 **First version**，然後選擇 Publish (發佈)。
9. 於 `greengrassObjectClassification`: 1 組態頁面，從操作功能表中選擇建立別名。



10. 在建立警示頁面上使用下列值：

- 對於名稱，輸入 **mlTest**。
- 針對 Version (版本) 輸入 **1**。

Note

AWS IoT Greengrass 不支援 \$LATEST 版本的 Lambda 別名。

11. 選擇儲存。

現在，將 Lambda 函數新增至 Greengrass 群組。

步驟 5：將 Lambda 函數新增至 Greengrass 群組

在此步驟中，將 Lambda 函數新增至 群組，然後設定其生命週期和環境變數。

首先，將 Lambda 函數新增至 Greengrass 群組。

1. 在 AWS IoT 主控台導覽窗格的管理下，展開 Greengrass 裝置，然後選擇群組 (V1)。
2. 從群組組態頁面，選擇 Lambda 函數索引標籤。
3. 在我的 Lambda 函數區段下，選擇新增。
4. 針對 Lambda 函數，選擇 greengrassObjectClassification。
5. 針對 Lambda 函數版本，選擇 Alias：mlTest。

接著，設定 Lambda 函數的生命週期和環境變數。

6. 在 Lambda 函數組態區段中，進行下列更新。

Note

建議您在沒有容器化的情況下執行 Lambda 函數，除非您的業務案例需要。這有助於啟用裝置 GPU 和攝影機的存取，而無需設定裝置資源。如果您在沒有容器化的情況下執行，您還必須授予 AWS IoT Greengrass Lambda 函數的根存取權。

a. 若要在沒有容器化的情況下執行：

- 針對系統使用者和群組，選擇 **Another user ID/group ID**。針對系統使用者 ID，輸入 **0**。針對系統群組 ID，輸入 **0**。

這可讓您的 Lambda 函數以根執行。如需以根身分執行的詳細資訊，請參閱 [the section called “為群組中的 Lambda 函數設定預設存取身分”](#)。

Tip

您也必須更新 config.json 檔案，以授予 Lambda 函數的根存取權。如需此程序，請參閱 [the section called “執行 Lambda 函數做為根”](#)。

- 針對 Lambda 函數容器化，選擇無容器。

如需在沒有容器化的情況下執行的詳細資訊，請參閱 [the section called “選擇 Lambda 函數容器化時的考量事項”](#)。

- 針對 Timeout (逾時)，輸入 **10 seconds**。
- 針對固定，選擇 **True**。

如需詳細資訊，請參閱 [the section called “生命週期組態”](#)。

b. 改為在容器化模式下執行：**Note**

除非您的商業案例需要，否則我們不建議在容器化模式下執行。

- 針對系統使用者和群組，選擇使用群組預設值。
- 針對 Lambda 函數容器化，選擇使用群組預設值。

- 針對 Memory limit (記憶體限制)，輸入 **96 MB**。
- 針對 Timeout (逾時)，輸入 **10 seconds**。
- 針對固定，選擇 True。

如需詳細資訊，請參閱[the section called “生命週期組態”](#)。

7. 在 Environment variables (環境變數) 下，建立索引鍵值對。與 Raspberry Pi 上 MXNet 模型互動的函數需要索引鍵/值組。

對於該索引鍵，請使用 MXNET_ENGINE_TYPE。對於此值，請使用 NaiveEngine。

Note

在您自己的使用者定義 Lambda 函數中，您可以選擇在函數程式碼中設定環境變數。

8. 保留所有其他屬性的預設值，然後選擇新增 Lambda 函數。

步驟 6：新增資源到 Greengrass 群組

在此步驟中，為攝影機模組和 ML 推論模型建立資源，並將資源與 Lambda 函數建立關聯。這可讓 Lambda 函數存取核心裝置上的資源。

Note

如果您在非容器化模式下執行，AWS IoT Greengrass 可以存取您的裝置 GPU 和攝影機，而無需設定這些裝置資源。

首先，為攝影機資源建立兩個本機裝置：一個使用共用記憶體，另一個使用裝置界面。如需本機資源存取的詳細資訊，請參閱[使用 Lambda 函數和連接器存取本機資源](#)。

1. 在群組組態頁面上，選擇資源索引標籤。
2. 在本機資源區段中，選擇新增本機資源。
3. 在新增本機資源頁面上，使用下列值：
 - 針對資源名稱，輸入 **videoCoreSharedMemory**。
 - 於資源類型選擇裝置。
 - 針對本機裝置路徑，輸入 **/dev/vcsm**。

裝置路徑為裝置資源的本機絕對路徑。此路徑只能參閱 `/dev` 底下的字元裝置或區塊型儲存設備。

- 針對系統群組擁有者和檔案存取許可，選擇自動新增擁有資源之系統群組的檔案系統許可。

系統群組擁有者和檔案存取許可選項可讓您將其他檔案存取許可授予 Lambda 程序。如需詳細資訊，請參閱[群組擁有者檔案存取許可](#)。

4. 接著，請您新增本機裝置資源給攝影機界面的資源。
5. 選擇新增本機資源。
6. 在新增本機資源頁面上，使用下列值：
 - 針對資源名稱，輸入 **videoCoreInterface**。
 - 於資源類型選擇裝置。
 - 針對本機裝置路徑，輸入 **/dev/vchiq**。
 - 針對系統群組擁有者和檔案存取許可，選擇自動新增擁有資源之系統群組的檔案系統許可。
7. 在頁面底部，選擇新增資源。

現在，請將推論模型做為機器學習資源。此步驟包含將 `squeezenet.zip` 模型套件上傳至 Amazon S3。

1. 在群組的資源索引標籤的 Machine Learning 區段下，選擇新增機器學習資源。
2. 在新增機器學習資源頁面上，針對資源名稱輸入 **squeezenet_model**。
3. 針對模型來源，選擇使用存放在 S3 中的模型，例如透過深度學習編譯器最佳化的模型。
4. 針對 S3 URI，輸入儲存 S3 儲存貯體的路徑。
5. 選擇 Browse S3 (瀏覽 S3)。這會開啟 Amazon S3 主控台的新標籤。
6. 在 Amazon S3 主控台索引標籤上，將 `squeezenet.zip` 檔案上傳至 S3 儲存貯體。如需詳細資訊，請參閱《Amazon Simple Storage Service 使用者指南》中的[如何將檔案和資料夾上傳至 S3 儲存貯體？](#)。

 Note

若要存取 S3 儲存貯體，您的儲存貯體名稱必須包含字串，**greengrass**且儲存貯體必須位於您使用的相同區域 AWS IoT Greengrass。選擇唯一名稱 (例如 **greengrass-bucket-*user-id-epoch-time***)。請勿在儲存貯體名稱中使用句號 (.)。

7. 在 AWS IoT Greengrass 主控台索引標籤上，尋找並選擇您的 S3 儲存貯體。尋找您已上傳的 `squeezenet.zip` 檔案，然後選擇 Select (選取)。您可能需要選擇 Refresh (重新整理) 以更新可用儲存貯體和檔案的清單。
8. 針對 Destination path (目的地路徑)，輸入 **/greengrass-machine-learning/mxnet/squeezenet**。

這是 Lambda 執行時間命名空間中本機模型的目的地。部署群組時，會 AWS IoT Greengrass 擷取來源模型套件，然後將內容擷取至指定的目錄。此教學課程的範例 Lambda 函數已設定為使用此路徑（在 `model_path` 變數中）。

9. 在系統群組擁有者和檔案存取許可下，選擇無系統群組。
10. 選擇 Add resource (新增資源)。

使用 SageMaker AI 訓練的模型

本教學課程使用存放在 Amazon S3 中的模型，但您也可以輕鬆使用 SageMaker AI 模型。AWS IoT Greengrass 主控台具有內建的 SageMaker AI 整合，因此您不需要手動將這些模型上傳至 Amazon S3。如需使用 SageMaker AI 模型的要求和限制，請參閱[the section called “支援的模型來源”](#)。

若要使用 SageMaker AI 模型：

- 針對模型來源，選擇使用在 SageMaker AI 中 AWS 訓練的模型，然後選擇模型訓練任務的名稱。
- 針對目的地路徑，輸入 Lambda 函數尋找模型的目錄路徑。

步驟 7：新增訂閱到 Greengrass 群組

在此步驟中，新增訂閱到群組。此訂閱可讓 Lambda 函數 AWS IoT 透過發佈至 MQTT 主題，將預測結果傳送至。

1. 在群組組態頁面上，選擇訂閱索引標籤，然後選擇新增訂閱。
2. 在訂閱詳細資訊頁面上，設定來源和目標，如下所示：

- a. 在來源類型中，選擇 Lambda 函數，然後選擇 greengrassObjectClassification。
 - b. 在目標類型中，選擇服務，然後選擇 IoT 雲端。
3. 在主題篩選條件中，輸入 **hello/world**，然後選擇建立訂閱。

步驟 8：部署 Greengrass 群組

在此步驟中，將群組定義的目前版本部署到 Greengrass 核心裝置。定義包含您新增的 Lambda 函數、資源和訂閱組態。

1. 確定 AWS IoT Greengrass 核心正在執行。如果需要，請在您的 Raspberry Pi 終端機執行以下命令。

- a. 檢查精靈是否有在運作：

```
ps aux | grep -E 'greengrass.*daemon'
```

若輸出的 root 含有 /greengrass/ggc/packages/1.11.6/bin/daemon 項目，則精靈有在運作。

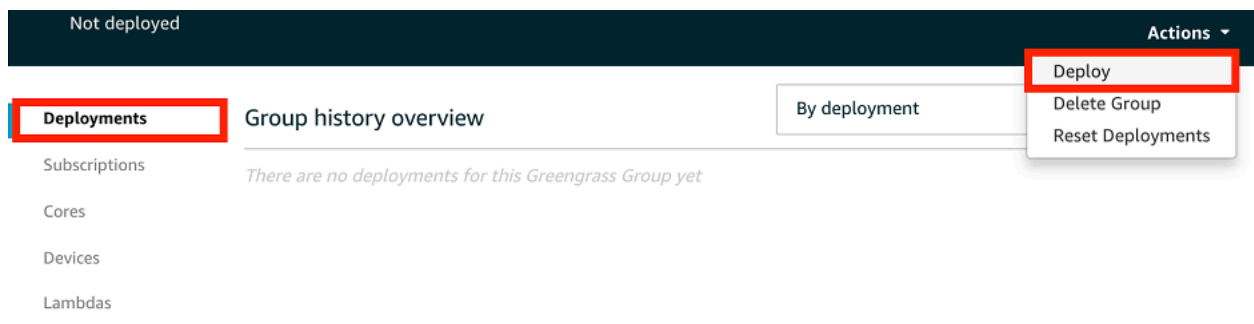
Note

路徑中的版本取決於核心裝置上安裝的核心 AWS IoT Greengrass 軟體版本。

- b. 若要啟動協助程式：

```
cd /greengrass/ggc/core/  
sudo ./greengrassd start
```

2. 在群組組態頁面上，選擇部署。



3. 在 Lambda 函數索引標籤的系統 Lambda 函數區段下，選取 IP 偵測器，然後選擇編輯。

4. 在編輯 IP 偵測器設定對話方塊中，選取自動偵測和覆寫 MQTT 代理程式端點。
5. 選擇儲存。

這可讓裝置自動取得核心的連接資訊，例如 IP 位址、DNS、連接埠編號。建議自動偵測，但 AWS IoT Greengrass 也支援手動指定的端點。只會在第一次部署群組時收到復原方法的提示。

Note

如果出現提示，請授予許可來建立 [Greengrass 服務角色](#)，並將其與目前 AWS 帳戶中的建立關聯 AWS 區域。此角色允許 AWS IoT Greengrass 存取 AWS 服務中的資源。

此部署頁面會顯示部署時間戳記、版本 ID 和狀態。完成後，部署顯示的狀態應為已完成。

如需有關部署的詳細資訊，請參閱 [部署 AWS IoT Greengrass 群組](#)。如需故障診斷協助，請參閱 [疑難排解](#)。

步驟 9：測試推論應用程式

現在您可以驗證部署是否已正確設定。若要測試，您可以訂閱hello/world主題，並檢視 Lambda 函數發佈的預測結果。

Note

如果 Raspberry Pi 附加監控功能，即時攝影機反饋會顯示在預覽視窗中。

1. 在 AWS IoT 主控台的測試下，選擇 MQTT 測試用戶端。
2. 於訂閱使用以下值：
 - 對於訂閱主題，請使用 hello/world。
 - 在其他組態下，針對 MQTT 承載顯示，選擇將承載顯示為字串。
3. 選擇 Subscribe (訂閱)。

如果測試成功，來自 Lambda 函數的訊息會出現在頁面底部。每個訊息含有搜尋映像的前 5 個預測結果，使用格式：成功率、預估類別 ID，和對應的類別名稱。

Subscribe to a topic

Publish to a topic

hello/world

Publish

Specify a topic and a message to publish with a QoS of 0.

hello/world

```
1 {
2   "message": "Hello from AWS IoT console"
3 }
```

Publish to topic

hello/world	Mar 30, 2018 1:47:07 PM -0700	Export Hide
New Prediction: [(0.31046376, 'n03637318 lampshade, lamp shade'), (0.11445289, 'n04380533 table lamp'), (0.04436367, 'n04254120 soap dispenser'), (0.035816364, 'n04286575 spotlight, spot'), (0.028093718, 'n03201208 dining table, board')]		
hello/world Mar 30, 2018 1:47:01 PM -0700 Export Hide		
New Prediction: [(0.16117829, 'n03876231 paintbrush'), (0.13750333, 'n04442312 toaster'), (0.081819646, 'n03924679 photocopier'), (0.068144165, 'n02783161 ballpoint, ballpoint pen, ballpen, Biro'), (0.044701375, 'n04209239 shower curtain')]		
hello/world Mar 30, 2018 1:46:55 PM -0700 Export Hide		
New Prediction: [(0.46284258, 'n04442312 toaster'), (0.16061385, 'n03908618 pencil box, pencil case'), (0.043834824, 'n03291819 envelope'), (0.027529096, 'n03908714 pencil sharpener'), (0.027273422, 'n04209239 shower curtain')]		

針對 AWS IoT Greengrass ML 推論進行故障診斷

如果測試不成功，您可以嘗試以下故障診斷的步驟。請在您的 Raspberry Pi 終端機執行此命令。

檢查錯誤日誌

1. 切換到根使用者和導覽至 log 目錄。存取 AWS IoT Greengrass 日誌需要根許可。

```
sudo su
cd /greengrass/ggc/var/log
```

2. 在 system 目錄中，勾選 runtime.log 或 python_runtime.log。

在 user/*region/account-id* 目錄中，勾選 greengrassObjectClassification.log。

如需詳細資訊，請參閱[the section called “日誌故障診斷”](#)。

Runtime.log 中的「解壓縮」錯誤

若 runtime.log 包含類似以下的錯誤，請確認您的 tar.gz 來源模型套件有一個父目錄。

```
Greengrass deployment error: unable to download the artifact model-arn: Error while
processing.
Error while unpacking the file from /tmp/greengrass/artifacts/model-arn/path to /
greengrass/ggc/deployment/path/model-arn,
```

```
error: open /greengrass/ggc/deployment/path/model-arn/squeezenet/  
squeezenet_v1.1-0000.params: no such file or directory
```

如果您的套件沒有含有模型檔案的主目錄，請使用以下命令重新封裝模型：

```
tar -zcvf model.tar.gz ./model
```

例如：

```
#$ tar -zcvf test.tar.gz ./test  
./test  
./test/some.file  
./test/some.file2  
./test/some.file3
```

Note

此命令結尾請勿含有 /*。

確認 Lambda 函數已成功部署

1. 在 /lambda 目錄中列出已部署 Lambda 的內容。先執行命令，再取代預留位置值。

```
cd /greengrass/ggc/deployment/lambda/  
arn:aws:lambda:region:account:function:function-name:function-version  
ls -la
```

2. 驗證目錄是否包含與您在 中上傳的greengrassObjectClassification.zip部署套件相同的內容[步驟 4：建立和發佈 Lambda 函數](#)。

確保 .py 檔案和相依性皆位於目錄的根中。

驗證推論模型已成功部署

1. 尋找 Lambda 執行時間程序的程序識別號碼 (PID)：

```
ps aux | grep 'lambda-function-name'
```

在輸出中，PID 會出現在 Lambda 執行時間程序行的第二欄。

- 輸入 Lambda 執行時間命名空間。執行命令前，務必先取代預留位置 *pid* 值。

Note

此目錄及其內容位於 Lambda 執行時間命名空間中，因此不會在一般 Linux 命名空間中顯示。

```
sudo nsenter -t pid -m /bin/bash
```

- 列出您為機器學習資源所指定的本機目錄內容。

```
cd /greengrass-machine-learning/mxnet/squeezenet/  
ls -ls
```

您會看到以下檔案：

```
32 -rw-r--r-- 1 ggc_user ggc_group 31675 Nov 18 15:19 synset.txt  
32 -rw-r--r-- 1 ggc_user ggc_group 28707 Nov 18 15:19 squeezenet_v1.1-symbol.json  
4832 -rw-r--r-- 1 ggc_user ggc_group 4945062 Nov 18 15:19  
squeezenet_v1.1-0000.params
```

後續步驟

接下來，探索其他推論應用程式。AWS IoT Greengrass 提供可用於嘗試本機推論的其他 Lambda 函數。您可以在 [the section called “安裝 MXNet 架構”](#) 的預先編譯程式庫資料中夾找到範例套件。

設定 Intel Atom

若要在 Intel Atom 裝置上執行此教學課程，您必須提供來源映像、設定 Lambda 函數，以及新增另一個本機裝置資源。若要使用 GPU 進行推論，請確定裝置上已安裝下列軟體：

- OpenCL 1.0 版或更高版本
- Python 3.7 和 pip

Note

如果您的裝置是用 Python 3.6 預先建置的，則可以改為建立 Python 3.7 的符號連結。如需詳細資訊，請參閱[Step 2](#)。

- [NumPy](#)
- [OpenCV on Wheels](#)

1. 下載 Lambda 函數的靜態 PNG 或 JPG 影像，以用於影像分類。此範例最適合與小型映像檔案一同運作。

儲存含有 `greengrassObjectClassification.py` 檔案 (或於該目錄的子目錄中) 目錄中的映像檔案。這是您在 中上傳的 Lambda 函數部署套件。 [the section called “建立和發佈 Lambda 函數”](#)

Note

如果您使用的是 AWS DeepLens，則可以使用內建攝影機或掛載自己的攝影機，在擷取的影像上執行推論，而不是靜態影像。不過，強烈建議您先從靜態影像開始。
如果您使用攝影機，請確定 `awscam` APT 套件已安裝並保持最新狀態。如需詳細資訊，請參閱《AWS DeepLens 開發人員指南》中的[更新 AWS DeepLens 裝置](#)。

2. 如果您不是使用 Python 3.7，請務必建立從 Python 3.x 到 Python 3.7 的符號連結。這會將您的裝置設定為搭配 Python 3 使用 AWS IoT Greengrass。執行以下命令來找到您的 Python 安裝：

```
which python3
```

執行以下命令來建立符號連結：

```
sudo ln -s path-to-python-3.x/python3.x path-to-python-3.7/python3.7
```

重新啟動裝置。

3. 編輯 Lambda 函數的組態。請遵循 [the section called “將 Lambda 函數新增至群組”](#) 中的程序。

Note

建議您在沒有容器化的情況下執行 Lambda 函數，除非您的業務案例需要。這有助於啟用裝置 GPU 和攝影機的存取，而無需設定裝置資源。如果您在沒有容器化的情況下執行，您還必須授予 AWS IoT Greengrass Lambda 函數的根存取權。

a. 若要在沒有容器化的情況下執行：

- 針對系統使用者和群組，選擇 **Another user ID/group ID**。針對系統使用者 ID，輸入 **0**。針對系統群組 ID，輸入 **0**。

這可讓您的 Lambda 函數以根執行。如需以根身分執行的詳細資訊，請參閱 [the section called “為群組中的 Lambda 函數設定預設存取身分”](#)。

Tip

您也必須更新 config.json 檔案，以授予 Lambda 函數的根存取權。如需程序，請參閱 [the section called “執行 Lambda 函數做為根”](#)。

- 針對 Lambda 函數容器化，選擇無容器。

如需在沒有容器化的情況下執行的詳細資訊，請參閱 [the section called “選擇 Lambda 函數容器化時的考量事項”](#)。

- 將 Timeout (逾時) 值更新為 5 秒鐘。這可確保請求不會太早逾時。設定完成後，執行推論需要幾分鐘的時間。
- 在固定下，選擇 True。
- 在其他參數下，針對 /sys 目錄的讀取存取權，選擇已啟用。
- 針對 Lambda lifecycle (Lambda 生命週期)，選擇 Make this function long-lived and keep it running indefinitely (將此函數設定為長時間存留且無限期持續執行)。

b. 改為在容器化模式下執行：**Note**

除非您的商業案例需要，否則我們不建議在容器化模式下執行。

- 將 Timeout (逾時) 值更新為 5 秒鐘。這可確保請求不會太早逾時。設定完成後，執行推論需要幾分鐘的時間。
 - 針對固定，選擇 True。
 - 在其他參數下，針對 /sys 目錄的讀取存取權，選擇已啟用。
4. 如果以容器化模式執行，請新增必要的本機裝置資源，以授予裝置 GPU 的存取權。

 Note

如果您在非容器化模式下執行，AWS IoT Greengrass 可以存取裝置 GPU，而無需設定裝置資源。

- a. 在群組組態頁面上，選擇資源索引標籤。
- b. 選擇新增本機資源。
- c. 定義資源：
 - 針對資源名稱，輸入 **renderD128**。
 - 針對資源類型，選擇本機裝置。
 - 針對 Device path (裝置路徑)，輸入 **/dev/dri/renderD128**。
 - 針對系統群組擁有者和檔案存取許可，選擇自動新增擁有資源之系統群組的檔案系統許可。
 - 對於 Lambda 函數關聯，授予 Lambda 函數的讀取和寫入存取權。

設定 NVIDIA Jetson TX2

若要在 NVIDIA Jetson TX2 上執行此教學課程，請提供來源映像並設定 Lambda 函數。如果您使用的是 GPU，請您還必須新增本機裝置。

1. 請確定您的 Jetson 裝置已設定好，以便您可以安裝 AWS IoT Greengrass 核心軟體。如需如何設定裝置的詳細資訊，請參閱[the section called “設定其他裝置”](#)。
2. 開啟 MXNet 文件、移至[在 Jetson 上安裝 MXNet](#)，然後依照指示在 Jetson 裝置上安裝 MXNet。

Note

如果您想從來源建置 MXNet，請按照指示建置共用程式庫。編輯 config.mk 檔案中的下列設定，以便與 Jetson TX2 裝置搭配使用：

- 新增 `-gencode arch=compute-62, code=sm_62` 至 `CUDA_ARCH` 設定。
- 開啟 CUDA。

```
USE_CUDA = 1
```

3. 下載 Lambda 函數的靜態 PNG 或 JPG 影像，以用於影像分類。該應用程式最適合與小型映像檔案一同運作。或者，您可以在 Jetson 電路板檢測攝影機來擷取來源映像。

在含有 `greengrassObjectClassification.py` 檔案的目錄中儲存您的映像檔案。您也可以將它們儲存在此目錄的子目錄中。此目錄位於您在 中上傳的 Lambda 函數部署套件中 [the section called “建立和發佈 Lambda 函數”](#)。

4. 建立從 Python 3.7 到 Python 3.6 的符號連結，以搭配 Python 3 使用 AWS IoT Greengrass。執行以下命令來找到您的 Python 安裝：

```
which python3
```

執行以下命令來建立符號連結：

```
sudo ln -s path-to-python-3.6/python3.6 path-to-python-3.7/python3.7
```

重新啟動裝置。

5. 確定 `ggc_user` 系統帳戶可以使用 MXNet 架構：

```
"sudo -u ggc_user bash -c 'python3 -c "import mxnet"'
```

6. 編輯 Lambda 函數的組態。請遵循 [the section called “將 Lambda 函數新增至群組”](#) 中的程序。

Note

建議您在沒有容器化的情況下執行 Lambda 函數，除非您的業務案例需要。這有助於啟用裝置 GPU 和攝影機的存取，而無需設定裝置資源。如果您在沒有容器化的情況下執行，您還必須授予 AWS IoT Greengrass Lambda 函數的根存取權。

a. 若要在沒有容器化的情況下執行：

- 針對系統使用者和群組，選擇 **Another user ID/group ID**。針對系統使用者 ID，輸入 **0**。針對系統群組 ID，輸入 **0**。

這可讓您的 Lambda 函數以根執行。如需以根身分執行的詳細資訊，請參閱 [the section called “為群組中的 Lambda 函數設定預設存取身分”](#)。

Tip

您也必須更新 config.json 檔案，以授予 Lambda 函數的根存取權。如需此程序，請參閱 [the section called “執行 Lambda 函數做為根”](#)。

- 針對 Lambda 函數容器化，選擇無容器。

如需在沒有容器化的情況下執行 的詳細資訊，請參閱 [the section called “選擇 Lambda 函數容器化時的考量事項”](#)。

- 在其他參數下，針對 /sys 目錄的讀取存取權，選擇已啟用。
- 在環境變數下，將下列鍵值對新增至 Lambda 函數。這 AWS IoT Greengrass 會將 設定為使用 MXNet 架構。

金鑰	值
PATH	/usr/local/cuda/bin:\$PATH
MXNET_HOME	\$HOME/mxnet/
PYTHONPATH	\$MXNET_HOME/python:\$PYTHONPATH
CUDA_HOME	/usr/local/cuda

金鑰	值
LD_LIBRARY_PATH	\$LD_LIBRARY_PATH:\${CUDA_HOME}/lib64

b. 改為在容器化模式下執行：

Note

除非您的商業案例需要，否則我們不建議在容器化模式下執行。

- 增加 Memory limit (記憶體限制) 值。請使用 500 MB 的 CPU 或至少 2000 MB 的 GPU。
- 在其他參數下，針對 /sys 目錄的讀取存取權，選擇已啟用。
- 在環境變數下，將下列鍵值對新增至 Lambda 函數。這 AWS IoT Greengrass 會將設定為使用 MXNet 架構。

金鑰	值
PATH	/usr/local/cuda/bin:\$PATH
MXNET_HOME	\$HOME/mxnet/
PYTHONPATH	\$MXNET_HOME/python:\$PYTHONPATH
CUDA_HOME	/usr/local/cuda
LD_LIBRARY_PATH	\$LD_LIBRARY_PATH:\${CUDA_HOME}/lib64

7. 如果在容器化模式下執行，請新增下列本機裝置資源，以授予裝置 GPU 的存取權。請遵循 [the section called “新增資源到群組”](#) 中的程序。

Note

如果您在非容器化模式下執行，AWS IoT Greengrass 可以存取裝置 GPU，而無需設定裝置資源。

對於每個資源：

- 於資源類型選擇裝置。
- 針對系統群組擁有者和檔案存取許可，選擇自動新增擁有資源之系統群組的檔案系統許可。

名稱	裝置路徑
nvhost-ctrl	/dev/nvhost-ctrl
nvhost-gpu	/dev/nvhost-gpu
nvhost-ctrl-gpu	/dev/nvhost-ctrl-gpu
nvhost-dbg-gpu	/dev/nvhost-dbg-gpu
nvhost-prof-gpu	/dev/nvhost-prof-gpu
nvmap	/dev/nvmap
nvhost-vic	/dev/nvhost-vic
tegra_dc_ctrl	/dev/tegra_dc_ctrl

8. 如果以容器化模式執行，請新增下列本機磁碟區資源，以授予裝置攝影機的存取權。請遵循 [the section called “新增資源到群組”](#) 中的程序。

 Note

如果您在非容器化模式下執行，AWS IoT Greengrass 可以存取您的裝置攝影機，而無需設定磁碟區資源。

- 針對 Resource type (資源類型)，選擇 Volume (磁碟區)。
- 針對系統群組擁有者和檔案存取許可，選擇自動新增擁有資源之系統群組的檔案系統許可。

名稱	來源路徑	目的地路徑
shm	/dev/shm	/dev/shm
tmp	/tmp	/tmp

如何使用 設定最佳化機器學習推論 AWS 管理主控台

若要遵循本教學課程中的步驟，您必須使用 AWS IoT Greengrass Core v1.10 或更新版本。

您可以使用 SageMaker AI Neo 深度學習編譯器，在 Tensorflow、Apache MXNet、PyTorch、ONNX 和 XGBoost 架構中最佳化原生機器學習推論模型的預測效率，以獲得更小的足跡和更快的效能。然後，您可以下載最佳化模型並安裝 SageMaker AI Neo 深度學習執行期，然後將它們部署到 AWS IoT Greengrass 您的裝置，以加快推論速度。

本教學課程說明如何使用 AWS 管理主控台 來設定 Greengrass 群組，以執行 Lambda 推論範例，在本機辨識攝影機的影像，而無需將資料傳送至雲端。推論範例會存取 Raspberry Pi 上的相機模組。在本教學課程中，您會下載由 Resnet-50 訓練並在 Neo 深度學習編譯器中最佳化的預先封裝模型。然後，您可以使用模型在 AWS IoT Greengrass 裝置上執行本機映像分類。

本教學課程所述以下高階執行步驟：

1. [設定 Raspberry Pi](#)
2. [安裝 Neo 深度學習執行時間](#)
3. [建立推論 Lambda 函數](#)
4. [將 Lambda 函數新增至群組](#)
5. [將 Neo 最佳化模型資源新增至群組](#)
6. [將您的相機裝置資源新增至群組](#)
7. [新增訂閱到群組](#)
8. [部署群組](#)
9. [測試範例](#)

先決條件

為完成此教學課程您需要：

- Raspberry Pi 4 模型 B 或 Raspberry Pi 3 模型 B/B+，設定與 搭配使用 AWS IoT Greengrass。若要使用 設定 Raspberry Pi AWS IoT Greengrass，請執行 [Greengrass 裝置設定](#) 指令碼，或確認您已完成的 [單元 1](#) 和 [單元 2 入門 AWS IoT Greengrass](#)。

 Note

Raspberry Pi 可能需要 2.5A [電源供應器](#) 來執行通常用於影像分類的深度學習架構。額定值較低的電源供應器可能會導致裝置重新啟動。

- [Raspberry Pi 攝影機模組 V2 - 8 百萬像素、1080p](#)。若要了解如何設定相機，請參閱 Raspberry Pi 文件中的 [連接相機](#)。
- Greengrass 群組和 Greengrass 核心。若要了解如何建立 Greengrass 群組或是 Greengrass 核心，請參閱 [入門 AWS IoT Greengrass](#)。

 Note

本教學課程使用 Raspberry Pi，但 AWS IoT Greengrass 支援其他平台，例如 [Intel Atom](#) 和 [NVIDIA Jetson TX2](#)。如果使用 Intel Atom 範例，您可能需要安裝 Python 3.6 而不是 Python 3.7。如需設定裝置以便安裝 AWS IoT Greengrass 核心軟體的詳細資訊，請參閱 [the section called “設定其他裝置”](#)。

對於 AWS IoT Greengrass 不支援的第三方平台，您必須在非容器化模式下執行 Lambda 函數。若要在非容器化模式下執行，您必須以根身分執行 Lambda 函數。如需詳細資訊，請參閱 [the section called “選擇 Lambda 函數容器化時的考量事項”](#) 及 [the section called “為群組中的 Lambda 函數設定預設存取身分”](#)。

步驟一：設定 Raspberry Pi

在此步驟中，請安裝 Raspbian 作業系統的更新、安裝相機模組軟體和 Python 相依性，以及啟用相機界面。

在您的 Raspberry Pi 終端機執行以下命令。

1. 安裝 Raspbian 的更新。

```
sudo apt-get update
sudo apt-get dist-upgrade
```

2. 安裝攝影機模組的 picamera 界面和本單元其他所需的 Python 程式庫。

```
sudo apt-get install -y python3-dev python3-setuptools python3-pip python3-picamera
```

驗證安裝：

- 確保您的 Python 3.7 安裝包含 pip。

```
python3 -m pip
```

如果未安裝 pip，請從 [pip 網站](#) 下載它，然後執行以下命令。

```
python3 get-pip.py
```

- 請確保您的 Python 版本是 3.7 或更高版本。

```
python3 --version
```

如果輸出中列出較早的版本，請執行下列命令。

```
sudo apt-get install -y python3.7-dev
```

- 請確定 Setuptools 和 Picamera 已安裝成功。

```
sudo -u ggc_user bash -c 'python3 -c "import setuptools"'  
sudo -u ggc_user bash -c 'python3 -c "import picamera"'
```

如果輸出中未包含錯誤，則表示驗證成功。

Note

如果裝置上安裝的 Python 可執行檔是 python3.7，請針對本教學課程中的命令使用 python3.7 而不是 python3。確保您的 pip 安裝對應到正確的 python3.7 或 python3 版本，以避免相依性錯誤。

3. 重新啟動 Raspberry Pi。

```
sudo reboot
```


4. 請開啟 Raspberry Pi 組態工具。

```
sudo raspi-config
```

5. 請使用箭頭鍵開啟 Interfacing Options (連接選項) 並啟用攝影機界面。如果出現提示，請允許重新啟動裝置。

6. 請使用以下命令測試攝影機建立。

```
raspistill -v -o test.jpg
```

這會在 Raspberry Pi 上開啟預覽視窗、將名為 `test.jpg` 的圖片儲存至現行目錄，並在 Raspberry Pi 終端機中顯示相機的相關資訊。

步驟 2：安裝 Amazon SageMaker Neo 深度學習執行時間

在此步驟中，在 Raspberry Pi 上安裝 Neo 深度學習執行期 (DLR)。

Note

針對本教學課程，建議您安裝 1.1.0 版。

1. 遠端登入到 Raspberry Pi。

```
ssh pi@your-device-ip-address
```

2. 開啟 DLR 文件、開啟[安裝 DLR](#)，然後找到 Raspberry Pi 裝置的 wheel URL。然後，依照指示在裝置上安裝 DLR。例如，你可以使用 pip：

```
pip3 install rasp3b-wheel-url
```

3. 安裝 DLR 之後，請驗證下列組態：

- 確定 `ggc_user` 系統帳戶可以使用 DLR 程式庫。

```
sudo -u ggc_user bash -c 'python3 -c "import dlr"'
```

- 確保已安裝 NumPy。

```
sudo -u ggc_user bash -c 'python3 -c "import numpy"'
```

步驟 3：建立推論 Lambda 函數

在此步驟中，建立 Lambda 函數部署套件和 Lambda 函數。然後，發佈函數版本和建立別名。

1. 在您的電腦上，從 [the section called “機器學習範例”](#) 下載 Raspberry Pi 的 DLR 範例。
2. 解壓縮所下載的 `dlr-py3-armv7l.tar.gz` 檔案。

```
cd path-to-downloaded-sample  
tar -xvzf dlr-py3-armv7l.tar.gz
```

解壓縮的範例套件中的 `examples` 目錄內包含函數程式碼和相依性。

- `inference.py` 是本教學課程中使用的推論程式碼。您可以使用此程式碼作為範本來建立您自己的推論函數。
- `greengrasssdk` 是適用於 Python 的核心 AWS IoT Greengrass SDK 的 1.5.0 版。

Note

如果有新版本可用，您可以下載並升級部署套件中的 SDK 版本。如需詳細資訊，請參閱 GitHub 上的 [AWS IoT Greengrass 適用於 Python 的核心 SDK](#)。

3. 將 `examples` 目錄的內容壓縮成名為 `optimizedImageClassification.zip` 的檔案。這是您的部署套件。

```
cd path-to-downloaded-sample/dlr-py3-armv7l/examples  
zip -r optimizedImageClassification.zip .
```

部署套件包含函數程式碼和相依性。這包括叫用 Neo 深度學習執行時間 Python APIs 以使用 Neo 深度學習編譯器模型執行推論的程式碼。

Note

確保 `.py` 檔案和相依性皆位於目錄的根中。

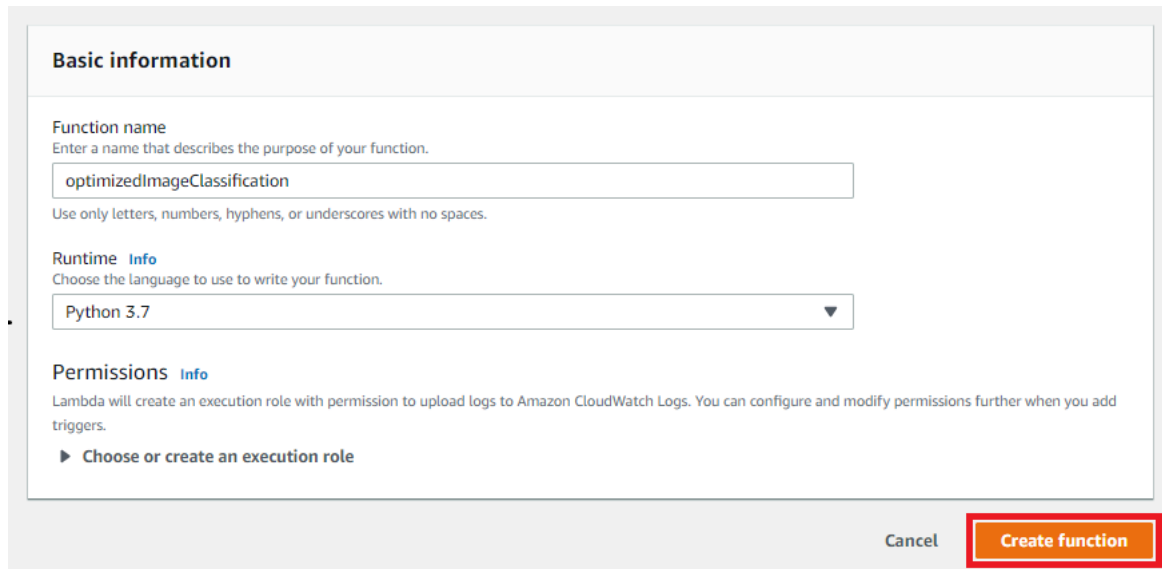
4. 現在，將 Lambda 函數新增至 Greengrass 群組。

從 Lambda 主控台頁面，選擇函數，然後選擇建立函數。

5. 選擇從頭開始撰寫，並使用下列值來建立函數：

- 針對函數名稱，請輸入 **optimizedImageClassification**。
- 針對執行期，選擇 Python 3.7。

對於許可，請保留預設設定。這會建立授予基本 Lambda 許可的執行角色。不會使用此角色 AWS IoT Greengrass。

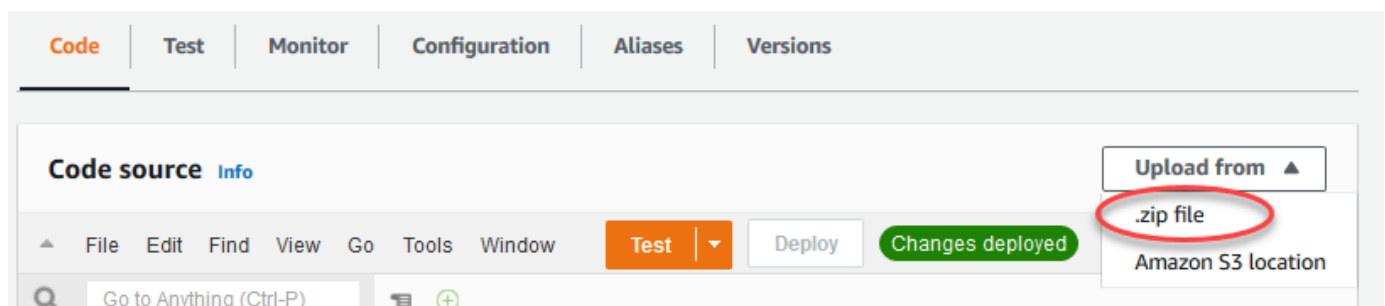


The screenshot shows the 'Basic information' tab of the AWS Lambda 'Create function' wizard. It includes a 'Function name' field with the value 'optimizedImageClassification', a 'Runtime' dropdown menu set to 'Python 3.7', and a 'Permissions' section with a link to 'Choose or create an execution role'. At the bottom right, there are 'Cancel' and 'Create function' buttons, with the latter highlighted by a red rectangle.

6. 選擇建立函數。

現在，上傳您的 Lambda 函數部署套件並註冊處理常式。

1. 在程式碼索引標籤的程式碼來源下，選擇上傳來源。從下拉式清單中，選擇 .zip 檔案。



The screenshot shows the 'Code source' tab of the AWS Lambda console. It features a toolbar with 'File', 'Edit', 'Find', 'View', 'Go', 'Tools', 'Window', 'Test', 'Deploy', and 'Changes deployed' buttons. A dropdown menu is open next to the 'Upload from' button, showing options for '.zip file' (circled in red) and 'Amazon S3 location'.

2. 選擇您的 optimizedImageClassification.zip 部署套件，然後選擇儲存。

3. 在函數的程式碼索引標籤的執行時間設定下，選擇編輯，然後輸入下列值。

- 針對執行期，選擇 Python 3.7。
- 對於 Handler (處理常式)，輸入 **inference.handler**。

選擇儲存。

接著，發佈 Lambda 函數的第一個版本。然後，建立[版本的別名](#)。

Note

Greengrass 群組可以依別名（建議）或版本參考 Lambda 函數。使用別名可讓您更輕鬆地管理程式碼更新，因為您不必在更新函數程式碼時變更訂閱資料表或群組定義。反之，您只需將別名指向新的函數版本。

1. 請從操作功能表中選擇發行新版本。

2. 針對 Version description (版本描述)，輸入 **First version**，然後選擇 Publish (發佈)。

3. 在 optimizedImageClassification: 1 組態頁面上，從 Actions (動作) 功能表中選擇 Create alias (建立別名)。

4. 在建立警示頁面上使用下列值：

- 對於名稱，輸入 **mlTestOpt**。
- 針對 Version (版本) 輸入 **1**。

 Note

AWS IoT Greengrass 不支援 \$LATEST 版本的 Lambda 別名。

5. 選擇建立。

現在，將 Lambda 函數新增至 Greengrass 群組。

步驟 4：將 Lambda 函數新增至 Greengrass 群組

在此步驟中，將 Lambda 函數新增至 群組，然後設定其生命週期。

首先，將 Lambda 函數新增至 Greengrass 群組。

1. 在 AWS IoT 主控台導覽窗格的管理下，展開 Greengrass 裝置，然後選擇群組 (V1)。
2. 在群組組態頁面上，選擇 Lambda 函數索引標籤，然後選擇新增。
3. 選擇 Lambda 函數，然後選取 `optimizedImageClassification`。
4. 在 Lambda 函數版本上，選擇您發佈的版本別名。

接著，設定 Lambda 函數的生命週期。

1. 在 Lambda 函數組態區段中，進行下列更新。

 Note

建議您在沒有容器化的情況下執行 Lambda 函數，除非您的業務案例需要。這有助於啟用裝置 GPU 和攝影機的存取，而無需設定裝置資源。如果您在沒有容器化的情況下執行，您還必須授予 AWS IoT Greengrass Lambda 函數的根存取權。

- a. 若要在沒有容器化的情況下執行：

- 針對系統使用者和群組，選擇 **Another user ID/group ID**。針對系統使用者 ID，輸入 **0**。針對系統群組 ID，輸入 **0**。

這可讓您的 Lambda 函數以根執行。如需以根身分執行的詳細資訊，請參閱 [the section called “為群組中的 Lambda 函數設定預設存取身分”](#)。

 Tip

您也必須更新 `config.json` 檔案，以授予 Lambda 函數的根存取權。如需此程序，請參閱 [the section called “執行 Lambda 函數做為根”](#)。

- 針對 Lambda 函數容器化，選擇無容器。

如需在沒有容器化的情況下執行 的詳細資訊，請參閱 [the section called “選擇 Lambda 函數容器化時的考量事項”](#)。

- 針對 Timeout (逾時)，輸入 **10 seconds**。
- 針對固定，選擇 True。

如需詳細資訊，請參閱[the section called “生命週期組態”](#)。

- 在其他參數下，針對 `/sys` 目錄的讀取存取權，選擇已啟用。

b. 改為在容器化模式下執行：

 Note

除非您的商業案例需要，否則我們不建議在容器化模式下執行。

- 針對系統使用者和群組，選擇使用群組預設值。
- 針對 Lambda 函數容器化，選擇使用群組預設值。
- 針對 Memory limit (記憶體限制)，輸入 **1024 MB**。
- 針對 Timeout (逾時)，輸入 **10 seconds**。
- 針對固定，選擇 True。

如需詳細資訊，請參閱[the section called “生命週期組態”](#)。

- 在其他參數下，針對 `/sys` 目錄的讀取存取權，選擇已啟用。

2. 選擇新增 Lambda 函數。

將 Lambda 函數新增至群組

步驟 5：將 SageMaker AI Neo 最佳化模型資源新增至 Greengrass 群組

在此步驟中，建立最佳化 ML 推論模型的資源，並將其上傳至 Amazon S3 儲存貯體。然後，在 AWS IoT Greengrass 主控台中找到 Amazon S3 上傳的模型，並將新建立的資源與 Lambda 函數建立關聯。這使得函數可在核心裝置上存取其資源。

1. 在您的電腦上，瀏覽至您在 [the section called “建立推論 Lambda 函數”](#) 中解壓縮的範例套件內的 `resnet50` 目錄。

Note

如果使用 NVIDIA Jetson 範例，則需要改用範例套件中的 `resnet18` 目錄。如需詳細資訊，請參閱 [the section called “設定 NVIDIA Jetson TX2”](#)。

```
cd path-to-downloaded-sample/dlr-py3-armv7l/models/resnet50
```

此目錄中包含預先編譯的模型成品，其適用於利用 Resnet-50 訓練的映像分類模型。

2. 將 `resnet50` 目錄內的檔案壓縮成名為 `resnet50.zip` 的檔案。

```
zip -r resnet50.zip .
```

3. 在群組的 AWS IoT Greengrass 群組組態頁面上，選擇資源索引標籤。導覽至 Machine Learning (機器學習) 區段，然後選擇 Add machine learning resource (新增機器學習資源)。在 Create a machine learning resource (建立機器學習資源) 頁面上，於 Resource name (資源名稱) 輸入 **resnet50_model**。
4. 針對模型來源，選擇使用存放在 S3 中的模型，例如透過深度學習編譯器最佳化的模型。
5. 在 S3 URI 下，選擇瀏覽 S3。

Note

目前，最佳化的 SageMaker AI 模型會自動儲存在 Amazon S3 中。您可以使用此選項在 Amazon S3 儲存貯體中找到最佳化的模型。如需 SageMaker AI 中模型最佳化的詳細資訊，請參閱 [SageMaker AI Neo 文件](#)。

6. 選擇 Upload a model (上傳模型)。

7. 在 Amazon S3 主控台索引標籤上，將您的 zip 檔案上傳至 Amazon S3 儲存貯體。如需詳細資訊，請參閱《Amazon Simple Storage Service 使用者指南》中的[如何將檔案和資料夾上傳至 S3 儲存貯體？](#)。

 Note

您的儲存貯體名稱必須包含字串 **greengrass**。選擇唯一名稱 (例如 **greengrass-dlr-bucket-*user-id-epoch-time***)。請勿在儲存貯體名稱中使用句號 (.)。

8. 在 AWS IoT Greengrass 主控台索引標籤中，尋找並選擇您的 Amazon S3 儲存貯體。尋找您已上傳的 `resnet50.zip` 檔案，然後選擇 Select (選取)。您可能需要重新整理頁面來更新可用儲存貯體和檔案的清單。
9. 在目的地路徑中，輸入 `/ml_model`。

Local path

`/ml_model`

這是 Lambda 執行時間命名空間中本機模型的目的地。部署群組時，會 AWS IoT Greengrass 擷取來源模型套件，然後將內容擷取至指定的目錄。

 Note

我們強烈建議您使用針對本機路徑提供的確切路徑。在此步驟中使用不同的本機模型目的地路徑，會導致此教學課程中提供的一些故障診斷命令不精確。如果使用不同的路徑，請設定 `MODEL_PATH` 環境變數，其必須使用這裡提供的確切路徑。如需環境變數的相關資訊，請參閱 [AWS Lambda 環境變數](#)。

10. 如果以容器化模式執行：
 - a. 在系統群組擁有者和檔案存取許可下，選擇指定系統群組和許可。
 - b. 選擇唯讀存取，然後選擇新增資源。

步驟 6：將您的相機裝置資源新增至 Greengrass 群組

在此步驟中，為攝影機模組建立資源，並將其與 Lambda 函數建立關聯。這可讓 Lambda 函數存取核心裝置上的資源。

Note

如果您在非容器化模式下執行，AWS IoT Greengrass 可以存取您的裝置 GPU 和攝影機，而無需設定此裝置資源。

1. 在群組組態頁面上，選擇資源索引標籤。
2. 在本機資源索引標籤上，選擇新增本機資源。
3. 在新增本機資源頁面上，使用下列值：

- 針對資源名稱，輸入 **videoCoreSharedMemory**。
- 於資源類型選擇裝置。
- 針對本機裝置路徑，輸入 **/dev/vcsm**。

裝置路徑為裝置資源的本機絕對路徑。此路徑只能參考 /dev 底下的字元裝置或區塊型儲存設備。

- 針對系統群組擁有者和檔案存取許可，選擇自動新增擁有資源之系統群組的檔案系統許可。

Group owner file access permission (群組擁有者檔案存取許可) 選項讓您授與其他檔案可存取 Lambda 的許可。如需詳細資訊，請參閱[群組擁有者檔案存取許可](#)。

4. 在頁面底部，選擇新增資源。
5. 從資源索引標籤中，選擇新增並使用下列值來建立另一個本機資源：

- 針對資源名稱，輸入 **videoCoreInterface**。
- 於資源類型選擇裝置。
- 針對本機裝置路徑，輸入 **/dev/vchiq**。
- 針對系統群組擁有者和檔案存取許可，選擇自動新增擁有資源之系統群組的檔案系統許可。

6. 選擇 Add resource (新增資源)。

步驟 7：將訂閱新增到 Greengrass 群組

在此步驟中，將訂閱新增至群組。這些訂閱可讓 Lambda 函數 AWS IoT 透過發佈至 MQTT 主題，將預測結果傳送至。

1. 在群組組態頁面上，選擇訂閱索引標籤，然後選擇新增訂閱。
2. 在建立訂閱頁面上，設定來源和目標，如下所示：
 - a. 在來源類型中，選擇 Lambda 函數，然後選擇 `optimizedImageClassification`。
 - b. 在目標類型中，選擇服務，然後選擇 IoT 雲端。
 - c. 在主題篩選條件中，輸入 `/resnet-50/predictions`，然後選擇建立訂閱。
3. 新增第二個訂閱。選擇訂閱索引標籤，選擇新增訂閱，並設定來源和目標，如下所示：
 - a. 在來源類型中，選擇服務，然後選擇 IoT Cloud。
 - b. 在目標類型中，選擇 Lambda 函數，然後選擇 `optimizedImageClassification`。
 - c. 在主題篩選條件中，輸入 `/resnet-50/test`，然後選擇建立訂閱。

步驟 8：部署 Greengrass 群組

在此步驟中，將群組定義的目前版本部署到 Greengrass 核心裝置。定義包含您新增的 Lambda 函數、資源和訂閱組態。

1. 確定 AWS IoT Greengrass 核心正在執行。如果需要，請在您的 Raspberry Pi 終端機執行以下命令。
 - a. 檢查精靈是否有在運作：

```
ps aux | grep -E 'greengrass.*daemon'
```

若輸出的 root 含有 `/greengrass/ggc/packages/latest-core-version/bin/daemon` 項目，則精靈有在運作。

- b. 若要啟動協助程式：

```
cd /greengrass/ggc/core/  
sudo ./greengrassd start
```

2. 在群組組態頁面上，選擇部署。

3. 在 Lambda 函數索引標籤上，選取 IP 偵測器，然後選擇編輯。
4. 從編輯 IP 偵測器設定對話方塊中，選取自動偵測和覆寫 MQTT 代理程式端點，然後選擇儲存。

這可讓裝置自動取得核心的連接資訊，例如 IP 位址、DNS、連接埠編號。建議自動偵測，但 AWS IoT Greengrass 也支援手動指定的端點。只會在第一次部署群組時收到復原方法的提示。

Note

如果出現提示，請授予許可來建立 [Greengrass 服務角色](#)，並將其與目前 AWS 帳戶中的建立關聯 AWS 區域。此角色允許 AWS IoT Greengrass 存取 AWS 服務中的資源。

此部署頁面會顯示部署時間戳記、版本 ID 和狀態。完成後，部署顯示的狀態應為已完成。

如需有關部署的詳細資訊，請參閱 [部署 AWS IoT Greengrass 群組](#)。如需故障診斷協助，請參閱 [疑難排解](#)。

測試推論範例

現在您可以驗證部署是否已正確設定。若要測試，請訂閱 `/resnet-50/predictions` 主題，並將任何訊息發佈到 `/resnet-50/test` 主題。這會觸發 Lambda 函數使用您的 Raspberry Pi 拍攝相片，並在其擷取的影像上執行推論。

Note

如果使用 NVIDIA Jetson 範例，請務必改用 `resnet-18/predictions` 和 `resnet-18/test` 主題。

Note

如果 Raspberry Pi 附加監控功能，即時攝影機反饋會顯示在預覽視窗中。

1. 在 AWS IoT 主控台首頁的測試下，選擇 MQTT 測試用戶端。
2. 針對訂閱，選擇訂閱主題。使用下列數值。將剩餘選項保留為其預設值。
 - 針對 Subscription topic (訂閱主題)，輸入 **`/resnet-50/predictions`**。

- 在其他組態下，針對 MQTT 承載顯示，選擇將承載顯示為字串。
3. 選擇 Subscribe (訂閱)。
 4. 選擇發佈至主題，輸入 **/resnet-50/test** 做為主題名稱，然後選擇發佈。
 5. 如果測試成功，已發佈的訊息會導致 Raspberry Pi 相機擷取映像。頁面底部會顯示來自 Lambda 函數的訊息。此訊息包含映像的預測結果，其使用的格式為：預測的類別名稱、機率，以及尖峰記憶體使用量。

設定 Intel Atom

若要在 Intel Atom 裝置上執行此教學課程，您必須提供來源映像、設定 Lambda 函數，以及新增另一個本機裝置資源。若要使用 GPU 進行推論，請確定裝置上已安裝下列軟體：

- OpenCL 1.0 版或更高版本
- Python 3.7 和 pip
- [NumPy](#)
- [OpenCV on Wheels](#)

1. 下載 Lambda 函數的靜態 PNG 或 JPG 影像，以用於影像分類。此範例最適合與小型映像檔案一同運作。

儲存含有 `inference.py` 檔案 (或於該目錄的子目錄中) 目錄中的映像檔案。這是您在 中上傳的 Lambda 函數部署套件。 [the section called “建立推論 Lambda 函數”](#)

Note

如果您使用的是 AWS DeepLens，則可以使用內建攝影機或掛載自己的攝影機，在擷取的影像上執行推論，而不是靜態影像。不過，強烈建議您先從靜態影像開始。

如果您使用攝影機，請確定 awscam APT 套件已安裝並保持最新狀態。如需詳細資訊，請參閱《AWS DeepLens 開發人員指南》中的 [更新 AWS DeepLens 裝置](#)。

2. 編輯 Lambda 函數的組態。請遵循 [the section called “將 Lambda 函數新增至群組”](#) 中的程序。

Note

建議您在沒有容器化的情況下執行 Lambda 函數，除非您的業務案例需要。這有助於啟用裝置 GPU 和攝影機的存取，而無需設定裝置資源。如果您在沒有容器化的情況下執行，您還必須授予 AWS IoT Greengrass Lambda 函數的根存取權。

a. 若要在沒有容器化的情況下執行：

- 針對系統使用者和群組，選擇 **Another user ID/group ID**。針對系統使用者 ID，輸入 **0**。針對系統群組 ID，輸入 **0**。

這可讓您的 Lambda 函數以根執行。如需以根身分執行的詳細資訊，請參閱 [the section called “為群組中的 Lambda 函數設定預設存取身分”](#)。

Tip

您也必須更新 config.json 檔案，以授予 Lambda 函數的根存取權。如需此程序，請參閱 [the section called “執行 Lambda 函數做為根”](#)。

- 針對 Lambda 函數容器化，選擇無容器。

如需在沒有容器化的情況下執行的詳細資訊，請參閱 [the section called “選擇 Lambda 函數容器化時的考量事項”](#)。

- 將 Timeout (逾時) 值增加到 2 分鐘。這可確保請求不會太早逾時。設定完成後，執行推論需要幾分鐘的時間。
- 針對固定，選擇 True。
- 在其他參數下，針對 /sys 目錄的讀取存取權，選擇已啟用。

b. 改為在容器化模式下執行：**Note**

除非您的商業案例需要，否則我們不建議在容器化模式下執行。

- 將 Memory limit (記憶體限制) 值增加到 3000 MB。

- 將 Timeout (逾時) 值增加到 2 分鐘。這可確保請求不會太早逾時。設定完成後，執行推論需要幾分鐘的時間。
 - 針對固定，選擇 True。
 - 在其他參數下，針對 /sys 目錄的讀取存取權，選擇已啟用。
3. 將 Neo 最佳化模型資源新增至群組。上傳您在 [the section called “建立推論 Lambda 函數”](#) 中解壓縮的範例套件 resnet50 目錄中的模型資源。此目錄中包含預先編譯的模型成品，其適用於利用 Resnet-50 訓練的映像分類模型。遵循[the section called “將 Neo 最佳化模型資源新增至群組”](#)中的程序，但有下列更新。
- 將 resnet50 目錄內的檔案壓縮成名為 resnet50.zip 的檔案。
 - 在 Create a machine learning resource (建立機器學習資源) 頁面上，於 Resource name (資源名稱) 輸入 **resnet50_model**。
 - 上傳 resnet50.zip 檔案。
4. 如果以容器化模式執行，請新增必要的本機裝置資源，以授予裝置 GPU 的存取權。

Note

如果您在非容器化模式下執行，AWS IoT Greengrass 可以存取裝置 GPU，而無需設定裝置資源。

- a. 在群組組態頁面上，選擇資源索引標籤。
- b. 在本機資源區段中，選擇新增本機資源。
- c. 定義資源：
 - 針對資源名稱，輸入 **renderD128**。
 - 於資源類型選擇裝置。
 - 針對本機裝置路徑，輸入 **/dev/dri/renderD128**。
 - 針對系統群組擁有者和檔案存取許可，選擇自動新增擁有資源之系統群組的檔案系統許可。

設定 NVIDIA Jetson TX2

若要在 NVIDIA Jetson TX2 上執行此教學課程，請提供來源映像、設定 Lambda 函數，以及新增更多本機裝置資源。

1. 請確定您的 Jetson 裝置已設定好，以便您可以安裝 AWS IoT Greengrass 核心軟體並使用 GPU 進行推論。如需如何設定裝置的詳細資訊，請參閱[the section called “設定其他裝置”](#)。若要在 NVIDIA Jetson TX2 上使用 GPU 進行推論，當您使用 Jetpack 4.3 為電路板建立映像時，必須在裝置上安裝 CUDA 10.0 和 cuDNN 7.0。
2. 下載 Lambda 函數的靜態 PNG 或 JPG 影像，以用於影像分類。此範例最適合與小型映像檔案一同運作。

在含有 `inference.py` 檔案的目錄中儲存您的映像檔案。您也可以將它們儲存在此目錄的子目錄中。此目錄位於您在 中上傳的 Lambda 函數部署套件中[the section called “建立推論 Lambda 函數”](#)。

Note

您可以改為選擇在 Jetson 電路板檢測攝影機來擷取來源影像。不過，強烈建議您先從靜態影像開始。

3. 編輯 Lambda 函數的組態。請遵循 [the section called “將 Lambda 函數新增至群組”](#) 中的程序。

Note

建議您在沒有容器化的情況下執行 Lambda 函數，除非您的業務案例需要。這有助於啟用裝置 GPU 和攝影機的存取，而無需設定裝置資源。如果您在沒有容器化的情況下執行，您還必須授予 AWS IoT Greengrass Lambda 函數的根存取權。

a. 若要在沒有容器化的情況下執行：

- 針對執行身分，選擇 **Another user ID/group ID**。針對 UID，輸入 **0**。針對 GUID，輸入 **0**。

這可讓您的 Lambda 函數以根執行。如需以根身分執行的詳細資訊，請參閱 [the section called “為群組中的 Lambda 函數設定預設存取身分”](#)。

Tip

您也必須更新 `config.json` 檔案，以授予 Lambda 函數的根存取權。如需此程序，請參閱 [the section called “執行 Lambda 函數做為根”](#)。

- 針對 Lambda 函數容器化，選擇無容器。

如需在沒有容器化的情況下執行 的詳細資訊，請參閱 [the section called “選擇 Lambda 函數容器化時的考量事項”](#)。

- 將 Timeout (逾時) 值增加到 5 分鐘。這可確保請求不會太早逾時。設定完成後，執行推論需要幾分鐘的時間。
- 針對固定，選擇 True。
- 在其他參數下，針對 /sys 目錄的讀取存取權，選擇已啟用。

b. 改為在容器化模式下執行：

 Note

除非您的商業案例需要，否則我們不建議在容器化模式下執行。

- 增加 Memory limit (記憶體限制) 值。若要在 GPU 模式下使用提供的模型，請至少使用 2000 MB。
 - 將 Timeout (逾時) 值增加到 5 分鐘。這可確保請求不會太早逾時。設定完成後，執行推論需要幾分鐘的時間。
 - 針對固定，選擇 True。
 - 在其他參數下，針對 /sys 目錄的讀取存取權，選擇已啟用。
4. 將 Neo 最佳化模型資源新增至群組。上傳您在 [the section called “建立推論 Lambda 函數”](#) 中解壓縮的範例套件 resnet18 目錄中的模型資源。此目錄中包含預先編譯的模型成品，其適用於利用 Resnet-18 訓練的映像分類模型。遵循[the section called “將 Neo 最佳化模型資源新增至群組”](#)中的程序，但有下列更新。
- 將 resnet18 目錄內的檔案壓縮成名為 resnet18.zip 的檔案。
 - 在 Create a machine learning resource (建立機器學習資源) 頁面上，於 Resource name (資源名稱) 輸入 **resnet18_model**。
 - 上傳 resnet18.zip 檔案。
5. 如果以容器化模式執行，請新增必要的本機裝置資源，以授予裝置 GPU 的存取權。

 Note

如果您在非容器化模式下執行，AWS IoT Greengrass 可以存取裝置 GPU，而無需設定裝置資源。

- a. 在群組組態頁面上，選擇資源索引標籤。
- b. 在本機資源區段中，選擇新增本機資源。
- c. 定義每個資源：
 - 針對 Resource name (資源名稱) 和 Device path (裝置路徑)，請使用下表中的值。為表格中的每一列建立一個裝置資源。
 - 於資源類型選擇裝置。
 - 針對系統群組擁有者和檔案存取許可，選擇自動新增擁有資源之系統群組的檔案系統許可。

名稱	裝置路徑
nvhost-ctrl	/dev/nvhost-ctrl
nvhost-gpu	/dev/nvhost-gpu
nvhost-ctrl-gpu	/dev/nvhost-ctrl-gpu
nvhost-dbg-gpu	/dev/nvhost-dbg-gpu
nvhost-prof-gpu	/dev/nvhost-prof-gpu
nvmap	/dev/nvmap
nvhost-vic	/dev/nvhost-vic
tegra_dc_ctrl	/dev/tegra_dc_ctrl

6. 如果以容器化模式執行，請新增下列本機磁碟區資源，以授予裝置攝影機的存取權。請遵循 [the section called “將 Neo 最佳化模型資源新增至群組”](#) 中的程序。

 Note

如果您在非容器化模式下執行，AWS IoT Greengrass 可以存取您的裝置攝影機，而無需設定裝置資源。

- 針對 Resource type (資源類型)，選擇 Volume (磁碟區)。
- 針對系統群組擁有者和檔案存取許可，選擇自動新增擁有資源之系統群組的檔案系統許可。

名稱	來源路徑	目的地路徑
shm	/dev/shm	/dev/shm
tmp	/tmp	/tmp

- 更新您的群組訂閱以使用正確的目錄。遵循[the section called “新增訂閱到群組”](#)中的程序，但有下列更新。
 - 對於您的第一個主題篩選，請輸入 **/resnet-18/predictions**。
 - 對於您的第二個主題篩選，請輸入 **/resnet-18/test**。
- 更新您的測試訂閱以使用正確的目錄。遵循[the section called “測試範例”](#)中的程序，但有下列更新。
 - 針對訂閱，選擇訂閱主題。針對 Subscription topic (訂閱主題)，輸入 **/resnet-18/predictions**。
 - 在 **/resnet-18/predictions** 頁面上，指定要發佈至其中的 **/resnet-18/test** 主題。

針對 AWS IoT Greengrass ML 推論進行故障診斷

如果測試不成功，您可以嘗試以下故障診斷的步驟。請在您的 Raspberry Pi 終端機執行此命令。

檢查錯誤日誌

- 切換到根使用者和導覽至 log 目錄。存取 AWS IoT Greengrass 日誌需要根許可。

```
sudo su
cd /greengrass/ggc/var/log
```

- 檢查 **runtime.log** 是否有任何錯誤。

```
cat system/runtime.log | grep 'ERROR'
```

您也可以在使用者定義的 Lambda 函數日誌中查看是否有任何錯誤：

```
cat user/your-region/your-account-id/lambda-function-name.log | grep 'ERROR'
```

如需詳細資訊，請參閱[the section called “日誌故障診斷”](#)。

確認 Lambda 函數已成功部署

1. 在 /lambda 目錄中列出已部署 Lambda 的內容。先執行命令，再取代預留位置值。

```
cd /greengrass/ggc/deployment/lambda/  
arn:aws:lambda:region:account:function:function-name:function-version  
ls -la
```

2. 驗證目錄是否包含與您在 中上傳的 `optimizedImageClassification.zip` 部署套件相同的內容 [步驟 3：建立推論 Lambda 函數](#)。

確保 `.py` 檔案和相依性皆位於目錄的根中。

驗證是否已成功部署推論模型

1. 尋找 Lambda 執行時間程序的程序識別號碼 (PID)：

```
ps aux | grep lambda-function-name
```

在輸出中，PID 會出現在 Lambda 執行時間程序行的第二欄。

2. 輸入 Lambda 執行時間命名空間。執行命令前，務必先取代預留位置 *pid* 值。

Note

此目錄及其內容位於 Lambda 執行時間命名空間中，因此不會在一般 Linux 命名空間中顯示。

```
sudo nsenter -t pid -m /bin/bash
```

3. 列出您為機器學習資源所指定的本機目錄內容。

Note

如果您的 ML 資源路徑是 `ml_model` 以外的路徑，則您必須在這裡取代該路徑。

```
cd /ml_model  
ls -ls
```

您會看到以下檔案：

```
56 -rw-r--r-- 1 ggc_user ggc_group    56703 Oct 29 20:07 model.json  
196152 -rw-r--r-- 1 ggc_user ggc_group 200855043 Oct 29 20:08 model.params  
256 -rw-r--r-- 1 ggc_user ggc_group   261848 Oct 29 20:07 model.so  
32 -rw-r--r-- 1 ggc_user ggc_group    30564 Oct 29 20:08 synset.txt
```

Lambda 函數找不到 `/dev/dri/renderD128`

如果 OpenCL 無法連線到所需的 GPU 裝置，可能會發生這種情況。您必須為 Lambda 函數的必要裝置建立裝置資源。

後續步驟

接著，探索其他最佳化的模型。如需詳細資訊，請參閱 [SageMaker AI Neo 文件](#)。

管理 AWS IoT Greengrass 核心上的資料串流

AWS IoT Greengrass 串流管理員可讓您更輕鬆、更可靠地將大量 IoT 資料傳輸至 AWS 雲端。串流管理員會在本機處理資料串流，並 AWS 雲端 自動將其匯出至 。此功能會與常見的邊緣案例整合，例如機器學習 (ML) 推論，其中資料會在本機處理和分析，然後再匯出至 AWS 雲端 或本機儲存目的地。

串流管理員簡化了應用程式的開發。您的 IoT 應用程式可以使用標準化機制來處理大量串流並管理本機資料保留政策，而無須建立自訂串流管理功能。IoT 應用程式可以讀取和寫入串流。它們可以針對每個串流定義儲存類型、大小和資料保留政策，以控制串流管理員處理和匯出串流的方式。

串流管理員是為了在具有間歇性或連線能力有限的環境中工作而設計。您可以定義頻寬使用、逾時行為，以及當核心已連線或中斷連線時如何處理串流資料。對於關鍵資料，您可以設定優先順序，以控制串流匯出至 的順序 AWS 雲端。

您可以設定自動匯出至 AWS 雲端 以供儲存或進一步處理和分析。串流管理員支援匯出至下列 AWS 雲端 目的地。

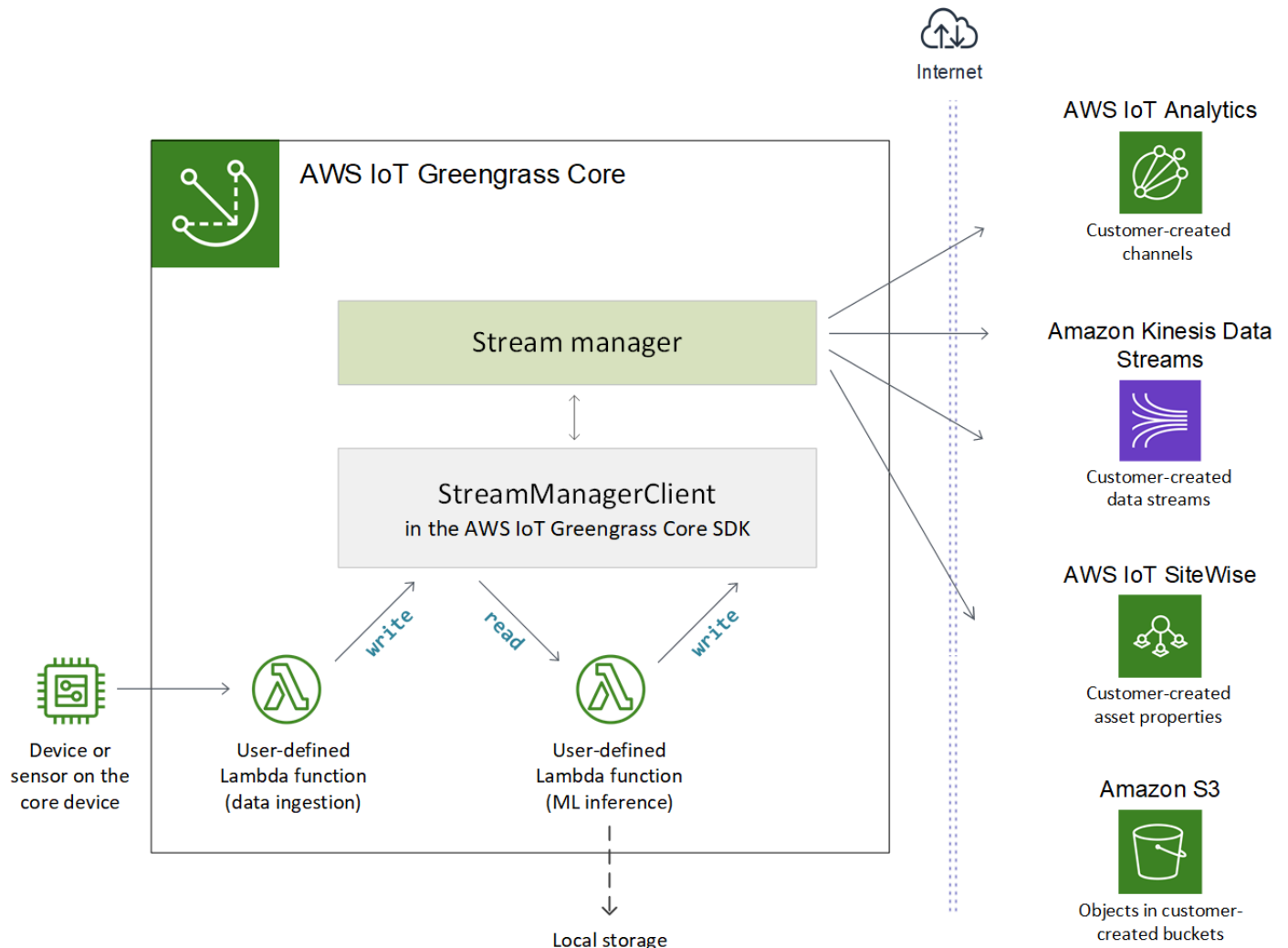
- 中的頻道 AWS IoT Analytics。AWS IoT Analytics 可讓您對資料執行進階分析，以協助做出業務決策並改善機器學習模型。如需詳細資訊，請參閱AWS IoT Analytics 《使用者指南》中的[什麼是 AWS IoT Analytics ?](#)。
- Kinesis Data Streams 中的串流。Kinesis Data Streams 通常用於彙總大量資料，並將其載入資料倉儲或地圖縮減叢集。如需詳細資訊，請參閱 [《Amazon Kinesis 開發人員指南》中的什麼是 Amazon Kinesis Data Streams ?](#)。Amazon Kinesis
- 中的資產屬性 AWS IoT SiteWise。AWS IoT SiteWise 可讓您大規模收集、組織和分析工業設備的資料。如需詳細資訊，請參閱AWS IoT SiteWise 《使用者指南》中的[什麼是 AWS IoT SiteWise ?](#)。
- Amazon S3 中的物件。您可以使用 Amazon S3 來存放和擷取大量資料。如需詳細資訊，請參閱 [《Amazon Simple Storage Service 開發人員指南》中的什麼是 Amazon S3 ?](#)。

串流管理工作流程

您的 IoT 應用程式會透過 AWS IoT Greengrass 核心 SDK 與串流管理員互動。在簡單的工作流程中，在 Greengrass 核心上執行的使用者定義 Lambda 函數會使用 IoT 資料，例如時間序列溫度和壓力指標。Lambda 函數可能會篩選或壓縮資料，然後呼叫 AWS IoT Greengrass 核心 SDK 將資料寫入串流管理員中的串流。串流管理員可以根據為串流定義的政策，自動將串流匯出至 AWS 雲端。使用者定義的 Lambda 函數也可以將資料直接傳送到本機資料庫或儲存儲存庫。

您的 IoT 應用程式可以包含多個使用者定義的 Lambda 函數，這些函數可讀取或寫入串流。這些本機 Lambda 函數可以讀取和寫入串流，以在本機篩選、彙總和分析資料。如此便可在資料從核心傳輸到雲端或本機目的地之前，快速回應本機事件並擷取有價值的資訊。

工作流程範例如下圖所示。



若要使用串流管理員，請先設定串流管理員參數，以定義套用至 Greengrass 核心上所有串流的群組層級執行時間設定。這些可自訂的設定可讓您根據您的業務需求和環境限制，控制串流管理員如何存放、處理和匯出串流。如需詳細資訊，請參閱[the section called “設定串流管理員”](#)。

設定串流管理員之後，您可以建立和部署 IoT 應用程式。這些通常是使用者定義的 Lambda 函數，在 AWS IoT Greengrass 核心 SDK StreamManagerClient 中使用來建立串流並與之互動。在建立串流期間，Lambda 函數會定義每個串流政策，例如匯出目的地、優先順序和持久性。如需詳細資訊，包括 StreamManagerClient 操作的程式碼片段，請參閱[the section called “使用 StreamManagerClient 搭配串流”](#)。

如需設定簡單工作流程的教學課程，請參閱 [the section called “匯出資料串流 \(主控台\)”](#) 或 [the section called “匯出資料串流 \(CLI\)”](#)。

要求

下列需求適用於使用串流管理員：

- 您必須使用 AWS IoT Greengrass Core 軟體 v1.10 或更新版本，並啟用串流管理員。如需詳細資訊，請參閱 [the section called “設定串流管理員”](#)。

OpenWrt 發行版本不支援串流管理員。

- Java 8 執行時間 (JDK 8) 必須安裝在核心上。
 - 針對以 Debian 為基礎的發行版本 (包括 Raspbian) 或以 Ubuntu 為基礎的發行版本，請執行下列命令：

```
sudo apt install openjdk-8-jdk
```

- 針對以 Red Hat 為基礎的發行版本 (包括 Amazon Linux)，請執行下列命令：

```
sudo yum install java-1.8.0-openjdk
```

如需詳細資訊，請參閱 OpenJDK 文件上的 [如何下載和安裝預先建置的 OpenJDK 套件](#)。

- 除了基本 AWS IoT Greengrass 核心軟體之外，串流管理員至少需要 70 MB RAM。您的總記憶體需求視您的工作負載而定。
- 使用者定義的 Lambda 函數必須使用 [AWS IoT Greengrass 核心 SDK](#) 與串流管理員互動。AWS IoT Greengrass 核心 SDK 提供多種語言，但只有下列版本支援串流管理員操作：
 - Java SDK (v1.4.0 或更新版本)
 - Python SDK (1.5.0 版或更新版本)
 - Node.js SDK (1.6.0 版或更新版本)

下載對應至 Lambda 函數執行時間的 SDK 版本，並將其包含在 Lambda 函數部署套件中。

Note

適用於 Python 的 AWS IoT Greengrass 核心 SDK 需要 Python 3.7 或更新版本，並具有其他套件相依性。如需詳細資訊，請參閱[建立 Lambda 函數部署套件（主控台）](#)或[建立 Lambda 函數部署套件 \(CLI\)](#)。

- 如果您定義串流的 AWS 雲端 匯出目的地，則必須建立匯出目標，並在 Greengrass 群組角色中授予存取許可。視目的地而定，可能也會套用其他要求。如需詳細資訊，請參閱：
 - [the section called “AWS IoT Analytics 頻道”](#)
 - [the section called “Amazon Kinesis 資料串流”](#)
 - [the section called “AWS IoT SiteWise 資產屬性”](#)
 - [the section called “Amazon S3 物件”](#)

您負責維護這些 AWS 雲端 資源。

資料安全

當您使用串流管理員時，請注意下列安全性考量。

本機資料安全性

AWS IoT Greengrass 不會在核心裝置上的元件之間加密靜態或本機傳輸的串流資料。

- 靜態資料。串流資料會在本機存放在 Greengrass 核心的儲存目錄中。為了資料安全，如果啟用，AWS IoT Greengrass 則依賴 Unix 檔案許可和全磁碟加密。您可以使用選用的 [STREAM_MANAGER_STORE_ROOT_DIR](#) 參數來指定儲存目錄。如果您稍後將此參數變更為使用不同的儲存目錄，AWS IoT Greengrass 不會刪除先前的儲存目錄或其內容。
- 本機傳輸中的資料。AWS IoT Greengrass 不會加密 核心上本機傳輸中的串流資料，包括資料來源、Lambda 函數、AWS IoT Greengrass 核心 SDK 和串流管理員。
- 傳輸到 的資料 AWS 雲端。串流管理員匯出至 的資料串流 AWS 雲端，使用標準 AWS 服務用戶端加密搭配 Transport Layer Security (TLS)。

如需詳細資訊，請參閱[the section called “資料加密”](#)。

用戶端身分驗證

串流管理員用戶端使用 AWS IoT Greengrass 核心 SDK 與串流管理員通訊。啟用用戶端身分驗證時，只有 Greengrass 群組中的 Lambda 函數可以與串流管理員中的串流互動。用戶端身分驗證停用時，Greengrass 核心上執行的任何處理程序 (例如 [Docker 容器](#)) 都可以與串流管理員中的串流互動。您應該只在商業案例需要時，才停用身分驗證。

您可以使用 [STREAM_MANAGER_AUTHENTICATE_CLIENT](#) 參數來設定用戶端身分驗證模式。您可以從 主控台或 AWS IoT Greengrass API 設定此參數。變更會在部署群組之後生效。

	已啟用	已停用
參數值	true (預設和建議)	false
允許的用戶端	Greengrass 群組中的使用者定義 Lambda 函數	Greengrass 群組中的使用者定義 Lambda 函數 在 Greengrass 核心裝置上執行的其他程序

另請參閱

- [the section called “設定串流管理員”](#)
- [the section called “使用 StreamManagerClient 搭配串流”](#)
- [the section called “匯出支援 AWS 雲端 目的地的組態”](#)
- [the section called “匯出資料串流 \(主控台\)”](#)
- [the section called “匯出資料串流 \(CLI\)”](#)

設定 AWS IoT Greengrass 串流管理員

在 AWS IoT Greengrass 核心上，串流管理員可以存放、處理和匯出 IoT 裝置資料。串流管理員可提供您用來設定群組層級執行時間設定的參數。這些設定適用於 Greengrass 核心上的所有串流。您可以使用 AWS IoT 主控台或 AWS IoT Greengrass API 來設定串流管理員設定。變更會在部署群組之後生效。

Note

設定串流管理員之後，您可以建立和部署在 Greengrass 核心上執行並與串流管理員互動的 IoT 應用程式。這些 IoT 應用程式通常是使用者定義的 Lambda 函數。如需詳細資訊，請參閱[the section called “使用 StreamManagerClient 搭配串流”](#)。

串流管理員參數

串流管理員提供下列參數，可讓您定義群組層級的設定。所有參數都是選用的。

儲存目錄

參數名稱：STREAM_MANAGER_STORE_ROOT_DIR

用來儲存串流的本機目錄絕對路徑。此值必須以正斜線開頭 (例如 /data)。

如需保護串流資料安全的資訊，請參閱 [the section called “本機資料安全性”](#)。

最低 AWS IoT Greengrass 核心版本：1.10.0

伺服器連接埠

參數名稱：STREAM_MANAGER_SERVER_PORT

用於與串流管理員通訊的本機連接埠號碼。預設值為 8088。

最低 AWS IoT Greengrass 核心版本：1.10.0

驗證用戶端

參數名稱：STREAM_MANAGER_AUTHENTICATE_CLIENT

表示用戶端是否須經過驗證才能與串流管理員互動。用戶端和串流管理員之間的所有互動都由 AWS IoT Greengrass 核心 SDK 控制。此參數會決定哪些用戶端可以呼叫 AWS IoT Greengrass 核心 SDK 來使用串流。如需詳細資訊，請參閱[the section called “用戶端身分驗證”](#)。

有效值為 true 或 false。預設值為 true (建議)。

- true。僅允許 Greengrass Lambda 函數做為用戶端。Lambda 函數用戶端使用內部 AWS IoT Greengrass 核心通訊協定來驗證 AWS IoT Greengrass 核心 SDK。

- `false`。允許在 AWS IoT Greengrass 核心上執行的任何程序成為用戶端。除非業務案例需要，否則請勿設定為 `false`。例如，`false` 只有在核心裝置上的非 Lambda 程序必須直接與串流管理員通訊時，才會將此值設定為 `false`，例如在核心上執行的 [Docker 容器](#)。

最低 AWS IoT Greengrass 核心版本：1.10.0

最高頻寬

參數名稱：STREAM_MANAGER_EXPORTER_MAX_BANDWIDTH

可用來匯出資料的平均最高頻寬 (以千位元數/秒為單位)。預設允許無限使用可用頻寬。

最低 AWS IoT Greengrass 核心版本：1.10.0

執行緒集區大小

參數名稱：STREAM_MANAGER_EXPORTER_THREAD_POOL_SIZE

可用於匯出資料的作用中執行緒數量上限。預設值為 5。

最佳大小取決於您的硬體、串流磁碟區和規劃的匯出串流數量。如果匯出速度很慢，您可以調整此設定，找出適合您硬體和商務案例的最佳大小。核心裝置硬體的 CPU 和記憶體是限制因素。首先，您可以嘗試將此值設定為等同於裝置上處理器核心的數量。

請小心不要設定高於硬體可支援的大小。每個串流都會消耗硬體資源，因此您應該嘗試限制受限裝置上的匯出串流數目。

最低 AWS IoT Greengrass 核心版本：1.10.0

JVM 引數

參數名稱：JVM_ARGS

自訂 Java 虛擬機器參數，以在啟動時傳遞給串流管理員。多個引數應用空格分隔。

僅限必須覆寫 JVM 使用的預設設定時，才能使用此參數。例如，如果您計劃匯出大量串流，可能需要增加預設堆積大小。

最低 AWS IoT Greengrass 核心版本：1.10.0

唯讀輸入檔案目錄

參數名稱：STREAM_MANAGER_READ_ONLY_DIRS

以逗號分隔的清單，列出存放輸入檔案之根檔案系統外部目錄的絕對路徑。串流管理員會讀取檔案並將其上傳至 Amazon S3，並將目錄掛載為唯讀。如需匯出至 Amazon S3 的詳細資訊，請參閱 [the section called “Amazon S3 物件”](#)。

只有在下列情況為 true 時，才使用此參數：

- 匯出至 Amazon S3 之串流的輸入檔案目錄位於下列其中一個位置：
 - 根檔案系統以外的分割區。
 - 在根檔案系統的 /tmp 下。
- Greengrass 群組的 [預設容器化](#) 為 Greengrass 容器。

範例值：/mnt/directory-1,/mnt/directory-2,/tmp

最低 AWS IoT Greengrass 核心版本：1.11.0

分段上傳的大小下限

參數名

稱：STREAM_MANAGER_EXPORTER_S3_DESTINATION_MULTIPART_UPLOAD_MIN_PART_SIZE_BYTE

分段中組件的大小下限（以位元組為單位）上傳至 Amazon S3。串流管理員使用此設定和輸入檔案的大小，來決定如何在分段 PUT 請求中批次處理資料。預設和最小值為 5242880 位元組 (5 MB)。

Note

串流管理員使用串流的 `sizeThresholdForMultipartUploadBytes` 屬性來判斷要匯出至 Amazon S3 做為單一或分段上傳。使用者定義的 Lambda 函數會在建立匯出至 Amazon S3 的串流時設定此閾值。預設閾值為 5 MB。

最低 AWS IoT Greengrass 核心版本：1.11.0

設定串流管理員設定 (主控台)

您可以使用 AWS IoT 主控台執行下列管理任務：

- [檢查是否已啟用串流管理員](#)
- [在群組建立期間啟用或停用串流管理員](#)

- [啟用或停用現有群組的串流管理員](#)
- [變更串流管理員設定](#)

部署 Greengrass 群組之後，變更就會生效。如需說明如何部署 Greengrass 群組的教學課程，其中包含與串流管理員互動的 Lambda 函數，請參閱 [the section called “匯出資料串流 \(主控台\)”](#)。

 Note

當您使用主控台啟用串流管理員並部署群組時，串流管理員的記憶體大小預設為 4194304 KB (4 GB)。我們建議您將記憶體大小設定為至少 128000 KB。

檢查是否已啟用串流管理員 (主控台)

1. 在 AWS IoT 主控台導覽窗格的管理下，展開 Greengrass 裝置，然後選擇群組 (V1)。
2. 選擇目標群組。
3. 選擇 Lambda 函數索引標籤。
4. 在 System Lambda 函數下，選取串流管理員，然後選擇編輯。
5. 檢查啟用或停用狀態。還會顯示已設定的任何自訂串流管理員設定。

在群組建立期間啟用或停用串流管理員 (主控台)

1. 在 AWS IoT 主控台導覽窗格的管理下，展開 Greengrass 裝置，然後選擇群組 (V1)。
2. 選擇 Create Group (建立群組)。您在下一個頁面的選擇會決定設定群組串流管理員的方法。
3. 繼續進行命名您的群組，然後選擇 Greengrass 核心頁面。
4. 選擇建立群組。
5. 在群組組態頁面上，選擇 Lambda 函數索引標籤，選取串流管理員，然後選擇編輯。
 - 若要使用預設設定啟用串流管理員，請選擇使用預設設定啟用。
 - 若要以自訂設定啟用串流管理員，請選擇 Customize settings (自訂設定)。

1. 在設定串流管理員頁面上，選擇使用自訂設定啟用。
 2. 在 Custom settings (自訂設定) 下，輸入串流管理員參數的值。如需詳細資訊，請參閱[the section called “串流管理員參數”](#)。將欄位保留空白 AWS IoT Greengrass，以允許使用其預設值。
- 若要停用串流管理員，請選擇停用。
 1. 在 Configure stream manager (設定串流管理員) 頁面上，選擇 Disable (停用)。
6. 選擇儲存。
 7. 繼續透過剩餘的頁面建立群組。
 8. 在用戶端裝置頁面上，下載您的安全資源，檢閱資訊，然後選擇完成。

Note

啟用串流管理員時，您必須先在核心裝置上[安裝 Java 8 執行時間](#)，才能部署群組。

啟用或停用現有群組 (主控台) 的串流管理員

1. 在 AWS IoT 主控台導覽窗格的管理下，展開 Greengrass 裝置，然後選擇群組 (V1)。
2. 選擇目標群組。
3. 選擇 Lambda 函數索引標籤。
4. 在 System Lambda 函數下，選取串流管理員，然後選擇編輯。
5. 檢查啟用或停用狀態。還會顯示已設定的任何自訂串流管理員設定。

變更串流管理員設定 (主控台)

1. 在 AWS IoT 主控台導覽窗格的管理下，展開 Greengrass 裝置，然後選擇群組 (V1)。
2. 選擇目標群組。
3. 選擇 Lambda 函數索引標籤。

4. 在 System Lambda 函數下，選取串流管理員，然後選擇編輯。
5. 檢查啟用或停用狀態。還會顯示已設定的任何自訂串流管理員設定。
6. 選擇儲存。

設定串流管理員設定 (CLI)

在 AWS CLI 中，使用系統 GGStreamManager Lambda 函數來設定串流管理員。System Lambda 函數是 AWS IoT Greengrass Core 軟體的元件。對於串流管理員和其他一些系統 Lambda 函數，您可以透過管理 Greengrass 群組中對應的 Function 和 FunctionDefinitionVersion 物件來設定 Greengrass 功能。如需詳細資訊，請參閱[the section called “群組物件模型概觀”](#)。

您可以使用 API 執行下列管理任務。本節中的範例示範如何使用 AWS CLI，但您也可以直接呼叫 AWS IoT Greengrass API 或使用 SDK。

- [檢查是否已啟用串流管理員](#)
- [啟用、停用或設定串流管理員](#)

變更會在部署群組之後生效。如需說明如何使用與串流管理員互動的 Lambda 函數部署 Greengrass 群組的教學課程，請參閱[the section called “匯出資料串流 \(CLI\)”](#)。

Tip

若要查看串流管理員是否已啟用並從核心裝置執行，您可以在裝置上的終端機中執行下列命令。

```
ps aux | grep -i 'streammanager'
```

檢查是否已啟用串流管理員 (CLI)

如果您部署的函數定義版本包含系統 GGStreamManager Lambda 函數，則會啟用串流管理員。若要檢查，請執行下列動作：

1. 取得目標 Greengrass 群組 ID 和目標群組版本 ID。此程序假設這是最新的群組和群組版本。下列查詢會傳回最近建立的群組。

```
aws greengrass list-groups --query "reverse(sort_by(Groups, &CreationTimestamp))
[0]"
```

或者，您可以依名稱查詢。群組名稱不需要是唯一名稱，因此可能會傳回多個群組。

```
aws greengrass list-groups --query "Groups[?Name=='MyGroup']"
```

 Note

您也可以在 AWS IoT 主控台中找到這些值。群組 ID 會顯示在群組的 Settings (設定) 頁面上。群組版本 IDs 會顯示在群組的部署索引標籤上。

2. 從輸出中的目標群組複製 Id 和 LatestVersion 值。

3. 取得最新的群組版本。

- 將 *group-id* 取代為您複製 Id 的。
- 將 *latest-group-version-id* 取代為您複製 LatestVersion 的。

```
aws greengrass get-group-version \
--group-id group-id \
--group-version-id latest-group-version-id
```

4. 從輸出中的 FunctionDefinitionVersionArn，取得函數定義的 ID 和函數定義版本。

- 函數定義 ID 是遵循 Amazon Resource Name (ARN) 中客 functions 群的 GUID。
- 函數定義版本 ID 是遵循 ARN 中的 versions 區段的 GUID。

```
arn:aws:greengrass:us-west-2:123456789012:/greengrass/definition/
functions/function-definition-id/versions/function-definition-version-id
```

5. 取得函數定義版本。

- 用函數定義 ID 替換 *function-definition-id*。
- 用函數定義版本 ID 替換 *function-definition-version-id*。


```
aws greengrass get-function-definition-version \  
--function-definition-id function-definition-id \  
--function-definition-version-id function-definition-version-id
```

如果輸出中的 `functions` 陣列包括 `GGStreamManager` 函數，則會啟用串流管理員。為函數定義的任何環境變數代表串流管理員的自訂設定。

若要啟用、停用或設定串流管理員 (CLI)

在 `aws greengrass` 中，使用系統 `GGStreamManager` Lambda 函數來設定串流管理員。變更會在部署群組之後生效。

- 若要啟用串流管理員，請加入函數定義版本的 `functions` 陣列中的 `GGStreamManager`。若要設定自訂設定，請為對應的[串流管理員參數](#)定義環境變數。
- 若要停用串流管理員，請從函數定義版本的 `functions` 陣列中移除 `GGStreamManager`。

採用預設設定的串流管理員

下列範例設定會以預設設定啟用串流管理員。它將任意函數 ID 設為 `streamManager`。

```
{  
  "FunctionArn": "arn:aws:lambda::function:GGStreamManager:1",  
  "FunctionConfiguration": {  
    "MemorySize": 4194304,  
    "Pinned": true,  
    "Timeout": 3  
  },  
  "Id": "streamManager"  
}
```

Note

對於 `FunctionConfiguration` 屬性，您可能知道下列事項：

- `MemorySize` 設定為具有預設設定的 4194304 KB (4 GB)。您可以隨時變更此值。我們建議您將 `MemorySize` 設定為至少 128000 KB。
- `Pinned` 必須設定為 `true`。

- 函數定義版本需要 Timeout，但 GGStreamManager 不會使用它。

採用自訂設定的串流管理員

下列範例組態可讓串流管理員具有儲存目錄、伺服器連接埠和執行緒集區大小參數的自訂值。

```
{
  "FunctionArn": "arn:aws:lambda:::function:GGStreamManager:1",
  "FunctionConfiguration": {
    "Environment": {
      "Variables": {
        "STREAM_MANAGER_STORE_ROOT_DIR": "/data",
        "STREAM_MANAGER_SERVER_PORT": "1234",
        "STREAM_MANAGER_EXPORTER_THREAD_POOL_SIZE": "4"
      }
    },
    "MemorySize": 4194304,
    "Pinned": true,
    "Timeout": 3
  },
  "Id": "streamManager"
}
```

AWS IoT Greengrass 對於未指定為環境變數的[串流管理員參數](#)，會使用預設值。

具有 Amazon S3 匯出自訂設定的串流管理員

下列範例組態可讓串流管理員具有上傳目錄的自訂值，以及最小分段上傳大小參數。

```
{
  "FunctionArn": "arn:aws:lambda:::function:GGStreamManager:1",
  "FunctionConfiguration": {
    "Environment": {
      "Variables": {
        "STREAM_MANAGER_READ_ONLY_DIRS": "/mnt/directory-1,/mnt/
directory-2,/tmp",
        "STREAM_MANAGER_EXPORTER_S3_DESTINATION_MULTIPART_UPLOAD_MIN_PART_SIZE_BYTES":
        "10485760"
      }
    },
  },
}
```

```
    "MemorySize": 4194304,  
    "Pinned": true,  
    "Timeout": 3  
  },  
  "Id": "streamManager"  
}
```

若要啟用、停用或設定串流管理員 (CLI)

1. 取得目標 Greengrass 群組 ID 和目標群組版本 ID。此程序假設這是最新的群組和群組版本。下列查詢會傳回最近建立的群組。

```
aws greengrass list-groups --query "reverse(sort_by(Groups, &CreationTimestamp))  
[0]"
```

或者，您可以依名稱查詢。群組名稱不需要是唯一名稱，因此可能會傳回多個群組。

```
aws greengrass list-groups --query "Groups[?Name=='MyGroup']"
```

Note

您也可以在此 AWS IoT 主控台找到這些值。群組 ID 會顯示在群組的 Settings (設定) 頁面上。群組版本 IDs 會顯示在群組的部署索引標籤上。

2. 從輸出中的目標群組複製 Id 和 LatestVersion 值。
3. 取得最新的群組版本。
 - 將 *group-id* 取代為您複製 Id 的。
 - 將 *latest-group-version-id* 取代為您複製 LatestVersion 的。

```
aws greengrass get-group-version \  
--group-id group-id \  
--group-version-id latest-group-version-id
```

4. 從輸出複製 CoreDefinitionVersionArn 和所有其他版本 ARNs，但除外 FunctionDefinitionVersionArn。您稍後會在建立群組版本時使用這些值。

5. 從輸出中的 `FunctionDefinitionVersionArn`，複製函數定義的 ID。該 ID 是 ARN 中跟在 `functions` 區段後面的 GUID，如下列範例所示。

```
arn:aws:greengrass:us-west-2:123456789012:/greengrass/
definition/functions/bcfc6b49-beb0-4396-b703-6dEXAMPLEcu5/
versions/0f7337b4-922b-45c5-856f-1aEXAMPLEsf6
```

 Note

或者，您可以執行 [create-function-definition](#) 命令，然後從輸出複製 ID 來建立函數定義。

6. 新增函數定義版本至函數定義。
 - 將 *function-definition-id* 取代為您為函數定義複製的 Id。
 - 在 `functions` 陣列中，包含您要在 Greengrass 核心上使用的所有其他函數。您可以使用 `get-function-definition-version` 命令來取得現有函數的清單。

以預設設定啟用串流管理員

下列範例透過在 `functions` 陣列中包含 `GGStreamManager` 函數來啟用串流管理員。此範例使用 [串流管理員參數](#) 的預設值。

```
aws greengrass create-function-definition-version \
--function-definition-id function-definition-id \
--functions '[
    {
        "FunctionArn": "arn:aws:lambda::function:GGStreamManager:1",
        "FunctionConfiguration": {
            "MemorySize": 4194304,
            "Pinned": true,
            "Timeout": 3
        },
        "Id": "streamManager"
    },
    {
        "FunctionArn": "arn:aws:lambda:us-
west-2:123456789012:function:MyLambdaFunction:MyAlias",
```

```

        "FunctionConfiguration": {
            "Executable": "myLambdaFunction.function_handler",
            "MemorySize": 16000,
            "Pinned": true,
            "Timeout": 5
        },
        "Id": "myLambdaFunction"
    },
    ... more user-defined functions
]
}'

```

Note

範例中的 myLambdaFunction 函數代表其中一個使用者定義的 Lambda 函數。

用自訂設定啟用串流管理員

下列範例會藉由在 functions 陣列中包括 GGStreamManager 函數來啟用串流管理員。除非您想要變更預設值，否則所有串流管理員設定都是選用的。此範例說明如何使用環境變數來設定自訂值。

```

aws greengrass create-function-definition-version \
--function-definition-id function-definition-id \
--functions '[
    {
        "FunctionArn": "arn:aws:lambda::function:GGStreamManager:1",
        "FunctionConfiguration": {
            "Environment": {
                "Variables": {
                    "STREAM_MANAGER_STORE_ROOT_DIR": "/data",
                    "STREAM_MANAGER_SERVER_PORT": "1234",
                    "STREAM_MANAGER_EXPORTER_THREAD_POOL_SIZE": "4"
                }
            },
            "MemorySize": 4194304,
            "Pinned": true,
            "Timeout": 3
        },
        "Id": "streamManager"
    },
]'

```

```
{
  "FunctionArn": "arn:aws:lambda:us-
west-2:123456789012:function:MyLambdaFunction:MyAlias",
  "FunctionConfiguration": {
    "Executable": "myLambdaFunction.function_handler",
    "MemorySize": 16000,
    "Pinned": true,
    "Timeout": 5
  },
  "Id": "myLambdaFunction"
},
... more user-defined functions
]
```

Note

對於 FunctionConfiguration 屬性，您可能知道下列事項：

- MemorySize 設定為具有預設設定的 4194304 KB (4 GB)。您可以隨時變更此值。我們建議您將 MemorySize 設定為至少 128000 KB。
- Pinned 必須設定為 true。
- 函數定義版本需要 Timeout，但 GGStreamManager 不會使用它。

停用串流管理員

下列範例會省略停用串流管理員的 GGStreamManager 函數。

```
aws greengrass create-function-definition-version \
--function-definition-id function-definition-id \
--functions '[
  {
    "FunctionArn": "arn:aws:lambda:us-
west-2:123456789012:function:MyLambdaFunction:MyAlias",
    "FunctionConfiguration": {
      "Executable": "myLambdaFunction.function_handler",
      "MemorySize": 16000,
      "Pinned": true,
      "Timeout": 5
    },
    "Id": "myLambdaFunction"
  }
]
```

```
    },
    ... more user-defined functions
  ]
}'
```

 Note

如果您不想部署任何 Lambda 函數，您可以完全省略函數定義版本。

7. 從輸出複製函數定義版本的 Arn。
8. 建立包含系統 Lambda 函數的群組版本。
 - 為該群組將 *group-id* 取代為 Id。
 - 將 *core-definition-version-arn* 取代為您從最新群組版本所複製的 CoreDefinitionVersionArn。
 - 將 *function-definition-version-arn* 取代為您為功能定義版本所複製的 Arn。
 - 取代您已從最新群組版本複製之其他群組元件 (例如 SubscriptionDefinitionVersionArn 或 DeviceDefinitionVersionArn) 的 ARN。
 - 移除任何未使用的參數。比如說若您的群組版本不包含任何資源，則請移除 `--resource-definition-version-arn`。

```
aws greengrass create-group-version \
--group-id group-id \
--core-definition-version-arn core-definition-version-arn \
--function-definition-version-arn function-definition-version-arn \
--device-definition-version-arn device-definition-version-arn \
--logger-definition-version-arn logger-definition-version-arn \
--resource-definition-version-arn resource-definition-version-arn \
--subscription-definition-version-arn subscription-definition-version-arn
```

9. 從輸出複製 Version。這是新群組版本的 ID。
10. 部署具有新群組版本的群組。
 - 將 *group-id* 取代為您為群組所複製的 Id。
 - 將 *group-version-id* 取代為您為新群組版本所複製的 Version。

```
aws greengrass create-deployment \
```

```
--group-id group-id \  
--group-version-id group-version-id \  
--deployment-type NewDeployment
```

如果您想要稍後再次編輯串流管理員設定，請遵循此程序。請務必建立函數定義版本，其中包含具有更新組態的GGStreamManager函數。群組版本必須參考您要部署到核心的所有元件版本 ARNs。變更會在部署群組之後生效。

另請參閱

- [管理資料串流](#)
- [the section called “使用 StreamManagerClient 搭配串流”](#)
- [the section called “匯出支援 AWS 雲端 目的地的組態”](#)
- [the section called “匯出資料串流 \(主控台\)”](#)
- [the section called “匯出資料串流 \(CLI\)”](#)

使用 StreamManagerClient 搭配串流

在 AWS IoT Greengrass 核心上執行的使用者定義 Lambda 函數可以使用[AWS IoT Greengrass 核心 SDK](#) 中的 StreamManagerClient 物件，在串流[管理員](#)中建立串流，然後與串流互動。當 Lambda 函數建立串流時，它會定義串流的 AWS 雲端 目的地、優先順序和其他匯出和資料保留政策。若要將資料傳送至串流管理員，Lambda 函數會將資料附加至串流。如果為串流定義匯出目的地，串流管理員會自動匯出串流。

Note

一般而言，串流管理員的用戶端是使用者定義的 Lambda 函數。如果您的商業案例需要，您也可以允許在 Greengrass 核心（例如 Docker 容器）上執行的非 Lambda 程序與串流管理員互動。如需詳細資訊，請參閱[the section called “用戶端身分驗證”](#)。

本主題中的程式碼片段會示範用戶端如何呼叫StreamManagerClient方法以使用串流。如需方法及其引數的實作詳細資訊，請使用每個程式碼片段後列出的 SDK 參考連結。如需包含完整 Python Lambda 函數的教學課程，請參閱 [the section called “匯出資料串流 \(主控台\)”](#)或 [the section called “匯出資料串流 \(CLI\)”](#)。

您的 Lambda 函數應該在函數處理常式 `StreamManagerClient` 之外執行個體化。如果在處理常式內執行個體化，則該函數在每次叫用時都會建立 `client` 以及與串流管理員的連線。

Note

如果您在處理常式內將 `StreamManagerClient` 執行個體化，則必須在 `client` 完成其工作時明確呼叫 `close()` 方法。否則，`client` 會將連線保持在開啟狀態，並將另一個執行緒保持在執行狀態，直到指令碼結束為止。

`StreamManagerClient` 支援下列操作：

- [the section called “建立訊息串流”](#)
- [the section called “附加訊息”](#)
- [the section called “讀取訊息”](#)
- [the section called “列出串流”](#)
- [the section called “描述訊息串流”](#)
- [the section called “更新訊息串流”](#)
- [the section called “刪除訊息串流”](#)

建立訊息串流

若要建立串流，使用者定義的 Lambda 函數會呼叫建立方法並傳入 `MessageStreamDefinition` 物件。此物件會指定串流的唯一名稱，並定義串流管理員在達到串流大小上限時應如何處理新資料。您可以使用 `MessageStreamDefinition` 及其資料類型 (例如 `ExportDefinition`、`StrategyOnFull` 和 `Persistence`) 來定義其他串流屬性。其中包含：

- 自動匯出的目標 AWS IoT SiteWise、AWS IoT Analytics Kinesis Data Streams 和 Amazon S3 目的地。如需詳細資訊，請參閱[the section called “匯出支援 AWS 雲端 目的地的組態”](#)。
- 匯出優先順序。串流管理員會先匯出優先順序較高的串流，然後再匯出較低的串流。
- AWS IoT Analytics、Kinesis Data Streams 和目的地的批次大小上限和 AWS IoT SiteWise 批次間隔。符合任一條件時，串流管理員會匯出訊息。
- 存留時間 (TTL)。保證串流資料可供處理的時間量。您應該確定資料可以在這段期間內使用。這不是刪除政策。TTL 期間後，資料可能不會立即刪除。

- 串流持久性。選擇此選項可將串流儲存至檔案系統，以便在核心重新啟動期間保留資料，或將串流儲存在記憶體中。
- 起始序號。指定要用作匯出中起始訊息的訊息序號。

如需的詳細資訊 `MessageStreamDefinition`，請參閱目標語言的 SDK 參考：

- Java 開發套件中的 [MessageStreamDefinition](#)
- Node.js SDK 中的 [MessageStreamDefinition](#)
- Python SDK 中的 [MessageStreamDefinition](#)

Note

`StreamManagerClient` 也提供目標目的地，可用來將串流匯出至 HTTP 伺服器。此目標僅供測試之用。它不穩定或不支援在生產環境中使用。

建立串流後，您的 Lambda 函數可以將[訊息附加](#)至串流，以傳送資料進行匯出，並從串流[讀取訊息](#)以進行本機處理。您建立的串流數量取決於您的硬體功能和商業案例。其中一個策略是為 AWS IoT Analytics 或 Kinesis 資料串流中的每個目標頻道建立串流，但您可以定義串流的多個目標。串流的生命週期相當耐久。

要求

此操作有下列需求：

- 最低 AWS IoT Greengrass 核心版本：1.10.0
- 最低 AWS IoT Greengrass 核心 SDK 版本：Python：1.5.0 | Java：1.4.0 | Node.js：1.6.0

Note

使用 AWS IoT SiteWise 或 Amazon S3 匯出目的地建立串流有下列需求：

- 最低 AWS IoT Greengrass 核心版本：1.11.0
- 最低 AWS IoT Greengrass 核心 SDK 版本：Python：1.6.0 | Java：1.5.0 | Node.js：1.7.0

範例

以下程式碼片段會建立名為 `StreamName` 的串流。它定義 `MessageStreamDefinition` 和次級資料類型中的串流屬性。

Python

```
client = StreamManagerClient()

try:
    client.create_message_stream(MessageStreamDefinition(
        name="StreamName", # Required.
        max_size=268435456, # Default is 256 MB.
        stream_segment_size=16777216, # Default is 16 MB.
        time_to_live_millis=None, # By default, no TTL is enabled.
        strategy_on_full=StrategyOnFull.OverwriteOldestData, # Required.
        persistence=Persistence.File, # Default is File.
        flush_on_write=False, # Default is false.
        export_definition=ExportDefinition( # Optional. Choose where/how the stream
            is exported to the AWS ##.
            kinesis=None,
            iot_analytics=None,
            iot_sitewise=None,
            s3_task_executor=None
        )
    ))
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Python SDK 參考：[create_message_stream](#) | [MessageStreamDefinition](#)

Java

```
try (final StreamManagerClient client =
    GreengrassClientBuilder.streamManagerClient().build()) {
    client.createMessageStream(
        new MessageStreamDefinition()
            .withName("StreamName") // Required.
            .withMaxSize(268435456L) // Default is 256 MB.
```

```

        .withStreamSegmentSize(16777216L) // Default is 16 MB.
        .withTimeToLiveMillis(null) // By default, no TTL is enabled.
        .withStrategyOnFull(StrategyOnFull.OverwriteOldestData) //
Required.

        .withPersistence(Persistence.File) // Default is File.
        .withFlushOnWrite(false) // Default is false.
        .withExportDefinition( // Optional. Choose where/how the stream
is exported to the AWS ##.
            new ExportDefinition()
                .withKinesis(null)
                .withIotAnalytics(null)
                .withIotSitewise(null)
                .withS3TaskExecutor(null)
            )
    );
} catch (StreamManagerException e) {
    // Properly handle exception.
}

```

Java 開發套件參考：[createMessageStream](#) | [MessageStreamDefinition](#)

Node.js

```

const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        await client.createMessageStream(
            new MessageStreamDefinition()
                .withName("StreamName") // Required.
                .withMaxSize(268435456) // Default is 256 MB.
                .withStreamSegmentSize(16777216) // Default is 16 MB.
                .withTimeToLiveMillis(null) // By default, no TTL is enabled.
                .withStrategyOnFull(StrategyOnFull.OverwriteOldestData) //
Required.

                .withPersistence(Persistence.File) // Default is File.
                .withFlushOnWrite(false) // Default is false.
                .withExportDefinition( // Optional. Choose where/how the stream is
exported to the AWS ##.
                    new ExportDefinition()
                        .withKinesis(null)
                        .withIotAnalytics(null)
                        .withIotSitewise(null)
                        .withS3TaskExecutor(null)
                    )
                )
    }
}

```

```
        );
    };
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Node.js SDK 參考：[createMessageStream](#) | [MessageStreamDefinition](#)

如需設定匯出目的地的詳細資訊，請參閱 [the section called “匯出支援 AWS 雲端 目的地的組態”](#)。

附加訊息

若要將資料傳送至串流管理員以進行匯出，您的 Lambda 函數會將資料附加至目標串流。匯出目的地會決定要傳遞至此方法的資料類型。

要求

此操作有下列需求：

- 最低 AWS IoT Greengrass 核心版本：1.10.0
- 最低 AWS IoT Greengrass 核心 SDK 版本：Python：1.5.0 | Java：1.4.0 | Node.js：1.6.0

Note

使用 AWS IoT SiteWise 或 Amazon S3 匯出目的地附加訊息有下列需求：

- 最低 AWS IoT Greengrass 核心版本：1.11.0
- 最低 AWS IoT Greengrass 核心 SDK 版本：Python：1.6.0 | Java：1.5.0 | Node.js：1.7.0

範例

AWS IoT Analytics 或 Kinesis Data Streams 匯出目的地

下列程式碼片段附加一個訊息到名為 `StreamName` 的串流。對於 AWS IoT Analytics 或 Kinesis Data Streams 目的地，您的 Lambda 函數會附加資料的 Blob。

此程式碼片段有下列需求：

- 最低 AWS IoT Greengrass 核心版本：1.10.0
- 最低 AWS IoT Greengrass 核心 SDK 版本：Python：1.5.0 | Java：1.4.0 | Node.js：1.6.0

Python

```
client = StreamManagerClient()

try:
    sequence_number = client.append_message(stream_name="StreamName",
    data=b'Arbitrary bytes data')
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Python SDK 參考：[end_message](#)

Java

```
try (final StreamManagerClient client =
    GreengrassClientBuilder.streamManagerClient().build()) {
    long sequenceNumber = client.appendMessage("StreamName", "Arbitrary byte
    array".getBytes());
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Java 開發套件參考：[appendMessage](#)

Node.js

```
const client = new StreamManagerClient();
```

```
client.onConnected(async () => {
  try {
    const sequenceNumber = await client.appendMessage("StreamName",
      Buffer.from("Arbitrary byte array"));
  } catch (e) {
    // Properly handle errors.
  }
});
client.onError((err) => {
  // Properly handle connection errors.
  // This is called only when the connection to the StreamManager server fails.
});
```

Node.js SDK 參考：[appendMessage](#)

AWS IoT SiteWise 匯出目的地

下列程式碼片段附加一個訊息到名為 StreamName 的串流。對於 AWS IoT SiteWise 目的地，您的 Lambda 函數會附加序列化PutAssetPropertyValueEntry物件。如需詳細資訊，請參閱[the section called “匯出至 AWS IoT SiteWise”](#)。

Note

當您傳送資料至 AWS IoT SiteWise，您的資料必須符合 BatchPutAssetPropertyValue動作的要求。如需詳細資訊，請參閱《AWS IoT SiteWise API 參考》中的 [BatchPutAssetPropertyValue](#)。

此程式碼片段有下列需求：

- 最低 AWS IoT Greengrass 核心版本：1.11.0
- 最低 AWS IoT Greengrass 核心 SDK 版本：Python：1.6.0 | Java：1.5.0 | Node.js：1.7.0

Python

```
client = StreamManagerClient()

try:
    # SiteWise requires unique timestamps in all messages. Add some randomness to
    # time and offset.
```

```

    # Note: To create a new asset property data, you should use the classes defined
    in the
    # greengrasssdk.stream_manager module.

    time_in_nanos = TimeInNanos(
        time_in_seconds=calendar.timegm(time.gmtime()) - random.randint(0, 60),
        offset_in_nanos=random.randint(0, 10000)
    )
    variant = Variant(double_value=random.random())
    asset = [AssetPropertyValue(value=variant, quality=Quality.GOOD,
        timestamp=time_in_nanos)]
    putAssetPropertyValueEntry =
    PutAssetPropertyValueEntry(entry_id=str(uuid.uuid4()),
        property_alias="PropertyAlias", property_values=asset)
    sequence_number = client.append_message(stream_name="StreamName",
        data=Util.validate_and_serialize_to_json_bytes(putAssetPropertyValueEntry))
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.

```

Python SDK 參考：[end_message](#) | [PutAssetPropertyValueEntry](#)

Java

```

try (final StreamManagerClient client =
    GreengrassClientBuilder.streamManagerClient().build()) {
    Random rand = new Random();
    // Note: To create a new asset property data, you should use the classes defined
    in the
    // com.amazonaws.greengrass.streammanager.model.sitewise package.
    List<AssetPropertyValue> entries = new ArrayList<>();

    // IoTSiteWise requires unique timestamps in all messages. Add some randomness
    to time and offset.
    final int maxTimeRandomness = 60;
    final int maxOffsetRandomness = 10000;
    double randomValue = rand.nextDouble();
    TimeInNanos timestamp = new TimeInNanos()
        .withTimeInSeconds(Instant.now().getEpochSecond() -
            rand.nextInt(maxTimeRandomness))

```



```

        .withOffsetInNanos((long) (rand.nextInt(maxOffsetRandomness)));
AssetPropertyValue entry = new AssetPropertyValue()
    .withValue(new Variant().withDoubleValue(randomValue))
    .withQuality(Quality.GOOD)
    .withTimestamp(timestamp);
entries.add(entry);

PutAssetPropertyValueEntry putAssetPropertyValueEntry = new
PutAssetPropertyValueEntry()
    .withEntryId(UUID.randomUUID().toString())
    .withPropertyAlias("PropertyAlias")
    .withPropertyValues(entries);
long sequenceNumber = client.appendMessage("StreamName",
ValidateAndSerialize.validateAndSerializeToJsonBytes(putAssetPropertyValueEntry));
} catch (StreamManagerException e) {
    // Properly handle exception.
}

```

Java 開發套件參考：[appendMessage](#) | [PutAssetPropertyValueEntry](#)

Node.js

```

const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        const maxTimeRandomness = 60;
        const maxOffsetRandomness = 10000;
        const randomValue = Math.random();
        // Note: To create a new asset property data, you should use the classes
        defined in the
        // aws-greengrass-core-sdk StreamManager module.
        const timestamp = new TimeInNanos()
            .withTimeInSeconds(Math.round(Date.now() / 1000) -
Math.floor(Math.random() * maxTimeRandomness))
            .withOffsetInNanos(Math.floor(Math.random() * maxOffsetRandomness));
        const entry = new AssetPropertyValue()
            .withValue(new Variant().withDoubleValue(randomValue))
            .withQuality(Quality.GOOD)
            .withTimestamp(timestamp);

        const putAssetPropertyValueEntry = new PutAssetPropertyValueEntry()
            .withEntryId(`${ENTRY_ID_PREFIX}${i}`)
            .withPropertyAlias("PropertyAlias")
            .withPropertyValues([entry]);
    }
}

```

```
        const sequenceNumber = await client.appendMessage("StreamName",
        util.validateAndSerializeToJsonBytes(putAssetPropertyValueEntry));
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Node.js SDK 參考：[appendMessage](#) | [PutAssetPropertyValueEntry](#)

Amazon S3 匯出目的地

下列程式碼片段會將匯出任務附加至名為 `StreamName` 的串流。對於 Amazon S3 目的地，您的 Lambda 函數會附加序列化 `S3ExportTaskDefinition` 物件，其中包含來源輸入檔案和目標 Amazon S3 物件的相關資訊。如果指定的物件不存在，串流管理員會為您建立它。如需詳細資訊，請參閱[the section called “匯出至 Amazon S3”](#)。

此程式碼片段有下列需求：

- 最低 AWS IoT Greengrass 核心版本：1.11.0
- 最低 AWS IoT Greengrass 核心 SDK 版本：Python：1.6.0 | Java：1.5.0 | Node.js：1.7.0

Python

```
client = StreamManagerClient()

try:
    # Append an Amazon S3 Task definition and print the sequence number.
    s3_export_task_definition = S3ExportTaskDefinition(input_url="URLToFile",
    bucket="BucketName", key="KeyName")
    sequence_number = client.append_message(stream_name="StreamName",
    data=Util.validate_and_serialize_to_json_bytes(s3_export_task_definition))
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
```

```
# Properly handle errors.
```

Python SDK 參考：[end_message](#) | [S3ExportTaskDefinition](#)

Java

```
try (final StreamManagerClient client =
    GreengrassClientBuilder.streamManagerClient().build()) {
    // Append an Amazon S3 export task definition and print the sequence number.
    S3ExportTaskDefinition s3ExportTaskDefinition = new S3ExportTaskDefinition()
        .withBucket("BucketName")
        .withKey("KeyName")
        .withInputUrl("URLToFile");
    long sequenceNumber = client.appendMessage("StreamName",
        ValidateAndSerialize.validateAndSerializeToJsonBytes(s3ExportTaskDefinition));
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Java 開發套件參考：[appendMessage](#) | [S3ExportTaskDefinition](#)

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        // Append an Amazon S3 export task definition and print the sequence number.
        const taskDefinition = new S3ExportTaskDefinition()
            .withBucket("BucketName")
            .withKey("KeyName")
            .withInputUrl("URLToFile");
        const sequenceNumber = await client.appendMessage("StreamName",
            util.validateAndSerializeToJsonBytes(taskDefinition));
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Node.js SDK 參考：[appendMessage](#) | [S3ExportTaskDefinition](#)

讀取訊息

從串流讀取訊息。

要求

此操作有下列需求：

- 最低 AWS IoT Greengrass 核心版本：1.10.0
- 最低 AWS IoT Greengrass 核心 SDK 版本：Python：1.5.0 | Java：1.4.0 | Node.js：1.6.0

範例

下列程式碼片段可從名為 `StreamName` 的串流讀取訊息。讀取方法需要一個選用的 `ReadMessagesOptions` 物件以指定序號，從要讀取的最小、最大數字和讀取訊息的逾時開始讀取。

Python

```
client = StreamManagerClient()

try:
    message_list = client.read_messages(
        stream_name="StreamName",
        # By default, if no options are specified, it tries to read one message from
        the beginning of the stream.
        options=ReadMessagesOptions(
            desired_start_sequence_number=100,
            # Try to read from sequence number 100 or greater. By default, this is
            0.
            min_message_count=10,
            # Try to read 10 messages. If 10 messages are not available, then
            NotEnoughMessagesException is raised. By default, this is 1.
            max_message_count=100, # Accept up to 100 messages. By default this is
            1.
            read_timeout_millis=5000
            # Try to wait at most 5 seconds for the min_message_count to be
            fulfilled. By default, this is 0, which immediately returns the messages or an
            exception.
        )
    )
```

```
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Python SDK 參考：[Read_messages](#) | [ReadMessagesOptions](#)

Java

```
try (final StreamManagerClient client =
    GreengrassClientBuilder.streamManagerClient().build()) {
    List<Message> messages = client.readMessages("StreamName",
        // By default, if no options are specified, it tries to read one message
        from the beginning of the stream.
        new ReadMessagesOptions()
            // Try to read from sequence number 100 or greater. By default
            this is 0.
            .withDesiredStartSequenceNumber(100L)
            // Try to read 10 messages. If 10 messages are not available,
            then NotEnoughMessagesException is raised. By default, this is 1.
            .withMinMessageCount(10L)
            // Accept up to 100 messages. By default this is 1.
            .withMaxMessageCount(100L)
            // Try to wait at most 5 seconds for the min_message_count to
            be fulfilled. By default, this is 0, which immediately returns the messages or an
            exception.
            .withReadTimeoutMillis(Duration.ofSeconds(5L).toMillis())
    );
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Java 開發套件參考：[readMessages](#) | [ReadMessagesOptions](#)

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        const messages = await client.readMessages("StreamName",
            // By default, if no options are specified, it tries to read one message
            from the beginning of the stream.
        );
    }
});
```

```
        new ReadMessagesOptions()
            // Try to read from sequence number 100 or greater. By default this
            // is 0.
            .withDesiredStartSequenceNumber(100)
            // Try to read 10 messages. If 10 messages are not available, then
            // NotEnoughMessagesException is thrown. By default, this is 1.
            .withMinMessageCount(10)
            // Accept up to 100 messages. By default this is 1.
            .withMaxMessageCount(100)
            // Try to wait at most 5 seconds for the minMessageCount to be
            // fulfilled. By default, this is 0, which immediately returns the messages or an
            // exception.
            .withReadTimeoutMillis(5 * 1000)
    );
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Node.js SDK 參考：[readMessages](#) | [ReadMessagesOptions](#)

列出串流

取得串流管理員中的串流清單。

要求

此操作有下列需求：

- 最低 AWS IoT Greengrass 核心版本：1.10.0
- 最低 AWS IoT Greengrass 核心 SDK 版本：Python：1.5.0 | Java：1.4.0 | Node.js：1.6.0

範例

下列程式碼片段可獲取串流管理員中的串流清單 (依據名稱)。

Python

```
client = StreamManagerClient()

try:
    stream_names = client.list_streams()
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Python SDK 參考：[list_streams](#)

Java

```
try (final StreamManagerClient client =
    GreengrassClientBuilder.streamManagerClient().build()) {
    List<String> streamNames = client.listStreams();
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Java 開發套件參考：[listStreams](#)

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        const streams = await client.listStreams();
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Node.js SDK 參考：[listStreams](#)

描述訊息串流

取得串流的中繼資料，包括串流定義、大小和匯出狀態。

要求

此操作有下列需求：

- 最低 AWS IoT Greengrass 核心版本：1.10.0
- 最低 AWS IoT Greengrass 核心 SDK 版本：Python：1.5.0 | Java：1.4.0 | Node.js：1.6.0

範例

下列程式碼片段可獲取名為 StreamName 的串流相關的中繼資料，包括串流定義、大小和匯出工具狀態。

Python

```
client = StreamManagerClient()

try:
    stream_description = client.describe_message_stream(stream_name="StreamName")
    if stream_description.export_statuses[0].error_message:
        # The last export of export destination 0 failed with some error
        # Here is the last sequence number that was successfully exported
        stream_description.export_statuses[0].last_exported_sequence_number

    if (stream_description.storage_status.newest_sequence_number >
        stream_description.export_statuses[0].last_exported_sequence_number):
        pass
        # The end of the stream is ahead of the last exported sequence number
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Python SDK 參考：[describe_message_stream](#)

Java

```
try (final StreamManagerClient client =
    GreengrassClientBuilder.streamManagerClient().build()) {
    MessageStreamInfo description = client.describeMessageStream("StreamName");
    String lastErrorMessage =
description.getExportStatuses().get(0).getErrorMessage();
    if (lastErrorMessage != null && !lastErrorMessage.equals("")) {
        // The last export of export destination 0 failed with some error.
        // Here is the last sequence number that was successfully exported.
        description.getExportStatuses().get(0).getLastExportedSequenceNumber();
    }

    if (description.getStorageStatus().getNewestSequenceNumber() >
        description.getExportStatuses().get(0).getLastExportedSequenceNumber())
    {
        // The end of the stream is ahead of the last exported sequence number.
    }
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Java 開發套件參考：[describeMessageStream](#)

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        const description = await client.describeMessageStream("StreamName");
        const lastErrorMessage = description.exportStatuses[0].errorMessage;
        if (lastErrorMessage) {
            // The last export of export destination 0 failed with some error.
            // Here is the last sequence number that was successfully exported.
            description.exportStatuses[0].lastExportedSequenceNumber;
        }

        if (description.storageStatus.newestSequenceNumber >
            description.exportStatuses[0].lastExportedSequenceNumber) {
            // The end of the stream is ahead of the last exported sequence number.
        }
    } catch (e) {
        // Properly handle errors.
    }
}
```

```
});  
client.onError((err) => {  
    // Properly handle connection errors.  
    // This is called only when the connection to the StreamManager server fails.  
});
```

Node.js SDK 參考：[describeMessageStream](#)

更新訊息串流

更新現有串流的屬性。如果您的需求在建立串流之後有所變更，建議您更新串流。例如：

- 新增 AWS 雲端 目的地的新[匯出組態](#)。
- 增加串流的大小上限，以變更資料匯出或保留的方式。例如，串流大小結合您在完整設定上的策略，可能會導致資料遭到刪除或拒絕，然後串流管理員才能處理資料。
- 暫停和繼續匯出；例如，如果匯出任務長時間執行，而且您想要對上傳資料進行評價。

您的 Lambda 函數遵循此高階程序來更新串流：

1. [取得串流的描述](#)。
2. 更新對應MessageStreamDefinition和次級物件的目標屬性。
3. 在更新的 中傳遞 MessageStreamDefinition。請務必包含更新串流的完整物件定義。未定義的屬性會還原為預設值。

您可以指定要用作匯出中起始訊息的訊息序號。

要求

此操作有下列需求：

- 最低 AWS IoT Greengrass 核心版本：1.11.0
- 最低 AWS IoT Greengrass 核心 SDK 版本：Python：1.6.0 | Java：1.5.0 | Node.js：1.7.0

範例

下列程式碼片段會更新名為 `StreamName` 的串流。它會更新匯出至 Kinesis Data Streams 之串流的多個屬性。

Python

```
client = StreamManagerClient()

try:
    message_stream_info = client.describe_message_stream(STREAM_NAME)
    message_stream_info.definition.max_size=536870912
    message_stream_info.definition.stream_segment_size=33554432
    message_stream_info.definition.time_to_live_millis=3600000
    message_stream_info.definition.strategy_on_full=StrategyOnFull.RejectNewData
    message_stream_info.definition.persistence=Persistence.Memory
    message_stream_info.definition.flush_on_write=False
    message_stream_info.definition.export_definition.kinesis=[
        KinesisConfig(
            # Updating Export definition to add a Kinesis Stream configuration.
            identifier=str(uuid.uuid4()), kinesis_stream_name=str(uuid.uuid4()))]
    client.update_message_stream(message_stream_info.definition)
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Python SDK 參考：[updateMessageStream](#) | [MessageStreamDefinition](#)

Java

```
try (final StreamManagerClient client =
    GreengrassClientBuilder.streamManagerClient().build()) {
    MessageStreamInfo messageStreamInfo = client.describeMessageStream(STREAM_NAME);
    // Update the message stream with new values.
    client.updateMessageStream(
        messageStreamInfo.getDefinition()
            .withStrategyOnFull(StrategyOnFull.RejectNewData) // Required. Updating
            Strategy on full to reject new data.
            // Max Size update should be greater than initial Max Size defined in
            Create Message Stream request
```

```

        .withMaxSize(536870912L) // Update Max Size to 512 MB.
        .withStreamSegmentSize(33554432L) // Update Segment Size to 32 MB.
        .withFlushOnWrite(true) // Update flush on write to true.
        .withPersistence(Persistence.Memory) // Update the persistence to
Memory.

        .withTimeToLiveMillis(3600000L) // Update TTL to 1 hour.
        .withExportDefinition(
            // Optional. Choose where/how the stream is exported to the AWS ##.
            messageStreamInfo.getDefinition().getExportDefinition().
            // Updating Export definition to add a Kinesis Stream
configuration.

            .withKinesis(new ArrayList<KinesisConfig>() {{
                add(new KinesisConfig()
                    .withIdentifier(EXPORT_IDENTIFIER)
                    .withKinesisStreamName("test"));
            }}))
        );
    } catch (StreamManagerException e) {
        // Properly handle exception.
    }
}

```

Java 開發套件參考：[update_message_stream](#) | [MessageStreamDefinition](#)

Node.js

```

const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        const messageStreamInfo = await c.describeMessageStream(STREAM_NAME);
        await client.updateMessageStream(
            messageStreamInfo.definition
            // Max Size update should be greater than initial Max Size defined
            in Create Message Stream request
            .withMaxSize(536870912) // Default is 256 MB. Updating Max Size to
512 MB.

            .withStreamSegmentSize(33554432) // Default is 16 MB. Updating
Segment Size to 32 MB.

            .withTimeToLiveMillis(3600000) // By default, no TTL is enabled.
Update TTL to 1 hour.

            .withStrategyOnFull(StrategyOnFull.RejectNewData) // Required.
Updating Strategy on full to reject new data.

            .withPersistence(Persistence.Memory) // Default is File. Update the
persistence to Memory

            .withFlushOnWrite(true) // Default is false. Updating to true.

```

```

        .withExportDefinition(
            // Optional. Choose where/how the stream is exported to the AWS
            ##.
            messageStreamInfo.definition.exportDefinition
            // Updating Export definition to add a Kinesis Stream
            configuration.
                .withKinesis([new
KinesisConfig().withIdentifier(uuidv4()).withKinesisStreamName(uuidv4())])
                )
        );
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});

```

Node.js SDK 參考：[updateMessageStream](#) | [MessageStreamDefinition](#)

更新串流的限制

更新串流時，適用下列限制條件。除非以下清單中另有說明，否則更新會立即生效。

- 您無法更新串流的持久性。若要變更此行為，[請刪除串流](#)並[建立定義新持久性政策的串流](#)。
- 您只能在下列情況下更新串流的大小上限：
 - 大小上限必須大於或等於串流的目前大小。若要尋找此資訊，[請描述串流](#)，然後檢查傳回MessageStreamInfo物件的儲存狀態。
 - 大小上限必須大於或等於串流的區段大小。
- 您可以將串流區段大小更新為小於串流大小上限的值。更新的設定會套用至新的客群。
- 存留時間 (TTL) 屬性的更新適用於新的附加操作。如果您減少此值，串流管理員也可能會刪除超過 TTL 的現有區段。
- 完整屬性策略的更新適用於新的附加操作。如果您設定策略來覆寫最舊的資料，串流管理員也可能會根據新設定覆寫現有的區段。
- 寫入屬性排清的更新會套用至新訊息。
- 匯出組態的更新會套用至新的匯出。更新請求必須包含您要支援的所有匯出組態。否則，串流管理員會刪除它們。

- 當您更新匯出組態時，請指定目標匯出組態的識別符。
- 若要新增匯出組態，請指定新匯出組態的唯一識別符。
- 若要刪除匯出組態，請省略匯出組態。
- 若要[更新](#)串流中匯出組態的起始序號，您必須指定小於最新序號的值。若要尋找此資訊，[請描述串流](#)，然後檢查傳回MessageStreamInfo物件的儲存狀態。

刪除訊息串流

刪除串流。刪除串流時，磁碟中該串流的所有儲存資料都會刪除。

要求

此操作有下列需求：

- 最低 AWS IoT Greengrass 核心版本：1.10.0
- 最低 AWS IoT Greengrass 核心 SDK 版本：Python：1.5.0 | Java：1.4.0 | Node.js：1.6.0

範例

下列程式碼片段會刪除名為 StreamName 的串流。

Python

```
client = StreamManagerClient()

try:
    client.delete_message_stream(stream_name="StreamName")
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Python SDK 參考：[deleteMessageStream](#)

Java

```
try (final StreamManagerClient client =
    GreengrassClientBuilder.streamManagerClient().build()) {
    client.deleteMessageStream("StreamName");
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Java 開發套件參考：[Delete_message_stream](#)

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        await client.deleteMessageStream("StreamName");
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Node.js SDK 參考：[deleteMessageStream](#)

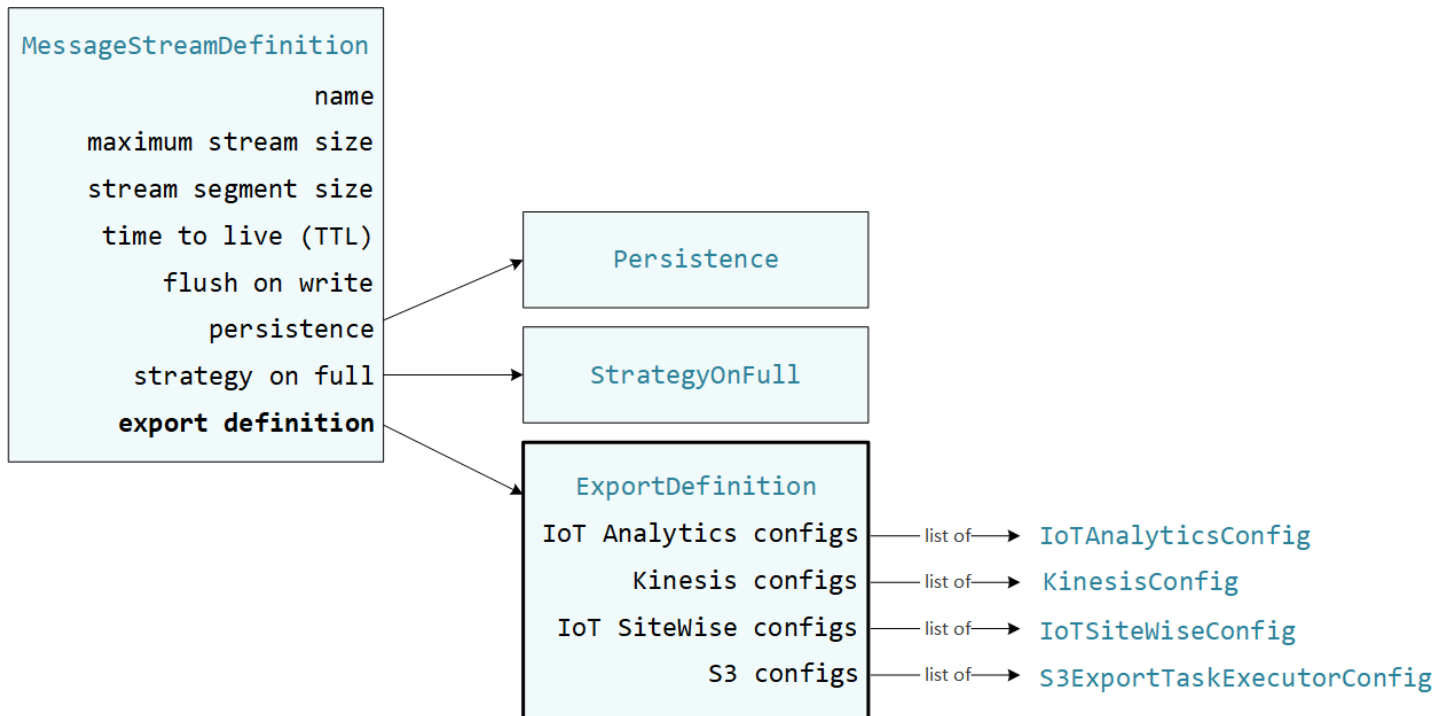
另請參閱

- [管理資料串流](#)
- [the section called “設定串流管理員”](#)
- [the section called “匯出支援 AWS 雲端 目的地的組態”](#)
- [the section called “匯出資料串流 \(主控台\)”](#)
- [the section called “匯出資料串流 \(CLI\)”](#)
- StreamManagerClient AWS IoT Greengrass 核心 SDK 參考中的：
 - [Python](#)
 - [Java](#)

- [Node.js](#)

匯出支援 AWS 雲端 目的地的組態

使用者定義的 Lambda 函數使用 AWS IoT Greengrass 核心 SDK `StreamManagerClient` 與串流管理員互動。當 Lambda [函數建立串流](#)或[更新串流](#)時，它會傳遞代表串流屬性的 `MessageStreamDefinition` 物件，包括匯出定義。 `ExportDefinition` 物件包含為串流定義的匯出組態。串流管理員使用這些匯出組態來判斷匯出串流的位置和方式。



您可以在串流上定義零或多個匯出組態，包括單一目的地類型的多個匯出組態。例如，您可以將串流匯出至兩個 AWS IoT Analytics 頻道和一個 Kinesis 資料串流。

對於失敗的匯出嘗試，串流管理員會持續重試 AWS 雲端，以最多五分鐘的間隔將資料匯出至。重試嘗試次數沒有上限。

Note

`StreamManagerClient` 也提供目標目的地，可用來將串流匯出至 HTTP 伺服器。此目標僅供測試之用。它不穩定或不支援在生產環境中使用。

支援的 AWS 雲端 目的地

- [AWS IoT Analytics 頻道](#)
- [Amazon Kinesis 資料串流](#)
- [AWS IoT SiteWise 資產屬性](#)
- [Amazon S3 物件](#)

您負責維護這些 AWS 雲端 資源。

AWS IoT Analytics 頻道

串流管理員支援自動匯出至 AWS IoT Analytics。AWS IoT Analytics 可讓您對資料執行進階分析，以協助做出商業決策並改善機器學習模型。如需詳細資訊，請參閱AWS IoT Analytics 《使用者指南》中的[什麼是 AWS IoT Analytics ?](#)。

在 AWS IoT Greengrass 核心 SDK 中，您的 Lambda 函數會使用 `IoTAnalyticsConfig` 來定義此目的地類型的匯出組態。如需詳細資訊，請參閱目標語言的 SDK 參考：

- Python SDK 中的 [IoTAnalyticsConfig](#)
- Java 開發套件中的 [IoTAnalyticsConfig](#)
- Node.js SDK 中的 [IoTAnalyticsConfig](#)

要求

此匯出目的地有下列需求：

- 中的目標頻道 AWS IoT Analytics 必須與 Greengrass 群組位於相同 AWS 帳戶 和 AWS 區域。
- 必須[the section called “Greengrass 群組角色”](#)允許將 `iotanalytics:BatchPutMessage` 許可設為目標頻道。例如：

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iotanalytics:BatchPutMessage"
      ],
      "Resource": [
```

```
"arn:aws:iotanalytics:us-east-1:123456789012:channel/channel_1_name",
    "arn:aws:iotanalytics:us-east-1:123456789012:channel/channel_2_name"
  ]
}
```

您可以使用萬用字元*命名機制，授予對資源的精細或條件式存取。如需詳細資訊，請參閱 [《IAM 使用者指南》中的新增和移除 IAM 政策](#)。

匯出至 AWS IoT Analytics

若要建立匯出至的串流 AWS IoT Analytics，您的 Lambda [函數會建立](#) 包含一或多個 IoTAnalyticsConfig 物件的匯出定義串流。此物件會定義匯出設定，例如目標頻道、批次大小、批次間隔和優先順序。

當您的 Lambda 函數從裝置接收資料時，它們會將包含資料 Blob [的訊息附加](#) 到目標串流。

然後，串流管理員會根據串流匯出組態中定義的批次設定和優先順序匯出資料。

Amazon Kinesis 資料串流

串流管理員支援自動匯出至 Amazon Kinesis Data Streams。Kinesis Data Streams 通常用於彙總大量資料，並將其載入資料倉儲或地圖縮減叢集。如需詳細資訊，請參閱 [《Amazon Kinesis 開發人員指南》中的什麼是 Amazon Kinesis Data Streams？](#)。Amazon Kinesis

在 AWS IoT Greengrass 核心 SDK 中，您的 Lambda 函數會使用 KinesisConfig 來定義此目的地類型的匯出組態。如需詳細資訊，請參閱目標語言的 SDK 參考：

- Python SDK 中的 [KinesisConfig](#)
- Java 開發套件中的 [KinesisConfig](#)
- Node.js SDK 中的 [KinesisConfig](#)

要求

此匯出目的地有下列需求：

- Kinesis Data Streams 中的目標串流必須與 Greengrass 群組位於相同 AWS 帳戶和 AWS 區域。

- 必須[the section called “Greengrass 群組角色”](#)允許 以資料串流為目標的kinesis:PutRecords許可。例如：

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kinesis:PutRecords"
      ],
      "Resource": [
        "arn:aws:kinesis:us-east-1:123456789012:stream/stream_1_name",
        "arn:aws:kinesis:us-east-1:123456789012:stream/stream_2_name"
      ]
    }
  ]
}
```

您可以使用萬用字元*命名機制，授予對資源的精細或條件式存取。如需詳細資訊，請參閱《[IAM 使用者指南](#)》中的[新增和移除 IAM 政策](#)。

匯出至 Kinesis Data Streams

若要建立匯出至 Kinesis Data Streams 的串流，您的 Lambda [函數會建立](#) 包含一或多個KinesisConfig物件的匯出定義串流。此物件會定義匯出設定，例如目標資料串流、批次大小、批次間隔和優先順序。

當您的 Lambda 函數從裝置接收資料時，它們會將包含資料 Blob [的訊息附加](#)到目標串流。然後，串流管理員會根據串流匯出組態中定義的批次設定和優先順序匯出資料。

串流管理員會為每個上傳至 Amazon Kinesis 的記錄產生唯一的隨機 UUID 做為分割區索引鍵。

AWS IoT SiteWise 資產屬性

串流管理員支援自動匯出至 AWS IoT SiteWise。AWS IoT SiteWise 可讓您大規模收集、組織和分析工業設備的資料。如需詳細資訊，請參閱AWS IoT SiteWise 《[使用者指南](#)》中的[什麼是 AWS IoT SiteWise ?](#)。

在 AWS IoT Greengrass 核心 SDK 中，您的 Lambda 函數會使用 `IoTSiteWiseConfig` 來定義此目的地的類型的匯出組態。如需詳細資訊，請參閱目標語言的 SDK 參考：

- Python SDK 中的 [IoTSiteWiseConfig](#)
- Java 開發套件中的 [IoTSiteWiseConfig](#)
- Node.js SDK 中的 [IoTSiteWiseConfig](#)

Note

AWS 也提供 [the section called “IoT SiteWise”](#)，這是您可以與 OPC-UA 來源搭配使用的預先建置解決方案。

要求

此匯出目的地有下列需求：

- 中的目標資產屬性 AWS IoT SiteWise 必須與 Greengrass 群組位於相同 AWS 帳戶 和 AWS 區域。

Note

如需 AWS IoT SiteWise 支援的區域清單，請參閱《AWS 一般參考》中的 [AWS IoT SiteWise 端點和配額](#)。

- 必須[the section called “Greengrass 群組角色”](#)允許 以資產屬性為目標的 `iotsitewise:BatchPutAssetPropertyValue` 許可。下列範例政策使用 `iotsitewise:assetHierarchyPath` 條件索引鍵授予對目標根資產及其子資產的存取權。您可以從 `Condition` 政策中移除，以允許存取所有 AWS IoT SiteWise 資產或指定個別資產 ARNs。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iotsitewise:BatchPutAssetPropertyValue",
      "Resource": "*",
      "Condition": {
```

```

    "StringLike": {
      "iotsitewise:assetHierarchyPath": [
        "/root node asset ID",
        "/root node asset ID/*"
      ]
    }
  }
}

```

您可以使用萬用字元*命名機制，授予對資源的精細或條件式存取。如需詳細資訊，請參閱 [《IAM 使用者指南》](#) 中的 [新增和移除 IAM 政策](#)。

如需重要的安全性資訊，請參閱AWS IoT SiteWise 《使用者指南》中的 [BatchPutAssetPropertyValue 授權](#)。

匯出至 AWS IoT SiteWise

若要建立匯出至 的串流 AWS IoT SiteWise，您的 Lambda [函數會建立](#) 包含一或多個IoTSiteWiseConfig物件的匯出定義串流。此物件會定義匯出設定，例如批次大小、批次間隔和優先順序。

當您的 Lambda 函數從裝置接收資產屬性資料時，它們會將包含資料的訊息附加到目標串流。訊息是JSON 序列化PutAssetPropertyValueEntry物件，其中包含一或多個資產屬性的屬性值。如需詳細資訊，請參閱[附加訊息](#)以取得 AWS IoT SiteWise 匯出目的地。

Note

當您傳送資料至 時 AWS IoT SiteWise，您的資料必須符合 BatchPutAssetPropertyValue動作的要求。如需詳細資訊，請參閱《AWS IoT SiteWise API 參考》中的 [BatchPutAssetPropertyValue](#)。

然後，串流管理員會根據串流匯出組態中定義的批次設定和優先順序匯出資料。

您可以調整串流管理員設定和 Lambda 函數邏輯，以設計匯出策略。例如：

- 對於近乎即時的匯出，請設定低批次大小和間隔設定，並在接收資料時將資料附加至串流。

- 為了最佳化批次處理、降低頻寬限制或將成本降至最低，您的 Lambda 函數可以在將資料附加至串流之前，合併單一資產屬性收到的timestamp-quality-value(TQV) 資料點。一種策略是在一個訊息中批次處理最多 10 個不同的屬性資產組合或屬性別名的項目，而不是為相同的屬性傳送多個項目。這有助於串流管理員保持在[AWS IoT SiteWise 配額](#)內。

Amazon S3 物件

串流管理員支援自動匯出至 Amazon S3。您可以使用 Amazon S3 來存放和擷取大量資料。如需詳細資訊，請參閱 [《Amazon Simple Storage Service 開發人員指南》中的什麼是 Amazon S3？](#)。

在 AWS IoT Greengrass 核心 SDK 中，您的 Lambda 函數會使用 `S3ExportTaskExecutorConfig` 來定義此目的地類型的匯出組態。如需詳細資訊，請參閱目標語言的 SDK 參考：

- Python SDK 中的 [S3ExportTaskExecutorConfig](#)
- Java 開發套件中的 [S3ExportTaskExecutorConfig](#)
- Node.js SDK 中的 [S3ExportTaskExecutorConfig](#)

要求

此匯出目的地有下列需求：

- 目標 Amazon S3 儲存貯體必須與 Greengrass 群組 AWS 帳戶 位於相同的 中。
- 如果 Greengrass 群組的[預設容器化](#)是 Greengrass 容器，您必須將 [STREAM_MANAGER_READ_ONLY_DIRS](#) 參數設定為使用根檔案系統下/tmp或不在其中的輸入檔案目錄。
- 如果在 Greengrass 容器模式下執行的 Lambda 函數將輸入檔案寫入輸入檔案目錄，您必須為目錄建立本機磁碟區資源，並將目錄掛載到具有寫入許可的容器。這可確保檔案寫入根檔案系統，並在容器外部可見。如需詳細資訊，請參閱[存取本機資源](#)。
- 必須[the section called “Greengrass 群組角色”](#)允許目標儲存貯體的下列許可。例如：

JSON

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

```
        "Effect": "Allow",
        "Action": [
            "s3:PutObject",
            "s3:AbortMultipartUpload",
            "s3:ListMultipartUploadParts"
        ],
        "Resource": [
            "arn:aws:s3:::bucket-1-name/*",
            "arn:aws:s3:::bucket-2-name/*"
        ]
    }
}
```

您可以使用萬用字元*命名機制，授予對資源的精細或條件式存取。如需詳細資訊，請參閱 [《IAM 使用者指南》中的新增和移除 IAM 政策](#)。

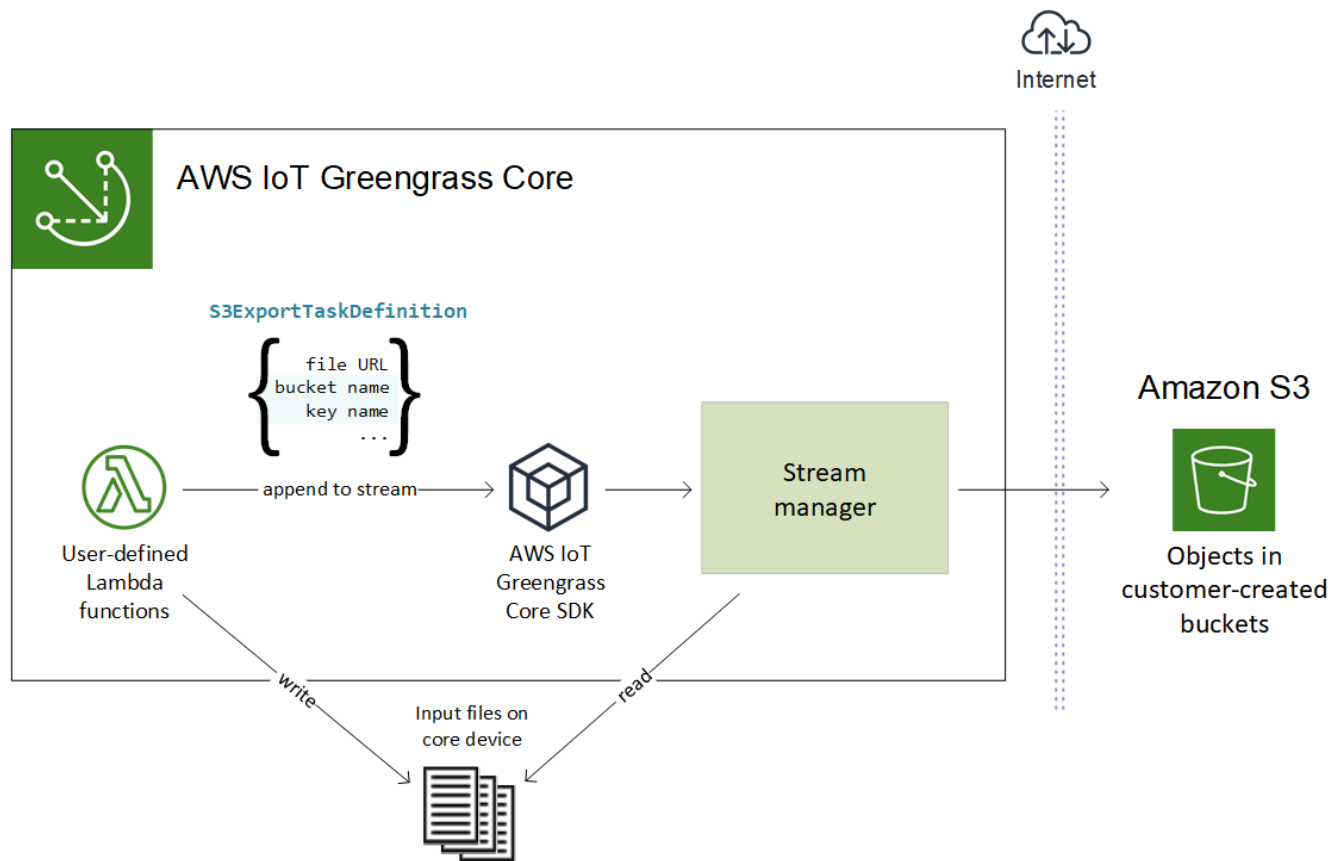
匯出至 Amazon S3

若要建立匯出至 Amazon S3 的串流，您的 Lambda 函數會使用 `S3ExportTaskExecutorConfig` 物件來設定匯出政策。政策會定義匯出設定，例如分段上傳閾值和優先順序。對於 Amazon S3 匯出，串流管理員會上傳從核心裝置上的本機檔案讀取的資料。若要啟動上傳，您的 Lambda 函數會將匯出任務附加至目標串流。匯出任務包含輸入檔案和目標 Amazon S3 物件的相關資訊。串流管理員會依照任務附加至串流的順序來執行任務。

Note

目標儲存貯體必須已存在於您的 AWS 帳戶。如果指定金鑰的物件不存在，串流管理員會為您建立物件。

此高階工作流程如下圖所示。



串流管理員會使用分段上傳閾值屬性、[最小組件大小](#)設定，以及輸入檔案的大小來判斷如何上傳資料。分段上傳閾值必須大於或等於最小分段大小。如果您想要平行上傳資料，您可以建立多個串流。

指定目標 Amazon S3 物件的金鑰可以在 `!{timestamp: value}` 預留位置中包含有效的 [Java DateTimeFormatter](#) 字串。您可以根據輸入檔案資料上傳的時間，使用這些時間戳記預留位置來分割 Amazon S3 中的資料。例如，下列金鑰名稱會解析為 等值 `my-key/2020/12/31/data.txt`。

```
my-key/!{timestamp:YYYY}/!{timestamp:MM}/!{timestamp:dd}/data.txt
```

Note

如果您想要監控串流的匯出狀態，請先建立狀態串流，然後將匯出串流設定為使用它。如需詳細資訊，請參閱[the section called “監控匯出任務”](#)。

管理輸入資料

您可以編寫程式碼，讓 IoT 應用程式用來管理輸入資料的生命週期。下列工作流程範例示範如何使用 Lambda 函數來管理此資料。

1. 本機程序會從裝置或周邊接收資料，然後將資料寫入核心裝置上的目錄中的檔案。這些是串流管理員的輸入檔案。

Note

若要判斷您是否必須設定輸入檔案目錄的存取權，請參閱

[STREAM_MANAGER_READ_ONLY_DIRS](#) 參數。

串流管理員在 中執行的程序會繼承群組[預設存取身分](#)的所有檔案系統許可。串流管理員必須具有存取輸入檔案的許可。如有必要，您可以使用 `chmod(1)` 命令來變更檔案的許可。

2. Lambda 函數會掃描目錄，並在[建立新檔案時將匯出任務附加](#)至目標串流。任務是 JSON 序列化 `S3ExportTaskDefinition` 物件，可指定輸入檔案的 URL、目標 Amazon S3 儲存貯體和金鑰，以及選用的使用者中繼資料。
3. 串流管理員會讀取輸入檔案，並依附加任務的順序將資料匯出至 Amazon S3。目標儲存貯體必須已存在於您的 中 AWS 帳戶。如果指定金鑰的物件不存在，串流管理員會為您建立物件。
4. Lambda 函數會從狀態串流[讀取訊息](#)，以監控匯出狀態。匯出任務完成後，Lambda 函數可以刪除對應的輸入檔案。如需詳細資訊，請參閱[the section called “監控匯出任務”](#)。

監控匯出任務

您可以編寫程式碼，讓 IoT 應用程式用來監控 Amazon S3 匯出的狀態。您的 Lambda 函數必須建立狀態串流，然後將匯出串流設定為將狀態更新寫入狀態串流。單一狀態串流可以從匯出至 Amazon S3 的多個串流接收狀態更新。

首先，[建立](#)要用作狀態串流的串流。您可以設定串流的大小和保留政策，以控制狀態訊息的生命週期。例如：

- Memory 如果您不想儲存狀態訊息，請將 Persistence 設定為。
- StrategyOnFull 設定為 `OverwriteOldestData` 以免遺失新的狀態訊息。

然後，建立或更新匯出串流以使用狀態串流。具體而言，請設定串流 `S3ExportTaskExecutorConfig` 匯出組態的狀態組態屬性。這會通知串流管理員將有關匯出任務的狀態訊息寫入狀態串流。在 `StatusConfig` 物件中，指定狀態串流的名稱和詳細程度。下列支援的值範圍從最詳細 (ERROR) 到最詳細 (TRACE)。預設值為 INFO。

- ERROR
- WARN

- INFO
- DEBUG
- TRACE

下列範例工作流程顯示 Lambda 函數如何使用狀態串流來監控匯出狀態。

1. 如前一個工作流程所述，Lambda 函數[會將匯出任務附加](#)至設定為將有關匯出任務的狀態訊息寫入狀態串流的串流。附加操作會傳回代表任務 ID 的序號。
2. Lambda 函數會從狀態串流循序[讀取訊息](#)，然後根據串流名稱和任務 ID 或根據訊息內容的匯出任務屬性來篩選訊息。例如，Lambda 函數可以依匯出任務的輸入檔案 URL 進行篩選，其由訊息內容中的 `S3ExportTaskDefinition` 物件表示。

下列狀態碼表示匯出任務已達到已完成狀態：

- Success。上傳已成功完成。
- Failure。串流管理員發生錯誤，例如，指定的儲存貯體不存在。解決問題後，您可以再次將匯出任務附加到串流。
- Canceled。任務因為串流或匯出定義已刪除，或任務的time-to-live(TTL) 期間過期而中止。

Note

任務也可能的狀態為 `InProgress` 或 `Warning`。當事件傳回不會影響任務執行的錯誤時，串流管理員會發出警告。例如，無法清除中止的部分上傳會傳回警告。

3. 匯出任務完成後，Lambda 函數可以刪除對應的輸入檔案。

下列範例顯示 Lambda 函數如何讀取和處理狀態訊息。

Python

```
import time
from greengrasssdk.stream_manager import (
    ReadMessagesOptions,
    Status,
    StatusConfig,
    StatusLevel,
    StatusMessage,
```

```

    StreamManagerClient,
)
from greengrasssdk.stream_manager.util import Util

client = StreamManagerClient()

try:
    # Read the statuses from the export status stream
    is_file_uploaded_to_s3 = False
    while not is_file_uploaded_to_s3:
        try:
            messages_list = client.read_messages(
                "StatusStreamName", ReadMessagesOptions(min_message_count=1,
read_timeout_millis=1000)
            )
            for message in messages_list:
                # Deserialize the status message first.
                status_message = Util.deserialize_json_bytes_to_obj(message.payload,
StatusMessage)

                # Check the status of the status message. If the status is
"Success",
                # the file was successfully uploaded to S3.
                # If the status was either "Failure" or "Cancelled", the server was
unable to upload the file to S3.
                # We will print the message for why the upload to S3 failed from the
status message.
                # If the status was "InProgress", the status indicates that the
server has started uploading
                # the S3 task.
                if status_message.status == Status.Success:
                    logger.info("Successfully uploaded file at path " + file_url + "
to S3.")
                    is_file_uploaded_to_s3 = True
                elif status_message.status == Status.Failure or
status_message.status == Status.Canceled:
                    logger.info(
                        "Unable to upload file at path " + file_url + " to S3.
Message: " + status_message.message
                    )
                    is_file_uploaded_to_s3 = True
                time.sleep(5)
        except StreamManagerException:
            logger.exception("Exception while running")

```

```
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Python SDK 參考：[read_messages](#) | [StatusMessage](#)

Java

```
import com.amazonaws.greengrass.streammanager.client.StreamManagerClient;
import com.amazonaws.greengrass.streammanager.client.utils.ValidateAndSerialize;
import com.amazonaws.greengrass.streammanager.model.ReadMessagesOptions;
import com.amazonaws.greengrass.streammanager.model.Status;
import com.amazonaws.greengrass.streammanager.model.StatusConfig;
import com.amazonaws.greengrass.streammanager.model.StatusLevel;
import com.amazonaws.greengrass.streammanager.model.StatusMessage;

try (final StreamManagerClient client =
    GreengrassClientBuilder.streamManagerClient().build()) {
    try {
        boolean isS3UploadComplete = false;
        while (!isS3UploadComplete) {
            try {
                // Read the statuses from the export status stream
                List<Message> messages = client.readMessages("StatusStreamName",
                    new
                ReadMessagesOptions().withMinMessageCount(1L).withReadTimeoutMillis(1000L));
                for (Message message : messages) {
                    // Deserialize the status message first.
                    StatusMessage statusMessage =
                ValidateAndSerialize.deserializeJsonBytesToObj(message.getPayload(),
                StatusMessage.class);
                    // Check the status of the status message. If the status is
                "Success", the file was successfully uploaded to S3.
                    // If the status was either "Failure" or "Canceled", the server
                was unable to upload the file to S3.
                    // We will print the message for why the upload to S3 failed
                from the status message.
                    // If the status was "InProgress", the status indicates that the
                server has started uploading the S3 task.
                    if (Status.Success.equals(statusMessage.getStatus())) {
```

```

        System.out.println("Successfully uploaded file at path " +
FILE_URL + " to S3.");
        isS3UploadComplete = true;
    } else if (Status.Failure.equals(statusMessage.getStatus()) ||
Status.Canceled.equals(statusMessage.getStatus())) {
        System.out.println(String.format("Unable to upload file at
path %s to S3. Message %s",
statusMessage.getStatusContext().getS3ExportTaskDefinition().getInputUrl(),
        statusMessage.getMessage()));
        sS3UploadComplete = true;
    }
}
} catch (StreamManagerException ignored) {
} finally {
    // Sleep for sometime for the S3 upload task to complete before
trying to read the status message.
    Thread.sleep(5000);
}
} catch (e) {
    // Properly handle errors.
}
} catch (StreamManagerException e) {
    // Properly handle exception.
}
}

```

Java 開發套件參考：[readMessages](#) | [StatusMessage](#)

Node.js

```

const {
    StreamManagerClient, ReadMessagesOptions,
    Status, StatusConfig, StatusLevel, StatusMessage,
    util,
} = require('aws-greengrass-core-sdk').StreamManager;

const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        let isS3UploadComplete = false;
        while (!isS3UploadComplete) {
            try {
                // Read the statuses from the export status stream
                const messages = await c.readMessages("StatusStreamName",

```

```

        new ReadMessagesOptions()
            .withMinMessageCount(1)
            .withReadTimeoutMillis(1000));

    messages.forEach((message) => {
        // Deserialize the status message first.
        const statusMessage =
util.deserializeJsonBytesToObj(message.payload, StatusMessage);
        // Check the status of the status message. If the status is
'Success', the file was successfully uploaded to S3.
        // If the status was either 'Failure' or 'Cancelled', the server
was unable to upload the file to S3.
        // We will print the message for why the upload to S3 failed
from the status message.
        // If the status was "InProgress", the status indicates that the
server has started uploading the S3 task.
        if (statusMessage.status === Status.Success) {
            console.log(`Successfully uploaded file at path ${FILE_URL}
to S3.`);

            isS3UploadComplete = true;
        } else if (statusMessage.status === Status.Failure ||
statusMessage.status === Status.Canceled) {
            console.log(`Unable to upload file at path ${FILE_URL} to
S3. Message: ${statusMessage.message}`);
            isS3UploadComplete = true;
        }
    });
    // Sleep for sometime for the S3 upload task to complete before
trying to read the status message.
    await new Promise((r) => setTimeout(r, 5000));
    } catch (e) {
        // Ignored
    }
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});

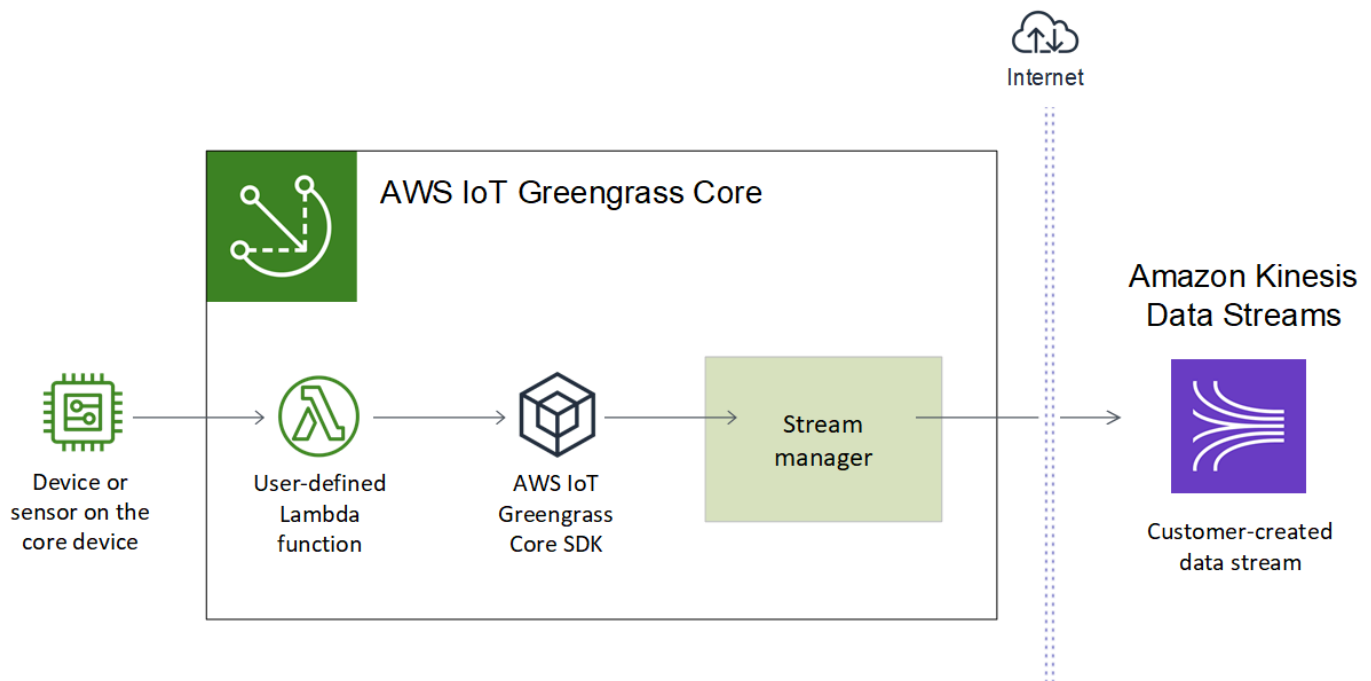
```

Node.js SDK 參考：[readMessages](#) | [StatusMessage](#)

將資料串流匯出至 AWS 雲端（主控台）

本教學課程說明如何使用 AWS IoT 主控台來設定和部署已啟用串流管理員的 AWS IoT Greengrass 群組。群組包含使用者定義的 Lambda 函數，可寫入串流管理員中的串流，然後自動匯出至 AWS 雲端。

串流管理員可讓擷取、處理和匯出大量資料串流更有效率且更可靠。在本教學課程中，您會建立使用 IoT 資料的 TransferStream Lambda 函數。Lambda 函數使用 AWS IoT Greengrass 核心 SDK 在串流管理員中建立串流，然後讀取和寫入該串流。串流管理員接著會將串流匯出至 Kinesis Data Streams。下圖顯示此工作流程。



本教學課程的重點是展示使用者定義的 Lambda 函數如何使用 AWS IoT Greengrass Core SDK 中的 `StreamManagerClient` 物件與串流管理員互動。為了簡化，您為此教學課程建立的 Python Lambda 函數會產生模擬的裝置資料。

先決條件

為完成此教學課程您需要：

- Greengrass 群組和 Greengrass 核心 (1.10 版或更新版本)。如需如何建立 Greengrass 群組和核心的詳細資訊，請參閱 [入門 AWS IoT Greengrass](#)。入門教學課程也包含安裝 AWS IoT Greengrass 核心軟體的步驟。

Note

OpenWrt 發行版本不支援串流管理員。

- 在核心裝置上安裝 Java 8 執行時間 (JDK 8)。
- 針對以 Debian 為基礎的發行版本 (包括 Raspbian) 或以 Ubuntu 為基礎的發行版本，請執行下列命令：

```
sudo apt install openjdk-8-jdk
```

- 針對以 Red Hat 為基礎的發行版本 (包括 Amazon Linux)，請執行下列命令：

```
sudo yum install java-1.8.0-openjdk
```

如需詳細資訊，請參閱 OpenJDK 文件上的[如何下載和安裝預先建置的 OpenJDK 套件](#)。

- AWS IoT Greengrass 適用於 Python 1.5.0 版或更新版本的核心 SDK。若要在適用於 Python AWS IoT Greengrass 的核心 SDK `StreamManagerClient` 中使用，您必須：
 - 在核心裝置上安裝 Python 3.7 或更新版本。
 - 在 Lambda 函數部署套件中包含 SDK 及其相依性。本教學課程提供相關指示。

Tip

您可以搭配使用 `StreamManagerClient` 與 Java 或 NodeJS。如需程式碼範例，請參閱 GitHub 上的[AWS IoT Greengrass 適用於 Java 的 Core SDK](#) 和[AWS IoT Greengrass 適用於 Node.js 的 Core SDK](#)。

- 在與 Greengrass 群組相同的 Amazon Kinesis Data Streams 中 **MyKinesisStream** 建立名為 AWS 區域 的目的地串流。如需詳細資訊，請參閱《Amazon Kinesis 開發人員指南》中的[建立串流](#)。

Note

在本教學課程中，串流管理員會將資料匯出至 Kinesis Data Streams，這會向您的收取費用 AWS 帳戶。如需定價的相關資訊，請參閱 [Kinesis Data Streams 定價](#)。若要避免產生費用，您可以執行本教學課程而不建立 Kinesis 資料串流。在此情況下，您會檢查日誌，以查看串流管理員嘗試將串流匯出至 Kinesis Data Streams。

- 新增至的 IAM 政策[the section called “Greengrass 群組角色”](#)，允許對目標資料串流執行 `kinesis:PutRecords` 動作，如下列範例所示：

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kinesis:PutRecords"
      ],
      "Resource": [
        "arn:aws:kinesis:us-east-1:123456789012:stream/MyKinesisStream"
      ]
    }
  ]
}
```

本教學課程所述以下高階執行步驟：

1. [建立 Lambda 函數部署套件](#)
2. [建立 Lambda 函式](#)
3. [將函數新增到群組](#)
4. [啟用串流管理員](#)
5. [設定本機記錄](#)
6. [部署群組](#)
7. [測試應用程式。](#)

此教學課程需約 20 分鐘完成。

步驟 1：建立 Lambda 函數部署套件

在此步驟中，您會建立包含 Python 函數程式碼和相依性的 Lambda 函數部署套件。您稍後在其中建立 Lambda 函數時上傳此套件 AWS Lambda。Lambda 函數使用 AWS IoT Greengrass 核心 SDK 來建立本機串流並與之互動。

Note

使用者定義的 Lambda 函數必須使用 [AWS IoT Greengrass 核心 SDK](#) 與串流管理員互動。如需有關 Greengrass 串流管理員需求的詳細資訊，請參閱 [Greengrass 串流管理員需求](#)。

1. 下載[AWS IoT Greengrass 適用於 Python 的核心 SDK](#) 1.5.0 版或更新版本。
2. 解壓縮下載的封裝，以取得軟體開發套件。SDK 為 greengrasssdk 資料夾。
3. 安裝套件相依性，以在 Lambda 函數部署套件中包含 開發套件。
 1. 前往包含 requirements.txt 檔案的軟體開發套件目錄。這個檔案會列出相依性。
 2. 安裝軟體開發套件相依性。例如，執行下列 pip 命令，將它們安裝在目前的目錄中：

```
pip install --target . -r requirements.txt
```

4. 將以下 Python 程式碼函數儲存在名為 transfer_stream.py 的本機檔案中。

Tip

如需使用 Java 和 NodeJS 的範例程式碼，請參閱 GitHub 上的[AWS IoT Greengrass 適用於 Java 的核心開發套件](#)和[AWS IoT Greengrass 適用於 Node.js 的核心開發套件](#)。

```
import asyncio
import logging
import random
import time

from greengrasssdk.stream_manager import (
    ExportDefinition,
    KinesisConfig,
    MessageStreamDefinition,
    ReadMessagesOptions,
    ResourceNotFoundException,
    StrategyOnFull,
    StreamManagerClient,
)

# This example creates a local stream named "SomeStream".
```

```
# It starts writing data into that stream and then stream manager automatically
# exports
# the data to a customer-created Kinesis data stream named "MyKinesisStream".
# This example runs forever until the program is stopped.

# The size of the local stream on disk will not exceed the default (which is 256
# MB).
# Any data appended after the stream reaches the size limit continues to be
# appended, and
# stream manager deletes the oldest data until the total stream size is back under
# 256 MB.
# The Kinesis data stream in the cloud has no such bound, so all the data from this
# script is
# uploaded to Kinesis and you will be charged for that usage.

def main(logger):
    try:
        stream_name = "SomeStream"
        kinesis_stream_name = "MyKinesisStream"

        # Create a client for the StreamManager
        client = StreamManagerClient()

        # Try deleting the stream (if it exists) so that we have a fresh start
        try:
            client.delete_message_stream(stream_name=stream_name)
        except ResourceNotFoundException:
            pass

        exports = ExportDefinition(
            kinesis=[KinesisConfig(identifier="KinesisExport" + stream_name,
            kinesis_stream_name=kinesis_stream_name)]
        )
        client.create_message_stream(
            MessageStreamDefinition(
                name=stream_name,
                strategy_on_full=StrategyOnFull.OverwriteOldestData, export_definition=exports
            )
        )

        # Append two messages and print their sequence numbers
        logger.info(
            "Successfully appended message to stream with sequence number %d",
```

```

        client.append_message(stream_name, "ABCDEFGHIIJKLMNO".encode("utf-8")),
    )
    logger.info(
        "Successfully appended message to stream with sequence number %d",
        client.append_message(stream_name, "PQRSTUVWXYZ".encode("utf-8")),
    )

    # Try reading the two messages we just appended and print them out
    logger.info(
        "Successfully read 2 messages: %s",
        client.read_messages(stream_name,
ReadMessagesOptions(min_message_count=2, read_timeout_millis=1000)),
    )

    logger.info("Now going to start writing random integers between 0 and 1000
to the stream")
    # Now start putting in random data between 0 and 1000 to emulate device
sensor input
    while True:
        logger.debug("Appending new random integer to stream")
        client.append_message(stream_name, random.randint(0,
1000).to_bytes(length=4, signed=True, byteorder="big"))
        time.sleep(1)

    except asyncio.TimeoutError:
        logger.exception("Timed out while executing")
    except Exception:
        logger.exception("Exception while running")

def function_handler(event, context):
    return

logging.basicConfig(level=logging.INFO)
# Start up this sample code
main(logger=logging.getLogger())

```

5. 將下列項目壓縮成名為 `transfer_stream_python.zip` 的檔案。這是您的 Lambda 函數部署套件。

- `transfer_stream.py`。應用程式邏輯。
- `greengrasssdk`。發佈 MQTT 訊息的 Python Greengrass Lambda 函數所需的程式庫。

[串流管理員操作](#)可在適用於 Python 的核心 AWS IoT Greengrass SDK 1.5.0 版或更新版本中使用。

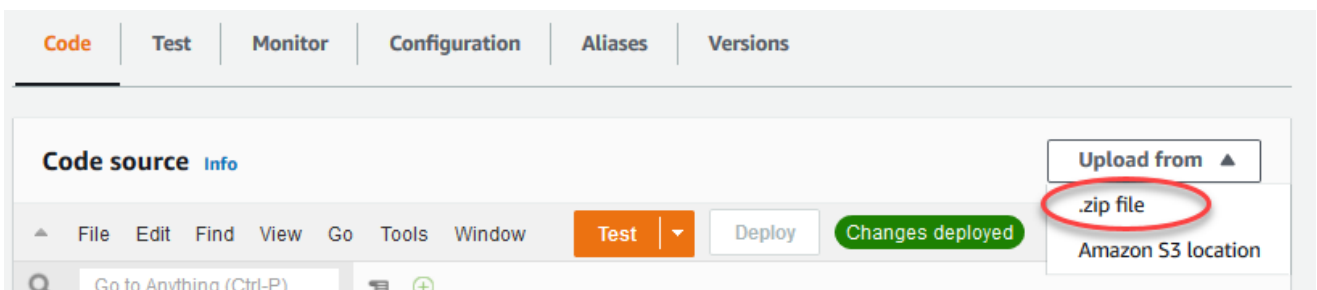
- 您為適用於 Python AWS IoT Greengrass 的核心 SDK 安裝的相依性（例如，cbor2目錄）。

當您建立 zip 檔案時，請只包含這些項目，而非包含的資料夾。

步驟 2：建立 Lambda 函數

在此步驟中，您會使用 AWS Lambda 主控台來建立 Lambda 函數，並將其設定為使用您的部署套件。然後，您會發佈函數版本和建立別名。

1. 首先，建立 Lambda 函數。
 - a. 在 AWS 管理主控台，選擇服務，然後開啟 AWS Lambda 主控台。
 - b. 選擇建立函數，然後選擇從頭開始撰寫。
 - c. 在 Basic information (基本資訊) 區段中，使用下列值：
 - 針對函數名稱，請輸入 **TransferStream**。
 - 針對執行期，選擇 Python 3.7。
 - 對於許可，請保留預設設定。這會建立授予基本 Lambda 許可的執行角色。不會使用此角色 AWS IoT Greengrass。
 - d. 請在頁面底部，選擇建立函式。
2. 接著，註冊處理常式並上傳您的 Lambda 函數部署套件。
 - a. 在程式碼索引標籤的程式碼來源下，選擇上傳來源。從下拉式清單中，選擇 .zip 檔案。



- b. 選擇上傳，然後選擇您的transfer_stream_python.zip部署套件。然後選擇 Save (儲存)。
- c. 在函數的程式碼索引標籤的執行時間設定下，選擇編輯，然後輸入下列值。

- 針對執行期，選擇 Python 3.7。
 - 對於 Handler (處理常式)，輸入 `transfer_stream.function_handler`。
- d. 選擇儲存。

 Note

AWS Lambda 主控台上的測試按鈕不適用於此函數。AWS IoT Greengrass 核心 SDK 不包含在 AWS Lambda 主控台中獨立執行 Greengrass Lambda 函數所需的模組。這些模組 (例如 `greengrass_common`) 會在部署到您的 Greengrass 核心之後提供給函數。

3. 現在，發佈 Lambda 函數的第一個版本，並為[該版本建立別名](#)。

 Note

Greengrass 群組可以依別名 (建議) 或版本參考 Lambda 函數。使用別名可讓您更輕鬆地管理程式碼更新，因為您不必在更新函數程式碼時變更訂閱資料表或群組定義。反之，您只需將別名指向新的函數版本。

- a. 請從操作功能表中選擇發行新版本。
- b. 針對 Version description (版本描述)，輸入 **First version**，然後選擇 Publish (發佈)。
- c. 從 TransferStream: 1 組態頁面上的 Actions (動作) 選單中選擇 Create alias (建立別名)。
- d. 在建立警示頁面上使用下列值：
 - 對於名稱，輸入 **GG_TransferStream**。
 - 對於版本，選擇 1。

 Note

AWS IoT Greengrass 不支援 \$LATEST 版本的 Lambda 別名。

- e. 選擇建立。

現在您已準備好將 Lambda 函數新增至 Greengrass 群組。

步驟 3：將 Lambda 函數新增至 Greengrass 群組

在此步驟中，您將 Lambda 函數新增至 群組，然後設定其生命週期和環境變數。如需詳細資訊，請參閱[the section called “控制 Greengrass Lambda 函數執行”](#)。

1. 在 AWS IoT 主控台導覽窗格的管理下，展開 Greengrass 裝置，然後選擇群組 (V1)。
2. 選擇目標群組。
3. 在群組組態頁面上，選擇 Lambda 函數索引標籤。
4. 在我的 Lambda 函數下，選擇新增。
5. 在新增 Lambda 函數頁面上，為您的 Lambda 函數選擇 Lambda 函數。
6. 針對 Lambda 版本，選擇別名：GG_TransferStream。

現在，設定屬性來決定 Greengrass 群組中 Lambda 函數的行為。

7. 在 Lambda 函數組態區段中，進行下列變更：
 - 將 Memory limit (記憶體限制) 設為 32 MB。
 - 針對固定，選擇 True。

Note

長期（或固定）Lambda 函數會在啟動後自動 AWS IoT Greengrass 啟動，並在自己的容器中繼續執行。這與隨需 Lambda 函數相反，該函數會在叫用時啟動，並在沒有剩餘的任務執行時停止。如需詳細資訊，請參閱[the section called “生命週期組態”](#)。

8. 選擇新增 Lambda 函數。

步驟 4：啟用串流管理員

在此步驟中，您將確定串流管理員已啟用。

1. 在群組組態頁面上，選擇 Lambda 函數索引標籤。
2. 在 System Lambda 函數下，選取串流管理員，然後檢查狀態。如果是已停用，請選擇 Edit (編輯)。然後，選擇 Enable (啟用) 並 Save (儲存)。您可以使用本教學課程的預設參數設定。如需詳細資訊，請參閱[the section called “設定串流管理員”](#)。

Note

當您使用主控台啟用串流管理員並部署群組時，串流管理員的記憶體大小預設為 4194304 KB (4 GB)。我們建議您將記憶體大小設定為至少 128000 KB。

步驟 5：設定本機記錄

在此步驟中，您可以設定群組中的 AWS IoT Greengrass 系統元件、使用者定義的 Lambda 函數和連接器，將日誌寫入核心裝置的檔案系統。您可以使用記錄檔針對可能遇到的任何問題進行疑難排解。如需詳細資訊，請參閱[the section called “使用 AWS IoT Greengrass 日誌進行監控”](#)。

1. 在 Local logs configuration (本機日誌組態) 下，檢查是否已設定本機記錄。
2. 如果未針對 Greengrass 系統元件或使用者定義的 Lambda 函數設定日誌，請選擇編輯。
3. 選擇使用者 Lambda 函數日誌層級和 Greengrass 系統日誌層級。
4. 保留記錄層級和磁碟空間限制的預設值，然後選擇 Save (儲存)。

步驟 6：部署 Greengrass 群組

將群組部署到核心裝置。

1. 確定 AWS IoT Greengrass 核心正在執行。如果需要，請在您的 Raspberry Pi 終端機執行以下命令。
 - a. 檢查精靈是否有在運作：

```
ps aux | grep -E 'greengrass.*daemon'
```

若輸出的 root 含有 /greengrass/ggc/packages/*ggc-version*/bin/daemon 項目，則精靈有在運作。

Note

路徑中的版本取決於核心裝置上安裝的核心 AWS IoT Greengrass 軟體版本。

- b. 若要啟動協助程式：

```
cd /greengrass/ggc/core/
```



```
sudo ./greengrassd start
```

2. 在群組組態頁面上，選擇部署。
3.
 - a. 在 Lambda 函數索引標籤的系統 Lambda 函數區段下，選取 IP 偵測器，然後選擇編輯。
 - b. 在編輯 IP 偵測器設定對話方塊中，選取自動偵測和覆寫 MQTT 代理程式端點。
 - c. 選擇儲存。

這可讓裝置自動取得核心的連接資訊，例如 IP 位址、DNS、連接埠編號。建議自動偵測，但 AWS IoT Greengrass 也支援手動指定的端點。只會在第一次部署群組時收到復原方法的提示。

Note

如果出現提示，請授予許可來建立 [Greengrass 服務角色](#)，並將其與目前 AWS 帳戶中的 建立關聯 AWS 區域。此角色允許 AWS IoT Greengrass 存取 AWS 服務中的資源。

此部署頁面會顯示部署時間戳記、版本 ID 和狀態。完成後，部署顯示的狀態應為已完成。

如需故障診斷協助，請參閱[疑難排解](#)。

步驟 7：測試應用程式

TransferStream Lambda 函數會產生模擬的裝置資料。它會將資料寫入串流管理員匯出至目標 Kinesis 資料串流的串流。

1. 在 Amazon Kinesis 主控台的 Kinesis 資料串流下，選擇 MyKinesisStream。

Note

如果您在沒有目標 Kinesis 資料串流的情況下執行教學課程，[請檢查串流管理員的日誌檔案](#) (GGStreamManager)。如果日誌檔案包含錯誤消息中的 `export stream MyKinesisStream doesn't exist`，則測試成功。此錯誤表示服務嘗試匯出至串流，但串流不存在。

2. 在 MyKinesisStream 頁面上，選擇 Monitoring (監控)。如果測試成功，您應該會看到 Put Records (Put 記錄) 圖表中的資料。視您的連線而定，可能需要一分鐘才會顯示資料。

⚠ Important

完成測試後，請刪除 Kinesis 資料串流以免產生更多費用。
或者，執行下列命令來停止 Greengrass 協助程式。這樣可以防止核心傳送訊息，直到您準備好繼續測試為止。

```
cd /greengrass/ggc/core/  
sudo ./greengrassd stop
```

3. 從核心移除 TransferStream Lambda 函數。

- a. 在 AWS IoT 主控台導覽窗格的管理下，展開 Greengrass 裝置，然後選擇群組 (V1)。
- b. 在 Greengrass 群組下，選擇您的群組。
- c. 在 Lambdas 頁面上，選擇 TransferStream 函數的省略符號 (...)，然後選擇移除函數。
- d. 從 Actions (動作) 中，選擇 Deploy (部署)。

若要檢視記錄資訊或針對串流問題進行疑難排解，請檢查 TransferStream 和 GGStreamManager 函數的記錄。您必須具有讀取檔案系統 AWS IoT Greengrass 日誌的 root 許可。

- TransferStream 會將日誌項目寫入 *greengrass-root*/ggc/var/log/user/*region*/*account-id*/TransferStream.log。
- GGStreamManager 會將日誌項目寫入 *greengrass-root*/ggc/var/log/system/GGStreamManager.log。

如果您需要更多疑難排解資訊，您可以將 User Lambda 日誌的記錄[層級設定為](#)偵錯日誌，然後再次部署群組。

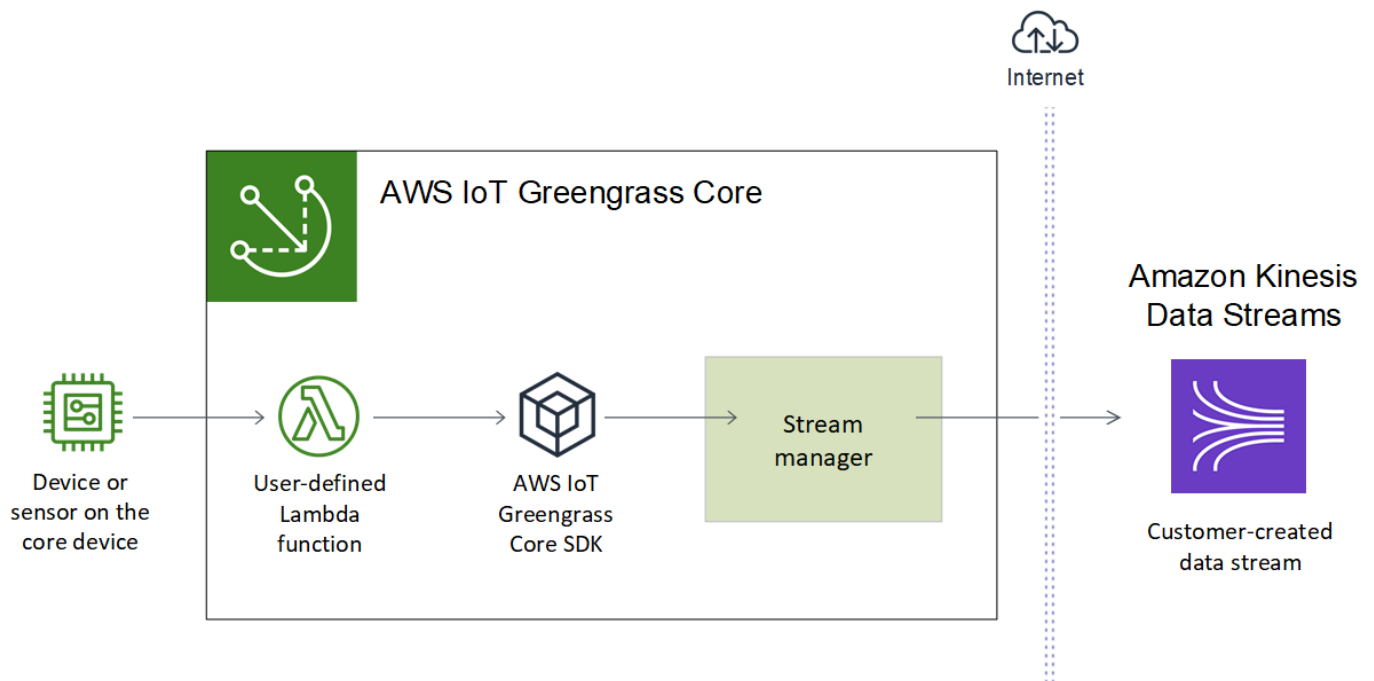
另請參閱

- [管理資料串流](#)
- [the section called “設定串流管理員”](#)
- [the section called “使用 StreamManagerClient 搭配串流”](#)
- [the section called “匯出支援 AWS 雲端 目的地的組態”](#)
- [the section called “匯出資料串流 \(CLI\)”](#)

將資料串流匯出至 AWS 雲端 (CLI)

本教學課程說明如何使用 AWS CLI 來設定和部署已啟用串流管理員的 AWS IoT Greengrass 群組。群組包含使用者定義的 Lambda 函數，可寫入串流管理員中的串流，然後自動匯出至 AWS 雲端。

串流管理員可讓擷取、處理和匯出大量資料串流更有效率且更可靠。在本教學課程中，您會建立使用 IoT 資料的 TransferStream Lambda 函數。Lambda 函數使用 AWS IoT Greengrass Core SDK 在串流管理員中建立串流，然後讀取和寫入。串流管理員接著會將串流匯出至 Kinesis Data Streams。下圖顯示此工作流程。



本教學課程的重點是展示使用者定義的 Lambda 函數如何使用 AWS IoT Greengrass Core SDK 中的 `StreamManagerClient` 物件與串流管理員互動。為了簡化，您為此教學課程建立的 Python Lambda 函數會產生模擬的裝置資料。

當您使用包含 `Greengrass` 命令的 AWS IoT Greengrass API AWS CLI 來建立群組時，預設會停用串流管理員。若要在核心上啟用串流管理員，您可以[建立包含系統 Lambda 函數的函數定義版本](#)，以及參考新函數定義版本的群組版本。GGStreamManager 然後您將部署群組。

先決條件

為完成此教學課程您需要：

- Greengrass 群組和 Greengrass 核心 (v1.10 或更新版本)。如需如何建立 Greengrass 群組和核心的詳細資訊，請參閱 [入門 AWS IoT Greengrass](#)。入門教學課程也包含安裝 AWS IoT Greengrass 核心軟體的步驟。

 Note

OpenWrt 發行版本不支援串流管理員。

- 在核心裝置上安裝 Java 8 執行時間 (JDK 8)。
- 針對以 Debian 為基礎的發行版本 (包括 Raspbian) 或以 Ubuntu 為基礎的發行版本，請執行下列命令：

```
sudo apt install openjdk-8-jdk
```

- 針對以 Red Hat 為基礎的發行版本 (包括 Amazon Linux)，請執行下列命令：

```
sudo yum install java-1.8.0-openjdk
```

如需詳細資訊，請參閱 OpenJDK 文件上的[如何下載和安裝預先建置的 OpenJDK 套件](#)。

- AWS IoT Greengrass 適用於 Python 1.5.0 版或更新版本的核心 SDK。若要在適用於 Python 的 AWS IoT Greengrass 核心 SDK `StreamManagerClient` 中使用，您必須：
 - 在核心裝置上安裝 Python 3.7 或更新版本。
 - 在 Lambda 函數部署套件中包含 SDK 及其相依性。本教學課程提供相關指示。

 Tip

您可以搭配使用 `StreamManagerClient` 與 Java 或 NodeJS。如需範例程式碼，請參閱 GitHub 上的[AWS IoT Greengrass 適用於 Java 的 Core SDK](#) 和[AWS IoT Greengrass 適用於 Node.js 的 Core SDK](#)。

- 在與 Greengrass 群組相同的 Amazon Kinesis Data Streams 中 `MyKinesisStream` 建立名為 AWS 區域 的目的地串流。如需詳細資訊，請參閱《Amazon Kinesis 開發人員指南》中的[建立串流](#)。

 Note

在本教學課程中，串流管理員會將資料匯出至 Kinesis Data Streams，這會向您的收取費用 AWS 帳戶。如需定價的相關資訊，請參閱 [Kinesis Data Streams 定價](#)。

若要避免產生費用，您可以執行本教學課程而不建立 Kinesis 資料串流。在此情況下，您會檢查日誌，以查看串流管理員嘗試將串流匯出至 Kinesis Data Streams。

- 新增至的 IAM 政策 [the section called “Greengrass 群組角色”](#)，允許對目標資料串流執行 `kinesis:PutRecords` 動作，如下列範例所示：

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kinesis:PutRecords"
      ],
      "Resource": [
        "arn:aws:kinesis:us-east-1:123456789012:stream/MyKinesisStream"
      ]
    }
  ]
}
```

- 在您電腦上 AWS CLI 安裝和設定的。如需詳細資訊，請參閱 AWS Command Line Interface 《使用者指南》中的 [安裝 AWS Command Line Interface](#) 和 [設定 AWS CLI](#)。

此教學課程中的範例命令是針對 Linux 和其他以 Unix 為基礎的系統所撰寫。如果您使用的是 Windows，請參閱 [指定 AWS 命令列界面的參數值](#)，以取得語法差異的詳細資訊。

如果命令包含 JSON 字串，教學課程提供的範例會在單行上有 JSON。在某些系統上，使用此格式編輯和執行命令可能更有效率。

本教學課程所述以下高階執行步驟：

1. [建立 Lambda 函數部署套件](#)
2. [建立 Lambda 函數](#)

3. [建立函數定義和版本](#)
4. [建立記錄器定義和版本](#)
5. [取得您核心定義版本的 ARN](#)
6. [建立群組版本](#)
7. [建立部署](#)
8. [測試應用程式。](#)

此教學課程需約 30 分鐘完成。

步驟 1：建立 Lambda 函數部署套件

在此步驟中，您會建立包含 Python 函數程式碼和相依性的 Lambda 函數部署套件。您稍後在其中建立 Lambda 函數時上傳此套件到 AWS Lambda。Lambda 函數使用 AWS IoT Greengrass 核心 SDK 來建立本機串流並與之互動。

Note

使用者定義的 Lambda 函數必須使用 [AWS IoT Greengrass 核心 SDK](#) 與串流管理員互動。如需有關 Greengrass 串流管理員需求的詳細資訊，請參閱 [Greengrass 串流管理員需求](#)。

1. 下載[AWS IoT Greengrass 適用於 Python 的核心 SDK](#) 1.5.0 版或更新版本。
2. 解壓縮下載的封裝，以取得軟體開發套件。SDK 為 greengrasssdk 資料夾。
3. 安裝套件相依性，以在 Lambda 函數部署套件中包含開發套件。
 1. 前往包含 requirements.txt 檔案的軟體開發套件目錄。這個檔案會列出相依性。
 2. 安裝軟體開發套件相依性。例如，執行下列 pip 命令，將它們安裝在目前的目錄中：

```
pip install --target . -r requirements.txt
```

4. 將以下 Python 程式碼函數儲存在名為 transfer_stream.py 的本機檔案中。

Tip

如需使用 Java 和 NodeJS 的範例程式碼，請參閱 GitHub 上的[AWS IoT Greengrass 適用於 Java 的 Core SDK](#) 和[AWS IoT Greengrass 適用於 Node.js 的 Core SDK](#)。

```
import asyncio
import logging
import random
import time

from greengrasssdk.stream_manager import (
    ExportDefinition,
    KinesisConfig,
    MessageStreamDefinition,
    ReadMessagesOptions,
    ResourceNotFoundException,
    StrategyOnFull,
    StreamManagerClient,
)

# This example creates a local stream named "SomeStream".
# It starts writing data into that stream and then stream manager automatically
# exports
# the data to a customer-created Kinesis data stream named "MyKinesisStream".
# This example runs forever until the program is stopped.

# The size of the local stream on disk will not exceed the default (which is 256
# MB).
# Any data appended after the stream reaches the size limit continues to be
# appended, and
# stream manager deletes the oldest data until the total stream size is back under
# 256 MB.
# The Kinesis data stream in the cloud has no such bound, so all the data from this
# script is
# uploaded to Kinesis and you will be charged for that usage.

def main(logger):
    try:
        stream_name = "SomeStream"
        kinesis_stream_name = "MyKinesisStream"

        # Create a client for the StreamManager
        client = StreamManagerClient()

        # Try deleting the stream (if it exists) so that we have a fresh start
```

```
try:
    client.delete_message_stream(stream_name=stream_name)
except ResourceNotFoundException:
    pass

exports = ExportDefinition(
    kinesisis=[KinesisConfig(identifier="KinesisExport" + stream_name,
kinesis_stream_name=kinesis_stream_name)]
)
client.create_message_stream(
    MessageStreamDefinition(
        name=stream_name,
strategy_on_full=StrategyOnFull.OverwriteOldestData, export_definition=exports
    )
)

# Append two messages and print their sequence numbers
logger.info(
    "Successfully appended message to stream with sequence number %d",
    client.append_message(stream_name, "ABCDEFGHJKLMNO".encode("utf-8")),
)
logger.info(
    "Successfully appended message to stream with sequence number %d",
    client.append_message(stream_name, "PQRSTUVWXYZ".encode("utf-8")),
)

# Try reading the two messages we just appended and print them out
logger.info(
    "Successfully read 2 messages: %s",
    client.read_messages(stream_name,
ReadMessagesOptions(min_message_count=2, read_timeout_millis=1000)),
)

logger.info("Now going to start writing random integers between 0 and 1000
to the stream")
# Now start putting in random data between 0 and 1000 to emulate device
sensor input
while True:
    logger.debug("Appending new random integer to stream")
    client.append_message(stream_name, random.randint(0,
1000).to_bytes(length=4, signed=True, byteorder="big"))
    time.sleep(1)

except asyncio.TimeoutError:
```



```
        logger.exception("Timed out while executing")
    except Exception:
        logger.exception("Exception while running")

def function_handler(event, context):
    return

logging.basicConfig(level=logging.INFO)
# Start up this sample code
main(logger=logging.getLogger())
```

5. 將下列項目壓縮成名為 `transfer_stream_python.zip` 的檔案。這是您的 Lambda 函數部署套件。

- `transfer_stream.py`。應用程式邏輯。
- `greengrasssdk`。發佈 MQTT 訊息的 Python Greengrass Lambda 函數所需的程式庫。

適用於 Python 的 AWS IoT Greengrass Core SDK 1.5.0 版或更新版本提供[串流管理員操作](#)。

- 您為適用於 Python AWS IoT Greengrass 的核心 SDK 安裝的相依性（例如，`cbor2`目錄）。

當您建立 zip 檔案時，請只包含這些項目，而非包含的資料夾。

步驟 2：建立 Lambda 函數

1. 建立 IAM 角色，以便在建立函數時傳入角色 ARN。

JSON Expanded

```
aws iam create-role --role-name Lambda_empty --assume-role-policy '{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

```
}'
```

JSON Single-line

```
aws iam create-role --role-name Lambda_empty --assume-role-policy '{"Version":
"2012-10-17", "Statement": [{"Effect": "Allow", "Principal": {"Service":
"lambda.amazonaws.com"}, "Action": "sts:AssumeRole"}]}'
```

Note

AWS IoT Greengrass 不會使用此角色，因為 Greengrass Lambda 函數的許可是在 Greengrass 群組角色中指定。本教學課程將建立空白角色。

2. 從輸出複製 Arn。
3. 使用 AWS Lambda API 建立 TransferStream 函數。以下命令假設 zip 檔是在目前的目錄。
 - 將 *role-arn* 取代為您複製 Arn 的。

```
aws lambda create-function \
--function-name TransferStream \
--zip-file fileb://transfer_stream_python.zip \
--role role-arn \
--handler transfer_stream.function_handler \
--runtime python3.7
```

4. 發佈函數的一個版本。

```
aws lambda publish-version --function-name TransferStream --description 'First
version'
```

5. 建立發佈版本的別名。

Greengrass 群組可以依別名（建議）或版本參考 Lambda 函數。使用別名可讓您更輕鬆地管理程式碼更新，因為您不必在更新函數程式碼時變更訂閱資料表或群組定義。反之，您只需將別名指向新的函數版本。

```
aws lambda create-alias --function-name TransferStream --name GG_TransferStream --
function-version 1
```

Note

AWS IoT Greengrass 不支援 \$LATEST 版本的 Lambda 別名。

6. 從輸出複製 AliasArn。您在設定 函數時使用此值 AWS IoT Greengrass。

現在您已準備好設定的 函數 AWS IoT Greengrass。

步驟 3：建立函數定義和版本

此步驟會建立參考系統 GGStreamManager Lambda 函數和使用者定義 TransferStream Lambda 函數的函數定義版本。若要在使用 AWS IoT Greengrass API 時啟用串流管理員，您的函數定義版本必須包含 GGStreamManager 函數。

1. 使用包含系統和使用者定義 Lambda 函數的初始版本來建立函數定義。

下列定義版本使用預設[參數設定](#)啟用串流管理員。若要設定自訂設定，您必須定義對應串流管理員參數的環境變數。如需範例，請參閱 [the section called “啟用、停用或設定串流管理員”](#)。針對省略的參數 AWS IoT Greengrass 使用預設設定。MemorySize 應至少為 128000。Pinned 必須設定為 true。

Note

長期（或固定）Lambda 函數會在啟動後自動 AWS IoT Greengrass 啟動，並在自己的容器中繼續執行。這與隨需 Lambda 函數相反，該函數會在叫用時啟動，並在沒有剩餘的任務執行時停止。如需詳細資訊，請參閱[the section called “生命週期組態”](#)。

- 將 *arbitrary-function-id* 取代為函數的名稱，例如 **stream-manager**。
- 將 *alias-arn* 取代為您為 TransferStream Lambda 函數建立別名時 AliasArn 複製的。

JSON expanded

```
aws greengrass create-function-definition --name MyGreengrassFunctions --  
initial-version '{
```

```

"Functions": [
  {
    "Id": "arbitrary-function-id",
    "FunctionArn": "arn:aws:lambda::function:GGStreamManager:1",
    "FunctionConfiguration": {
      "MemorySize": 128000,
      "Pinned": true,
      "Timeout": 3
    }
  },
  {
    "Id": "TransferStreamFunction",
    "FunctionArn": "alias-arn",
    "FunctionConfiguration": {
      "Executable": "transfer_stream.function_handler",
      "MemorySize": 16000,
      "Pinned": true,
      "Timeout": 5
    }
  }
]
}'

```

JSON single

```

aws greengrass create-function-definition \
--name MyGreengrassFunctions \
--initial-version '{"Functions": [{"Id": "arbitrary-function-id", "FunctionArn": "arn:aws:lambda::function:GGStreamManager:1",
  "FunctionConfiguration": {"Environment": {"Variables":
{"STREAM_MANAGER_STORE_ROOT_DIR": "/data", "STREAM_MANAGER_SERVER_PORT":
"1234", "STREAM_MANAGER_EXPORTER_MAX_BANDWIDTH": "20000"}}}, {"Id": "TransferStreamFunction",
  "FunctionArn": "alias-arn", "FunctionConfiguration": {"Executable":
"transfer_stream.function_handler", "MemorySize": 16000, "Pinned":
true, "Timeout": 5}}}]}'

```

Note

函數定義版本需要 Timeout，但 GGStreamManager 不會使用它。如需 Timeout 和其他群組層級設定的詳細資訊，請參閱 [the section called “控制 Greengrass Lambda 函數執行”](#)。

2. 從輸出複製 LatestVersionArn。您可以使用這個值，將函數定義版本新增至您部署到核心的群組版本。

步驟 4：建立一個記錄器定義和版本

設定群組的記錄設定。在本教學課程中，您會設定 AWS IoT Greengrass 系統元件、使用者定義的 Lambda 函數和連接器，將日誌寫入核心裝置的檔案系統。您可以使用記錄檔針對可能遇到的任何問題進行疑難排解。如需詳細資訊，請參閱[the section called “使用 AWS IoT Greengrass 日誌進行監控”](#)。

1. 建立包含最初版本的記錄器定義。

JSON Expanded

```
aws greengrass create-logger-definition --name "LoggingConfigs" --initial-
version '{
  "Loggers": [
    {
      "Id": "1",
      "Component": "GreengrassSystem",
      "Level": "INFO",
      "Space": 10240,
      "Type": "FileSystem"
    },
    {
      "Id": "2",
      "Component": "Lambda",
      "Level": "INFO",
      "Space": 10240,
      "Type": "FileSystem"
    }
  ]
}'
```

JSON Single-line

```
aws greengrass create-logger-definition \  
  --name "LoggingConfigs" \  
  --initial-version '{"Loggers":  
[{"Id":"1","Component":"GreengrassSystem","Level":"INFO","Space":10240,"Type":"FileSystem"  
{"Id":"2","Component":"Lambda","Level":"INFO","Space":10240,"Type":"FileSystem"}]}'
```

2. 從輸出複製記錄器定義的 LatestVersionArn。您可以使用這個值，將紀錄器定義版本新增至您部署到核心的群組版本。

步驟 5：取得您核心定義版本的 ARN

取得核心定義版本的 ARN，以新增至新群組版本。若要部署群組版本，其必須參考包含一個核心的核心定義版本。

1. 取得目標 Greengrass 群組 ID 和目標群組版本 ID。此程序假設這是最新的群組和群組版本。下列查詢會傳回最近建立的群組。

```
aws greengrass list-groups --query "reverse(sort_by(Groups, &CreationTimestamp))  
[0]"
```

或者，您可以依名稱查詢。群組名稱不需要是唯一名稱，因此可能會傳回多個群組。

```
aws greengrass list-groups --query "Groups[?Name=='MyGroup']"
```

Note

您也可以在此 AWS IoT 主控台找到這些值。群組 ID 會顯示在群組的 Settings (設定) 頁面上。群組版本 IDs 會顯示在群組的部署索引標籤上。

2. 從輸出複製目標群組的 Id。部署群組時，您可以使用此 ID 取得核心定義版本。
3. 從輸出複製 LatestVersion，這是新增至群組的最新版本 ID。您可以使用此取得核心定義版本。
4. 取得核心定義版本的 ARN：
 - a. 取得群組版本。

- 將 *group-id* 取代為您為群組所複製的 Id。
- 將 *group-version-id* 換成您為群組所複製的 LatestVersion。

```
aws greengrass get-group-version \  
--group-id group-id \  
--group-version-id group-version-id
```

- b. 從輸出複製 CoreDefinitionVersionArn。您可以使用這個值，將核心定義版本新增至您部署到核心的群組版本。

步驟 6：建立群組版本

現在，您可以開始建立群組版本，其中包含您要部署的實體。若要這樣做，您需要建立群組版本來參考每個元件類型的目標版本。在本教學課程中，您會包含核心定義版本、函數定義版本和記錄器定義版本。

1. 建立群組版本。

- 將 *group-id* 取代為您為群組所複製的 Id。
- 將 *core-definition-version-arn* 換成您為核心定義版本所複製的 CoreDefinitionVersionArn。
- 將 *function-definition-version-arn* 取代LatestVersionArn為您為新函數定義版本複製的。
- 將 *logger-definition-version-arn* 取代LatestVersionArn為您為新記錄器定義版本複製的。

```
aws greengrass create-group-version \  
--group-id group-id \  
--core-definition-version-arn core-definition-version-arn \  
--function-definition-version-arn function-definition-version-arn \  
--logger-definition-version-arn logger-definition-version-arn
```

2. 從輸出複製 Version。這是新群組版本的 ID。

步驟 7：建立部署

將群組部署到核心裝置。

1. 確定 AWS IoT Greengrass 核心正在執行。如果需要，請在您的 Raspberry Pi 終端機執行以下命令。

- a. 檢查精靈是否有在運作：

```
ps aux | grep -E 'greengrass.*daemon'
```

若輸出的 root 含有 `/greengrass/ggc/packages/ggc-version/bin/daemon` 項目，則精靈有在運作。

Note

路徑中的版本取決於核心裝置上安裝的核心 AWS IoT Greengrass 軟體版本。

- b. 若要啟動協助程式：

```
cd /greengrass/ggc/core/  
sudo ./greengrassd start
```

2. 建立 部署。

- 將 *group-id* 取代為您為群組所複製的 Id。
- 將 *group-version-id* 取代為您為新群組版本所複製的 Version。

```
aws greengrass create-deployment \  
--deployment-type NewDeployment \  
--group-id group-id \  
--group-version-id group-version-id
```

3. 從輸出複製 DeploymentId。

4. 取得部署狀態。

- 將 *group-id* 取代為您為群組所複製的 Id。
- 將 *deployment-id* 換成您為部署所複製的 DeploymentId。


```
aws greengrass get-deployment-status \  
--group-id group-id \  
--deployment-id deployment-id
```

如果狀態為 Success，則部署成功。如需故障診斷協助，請參閱[疑難排解](#)。

步驟 8：測試應用程式

TransferStream Lambda 函數會產生模擬的裝置資料。它會將資料寫入串流管理員匯出至目標 Kinesis 資料串流的串流。

1. 在 Amazon Kinesis 主控台的 Kinesis 資料串流下，選擇 MyKinesisStream。

Note

如果您在沒有目標 Kinesis 資料串流的情況下執行教學課程，[請檢查串流管理員的日誌檔案](#) (GGStreamManager)。如果日誌檔案包含錯誤消息中的 export stream MyKinesisStream doesn't exist，則測試成功。此錯誤表示服務嘗試匯出至串流，但串流不存在。

2. 在 MyKinesisStream 頁面上，選擇 Monitoring (監控)。如果測試成功，您應該會看到 Put Records (Put 記錄) 圖表中的資料。視您的連線而定，可能需要一分鐘才會顯示資料。

Important

完成測試後，請刪除 Kinesis 資料串流以免產生更多費用。
或者，執行下列命令來停止 Greengrass 協助程式。這樣可以防止核心傳送訊息，直到您準備好繼續測試為止。

```
cd /greengrass/ggc/core/  
sudo ./greengrassd stop
```

3. 從核心移除 TransferStream Lambda 函數。

- a. 按照 [the section called “建立群組版本”](#) 以建立新群組版本。但移除 create-group-version 命令中的 --function-definition-version-arn 選項。或者，建立不包含 TransferStream Lambda 函數的函數定義版本。

 Note

透過從部署的群組版本省略系統 GGStreamManager Lambda 函數，您可以停用核心上的串流管理。

- b. 請依照 [the section called “建立部署”](#) 以部署新的群組版本。

若要檢視記錄資訊或針對串流問題進行疑難排解，請檢查 TransferStream 和 GGStreamManager 函數的記錄。您必須具有讀取檔案系統 AWS IoT Greengrass 日誌的 root 許可。

- TransferStream 會將日誌項目寫入 *greengrass-root*/ggc/var/log/user/*region*/*account-id*/TransferStream.log。
- GGStreamManager 會將日誌項目寫入 *greengrass-root*/ggc/var/log/system/GGStreamManager.log。

如果您需要更多疑難排解資訊，可以將 Lambda 記錄等級設定為 DEBUG，然後建立並部署新的群組版本。

另請參閱

- [管理資料串流](#)
- [the section called “使用 StreamManagerClient 搭配串流”](#)
- [the section called “匯出支援 AWS 雲端 目的地的組態”](#)
- [the section called “設定串流管理員”](#)
- [the section called “匯出資料串流 \(主控台\)”](#)
- 《[AWS Identity and Access Management 命令參考](#)》中的 [\(IAM\)](#) AWS CLI 命令
- [AWS Lambda 命令](#) AWS CLI 參考中的 命令
- [AWS IoT Greengrass 命令](#) AWS CLI 參考中的 命令

將秘密部署到 AWS IoT Greengrass 核心

此功能適用於 AWS IoT Greengrass Core v1.7 和更新版本。

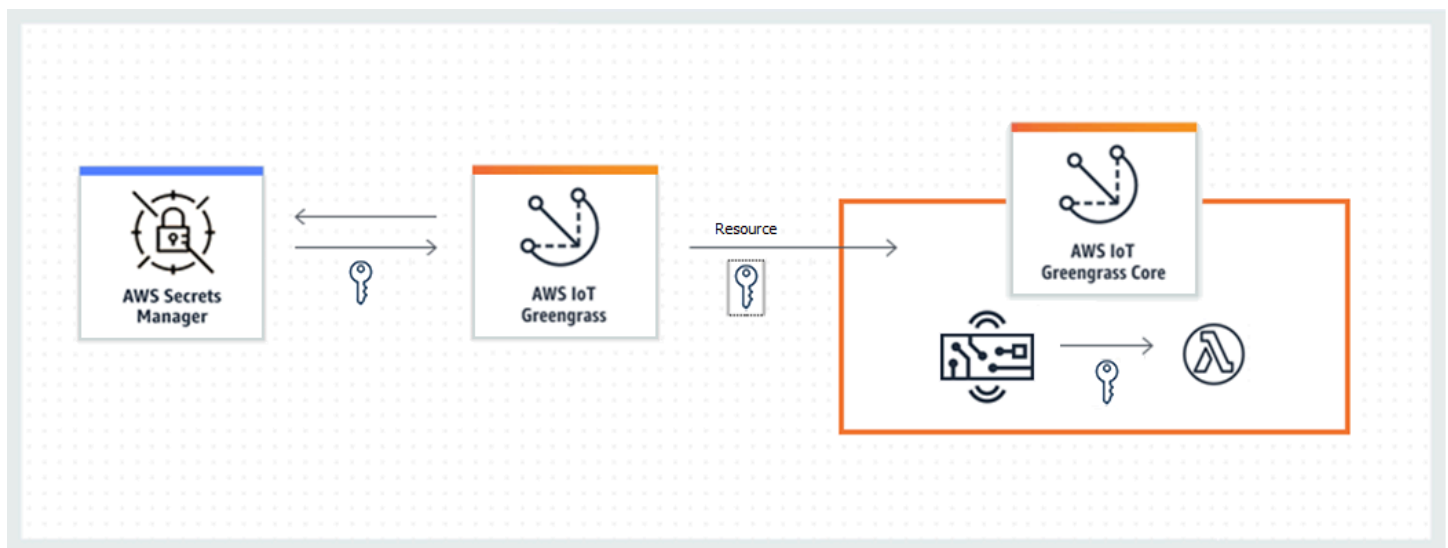
AWS IoT Greengrass 可讓您使用 Greengrass 裝置的服務和應用程式進行身分驗證，無需硬式編碼密碼、字符或其他秘密。

AWS Secrets Manager 是一項服務，可用來在雲端中安全地存放和管理秘密。會將 Secrets Manager AWS IoT Greengrass 延伸到 Greengrass 核心裝置，因此您的[連接器](#)和 Lambda 函數可以使用本機秘密與服務和應用程式互動。例如，Twilio 通知連接器會使用本機儲存的身分驗證字符。

若要將私密整合到 Greengrass 群組，您可以建立一個參考 Secrets Manager 私密的群組資源。此私密資源透過 ARN 參考雲端私密。若要了解如何建立、管理及使用秘密資源，請參閱[the section called “使用秘密資源”](#)。

AWS IoT Greengrass 會在傳輸中和靜態時加密您的秘密。在群組部署期間，從 Secrets Manager AWS IoT Greengrass 擷取秘密，並在 Greengrass 核心上建立本機的加密複本。在 Secrets Manager 中輪換雲端秘密之後，請重新部署 群組，將更新的值傳播到核心。

下圖顯示將私密部署至核心的高階程序。私密是在傳輸和靜態時加密。



使用 AWS IoT Greengrass 在本機存放秘密可提供下列優點：

- 從程式碼分離 (而非寫入程式碼內)。此舉支援集中管理的登入資料，並可協助保護敏感資料免於洩露的風險。
- 適用於離線案例。連接器和函數可以在與網際網路中斷連線時安全地存取本機服務和軟體。

- 控制私密的存取 只有群組中授權的連接器和函數才能存取您的私密。AWS IoT Greengrass 會使用私有金鑰加密來保護您的私密。私密是在傳輸和靜態時加密。如需詳細資訊，請參閱[the section called “秘密金鑰加密”](#)。
- 控制輪換。在 Secrets Manager 中輪換秘密之後，請重新部署 Greengrass 群組以更新秘密的本機複本。如需詳細資訊，請參閱[the section called “建立和管理秘密”](#)。

Important

AWS IoT Greengrass 在輪換雲端版本之後，不會自動更新本機秘密的值。若要更新本機的值，您必須重新部署群組。

秘密金鑰加密

AWS IoT Greengrass 會加密傳輸中和靜態的秘密。

Important

請確定您的使用者定義的 Lambda 函數安全地處理秘密，而且不會記錄存放在秘密中的任何敏感資料。如需詳細資訊，請參閱 AWS Secrets Manager 《使用者指南》中的[減輕記錄和偵錯 Lambda 函數的風險](#)。雖然本文件特別參考輪換函數，但建議也適用於 Greengrass Lambda 函數。

傳輸中加密

AWS IoT Greengrass 使用 Transport Layer Security (TLS) 來加密透過網際網路和本機網路的所有通訊。這可保護傳輸中的秘密，這發生在從 Secrets Manager 擷取秘密並部署到核心時。如需支援的 TLS 密碼套件，請參閱[the section called “TLS 密碼套件支援”](#)。

靜態加密

AWS IoT Greengrass 使用 中指定的私有金鑰[config.json](#)來加密存放在核心上的秘密。基於這個原因，安全儲存私有金鑰對於保護本機秘密來說非常重要。在 AWS [共同責任模型](#)中，客戶有責任保證在核心裝置上安全儲存私有金鑰。

AWS IoT Greengrass 支援兩種私有金鑰儲存模式：

- 使用硬體安全模組。如需詳細資訊，請參閱[the section called “硬體安全整合”](#)。

Note

目前，在使用硬體型私有金鑰時，僅 AWS IoT Greengrass 支援用於加密和解密本機秘密的 [PKCS#1 v1.5](#) 填補機制。如果您遵循廠商提供的指示手動產生硬體型私有金鑰，請務必選擇 PKCS#1 v1.5。AWS IoT Greengrass doesn 不支援最佳非對稱加密填補 (OAEP)。

- 使用檔案系統許可 (預設)。

私有金鑰是用來保護資料金鑰，這是用來加密本機私密。每次部署群組，就會輪換資料金鑰。

AWS IoT Greengrass 核心是唯一可存取私有金鑰的實體。與秘密資源關聯的 Greengrass 連接器或 Lambda 函數會從核心取得秘密的值。

要求

這些是本機私密支援的需求：

- 您必須使用 AWS IoT Greengrass Core v1.7 或更新版本。
- 若要取得本機秘密的值，使用者定義的 Lambda 函數必須使用 AWS IoT Greengrass Core SDK v1.3.0 或更新版本。
- 您必須在 Greengrass 組態檔案中指定用於本機私密加密的私有金鑰。根據預設，AWS IoT Greengrass 會使用存放在檔案系統中的核心私有金鑰。若要提供自己的私有金鑰，請參閱[the section called “指定用於秘密加密的私有金鑰”](#)。僅支援 RSA 金鑰類型。

Note

目前，在使用硬體型私有金鑰時，僅 AWS IoT Greengrass 支援用於加密和解密本機秘密的 [PKCS#1 v1.5](#) 填補機制。如果您遵循廠商提供的指示手動產生硬體型私有金鑰，請務必選擇 PKCS#1 v1.5。AWS IoT Greengrass doesn 不支援最佳非對稱加密填補 (OAEP)。

- AWS IoT Greengrass 必須獲得許可才能取得您的秘密值。這可讓 AWS IoT Greengrass 在群組部署期間擷取值。如果您使用的是預設 Greengrass 服務角色，則 AWS IoT Greengrass 已經可以存取名稱開頭為 greengrass- 的秘密。若要自訂存取，請參閱[the section called “允許 AWS IoT Greengrass 取得秘密值”](#)。

Note

即使您自訂許可，仍建議您使用此命名慣例來識別 AWS IoT Greengrass 允許存取的秘密。主控台會使用不同的許可來讀取您的秘密，因此您可以在主控台中選取 AWS IoT Greengrass 沒有擷取許可的秘密。使用命名慣例可協助避免許可衝突，而此衝突會造成部署錯誤。

指定用於秘密加密的私有金鑰

在此程序中，您會提供用於本機密碼加密的私有金鑰路徑。這必須是具有的最小長度為 2048 位元的 RSA 金鑰。如需 AWS IoT Greengrass 核心上使用之私有金鑰的詳細資訊，請參閱 [the section called “安全性主體”](#)。

AWS IoT Greengrass 支援兩種私有金鑰儲存模式：硬體型或檔案系統型（預設）。如需詳細資訊，請參閱 [the section called “秘密金鑰加密”](#)。

只在您想要變更預設組態時，才遵循此程序，此舉會使用檔案系統中的核心私密金鑰。撰寫這些步驟時，假設您已建立群組和核心，如入門教學課程的 [單元 2](#) 中所述。

1. 開啟位於 `/greengrass-root/config` 目錄的 [config.json](#) 檔案。

Note

`greengrass-root` 代表 Core AWS IoT Greengrass 軟體安裝在您裝置上的路徑。通常，這是 `/greengrass` 目錄。

2. 在 `crypto.principals.SecretsManager` 物件中，對於 `privateKeyPath` 屬性，輸入私有金鑰的路徑：
 - 如果您的私密金鑰儲存在檔案系統，請指定金鑰的絕對路徑。例如：

```
"SecretsManager" : {  
  "privateKeyPath" : "file:///somepath/hash.private.key"  
}
```

- 如果您的私有金鑰存放在硬體安全模組 (HSM) 中，請指定使用 [RFC 7512 PKCS#11](#) URI 配置的路徑。例如：

```
"SecretsManager" : {  
  "privateKeyPath" : "pkcs11:object=private-key-label;type=private"  
}
```

如需詳細資訊，請參閱[the section called “硬體安全組態”](#)。

Note

目前，在使用硬體型私有金鑰時，僅 AWS IoT Greengrass 支援用於加密和解密本機秘密的 [PKCS#1 v1.5](#) 填補機制。如果您遵循廠商提供的指示手動產生硬體型私有金鑰，請務必選擇 PKCS#1 v1.5。AWS IoT Greengrass doesn 不支援最佳非對稱加密填補 (OAEP)。

允許 AWS IoT Greengrass 取得秘密值

在此程序中，您會將內嵌政策新增至允許 AWS IoT Greengrass 取得秘密值的 Greengrass 服務角色。

只有在您想要將自訂許可授予秘密，或您的 Greengrass 服務角色不包含 受管政策時，才遵循此程序。AWSGreengrassResourceAccessRolePolicy 會授予名稱開頭為 greengrass- 的秘密存取權。AWS IoT Greengrass AWSGreengrassResourceAccessRolePolicy

1. 請執行下列 CLI 命令來取得 Greengrass 服務角色的 ARN：

```
aws greengrass get-service-role-for-account --region region
```

傳回的 ARN 包含角色名稱。

```
{  
  "AssociatedAt": "time-stamp",  
  "RoleArn": "arn:aws:iam::account-id:role/service-role/role-name"  
}
```

您會在下列步驟中使用 ARN 或名稱。

2. 新增允許 secretsmanager:GetSecretValue 動作的內嵌政策。如需說明，請參閱《[IAM 使用者指南](#)》中的[新增和移除 IAM 政策](#)。

您可以明確地列出私密或使用萬用字元 * 命名機制，來授予精細的存取，或者您可以授予有條件存取版本控制或已標記私密的權限。例如，下列政策允許 只 AWS IoT Greengrass 讀取指定的秘密。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:GetSecretValue"
      ],
      "Resource": [
        "arn:aws:secretsmanager:us-east-1:123456789012:secret:greengrass-SecretA-abc",
        "arn:aws:secretsmanager:us-east-1:123456789012:secret:greengrass-SecretB-xyz"
      ]
    }
  ]
}
```

Note

如果您使用客戶受管 AWS KMS 金鑰來加密秘密，您的 Greengrass 服務角色也必須允許 kms:Decrypt 動作。

如需有關 Secrets Manager 的 IAM 政策的詳細資訊，請參閱 AWS Secrets Manager 《使用者指南》中的 [您可以在 IAM 政策或 的秘密政策中使用的 和動作、資源和內容金鑰 AWS Secrets Manager 的身分驗證和存取控制 AWS Secrets Manager](#)。

另請參閱

- AWS Secrets Manager 《使用者指南》中的 [什麼是 AWS Secrets Manager？](#)

- [PKCS #1: RSA Encryption Version 1.5](#)

使用秘密資源

AWS IoT Greengrass 使用秘密資源將來自 的秘密整合 AWS Secrets Manager 到 Greengrass 群組。秘密資源是 Secrets Manager 秘密的參考。如需詳細資訊，請參閱[將私密部署至核心](#)。

在 AWS IoT Greengrass 核心裝置上，連接器和 Lambda 函數可以使用秘密資源來驗證 服務和應用程式，而無需硬式編碼密碼、字符或其他登入資料。

建立和管理秘密

在 Greengrass 群組中，秘密資源會參考 Secrets Manager 秘密的 ARN。當秘密資源部署到核心時，秘密的值會加密，並提供給附屬連接器和 Lambda 函數。如需詳細資訊，請參閱[the section called “秘密金鑰加密”](#)。

您可以使用 Secrets Manager 來建立和管理秘密的雲端版本。您可以使用 AWS IoT Greengrass 來建立、管理和部署秘密資源。

Important

建議您遵循在 Secrets Manager 中輪換秘密的最佳實務。然後，部署 Greengrass 群組以更新您私密的本地副本。如需詳細資訊，請參閱AWS Secrets Manager 《使用者指南》中的[輪換您的 AWS Secrets Manager 秘密](#)。

讓秘密可在 Greengrass 核心上使用

1. 在 Secrets Manager 中建立秘密。這是您秘密的雲端版本，會在 Secrets Manager 中集中儲存和管理。管理任務包括輪換私密值和套用資源政策。
2. 在中建立秘密資源 AWS IoT Greengrass。這是依 ARN 參考雲端私密之群組資源的類型。每個群組您只能參考私密一次。
3. 設定您的連接器或 Lambda 函數。您必須指定對應參數或屬性，將資源與連接器或函數建立隸屬關係。這可讓它們取得本地部署之私密資源的值。如需詳細資訊，請參閱[the section called “使用本地私密”](#)。
4. 部署 Greengrass 群組。在部署期間，AWS IoT Greengrass 會擷取雲端私密的值，並在核心上建立 (或更新) 本地私密。

每次擷取秘密值時，AWS IoT Greengrass Secrets Manager AWS CloudTrail 都會記錄事件。AWS IoT Greengrass 不會記錄與部署或使用本機秘密相關的任何事件。如需 Secrets Manager 記錄的詳細資訊，請參閱AWS Secrets Manager 《使用者指南》中的[監控 AWS Secrets Manager 秘密的使用](#)。

包括私密資源中的預備標籤

Secrets Manager 使用預備標籤來識別秘密值的特定版本。預備標籤可以是系統定義或使用者定義。Secrets Manager 會將AWSCURRENT標籤指派給秘密值的最新版本。預備標籤通常用來管理私密輪換。如需 Secrets Manager 版本控制的詳細資訊，請參閱AWS Secrets Manager 《使用者指南》中的[的主要術語和概念 AWS Secrets Manager](#)。

秘密資源一律包含AWSCURRENT預備標籤，如果 Lambda 函數或連接器需要預備標籤，則可以選擇性地包含其他預備標籤。在群組部署期間，會 AWS IoT Greengrass 擷取群組中參考的預備標籤值，然後在核心上建立或更新對應的值。

建立和管理私密資源 (主控台)

建立私密資源 (主控台)

在 AWS IoT Greengrass 主控台中，您可以從群組資源頁面上的秘密索引標籤建立和管理秘密資源。如需建立私密資源，並將其新增到群組的詳細資訊，請參閱[the section called “如何建立秘密資源 \(主控台\)”](#)和[the section called “連接器入門 \(主控台\)”](#)。

Deployments	Resources			
Subscriptions				
Cores	Local	Machine Learning	Secret	
Devices				
Lambdas				
Resources				
Connectors	MyTwilioAuthToken	greengrass-TwilioAuthTo...	● Unaffiliated	AWSCURRENT ...
Tags				
Settings				

Note

或者，主控台可讓您在設定連接器或 Lambda 函數時建立秘密和秘密資源。您可以從連接器的設定參數頁面或 Lambda 函數的資源頁面執行此操作。

管理私密資源 (主控台)

Greengrass 群組中秘密資源的管理任務包括將秘密資源新增至群組、從群組中移除秘密資源，以及變更秘密資源中包含的[預備標籤集](#)。

如果您指向與 Secrets Manager 不同的秘密，您還必須編輯任何使用秘密的連接器：

1. 在群組組態頁面上，選擇 Connectors (連接器)。
2. 從連接器的內容功能表中，選擇 Edit (編輯)。
3. Edit parameters (編輯參數) 頁面會顯示一則訊息，通知您私密 ARN 已變更。若要確認變更，請選擇 Save (儲存)。

如果您在 Secrets Manager 中刪除秘密，請從群組以及參考秘密的連接器和 Lambda 函數中移除對應的秘密資源。否則，在群組部署期間，會 AWS IoT Greengrass 傳回找不到秘密的錯誤。視需要更新您的 Lambda 函數程式碼。

建立和管理私密資源 (CLI)

建立私密資源 (CLI)

在 AWS IoT Greengrass API 中，秘密是一種群組資源。下列範例會建立一個資源定義，其初始版本包含名為 MySecretResource 的私密資源。如需建立私密資源，並將其新增到群組版本的教學課程，請參閱[the section called “連接器入門 \(CLI\)”](#)。

秘密資源參考對應 Secrets Manager 秘密的 ARN，除了 之外，還包含兩個臨時標籤AWSCURRENT，一律包含此標籤。

```
aws greengrass create-resource-definition --name MyGreengrassResources --initial-  
version '{  
  "Resources": [  
    {  
      "Id": "my-resource-id",  
      "Name": "MySecretResource",  
      "ResourceDataContainer": {  
        "SecretsManagerSecretResourceData": {  
          "ARN": "arn:aws:secretsmanager:us-  
west-2:123456789012:secret:greengrass-SomeSecret-KUj89s",  
          "AdditionalStagingLabelsToDownload": [  
            "Label1",  
            "Label2"  
          ]  
        }  
      }  
    ]  
  }
```

```

    }
  }
}
]
}'

```

管理私密資源 (CLI)

Greengrass 群組中秘密資源的管理任務包括將秘密資源新增至群組、從群組中移除秘密資源，以及變更秘密資源中包含的[預備標籤集](#)。

在 AWS IoT Greengrass API 中，這些變更是使用 版本來實作。

AWS IoT Greengrass API 使用版本來管理群組。版本不可變，因此若要新增或變更群組元件，例如群組的用戶端裝置、函數和資源，您必須建立新的或更新的元件版本。然後，您可以建立和部署包含每個元件目標版本的群組版本。若要進一步了解 群組，請參閱 [the section called “AWS IoT Greengrass 群組”](#)。

例如，若要變更私密資源的一組預備標籤，請：

1. 建立一個資源定義版本，其中包含更新的私密資源。下列範例會將第三個預備標籤新增到來自前一節的私密資源。

Note

若要將更多資源新增到版本，請在 `Resources` 陣列中包括它們。

```

aws greengrass create-resource-definition --name MyGreengrassResources --initial-
version '{
  "Resources": [
    {
      "Id": "my-resource-id",
      "Name": "MySecretResource",
      "ResourceDataContainer": {
        "SecretsManagerSecretResourceData": {
          "ARN": "arn:aws:secretsmanager:us-
west-2:123456789012:secret:greengrass-SomeSecret-KUj89s",
          "AdditionalStagingLabelsToDownload": [
            "Label1",
            "Label2",

```

```

    "Label13"
  ]
}
}
}
]
}'

```

2. 如果私密資源的 ID 已變更，請更新使用私密資源的連接器和函數。在新的版本中，更新對應到資源 ID 的參數或屬性。如果私密的 ARN 已變更，您也須針對任何使用私密的連接器更新對應參數。

Note

資源 ID 是客戶提供的任意識別符。

3. 建立一個群組版本，其中包含您要傳送到核心之每個元件的目標版本。
4. 部署群組版本。

如需顯示如何建立和部署私密資源、連接器以及函數的教學課程，請參閱 [the section called “連接器入門 \(CLI\)”](#)。

如果您在 Secrets Manager 中刪除秘密，請從群組以及參考秘密的連接器和 Lambda 函數中移除對應的秘密資源。否則，在群組部署期間，會 AWS IoT Greengrass 傳回找不到秘密的錯誤。視需要更新您的 Lambda 函數程式碼。您可以部署不包含對應秘密資源的資源定義版本，以移除本機秘密。

在連接器和 Lambda 函數中使用本機秘密

Greengrass 連接器和 Lambda 函數使用本機秘密與服務和應用程式互動。根據預設，會使用 AWSCURRENT 值，但也可以使用秘密資源中所含其他 [預備標籤](#) 的值。

您必須先設定連接器和函數，然後它們才能存取本機私密。這會將私密資源與連接器或函數建立隸屬關係。

連接器

如果連接器需要存取本機私密，則它會提供您使用其存取私密所需之資訊設定的參數。

- 若要了解如何在 AWS IoT Greengrass 主控台中執行此操作，請參閱 [the section called “連接器入門 \(主控台\)”](#)。
- 若要了解如何使用 CLI AWS IoT Greengrass 執行此操作，請參閱 [the section called “連接器入門 \(CLI\)”](#)。

如需個別連接器要求的相關資訊，請參閱 [the section called “AWS提供的 Greengrass 連接器”](#)。

連接器內建存取和使用私密的邏輯。

Lambda 函數

若要允許 Greengrass Lambda 函數存取本機秘密，您可以設定函數的屬性。

- 若要了解如何在 AWS IoT Greengrass 主控台中執行此操作，請參閱 [the section called “如何建立秘密資源 \(主控台\)”](#)。
- 若要在 AWS IoT Greengrass API 中執行此操作，請在 ResourceAccessPolicies 屬性中提供下列資訊。
 - ResourceId：Greengrass 群組中秘密資源的 ID。這是參考對應 Secrets Manager 秘密 ARN 的資源。
 - Permission：函數對資源擁有的存取類型。秘密資源僅支援 ro (唯讀) 許可。

下列範例會建立可存取MyApiKey秘密資源的 Lambda 函數。

```
aws greengrass create-function-definition --name MyGreengrassFunctions --initial-  
version '{  
  "Functions": [  
    {  
      "Id": "MyLambdaFunction",  
      "FunctionArn": "arn:aws:lambda:us-  
west-2:123456789012:function:myFunction:1",  
      "FunctionConfiguration": {  
        "Pinned": false,  
        "MemorySize": 16384,  
        "Timeout": 10,  
        "Environment": {  
          "ResourceAccessPolicies": [  
            {  
              "ResourceId": "MyApiKey",  
              "Permission": "ro"  
            }  
          ],  
          "AccessSysfs": true  
        }  
      }  
    }  
  ]  
}'
```

若要在執行時間存取本機秘密，Greengrass Lambda 函數會從 AWS IoT Greengrass 核心 SDK (1.3.0 版或更新版本) 中的 `secretsmanager` 用戶端呼叫 `get_secret_value` 函數。

下列範例示範如何使用適用於 Python 的 AWS IoT Greengrass 核心 SDK 來取得秘密。它會將秘密的名稱傳遞給 `get_secret_value` 函數。SecretId 可以是 Secrets Manager 秘密的名稱或 ARN (而非秘密資源)。

```
import greengrasssdk

secrets_client = greengrasssdk.client("secretsmanager")
secret_name = "greengrass-MySecret-abc"

def function_handler(event, context):
    response = secrets_client.get_secret_value(SecretId=secret_name)
    secret = response.get("SecretString")
```

對於文字類型秘密，`get_secret_value` 函數會傳回一個字串。對於二進位類型秘密，它會傳回一個 base64 編碼字串。

Important

請確定您的使用者定義的 Lambda 函數安全地處理秘密，而且不會記錄存放在秘密中的任何敏感資料。如需詳細資訊，請參閱 AWS Secrets Manager 《使用者指南》中的 [降低記錄和偵錯 Lambda 函數的風險](#)。雖然本文件特別參考輪換函數，但建議也適用於 Greengrass Lambda 函數。

根據預設，會傳回秘密的目前值。這是連接了 `AWSCURRENT` 預備標籤的版本。若要存取不同的版本，請針對選用的 `VersionStage` 引數傳遞對應預備標籤的名稱。例如：

```
import greengrasssdk

secrets_client = greengrasssdk.client("secretsmanager")
secret_name = "greengrass-TestSecret"
```

```
secret_version = "MyTargetLabel"

# Get the value of a specific secret version
def function_handler(event, context):
    response = secrets_client.get_secret_value(
        SecretId=secret_name, VersionStage=secret_version
    )
    secret = response.get("SecretString")
```

如需另一個呼叫 `get_secret_value` 的函數範例，請參閱[建立 Lambda 函數部署套件](#)。

如何建立秘密資源 (主控台)

此功能適用於 AWS IoT Greengrass Core v1.7 和更新版本。

本教學課程說明如何使用 AWS 管理主控台 將秘密資源新增至 Greengrass 群組。秘密資源是來自 秘密的參考 AWS Secrets Manager。如需詳細資訊，請參閱[將私密部署至核心](#)。

在 AWS IoT Greengrass 核心裝置上，連接器和 Lambda 函數可以使用秘密資源來驗證 服務和應用程式，而無需硬式編碼密碼、字符或其他登入資料。

在本教學課程中，您會從在 AWS Secrets Manager 主控台中建立秘密開始。然後，在 AWS IoT Greengrass 主控台中，您可以從群組的資源頁面將秘密資源新增至 Greengrass 群組。此秘密資源參考 Secrets Manager 秘密。稍後，您將秘密資源連接到 Lambda 函數，這可讓函數取得本機秘密的值。

Note

或者，主控台可讓您在設定連接器或 Lambda 函數時建立秘密和秘密資源。您可以從連接器的設定參數頁面或 Lambda 函數的資源頁面執行此操作。

只有包含私密參數的連接器才能存取私密。如需示範 Twilio Notifications 連接器如何使用本機儲存的身分驗證字符的教學課程，請參閱 [the section called “連接器入門 \(主控台\)”](#)。

本教學課程所述以下高階執行步驟：

1. [建立 Secrets Manager 秘密](#)

2. [將私密資源新增至群組](#)
3. [建立 Lambda 函數部署套件](#)
4. [建立 Lambda 函式](#)
5. [新增 函數至群組](#)
6. [將秘密資源連接到函數](#)
7. [新增訂閱到群組](#)
8. [部署群組](#)
9. [the section called “測試 Lambda 函數”](#)

此教學課程需約 20 分鐘完成。

先決條件

為完成此教學課程您需要：

- Greengrass 群組和 Greengrass 核心 (1.7 版或更新版本)。若要了解如何建立 Greengrass 群組或核心，請參閱 [入門 AWS IoT Greengrass](#)。入門教學課程也包含安裝 AWS IoT Greengrass 核心軟體的步驟。
- AWS IoT Greengrass 必須設定為支援本機秘密。如需詳細資訊，請參閱[秘密要求](#)。

Note

此要求包括允許存取您的 Secrets Manager 秘密。如果您使用預設 Greengrass 服務角色，Greengrass 有權取得名稱開頭為 greengrass- 的秘密值。

- 若要取得本機秘密的值，使用者定義的 Lambda 函數必須使用 AWS IoT Greengrass Core SDK v1.3.0 或更新版本。

步驟 1：建立 Secrets Manager 秘密

在此步驟中，您會使用 AWS Secrets Manager 主控台來建立秘密。

1. 登入 [AWS Secrets Manager 主控台](#)。

Note

如需此程序的詳細資訊，請參閱AWS Secrets Manager 《使用者指南》中的[步驟 1：建立和存放您的秘密 AWS Secrets Manager](#)。

2. 選擇儲存新機密。
3. 在選擇秘密類型下，選擇其他類型的秘密。
4. 在 Specify the key/value pairs to be stored for this secret (指定此秘密要儲存的金鑰/數值組) 下：
 - 在 Key (索引鍵) 欄位，輸入 **test**。
 - 針對數值，輸入 **abcdefghi**。
5. 保持選取加密金鑰的 `aws/secretsmanager`，然後選擇下一步。

Note

AWS KMS 如果您使用 Secrets Manager 在帳戶中建立的預設 AWS 受管金鑰，則不會向您收費。

6. 針對 Secret name (私密名稱)，輸入 **greengrass-TestSecret**，然後選擇 Next (下一步)。

Note

根據預設，Greengrass 服務角色允許 AWS IoT Greengrass 取得名稱開頭為 greengrass- 的私密值。如需詳細資訊，請參閱[私密需求](#)。

7. 本教學課程不需要輪換，因此請選擇停用自動輪換，然後選擇下一步。
8. 在 Review (檢閱) 頁面上，檢閱設定，然後選擇 Store (儲存)。

接著，您可以在 Greengrass 群組中建立一個參考私密的私密資源。

步驟 2：將私密資源新增至 Greengrass 群組

在此步驟中，您會設定參考 Secrets Manager 秘密的群組資源。

1. 在 AWS IoT 主控台導覽窗格的管理下，展開 Greengrass 裝置，然後選擇群組 (V1)。
2. 選擇您要將私密資源新增至其中的群組。

3. 在群組組態頁面上，選擇資源索引標籤，然後向下捲動至秘密區段。秘密區段會顯示屬於群組的秘密資源。您可以從本節新增、編輯和移除秘密資源。

 Note

或者，主控台可讓您在設定連接器或 Lambda 函數時建立秘密和秘密資源。您可以從連接器的設定參數頁面或 Lambda 函數的資源頁面執行此操作。

4. 在秘密區段下選擇新增。
5. 在新增秘密資源頁面上，**MyTestSecret**輸入資源名稱。
6. 在秘密下，選擇 greengrass-TestSecret。
7. 在選取標籤（選用）區段中，AWSCURRENT 預備標籤代表秘密的最新版本。此標籤一律包含在秘密資源中。

 Note

本教學課程僅需要 AWSCURRENT 標籤。您可以選擇性地包含 Lambda 函數或連接器所需的標籤。

8. 選擇 Add resource (新增資源)。

步驟 3：建立 Lambda 函數部署套件

若要建立 Lambda 函數，您必須先建立包含函數程式碼和相依性的 Lambda 函數部署套件。Greengrass Lambda 函數需要 [AWS IoT Greengrass 核心 SDK](#) 來處理任務，例如與核心環境中的 MQTT 訊息通訊，以及存取本機秘密。本教學課程會建立 Python 函數，因此您可以在部署套件中使用 SDK 的 Python 版本。

 Note

若要取得本機秘密的值，使用者定義的 Lambda 函數必須使用 AWS IoT Greengrass Core SDK v1.3.0 或更新版本。

1. 從[AWS IoT Greengrass 核心開發套件](#)下載頁面，將適用於 Python AWS IoT Greengrass 的核心開發套件下載至您的電腦。
2. 解壓縮下載的封裝，以取得軟體開發套件。SDK 為 greengrasssdk 資料夾。

3. 將以下 Python 程式碼函數儲存在名為 `secret_test.py` 的本機檔案中。

```
import greengrasssdk

secrets_client = greengrasssdk.client("secretsmanager")
iot_client = greengrasssdk.client("iot-data")
secret_name = "greengrass-TestSecret"
send_topic = "secrets/output"

def function_handler(event, context):
    """
    Gets a secret and publishes a message to indicate whether the secret was
    successfully retrieved.
    """
    response = secrets_client.get_secret_value(SecretId=secret_name)
    secret_value = response.get("SecretString")
    message = (
        f"Failed to retrieve secret {secret_name}."
        if secret_value is None
        else f"Successfully retrieved secret {secret_name}."
    )
    iot_client.publish(topic=send_topic, payload=message)
    print("Published: " + message)
```

`get_secret_value` 函數支援 `SecretId` 值的 Secrets Manager 秘密名稱或 ARN。此範例使用私密名稱。在此範例秘密中，AWS IoT Greengrass 會傳回鍵/值對：`{"test": "abcdefghi"}`。

Important

請確定您的使用者定義的 Lambda 函數安全地處理秘密，而且不會記錄存放在秘密中的任何敏感資料。如需詳細資訊，請參閱 AWS Secrets Manager 《使用者指南》中的 [降低記錄和偵錯 Lambda 函數的風險](#)。雖然本文件特別參考輪換函數，但建議也適用於 Greengrass Lambda 函數。

4. 將下列項目壓縮成名為 `secret_test_python.zip` 的檔案。當您建立 ZIP 檔案時，只包含程式碼及其依存項目，而不包含資料夾。

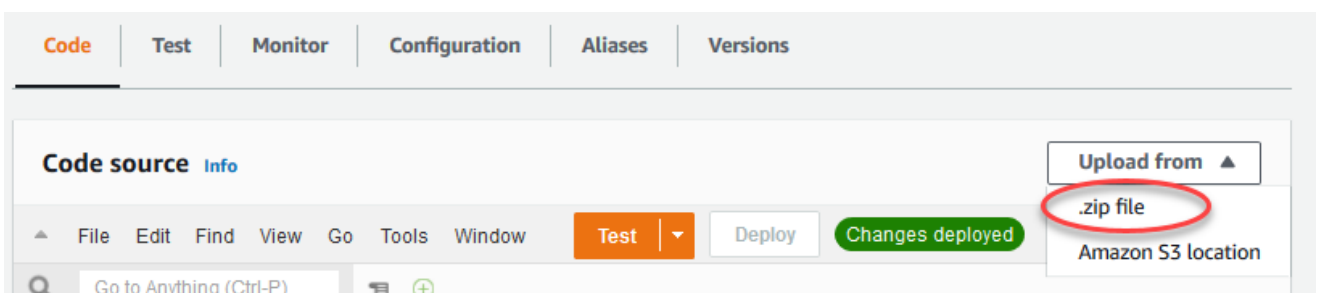
- `secret_test.py`。應用程式邏輯。
- `greengrasssdk`。所有 Python Greengrass Lambda 函數所需的程式庫。

這是您的 Lambda 函數部署套件。

步驟 4：建立 Lambda 函數

在此步驟中，您會使用 AWS Lambda 主控台來建立 Lambda 函數，並將其設定為使用您的部署套件。然後，您會發佈函數版本和建立別名。

1. 首先，建立 Lambda 函數。
 - a. 在 AWS 管理主控台，選擇服務，然後開啟 AWS Lambda 主控台。
 - b. 選擇建立函數，然後選擇從頭開始撰寫。
 - c. 在 Basic information (基本資訊) 區段中，使用下列值：
 - 針對函數名稱，請輸入 **SecretTest**。
 - 針對執行期，選擇 Python 3.7。
 - 對於許可，請保留預設設定。這會建立授予基本 Lambda 許可的執行角色。不會使用此角色 AWS IoT Greengrass。
 - d. 請在頁面底部，選擇建立函式。
2. 接著，註冊處理常式並上傳您的 Lambda 函數部署套件。
 - a. 在程式碼索引標籤的程式碼來源下，選擇上傳來源。從下拉式清單中，選擇 `.zip` 檔案。



- b. 選擇上傳，然後選擇您的 `secret_test_python.zip` 部署套件。然後選擇 Save (儲存)。
- c. 在函數的程式碼索引標籤的執行時間設定下，選擇編輯，然後輸入下列值。
 - 針對執行期，選擇 Python 3.7。

- 對於 Handler (處理常式)，輸入 `secret_test.function_handler`。

d. 選擇儲存。

 Note

AWS Lambda 主控台上的測試按鈕不適用於此函數。AWS IoT Greengrass 核心 SDK 不包含在 AWS Lambda 主控台中獨立執行 Greengrass Lambda 函數所需的模組。這些模組 (例如 `greengrass_common`) 會在部署到您的 Greengrass 核心之後提供給函數。

3. 現在，發佈 Lambda 函數的第一個版本，並為[該版本建立別名](#)。

 Note

Greengrass 群組可以依別名 (建議) 或版本參考 Lambda 函數。使用別名可讓您更輕鬆地管理程式碼更新，因為您不必在更新函數程式碼時變更訂閱資料表或群組定義。反之，您只需將別名指向新的函數版本。

- 請從操作功能表中選擇發行新版本。
- 針對 Version description (版本描述)，輸入 **First version**，然後選擇 Publish (發佈)。
- 在 SecretTest: 1 組態頁面上，從 Actions (動作) 功能表中選擇 Create alias (建立別名)。
- 在建立警示頁面上使用下列值：
 - 對於名稱，輸入 **GG_SecretTest**。
 - 對於版本，選擇 1。

 Note

AWS IoT Greengrass 不支援 \$LATEST 版本的 Lambda 別名。

e. 選擇建立。

現在您已準備好將 Lambda 函數新增至 Greengrass 群組，並連接秘密資源。

步驟 5：將 Lambda 函數新增至 Greengrass 群組

在此步驟中，您將 Lambda 函數新增至 AWS IoT 主控台中的 Greengrass 群組。

1. 在群組組態頁面上，選擇 Lambda 函數索引標籤。
2. 在我的 Lambda 函數區段下，選擇新增。
3. 針對 Lambda 函數，選擇 SecretTest。
4. 針對 Lambda 函數版本，選擇您發佈的版本別名。

接著，設定 Lambda 函數的生命週期。

1. 在 Lambda 函數組態區段中，進行下列更新。

Note

建議您在沒有容器化的情況下執行 Lambda 函數，除非您的業務案例需要。這有助於啟用裝置 GPU 和攝影機的存取，而無需設定裝置資源。如果您在沒有容器化的情況下執行，您還必須授予 AWS IoT Greengrass Lambda 函數的根存取權。

- a. 若要在沒有容器化的情況下執行：

- 針對系統使用者和群組，選擇 **Another user ID/group ID**。針對系統使用者 ID，輸入 **0**。針對系統群組 ID，輸入 **0**。

這可讓您的 Lambda 函數以根執行。如需以根身分執行的詳細資訊，請參閱 [the section called “為群組中的 Lambda 函數設定預設存取身分”](#)。

Tip

您也必須更新 config.json 檔案，以授予 Lambda 函數的根存取權。如需此程序，請參閱 [the section called “執行 Lambda 函數做為根”](#)。

- 針對 Lambda 函數容器化，選擇無容器。

如需在沒有容器化的情況下執行 的詳細資訊，請參閱 [the section called “選擇 Lambda 函數容器化時的考量事項”](#)。

- 針對 Timeout (逾時)，輸入 **10 seconds**。

- 針對固定，選擇 True。

如需詳細資訊，請參閱[the section called “生命週期組態”](#)。

- 在其他參數下，針對 /sys 目錄的讀取存取權，選擇已啟用。

b. 改為在容器化模式下執行：

 Note

除非您的商業案例需要，否則我們不建議在容器化模式下執行。

- 針對系統使用者和群組，選擇使用群組預設值。
- 針對 Lambda 函數容器化，選擇使用群組預設值。
- 針對 Memory limit (記憶體限制)，輸入 **1024 MB**。
- 針對 Timeout (逾時)，輸入 **10 seconds**。
- 針對固定，選擇 True。

如需詳細資訊，請參閱[the section called “生命週期組態”](#)。

- 在其他參數下，針對 /sys 目錄的讀取存取權，選擇已啟用。

2. 選擇新增 Lambda 函數。

接著，將秘密資源與 函數建立關聯。

步驟 6：將秘密資源連接至 Lambda 函數

在此步驟中，您將秘密資源與 Greengrass 群組中的 Lambda 函數建立關聯。這會將資源與 函數建立關聯，讓函數取得本機秘密的值。

1. 在群組組態頁面上，選擇 Lambda 函數索引標籤。
2. 選擇 SecretTest 函數。
3. 在函數的詳細資訊頁面上，選擇資源。
4. 捲動至秘密區段，然後選擇關聯。
5. 選擇 MyTestSecret，然後選擇關聯。

步驟 7：將訂閱新增到 Greengrass 群組

在此步驟中，您可以新增允許 AWS IoT 和 Lambda 函數交換訊息的訂閱。一個訂閱允許 AWS IoT 调用函數，一個則允許函數將輸出資料傳送到其中 AWS IoT。

1. 在群組組態頁面上，選擇訂閱索引標籤，然後選擇新增訂閱。
2. 建立允許 AWS IoT 將訊息發佈至 函數的訂閱。

在群組組態頁面上，選擇訂閱索引標籤，然後選擇新增訂閱。

3. 在建立訂閱頁面上，設定來源和目標，如下所示：
 - a. 在來源類型中，選擇 Lambda 函數，然後選擇 IoT Cloud。
 - b. 在目標類型中，選擇服務，然後選擇 SecretTest。
 - c. 在主題篩選條件中，輸入 **secrets/input**，然後選擇建立訂閱。
4. 新增第二個訂閱。選擇訂閱索引標籤，選擇新增訂閱，並設定來源和目標，如下所示：
 - a. 在來源類型中，選擇服務，然後選擇 SecretTest。
 - b. 在目標類型中，選擇 Lambda 函數，然後選擇 IoT Cloud。
 - c. 在主題篩選條件中，輸入 **secrets/output**，然後選擇建立訂閱。

步驟 8：部署 Greengrass 群組

將群組部署到核心裝置。在部署期間，會從 Secrets Manager AWS IoT Greengrass 擷取秘密的值，並在核心上建立本機的加密複本。

1. 確定 AWS IoT Greengrass 核心正在執行。如果需要，請在您的 Raspberry Pi 終端機執行以下命令。
 - a. 檢查精靈是否有在運作：

```
ps aux | grep -E 'greengrass.*daemon'
```

若輸出的 root 含有 /greengrass/ggc/packages/*ggc-version*/bin/daemon 項目，則精靈有在運作。

 Note

路徑中的版本取決於核心裝置上安裝的核心 AWS IoT Greengrass 軟體版本。

- b. 若要啟動協助程式：

```
cd /greengrass/ggc/core/  
sudo ./greengrassd start
```

2. 在群組組態頁面上，選擇部署。
3.
 - a. 在 Lambda 函數索引標籤的系統 Lambda 函數區段下，選取 IP 偵測器，然後選擇編輯。
 - b. 在編輯 IP 偵測器設定對話方塊中，選取自動偵測和覆寫 MQTT 代理程式端點。
 - c. 選擇儲存。

這可讓裝置自動取得核心的連接資訊，例如 IP 位址、DNS、連接埠編號。建議自動偵測，但 AWS IoT Greengrass 也支援手動指定的端點。只會在第一次部署群組時收到復原方法的提示。

 Note

如果出現提示，請授予許可來建立 [Greengrass 服務角色](#)，並將其與目前 AWS 帳戶中的 建立關聯 AWS 區域。此角色允許 AWS IoT Greengrass 存取 AWS 服務中的資源。

此部署頁面會顯示部署時間戳記、版本 ID 和狀態。完成後，部署顯示的狀態應為已完成。

如需故障診斷協助，請參閱[疑難排解](#)。

測試 Lambda 函數

1. 在 AWS IoT 主控台首頁上，選擇測試。
2. 對於訂閱主題，請使用下列值，然後選擇訂閱。

屬性	Value
訂閱主題	私密/輸出
MQTT 承載顯示	將承載顯示為字串

3. 針對發佈至主題，請使用下列值，然後選擇發佈以叫用函數。

屬性	Value
主題	私密/輸入
訊息	保留預設訊息。發佈訊息會叫用 Lambda 函數，但本教學課程中的函數不會處理訊息內文。

如果成功，函數會發佈「Success (成功)」訊息。

另請參閱

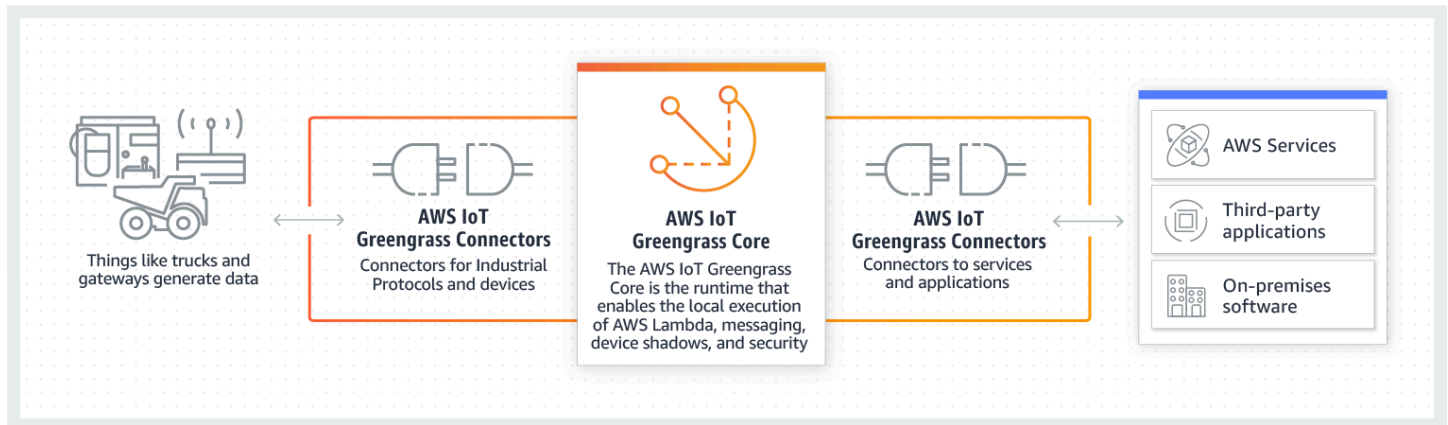
- [將私密部署至核心](#)

使用 Greengrass 連接器來整合服務和通訊協定

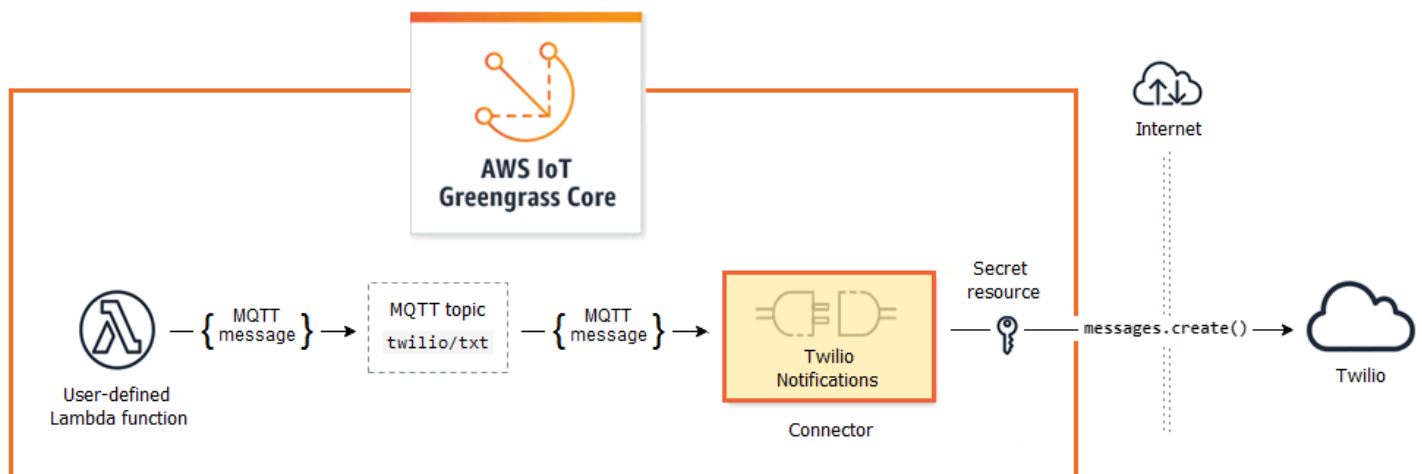
此功能適用於 AWS IoT Greengrass Core v1.7 和更新版本。

中的連接器 AWS IoT Greengrass 是預先建置的模組，可讓您更有效率地與本機基礎設施 AWS、裝置通訊協定和其他雲端服務互動。透過使用連接器，您可以花較少的時間學習新的通訊協定和 APIs 並有更多時間專注於對您業務至關重要的邏輯。

下圖顯示連接器可以適應 AWS IoT Greengrass 環境的位置。



許多連接器使用 MQTT 訊息與群組中的用戶端裝置和 Greengrass Lambda 函數，或與 AWS IoT 和本機影子服務通訊。在下列範例中，Twilio Notifications 連接器會從使用者定義的 Lambda 函數接收 MQTT 訊息、使用來自秘密的本機參考 AWS Secrets Manager，並呼叫 Twilio API。



如需建立此解決方案的教學課程，請參閱 [the section called “連接器入門 \(主控台\)”](#) 和 [the section called “連接器入門 \(CLI\)”](#)。

Greengrass 連接器可協助您擴展裝置功能或建立單一用途裝置。透過使用連接器，您可以：

- 實作可重複使用的商業邏輯。
- 與雲端和本機服務互動，包括 AWS 和第三方服務。
- 擷取和處理裝置資料。
- 使用 MQTT 主題訂閱和使用使用者定義的 Lambda 函數啟用 device-to-device 呼叫。

AWS 提供一組 Greengrass 連接器，可簡化與常見服務和資料來源的互動。這些預先建置的模組可提供記錄和診斷、補充、產業資料處理及警示和簡訊的案例。如需詳細資訊，請參閱[the section called “AWS提供的 Greengrass 連接器”](#)。

需求

若要使用連接器，請謹記以下幾點：

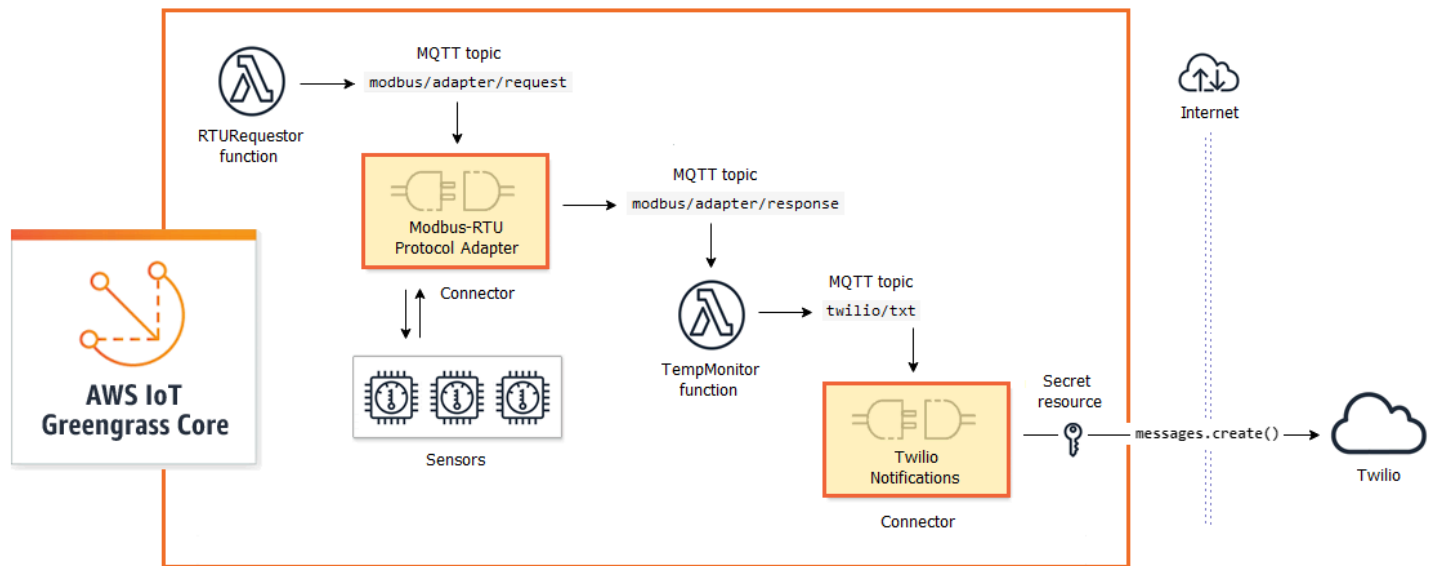
- 您使用的每個連接器都有您必須符合的要求。這些要求可能包括最低 AWS IoT Greengrass Core 軟體版本、裝置先決條件、必要的許可和限制。如需詳細資訊，請參閱[the section called “AWS提供的 Greengrass 連接器”](#)。
- Greengrass 群組只能包含指定連接器的一個已設定執行個體。不過，您可以在多個訂閱中使用執行個體。如需詳細資訊，請參閱[the section called “組態參數”](#)。
- 當 Greengrass 群組的預設容器化設定為 [No container \(無容器\)](#) 時，群組中的連接器必須在沒有容器化的情況下執行。若要尋找支援 No container (無容器) 模式的連接器，請參閱 [the section called “AWS提供的 Greengrass 連接器”](#)。

使用 Greengrass 連接器

連接器是一種群組元件。與其他群組元件一樣，例如用戶端裝置和使用使用者定義的 Lambda 函數，您可以將連接器新增至群組、設定其設定，並將它們部署到 AWS IoT Greengrass 核心。連接器在核心環境中執行。

您可以將某些連接器部署為簡單的獨立應用程式。例如，Device Defender 連接器會從核心裝置讀取系統指標，並將其傳送至 AWS IoT Device Defender 進行分析。

您可以在較大的解決方案中新增其他連接器做為建置區塊。下列範例解決方案使用 Modbus-RTU 通訊協定轉接器連接器來處理來自感應器的訊息，並使用 Twilio Notifications 連接器啟動 Twilio 訊息。



解決方案通常包含使用者定義的 Lambda 函數，這些函數位於連接器旁邊，並處理連接器傳送或接收的資料。在此範例中，TempMonitor 函數會從 Modbus-RTU 通訊協定轉接器接收資料、執行一些商業邏輯，然後將資料傳送至 Twilio Notifications。

若要建立和部署解決方案，請依照此一般程序：

1. 規劃高階資料流程。識別您需要使用的資料來源、資料管道、服務、通訊協定和資源。在範例解決方案中，這包括經過 Modbus RTU 通訊協定、實體 Modbus 序列埠和 Twilio 的資料。
2. 識別要在解決方案中包含的連接器，並將其新增到群組。範例解決方案使用 Modbus-RTU 通訊協定轉接器和 Twilio Notifications。為了協助尋找適用您情境的連接器並了解其個別要求，請參閱 [the section called “AWS提供的 Greengrass 連接器”](#)。
3. 識別是否需要使用者定義的 Lambda 函數、用戶端裝置或資源，然後建立它們並將其新增至群組。這可能包括包含了商業邏輯的函數，或將資料處理成解決方案中另一個實體所需的格式。範例解決方案使用函數傳送 Modbus RTU 請求並啟動 Twilio 通知。此外也包含 Modbus RTU 序列埠的本機裝置資源和 Twilio 驗證字符的私密資源。

Note

私密資源會參考來自 AWS Secrets Manager 的密碼、字符和其他私密。連接器和 Lambda 函數可以使用秘密來驗證服務和應用程式。根據預設，AWS IoT Greengrass 可以存取名稱開頭為 "greengrass-" 的秘密。如需詳細資訊，請參閱[將私密部署至核心](#)。

4. 建立訂閱以允許解決方案中的實體交換 MQTT 訊息。若訂閱中使用連接器，則連接器和訊息來源或目標必須使用連接器支援的預先定義主題語法。如需詳細資訊，請參閱[the section called “輸入和輸出”](#)。
5. 將群組部署至 Greengrass 核心。

如需有關建立和部署連接器的資訊，請參閱下列教學課程：

- [the section called “連接器入門 \(主控台\)”](#)
- [the section called “連接器入門 \(CLI\)”](#)

組態參數

許多連接器會提供參數讓您自訂行為或輸出。這些參數用於連接器生命週期的初始化期間、執行時間或其他時間。

參數類型和用法依連接器而有不同。例如，SNS 連接器具有設定預設 SNS 主題的參數，而 Device Defender 具有設定資料取樣率的參數。

一個群組版本可以包含多個連接器，但特定連接器一次只能有一個執行個體。這表示群組中的每個連接器都僅會有一個作用中的組態。然而，連接器執行個體可在群組中的多個訂閱中使用。例如，您可以建立訂閱，允許許多裝置將資料傳送至 Kinesis Firehose 連接器。

用於存取群組資源的參數

Greengrass 連接器使用群組資源存取核心裝置上的檔案系統、連接埠、周邊設備和其他本機資源。如果連接器需要存取群組資源，則會提供相關的組態參數。

群組資源包括：

- [本機資源](#)。Greengrass 核心裝置上存在的目錄、檔案、連接埠、接腳和周邊設備。
- [機器學習資源](#)。在雲端中訓練並部署至核心的機器學習模型，用於本機推論。
- [私密資源](#)。密碼、金鑰、字符或任意文字的本機加密副本 AWS Secrets Manager。連接器可以安全地存取這些本機私密，並使用它們來向服務或本機基礎設施進行驗證。

例如，Device Defender 的參數可讓您存取主機/proc目錄中的系統指標，而 Twilio Notifications 的參數可讓您存取本機儲存的 Twilio 身分驗證字符。

更新連接器參數

參數是在連接器新增到 Greengrass 群組時設定。新增連接器之後，您可以變更參數值。

- 在主控台中：從群組組態頁面，開啟 Connectors (連接器)，並從連接器的內容選單選擇 Edit (編輯)。

Note

如果連接器使用的私密資源後來變更為參考不同的私密，則您必須編輯連接器的參數並確認變更。

- 在 API 中：建立連接器的另一個版本來定義新組態。

AWS IoT Greengrass API 使用版本來管理群組。版本不可變，因此若要新增或變更群組元件，例如群組的用戶端裝置、函數和資源，您必須建立新的或更新的元件版本。然後，您可以建立和部署包含每個元件目標版本的群組版本。

變更連接器組態後，您必須部署群組，以將變更傳播至核心。

輸入和輸出

許多 Greengrass 連接器可藉由傳送和接收 MQTT 訊息與其他實體通訊。MQTT 通訊是由允許連接器與 Greengrass 群組中的 Lambda 函數、用戶端裝置和其他連接器，或與 AWS IoT 和本機影子服務交換資料的訂閱所控制。若要允許此通訊，您必須在連接器所屬的群組中建立訂閱。如需詳細資訊，請參閱 [the section called “MQTT 簡訊工作流程中的受管訂閱”](#)。

連接器可以是訊息發佈者、訊息訂閱者或兩者。每個連接器都會定義其發佈或訂閱的 MQTT 主題。這些預先定義的主題必須在連接器是訊息來源或訊息目標的訂閱中使用。如需包括為連接器設定訂閱之步驟的教學課程，請參閱 [the section called “連接器入門 \(主控台\)”](#) 和 [the section called “連接器入門 \(CLI\)”](#)。

Note

許多連接器也有內建的通訊模式可以與雲端或本機服務互動。這些會隨著連接器而有不同，可能需要您設定參數或將許可新增到 [群組角色](#)。如需連接器要求的相關資訊，請參閱 [the section called “AWS提供的 Greengrass 連接器”](#)。

輸入主題

大多數連接器會接收 MQTT 主題的輸入資料。有些連接器會訂閱輸入資料的多個主題。例如，序列串流連接器支援兩個主題：

- `serial/+/read/#`
- `serial/+/write/#`

針對此連接器，讀取和寫入請求會傳送至對應的主題。建立訂閱時，務必使用適用於實作的主題。

前述範例中的 `+` 和 `#` 字元是萬用字元。這些萬用字元可讓訂閱者在多個主題上接收訊息，並可讓發佈者自訂他們發佈到的主題。

- `+` 萬用字元可出現在主題階層的任意處。這可替換成一個階層項目。

例如，針對 `sensor/+/input` 主題，訊息可以發佈到 `sensor/id-123/input` 主題，但不能發佈到 `sensor/group-a/id-123/input`。

- `#` 萬用字元只能出現在主題階層的結尾。這可替換為零或多個階層。

例合，針對 `sensor/#` 主題，訊息可以發佈到 `sensor/`、`sensor/id-123` 和 `sensor/group-a/id-123`，但不能發佈到 `sensor`。

只有在訂閱主題時，萬用字元才有效。無法將訊息發佈到包含萬用字元的主題。如需連接器輸入或輸出主題需求的詳細資訊，請參閱連接器的文件。如需詳細資訊，請參閱[the section called “AWS提供的 Greengrass 連接器”](#)。

容器化支援

根據預設，大部分的連接器會在由 AWS IoT Greengrass 管理之隔離執行階段環境中的 Greengrass 核心上執行。這些執行階段環境稱為容器，可在連接器與主機系統之間提供隔離，為主機與連接器提供更高的安全性。

不過，在某些環境中不支援此 Greengrass 容器化，例如當您 AWS IoT Greengrass 在 Docker 容器或沒有 `cgroup` 的較舊 Linux 核心上執行時。在這些環境中，連接器必須以 `No container` (無容器) 模式執行。若要尋找支援 `No container` (無容器) 模式的連接器，請參閱[the section called “AWS提供的 Greengrass 連接器”](#)。有些連接器會以此模式原生執行，而有些連接器可讓您設定隔離模式。

在支援 Greengrass 容器化的環境中，您也可以將隔離模式設定為 No container (無容器)，但我們建議盡可能使用 Greengrass container (Greengrass 容器) 模式。

 Note

Greengrass 群組的預設容器化設定不適用於[連接器](#)。

升級連接器版本

連接器提供者可能會發佈新增功能、修正問題或改善效能的新版連接器。如需可用版本和相關變更的資訊，請參閱[每個連接器的文件](#)。

在 AWS IoT 主控台中，您可以檢查 Greengrass 群組中連接器的新版本。

1. 在 AWS IoT 主控台導覽窗格的管理下，展開 Greengrass 裝置，然後選擇群組 (V1)。
2. 在 Greengrass 群組下，選擇您的群組。
3. 選擇 Connectors (連接器) 以顯示群組中的連接器。

如果連接器有新版本，則升級欄中會顯示可用按鈕。

4. 若要升級連接器版本：
 - a. 在 Connectors (連接器) 頁面的 Upgrade (升級) 欄中，選擇 Available (可用)。Upgrade connector (升級連接器) 頁面隨即開啟，並顯示目前的參數設定 (若適用)。

選擇新的連接器版本，視需要定義參數，然後選擇 Upgrade (升級)。

- b. 在 Subscriptions (訂閱) 頁面上，在群組中新增新訂閱，以取代使用連接器做為來源或目標的任何訂閱。然後，移除舊訂閱。

訂閱會根據版本參考連接器，因此如果您變更群組中的連接器版本，訂閱則會變成無效。

- c. 從 Actions (動作) 功能表中，選擇 Deploy (部署) 以將變更部署至核心。

若要從 AWS IoT Greengrass API 升級連接器，請建立並部署包含已更新連接器和訂閱的群組版本。使用與將連接器新增至群組相同的程序。如需示範如何使用 AWS CLI 來設定和部署範例 Twilio Notifications 連接器的詳細步驟，請參閱 [the section called “連接器入門 \(CLI\)”](#)。

連接器的記錄

Greengrass 連接器包含 Lambda 函數，可將事件和錯誤寫入 Greengrass 日誌。根據您的群組設定，日誌會寫入 CloudWatch Logs、本機檔案系統或兩者。來自連接器的日誌包含對應函數的 ARN。下列範例 ARN 來自 Kinesis Firehose 連接器：

```
arn:aws:lambda:aws-region:account-id:function:KinesisFirehoseClient:1
```

預設記錄組態使用以下目錄結構，將資訊層級日誌寫入檔案系統：

```
greengrass-root/ggc/var/log/user/region/aws/function-name.log
```

如需 Greengrass 記錄的詳細資訊，請參閱 [the section called “使用 AWS IoT Greengrass 日誌進行監控”](#)。

AWS提供的 Greengrass 連接器

AWS 提供下列支援常見 AWS IoT Greengrass 案例的連接器。如需連接器如何運作的詳細資訊，請參閱下列文件：

- [使用連接器整合服務和通訊協定](#)
- [連接器入門 \(主控台\)](#) 或 [連接器入門 \(CLI\)](#)

連接器	描述	支援的 Lambda 執行時間	支援 No container (無容器) 模式
CloudWatch 指標	將自訂指標發佈至 Amazon CloudWatch。	• Python 3.8 * • Python 3.7 • Python 2.7	是
Device Defender	傳送系統指標至 AWS IoT Device Defender。	• Python 3.8 * • Python 3.7 • Python 2.7	否

連接器	描述	支援的 Lambda 執行時間	支援 No container (無容器) 模式
Docker 應用程式部署	執行 Docker Compose 檔案，以在核心裝置上啟動 Docker 應用程式。	- Python 3.8 - Python 3.7	是
IoT Analytics	將資料從裝置和感應器傳送至 AWS IoT Analytics。	- Python 3.8 * - Python 3.7 - Python 2.7	是
IoT 乙太網路 IP 通訊協定轉接器	從乙太網路/IP 裝置收集資料。	Java 8	是
IoT SiteWise	將資料從裝置和感測器傳送至 AWS IoT SiteWise 中的資產屬性。	Java 8	是
Kinesis Firehose	將資料傳送至 Amazon Data Firehose 交付串流。	- Python 3.8 * - Python 3.7 - Python 2.7	是
ML 意見回饋	將機器學習模型輸入發佈至雲端，並將輸出發佈至 MQTT 主題。	- Python 3.8 - Python 3.7	否
ML 影像分類	執行本機映像分類推論服務。此連接器提供適用於多種平台的版本。	- Python 3.8 * - Python 3.7 - Python 2.7	否
ML 物件偵測	執行本機物件偵測推論服務。此連接器提供適用於多種平台的版本。	- Python 3.8 - Python 3.7	否

連接器	描述	支援的 Lambda 執行時間	支援 No container (無容器) 模式
Modbus-RTU 通訊協定轉接器	將請求傳送到 Modbus RTU 裝置。	• Python 3.8 * • Python 3.7 • Python 2.7	否
Modbus-TCP 通訊協定轉接器	從 ModbusTCP 裝置收集資料。	• Java 8	是
Raspberry Pi GPIO	控制 Raspberry Pi 核心裝置上的 GPIO 接腳。	• Python 3.8 * • Python 3.7 • Python 2.7	否
序列串流	讀取和寫入核心裝置的序列埠。	• Python 3.8 * • Python 3.7 • Python 2.7	否
ServiceNow MetricBase 整合	將時間序列指標發佈到 ServiceNow MetricBase。	• Python 3.8 * • Python 3.7 • Python 2.7	是
SNS	傳送訊息至 Amazon SNS 主題。	• Python 3.8 * • Python 3.7 • Python 2.7	是

連接器	描述	支援的 Lambda 執行時間	支援 No container (無容器) 模式
Splunk 整合	將資料發佈到 Splunk HEC。	- Python 3.8 * - Python 3.7 - Python 2.7	是
Twilio 通知	啟動 Twilio 文字或語音訊息。	- Python 3.8 * - Python 3.7 - Python 2.7	是

* 若要使用 Python 3.8 執行期，您必須從預設 Python 3.7 安裝資料夾建立符號連結，以連接至安裝的 Python 3.8 二進位檔。如需詳細資訊，請參閱連接器特定需求。

Note

建議您[升級連接器版本](#)：從 Python 2.7 升級至 Python 3.7。持續支援 Python 2.7 連接器取決於 AWS Lambda 執行時間支援。如需詳細資訊，請參閱《AWS Lambda 開發人員指南》中的[執行期支援政策](#)。

CloudWatch 指標連接器

CloudWatch Metrics [連接器](#) 會將自訂指標從 Greengrass 裝置發佈至 Amazon CloudWatch。連接器提供集中式基礎設施來發佈 CloudWatch 指標，您可以使用它來監控和分析 Greengrass 核心環境，並對本機事件採取行動。如需詳細資訊，請參閱《[Amazon CloudWatch 使用者指南](#)》中的使用 [Amazon CloudWatch 指標](#)。Amazon CloudWatch

此連接器會接收指標資料當做 MQTT 訊息。連接器會批次處理位於相同命名空間中的指標，並定期將其發佈至 CloudWatch。

此連接器具有下列版本。

版本	ARN
5	arn:aws:greengrass: <i>region</i> ::/connectors/CloudWatchMetrics/versions/5
4	arn:aws:greengrass: <i>region</i> ::/connectors/CloudWatchMetrics/versions/4
3	arn:aws:greengrass: <i>region</i> ::/connectors/CloudWatchMetrics/versions/3
2	arn:aws:greengrass: <i>region</i> ::/connectors/CloudWatchMetrics/versions/2
1	arn:aws:greengrass: <i>region</i> ::/connectors/CloudWatchMetrics/versions/1

如需版本變更的詳細資訊，請參閱 [Changelog](#)。

要求

此連接器有下列要求：

Version 3 - 5

- AWS IoT Greengrass 核心軟體 v1.9.3 或更新版本。
- 安裝在核心裝置上並新增至 PATH 環境變數的 [Python](#) 3.7 或 3.8 版。

Note

若要使用 Python 3.8，請執行下列命令，以建立從預設 Python 3.7 安裝資料夾到已安裝 Python 3.8 二進位檔的符號連結。

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

這會設定您的裝置以符合 AWS IoT Greengrass 的 Python 需求。

- 設定為允許 `cloudwatch:PutMetricData` 動作的 [Greengrass 群組角色](#)，如下列範例 AWS Identity and Access Management (IAM) 政策所示。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1528133056761",
      "Action": [
        "cloudwatch:PutMetricData"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

針對群組角色要求，您必須設定角色以授與必要的許可，並確認已將角色新增至群組。如需詳細資訊，請參閱 [the section called “管理群組角色 \(主控台\)”](#) 或 [the section called “管理群組角色 \(CLI\)”](#)。

如需 CloudWatch 許可的詳細資訊，請參閱《IAM 使用者指南》中的 [Amazon CloudWatch 許可參考](#)。

Versions 1 - 2

- AWS IoT Greengrass 核心軟體 v1.7 或更新版本。
- 安裝在核心裝置上並新增至 PATH 環境變數的 [Python](#) 2.7 版。
- 設定為允許 `cloudwatch:PutMetricData` 動作的 [Greengrass 群組角色](#)，如下列範例 AWS Identity and Access Management (IAM) 政策所示。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1528133056761",
      "Action": [
        "cloudwatch:PutMetricData"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

針對群組角色要求，您必須設定角色以授與必要的許可，並確認已將角色新增至群組。如需詳細資訊，請參閱 [the section called “管理群組角色 \(主控台\)”](#) 或 [the section called “管理群組角色 \(CLI\)”](#)。

如需 CloudWatch 許可的詳細資訊，請參閱《IAM 使用者指南》中的 [Amazon CloudWatch 許可參考](#)。

連接器參數

此連接器提供下列參數：

Versions 4 - 5

PublishInterval

對指定的命名空間發佈批次處理指標之前等待的秒數上限。最大值為 900。若要將連接器設定為一收到指標就立刻將其發佈 (不批次處理)，請指定 0。

連接器在相同命名空間中收到 20 個指標後或在指定的間隔後，會發佈至 CloudWatch。

Note

連接器不保證發佈事件的順序。

AWS IoT 主控台中的顯示名稱：發佈間隔

必要：true

類型：string

有效值：0 - 900

有效模式：[0-9]|[1-9]\d|[1-9]\d\d|900

PublishRegion

AWS 區域 要發佈 CloudWatch 指標的。這個值覆寫預設 Greengrass 指標區域。只在張貼跨區域指標時才需要。

AWS IoT 主控台中的顯示名稱：發布區域

必要：false

類型：string

有效模式：^\$|([a-z]{2}-[a-z]+\d{1})

MemorySize

分配給連接器的記憶體 (KB)。

AWS IoT 主控台中的顯示名稱：記憶體大小

必要：true

類型：string

有效模式：^[0-9]+\$

MaxMetricsToRetain

在所有命名空間中，由新指標取代之之前在記憶體中儲存的指標數目上限。最小值為 2000。

在沒有網際網路連線且連接器開始緩衝指標來稍後發佈時，就受此限制。當緩衝區已滿時，新指標會取代最舊指標。指定命名空間中的指標只會由相同命名空間中的指標取代。

Note

如果連接器的主機程序中斷，則不會儲存指標。例如，在群組部署或裝置重新啟動期間就可能發生這種中斷情況。

在 AWS IoT 主控台中顯示名稱：要保留的指標上限

必要：true

類型：string

有效模式：`^([2-9]\d{3}|[1-9]\d{4,})$`

IsolationMode

此連接器的[容器化](#)模式。預設值為 `GreengrassContainer`，這表示連接器會在 AWS IoT Greengrass 容器內的隔離執行時間環境中執行。

Note

群組的預設容器化設定不會套用至連接器。

在 AWS IoT 主控台中顯示名稱：容器隔離模式

必要：false

類型：string

有效值：`GreengrassContainer` 或 `NoContainer`

有效模式：`^NoContainer$|^GreengrassContainer$`

Versions 1 - 3

PublishInterval

對指定的命名空間發佈批次處理指標之前等待的秒數上限。最大值為 900。若要將連接器設定為一收到指標就立刻將其發佈 (不批次處理)，請指定 0。

連接器在相同命名空間中收到 20 個指標後或在指定的間隔後，會發佈至 CloudWatch。

Note

連接器不保證發佈事件的順序。

AWS IoT 主控台顯示名稱：發佈間隔

必要：true

類型：string

有效值：0 - 900

有效模式：[0-9]|[1-9]\d|[1-9]\d\d|900

PublishRegion

AWS 區域 要發佈 CloudWatch 指標的。這個值覆寫預設 Greengrass 指標區域。只在張貼跨區域指標時才需要。

AWS IoT 主控台中的顯示名稱：發布區域

必要：false

類型：string

有效模式：^\$|([a-z]{2}-[a-z]+-\d{1})

MemorySize

分配給連接器的記憶體 (KB)。

AWS IoT 主控台中的顯示名稱：記憶體大小

必要：true

類型：string

有效模式：^[0-9]+\$

MaxMetricsToRetain

在所有命名空間中，由新指標取代之之前在記憶體中儲存的指標數目上限。最小值為 2000。

在沒有網際網路連線且連接器開始緩衝指標來稍後發佈時，就受此限制。當緩衝區已滿時，新指標會取代最舊指標。指定命名空間中的指標只會由相同命名空間中的指標取代。

Note

如果連接器的主機程序中斷，則不會儲存指標。例如，在群組部署或裝置重新啟動期間就可能發生這種中斷情況。

在 AWS IoT 主控台中顯示名稱：要保留的指標上限

必要：true

類型：string

有效模式：`^([2-9]\d{3}|[1-9]\d{4,})$`

建立範例連接器 (AWS CLI)

下列 CLI 命令 `ConnectorDefinition` 會使用包含 CloudWatch Metrics 連接器的初始版本來建立。

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
  "Connectors": [
    {
      "Id": "MyCloudWatchMetricsConnector",
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/CloudWatchMetrics/
versions/4",
      "Parameters": {
        "PublishInterval" : "600",
        "PublishRegion" : "us-west-2",
        "MemorySize" : "16",
        "MaxMetricsToRetain" : "2500",
        "IsolationMode" : "GreengrassContainer"
      }
    }
  ]
}'
```

在 AWS IoT Greengrass 主控台中，您可以從群組的連接器頁面新增連接器。如需詳細資訊，請參閱 [the section called “連接器入門 \(主控台\)”](#)。

輸入資料

此連接器接受 MQTT 主題的指標，並將指標發佈至 CloudWatch。輸入訊息必須是 JSON 格式。

訂閱中的主題篩選條件

`cloudwatch/metric/put`

訊息屬性

request

此訊息中指標的相關資訊。

請求物件包含要發佈至 CloudWatch 的指標資料。指標值必須符合 [PutMetricData](#) API 的規格。只需要 namespace、metricData.metricName 和 metricData.value 屬性。

必要：true

類型：object 包含下列屬性：

namespace

此請求中指標資料的使用者定義命名空間。CloudWatch 使用命名空間做為指標資料點的容器。

Note

您無法指定以預留字串 開頭的命名空間AWS/。

必要：true

類型：string

有效模式：`[^:].*`

metricData

指標的資料。

必要：true

類型：object 包含下列屬性：

metricName

指標的名稱

必要：true

類型：string

dimensions

與指標相關聯的維度。維度提供指標及其資料的詳細資訊。一個指標可以定義最多 10 個維度。

此連接器會自動包含名為 `coreName` 的維度，其中 `coreName` 值是核心的名稱。

必要：false

類型：包含下列屬性 `array` 的維度物件：

`name`

維度名稱。

必要：false

類型：string

`value`

維度值。

必要：false

類型：string

`timestamp`

收到指標資料的時間，以自 1970 年 1 月 1 日 00:00:00 UTC 以來的秒數表示。如果省略這個值，連接器會使用收到訊息的時間。

必要：false

類型：timestamp

Note

如果您在此連接器的版本 1 和 4 之間使用，我們建議您從單一來源傳送多個指標時，分別擷取每個指標的時間戳記。請勿使用變數來存放時間戳記。

`value`

指標的值。

 Note

CloudWatch 會拒絕太小或太大的值。值的範圍必須在 $8.515920e-109$ 到 $1.174271e+108$ (Base 10) 或 $2e-360$ 到 $2e360$ (Base 2) 之間。不支援特殊值 (例如, NaN、+Infinity、-Infinity)。

必要 : true

類型 : double

unit

指標的單位。

必要 : false

類型 : string

有效值 : Seconds, Microseconds, Milliseconds, Bytes, Kilobytes, Megabytes, Gigabytes, Terabytes, Bits, Kilobits, Megabits, Gigabits, Terabits, Percent, Count, Bytes/Second, Kilobytes/Second, Megabytes/Second, Gigabytes/Second, Terabytes/Second, Bits/Second, Kilobits/Second, Megabits/Second, Gigabits/Second, Terabits/Second, Count/Second, None

限制

使用此連接器時，CloudWatch [PutMetricData](#) API 實施的所有限制都適用於指標。以下限制特別重要：

- API 承載有 40 KB 限制
- 每一 API 請求 20 個指標
- PutMetricData API 每秒 150 筆交易 (TPS)

如需詳細資訊，請參閱《Amazon [CloudWatch 使用者指南](#)》中的 [CloudWatch 限制](#)。Amazon CloudWatch

範例輸入

```
{
  "request": {
    "namespace": "Greengrass",
```



```
    "metricData":
      {
        "metricName": "latency",
        "dimensions": [
          {
            "name": "hostname",
            "value": "test_hostname"
          }
        ],
        "timestamp": 1539027324,
        "value": 123.0,
        "unit": "Seconds"
      }
    }
  }
```

輸出資料

這個連接器會將狀態資訊發佈為輸出資料，且主題為 MQTT。

訂閱中的主題篩選條件

cloudwatch/metric/put/status

範例輸出：成功

回應包含指標資料的命名空間，以及 CloudWatch 回應中的 RequestId 欄位。

```
{
  "response": {
    "cloudwatch_rid": "70573243-d723-11e8-b095-75ff2EXAMPLE",
    "namespace": "Greengrass",
    "status": "success"
  }
}
```

範例輸出：失敗

```
{
  "response" : {
    "namespace": "Greengrass",
    "error": "InvalidInputException",
    "error_message": "cw metric is invalid",
  }
}
```

```
    "status": "fail"
  }
}
```

Note

如果連接器偵測到可重試的錯誤（例如連線錯誤），它會在下一個批次中重試發佈。

用量範例

使用下列高階步驟來設定範例 Python 3.7 Lambda 函數，您可以用來嘗試連接器。

Note

- 如果您使用其他 Python 執行期，則可以建立從 Python3.x 到 Python 3.7 的符號連結。
- [連接器入門 \(主控台\)](#) 和 [連接器入門 \(CLI\)](#) 主題包含詳細步驟，說明如何設定和部署範例 Twilio 通知連接器。

1. 確定您符合連接器的[要求](#)。

針對群組角色要求，您必須設定角色以授與必要的許可，並確認已將角色新增至群組。如需詳細資訊，請參閱 [the section called “管理群組角色 \(主控台\)”](#) 或 [the section called “管理群組角色 \(CLI\)”](#)。

2. 建立並發佈 Lambda 函數，將輸入資料傳送至連接器。

將範例程式碼儲存為 PY 檔案。下載並解壓縮[AWS IoT Greengrass 適用於 Python 的 Core SDK](#)。然後，建立在根層級包含 PY 檔案和 greengrasssdk 資料夾的 zip 套件。此 zip 套件是您上傳至的部署套件 AWS Lambda。

建立 Python 3.7 Lambda 函數後，發佈函數版本並建立別名。

3. 設定 Greengrass 群組。

- 依別名新增 Lambda 函數（建議）。將 Lambda 生命週期設定為長期（或在 CLI `"Pinned": true` 中）。
- 新增連接器並設定其[參數](#)。
- 新增訂閱，允許連接器在支援主題篩選條件上接收[輸入資料](#)並傳送[輸出資料](#)。

- 將 Lambda 函數設定為來源，將連接器設定為目標，並使用支援的輸入主題篩選條件。
- 將連接器設為來源、將 AWS IoT Core 設為目標，並使用支援的輸出主題篩選條件。您可以使用此訂閱在 AWS IoT 主控台中檢視狀態訊息。

4. 部署群組。

5. 在 AWS IoT 主控台的測試頁面上，訂閱輸出資料主題，以檢視連接器的狀態訊息。Lambda 函數範例為長期函數，並在部署群組後立即開始傳送訊息。

完成測試後，您可以將 Lambda 生命週期設定為隨需（或在 CLI "Pinned": false 中）並部署群組。這會讓函數停止傳送訊息。

範例

下列範例 Lambda 函數會將輸入訊息傳送至連接器。

```
import greengrasssdk
import time
import json

iot_client = greengrasssdk.client('iot-data')
send_topic = 'cloudwatch/metric/put'

def create_request_with_all_fields():
    return {
        "request": {
            "namespace": "Greengrass_CW_Connector",
            "metricData": {
                "metricName": "Count1",
                "dimensions": [
                    {
                        "name": "test",
                        "value": "test"
                    }
                ],
                "value": 1,
                "unit": "Seconds",
                "timestamp": time.time()
            }
        }
    }
```

```
def publish_basic_message():
    messageToPublish = create_request_with_all_fields()
    print("Message To Publish: ", messageToPublish)
    iot_client.publish(topic=send_topic,
                       payload=json.dumps(messageToPublish))

publish_basic_message()

def lambda_handler(event, context):
    return
```

授權

CloudWatch Metrics 連接器包含下列第三方軟體/授權：

- [適用於 Python \(Boto3\) 的 AWS SDK](#)/Apache License 2.0
- [botocore](#)/Apache License 2.0
- [dateutil](#)/PSF License
- [docutils](#)/BSD 授權、GNU 一般公有授權 (GPL)、Python 軟體基金會授權、公有網域
- [jmespath](#)/MIT License
- [s3transfer](#)/Apache License 2.0
- [urllib3](#)/MIT License

此連接器根據 [Greengrass 核心軟體授權合約](#) 發行。

變更記錄

下表說明每個版本連接器的變更。

版本	變更
5	修正在輸入資料中新增重複時間戳記的支援。
4	已新增 IsolationMode 參數，以設定連接器的容器化模式。
3	將 Lambda 執行時間升級至 Python 3.7，這會變更執行時間需求。

版本	變更
2	可減少過多記錄的修正。
1	初始版本。

Greengrass 群組一次只能包含一個版本的連接器。若要取得有關升級連接器版本的資訊，請參閱[the section called “升級連接器版本”](#)。

另請參閱

- [使用連接器整合服務和通訊協定](#)
- [the section called “連接器入門 \(主控台\)”](#)
- [the section called “連接器入門 \(CLI\)”](#)
- 《[Amazon CloudWatch 使用者指南](#)》中的[使用 Amazon CloudWatch 指標](#) Amazon CloudWatch
- Amazon CloudWatch API 參考中的 [PutMetricData](#)

Device Defender 連接器

Device Defender [連接器](#)會通知管理員 Greengrass 核心裝置狀態的變更。這可協助識別可能會表示裝置受損的不尋常行為。

此連接器會從核心裝置上的 /proc 目錄讀取系統指標，然後將指標發佈至 AWS IoT Device Defender。如需指標報告詳細資訊，請參閱《AWS IoT 開發人員指南》中的[裝置指標文件規格](#)。

此連接器具有下列版本。

版本	ARN
3	arn:aws:greengrass: <i>region</i> ::/connectors/DeviceDefender/versions/3
2	arn:aws:greengrass: <i>region</i> ::/connectors/DeviceDefender/versions/2

版本	ARN
1	arn:aws:greengrass: <i>region</i> ::/connectors/DeviceDefender/versions/1

如需版本變更的詳細資訊，請參閱 [Changelog](#)。

要求

此連接器有下列要求：

Version 3

- AWS IoT Greengrass 核心軟體 1.9.3 版或更新版本。
- 安裝在核心裝置上並新增至 PATH 環境變數的 [Python](#) 3.7 或 3.8 版。

Note

若要使用 Python 3.8，請執行下列命令，從預設 Python 3.7 安裝資料夾建立符號連結，以連接至已安裝的 Python 3.8 二進位檔。

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

這會設定您的裝置以符合 AWS IoT Greengrass 的 Python 需求。

- AWS IoT Device Defender 設定為使用 Detect 功能來追蹤違規。如需詳細資訊，請參閱《AWS IoT 開發人員指南》中的 [偵測](#)。
- Greengrass 群組中指向 /proc 目錄的 [本機磁碟區資源](#)。資源必須使用下列屬性：
 - 來源路徑：/proc
 - 目的地路徑：/host_proc (或符合 [有效模式](#) 的值)
 - AutoAddGroupOwner：true
- 安裝在 Greengrass 核心上的 [psutil](#) 程式庫。5.7.0 版是經過驗證，可與連接器搭配使用的最新版本。
- 安裝在 Greengrass 核心上的 [cbor](#) 程式庫。1.0.0 版是經過驗證，可與連接器搭配使用的最新版本。

Versions 1 - 2

- AWS IoT Greengrass 核心軟體 1.7 版或更新版本。
- 核心裝置上安裝並新增至 PATH 環境變數的 [Python 2.7](#) 版。
- AWS IoT Device Defender 設定為使用 Detect 功能來追蹤違規。如需詳細資訊，請參閱《AWS IoT 開發人員指南》中的[偵測](#)。
- Greengrass 群組中指向 /proc 目錄的[本機磁碟區資源](#)。資源必須使用下列屬性：
 - 來源路徑：/proc
 - 目的地路徑：/host_proc (或符合[有效模式](#)的值)
 - AutoAddGroupOwner：true
- 安裝在 Greengrass 核心上的 [psutil](#) 程式庫。
- 安裝在 Greengrass 核心上的 [cbor](#) 程式庫。

連接器參數

此連接器提供下列參數：

SampleIntervalSeconds

蒐集和報告指標每個週期之間的秒數。最低值為 300 秒 (5 分鐘)。

在 AWS IoT 主控台中顯示名稱：指標報告間隔

必要：true

類型：string

有效模式：`^[0-9]*(?:3[0-9][0-9]|[4-9][0-9]{2}|[1-9][0-9]{3,})$`

ProcDestinationPath-ResourceId

/proc 磁碟區資源的 ID。

Note

此連接器已授予資源的唯讀存取權。

AWS IoT 主控台顯示名稱：/proc 目錄的資源

必要：true

類型：string

有效模式：[a-zA-Z0-9_-]+

ProcDestinationPath

/proc 磁碟區資源的目的地路徑。

在 AWS IoT 主控台中顯示名稱：/proc 資源的目的地路徑

必要：true

類型：string

有效模式：\[a-zA-Z0-9_-]+

建立範例連接器 (AWS CLI)

下列 CLI 命令會使用包含 Device Defender 連接器的ConnectorDefinition初始版本建立。

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
  "Connectors": [
    {
      "Id": "MyDeviceDefenderConnector",
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/DeviceDefender/
versions/3",
      "Parameters": {
        "SampleIntervalSeconds": "600",
        "ProcDestinationPath": "/host_proc",
        "ProcDestinationPath-ResourceId": "my-proc-resource"
      }
    }
  ]
}'
```

Note

此連接器中的 Lambda 函數具有[長期生命週期](#)。

在 AWS IoT Greengrass 主控台中，您可以從群組的連接器頁面新增連接器。如需詳細資訊，請參閱[the section called “連接器入門 \(主控台\)”](#)。

輸入資料

此連接器不接受 MQTT 訊息做為輸入資料。

輸出資料

此連接器會將安全指標發佈至 AWS IoT Device Defender，做為輸出資料。

訂閱中的主題篩選條件

`$aws/things/+/defender/metrics/json`

Note

這是 AWS IoT Device Defender 預期的主題語法。連接器會將 + 萬用字元換成裝置名稱 (例如，`$aws/things/thing-name/defender/metrics/json`)。

範例輸出

如需指標報告詳細資訊，請參閱《AWS IoT 開發人員指南》中的[裝置指標文件規格](#)。

```
{
  "header": {
    "report_id": 1529963534,
    "version": "1.0"
  },
  "metrics": {
    "listening_tcp_ports": {
      "ports": [
        {
          "interface": "eth0",
          "port": 24800
        },
        {
          "interface": "eth0",
          "port": 22
        },
        {
          "interface": "eth0",
```

```

        "port": 53
      }
    ],
    "total": 3
  },
  "listening_udp_ports": {
    "ports": [
      {
        "interface": "eth0",
        "port": 5353
      },
      {
        "interface": "eth0",
        "port": 67
      }
    ],
    "total": 2
  },
  "network_stats": {
    "bytes_in": 1157864729406,
    "bytes_out": 1170821865,
    "packets_in": 693092175031,
    "packets_out": 738917180
  },
  "tcp_connections": {
    "established_connections": {
      "connections": [
        {
          "local_interface": "eth0",
          "local_port": 80,
          "remote_addr": "192.168.0.1:8000"
        },
        {
          "local_interface": "eth0",
          "local_port": 80,
          "remote_addr": "192.168.0.1:8000"
        }
      ],
      "total": 2
    }
  }
}

```

授權

此連接器根據 [Greengrass 核心軟體授權合約](#) 發行。

變更記錄

下表說明每個連接器版本中的變更。

版本	變更
3	將 Lambda 執行時間升級至 Python 3.7，這會變更執行時間需求。
2	可減少過多記錄的修正。
1	初始版本。

Greengrass 群組一次只能包含一個版本的連接器。若要取得有關升級連接器版本的資訊，請參閱[the section called “升級連接器版本”](#)。

另請參閱

- [使用連接器整合服務和通訊協定](#)
- [the section called “連接器入門 \(主控台\)”](#)
- [the section called “連接器入門 \(CLI\)”](#)
- 《AWS IoT 開發人員指南》中的 [Device Defender](#)

Docker 應用程式部署連接器

Greengrass Docker 應用程式部署連接器可讓您更輕鬆地在 AWS IoT Greengrass 核心上執行 Docker 映像。連接器使用 Docker Compose 從 `docker-compose.yml` 檔案啟動多容器 Docker 應用程式。具體而言，連接器會執行 `docker-compose` 命令來管理單一核心裝置上的 Docker 容器。如需詳細資訊，請參閱 Docker 文件中的 [Docker Compose 概觀](#)。連接器可以存取存放在 Docker 容器登錄檔中的 Docker 映像，例如 Amazon Elastic Container Registry (Amazon ECR)、Docker Hub 和私有 Docker 信任登錄檔。

部署 Greengrass 群組之後，連接器會提取最新的映像並啟動 Docker 容器。它會執行 `docker-compose pull` 和 `docker-compose up` 命令。然後，連接器會將命令的狀態發佈至[輸出 MQTT 主](#)

[題](#)。它也會記錄有關執行 Docker 容器的狀態資訊。這可讓您在 Amazon CloudWatch 中監控應用程式日誌。如需詳細資訊，請參閱[the section called “使用 AWS IoT Greengrass 日誌進行監控”](#)。每次 Greengrass 協助程式重新啟動時，連接器也會啟動 Docker 容器。可在核心上執行的 Docker 容器數量取決於您的硬體。

Docker 容器須在核心設備上的 Greengrass 網域外執行，因此無法存取核心的處理程序間通訊 (IPC)。不過，您可以使用 Greengrass 元件設定一些通訊通道，例如本機 Lambda 函數。如需詳細資訊，請參閱[the section called “與 Docker 容器通訊”](#)。

您可以使用連接器來處理在核心裝置上託管 Web 伺服器或 MySQL 伺服器等案例。Docker 應用程式中的本機服務可以互相通訊，也能和本機環境中的其他程序以及雲端服務通訊。例如，您可以在核心上執行 Web 伺服器，將請求從 Lambda 函數傳送至雲端中的 Web 服務。

此連接器會以[無容器](#)隔離模式執行，因此您可以將其部署到在沒有 Greengrass 容器化的情況下執行的 Greengrass 群組。

此連接器具有下列版本。

版本	ARN
7	arn:aws:greengrass: <i>region</i> ::/connectors/DockerApplicationDeployment/versions/7
6	arn:aws:greengrass: <i>region</i> ::/connectors/DockerApplicationDeployment/versions/6
5	arn:aws:greengrass: <i>region</i> ::/connectors/DockerApplicationDeployment/versions/5
4	arn:aws:greengrass: <i>region</i> ::/connectors/DockerApplicationDeployment/versions/4
3	arn:aws:greengrass: <i>region</i> ::/connectors/DockerApplicationDeployment/versions/3

版本	ARN
2	arn:aws:greengrass: <i>region</i> ::/connectors/DockerApplicationDeployment/versions/2
1	arn:aws:greengrass: <i>region</i> ::/connectors/DockerApplicationDeployment/versions/1

如需版本變更的詳細資訊，請參閱 [Changelog](#)。

要求

此連接器有下列要求：

- AWS IoT Greengrass 核心軟體 v1.10 或更新版本。

Note

OpenWrt 發行版本不支援此連接器。

- 安裝在核心裝置上並新增至 PATH 環境變數的 [Python](#) 3.7 或 3.8 版。

Note

若要使用 Python 3.8，請執行下列命令，以建立從預設 Python 3.7 安裝資料夾到已安裝 Python 3.8 二進位檔的符號連結。

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

這會設定您的裝置以符合 AWS IoT Greengrass 的 Python 需求。

- Greengrass 核心上至少要有 36 MB RAM，以供連接器監控執行 Docker 容器。總記憶體需求取決於核心上執行的 Docker 容器數目。
- 安裝在 Greengrass 核心上的 [Docker Engine](#) 1.9.1 或更新版本。19.0.3 版是經過驗證，可與連接器搭配使用的最新版本。

docker 可執行檔必須位於 /usr/bin 或 /usr/local/bin 目錄中。

⚠ Important

建議您安裝登入資料存放區，以保護 Docker 登入資料的本機複本。如需詳細資訊，請參閱[the section called “安全注意事項”](#)。

如需在 Amazon Linux 發行版本上安裝 Docker 的詳細資訊，請參閱《Amazon Elastic Container Service [開發人員指南](#)》中的 [Amazon ECS 的 Docker 基本概念](#)。

- 安裝在 Greengrass 核心上的 [Docker Compose](#)。docker-compose 可執行檔必須位於 /usr/bin 或 /usr/local/bin 目錄中。

下列 Docker Compose 版本經過驗證，可與連接器搭配使用。

連接器版本	經過驗證的 Docker Compose 版本
7	1.25.4
6	1.25.4
5	1.25.4
4	1.25.4
3	1.25.4
2	1.25.1
1	1.24.1

- 存放在 Amazon Simple Storage Service (Amazon S3) `docker-compose.yml` 中的單一 Docker Compose 檔案（例如）。格式必須與安裝在核心上的 Docker Compose 版本相容。在您的核心上使用該文件前，請先測試該文件。如果您在部署 Greengrass 群組後才編輯檔案，則必須重新部署群組，以更新核心上的本機複本。
- 具有呼叫本機 Docker 協助程式之權限的 Linux 使用者，以及寫入儲存 Compose 檔案本機複本的目錄。如需詳細資訊，請參閱[在核心上設定 Docker 使用者](#)。

- 已將 [Greengrass 群組角色](#) 設定為允許在包含 Compose 檔案的 S3 儲存貯體上執行 `s3:GetObject` 動作。此許可會顯示在下列範例 IAM 政策中。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowAccessToComposeFileS3Bucket",
      "Action": [
        "s3:GetObject",
        "s3:GetObjectVersion"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::bucket-name/*"
    }
  ]
}
```

Note

如果您的 S3 儲存貯體已啟用版本控制，則必須設定 角色以允許 `s3:GetObjectVersion` 動作。如需詳細資訊，請參閱《Amazon Simple Storage Service 使用者指南》中的 [使用版本控制](#)。

針對群組角色要求，您必須設定角色以授與必要的許可，並確認已將角色新增至群組。如需詳細資訊，請參閱 [the section called “管理群組角色 \(主控台\)”](#) 或 [the section called “管理群組角色 \(CLI\)”](#)。

- 如果您的 Docker Compose 檔案參考存放在 Amazon ECR 中的 Docker 映像，則 [Greengrass 群組角色](#) 會設定為允許下列項目：
 - `ecr:GetDownloadUrlForLayer` Amazon ECR 儲存庫上包含 Docker 映像的 和 `ecr:BatchGetImage` 動作。
 - 您資源上的 `ecr:GetAuthorizationToken` 動作。

儲存庫必須與連接器位於相同的 AWS 帳戶 和 AWS 區域。

⚠ Important

群組角色中的許可可由 Greengrass 群組中的所有 Lambda 函數和連接器擔任。如需詳細資訊，請參閱[the section called “安全注意事項”](#)。

這些權限會顯示在下列範例政策中。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowGetEcrRepositories",
      "Effect": "Allow",
      "Action": [
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage"
      ],
      "Resource": [
        "arn:aws:ecr:us-east-1:123456789012:repository/repository-name"
      ]
    },
    {
      "Sid": "AllowGetEcrAuthToken",
      "Effect": "Allow",
      "Action": "ecr:GetAuthorizationToken",
      "Resource": "*"
    }
  ]
}
```

如需詳細資訊，請參閱《[Amazon ECR 使用者指南](#)》中的 [Amazon ECR 儲存庫政策範例](#)。

針對群組角色要求，您必須設定角色以授與必要的許可，並確認已將角色新增至群組。如需詳細資訊，請參閱 [the section called “管理群組角色 \(主控台\)”](#) 或 [the section called “管理群組角色 \(CLI\)”](#)。

- 如果您的 Docker Compose 檔案從 [AWS Marketplace](#) 中參考 Docker 影像，則連接器也會有下列需求：

- 您必須訂閱 AWS Marketplace 容器產品。如需詳細資訊，請參閱《AWS Marketplace 訂閱者指南》中的[尋找和訂閱容器產品](#)。
- AWS IoT Greengrass 必須設定為支援本機秘密，如[秘密要求](#)中所述。連接器只會使用此功能從擷取您的秘密 AWS Secrets Manager，而不是存放秘密。
- 您必須在 Secrets Manager 中為儲存 Compose 檔案中參考之 Docker 映像的每個 AWS Marketplace 登錄檔建立秘密。如需詳細資訊，請參閱[the section called “從私有儲存庫存取 Docker 影像”](#)。
- 如果您的 Docker Compose 檔案參考 Amazon ECR 以外的登錄檔中私有儲存庫的 Docker 映像，例如 Docker Hub，連接器也具有下列需求：
 - AWS IoT Greengrass 必須設定為支援本機秘密，如[秘密要求](#)中所述。連接器只會使用此功能從擷取您的秘密 AWS Secrets Manager，而不是存放秘密。
 - 您必須在 Secrets Manager 中為儲存 Compose 檔案中參考之 Docker 映像的每個私有儲存庫建立秘密。如需詳細資訊，請參閱[the section called “從私有儲存庫存取 Docker 影像”](#)。
- 部署包含此連接器的 Greengrass 群組時，必須執行 Docker 協助程式。

從私有儲存庫存取 Docker 影像

如果您使用登入資料來存取 Docker 影像，則必須允許連接器存取它們。此作法取決於 Docker 影像的位置。

對於存放 Amazon ECR 的 Docker 映像，您授予在 Greengrass 群組角色中取得授權字符的許可。如需詳細資訊，請參閱[the section called “要求”](#)。

對於存放在其他私有儲存庫或登錄檔中的 Docker 映像，您必須在 中建立秘密 AWS Secrets Manager，以存放您的登入資訊。這包括您訂閱的 Docker 映像 AWS Marketplace。為每個儲存庫建立一個秘密。如果您在 Secrets Manager 中更新秘密，則變更會在您下次部署群組時傳播至核心。

Note

Secrets Manager 是一項服務，可用來在 中安全地存放和管理登入資料、金鑰和其他秘密 AWS 雲端。如需詳細資訊，請參閱AWS Secrets Manager 《使用者指南》中的[什麼是 AWS Secrets Manager ?](#)。

每個秘密都必須包含下列金鑰：

金鑰	值
username	用來存取儲存庫或登錄檔的使用者名稱。
password	用來存取儲存庫或登錄的密碼。
registryUrl	登錄檔的端點。這必須符合 Compose 檔案中對應的登錄檔 URL。

Note

若要允許 根據預設 AWS IoT Greengrass 存取秘密，秘密的名稱必須以 greengrass- 開頭。否則，您的 Greengrass 服務角色必須授與存取權。如需詳細資訊，請參閱[the section called “允許 AWS IoT Greengrass 取得秘密值”](#)。

從 AWS Marketplace 取得 Docker 影像的登入資訊

1. 使用 `aws ecr get-login-password` 命令 AWS Marketplace 從 取得 Docker 映像的密碼。如需詳細資訊，請參閱《AWS CLI 命令參考》中的 [get-login-password](#)。

```
aws ecr get-login-password
```

2. 擷取 Docker 映像的登錄 URL。開啟 AWS Marketplace 網站，並開啟容器產品啟動頁面。在容器映像下，選擇檢視容器映像詳細資訊以尋找使用者名稱和登錄 URL。

使用擷取的使用者名稱、密碼和登錄檔 URL，為儲存 Compose 檔案中參考之 Docker 映像的每個 AWS Marketplace 登錄檔建立秘密。

建立秘密 (主控台)

在 AWS Secrets Manager 主控台中，選擇其他類型的秘密。在 Specify the key/value pairs to be stored for this secret (指定此秘密要儲存的索引鍵/值組) 下，新增 username、password 和 registryUrl 的列。如需詳細資訊，請參閱 AWS Secrets Manager 《使用者指南》中的[建立基本秘密](#)。

Specify the key/value pairs to be stored in this secret [Info](#)

Secret key/value	Plaintext	
username	Mary_Major	Remove
password	abc123xyz456	Remove
registryUrl	https://docker.io	Remove

[+ Add row](#)

建立秘密 (CLI)

在 AWS CLI 中，使用 Secrets Manager `create-secret` 命令，如下列範例所示。如需詳細資訊，請參閱《AWS CLI 命令參考》中的 [create-secret](#)。

```
aws secretsmanager create-secret --name greengrass-MySecret --secret-string [{"username":"Mary_Major"}, {"password":"abc123xyz456"}, {"registryUrl":"https://docker.io"}]
```

Important

您有責任保護儲存 Docker Compose 檔案的 `DockerComposeFileDestinationPath` 目錄以及私有儲存庫的 Docker 影像登入資料的安全。如需詳細資訊，請參閱 [the section called “安全注意事項”](#)。

Parameters

此連接器提供下列參數：

Version 7

DockerComposeFileS3Bucket

包含 Docker Compose 檔案的 S3 儲存貯體名稱。當您建立儲存貯體時，請務必遵循 Amazon Simple Storage Service 使用者指南中所述的[儲存貯體名稱規則](#)。

在 AWS IoT 主控台中顯示名稱：S3 中的 Docker Compose 檔案

 Note

在主控台中，Docker Compose file in S3 (S3 中的 Docker Compose 檔案) 屬性會結合 DockerComposeFileS3Bucket、DockerComposeFileS3Key 和 DockerComposeFileS3Version 參數。

必要：true

類型：string

有效模式 [a-zA-Z0-9\\-\\.]{3,63}

DockerComposeFileS3Key

Amazon S3 中 Docker Compose 檔案的物件金鑰。如需詳細資訊，包括物件金鑰命名準則，請參閱《Amazon Simple Storage Service 使用者指南》中的[物件金鑰和中繼資料](#)。

 Note

在主控台中，Docker Compose file in S3 (S3 中的 Docker Compose 檔案) 屬性會結合 DockerComposeFileS3Bucket、DockerComposeFileS3Key 和 DockerComposeFileS3Version 參數。

必要：true

類型：string

有效模式 .+

DockerComposeFileS3Version

Amazon S3 中 Docker Compose 檔案的物件版本。如需詳細資訊，包括物件金鑰命名準則，請參閱《Amazon Simple Storage Service 使用者指南》中的[使用版本控制](#)。

Note

在主控台中，Docker Compose file in S3 (S3 中的 Docker Compose 檔案) 屬性會結合 DockerComposeFileS3Bucket、DockerComposeFileS3Key 和 DockerComposeFileS3Version 參數。

必要：false

類型：string

有效模式 .+

DockerComposeFileDestinationPath

用來儲存 Docker Compose 檔案複本的本機目錄絕對路徑。這必須是現有目錄。DockerUserId 的指定使用者必須具有在此目錄中建立檔案的權限。如需詳細資訊，請參閱[the section called “在核心上設定 Docker 使用者”](#)。

Important

儲存 Docker Compose 檔案的目錄以及私有儲存庫的 Docker 影像登入資料的安全。您有責任保護此目錄。如需詳細資訊，請參閱[the section called “安全注意事項”](#)。

在 AWS IoT 主控台中顯示名稱：本機 Compose 檔案的目錄路徑

必要：true

類型：string

有效模式 \/.*\/?

範例：/home/username/myCompose

DockerUserId

連接器用於執行的 Linux 使用者 UID。此使用者必須屬於核心裝置上的 docker Linux 群組，並具有 DockerComposeFileDestinationPath 目錄的寫入權限。如需詳細資訊，請參閱[在核心上設定 Docker 使用者](#)。

Note

我們建議您除非必要，避免以 root 身分執行。如果您指定根使用者，則必須允許 Lambda 函數以根身分在 AWS IoT Greengrass 核心上執行。如需詳細資訊，請參閱[the section called “執行 Lambda 函數做為根”](#)。

在 AWS IoT 主控台中顯示名稱：Docker 使用者 ID

必要：false

類型：string

有效模式：`^[0-9]{1,5}$`

AWSecretsArnList

中秘密的 Amazon Resource Name (ARNs)，AWS Secrets Manager 其中包含用於存取私有儲存庫中 Docker 映像的登入資訊。如需詳細資訊，請參閱[the section called “從私有儲存庫存取 Docker 映像”](#)。

在 AWS IoT 主控台中顯示名稱：私有儲存庫的登入資料

必要：false。此參數需要存取儲存在私有儲存庫中的 Docker 映像。

類型：array 的 string

有效模式：`[( ? , ? ? "(arn:(aws(-[a-z]+)):secretsmanager:[a-z0-9-]+:[0-9]{12}:secret:([a-zA-Z0-9\]+/)[a-zA-Z0-9/_+=, .@-]+-[a-zA-Z0-9]+))"]`

DockerContainerStatusLogFrequency

連接器記錄有關核心上 Docker 容器執行狀態資訊的頻率 (秒)。預設值為 300 秒 (5 分鐘)。

AWS IoT 主控台顯示名稱：記錄頻率

必要：false

類型：string

有效模式：`^[1-9]{1}[0-9]{0,3}$`

ForceDeploy

指出是否因為上次部署的不當清除而使 Docker 部署失敗。預設值為 False。

在 AWS IoT 主控台中顯示名稱：強制部署

必要：false

類型：string

有效模式：`^(true|false)$`

DockerPullBeforeUp

指出部署器是否應在執行 `docker-compose pull` 之前執行 `docker-compose up` 以進行下拉式行為。pull-down-up 預設值為 True。

在 AWS IoT 主控台中顯示名稱：Docker Pull Before Up

必要：false

類型：string

有效模式：`^(true|false)$`

StopContainersOnNewDeployment

指出連接器在 GGC 停止時是否應停止 Docker Deployer 受管 Docker 容器（在部署新群組或核心關閉時，GGC 會停止）。預設值為 True。

在 AWS IoT 主控台中顯示名稱：Docker 在新部署時停止

Note

我們建議將此參數設定為預設值 True。的參數 False 會導致您的 Docker 容器在終止 AWS IoT Greengrass 核心或啟動新部署後繼續執行。如果您將此參數設定為 False，則必須確保在 `docker-compose` 服務名稱變更或新增時，視需要維護 Docker 容器。

如需詳細資訊，請參閱 `docker-compose compose` 檔案文件。

必要：false

類型：string

有效模式：`^(true|false)$`

DockerOfflineMode

指出在離線 AWS IoT Greengrass 啟動時是否使用現有的 Docker Compose 檔案。預設值為 False。

必要：false

類型：string

有效模式：`^(true|false)$`

Version 6

DockerComposeFileS3Bucket

包含 Docker Compose 檔案的 S3 儲存貯體名稱。當您建立儲存貯體時，請務必遵循 Amazon Simple Storage Service 使用者指南中所述的[儲存貯體名稱規則](#)。

在 AWS IoT 主控台中顯示名稱：S3 中的 Docker Compose 檔案

Note

在主控台中，Docker Compose file in S3 (S3 中的 Docker Compose 檔案) 屬性會結合 `DockerComposeFileS3Bucket`、`DockerComposeFileS3Key` 和 `DockerComposeFileS3Version` 參數。

必要：true

類型：string

有效模式 `[a-zA-Z0-9\\-\\.]{3,63}`

DockerComposeFileS3Key

Amazon S3 中 Docker Compose 檔案的物件金鑰。如需詳細資訊，包括物件金鑰命名準則，請參閱《Amazon Simple Storage Service 使用者指南》中的[物件金鑰和中繼資料](#)。

Note

在主控台中，Docker Compose file in S3 (S3 中的 Docker Compose 檔案) 屬性會結合 DockerComposeFileS3Bucket、DockerComposeFileS3Key 和 DockerComposeFileS3Version 參數。

必要：true

類型：string

有效模式：.+

DockerComposeFileS3Version

Amazon S3 中 Docker Compose 檔案的物件版本。如需詳細資訊，包括物件金鑰命名準則，請參閱《Amazon Simple Storage Service 使用者指南》中的[使用版本控制](#)。

Note

在主控台中，Docker Compose file in S3 (S3 中的 Docker Compose 檔案) 屬性會結合 DockerComposeFileS3Bucket、DockerComposeFileS3Key 和 DockerComposeFileS3Version 參數。

必要：false

類型：string

有效模式：.+

DockerComposeFileDestinationPath

用來儲存 Docker Compose 檔案複本的本機目錄絕對路徑。這必須是現有目錄。DockerUserId 的指定使用者必須具有在此目錄中建立檔案的權限。如需詳細資訊，請參閱[the section called “在核心上設定 Docker 使用者”](#)。

 Important

儲存 Docker Compose 檔案的目錄以及私有儲存庫的 Docker 影像登入資料的安全。您有責任保護此目錄。如需詳細資訊，請參閱[the section called “安全注意事項”](#)。

在 AWS IoT 主控台中顯示名稱：本機 Compose 檔案的目錄路徑

必要：true

類型：string

有效模式：\/.*\/?

範例：/home/username/myCompose

DockerUserId

連接器用於執行的 Linux 使用者 UID。此使用者必須屬於核心裝置上的 docker Linux 群組，並具有 DockerComposeFileDestinationPath 目錄的寫入權限。如需詳細資訊，請參閱[在核心上設定 Docker 使用者](#)。

 Note

我們建議您除非必要，避免以 root 身分執行。如果您指定根使用者，則必須允許 Lambda 函數以根身分在 AWS IoT Greengrass 核心上執行。如需詳細資訊，請參閱[the section called “執行 Lambda 函數做為根”](#)。

在 AWS IoT 主控台中顯示名稱：Docker 使用者 ID

必要：false

類型：string

有效模式：^[0-9]{1,5}\$

AWSecretsArnList

中秘密的 Amazon Resource Name (ARNs)，AWS Secrets Manager 其中包含用於存取私有儲存庫中 Docker 映像的登入資訊。如需詳細資訊，請參閱[the section called “從私有儲存庫存取 Docker 映像”](#)。

AWS IoT 主控台顯示名稱：私有儲存庫的登入資料

必要：false。此參數需要存取儲存在私有儲存庫中的 Docker 影像。

類型：array 的 string

有效模式：[(? , ? ? "(arn:(aws(-[a-z]+)):secretsmanager:[a-z0-9-]+:[0-9]{12}:secret:([a-zA-Z0-9\]+/)[a-zA-Z0-9/_+=, .@-]+-[a-zA-Z0-9]+)"]

DockerContainerStatusLogFrequency

連接器記錄有關核心上 Docker 容器執行狀態資訊的頻率 (秒)。預設值為 300 秒 (5 分鐘)。

在 AWS IoT 主控台中顯示名稱：記錄頻率

必要：false

類型：string

有效模式：^[1-9]{1}[0-9]{0,3}\$

ForceDeploy

指出是否因為上次部署的不當清除而使 Docker 部署失敗。預設值為 False。

在 AWS IoT 主控台中顯示名稱：強制部署

必要：false

類型：string

有效模式：^(true|false)\$

DockerPullBeforeUp

指出部署器是否應在執行 docker-compose pull 之前執行 docker-compose up 以進行下拉式行為。pull-down-up 預設值為 True。

AWS IoT 主控台顯示名稱：Docker Pull Before Up

必要：false

類型：string

有效模式：^(true|false)\$

StopContainersOnNewDeployment

指出連接器在 GGC 停止時（建立新群組部署時，或核心關閉時），是否應停止 Docker Deployer 受管 Docker 容器。預設值為 True。

在 AWS IoT 主控台中顯示名稱：在新部署時停止 Docker

Note

我們建議將此參數設定為預設值 True。的參數 False 會導致您的 Docker 容器在終止 AWS IoT Greengrass 核心或啟動新部署後繼續執行。如果您將此參數設定為 False，則必須確保在 docker-compose 服務名稱變更或新增時，視需要維護 Docker 容器。如需詳細資訊，請參閱 docker-compose compose 檔案文件。

必要：false

類型：string

有效模式：^(true|false)\$

Version 5

DockerComposeFileS3Bucket

包含 Docker Compose 檔案的 S3 儲存貯體名稱。當您建立儲存貯體時，請務必遵循 Amazon Simple Storage Service 使用者指南中所述的[儲存貯體名稱規則](#)。

在 AWS IoT 主控台中顯示名稱：S3 中的 Docker Compose 檔案

Note

在主控台中，Docker Compose file in S3 (S3 中的 Docker Compose 檔案) 屬性會結合 DockerComposeFileS3Bucket、DockerComposeFileS3Key 和 DockerComposeFileS3Version 參數。

必要：true

類型：string

有效模式 [a-zA-Z0-9\\-\\.]{3,63}

DockerComposeFileS3Key

Amazon S3 中 Docker Compose 檔案的物件金鑰。如需詳細資訊，包括物件金鑰命名準則，請參閱《Amazon Simple Storage Service 使用者指南》中的[物件金鑰和中繼資料](#)。

Note

在主控台中，Docker Compose file in S3 (S3 中的 Docker Compose 檔案) 屬性會結合 DockerComposeFileS3Bucket、DockerComposeFileS3Key 和 DockerComposeFileS3Version 參數。

必要：true

類型：string

有效模式 .+

DockerComposeFileS3Version

Amazon S3 中 Docker Compose 檔案的物件版本。如需詳細資訊，包括物件金鑰命名準則，請參閱《Amazon Simple Storage Service 使用者指南》中的[使用版本控制](#)。

Note

在主控台中，Docker Compose file in S3 (S3 中的 Docker Compose 檔案) 屬性會結合 DockerComposeFileS3Bucket、DockerComposeFileS3Key 和 DockerComposeFileS3Version 參數。

必要：false

類型：string

有效模式 .+

DockerComposeFileDestinationPath

用來儲存 Docker Compose 檔案複本的本機目錄絕對路徑。這必須是現有目錄。DockerUserId 的指定使用者必須具有在此目錄中建立檔案的權限。如需詳細資訊，請參閱[the section called “在核心上設定 Docker 使用者”](#)。

 Important

儲存 Docker Compose 檔案的目錄以及私有儲存庫的 Docker 影像登入資料的安全。您有責任保護此目錄。如需詳細資訊，請參閱[the section called “安全注意事項”](#)。

在 AWS IoT 主控台中顯示名稱：本機 Compose 檔案的目錄路徑

必要：true

類型：string

有效模式：\/.*\/?

範例：/home/username/myCompose

DockerUserId

連接器用於執行的 Linux 使用者 UID。此使用者必須屬於核心裝置上的 docker Linux 群組，並具有 DockerComposeFileDestinationPath 目錄的寫入權限。如需詳細資訊，請參閱[在核心上設定 Docker 使用者](#)。

 Note

我們建議您除非必要，避免以 root 身分執行。如果您指定根使用者，則必須允許 Lambda 函數以根身分在 AWS IoT Greengrass 核心上執行。如需詳細資訊，請參閱[the section called “執行 Lambda 函數做為根”](#)。

在 AWS IoT 主控台中顯示名稱：Docker 使用者 ID

必要：false

類型：string

有效模式：^[0-9]{1,5}\$

AWSecretsArnList

中秘密的 Amazon Resource Name (ARNs)，AWS Secrets Manager 其中包含用於存取私有儲存庫中 Docker 映像的登入資訊。如需詳細資訊，請參閱[the section called “從私有儲存庫存取 Docker 影像”](#)。

AWS IoT 主控台顯示名稱：私有儲存庫的登入資料

必要：false。此參數需要存取儲存在私有儲存庫中的 Docker 影像。

類型：array 的 string

有效模式：[(?,""(arn:(aws(-[a-z]+)):secretsmanager:[a-z0-9-]+:[0-9]{12}:secret:([a-zA-Z0-9\]+/)[a-zA-Z0-9/_+=,.@-]+-[a-zA-Z0-9]+)))]

DockerContainerStatusLogFrequency

連接器記錄有關核心上 Docker 容器執行狀態資訊的頻率 (秒)。預設值為 300 秒 (5 分鐘)。

在 AWS IoT 主控台中顯示名稱：記錄頻率

必要：false

類型：string

有效模式：^[1-9]{1}[0-9]{0,3}\$

ForceDeploy

指出是否因為上次部署的不當清除而使 Docker 部署失敗。預設值為 False。

AWS IoT 主控台顯示名稱：強制部署

必要：false

類型：string

有效模式：^(true|false)\$

DockerPullBeforeUp

指出部署器是否應在執行 docker-compose pull 之前執行 docker-compose up 以進行下拉式行為。pull-down-up 預設值為 True。

在 AWS IoT 主控台中顯示名稱：Docker Pull Before Up

必要：false

類型：string

有效模式：`^(true|false)$`

Versions 2 - 4

DockerComposeFileS3Bucket

包含 Docker Compose 檔案的 S3 儲存貯體名稱。當您建立儲存貯體時，請務必遵循 Amazon Simple Storage Service 使用者指南中所述的[儲存貯體名稱規則](#)。

在 AWS IoT 主控台中顯示名稱：S3 中的 Docker Compose 檔案

Note

在主控台中，Docker Compose file in S3 (S3 中的 Docker Compose 檔案) 屬性會結合 DockerComposeFileS3Bucket、DockerComposeFileS3Key 和 DockerComposeFileS3Version 參數。

必要：`true`

類型：`string`

有效模式 `[a-zA-Z0-9\\-\\.]{3,63}`

DockerComposeFileS3Key

Amazon S3 中 Docker Compose 檔案的物件金鑰。如需詳細資訊，包括物件金鑰命名準則，請參閱《Amazon Simple Storage Service 使用者指南》中的[物件金鑰和中繼資料](#)。

Note

在主控台中，Docker Compose file in S3 (S3 中的 Docker Compose 檔案) 屬性會結合 DockerComposeFileS3Bucket、DockerComposeFileS3Key 和 DockerComposeFileS3Version 參數。

必要：`true`

類型：`string`

有效模式 .+

DockerComposeFileS3Version

Amazon S3 中 Docker Compose 檔案的物件版本。如需詳細資訊，包括物件金鑰命名準則，請參閱《Amazon Simple Storage Service 使用者指南》中的[使用版本控制](#)。

Note

在主控台中，Docker Compose file in S3 (S3 中的 Docker Compose 檔案) 屬性會結合 DockerComposeFileS3Bucket、DockerComposeFileS3Key 和 DockerComposeFileS3Version 參數。

必要 : false

類型 : string

有效模式 .+

DockerComposeFileDestinationPath

用來儲存 Docker Compose 檔案複本的本機目錄絕對路徑。這必須是現有目錄。DockerUserId 的指定使用者必須具有在此目錄中建立檔案的權限。如需詳細資訊，請參閱[the section called “在核心上設定 Docker 使用者”](#)。

Important

儲存 Docker Compose 檔案的目錄以及私有儲存庫的 Docker 影像登入資料的安全。您有責任保護此目錄。如需詳細資訊，請參閱[the section called “安全注意事項”](#)。

在 AWS IoT 主控台中顯示名稱：本機 Compose 檔案的目錄路徑

必要 : true

類型 : string

有效模式 \/.*\/?

範例 : /home/username/myCompose

DockerUserId

連接器用於執行的 Linux 使用者 UID。此使用者必須屬於核心裝置上的 docker Linux 群組，並具有 DockerComposeFileDestinationPath 目錄的寫入權限。如需詳細資訊，請參閱[在核心上設定 Docker 使用者](#)。

Note

我們建議您除非必要，避免以 root 身分執行。如果您指定根使用者，則必須允許 Lambda 函數以根身分在 AWS IoT Greengrass 核心上執行。如需詳細資訊，請參閱[the section called “執行 Lambda 函數做為根”](#)。

在 AWS IoT 主控台中顯示名稱：Docker 使用者 ID

必要：false

類型：string

有效模式：`^[0-9]{1,5}$`

AWSecretsArnList

中秘密的 Amazon Resource Name (ARNs)，AWS Secrets Manager 其中包含用於存取私有儲存庫中 Docker 映像的登入資訊。如需詳細資訊，請參閱[the section called “從私有儲存庫存取 Docker 影像”](#)。

AWS IoT 主控台顯示名稱：私有儲存庫的登入資料

必要：false。此參數需要存取儲存在私有儲存庫中的 Docker 影像。

類型：array 的 string

有效模式：`[( ? , ? ? "(arn:(aws(-[a-z]+)):secretsmanager:[a-z0-9-]+:[0-9]{12}:secret:([a-zA-Z0-9\]+/)[a-zA-Z0-9/_+=, .@-]+-[a-zA-Z0-9]+))"]`

DockerContainerStatusLogFrequency

連接器記錄有關核心上 Docker 容器執行狀態資訊的頻率 (秒)。預設值為 300 秒 (5 分鐘)。

AWS IoT 主控台顯示名稱：記錄頻率

必要：false

類型：string

有效模式：`^[1-9]{1}[0-9]{0,3}$`

ForceDeploy

指出是否因為上次部署的不當清除而使 Docker 部署失敗。預設值為 False。

AWS IoT 主控台顯示名稱：強制部署

必要：false

類型：string

有效模式：`^(true|false)$`

Version 1

DockerComposeFileS3Bucket

包含 Docker Compose 檔案的 S3 儲存貯體名稱。當您建立儲存貯體時，請務必遵循 Amazon Simple Storage Service 使用者指南中所述的[儲存貯體名稱規則](#)。

在 AWS IoT 主控台中顯示名稱：S3 中的 Docker Compose 檔案

Note

在主控台中，Docker Compose file in S3 (S3 中的 Docker Compose 檔案) 屬性會結合 DockerComposeFileS3Bucket、DockerComposeFileS3Key 和 DockerComposeFileS3Version 參數。

必要：true

類型：string

有效模式：`[a-zA-Z0-9\\-\\.]{3,63}`

DockerComposeFileS3Key

Amazon S3 中 Docker Compose 檔案的物件金鑰。如需詳細資訊，包括物件金鑰命名準則，請參閱《Amazon Simple Storage Service 使用者指南》中的[物件金鑰和中繼資料](#)。

Note

在主控台中，Docker Compose file in S3 (S3 中的 Docker Compose 檔案) 屬性會結合 DockerComposeFileS3Bucket、DockerComposeFileS3Key 和 DockerComposeFileS3Version 參數。

必要：true

類型：string

有效模式 .+

DockerComposeFileS3Version

Amazon S3 中 Docker Compose 檔案的物件版本。如需詳細資訊，包括物件金鑰命名準則，請參閱《Amazon Simple Storage Service 使用者指南》中的[使用版本控制](#)。

Note

在主控台中，Docker Compose file in S3 (S3 中的 Docker Compose 檔案) 屬性會結合 DockerComposeFileS3Bucket、DockerComposeFileS3Key 和 DockerComposeFileS3Version 參數。

必要：false

類型：string

有效模式 .+

DockerComposeFileDestinationPath

用來儲存 Docker Compose 檔案複本的本機目錄絕對路徑。這必須是現有目錄。DockerUserId 的指定使用者必須具有在此目錄中建立檔案的權限。如需詳細資訊，請參閱[the section called “在核心上設定 Docker 使用者”](#)。

 Important

儲存 Docker Compose 檔案的目錄以及私有儲存庫的 Docker 影像登入資料的安全。您有責任保護此目錄。如需詳細資訊，請參閱[the section called “安全注意事項”](#)。

在 AWS IoT 主控台中顯示名稱：本機 Compose 檔案的目錄路徑

必要：true

類型：string

有效模式：\/.*\/?

範例：/home/username/myCompose

DockerUserId

連接器用於執行的 Linux 使用者 UID。此使用者必須屬於核心裝置上的 docker Linux 群組，並具有 DockerComposeFileDestinationPath 目錄的寫入權限。如需詳細資訊，請參閱[在核心上設定 Docker 使用者](#)。

 Note

我們建議您除非必要，避免以 root 身分執行。如果您指定根使用者，則必須允許 Lambda 函數以根身分在 AWS IoT Greengrass 核心上執行。如需詳細資訊，請參閱[the section called “執行 Lambda 函數做為根”](#)。

在 AWS IoT 主控台中顯示名稱：Docker 使用者 ID

必要：false

類型：string

有效模式：^[0-9]{1,5}\$

AWSecretsArnList

中秘密的 Amazon Resource Name (ARNs)，AWS Secrets Manager 其中包含用於存取私有儲存庫中 Docker 映像的登入資訊。如需詳細資訊，請參閱[the section called “從私有儲存庫存取 Docker 映像”](#)。

AWS IoT 主控台顯示名稱：私有儲存庫的登入資料

必要：false。此參數需要存取儲存在私有儲存庫中的 Docker 影像。

類型：array 的 string

有效模式：[(?,? ?"(arn:(aws(-[a-z]+)):secretsmanager:[a-z0-9-]+:[0-9]{12}:secret:([a-zA-Z0-9\]+/)[a-zA-Z0-9/_+=,.@-]+-[a-zA-Z0-9]+)")]

DockerContainerStatusLogFrequency

連接器記錄有關核心上 Docker 容器執行狀態資訊的頻率 (秒)。預設值為 300 秒 (5 分鐘)。

在 AWS IoT 主控台中顯示名稱：記錄頻率

必要：false

類型：string

有效模式：^[1-9]{1}[0-9]{0,3}\$

建立範例連接器 (AWS CLI)

下列 CLI 命令ConnectorDefinition會使用包含 Greengrass Docker 應用程式部署連接器的初始版本來建立。

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
  "Connectors": [
    {
      "Id": "MyDockerApplicationDeploymentConnector",
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/
DockerApplicationDeployment/versions/5",
      "Parameters": {
        "DockerComposeFileS3Bucket": "amzn-s3-demo-bucket",
        "DockerComposeFileS3Key": "production-docker-compose.yml",
        "DockerComposeFileS3Version": "123",
        "DockerComposeFileDestinationPath": "/home/username/myCompose",
        "DockerUserId": "1000",
        "AWSecretsArnList": "[\"arn:aws:secretsmanager:region:account-
id:secret:greengrass-secret1-hash\", \"arn:aws:secretsmanager:region:account-
id:secret:greengrass-secret2-hash\"]",
```

```

        "DockerContainerStatusLogFrequency": "30",
        "ForceDeploy": "True",
        "DockerPullBeforeUp": "True"
    }
}
]
}'

```

Note

此連接器中的 Lambda 函數具有[長期生命週期](#)。

輸入資料

此連接器不需要或接受輸入資料。

輸出資料

此連接器將 `docker-compose up` 命令的狀態發佈為輸出資料。

訂閱中的主題篩選條件

`dockerapplicationdeploymentconnector/message/status`

範例輸出：成功

```

{
  "status": "success",
  "GreengrassDockerApplicationDeploymentStatus": "Successfully triggered docker-
compose up",
  "S3Bucket": "amzn-s3-demo-bucket",
  "ComposeFileName": "production-docker-compose.yml",
  "ComposeFileVersion": "123"
}

```

範例輸出：失敗

```

{
  "status": "fail",
  "error_message": "description of error",
  "error": "InvalidParameter"
}

```

```
}
```

錯誤類型可以是 `InvalidParameter` 或 `InternalError`。

在 AWS IoT Greengrass 核心上設定 Docker 使用者

Greengrass Docker 應用程式部署連接器會以您為 `DockerUserId` 參數指定的使用者身分執行。如果您未指定值，連接器會以 `ggc_user` 為身分執行，這也是預設 Greengrass 存取身分。

若要允許連接器與 Docker 協助程式互動，Docker 使用者必須屬於核心上的 `docker` Linux 群組。Docker 使用者也必須具有 `DockerComposeFileDestinationPath` 目錄的寫入權限。這是連接器儲存本機 `docker-compose.yml` 檔案和 Docker 登入資料的位置。

Note

- 建議您建立 Linux 使用者，不要使用預設值 `ggc_user`。否則，Greengrass 群組中的任何 Lambda 函數都可以存取 Compose 檔案和 Docker 登入資料。
- 我們建議您除非必要，避免以 `root` 身分執行。如果您指定根使用者，則必須允許 Lambda 函數以根身分在 AWS IoT Greengrass 核心上執行。如需詳細資訊，請參閱[the section called “執行 Lambda 函數做為根”](#)。

1. 建立使用者。您可以執行 `useradd` 命令，並包含選用的 `-u` 選項來指派 UID。例如：

```
sudo useradd -u 1234 user-name
```

2. 將使用者新增至核心上的 `docker` 群組。例如：

```
sudo usermod -aG docker user-name
```

如需詳細資訊 (包括如何建立 `docker` 群組)，請參閱 Docker 文件中 [Manage Docker as a non-root user](#) (以非根使用者身管理 Docker)。

3. 授予使用者寫入 `DockerComposeFileDestinationPath` 參數指定目錄的權限。例如：
 - a. 將使用者設定為目錄的擁有者。這個範例會使用步驟 1 中的 UID。

```
chown 1234 docker-compose-file-destination-path
```


- b. 將讀取和寫入權限提供給擁有者。

```
chmod 700 docker-compose-file-destination-path
```

如需詳細資訊，請參閱 Linux 基礎文件中的 [How To Manage File And Folder Permissions In Linux](#) (如何管理 Linux 中的檔案及資料夾權限)。

- c. 如果您在建立使用者時未指派 UID，或者，您使用現有使用者，請執行 `id` 命令以查詢 UID。

```
id -u user-name
```

您可以使用 UID 來設定連接器的 `DockerUserId` 參數。

用量資訊

當您使用 Greengrass Docker 應用程式部署連接器時，您應該注意下列實作特定的使用資訊。

- 已修正專案名稱的字首。連接器會在它啟動的 Docker 容器名稱前加上 `greengrassdockerapplicationdeployment` 字首。連接器在其執行的 `docker-compose` 指令中使用此字首做為專案名稱。
- 記錄行為。連接器會將狀態資訊和疑難排解資訊寫入日誌檔案。您可以設定 AWS IoT Greengrass 將日誌傳送至 CloudWatch Logs，並在本機寫入日誌。如需詳細資訊，請參閱 [the section called “日誌”](#)。這是連接器本機記錄檔的路徑：

```
/greengrass-root/ggc/var/log/user/region/aws/DockerApplicationDeployment.log
```

您必須具有 root 權限才能存取本機日誌。

- 更新 Docker 映像。Docker 會快取核心裝置上的影像。如果您更新 Docker 影像，並想將變更傳播到核心裝置，請務必變更 Compose 檔案中的影像標籤。部署 Greengrass 群組之後，變更就會生效。
- 清除操作的 10 分鐘逾時。當 Greengrass 協助程式在重新啟動期間停止時，會啟動 `docker-compose down` 命令。`docker-compose down` 啟動後，所有 Docker 容器最多有 10 分鐘的時間來執行任何清除操作。如果未在 10 分鐘內完成清除，您必須手動清除剩餘的容器。如需詳細資訊，請參閱 Docker CLI 文件中的 [docker rm](#)。
- 執行 Docker 命令。若要對問題進行疑難排解，可以在核心裝置的終端機視窗中執行 Docker 命令。例如，執行下列命令以查看連接器啟動的 Docker 容器：

```
docker ps --filter name="greengrassdockerapplicationdeployment"
```

- 預留資源 ID。連接器會使用其在 Greengrass 群組中建立的 Greengrass 資源 DOCKER_DEPLOYER_SECRET_RESOURCE_RESERVED_ID_*index* ID。資源 ID 在群組中不得重複，因此請勿指派可能與此保留資源 ID 衝突的資源 ID。
- 離線模式。當您將 DockerOfflineMode 組態參數設定為 True，Docker 連接器可以離線模式操作。當 Greengrass 群組部署在核心裝置離線時重新啟動，且連接器無法建立與 Amazon S3 或 Amazon ECR 的連線以擷取 Docker Compose 檔案時，就會發生這種情況。

啟用離線模式時，連接器會嘗試下載 Compose 檔案，並像在正常重新啟動時一樣執行 `docker login` 命令。如果這些嘗試失敗，則連接器會在使用 `DockerComposeFileDestinationPath` 參數指定的資料夾中尋找本機儲存的 Compose 檔案。如果本機 Compose 檔案存在，則連接器會遵循正常的 `docker-compose` 命令順序，並從本機映像提取。如果 Compose 檔案或本機映像不存在，則連接器會失敗。在離線模式下，`ForceDeploy` 和 `StopContainersOnNewDeployment` 參數的行為保持不變。

與 Docker 容器通訊

AWS IoT Greengrass 支援 Greengrass 元件和 Docker 容器之間的下列通訊管道：

- Greengrass Lambda 函數可以使用 REST APIs 與 Docker 容器中的程序通訊。您可以在開啟連接埠的 Docker 容器中設定伺服器。Lambda 函數可以與此連接埠上的容器通訊。
- Docker 容器中的處理程序可以透過本機 Greengrass 訊息中介裝置交換 MQTT 訊息。您可以在 Greengrass 群組中將 Docker 容器設定為用戶端裝置，然後建立訂閱，以允許容器與群組中的 Greengrass Lambda 函數、用戶端裝置和其他連接器通訊，或與 AWS IoT 和本機影子服務通訊。如需詳細資訊，請參閱[the section called “設定與 Docker 容器的 MQTT 通訊”](#)。
- Greengrass Lambda 函數可以更新共用檔案，以將資訊傳遞至 Docker 容器。您可以使用 Compose 檔案綁定掛載 Docker 容器的共用檔案路徑。

設定與 Docker 容器的 MQTT 通訊

您可以將 Docker 容器設定為用戶端裝置，並將其新增至 Greengrass 群組。然後，您可以建立訂閱，以允許 Docker 容器和 Greengrass 元件或 AWS IoT 之間的 MQTT 通訊。在下列程序中，您會建立訂閱，允許 Docker 容器裝置接收來自本機陰影服務的陰影更新訊息。您可以遵循此模式來建立其他訂閱。

Note

此程序假設您已建立 Greengrass 群組和 Greengrass 核心 (v1.10 或更新版本)。如需建立 Greengrass 群組和核心的詳細資訊，請參閱 [入門 AWS IoT Greengrass](#)。

將 Docker 容器設定為用戶端裝置，並將其新增至 Greengrass 群組

1. 在核心裝置上建立資料夾，以存放用於驗證 Greengrass 裝置的憑證和金鑰。

檔案路徑必須掛載在您要啟動的 Docker 容器上。下面的程式碼片段顯示了如何在您的 Compose 檔案中掛載文件路徑。在此範例中，*path-to-device-certs* 代表您在此步驟中建立的資料夾。

```
version: '3.3'
services:
  myService:
    image: user-name/repo:image-tag
    volumes:
      - /path-to-device-certs/:/path-accessible-in-container
```

2. 在 AWS IoT 主控台導覽窗格的管理下，展開 Greengrass 裝置，然後選擇群組 (V1)。
3. 選擇目標群組。
4. 在群組組態頁面上，選擇用戶端裝置，然後選擇關聯。
5. 在將用戶端裝置與此群組模型建立關聯中，選擇建立新 AWS IoT 物件。

建立物件頁面會在新標籤中開啟。

6. 在建立物件頁面上，選擇建立單一物件，然後選擇下一步。
7. 在指定物件屬性頁面上，輸入裝置的名稱，然後選擇下一步。
8. 在設定裝置憑證頁面上，選擇下一步。
9. 在將政策連接至憑證頁面上，執行下列其中一項操作：
 - 選取授予用戶端裝置所需許可的現有政策，然後選擇建立物件。

模態開啟，您可以在其中下載裝置用來連線至 和 核心的憑證 AWS 雲端 和金鑰。

- 建立並連接授予用戶端裝置許可的新政策。請執行下列操作：
 - a. 選擇建立政策。

Create policy (建立政策) 頁面隨即在新標籤中開啟。

- b. 在 Create policy (建立政策) 頁面上，執行下列動作：
 - i. 針對政策名稱，輸入描述政策的名稱，例如 **GreengrassV1ClientDevicePolicy**。
 - ii. 在政策陳述式索引標籤的政策文件下，選擇 JSON。
 - iii. 輸入下列政策文件。此政策可讓用戶端裝置探索 Greengrass 核心，並在所有 MQTT 主題上進行通訊。如需如何限制此政策存取的資訊，請參閱 [的裝置身分驗證和授權 AWS IoT Greengrass](#)。

JSON

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

- iv. 選擇 Create (建立) 以建立政策。

- c. 返回瀏覽器索引標籤，開啟附加政策至憑證頁面。請執行下列操作：
 - i. 在政策清單中，選取您建立的政策，例如 GreengrassV1ClientDevicePolicy。

如果您沒有看到政策，請選擇重新整理按鈕。

- ii. 選擇 Create thing (建立物件)。

模式開啟，您可以在其中下載裝置用來連線至 和 核心的憑證 AWS 雲端 和金鑰。

10. 在下載憑證和金鑰模式中，下載裝置的憑證。

Important

選擇完成之前，請下載安全資源。

請執行下列操作：

- a. 針對裝置憑證，選擇下載以下載裝置憑證。
- b. 針對公有金鑰檔案，選擇下載以下載憑證的公有金鑰。
- c. 針對私有金鑰檔案，選擇下載以下載憑證的私有金鑰檔案。
- d. 檢閱《AWS IoT 開發人員指南》中的[伺服器身分驗證](#)，然後選擇適當的根 CA 憑證。我們建議您使用 Amazon Trust Services (ATS) 端點和 ATS 根 CA 憑證。在根 CA 憑證下，選擇根 CA 憑證的下載。
- e. 選擇完成。

請記下裝置憑證和金鑰檔案名稱中常見的憑證 ID。供稍後使用。

11. 將憑證和金鑰複製到您在步驟 1 中建立的資料夾。

接下來，在群組中建立訂閱。在此範例中，您可以建立訂閱讓 Docker 容器裝置接收來自本機陰影服務的 MQTT 訊息。

Note

陰影文件的大小上限為 8 KB。如需詳細資訊，請參閱《AWS IoT 開發人員指南》中的[AWS IoT 配額](#)。

建立訂閱，允許 Docker 容器裝置接收來自本機陰影服務的 MQTT 訊息

1. 在群組組態頁面上，選擇訂閱索引標籤，然後選擇新增訂閱。
2. 在 Select your source and target (選擇您的來源和目標) 頁面設定來源和目標，如下所示：
 - a. 在 Select a source (選取來源) 中，選擇 Services (服務)，然後選擇 Local Shadow Service (本機陰影服務)。
 - b. 在 Select a target (選取目標)，選擇 Devices (裝置)，然後選擇您的裝置。
 - c. 選擇下一步。
 - d. 在使用主題篩選資料頁面上，針對主題篩選，選擇 **\$aws/things/MyDockerDevice/shadow/update/accepted**，然後選擇下一步。將 **MyDockerDevice** 取代為您先前建立的裝置名稱。
 - e. 選擇完成。

在您在 Compose 檔案中參考的 Docker 影像中，加入下列程式碼片段。這是 Greengrass 裝置程式碼。另外，請在 Docker 容器中新增程式碼，以啟動容器內的 Greengrass 設備。它可以在影像中或獨立的執行緒中，以個別程序來執行。

```
import os
import sys
import time
import uuid

from AWSIoTPythonSDK.core.greengrass.discovery.providers import DiscoveryInfoProvider
from AWSIoTPythonSDK.exception.AWSIoTExceptions import DiscoveryInvalidRequestException
from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient

# Replace thingName with the name you registered for the Docker device.
thingName = "MyDockerDevice"
clientId = thingName

# Replace host with the IoT endpoint for your &AWS-account;.
host = "myPrefix.iot.region.amazonaws.com"

# Replace topic with the topic where the Docker container subscribes.
topic = "$aws/things/MyDockerDevice/shadow/update/accepted"

# Replace these paths based on the download location of the certificates for the Docker container.
rootCAPath = "/path-accessible-in-container/AmazonRootCA1.pem"
```

```
certificatePath = "/path-accessible-in-container/certId-certificate.pem.crt"
privateKeyPath = "/path-accessible-in-container/certId-private.pem.key"

# Discover Greengrass cores.
discoveryInfoProvider = DiscoveryInfoProvider()
discoveryInfoProvider.configureEndpoint(host)
discoveryInfoProvider.configureCredentials(rootCAPath, certificatePath, privateKeyPath)
discoveryInfoProvider.configureTimeout(10) # 10 seconds.

GROUP_CA_PATH = "./groupCA/"
MQTT_QOS = 1

discovered = False
groupCA = None
coreInfo = None

try:
    # Get discovery info from AWS IoT.
    discoveryInfo = discoveryInfoProvider.discover(thingName)
    caList = discoveryInfo.getAllCas()
    coreList = discoveryInfo.getAllCores()

    # Use first discovery result.
    groupId, ca = caList[0]
    coreInfo = coreList[0]

    # Save the group CA to a local file.
    groupCA = GROUP_CA_PATH + groupId + "_CA_" + str(uuid.uuid4()) + ".crt"
    if not os.path.exists(GROUP_CA_PATH):
        os.makedirs(GROUP_CA_PATH)
    groupCAFile = open(groupCA, "w")
    groupCAFile.write(ca)
    groupCAFile.close()
    discovered = True
except DiscoveryInvalidRequestException as e:
    print("Invalid discovery request detected!")
    print("Type: %s" % str(type(e)))
    print("Error message: %s" % str(e))
    print("Stopping...")
except BaseException as e:
    print("Error in discovery!")
    print("Type: %s" % str(type(e)))
    print("Error message: %s" % str(e))
```

```
print("Stopping...")

myAWSIoTMQTTClient = AWSIoTMQTTClient(clientId)
myAWSIoTMQTTClient.configureCredentials(groupCA, privateKeyPath, certificatePath)

# Try to connect to the Greengrass core.
connected = False
for connectivityInfo in coreInfo.connectivityInfoList:
    currentHost = connectivityInfo.host
    currentPort = connectivityInfo.port
    myAWSIoTMQTTClient.configureEndpoint(currentHost, currentPort)
    try:
        myAWSIoTMQTTClient.connect()
        connected = True
    except BaseException as e:
        print("Error in connect!")
        print("Type: %s" % str(type(e)))
        print("Error message: %s" % str(e))
    if connected:
        break

if not connected:
    print("Cannot connect to core %s. Exiting..." % coreInfo.coreThingArn)
    sys.exit(-2)

# Handle the MQTT message received from GGShadowService.
def customCallback(client, userdata, message):
    print("Received an MQTT message")
    print(message)

# Subscribe to the MQTT topic.
myAWSIoTMQTTClient.subscribe(topic, MQTT_QOS, customCallback)

# Keep the process alive to listen for messages.
while True:
    time.sleep(1)
```

安全注意事項

當您使用 Greengrass Docker 應用程式部署連接器時，請注意下列安全考量。

Docker Compose 檔案的本機儲存區

連接器會將 Compose 檔案的複本儲存在為 `DockerComposeFileDestinationPath` 參數指定的目錄中。

您有責任保護此目錄。您應該使用檔案系統權限來限制目錄的存取。

Docker 登入資料的本機儲存區

如果您的 Docker 影像儲存在私有儲存庫中，連接器會將您的 Docker 登入資料儲存在為 `DockerComposeFileDestinationPath` 參數指定的目錄中。

您有責任保護這些登入資料。例如，安裝 Docker Engine 時，應在核心裝置上使用 [credential-helper](#)。

從可信任來源安裝 Docker Engine

您有責任從可信任來源安裝 Docker Engine。此連接器使用核心裝置上的 Docker 協助程式，存取您的 Docker 資產並管理 Docker 容器。

Greengrass 群組角色權限的範圍

您在 Greengrass 群組角色中新增的許可，可由 Greengrass 群組中的所有 Lambda 函數和連接器擔任。此連接器需要存取儲存 S3 儲存貯體中的 Docker Compose 檔案。如果您的 Docker 映像存放在 Amazon ECR 中的私有儲存庫中，它也需要存取您的 Amazon ECR 授權字符。

授權

Greengrass Docker 應用程式部署連接器包含下列第三方軟體/授權：

- [適用於 Python \(Boto3\) 的 AWS SDK](#)/Apache License 2.0
- [botocore](#)/Apache License 2.0
- [dateutil](#)/PSF License
- [docutils](#)/BSD 授權、GNU 一般公有授權 (GPL)、Python 軟體基金會授權、公有網域
- [jmespath](#)/MIT License
- [s3transfer](#)/Apache License 2.0
- [urllib3](#)/MIT License

此連接器根據 [Greengrass 核心軟體授權合約](#) 發行。

變更記錄

下表說明每個版本連接器的變更。

版本	變更
7	新增DockerOfflineMode 以在離線 AWS IoT Greengrass 啟動時使用現有的 Docker Compose 檔案。已實作docker login命令的重試。支援 32 位元 UIDs。
6	新增StopContainersOnNewDeployment 以在進行新部署或 GGC 停止時覆寫容器清除。更安全的關機和啟動機制。YAML 驗證錯誤修正。
5	映像會在執行 之前提取docker-compose down 。
4	新增pull-before-up行為以更新 Docker 映像。
3	修正了尋找環境變數的問題。
2	新增 ForceDeploy 參數。
1	初始版本。

Greengrass 群組一次只能包含一個版本的連接器。若要取得有關升級連接器版本的資訊，請參閱[the section called “升級連接器版本”](#)。

另請參閱

- [使用連接器整合服務和通訊協定](#)
- [the section called “連接器入門 \(主控台\)”](#)
- [the section called “連接器入門 \(CLI\)”](#)

IoT Analytics 連接器

Warning

此連接器已進入延長生命週期階段，且 AWS IoT Greengrass 不會發佈提供現有功能、增強功能、安全修補程式或錯誤修正的更新。如需詳細資訊，請參閱[AWS IoT Greengrass Version 1 維護政策](#)。

IoT Analytics 連接器會將本機裝置資料傳送至 AWS IoT Analytics。您可以使用此連接器做為中央中樞，從 Greengrass 核心裝置上的感應器和[連接的用戶端裝置](#)收集資料。連接器會將資料傳送至目前 AWS 帳戶和區域中的 AWS IoT Analytics 頻道。它可以將資料傳送到預設目的地管道和動態指定的管道。

Note

AWS IoT Analytics 是一種全受管服務，可讓您收集、存放、處理和查詢 IoT 資料。在 AWS IoT Analytics，可以進一步分析和處理資料。例如，您可以用於訓練機器學習模型來監控機器運作狀況，或測試新的建模策略。如需詳細資訊，請參閱 AWS IoT Analytics 《使用者指南》中的[什麼是 AWS IoT Analytics？](#)。

連接器接受[輸入 MQTT 主題](#)上的有格式和無格式資料。它支援兩種預先定義的主題，目的地管道為指定的內嵌。它也可以接收[在訂閱中配置的](#)客戶定義主題訊息。這可用於從發佈至固定主題的用戶端裝置路由訊息，或從資源受限的裝置處理非結構化或堆疊相依資料。

此連接器使用 [BatchPutMessage](#) API 將資料（以 JSON 或 base64 編碼字串的形式）傳送至目的地頻道。連接器可將原始資料處理為符合 API 要求的格式。連接器會依管道佇列緩衝輸入的訊息，並以非同步方式處理批次。它提供參數，讓您控制佇列和批次處理行為，以及限制記憶體用量。例如，您可以設定最大佇列大小、批次間隔、記憶體大小和作用中管道的數量。

此連接器具有下列版本。

版本	ARN
4	<code>arn:aws:greengrass: <i>region</i> ::/connectors/IoTAnalytics/versions/4</code>

版本	ARN
3	arn:aws:greengrass: <i>region</i> ::/connectors/IoTAnalytics/versions/3
2	arn:aws:greengrass: <i>region</i> ::/connectors/IoTAnalytics/versions/2
1	arn:aws:greengrass: <i>region</i> ::/connectors/IoTAnalytics/versions/1

如需版本變更的詳細資訊，請參閱 [Changelog](#)。

要求

此連接器有下列要求：

Version 3 - 4

- AWS IoT Greengrass 核心軟體 v1.9.3 或更新版本。
- 安裝在核心裝置上並新增至 PATH 環境變數的 [Python](#) 3.7 或 3.8 版。

Note

若要使用 Python 3.8，請執行下列命令，以建立從預設 Python 3.7 安裝資料夾到已安裝 Python 3.8 二進位檔的符號連結。

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

這會設定您的裝置以符合 AWS IoT Greengrass 的 Python 需求。

- 此連接器只能在同時[AWS IoT Analytics](#)支援 [AWS IoT Greengrass](#)和 的 Amazon Web Services 區域中使用。

- 建立並設定所有相關 AWS IoT Analytics 實體和工作流程。實體包含頻道、管道、資料存放區和資料集。如需詳細資訊，請參閱 AWS IoT Analytics 《使用者指南》中的 [AWS CLI](#) 或 [主控台](#) 程序。

 Note

目的地 AWS IoT Analytics 頻道必須使用相同的帳戶，且與此 AWS 區域 連接器位於相同的 中。

- [Greengrass 群組角色](#) 設定為允許對目的地頻道執行 `iotanalytics:BatchPutMessage` 動作，如下列範例 IAM 政策所示。頻道必須位於目前的 AWS 帳戶 和 區域中。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1528133056761",
      "Action": [
        "iotanalytics:BatchPutMessage"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:iotanalytics:us-east-1:123456789012:channel/channel_1_name",
        "arn:aws:iotanalytics:us-east-1:123456789012:channel/channel_2_name"
      ]
    }
  ]
}
```

針對群組角色要求，您必須設定角色以授與必要的許可，並確認已將角色新增至群組。如需詳細資訊，請參閱 [the section called “管理群組角色 \(主控台\)”](#) 或 [the section called “管理群組角色 \(CLI\)”](#)。

Versions 1 - 2

- AWS IoT Greengrass 核心軟體 v1.7 或更新版本。
- 安裝在核心裝置上並新增至 PATH 環境變數的 [Python 2.7](#) 版。

- 此連接器只能在同時[AWS IoT Analytics](#)支援 [AWS IoT Greengrass](#)和 的 Amazon Web Services 區域中使用。
- 建立並設定所有相關 AWS IoT Analytics 實體和工作流程。實體包含頻道、管道、資料存放區和資料集。如需詳細資訊，請參閱AWS IoT Analytics 《使用者指南》中的 [AWS CLI](#)或 [主控台](#)程序。

 Note

目的地 AWS IoT Analytics 頻道必須使用相同的帳戶，且與此 AWS 區域 連接器位於相同的 中。

- 設定為允許對目的地頻道*iotanalytics:BatchPutMessage*執行動作的 [Greengrass 群組角色](#)，如下列範例 IAM 政策所示。頻道必須位於目前的 AWS 帳戶 和 區域中。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1528133056761",
      "Action": [
        "iotanalytics:BatchPutMessage"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:iotanalytics:us-east-1:123456789012:channel/channel_1_name",
        "arn:aws:iotanalytics:us-east-1:123456789012:channel/channel_2_name"
      ]
    }
  ]
}
```

針對群組角色要求，您必須設定角色以授與必要的許可，並確認已將角色新增至群組。如需詳細資訊，請參閱 [the section called “管理群組角色 \(主控台\)”](#) 或 [the section called “管理群組角色 \(CLI\)”](#)。

Parameters

MemorySize

配置給此連接器的記憶體數量 (KB)。

AWS IoT 主控台中的顯示名稱：記憶體大小

必要：true

類型：string

有效模式：`^[0-9]+$`

PublishRegion

AWS 區域 AWS IoT Analytics 頻道建立所在的。使用與連接器相同的區域。

Note

這也必須與[群組角色](#)中指定的管道所在區域相同。

AWS IoT 主控台中的顯示名稱：發布區域

必要：false

類型：string

有效模式：`^$|([a-z]{2}-[a-z]+-\\d{1})`

PublishInterval

將一批接收資料發佈到其中的間隔（以秒為單位）AWS IoT Analytics。

在 AWS IoT 主控台中顯示名稱：發佈間隔

必要：false

類型：string

預設值：1

有效模式：`$|^[0-9]+$`

IotAnalyticsMaxActiveChannels

連接器主動監看的 AWS IoT Analytics 頻道數量上限。此數值必須大於 0，且至少等於您希望連接器在特定時間發佈的目標頻道數。

您可以使用此參數，透過限制連接器在特定時間可以管理的佇列總數，藉以限制記憶體用量。佇列中所有訊息皆傳送後，該佇列就會刪除。

AWS IoT 主控台中的顯示名稱：作用中頻道的數量上限

必要：false

類型：string

預設值：50

有效模式：`^[1-9][0-9]*$`

IotAnalyticsQueueDropBehavior

管道佇列已滿時，從佇列拋棄訊息的行為。

AWS IoT 主控台中的顯示名稱：佇列捨棄行為

必要：false

類型：string

有效值：DROP_NEWEST 或 DROP_OLDEST

預設值：DROP_NEWEST

有效模式：`^DROP_NEWEST$|^DROP_OLDEST$`

IotAnalyticsQueueSizePerChannel

訊息提交或拋棄前，在 (每個管道) 記憶體中保留的最大訊息數。此數值必須大於 0。

AWS IoT 主控台中的顯示名稱：每個通道的佇列大小上限

必要：false

類型：string

預設值：2048

有效模式：`^[1-9][0-9]*$`

`IotAnalyticsBatchSizePerChannel`

一個批次請求中要傳送到 AWS IoT Analytics 頻道的訊息數量上限。此數值必須大於 0。

在 AWS IoT 主控台中顯示名稱：每個頻道要批次處理的訊息數量上限

必要：false

類型：string

預設值：5

有效模式：`^[1-9][0-9]*$`

`IotAnalyticsDefaultChannelName`

此連接器用於傳送至客戶定義輸入主題之訊息的 AWS IoT Analytics 頻道名稱。

AWS IoT 主控台顯示名稱：預設頻道名稱

必要：false

類型：string

有效模式：`^[a-zA-Z0-9_]$`

`IsolationMode`

此連接器的[容器化](#)模式。預設值為 `GreengrassContainer`，這表示連接器會在 AWS IoT Greengrass 容器內的隔離執行時間環境中執行。

Note

群組的預設容器化設定不會套用至連接器。

在 AWS IoT 主控台中顯示名稱：容器隔離模式

必要：false

類型：string

有效值：GreengrassContainer 或 NoContainer

有效模式：^NoContainer\$|^GreengrassContainer\$

建立範例連接器 (AWS CLI)

下列 CLI 命令ConnectorDefinition會使用包含 IoT Analytics 連接器的初始版本來建立。

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-  
version '{  
  "Connectors": [  
    {  
      "Id": "MyIoTAnalyticsApplication",  
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/IoTAnalytics/  
versions/3",  
      "Parameters": {  
        "MemorySize": "65535",  
        "PublishRegion": "us-west-1",  
        "PublishInterval": "2",  
        "IotAnalyticsMaxActiveChannels": "25",  
        "IotAnalyticsQueueDropBehavior": "DROP_OLDEST",  
        "IotAnalyticsQueueSizePerChannel": "1028",  
        "IotAnalyticsBatchSizePerChannel": "5",  
        "IotAnalyticsDefaultChannelName": "my_channel"  
      }  
    }  
  ]  
}'
```

Note

此連接器中的 Lambda 函數具有[長期生命週期](#)。

在 AWS IoT Greengrass 主控台中，您可以從群組的連接器頁面新增連接器。如需詳細資訊，請參閱[the section called “連接器入門 \(主控台\)”](#)。

輸入資料

此連接器根據預先定義和客戶定義的 MQTT 主題接受資料。發佈者可以是用戶端裝置、Lambda 函數或其他連接器。

預先定義的主題

連接器支援以下兩個結構化 MQTT 主題，讓發佈方內嵌指定頻道名稱。

- `iotanalytics/channels/+/messages/put` 主題的[格式化訊息](#)。這些輸入訊息中的 IoT 資料，必須格式化為 JSON 或 base64 編碼字串。
- 有關 `iotanalytics/channels/+/messages/binary/put` 主題的非格式化訊息。在此主題接收到的輸入訊息皆視為二元資料，可包含任何資料類型。

若要發佈到預先定義的主題，請將 `+` 萬用字元改為管道名稱。例如：

```
iotanalytics/channels/my_channel/messages/put
```

客戶定義的主題

連接器支援 `#` 主題的語法，可接受您在訂閱中配置的任何 MQTT 主題中的輸入訊息。我們建議您指定主題路徑，而不是在訂閱中僅使用 `#` 萬用字元。這些訊息會傳送到您為連接器指定的預設頻道。

客戶定義主題上的輸入訊息視為二元資料。此類資料可使用任何訊息格式，並包含任何資料類型。您可以使用客戶定義的主題，從發佈至固定主題的裝置路由訊息。您也可以使用它們來接受來自用戶端裝置的輸入資料，這些用戶端裝置無法將資料處理為格式化訊息，以傳送至連接器。

如需更多關於訂閱和 MQTT 主題的資訊，請參閱 [the section called “輸入和輸出”](#)。

群組角色必須允許所有目標管道上的 `iotanalytics:BatchPutMessage` 動作。如需詳細資訊，請參閱[the section called “要求”](#)。

主題篩選條件：`iotanalytics/channels/+/messages/put`

使用此主題將格式化訊息傳送至連接器，並動態指定目的地頻道。此主題也可讓您指定 ID，在回應輸出中傳回。連接器會驗證所傳送傳出 `BatchPutMessage` 請求中每個訊息的 IDs 是唯一的 AWS IoT Analytics。ID 重複的訊息會遭到拋棄。

傳送到此主題的輸入資料必須使用下列訊息格式。

訊息屬性

`request`

傳送至指定管道的資料。

必要：`true`

類型：object 包含下列屬性：

message

JSON 或 base64 編碼字串的裝置或感應器資料。

必要：true

類型：string

id

請求的任意 ID。此屬性用於將輸入請求映射到輸出回應。當指定時，回應物件中的 id 屬性會設為這個值。如果省略此屬性，連接器會產生 ID。

必要：false

類型：string

有效模式：.*

範例輸入

```
{
  "request": {
    "message" : "{\"temp\":23.33}"
  },
  "id" : "req123"
}
```

主題篩選條件：iotanalytics/channels/+/messages/binary/put

使用此主題將無格式訊息傳送至連接器，並動態指定目的地頻道。

連接器資料不剖析此主題接收到的輸入訊息。它會將其視為二元資料。在傳送訊息到之前 AWS IoT Analytics，連接器會編碼並格式化它們以符合 BatchPutMessage API 要求：

- 連接器 base64 編碼原始資料，並將編碼的酬載加入傳出的 BatchPutMessage 請求。
- 連接器會為每個輸入訊息產生和指派 ID。

 Note

連接器的回應輸出不包含這些輸入訊息的 ID 相關性。

訊息屬性

無。

主題篩選條件：#

使用此主題將任何訊息格式傳送至預設管道。當您的用戶端裝置發佈至固定主題，或您想要將資料從無法以連接器[支援的訊息格式](#)處理資料的用戶端裝置傳送到預設頻道時，這特別有用。

您可以在您建立的訂閱中定義主題語法，以將此連接器連接到資料來源。我們建議您指定主題路徑，而不是在訂閱中僅使用#萬用字元。

連接器資料不剖析發佈至此輸入主題的訊息。所有輸入訊息視為二元資料。在傳送訊息到之前 AWS IoT Analytics，連接器會編碼並格式化它們以符合 BatchPutMessage API 要求：

- 連接器 base64 編碼原始資料，並將編碼的酬載加入傳出的 BatchPutMessage 請求。
- 連接器會為每個輸入訊息產生和指派 ID。



Note

連接器的回應輸出不包含這些輸入訊息的 ID 相關性。

訊息屬性

無。

輸出資料

這個連接器會將狀態資訊發佈為輸出資料，且主題為 MQTT。此資訊包含 AWS IoT Analytics 為其接收和傳送的每個輸入訊息傳回的回應 AWS IoT Analytics。

訂閱中的主題篩選條件

```
iotanalytics/messages/put/status
```

範例輸出：成功

```
{
  "response" : {
    "status" : "success"
  },
  "id" : "req123"
}
```

範例輸出：失敗

```
{
  "response" : {
    "status" : "fail",
    "error" : "ResourceNotFoundException",
    "error_message" : "A resource with the specified name could not be found."
  },
  "id" : "req123"
}
```

Note

如果連接器偵測到可重試的錯誤（例如連線錯誤），它會在下一個批次中重試發佈。指數退避由 AWS SDK 處理。因可重試錯誤而失敗的請求會加回管道佇列末端，依據 `IotAnalyticsQueueDropBehavior` 參數進一步發佈。

用量範例

使用以下高階步驟來設定範例 Python 3.7 Lambda 函數，您可以用來嘗試連接器。

Note

- 如果您使用其他 Python 執行期，則可以建立從 Python3.x 到 Python 3.7 的符號連結。
- [連接器入門 \(主控台\)](#) 和 [連接器入門 \(CLI\)](#) 主題包含詳細步驟，說明如何設定和部署範例 Twilio 通知連接器。

1. 確定您符合連接器的[要求](#)。

針對群組角色要求，您必須設定角色以授與必要的許可，並確認已將角色新增至群組。如需詳細資訊，請參閱 [the section called “管理群組角色 \(主控台\)”](#) 或 [the section called “管理群組角色 \(CLI\)”](#)。

2. 建立並發佈 Lambda 函數，將輸入資料傳送至連接器。

將範例程式碼儲存為 PY 檔案。下載並解壓縮 [AWS IoT Greengrass 適用於 Python 的 Core SDK](#)。然後，建立在根層級包含 PY 檔案和 greengrasssdk 資料夾的 zip 套件。此 zip 套件是您上傳至的部署套件 AWS Lambda。

建立 Python 3.7 Lambda 函數後，發佈函數版本並建立別名。

3. 設定 Greengrass 群組。

- a. 依別名新增 Lambda 函數（建議）。將 Lambda 生命週期設定為長期（或在 CLI "Pinned": true 中）。
- b. 新增連接器並設定其[參數](#)。
- c. 新增訂閱，允許連接器在支援主題篩選條件上接收[輸入資料](#)並傳送[輸出資料](#)。
 - 將 Lambda 函數設定為來源，將連接器設定為目標，並使用支援的輸入主題篩選條件。
 - 將連接器設為來源、將 AWS IoT Core 設為目標，並使用支援的輸出主題篩選條件。您可以使用此訂閱在 AWS IoT 主控台中檢視狀態訊息。

4. 部署群組。

5. 在 AWS IoT 主控台的測試頁面上，訂閱輸出資料主題，以檢視連接器的狀態訊息。Lambda 函數範例為長期函數，並在部署群組後立即開始傳送訊息。

完成測試後，您可以將 Lambda 生命週期設定為隨需（或在 CLI "Pinned": false 中）並部署群組。這會讓函數停止傳送訊息。

範例

下列範例 Lambda 函數會將輸入訊息傳送至連接器。

```
import greengrasssdk
import time
import json

iot_client = greengrasssdk.client('iot-data')
send_topic = 'iotanalytics/channels/my_channel/messages/put'

def create_request_with_all_fields():
    return {
        "request": {
            "message" : "{\"temp\":23.33}"
        },
        "id" : "req_123"
    }

def publish_basic_message():
    messageToPublish = create_request_with_all_fields()
```

```
print("Message To Publish: ", messageToPublish)
iot_client.publish(topic=send_topic,
    payload=json.dumps(messageToPublish))

publish_basic_message()

def lambda_handler(event, context):
    return
```

限制

此連接器受下列限制的約束。

- 適用於 Python (Boto3) 的 AWS SDK 為 AWS IoT Analytics [batch_put_message](#) 動作施加的所有限制。
- AWS IoT Analytics [BatchPutMessage](#) API 實施的所有配額。如需詳細資訊，請參閱《AWS IoT Analytics》中的適用於的 [Service Quotas](#) AWS 一般參考。
 - 每個管道每秒 100,000 個訊息。
 - 每個批次 100 則訊息。
 - 每則訊息 128 KB。

這個 API 使用管道名稱 (而非管道 ARN)，因此不支援跨區域或跨帳戶管道傳送資料。

- 核心強制實施的所有配額 AWS IoT Greengrass。如需詳細資訊，請參閱 [AWS IoT Greengrass 核心的 \[Service Quotas\]\(#\)](#) AWS 一般參考。

以下配額可能特別適用：

- 裝置傳送的訊息大小上限為 128 KB。
- Greengrass 核心路由器中的訊息佇列大小上限為 2.5 MB。
- 主題字串的長度上限為 256 位元組的 UTF-8 編碼字元。

授權

IoT Analytics 連接器包含下列第三方軟體/授權：

- [適用於 Python \(Boto3\) 的 AWS SDK](#)/Apache License 2.0
- [botocore](#)/Apache License 2.0
- [dateutil](#)/PSF License

- [docutils](#)/BSD 授權、GNU 一般公有授權 (GPL)、Python 軟體基金會授權、公有網域
- [jmespath](#)/MIT License
- [s3transfer](#)/Apache License 2.0
- [urllib3](#)/MIT License

此連接器根據 [Greengrass 核心軟體授權合約](#) 發行。

變更記錄

下表說明每個版本連接器的變更。

版本	變更
4	新增 IsolationMode 參數以設定連接器的容器化模式。
3	將 Lambda 執行時間升級至 Python 3.7，這會變更執行時間需求。
2	可減少過多記錄的修正。
1	初始版本。

Greengrass 群組一次只能包含一個版本的連接器。若要取得有關升級連接器版本的資訊，請參閱[the section called “升級連接器版本”](#)。

另請參閱

- [使用連接器整合服務和通訊協定](#)
- [the section called “連接器入門 \(主控台\)”](#)
- [the section called “連接器入門 \(CLI\)”](#)
- AWS IoT Analytics 使用者指南中的[什麼是 AWS IoT Analytics ?](#)

IoT 乙太網路 IP 通訊協定轉接器連接器

IoT 乙太網路 IP 通訊協定轉接器[連接器](#)會使用乙太網路/IP 通訊協定從本機裝置收集資料。您可以使用此連接器從多個裝置收集資料，並將其發佈至StreamManager訊息串流。

您也可以將此連接器與 IoT SiteWise 連接器和 IoT SiteWise 閘道搭配使用。您的閘道必須提供連接器的組態。如需詳細資訊，請參閱 IoT SiteWise 使用者指南中的[設定乙太網路/IP \(EIP\) 來源](#)。

Note

此連接器以[無容器](#)隔離模式執行，因此您可以將其部署到在 Docker 容器中執行的 AWS IoT Greengrass 群組。

此連接器具有下列版本。

版本	ARN
2 (建議)	arn:aws:greengrass: <i>region</i> ::/connectors/IoTEIPProtocolAdaptor/versions/2
1	arn:aws:greengrass: <i>region</i> ::/connectors/IoTEIPProtocolAdaptor/versions/1

如需版本變更的詳細資訊，請參閱 [Changelog](#)。

要求

此連接器有下列要求：

Version 1 and 2

- AWS IoT Greengrass 核心軟體 1.10.2 版或更新版本。
- AWS IoT Greengrass 群組上已啟用串流管理員。
- 安裝在核心裝置上並新增至PATH環境變數的 Java 8。
- 至少 256 MB 的額外 RAM。此需求是 AWS IoT Greengrass 核心記憶體需求以外的需求。

Note

此連接器僅適用於下列區域：

- cn-north-1
- ap-southeast-1
- ap-southeast-2
- eu-central-1
- eu-west-1
- us-east-1
- us-west-2

連接器參數

此連接器支援下列參數：

LocalStoragePath

IoT SiteWise 連接器可寫入持久性資料的 AWS IoT Greengrass 主機目錄。預設目錄為 `/var/sitewise`。

AWS IoT 主控台顯示名稱：本機儲存路徑

必要：false

類型：string

有效模式：`^\s*$|\/`。

ProtocolAdapterConfiguration

連接器從 收集資料或連線的一組乙太網路/IP 收集器組態。這可以是空白清單。

AWS IoT 主控台顯示名稱：通訊協定轉接器組態

必要：true

類型：格式正確的 JSON 字串，定義一組支援的意見回饋組態。

以下是 的範例ProtocolAdapterConfiguration：

```
{
  "sources": [
```

```

{
  "type": "EIPSource",
  "name": "TestSource",
  "endpoint": {
    "ipAddress": "52.89.2.42",
    "port": 44818
  },
  "destination": {
    "type": "StreamManager",
    "streamName": "MyOutput_Stream",
    "streamBufferSize": 10
  },
  "destinationPathPrefix": "EIPSource_Prefix",
  "propertyGroups": [
    {
      "name": "DriveTemperatures",
      "scanMode": {
        "type": "POLL",
        "rate": 10000
      },
      "tagPathDefinitions": [
        {
          "type": "EIPTagPath",
          "path": "arrayREAL[0]",
          "dstDataType": "double"
        }
      ]
    }
  ]
}

```

建立範例連接器 (AWS CLI)

下列 CLI 命令 `ConnectorDefinition` 會使用包含 IoT 乙太網路 IP 通訊協定轉接器連接器的初始版本來建立。

```

aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version
'{
  "Connectors": [
    {

```

```

      "Id": "MyIoTEIPProtocolConnector",
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/
IoTEIPProtocolAdaptor/versions/2",
      "Parameters": {
        "ProtocolAdaptorConfiguration": "{ \"sources\": [{ \"type
\": \"EIPSource\", \"name\": \"Source1\", \"endpoint\": { \"ipAddress\":
\"54.245.77.218\", \"port\": 44818 }, \"destinationPathPrefix\": \"EIPConnector_Prefix
\", \"propertyGroups\": [{ \"name\": \"Values\", \"scanMode\": { \"type\": \"POLL\",
\"rate\": 2000 }, \"tagPathDefinitions\": [{ \"type\": \"EIPTagPath\", \"path\":
\"arrayREAL[0]\", \"dstDataType\": \"double\" }]]]]}",
        "LocalStoragePath": "/var/MyIoTEIPProtocolConnectorState"
      }
    }
  ]
}'

```

Note

此連接器中的 Lambda 函數具有[長期生命週期](#)。

輸入資料

此連接器不接受 MQTT 訊息做為輸入資料。

輸出資料

此連接器會將資料發佈至 StreamManager。您必須設定目的地訊息串流。輸出訊息的結構如下：

```

{
  "alias": "string",
  "messages": [
    {
      "name": "string",
      "value": boolean|double|integer|string,
      "timestamp": number,
      "quality": "string"
    }
  ]
}

```

授權

IoT 乙太網路 IP 通訊協定轉接器連接器包含下列第三方軟體/授權：

- [Ethernet/IP 用戶端](#)
- [MapDB](#)
- [斯德羅](#)

此連接器根據 [Greengrass 核心軟體授權合約](#) 發行。

變更記錄

下表說明連接器每個版本中的變更。

版本	改變	日期
2	此版本包含錯誤修正。	2021 年 12 月 23 日
1	初始版本。	2020 年 12 月 15 日

Greengrass 群組一次只能包含一個版本的連接器。若要取得有關升級連接器版本的資訊，請參閱[the section called “升級連接器版本”](#)。

另請參閱

- [使用連接器整合服務和通訊協定](#)
- [the section called “連接器入門 \(主控台\)”](#)
- [the section called “連接器入門 \(CLI\)”](#)

IoT SiteWise 連接器

IoT SiteWise 連接器會將本機裝置和設備資料傳送至其中的資產屬性 AWS IoT SiteWise。您可以使用此連接器從多個 OPC-UA 伺服器收集資料，並將其發佈至 IoT SiteWise。連接器會將資料傳送至目前 AWS 帳戶 和區域中的資產屬性。

Note

IoT SiteWise 是一項全受管服務，可從工業裝置和設備收集、處理和視覺化資料。您可以設定資產屬性，以處理從此連接器傳送至資產測量內容的原始資料。例如，您可以定義一個將裝置的攝氏溫度資料點轉換為華氏溫度的轉換屬性，或者定義計算每小時平均溫度的指標屬性。如需詳細資訊，請參閱《AWS IoT SiteWise 使用者指南》中的[什麼是AWS IoT SiteWise ?](#)。

連接器使用從 OPC-UA 伺服器傳送的 OPC-UA 資料串流路徑，將資料傳送至 IoT SiteWise。例如，資料串流路徑 `/company/windfarm/3/turbine/7/temperature` 可能代表 3 號風力發電廠 7 號渦輪機的溫度感應器。如果 AWS IoT Greengrass 核心失去與網際網路的連線，連接器會快取資料，直到成功連線至 AWS 雲端為止。您可以設定用於快取資料的最大磁碟緩衝區大小。如果快取大小超過磁碟緩衝區大小上限，則連接器會捨棄佇列中最舊的資料。

設定和部署 IoT SiteWise 連接器之後，您可以在 [IoT SiteWise 主控台](#) 中新增閘道和 OPC-UA 來源。在主控台中設定來源時，您可以篩選或加上 IoT SiteWise 連接器傳送的 OPC-UA 資料串流路徑。如需完成設定閘道和來源的指示，請參閱 AWS IoT SiteWise 《使用者指南》中的[新增閘道](#)。

IoT SiteWise 只會從您已映射至 IoT SiteWise 資產測量屬性的資料串流接收資料。若要將資料串流對應至資產屬性，您可以將屬性的別名設定為和 OPC-UA 資料串流路徑相同。若要了解如何定義資產模型和建立資產，請參閱 AWS IoT SiteWise 《使用者指南》中的[建立工業資產的模型](#)。

備註

您可以使用串流管理員，從 OPC-UA 伺服器以外的來源將資料上傳至 IoT SiteWise。串流管理員也提供可自訂的持久性和頻寬管理支援。如需詳細資訊，請參閱[管理資料串流](#)。此連接器以[無容器](#)隔離模式執行，因此您可以將其部署到在 Docker 容器中執行的 Greengrass 群組。

此連接器具有下列版本。

版本	ARN
12 (建議)	<code>arn:aws:greengrass: <i>region</i> :/:connectors/IoTSiteWise/versions/ 12</code>

版本	ARN
11	arn:aws:greengrass: <i>region</i> ::/ connectors/IoTSiteWise/v ersions/ 11
10	arn:aws:greengrass: <i>region</i> ::/ connectors/IoTSiteWise/v ersions/ 10
9	arn:aws:greengrass: <i>region</i> ::/ connectors/IoTSiteWise/v ersions/ 9
8	arn:aws:greengrass: <i>region</i> ::/ connectors/IoTSiteWise/v ersions/ 8
7	arn:aws:greengrass: <i>region</i> ::/ connectors/IoTSiteWise/v ersions/ 7
6	arn:aws:greengrass: <i>region</i> ::/ connectors/IoTSiteWise/v ersions/ 6
5	arn:aws:greengrass: <i>region</i> ::/ connectors/IoTSiteWise/v ersions/ 5
4	arn:aws:greengrass: <i>region</i> ::/ connectors/IoTSiteWise/v ersions/ 4
3	arn:aws:greengrass: <i>region</i> ::/ connectors/IoTSiteWise/v ersions/ 3

版本	ARN
2	arn:aws:greengrass: <i>region</i> ::/connectors/IoTSiteWise/versions/ 2
1	arn:aws:greengrass: <i>region</i> ::/connectors/IoTSiteWise/versions/ 1

如需版本變更的詳細資訊，請參閱 [Changelog](#)。

要求

此連接器有下列要求：

Version 9, 10, 11, and 12

Important

此版本推出新需求：AWS IoT Greengrass Core 軟體 v1.10.2 和 [串流管理員](#)。

- AWS IoT Greengrass 核心軟體 v1.10.2。
- 已在 Greengrass 群組上啟用 [串流管理員](#)。
- Java 8 已安裝在核心裝置上並已新增至 PATH 環境變數。
- 此連接器只能在同時支援 [AWS IoT Greengrass](#) 和 [IoT SiteWise](#) 的 Amazon Web Services 區域中使用。
- 新增至 Greengrass 群組角色的 IAM 政策。此角色允許 AWS IoT Greengrass 群組存取目標根資產及其子系上的 `iotsitewise:BatchPutAssetPropertyValue` 動作，如下列範例所示。您可以從 Condition 政策中移除，以允許連接器存取所有 IoT SiteWise 資產。

JSON

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

```

        "Effect": "Allow",
        "Action": "iotsitewise:BatchPutAssetPropertyValue",
        "Resource": "*",
        "Condition": {
            "StringLike": {
                "iotsitewise:assetHierarchyPath": [
                    "/root node asset ID",
                    "/root node asset ID/*"
                ]
            }
        }
    }
}

```

如需詳細資訊，請參閱 [《IAM 使用者指南》](#) 中的 [新增和移除 IAM 政策](#)。

Versions 6, 7, and 8

Important

此版本推出新需求：AWS IoT Greengrass Core 軟體 v1.10.0 和 [串流管理員](#)。

- AWS IoT Greengrass 核心軟體 v1.10.0。
- 已在 Greengrass 群組上啟用 [串流管理員](#)。
- Java 8 已安裝在核心裝置上並已新增至 PATH 環境變數。
- 此連接器只能在同時支援 [AWS IoT Greengrass](#) 和 [IoT SiteWise](#) 的 Amazon Web Services 區域中使用。
- 新增至 Greengrass 群組角色的 IAM 政策。此角色允許 AWS IoT Greengrass 群組存取目標根資產及其子系上的 `iotsitewise:BatchPutAssetPropertyValue` 動作，如下列範例所示。您可以從 Condition 政策中移除，以允許連接器存取所有 IoT SiteWise 資產。

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",

```

```

        "Action": "iotsitewise:BatchPutAssetPropertyValue",
        "Resource": "*",
        "Condition": {
            "StringLike": {
                "iotsitewise:assetHierarchyPath": [
                    "/root node asset ID",
                    "/root node asset ID/*"
                ]
            }
        }
    }
}

```

如需詳細資訊，請參閱 [《IAM 使用者指南》](#) 中的 [新增和移除 IAM 政策](#)。

Version 5

- AWS IoT Greengrass 核心軟體 v1.9.4。
- Java 8 已安裝在核心裝置上並已新增至 PATH 環境變數。
- 此連接器只能在同時支援 [AWS IoT Greengrass](#) 和 [IoT SiteWise](#) 的 Amazon Web Services 區域中使用。
- 新增至 Greengrass 群組角色的 IAM 政策。此角色允許 AWS IoT Greengrass 群組存取目標根資產及其子系上的 `iotsitewise:BatchPutAssetPropertyValue` 動作，如下列範例所示。您可以從 Condition 政策中移除，以允許連接器存取所有 IoT SiteWise 資產。

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iotsitewise:BatchPutAssetPropertyValue",
      "Resource": "*",
      "Condition": {
        "StringLike": {
          "iotsitewise:assetHierarchyPath": [
            "/root node asset ID",
            "/root node asset ID/*"
          ]
        }
      }
    }
  ]
}

```

```

    }
  }
}

```

如需詳細資訊，請參閱 [《IAM 使用者指南》中的新增和移除 IAM 政策](#)。

Version 4

- AWS IoT Greengrass 核心軟體 v1.10.0。
- Java 8 已安裝在核心裝置上並已新增至 PATH 環境變數。
- 此連接器只能在同時支援 [AWS IoT Greengrass](#) 和 [IoT SiteWise](#) 的 Amazon Web Services 區域中使用。
- 新增至 Greengrass 群組角色的 IAM 政策。此角色允許 AWS IoT Greengrass 群組存取目標根資產及其子系上的 `iotsitewise:BatchPutAssetPropertyValue` 動作，如下列範例所示。您可以從 Condition 政策中移除，以允許連接器存取所有 IoT SiteWise 資產。

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iotsitewise:BatchPutAssetPropertyValue",
      "Resource": "*",
      "Condition": {
        "StringLike": {
          "iotsitewise:assetHierarchyPath": [
            "/root node asset ID",
            "/root node asset ID/*"
          ]
        }
      }
    }
  ]
}

```

如需詳細資訊，請參閱 [《IAM 使用者指南》中的新增和移除 IAM 政策](#)。

Version 3

- AWS IoT Greengrass 核心軟體 v1.9.4。
- Java 8 已安裝在核心裝置上並已新增至 PATH 環境變數。
- 此連接器只能在同時支援 [AWS IoT Greengrass](#) 和 [IoT SiteWise](#) 的 Amazon Web Services 區域中使用。
- 新增至 Greengrass 群組角色的 IAM 政策。此角色允許 AWS IoT Greengrass 群組存取目標根資產及其子系上的 `iotsitewise:BatchPutAssetPropertyValue` 動作，如下列範例所示。您可以從 Condition 政策中移除，以允許連接器存取所有 IoT SiteWise 資產。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iotsitewise:BatchPutAssetPropertyValue",
      "Resource": "*",
      "Condition": {
        "StringLike": {
          "iotsitewise:assetHierarchyPath": [
            "/root node asset ID",
            "/root node asset ID/*"
          ]
        }
      }
    }
  ]
}
```

如需詳細資訊，請參閱 [《IAM 使用者指南》](#) 中的新增和移除 IAM 政策。

Versions 1 and 2

- AWS IoT Greengrass 核心軟體 v1.9.4。
- Java 8 已安裝在核心裝置上並已新增至 PATH 環境變數。
- 此連接器只能在同時支援 [AWS IoT Greengrass](#) 和 [IoT SiteWise](#) 的 Amazon Web Services 區域中使用。

- 新增至 Greengrass 群組角色的 IAM 政策，允許存取目標根資產及其子系上的 和 AWS IoT Core `iotsitewise:BatchPutAssetPropertyValue` 動作，如下列範例所示。您可以從 Condition 政策中移除，以允許連接器存取所有 IoT SiteWise 資產。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iotsitewise:BatchPutAssetPropertyValue",
      "Resource": "*",
      "Condition": {
        "StringLike": {
          "iotsitewise:assetHierarchyPath": [
            "/root node asset ID",
            "/root node asset ID/*"
          ]
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Connect",
        "iot:DescribeEndpoint",
        "iot:Publish",
        "iot:Receive",
        "iot:Subscribe"
      ],
      "Resource": "*"
    }
  ]
}
```

如需詳細資訊，請參閱《IAM 使用者指南》中的[新增和移除 IAM 身分許可](#)。

Parameters

Versions 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, and 12

SiteWiseLocalStoragePath

IoT SiteWise 連接器可寫入持久性資料的 AWS IoT Greengrass 主機目錄。預設為 `/var/sitewise`。

AWS IoT 主控台中的顯示名稱：本機儲存路徑

必要：false

類型：string

有效模式：`^\s*$|\/`。

AWSecretsArnList

中的秘密清單 AWS Secrets Manager，其中每個都包含 OPC-UA 使用者名稱和密碼金鑰/值對。每個秘密都必須是金鑰/值對類型的秘密。

在 AWS IoT 主控台中顯示名稱：OPC-UA 使用者名稱/密碼秘密的 ARNs 清單

必要：false

類型：JsonArrayOfStrings

有效模式：`\[( ? , ? ?\"(arn:(aws(-[a-z]+)*):secretsmanager:[a-z0-9\\-]+:[0-9]{12}:secret:([a-zA-Z0-9\\-]+\/)*[a-zA-Z0-9\\/_+=, .@\\-]+-[a-zA-Z0-9]+)*\")*\]`

MaximumBufferSize

IoT SiteWise 磁碟用量的大小上限，以 GB 為單位。預設為 10GB。

AWS IoT 主控台中的顯示名稱：磁碟緩衝區大小上限

必要：false

類型：string

有效模式：`^\s*$|[0-9]+`

Version 1

SiteWiseLocalStoragePath

IoT SiteWise 連接器可寫入持久性資料的 AWS IoT Greengrass 主機目錄。預設為 `/var/sitewise`。

AWS IoT 主控台顯示名稱：本機儲存路徑

必要：false

類型：string

有效模式：`^\s*$|\/.`

SiteWiseOpcuaUserIdentityTokenSecretArn

中 AWS Secrets Manager 包含 OPC-UA 使用者名稱和密碼金鑰值對的秘密。此秘密必須是索引鍵/值組類型的秘密。

在 AWS IoT 主控台顯示名稱：OPC-UA 使用者名稱/密碼秘密的 ARN

必要：false

類型：string

有效模式：`^$|arn:(aws(-[a-z]+)*):secretsmanager:[a-z0-9\\-]+:[0-9]{12}:secret:([a-zA-Z0-9\\-]+/)*[a-zA-Z0-9/_+=, .@\\-]+-[a-zA-Z0-9]+`

SiteWiseOpcuaUserIdentityTokenSecretArn-ResourceId

AWS IoT Greengrass 群組中參考 OPC-UA 使用者名稱和密碼秘密的秘密資源。

在 AWS IoT 主控台顯示名稱：OPC-UA 使用者名稱/密碼秘密資源

必要：false

類型：string

有效模式：`^$|.+`

MaximumBufferSize

IoT SiteWise 磁碟用量的 GB 大小上限。預設為 10GB。

AWS IoT 主控台顯示名稱：磁碟緩衝區大小上限

必要：false

類型：string

有效模式：`^\s*$|[0-9]+`

建立範例連接器 (AWS CLI)

下列 AWS CLI 命令 `ConnectorDefinition` 會使用包含 IoT SiteWise 連接器的初始版本來建立。

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-  
version '{  
  "Connectors": [  
    {  
      "Id": "MyIoTSiteWiseConnector",  
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/IoTSiteWise/  
versions/11"  
    }  
  ]  
}'
```

Note

此連接器中的 Lambda 函數具有[長期生命週期](#)。

在 AWS IoT Greengrass 主控台中，您可以從群組的連接器頁面新增連接器。如需詳細資訊，請參閱[the section called “連接器入門 \(主控台\)”](#)。

輸入資料

此連接器不接受 MQTT 訊息做為輸入資料。

輸出資料

此連接器不會將 MQTT 訊息發佈為輸出資料。

限制

此連接器受以下 IoT SiteWise 施加的所有限制的約束，包括以下內容。如需詳細資訊，請參閱《》中的[AWS IoT SiteWise 端點和配額](#) AWS 一般參考。

- 每個閘道的數量上限 AWS 帳戶。
- 每個閘道的 OPC-UA 來源數目上限。
- 每個存放的timestamp-quality-value(TQV) 資料點的最大速率 AWS 帳戶。
- 每個資產屬性儲存的 TQV 資料點最大速率。

授權

Version 9, 10, 11, and 12

IoT SiteWise 連接器包含下列第三方軟體/授權：

- [MapDB](#)
- [埃爾森](#)
- [Eclipse Milo](#)

此連接器根據 [Greengrass 核心軟體授權合約](#) 發行。

Versions 6, 7, and 8

IoT SiteWise 連接器包含下列第三方軟體/授權：

- [Milo](#) / EDL 1.0

此連接器根據 [Greengrass 核心軟體授權合約](#) 發行。

Versions 1, 2, 3, 4, and 5

IoT SiteWise 連接器包含下列第三方軟體/授權：

- [Milo](#) / EDL 1.0
- [Chronicle-Queue](#) / Apache License 2.0

此連接器根據 [Greengrass 核心軟體授權合約](#) 發行。

變更記錄

下表說明每個版本的連接器的變更。

版本	改變	Date
12	此版本包含錯誤修正。	2021 年 12 月 22 日
11	- 支援包含隱藏或不可列印字元的字串。字串傳送到之前，會自動移除隱藏和無法列印的字元 AWS 雲端。 - 修正 IoT SiteWise 閘道無限重試無效請求的問題。 - 修正當 IoT SiteWise 閘道連線到高頻率資料來源時，導致檢查點損毀的問題。 - 改善錯誤訊息，以協助疑難排解閘道組態。	2021 年 3 月 24 日
10	設定為StreamManager 在來源連線中斷並重新建立時改善處理。當沒有可用ServerTimestamp 時，此版本也接受 OPC-UA SourceTimestamp 值與。	2021 年 1 月 22 日
9	為自訂 Greengrass StreamManager 串流目的地、OPC-UA 不動作帶、自訂掃描模式和自訂掃描速率啟動支援。也包括在從 IoT SiteWise 閘道進行組態更新期間改善的效能。	2020 年 12 月 15 日
8	改善連接器遇到間歇性網路連線時的穩定性。	2020 年 11 月 19 日
7	修正閘道指標的問題。	2020 年 8 月 14 日

版本	改變	Date
6	新增對 CloudWatch 指標和自動探索新 OPC-UA 標籤的支援。此版本需要 串流管理員 和 AWS IoT Greengrass Core 軟體 v1.10.0 或更高版本。	2020 年 4 月 29 日
5	修正 AWS IoT Greengrass Core 軟體 v1.9.4 的相容性問題。	2020 年 2 月 12 日
4	已修正 OPC-UA 伺服器重新連線的問題。	2020 年 2 月 7 日
3	移除 iot:* 許可需求。	2019 年 12 月 17 日
2	新增 OPC-UA 秘密資源的支援。	2019 年 12 月 10 日
1	初始版本。	2019 年 12 月 2 日

Greengrass 群組一次只能包含一個版本的連接器。若要取得有關升級連接器版本的資訊，請參閱[the section called “升級連接器版本”](#)。

另請參閱

- [使用連接器整合服務和通訊協定](#)
- [the section called “連接器入門 \(主控台\)”](#)
- [the section called “連接器入門 \(CLI\)”](#)
- 請參閱AWS IoT SiteWise 《使用者指南》中的下列主題：
 - [什麼是 AWS IoT SiteWise ?](#)
 - [使用閘道](#)
 - [Gateway CloudWatch 指標](#)
 - [疑難排解 IoT SiteWise 閘道](#)

Kinesis Firehose

Kinesis Firehose [連接器](#)會透過 Amazon Data Firehose 交付串流將資料發佈至目的地，例如 Amazon S3、Amazon Redshift 或 Amazon OpenSearch Service。

此連接器是 Kinesis 交付串流的資料生產者。它會接收 MQTT 主題上的輸入資料，然後將資料傳送到指定的交付串流。然後，交付串流會將資料記錄傳送到設定的目的地 (例如，S3 儲存貯體)。

此連接器具有下列版本。

版本	ARN
5	arn:aws:greengrass: <i>region</i> ::/ connectors/KinesisFirehose/ versions/5
4	arn:aws:greengrass: <i>region</i> ::/ connectors/KinesisFirehose/ versions/4
3	arn:aws:greengrass: <i>region</i> ::/ connectors/KinesisFirehose/ versions/3
2	arn:aws:greengrass: <i>region</i> ::/ connectors/KinesisFirehose/ versions/2
1	arn:aws:greengrass: <i>region</i> ::/ connectors/KinesisFirehose/ versions/1

如需版本變更的詳細資訊，請參閱 [Changelog](#)。

要求

此連接器有下列要求：

Version 4 - 5

- AWS IoT Greengrass 核心軟體 v1.9.3 或更新版本。
- 安裝在核心裝置上並新增至 PATH 環境變數的 [Python](#) 3.7 或 3.8 版。

 Note

若要使用 Python 3.8，請執行下列命令來建立從預設 Python 3.7 安裝資料夾到已安裝 Python 3.8 二進位檔的符號連結。

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

這會設定您的裝置以符合 AWS IoT Greengrass 的 Python 需求。

- 設定的 Kinesis 交付串流。如需詳細資訊，請參閱 [《Amazon Kinesis Firehose 開發人員指南》中的建立 Amazon Data Firehose 交付串流](#)。Amazon Kinesis Firehose
- [Greengrass 群組角色](#)設定為允許目標交付串流上的 `firehose:PutRecord` 和 `firehose:PutRecordBatch` 動作，如下列範例 IAM 政策所示。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1528133056761",
      "Action": [
        "firehose:PutRecord",
        "firehose:PutRecordBatch"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:firehose:us-east-1:123456789012:deliverystream/stream-
name"
      ]
    }
  ]
}
```

此連接器可讓您動態覆寫輸入訊息承載中的預設交付串流。如果您的實作使用此功能，IAM 政策應包含所有目標串流做為資源。您可以為資源授予細微或條件式存取 (例如，使用萬用字元 * 命名配置)。

針對群組角色要求，您必須設定角色以授與必要的許可，並確認已將角色新增至群組。如需詳細資訊，請參閱 [the section called “管理群組角色 \(主控台\)”](#) 或 [the section called “管理群組角色 \(CLI\)”](#)。

Versions 2 - 3

- AWS IoT Greengrass 核心軟體 v1.7 或更新版本。
- 安裝在核心裝置上並新增至 PATH 環境變數的 [Python](#) 2.7 版。
- 設定的 Kinesis 交付串流。如需詳細資訊，請參閱 [《Amazon Kinesis Firehose 開發人員指南》中的建立 Amazon Data Firehose 交付串流](#)。Amazon Kinesis Firehose
- 設定為允許目標交付串流上 `firehose:PutRecord` 和 `firehose:PutRecordBatch` 動作的 [Greengrass 群組角色](#)，如下列範例 IAM 政策所示。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1528133056761",
      "Action": [
        "firehose:PutRecord",
        "firehose:PutRecordBatch"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:firehose:us-east-1:123456789012:deliverystream/stream-
name"
      ]
    }
  ]
}
```

此連接器可讓您動態覆寫輸入訊息承載中的預設交付串流。如果您的實作使用此功能，IAM 政策應包含所有目標串流做為資源。您可以為資源授予細微或條件式存取 (例如，使用萬用字元 * 命名配置)。

針對群組角色要求，您必須設定角色以授與必要的許可，並確認已將角色新增至群組。如需詳細資訊，請參閱 [the section called “管理群組角色 \(主控台\)”](#) 或 [the section called “管理群組角色 \(CLI\)”](#)。

Version 1

- AWS IoT Greengrass 核心軟體 v1.7 或更新版本。
- 安裝在核心裝置上並新增至 PATH 環境變數的 [Python 2.7](#) 版。
- 設定的 Kinesis 交付串流。如需詳細資訊，請參閱 [《Amazon Kinesis Firehose 開發人員指南》中的建立 Amazon Data Firehose 交付串流](#)。Amazon Kinesis Firehose
- 設定為允許對目標交付串流 `firehose:PutRecord` 執行動作的 [Greengrass 群組角色](#)，如下列範例 IAM 政策所示。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1528133056761",
      "Action": [
        "firehose:PutRecord"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:firehose:us-east-1:123456789012:deliverystream/stream-  
name"
      ]
    }
  ]
}
```

此連接器可讓您動態覆寫輸入訊息承載中的預設交付串流。如果您的實作使用此功能，IAM 政策應包含所有目標串流做為資源。您可以為資源授予細微或條件式存取 (例如，使用萬用字元 * 命名配置)。

針對群組角色要求，您必須設定角色以授與必要的許可，並確認已將角色新增至群組。如需詳細資訊，請參閱 [the section called “管理群組角色 \(主控台\)”](#) 或 [the section called “管理群組角色 \(CLI\)”](#)。

連接器參數

此連接器提供下列參數：

Versions 5

DefaultDeliveryStreamArn

要傳送資料之預設 Firehose 交付串流的 ARN。輸入訊息承載中的 `delivery_stream_arn` 屬性可以覆寫目的地串流。

Note

群組角色必須允許所有目標交付串流上的適當動作。如需詳細資訊，請參閱[the section called “要求”](#)。

在 AWS IoT 主控台中顯示名稱：預設交付串流 ARN

必要：true

類型：string

有效模式：`arn:aws:firehose:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):deliverystream/([a-zA-Z0-9_\-\.]+)$`

DeliveryStreamQueueSize

同樣交付串流的新記錄遭拒絕前，記憶體中保留的記錄數上限。最小值為 2000。

在 AWS IoT 主控台中顯示名稱：要緩衝的記錄數目上限（每個串流）

必要：true

類型：string

有效模式：`^([2-9]\\d{3}|[1-9]\\d{4,})$`

MemorySize

配置給此連接器的記憶體數量 (KB)。

AWS IoT 主控台中的顯示名稱：記憶體大小

必要：true

類型：string

有效模式：`^[0-9]+$`

PublishInterval

將記錄發佈至 Firehose 的間隔（以秒為單位）。若要停止批次處理，請將此數值設為 0。

在 AWS IoT 主控台中顯示名稱：發佈間隔

必要：true

類型：string

有效值：0 - 900

有效模式：`[0-9]|[1-9]\\d|[1-9]\\d\\d|900`

IsolationMode

此連接器的[容器化](#)模式。預設值為 `GreengrassContainer`，這表示連接器會在 AWS IoT Greengrass 容器內的隔離執行時間環境中執行。

Note

群組的預設容器化設定不會套用至連接器。

AWS IoT 主控台中的顯示名稱：容器隔離模式

必要：false

類型：string

有效值：`GreengrassContainer` 或 `NoContainer`

有效模式：`^NoContainer$|^GreengrassContainer$`

Versions 2 - 4

DefaultDeliveryStreamArn

要傳送資料之預設 Firehose 交付串流的 ARN。輸入訊息承載中的 `delivery_stream_arn` 屬性可以覆寫目的地串流。

 Note

群組角色必須允許所有目標交付串流上的適當動作。如需詳細資訊，請參閱[the section called “要求”](#)。

在 AWS IoT 主控台中顯示名稱：預設交付串流 ARN

必要：true

類型：string

有效模式：`arn:aws:firehose:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):deliverystream/([a-zA-Z0-9_-.]+)$`

DeliveryStreamQueueSize

同樣交付串流的新記錄遭拒絕前，記憶體中保留的記錄數上限。最小值為 2000。

在 AWS IoT 主控台中顯示名稱：要緩衝的記錄數量上限（每個串流）

必要：true

類型：string

有效模式：`^([2-9]\\d{3}|[1-9]\\d{4,})$`

MemorySize

配置給此連接器的記憶體數量 (KB)。

AWS IoT 主控台顯示名稱：記憶體大小

必要：true

類型：string

有效模式：`^[0-9]+$`

PublishInterval

將記錄發佈至 Firehose 的間隔（以秒為單位）。若要停止批次處理，請將此數值設為 0。

AWS IoT 主控台中的顯示名稱：發佈間隔

必要：`true`

類型：`string`

有效值：`0 - 900`

有效模式：`[0-9]|[1-9]\\d|[1-9]\\d\\d|900`

Version 1

DefaultDeliveryStreamArn

要傳送資料之預設 Firehose 交付串流的 ARN。輸入訊息承載中的 `delivery_stream_arn` 屬性可以覆寫目的地串流。

Note

群組角色必須允許所有目標交付串流上的適當動作。如需詳細資訊，請參閱[the section called “要求”](#)。

在 AWS IoT 主控台中顯示名稱：預設交付串流 ARN

必要：`true`

類型：`string`

有效模式：`arn:aws:firehose:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):deliverystream/([a-zA-Z0-9_\-.]+)$`

Example

建立連接器範例 (AWS CLI)

下列 CLI 命令ConnectorDefinition會使用包含連接器的初始版本來建立。

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
  "Connectors": [
    {
      "Id": "MyKinesisFirehoseConnector",
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/KinesisFirehose/
versions/5",
      "Parameters": {
        "DefaultDeliveryStreamArn": "arn:aws:firehose:region:account-
id:deliverystream/stream-name",
        "DeliveryStreamQueueSize": "5000",
        "MemorySize": "65535",
        "PublishInterval": "10",
        "IsolationMode" : "GreengrassContainer"
      }
    }
  ]
}'
```

在 AWS IoT Greengrass 主控台中，您可以從群組的連接器頁面新增連接器。如需詳細資訊，請參閱[the section called “連接器入門 \(主控台\)”](#)。

輸入資料

此連接器接受 MQTT 主題上的串流內容，然後將內容傳送到目標交付串流。它接受兩種輸入資料：

- kinesisfirehose/message 主題上的 JSON 資料。
- kinesisfirehose/message/binary/# 主題上的二進位資料。

Versions 2 - 5

主題篩選條件：kinesisfirehose/message

使用此主題傳送包含 JSON 資料的訊息。

訊息屬性

request

要傳送到交付串流和目標交付串流 (如果與預設串流不同) 的資料。

必要：true

類型：object 包含下列屬性：

data

要傳送到交付串流的資料。

必要：true

類型：string

delivery_stream_arn

目標 Kinesis 交付串流的 ARN。包含此屬性來覆寫預設的交付串流。

必要：false

類型：string

有效模式：arn:aws:firehose:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):deliverystream/([a-zA-Z0-9_\-\.]+)\$

id

請求的任意 ID。此屬性用於將輸入請求映射到輸出回應。當指定時，回應物件中的 id 屬性會設為這個值。如果您不使用此功能，您可以省略此屬性或指定空白字串。

必要：false

類型：string

有效模式：.*

範例輸入

```
{
  "request": {
    "delivery_stream_arn": "arn:aws:firehose:region:account-  
id:deliverystream/stream2-name",
    "data": "Data to send to the delivery stream."
  },
  "id": "request123"
}
```

主題篩選條件：kinesisfirehose/message/binary/#

使用此主題傳送包含二進位資料的訊息。連接器不剖析二進位資料。資料會依現狀串流。

若要將輸入請求映射到輸出回應，請將訊息主題中的 # 萬用字元換成任意請求 ID。例如，如果您將訊息發佈到 kinesisfirehose/message/binary/request123，回應物件中的 id 屬性會設為 request123。

如果您不想將請求映射到回應，您可以將訊息發佈到 kinesisfirehose/message/binary/。請務必包含結尾的斜線。

Version 1

主題篩選條件：kinesisfirehose/message

使用此主題傳送包含 JSON 資料的訊息。

訊息屬性

request

要傳送到交付串流和目標交付串流 (如果與預設串流不同) 的資料。

必要：true

類型：object 包含下列屬性：

data

要傳送到交付串流的資料。

必要：true

類型：string

delivery_stream_arn

目標 Kinesis 交付串流的 ARN。包含此屬性來覆寫預設的交付串流。

必要：false

類型：string

有效模式：arn:aws:firehose:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):deliverystream/([a-zA-Z0-9_\-\.]+)\$

id

請求的任意 ID。此屬性用於將輸入請求映射到輸出回應。當指定時，回應物件中的 id 屬性會設為這個值。如果您不使用此功能，您可以省略此屬性或指定空白字串。

必要：false

類型：string

有效模式：.*

範例輸入

```
{
  "request": {
    "delivery_stream_arn": "arn:aws:firehose:region:account-
id:deliverystream/stream2-name",
    "data": "Data to send to the delivery stream."
  },
  "id": "request123"
}
```

主題篩選條件：kinesisfirehose/message/binary/#

使用此主題傳送包含二進位資料的訊息。連接器不剖析二進位資料。資料會依現狀串流。

若要將輸入請求映射到輸出回應，請將訊息主題中的 # 萬用字元換成任意請求 ID。例如，如果您將訊息發佈到 kinesisfirehose/message/binary/request123，回應物件中的 id 屬性會設為 request123。

如果您不想將請求映射到回應，您可以將訊息發佈到 kinesisfirehose/message/binary/。請務必包含結尾的斜線。

輸出資料

這個連接器會將狀態資訊發佈為輸出資料，且主題為 MQTT。

Versions 2 - 5

訂閱中的主題篩選條件

kinesisfirehose/message/status

範例輸出

回應中包含批次中所傳送之各筆資料記錄的狀態。

```
{
  "response": [
    {
      "ErrorCode": "error",
      "ErrorMessage": "test error",
      "id": "request123",
      "status": "fail"
    },
    {
      "firehose_record_id": "xyz2",
      "id": "request456",
      "status": "success"
    },
    {
      "firehose_record_id": "xyz3",
      "id": "request890",
      "status": "success"
    }
  ]
}
```

Note

如果連接器偵測到可重試的錯誤（例如連線錯誤），它會在下一個批次中重試發佈。指數退避由 AWS SDK 處理。因可重試錯誤而失敗的請求會加回佇列末端，進一步發佈。

Version 1

訂閱中的主題篩選條件

kinesisfirehose/message/status

範例輸出：成功

```
{
  "response": {
    "firehose_record_id": "1lxfuuuFomkpJYzt/34ZU/r8JYPf8Wyf7AXqlXm",
```

```
    "status": "success"
  },
  "id": "request123"
}
```

範例輸出：失敗

```
{
  "response" : {
    "error": "ResourceNotFoundException",
    "error_message": "An error occurred (ResourceNotFoundException) when calling the PutRecord operation: Firehose test1 not found under account 123456789012.",
    "status": "fail"
  },
  "id": "request123"
}
```

用量範例

使用下列高階步驟來設定範例 Python 3.7 Lambda 函數，您可以用來嘗試連接器。

Note

- 如果您使用其他 Python 執行期，則可以建立從 Python3.x 到 Python 3.7 的符號連結。
- [連接器入門 \(主控台\)](#) 和 [連接器入門 \(CLI\)](#) 主題包含詳細步驟，說明如何設定和部署範例 Twilio 通知連接器。

1. 確定您符合連接器的[要求](#)。

針對群組角色要求，您必須設定角色以授與必要的許可，並確認已將角色新增至群組。如需詳細資訊，請參閱 [the section called “管理群組角色 \(主控台\)”](#) 或 [the section called “管理群組角色 \(CLI\)”](#)。

2. 建立並發佈 Lambda 函數，將輸入資料傳送至連接器。

將範例程式碼儲存為 PY 檔案。下載並解壓縮[AWS IoT Greengrass 適用於 Python 的 Core SDK](#)。然後，建立在根層級包含 PY 檔案和 greengrasssdk 資料夾的 zip 套件。此 zip 套件是您上傳至的部署套件 AWS Lambda。

建立 Python 3.7 Lambda 函數後，發佈函數版本並建立別名。

3. 設定 Greengrass 群組。

- a. 依別名新增 Lambda 函數（建議）。將 Lambda 生命週期設定為長期（或在 CLI "Pinned": true 中）。
- b. 新增連接器並設定其[參數](#)。
- c. 新增訂閱，允許連接器在支援主題篩選條件上接收 [JSON 輸入資料](#)並傳送[輸出資料](#)。
 - 將 Lambda 函數設定為來源，將連接器設定為目標，並使用支援的輸入主題篩選條件。
 - 將連接器設為來源、將 AWS IoT Core 設為目標，並使用支援的輸出主題篩選條件。您可以使用此訂閱在 AWS IoT 主控台中檢視狀態訊息。

4. 部署群組。

5. 在 AWS IoT 主控台的測試頁面上，訂閱輸出資料主題，以檢視連接器的狀態訊息。Lambda 函數範例為長期函數，並在部署群組後立即開始傳送訊息。

完成測試後，您可以將 Lambda 生命週期設定為隨需（或在 CLI "Pinned": false 中）並部署群組。這會讓函數停止傳送訊息。

範例

下列範例 Lambda 函數會將輸入訊息傳送至連接器。此訊息包含 JSON 資料。

```
import greengrasssdk
import time
import json

iot_client = greengrasssdk.client('iot-data')
send_topic = 'kinesisfirehose/message'

def create_request_with_all_fields():
    return {
        "request": {
            "data": "Message from Firehose Connector Test"
        },
        "id" : "req_123"
    }

def publish_basic_message():
    messageToPublish = create_request_with_all_fields()
```

```
print("Message To Publish: ", messageToPublish)
iot_client.publish(topic=send_topic,
    payload=json.dumps(messageToPublish))

publish_basic_message()

def lambda_handler(event, context):
    return
```

授權

Kinesis Firehose 連接器包含下列第三方軟體/授權：

- [適用於 Python \(Boto3\) 的 AWS SDK](#)/Apache License 2.0
- [botocore](#)/Apache License 2.0
- [dateutil](#)/PSF License
- [docutils](#)/BSD 授權、GNU 一般公有授權 (GPL)、Python 軟體基金會授權、公有網域
- [jmespath](#)/MIT License
- [s3transfer](#)/Apache License 2.0
- [urllib3](#)/MIT License

此連接器根據 [Greengrass 核心軟體授權合約](#) 發行。

變更記錄

下表說明每個版本連接器的變更。

版本	變更
5	已新增 IsolationMode 參數，以設定連接器的容器化模式。
4	將 Lambda 執行時間升級至 Python 3.7，這會變更執行時間需求。
3	修正以降低過多記錄，以及其他次要錯誤修正。
2	新增支援以指定間隔將批次資料記錄傳送至 Firehose。

版本	變更
	- 群組角色中也需要有 firehose: PutRecordBatch 動作。 - 新的 MemorySize、DeliveryStreamQueueSize 和 PublishInterval 參數。 - 輸出訊息包含已發佈資料記錄的一系列狀態回應。
1	初始版本。

Greengrass 群組一次只能包含一個版本的連接器。若要取得有關升級連接器版本的資訊，請參閱[the section called “升級連接器版本”](#)。

另請參閱

- [使用連接器整合服務和通訊協定](#)
- [the section called “連接器入門 \(主控台\)”](#)
- [the section called “連接器入門 \(CLI\)”](#)
- 《[Amazon Kinesis 開發人員指南](#)》中的[什麼是 Amazon Kinesis Data Firehose ?](#) Amazon Kinesis

ML 回饋連接器

Warning

此連接器已進入延長的生命週期階段，而且 AWS IoT Greengrass 不會發行提供現有功能、增強功能、安全修補程式或錯誤修正的更新。如需詳細資訊，請參閱[AWS IoT Greengrass Version 1 維護政策](#)。

ML Feedback 連接器可讓您更輕鬆地存取機器學習 (ML) 模型資料，以進行模型重新訓練和分析。連接器：

- 將 ML 模型使用的輸入資料（範例）上傳至 Amazon S3。模型輸入可以採用任何格式，例如影像、JSON 或音訊。在範例上傳至雲端之後，您可以使用它們來重新訓練模型，以改善其預測的精確

性和準確性。例如，您可以使用 [SageMaker AI Ground Truth](#) 來標記範例，並使用 [SageMaker AI](#) 來重新訓練模型。

- 以 MQTT 訊息形式發佈模型的預測結果。這可讓您即時監控和分析模型的推論品質。您也可以存放預測結果，並使用它們來分析隨時間變化的趨勢。
- 將範例上傳和範例資料的指標發佈至 Amazon CloudWatch。

若要設定此連接器，請以 JSON 格式描述支援的意見回饋組態。意見回饋組態定義屬性，例如目的地 Amazon S3 儲存貯體、內容類型和[抽樣策略](#)。(抽樣策略用來決定要上傳哪些範例。)

您可以在下列案例中使用 ML Feedback 連接器：

- 使用使用者定義的 Lambda 函數。您的本機推論 Lambda 函數使用 AWS IoT Greengrass Machine Learning SDK 來叫用此連接器，並傳入目標意見回饋組態、模型輸入和模型輸出（預測結果）。如需範例，請參閱 [the section called “用量範例”](#)。
- 使用 [ML Image Classification 連接器](#) (v2)。若要搭配 ML Image Classification 連接器使用此連接器，請設定 ML Image Classification 連接器的 MLFeedbackConnectorConfigId 參數。
- 使用 [ML 物件偵測連接器](#)。若要搭配 ML 物件偵測連接器使用此連接器，請設定 ML 物件偵測連接器的 MLFeedbackConnectorConfigId 參數。

ARN : `arn:aws:greengrass:region::/connectors/MLFeedback/versions/1`

要求

此連接器有下列要求：

- AWS IoT Greengrass 核心軟體 1.9.3 版或更新版本。
- 安裝在核心裝置上並新增至 PATH 環境變數的 [Python](#) 3.7 或 3.8 版。

Note

若要使用 Python 3.8，請執行下列命令來建立從預設 Python 3.7 安裝資料夾到已安裝 Python 3.8 二進位檔的符號連結。

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

這會設定您的裝置以符合 AWS IoT Greengrass 的 Python 需求。

- 一或多個 Amazon S3 儲存貯體。您使用的儲存貯體數量取決於您的取樣策略。
- [Greengrass 群組角色](#) 設定為允許對目的地 Amazon S3 儲存貯體中的物件 `s3:PutObject` 執行動作，如下列範例 IAM 政策所示。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:PutObject",
      "Resource": [
        "arn:aws:s3:::bucket-name/*"
      ]
    }
  ]
}
```

政策應該包含所有目的地儲存貯體做為資源。您可以為資源授予細微或條件式存取 (例如，使用萬用字元 * 命名配置)。

針對群組角色要求，您必須設定角色以授與必要的許可，並確認已將角色新增至群組。如需詳細資訊，請參閱 [the section called “管理群組角色 \(主控台\)”](#) 或 [the section called “管理群組角色 \(CLI\)”](#)。

- 已將 [CloudWatch Metrics 連接器](#) 新增至 Greengrass 群組並設定。只有在您想要使用指標報告功能時，才需要此項目。
- 需要 [AWS IoT Greengrass Machine Learning SDK](#) 1.1.0 版才能與此連接器互動。

Parameters

FeedbackConfigurationMap

連接器可用來將範例上傳至 Amazon S3 的一組或多個意見回饋組態。意見回饋組態會定義目的地儲存貯體、內容類型和 [抽樣策略](#) 之類的屬性。叫用此連接器時，呼叫 Lambda 函數或連接器會指定目標意見回饋組態。

AWS IoT 主控台中的顯示名稱：意見回饋組態映射

必要：true

類型：格式正確的 JSON 字串，定義一組支援的意見回饋組態。如需範例，請參閱 [the section called “FeedbackConfigurationMap 範例”](#)。

意見回饋組態物件的 ID 具有下列要求。

ID：

- 在組態物件之間必須是唯一的。
- 開頭必須為字母或數字。可以包含大小寫字母、數字與連字號。
- 長度必須為 2 到 63 個字元。

必要：true

類型：string

有效模式：`^[a-zA-Z0-9][a-zA-Z0-9-]{1,62}$`

範例：MyConfig0、config-a、12id

意見回饋組態物件的內文包含下列屬性。

s3-bucket-name

目的地 Amazon S3 儲存貯體的名稱。

 Note

群組角色必須允許在所有目的地儲存貯體上執行 `s3:PutObject` 動作。如需詳細資訊，請參閱[the section called “要求”](#)。

必要：true

類型：string

有效模式：`^[a-z0-9\.\-]{3,63}$`

content-type

要上傳之範例的內容類型。個別意見回饋組態的所有內容必須是相同類型。

必要：true

類型：string

範例：image/jpeg、application/json、audio/ogg

s3-prefix

用於所上傳範例的金鑰前綴。前綴類似於目錄名稱。它可讓您將類似的資料存放在儲存貯體的相同目錄下。如需詳細資訊，請參閱《Amazon Simple Storage Service 使用者指南》中的[物件金鑰和中繼資料](#)。

必要：false

類型：string

file-ext

用於所上傳範例的副檔名。必須是內容類型的有效副檔名。

必要：false

類型：string

範例：jpg、json、ogg

sampling-strategy

用來篩選要上傳之範例的[抽樣策略](#)。如果省略，連接器會嘗試上傳它收到的所有範例。

必要：false

類型：格式正確的 JSON 字串，其中包含下列屬性。

strategy-name

抽樣策略的名稱。

必要：true

類型：string

有效值：RANDOM_SAMPLING、LEAST_CONFIDENCE、MARGIN 或 ENTROPY

rate

[隨機](#)取樣策略的速率。

必要：true 如果 strategy-name 為 RANDOM_SAMPLING。

類型：number

有效值：0.0 - 1.0

threshold

[最低可信度](#)、[邊界](#)或[熵](#)取樣策略的閾值。

必要：true 如果 strategy-name 是 LEAST_CONFIDENCE、MARGIN 或 ENTROPY。

類型：number

有效值：

- 0.0 - 1.0，適用於 LEAST_CONFIDENCE 或 MARGIN 策略。
- 0.0 - no limit，適用於 ENTROPY 策略。

RequestLimit

連接器一次可處理的請求數量上限。

您可以使用此參數來限制連接器同時處理的要求數量，以限制記憶體消耗。超過此限制的請求會被忽略。

AWS IoT 主控台顯示名稱：請求限制

必要：false

類型：string

有效值：0 - 999

有效模式：`^$|^[0-9]{1,3}$`

建立範例連接器 (AWS CLI)

下列 CLI 命令 ConnectorDefinition 會使用包含 ML Feedback 連接器的初始版本來建立。

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-  
version '{  
  "Connectors": [  

```

```

    {
      "Id": "MyMLFeedbackConnector",
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/MLFeedback/
versions/1",
      "Parameters": {
        "FeedbackConfigurationMap": "{ \"RandomSamplingConfiguration\":
{ \"s3-bucket-name\": \"my-aws-bucket-random-sampling\", \"content-type\":
\"image/png\", \"file-ext\": \"png\", \"sampling-strategy\": { \"strategy-name
\": \"RANDOM_SAMPLING\", \"rate\": 0.5 } }, \"LeastConfidenceConfiguration\": {
\"s3-bucket-name\": \"my-aws-bucket-least-confidence-sampling\", \"content-type\":
\"image/png\", \"file-ext\": \"png\", \"sampling-strategy\": { \"strategy-name\":
\"LEAST_CONFIDENCE\", \"threshold\": 0.4 } } }",
        "RequestLimit": "10"
      }
    }
  ]
}'

```

FeedbackConfigurationMap 範例

以下是 FeedbackConfigurationMap 參數的擴展範例值。此範例包含數個使用不同取樣策略的意見回饋組態。

```

{
  "ConfigID1": {
    "s3-bucket-name": "my-aws-bucket-random-sampling",
    "content-type": "image/png",
    "file-ext": "png",
    "sampling-strategy": {
      "strategy-name": "RANDOM_SAMPLING",
      "rate": 0.5
    }
  },
  "ConfigID2": {
    "s3-bucket-name": "my-aws-bucket-margin-sampling",
    "content-type": "image/png",
    "file-ext": "png",
    "sampling-strategy": {
      "strategy-name": "MARGIN",
      "threshold": 0.4
    }
  },
  "ConfigID3": {

```

```
    "s3-bucket-name": "my-aws-bucket-least-confidence-sampling",
    "content-type": "image/png",
    "file-ext": "png",
    "sampling-strategy": {
      "strategy-name": "LEAST_CONFIDENCE",
      "threshold": 0.4
    }
  },
  "ConfigID4": {
    "s3-bucket-name": "my-aws-bucket-entropy-sampling",
    "content-type": "image/png",
    "file-ext": "png",
    "sampling-strategy": {
      "strategy-name": "ENTROPY",
      "threshold": 2
    }
  },
  "ConfigID5": {
    "s3-bucket-name": "my-aws-bucket-no-sampling",
    "s3-prefix": "DeviceA",
    "content-type": "application/json"
  }
}
```

抽樣策略

連接器支援四種抽樣策略，這些抽樣策略會決定是否要上傳已傳遞至連接器的範例。範例是模型用於預測的離散資料執行個體。您可以使用取樣策略來篩選出最有可能提高模型準確性的範例。

RANDOM_SAMPLING

根據提供的速率隨機上傳範例。如果隨機產生的值小於此速率，它會上傳範例。速率越高，上傳的範例越多。

Note

此策略會忽略任何提供的模型預測。

LEAST_CONFIDENCE

上傳其最大可信度機率低於所提供閾值的範例。

範例案例：

閾值：.6

模型預測：[.2, .2, .4, .2]

最大可信度機率：.4

結果：

使用範例，因為最大可信度機率 (.4) $\leq$ 閾值 (.6)。

MARGIN

如果前兩個可信度機率之間的邊界落在所提供閾值內，則上傳範例。邊界是前兩個機率之間的差異。

範例案例：

閾值：.02

模型預測：[.3, .35, .34, .01]

前兩個可信度機率：[.35, .34]

邊界：.01 (.35 - .34)

結果：

使用範例，因為邊界1 (.01) $\leq$ 閾值 (.02)。

ENTROPY

上傳其熵大於所提供閾值的範例。使用模型預測的標準化熵。

範例案例：

閾值：0.75

模型預測：[.5, .25, .25]

預測的熵：1.03972

結果：

使用範例：因為熵 (1.03972) $>$ 閾值 (0.75)。

輸入資料

使用者定義的 Lambda 函數使用 AWS IoT Greengrass Machine Learning SDK 中的 `feedback` 用戶端 `publish` 函數來叫用連接器。如需範例，請參閱 [the section called “用量範例”](#)。

Note

此連接器不接受 MQTT 訊息做為輸入資料。

`publish` 函數採用下列引數：

ConfigId

目標意見回饋組態的 ID。這必須符合 ML 意見回饋連接器的 [FeedbackConfigurationMap](#) 參數中定義的意見回饋組態 ID。

必要：true

類型：字串

ModelInput

傳遞至模型以進行推論的輸入資料。除非根據取樣策略將此輸入資料篩選掉，否則會使用目標組態來上傳此輸入資料。

必要：true

類型：位元組

ModelPrediction

來自模型的預測結果。結果類型可以是字典或清單。例如，ML Image Classification 連接器的預測結果是機率清單（例如 `[0.25, 0.60, 0.15]`）。此資料會發佈至 `/feedback/message/prediction` 主題。

必要：true

類型：字典或float值清單

中繼資料

客戶定義的應用程式特定中繼資料，其會附加至上傳的範例並發佈至 `/feedback/message/prediction` 主題。連接器也會將具有時間戳記值的 `publish-ts` 金鑰插入至中繼資料。

必要：false

類型：字典

範例：{"some-key": "some value"}

輸出資料

此連接器會將資料發佈至三個 MQTT 主題：

- 在 feedback/message/status 主題上來自連接器的狀態資訊。
- feedback/message/prediction 主題上的預測結果。
- cloudwatch/metric/put 主題上以 CloudWatch 為目標的指標。

您必須設定訂閱，以允許連接器在 MQTT 主題上進行通訊。如需詳細資訊，請參閱[the section called “輸入和輸出”](#)。

主題篩選條件：feedback/message/status

使用此主題來監控範例上傳和所捨棄範例的狀態。每次收到請求時，連接器就會發佈至此主題。

範例輸出：範例上傳成功

```
{
  "response": {
    "status": "success",
    "s3_response": {
      "ResponseMetadata": {
        "HostId": "I0WQ4fDEXAMPLEQM+ey7N9WgVhSnQ6JEXAMPLEZb7hSQDASK
+Jd1vEXAMPLEEa3Km",
        "RetryAttempts": 1,
        "HTTPStatusCode": 200,
        "RequestId": "79104EXAMPLEB723",
        "HTTPHeaders": {
          "content-length": "0",
          "x-amz-id-2":
"1bbqaDVF0hMlyU3gRvAX1ZIdg8P0WkGkCSSFsYFvSwLZk3j7QZhG5EXAMPLEdd4/pEXAMPLEUqU=",
          "server": "AmazonS3",
          "x-amz-expiration": "expiry-date=\"Wed, 17 Jul 2019 00:00:00 GMT\",
rule-id=\"0GZjYWY30TgtYWI2Zi00ZDl1LWE4YmQtNzMyYzEXAMPLEoUw\"",
          "x-amz-request-id": "79104EXAMPLEB723",
```

```

        "etag": "\"b9c4f172e64458a5fd674EXAMPLE5628\"",
        "date": "Thu, 11 Jul 2019 00:12:50 GMT",
        "x-amz-server-side-encryption": "AES256"
      },
      {
        "bucket": "greengrass-feedback-connector-data-us-west-2",
        "ETag": "\"b9c4f172e64458a5fd674EXAMPLE5628\"",
        "Expiration": "expiry-date=\"Wed, 17 Jul 2019 00:00:00 GMT\", rule-id=\\\"OGZjYWY3OTgtYWl2Zi00ZDl1LWE4YmQtNzMyYzEXAMPLEoUw\\\"",
        "key": "s3-key-prefix/UUID.file_ext",
        "ServerSideEncryption": "AES256"
      }
    ],
    "id": "5aaa913f-97a3-48ac-5907-18cd96b89eeb"
  }

```

連接器會將 bucket 和 key 欄位新增至來自 Amazon S3 的回應。如需 Amazon S3 回應的詳細資訊，請參閱《Amazon Simple Storage Service API 參考》中的 [PUT 物件](#)。

範例輸出：由於取樣策略而捨棄的範例

```

{
  "response": {
    "status": "sample_dropped_by_strategy"
  },
  "id": "4bf5aeb0-d1e4-4362-5bb4-87c05de78ba3"
}

```

範例輸出：範例上傳失敗

失敗狀態包含錯誤訊息做為 error_message 值，以及例外類別做為 error 值。

```

{
  "response": {
    "status": "fail",
    "error_message": "[RequestId: 4bf5aeb0-d1e4-4362-5bb4-87c05de78ba3] Failed to upload model input data due to exception. Model prediction will not be published. Exception type: NoSuchBucket, error: An error occurred (NoSuchBucket) when calling the PutObject operation: The specified bucket does not exist",
    "error": "NoSuchBucket"
  },
  "id": "4bf5aeb0-d1e4-4362-5bb4-87c05de78ba3"
}

```


範例輸出：由於請求限制而請求調節

```
{
  "response": {
    "status": "fail",
    "error_message": "Request limit has been reached (max request: 10 ). Dropping request.",
    "error": "Queue.Full"
  },
  "id": "4bf5aeb0-d1e4-4362-5bb4-87c05de78ba3"
}
```

主題篩選條件：feedback/message/prediction

使用此主題，根據上傳的範例資料接聽預測。這可讓您即時分析您的模型效能。只有當資料成功上傳至 Amazon S3 時，模型預測才會發佈至此主題。在此主題發佈的訊息採用 JSON 格式。它們包含所上傳資料物件的連結、模型的預測，以及請求中包含的中繼資料。

您也可以存放預測結果，並使用它們來報告和分析隨時間變化的趨勢。趨勢可提供寶貴的洞見。例如，隨著時間準確度降低趨勢可協助您決定是否需要重新訓練模型。

範例輸出

```
{
  "source-ref": "s3://greengrass-feedback-connector-data-us-west-2/s3-key-prefix/UUID.file_ext",
  "model-prediction": [
    0.5,
    0.2,
    0.2,
    0.1
  ],
  "config-id": "ConfigID2",
  "metadata": {
    "publish-ts": "2019-07-11 00:12:48.816752"
  }
}
```

Tip

您可以設定 [IoT Analytics 連接器](#) 來訂閱此主題，並將資訊傳送至 AWS IoT Analytics 以進行進一步或歷史分析。

主題篩選條件：cloudwatch/metric/put

這是用來將指標發佈至 CloudWatch 的輸出主題。此功能需要您安裝和設定 [CloudWatch 指標連接器](#)。

指標包括：

- 上傳的範例數量。
- 上傳的範例大小。
- 從上傳到 Amazon S3 的錯誤數目。
- 根據抽樣策略捨棄的範例數量。
- 已調節的請求數目。

範例輸出：資料範例的大小 (在實際上傳之前發佈)

```
{
  "request": {
    "namespace": "GreengrassFeedbackConnector",
    "metricData": {
      "value": 47592,
      "unit": "Bytes",
      "metricName": "SampleSize"
    }
  }
}
```

範例輸出：範例上傳成功

```
{
  "request": {
    "namespace": "GreengrassFeedbackConnector",
    "metricData": {
      "value": 1,
      "unit": "Count",
      "metricName": "SampleUploadSuccess"
    }
  }
}
```

範例輸出：範例上傳成功，並發佈預測結果

```
{
```

```
"request": {
  "namespace": "GreengrassFeedbackConnector",
  "metricData": {
    "value": 1,
    "unit": "Count",
    "metricName": "SampleAndPredictionPublished"
  }
}
```

範例輸出：範例上傳失敗

```
{
  "request": {
    "namespace": "GreengrassFeedbackConnector",
    "metricData": {
      "value": 1,
      "unit": "Count",
      "metricName": "SampleUploadFailure"
    }
  }
}
```

範例輸出：由於取樣策略而捨棄的範例

```
{
  "request": {
    "namespace": "GreengrassFeedbackConnector",
    "metricData": {
      "value": 1,
      "unit": "Count",
      "metricName": "SampleNotUsed"
    }
  }
}
```

範例輸出：由於請求限制而請求調節

```
{
  "request": {
    "namespace": "GreengrassFeedbackConnector",
    "metricData": {
      "value": 1,
```

```
        "unit": "Count",
        "metricName": "ErrorRequestThrottled"
    }
}
```

用量範例

下列範例是使用者定義的 Lambda 函數，使用[AWS IoT Greengrass Machine Learning SDK](#) 將資料傳送至 ML Feedback 連接器。

Note

您可以從 AWS IoT Greengrass [下載頁面](#) 下載 AWS IoT Greengrass Machine Learning SDK。

```
import json
import logging
import os
import sys
import greengrass_machine_learning_sdk as ml

client = ml.client('feedback')

try:
    feedback_config_id = os.environ["FEEDBACK_CONFIG_ID"]
    model_input_data_dir = os.environ["MODEL_INPUT_DIR"]
    model_prediction_str = os.environ["MODEL_PREDICTIONS"]
    model_prediction = json.loads(model_prediction_str)
except Exception as e:
    logging.info("Failed to open environment variables. Failed with exception:
{}".format(e))
    sys.exit(1)

try:
    with open(os.path.join(model_input_data_dir, os.listdir(model_input_data_dir)[0]),
'rb') as f:
        content = f.read()
except Exception as e:
    logging.info("Failed to open model input directory. Failed with exception:
{}".format(e))
```

```
sys.exit(1)

def invoke_feedback_connector():
    logging.info("Invoking feedback connector.")
    try:
        client.publish(
            ConfigId=feedback_config_id,
            ModelInput=content,
            ModelPrediction=model_prediction
        )
    except Exception as e:
        logging.info("Exception raised when invoking feedback connector:{}".format(e))
        sys.exit(1)

invoke_feedback_connector()

def function_handler(event, context):
    return
```

授權

ML 意見回饋連接器包含下列第三方軟體/授權：

- [適用於 Python \(Boto3\) 的 AWS SDK](#)/Apache License 2.0
- [botocore](#)/Apache License 2.0
- [dateutil](#)/PSF License
- [docutils](#)/BSD 授權、GNU 一般公有授權 (GPL)、Python 軟體基金會授權、公有網域
- [jmespath](#)/MIT License
- [s3transfer](#)/Apache License 2.0
- [urllib3](#)/MIT License

- [six](#)/MIT

此連接器根據 [Greengrass 核心軟體授權合約](#) 發行。

另請參閱

- [使用連接器整合服務和通訊協定](#)

- [the section called “連接器入門 \(主控台\)”](#)
- [the section called “連接器入門 \(CLI\)”](#)

ML 影像分類連接器

Warning

此連接器已進入延長的生命週期階段，而且 AWS IoT Greengrass 不會發行提供現有功能、增強功能、安全修補程式或錯誤修正的更新。如需詳細資訊，請參閱[AWS IoT Greengrass Version 1 維護政策](#)。

ML Image Classification [連接器](#)提供在 AWS IoT Greengrass 核心上執行的機器學習 (ML) 推論服務。此本機推論服務會使用由 SageMaker AI 影像分類演算法訓練的模型來執行影像分類。

使用者定義的 Lambda 函數使用 AWS IoT Greengrass Machine Learning SDK 向本機推論服務提交推論請求。此服務在本機執行推論，並傳回輸入映像屬於特定類別的機率。

AWS IoT Greengrass 提供下列版本的此連接器，可用於多個平台。

Version 2

連接器	描述和 ARN
ML 影像分類 Aarch64 JTX2	適用於 NVIDIA Jetson TX2 的映像分類推論服務。支援 GPU 加速。 ARN : <code>arn:aws:greengrass: <i>region</i>:/connectors/ImageClassificationAarch64JTX2/versions/2</code>
ML 影像分類 x86_64	適用於 x86_64 平台的映像分類推論服務。 ARN : <code>arn:aws:greengrass: <i>region</i>:/connectors/ImageClassificationx86-64/versions/2</code>

連接器	描述和 ARN
ML 影像分類 ARMv7	適用於 ARMv7 平台的影像分類推論服務。 ARN : arn:aws:greengrass : <i>region</i>::/connectors/ImageClassificationARMv7/versions/2

Version 1

連接器	描述和 ARN
ML 影像分類 Aarch64 JTX2	適用於 NVIDIA Jetson TX2 的映像分類推論服務。支援 GPU 加速。 ARN : arn:aws:greengrass : <i>region</i>::/connectors/ImageClassificationAarch64JTX2/versions/1
ML 影像分類 x86_64	適用於 x86_64 平台的映像分類推論服務。 ARN : arn:aws:greengrass : <i>region</i>::/connectors/ImageClassificationx86-64/versions/1
ML 影像分類 Armv7	適用於 Armv7 平台的映像分類推論服務。 ARN : arn:aws:greengrass : <i>region</i>::/connectors/ImageClassificationARMv7/versions/1

如需版本變更的詳細資訊，請參閱 [Changelog](#)。

要求

這些連接器有下列要求：

Version 2

- AWS IoT Greengrass 核心軟體 1.9.3 版或更新版本。
- 安裝在核心裝置上並新增至 PATH 環境變數的 [Python](#) 3.7 或 3.8 版。

Note

若要使用 Python 3.8，請執行下列命令來建立從預設 Python 3.7 安裝資料夾到已安裝 Python 3.8 二進位檔的符號連結。

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

這會設定您的裝置以符合 AWS IoT Greengrass 的 Python 需求。

- 對安裝在核心裝置上的 Apache MXNet 架構的相依性。如需詳細資訊，請參閱[the section called “安裝 MXNet 相依性”](#)。
- Greengrass 群組中參考 SageMaker AI 模型來源的 [ML 資源](#)。此模型必須由 SageMaker AI 影像分類演算法訓練。如需詳細資訊，請參閱《Amazon SageMaker AI 開發人員指南》中的[映像分類演算法](#)。
- [ML Feedback 連接器](#)已新增至 Greengrass 群組並設定。只有在您想要使用連接器上傳模型輸入資料，並將預測發佈至 MQTT 主題時，才需要此項目。
- 設定為允許對目標訓練任務 `sagemaker:DescribeTrainingJob` 執行動作的 [Greengrass 群組角色](#)，如下列範例 IAM 政策所示。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sagemaker:DescribeTrainingJob"
      ]
    }
  ]
}
```



```

        "Resource": "arn:aws:sagemaker:us-east-1:123456789012:training-
job/training-job-name"
    }
]
}

```

針對群組角色要求，您必須設定角色以授與必要的許可，並確認已將角色新增至群組。如需詳細資訊，請參閱 [the section called “管理群組角色 \(主控台\)”](#) 或 [the section called “管理群組角色 \(CLI\)”](#)。

您可以為資源授予細微或條件式存取 (例如，使用萬用字元 * 命名配置)。如果您未來變更目標訓練任務，請務必更新群組角色。

- 需要 [AWS IoT Greengrass Machine Learning SDK](#) 1.1.0 版才能與此連接器互動。

Version 1

- AWS IoT Greengrass 核心軟體 1.7 版或更新版本。
- 安裝在核心裝置上並新增至 PATH 環境變數的 [Python](#) 2.7 版。
- 對安裝在核心裝置上的 Apache MXNet 架構的相依性。如需詳細資訊，請參閱 [the section called “安裝 MXNet 相依性”](#)。
- Greengrass 群組中參考 SageMaker AI 模型來源的 [ML 資源](#)。此模型必須由 SageMaker AI 影像分類演算法訓練。如需詳細資訊，請參閱《Amazon SageMaker AI 開發人員指南》中的 [映像分類演算法](#)。
- 設定為允許對目標訓練任務執行 sagemaker:DescribeTrainingJob 動作的 [Greengrass 群組角色](#)，如下列範例 IAM 政策所示。

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sagemaker:DescribeTrainingJob"
      ],
      "Resource": "arn:aws:sagemaker:us-east-1:123456789012:training-
job/training-job-name"
    }
  ]
}

```

```
]
}
```

針對群組角色要求，您必須設定角色以授與必要的許可，並確認已將角色新增至群組。如需詳細資訊，請參閱 [the section called “管理群組角色 \(主控台\)”](#) 或 [the section called “管理群組角色 \(CLI\)”](#)。

您可以為資源授予細微或條件式存取 (例如，使用萬用字元 * 命名配置)。如果您未來變更目標訓練任務，請務必更新群組角色。

- 需要 [AWS IoT Greengrass Machine Learning SDK](#) 1.0.0 版或更新版本，才能與此連接器互動。

連接器參數

這些連接器提供下列參數。

Version 2

MLModelDestinationPath

Lambda 環境中 ML 資源的絕對本機路徑。這是指定給機器學習資源的目的地路徑。

Note

如果是在主控台建立機器學習資源，則這是本機路徑。

AWS IoT 主控台中的顯示名稱：模型目的地路徑

必要：true

類型：string

有效模式：.+

MLModelResourceId

參考來源模型的機器學習資源 ID。

在 AWS IoT 主控台中顯示名稱：SageMaker 任務 ARN 資源

必要：true

類型：string

有效模式：[a-zA-Z0-9:_-]+

MLModelSageMakerJobArn

代表 SageMaker AI 模型來源的 SageMaker AI 訓練任務 ARN。模型必須由 SageMaker AI 影像分類演算法訓練。

在 AWS IoT 主控台中顯示名稱：SageMaker 任務 ARN

必要：true

類型：string

有效模式：^arn:aws:sagemaker:[a-zA-Z0-9-]+:[0-9]+:training-job/[a-zA-Z0-9][a-zA-Z0-9-]+\$

LocalInferenceServiceName

本機推論服務的名稱。使用者定義的 Lambda 函數透過將名稱傳遞至 AWS IoT Greengrass Machine Learning SDK 的 `invoke_inference_service` 函數來叫用服務。如需範例，請參閱 [the section called “用量範例”](#)。

AWS IoT 主控台顯示名稱：本機推論服務名稱

必要：true

類型：string

有效模式：[a-zA-Z0-9][a-zA-Z0-9-]{1,62}

LocalInferenceServiceTimeoutSeconds

推論請求終止之前的時間 (以秒為單位)。最小值為 1。

AWS IoT 主控台顯示名稱：逾時 (秒)

必要：true

類型：string

有效模式：[1-9][0-9]*

LocalInferenceServiceMemoryLimitKB

服務可存取的記憶體數量 (KB)。最小值為 1。

AWS IoT 主控台中的顯示名稱：記憶體限制 (KB)

必要：true

類型：string

有效模式：[1-9][0-9]*

GPUAcceleration

CPU 或 GPU (加速) 運算環境。此屬性僅適用於 ML Image Classification Aarch64 JTX2 連接器。

AWS IoT 主控台中的顯示名稱：GPU 加速

必要：true

類型：string

有效值：CPU 或 GPU

MLFeedbackConnectorConfigId

用來上傳模型輸入資料的意見回饋組態 ID。這必須符合針對 [ML 意見回饋連接器](#) 定義的意見回饋組態 ID。

只有在您想要使用 ML 意見回饋連接器上傳模型輸入資料，並將預測發佈至 MQTT 主題時，才需要此參數。

在 AWS IoT 主控台中顯示名稱：ML Feedback 連接器組態 ID

必要：false

類型：string

有效模式：^[a-zA-Z0-9]{1,62}\$

Version 1

MLModelDestinationPath

Lambda 環境中 ML 資源的絕對本機路徑。這是指定給機器學習資源的目的地路徑。

 Note

如果是在主控台建立機器學習資源，則這是本機路徑。

AWS IoT 主控台中的顯示名稱：模型目的地路徑

必要：true

類型：string

有效模式：.+

MLModelResourceId

參考來源模型的機器學習資源 ID。

在 AWS IoT 主控台中顯示名稱：SageMaker 任務 ARN 資源

必要：true

類型：string

有效模式：[a-zA-Z0-9:_-]+

MLModelSageMakerJobArn

代表 SageMaker AI 模型來源的 SageMaker AI 訓練任務 ARN。模型必須由 SageMaker AI 影像分類演算法訓練。

在 AWS IoT 主控台中顯示名稱：SageMaker 任務 ARN

必要：true

類型：string

有效模式：`^arn:aws:sagemaker:[a-zA-Z0-9-]+:[0-9]+:training-job/[a-zA-Z0-9][a-zA-Z0-9-]+$`

LocalInferenceServiceName

本機推論服務的名稱。使用者定義的 Lambda 函數透過將名稱傳遞至 AWS IoT Greengrass Machine Learning SDK 的 `invoke_inference_service` 函數來叫用服務。如需範例，請參閱 [the section called “用量範例”](#)。

在 AWS IoT 主控台中顯示名稱：本機推論服務名稱

必要：true

類型：string

有效模式：[a-zA-Z0-9][a-zA-Z0-9-]{1,62}

LocalInferenceServiceTimeoutSeconds

推論請求終止之前的時間 (以秒為單位)。最小值為 1。

AWS IoT 主控台顯示名稱：逾時 (秒)

必要：true

類型：string

有效模式：[1-9][0-9]*

LocalInferenceServiceMemoryLimitKB

服務可存取的記憶體數量 (KB)。最小值為 1。

AWS IoT 主控台顯示名稱：記憶體限制 (KB)

必要：true

類型：string

有效模式：[1-9][0-9]*

GPUAcceleration

CPU 或 GPU (加速) 運算環境。此屬性僅適用於 ML Image Classification Aarch64 JTX2 連接器。

AWS IoT 主控台顯示名稱：GPU 加速

必要：true

類型：string

有效值：CPU 或 GPU

建立範例連接器 (AWS CLI)

下列 CLI 命令 `ConnectorDefinition` 會使用包含 ML Image Classification 連接器的初始版本來建立。

範例：CPU 執行個體

此範例會建立 ML Image Classification Armv7l 連接器的執行個體。

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-  
version '{  
  "Connectors": [  
    {  
      "Id": "MyImageClassificationConnector",  
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/  
ImageClassificationARMv7/versions/2",  
      "Parameters": {  
        "MLModelDestinationPath": "/path-to-model",  
        "MLModelResourceId": "my-ml-resource",  
        "MLModelSageMakerJobArn": "arn:aws:sagemaker:us-  
west-2:123456789012:training-job:MyImageClassifier",  
        "LocalInferenceServiceName": "imageClassification",  
        "LocalInferenceServiceTimeoutSeconds": "10",  
        "LocalInferenceServiceMemoryLimitKB": "500000",  
        "MLFeedbackConnectorConfigId": "MyConfig0"  
      }  
    }  
  ]  
}'
```

範例：GPU 執行個體

此範例會建立 ML Image Classification Aarch64 JTX2 連接器的執行個體，該連接器支援 NVIDIA Jetson TX2 主機板上的 GPU 加速。

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-  
version '{  
  "Connectors": [  
    {  
      "Id": "MyImageClassificationConnector",  
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/  
ImageClassificationAarch64JTX2/versions/2",  
      "Parameters": {  

```

```

        "MLModelDestinationPath": "/path-to-model",
        "MLModelResourceId": "my-ml-resource",
        "MLModelSageMakerJobArn": "arn:aws:sagemaker:us-
west-2:123456789012:training-job:MyImageClassifier",
        "LocalInferenceServiceName": "imageClassification",
        "LocalInferenceServiceTimeoutSeconds": "10",
        "LocalInferenceServiceMemoryLimitKB": "500000",
        "GPUAcceleration": "GPU",
        "MLFeedbackConnectorConfigId": "MyConfig0"
    }
}
]
}'

```

Note

這些連接器中的 Lambda 函數具有[長期生命週期](#)。

在 AWS IoT Greengrass 主控台中，您可以從群組的連接器頁面新增連接器。如需詳細資訊，請參閱[the section called “連接器入門 \(主控台\)”](#)。

輸入資料

這些連接器接受影像檔做為輸入。輸入影像檔必須採用 jpeg 或 png 格式。如需詳細資訊，請參閱[the section called “用量範例”](#)。

這些連接器不接受 MQTT 訊息做為輸入資料。

輸出資料

這些連接器會傳回輸入影像中所識別物件的格式化預測：

```
[0.3,0.1,0.04,...]
```

預測包含一個清單，列出模型訓練期間與訓練資料集所使用之類別對應的值。每個值代表影像落在對應類別下的機率。具有最高機率的類別是主導預測。

這些連接器不會將 MQTT 訊息發佈為輸出資料。

用量範例

下列範例 Lambda 函數使用 [AWS IoT Greengrass Machine Learning SDK](#) 與 ML Image Classification 連接器互動。

Note

您可以從[AWS IoT Greengrass Machine Learning開發套件下載頁面下載開發套件](#)。

範例初始化軟體開發套件用戶端，並同步呼叫軟體開發套件的 `invoke_inference_service` 函數，以叫用本機推論服務。它會傳入演算法類型、服務名稱、映像類型和映像內容。然後，範例會剖析服務回應以獲得機率結果 (預測)。

Python 3.7

```
import logging
from threading import Timer

import numpy as np

import greengrass_machine_learning_sdk as ml

# We assume the inference input image is provided as a local file
# to this inference client Lambda function.
with open('/test_img/test.jpg', 'rb') as f:
    content = bytearray(f.read())

client = ml.client('inference')

def infer():
    logging.info('invoking Greengrass ML Inference service')

    try:
        resp = client.invoke_inference_service(
            AlgoType='image-classification',
            ServiceName='imageClassification',
            ContentType='image/jpeg',
            Body=content
        )
    except ml.GreengrassInferenceException as e:
        logging.info('inference exception {}'.format(e.__class__.__name__, e))
```

```

        return
    except ml.GreengrassDependencyException as e:
        logging.info('dependency exception {}'.format(e.__class__.__name__,
e))
        return

    logging.info('resp: {}'.format(resp))
    predictions = resp['Body'].read().decode("utf-8")
    logging.info('predictions: {}'.format(predictions))

    # The connector output is in the format: [0.3,0.1,0.04,...]
    # Remove the '[' and ']' at the beginning and end.
    predictions = predictions[1:-1]
    count = len(predictions.split(','))
    predictions_arr = np.fromstring(predictions, count=count, sep=',')

    # Perform business logic that relies on the predictions_arr, which is an array
    # of probabilities.

    # Schedule the infer() function to run again in one second.
    Timer(1, infer).start()
    return

infer()

def function_handler(event, context):
    return

```

Python 2.7

```

import logging
from threading import Timer

import numpy

import greengrass_machine_learning_sdk as gg_ml

# The inference input image.
with open("/test_img/test.jpg", "rb") as f:
    content = f.read()

client = gg_ml.client("inference")

```

```

def infer():
    logging.info("Invoking Greengrass ML Inference service")

    try:
        resp = client.invoke_inference_service(
            AlgoType="image-classification",
            ServiceName="imageClassification",
            ContentType="image/jpeg",
            Body=content,
        )
    except gg_ml.GreengrassInferenceException as e:
        logging.info('Inference exception %s("%s")', e.__class__.__name__, e)
        return
    except gg_ml.GreengrassDependencyException as e:
        logging.info('Dependency exception %s("%s")', e.__class__.__name__, e)
        return

    logging.info("Response: %s", resp)
    predictions = resp["Body"].read()
    logging.info("Predictions: %s", predictions)

    # The connector output is in the format: [0.3,0.1,0.04,...]
    # Remove the '[' and ']' at the beginning and end.
    predictions = predictions[1:-1]
    predictions_arr = numpy.fromstring(predictions, sep=",")
    logging.info("Split into %s predictions.", len(predictions_arr))

    # Perform business logic that relies on predictions_arr, which is an array
    # of probabilities.

    # Schedule the infer() function to run again in one second.
    Timer(1, infer).start()

infer()

# In this example, the required AWS Lambda handler is never called.
def function_handler(event, context):
    return

```

AWS IoT Greengrass Machine Learning SDK 中的 `invoke_inference_service` 函數接受下列引數。

引數	Description
AlgoType	用於推論之演算法類型的名稱。目前僅支援 <code>image-classification</code> 。 必要：true 類型：string 有效值：image-classification
ServiceName	本機推論服務的名稱。使用您設定連接器時為 <code>LocalInferenceServiceName</code> 參數指定的名稱。 必要：true 類型：string
ContentType	輸入映像的 mime 類型。 必要：true 類型：string 有效值：image/jpeg, image/png
Body	輸入映像檔的內容。 必要：true 類型：binary

在 AWS IoT Greengrass 核心上安裝 MXNet 相依性

若要使用 ML Image Classification 連接器，您必須在核心裝置上安裝 Apache MXNet 架構的相依性。連接器使用架構來支援機器學習模型。

Note

這些連接器隨附預先編譯的 MXNet 程式庫，因此您不需要在核心裝置上安裝 MXNet 架構。

AWS IoT Greengrass 提供指令碼來安裝下列常見平台和裝置的相依性（或做為安裝它們的參考）。如果您使用的是不同平台或裝置，請參閱 [MXNet 文件](#) 做為組態的參考。

在安裝 MXNet 相依性之前，請確保所需的 [系統程式庫](#) (使用指定的最低版本) 存在於裝置上。

NVIDIA Jetson TX2

1. 安裝 CUDA Toolkit 9.0 和 cuDNN 7.0。您可以按照入門教學課程的 [the section called “設定其他裝置”](#) 中的指示。
2. 啟用世界各地的儲存庫，讓連接器可以安裝社群維護的開放式軟體。如需詳細資訊，請參閱 Ubuntu 文件中的 [儲存庫/Ubuntu](#)。
 - a. 開啟 `/etc/apt/sources.list` 檔案。
 - b. 務必註銷以下幾行。

```
deb http://ports.ubuntu.com/ubuntu-ports/ xenial universe
deb-src http://ports.ubuntu.com/ubuntu-ports/ xenial universe
deb http://ports.ubuntu.com/ubuntu-ports/ xenial-updates universe
deb-src http://ports.ubuntu.com/ubuntu-ports/ xenial-updates universe
```

3. 在核心裝置上將以下安裝指令碼的複本儲存成名為 `nvidiajtx2.sh` 的檔案。

Python 3.7

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."
echo 'Assuming that universe repos are enabled and checking dependencies...'
apt-get -y update
apt-get -y dist-upgrade
apt-get install -y liblapack3 libopenblas-dev liblapack-dev libatlas-base-dev
apt-get install -y python3.7 python3.7-dev

python3.7 -m pip install --upgrade pip
```

```
python3.7 -m pip install numpy==1.15.0
python3.7 -m pip install opencv-python || echo 'Error: Unable to install
OpenCV with pip on this platform. Try building the latest OpenCV from source
(https://github.com/opencv/opencv).'

echo 'Dependency installation/upgrade complete.'
```

Note

如果 [OpenCV](#) 未使用此指令碼成功安裝，您可以嘗試從來源建置。如需詳細資訊，請參閱 OpenCV 文件中的 [Installation in Linux](#)，或參閱適用於您平台的其他線上資源。

Python 2.7

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."
echo 'Assuming that universe repos are enabled and checking dependencies...'
apt-get -y update
apt-get -y dist-upgrade
apt-get install -y liblapack3 libopenblas-dev liblapack-dev libatlas-base-dev
python-dev

echo 'Install latest pip...'
wget https://bootstrap.pypa.io/get-pip.py
python get-pip.py
rm get-pip.py

pip install numpy==1.15.0 scipy

echo 'Dependency installation/upgrade complete.'
```

4. 從您儲存檔案的目錄中，執行下列命令：

```
sudo nvidiajtx2.sh
```

x86_64 (Ubuntu or Amazon Linux)

1. 在核心裝置上將以下安裝指令碼的複本儲存成名為 x86_64.sh 的檔案。

Python 3.7

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."

release=$(awk -F= '/^NAME/{print $2}' /etc/os-release)

if [ "$release" == "Ubuntu" ]; then
    # Ubuntu. Supports EC2 and DeepLens. DeepLens has all the dependencies
    # installed, so
    # this is mostly to prepare dependencies on Ubuntu EC2 instance.
    apt-get -y update
    apt-get -y dist-upgrade

    apt-get install -y libgfortran3 libsm6 libxext6 libxrender1
    apt-get install -y python3.7 python3.7-dev
elif [ "$release" == "Amazon Linux" ]; then
    # Amazon Linux. Expect python to be installed already
    yum -y update
    yum -y upgrade

    yum install -y compat-gcc-48-libgfortran libSM libXrender libXext
else
    echo "OS Release not supported: $release"
    exit 1
fi

python3.7 -m pip install --upgrade pip
python3.7 -m pip install numpy==1.15.0
python3.7 -m pip install opencv-python || echo 'Error: Unable to install
OpenCV with pip on this platform. Try building the latest OpenCV from source
(https://github.com/opencv/opencv).'

echo 'Dependency installation/upgrade complete.'
```

 Note

如果 [OpenCV](#) 未使用此指令碼成功安裝，您可以嘗試從來源建置。如需詳細資訊，請參閱 OpenCV 文件中的 [Installation in Linux](#)，或參閱適用於您平台的其他線上資源。

Python 2.7

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."

release=$(awk -F= '/^NAME/{print $2}' /etc/os-release)

if [ "$release" == "Ubuntu" ]; then
    # Ubuntu. Supports EC2 and DeepLens. DeepLens has all the dependencies
    # installed, so
    # this is mostly to prepare dependencies on Ubuntu EC2 instance.
    apt-get -y update
    apt-get -y dist-upgrade

    apt-get install -y libgfortran3 libsm6 libxext6 libxrender1 python-dev
    python-pip
elif [ "$release" == "Amazon Linux" ]; then
    # Amazon Linux. Expect python to be installed already
    yum -y update
    yum -y upgrade

    yum install -y compat-gcc-48-libgfortran libSM libXrender libXext python-
    pip
else
    echo "OS Release not supported: $release"
    exit 1
fi

pip install numpy==1.15.0 scipy opencv-python

echo 'Dependency installation/upgrade complete.'
```


2. 從您儲存檔案的目錄中，執行下列命令：

```
sudo x86_64.sh
```

Armv7 (Raspberry Pi)

1. 在核心裝置上將以下安裝指令碼的複本儲存成名為 `armv7l.sh` 的檔案。

Python 3.7

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."

apt-get update
apt-get -y upgrade

apt-get install -y liblapack3 libopenblas-dev liblapack-dev
apt-get install -y python3.7 python3.7-dev

python3.7 -m pip install --upgrade pip
python3.7 -m pip install numpy==1.15.0
python3.7 -m pip install opencv-python || echo 'Error: Unable to install
OpenCV with pip on this platform. Try building the latest OpenCV from source
(https://github.com/opencv/opencv).'

echo 'Dependency installation/upgrade complete.'
```

Note

如果 [OpenCV](#) 未使用此指令碼成功安裝，您可以嘗試從來源建置。如需詳細資訊，請參閱 OpenCV 文件中的 [Installation in Linux](#)，或參閱適用於您平台的其他線上資源。

Python 2.7

```
#!/bin/bash
set -e
```

```
echo "Installing dependencies on the system..."

apt-get update
apt-get -y upgrade

apt-get install -y liblapack3 libopenblas-dev liblapack-dev python-dev

# python-opencv depends on python-numpy. The latest version in the APT
# repository is python-numpy-1.8.2
# This script installs python-numpy first so that python-opencv can be
# installed, and then install the latest
# numpy-1.15.x with pip
apt-get install -y python-numpy python-opencv
dpkg --remove --force-depends python-numpy

echo 'Install latest pip...'
wget https://bootstrap.pypa.io/get-pip.py
python get-pip.py
rm get-pip.py

pip install --upgrade numpy==1.15.0 picamera scipy

echo 'Dependency installation/upgrade complete.'
```

2. 從您儲存檔案的目錄中，執行下列命令：

```
sudo bash armv7l.sh
```

Note

在 Raspberry Pi 上，使用 pip 來安裝機器學習相依性是一種記憶體密集型操作，可能會導致裝置記憶體不足、無法回應。若要解決這項問題，您可以暫時增加交換大小：在 `/etc/dphys-swapfile` 中，增加 `CONF_SWAPSIZE` 變數的值，然後執行下列命令來重新啟動 `dphys-swapfile`。

```
/etc/init.d/dphys-swapfile restart
```

記錄和故障診斷

根據您的群組設定，事件和錯誤日誌會寫入 CloudWatch Logs、本機檔案系統或兩者。來自這個連接器的日誌使用字首 LocalInferenceServiceName。如果連接器發生非預期的行為，請檢查連接器的日誌。日誌通常會包含有用的偵錯資訊，例如遺失機器學習程式庫相依性或連接器啟動失敗的原因。

如果 AWS IoT Greengrass 群組設定為寫入本機日誌，連接器會將日誌檔案寫入 *greengrass-root/ggc/var/log/user/*region*/aws/*。如需 Greengrass 記錄的詳細資訊，請參閱 [the section called “使用 AWS IoT Greengrass 日誌進行監控”](#)。

使用以下資訊來協助疑難排解 ML Image Classification 連接器的問題。

必要系統程式庫

下列索引標籤列出每個 ML Image Classification 連接器所需的系統程式庫。

ML Image Classification Aarch64 JTX2

程式庫	最低版本
ld-linux-aarch64.so.1	GLIBC_2.17
libc.so.6	GLIBC_2.17
libcublas.so.9.0	不適用
libcudart.so.9.0	不適用
libcudnn.so.7	不適用
libcufft.so.9.0	不適用
libcurand.so.9.0	不適用
libcusolver.so.9.0	不適用
libgcc_s.so.1	GCC_4.2.0
libgomp.so.1	GOMP_4.0, OMP_1.0
libm.so.6	GLIBC_2.23

程式庫	最低版本
libpthread.so.0	GLIBC_2.17
librt.so.1	GLIBC_2.17
libstdc++.so.6	GLIBCXX_3.4.21, CXXABI_1.3.8

ML Image Classification x86_64

程式庫	最低版本
ld-linux-x86-64.so.2	GCC_4.0.0
libc.so.6	GLIBC_2.4
libgfortran.so.3	GFORTRAN_1.0
libm.so.6	GLIBC_2.23
libpthread.so.0	GLIBC_2.2.5
librt.so.1	GLIBC_2.2.5
libstdc++.so.6	CXXABI_1.3.8, GLIBCXX_3.4.21

ML Image Classification Armv7

程式庫	最低版本
ld-linux-armhf.so.3	GLIBC_2.4
libc.so.6	GLIBC_2.7
libgcc_s.so.1	GCC_4.0.0
libgfortran.so.3	GFORTRAN_1.0
libm.so.6	GLIBC_2.4

程式庫	最低版本
libpthread.so.0	GLIBC_2.4
librt.so.1	GLIBC_2.4
libstdc++.so.6	CXXABI_1.3.8、CXXABI_ARM_1.3.3、GLIBCXX_3.4.20

問題

徵狀	解決方案
在 Raspberry Pi 上，下列錯誤訊息已記錄，並且您不是使用相機：Failed to initialize libdc1394	執行下列命令以停用驅動程式： <pre>sudo ln /dev/null /dev/raw1394</pre> 此操作為暫時性，符號連結在重新啟動就會消失。請參閱您作業系統版本的手冊，了解如何在重新啟動時自動建立連結。

授權

ML Image Classification 連接器包含下列第三方軟體/授權：

- [適用於 Python \(Boto3\) 的 AWS SDK](#)/Apache License 2.0
- [botocore](#)/Apache License 2.0
- [dateutil](#)/PSF License
- [docutils](#)/BSD 授權、GNU 一般公有授權 (GPL)、Python 軟體基金會授權、公有網域
- [jmespath](#)/MIT License
- [s3transfer](#)/Apache License 2.0
- [urllib3](#)/MIT License
- [Deep Neural Network Library \(DNNL\)](#)/Apache License 2.0

- [OpenMP* Runtime Library](#)/請參閱 [Intel OpenMP Runtime Library 授權](#)。
- [mxnet](#)/Apache License 2.0
- [six](#)/MIT

Intel OpenMP Runtime Library 授權。Intel® OpenMP* 執行期是雙重授權，具有商業 (COM) 授權做為 Intel® Parallel Studio XE Suite 產品的一部分，以及 BSD 開放原始碼 (OSS) 授權。

此連接器根據 [Greengrass 核心軟體授權合約](#) 發行。

變更記錄

下表說明每個版本連接器的變更。

版本	變更
2	新增 MLFeedbackConnectorConfigId 參數以支援使用 ML Feedback 連接器 上傳模型輸入資料、將預測發佈至 MQTT 主題，以及將指標發佈至 Amazon CloudWatch。
1	初始版本。

Greengrass 群組一次只能包含一個版本的連接器。若要取得有關升級連接器版本的資訊，請參閱[the section called “升級連接器版本”](#)。

另請參閱

- [使用連接器整合服務和通訊協定](#)
- [the section called “連接器入門 \(主控台\)”](#)
- [the section called “連接器入門 \(CLI\)”](#)
- [執行機器學習推論](#)
- 《Amazon SageMaker AI 開發人員指南》中的[映像分類演算法](#)

ML 物件偵測連接器

Warning

此連接器已進入延長生命週期階段，AWS IoT Greengrass 且不會發佈提供現有功能、增強功能、安全修補程式或錯誤修正的更新。如需詳細資訊，請參閱[AWS IoT Greengrass Version 1 維護政策](#)。

ML 物件偵測[連接器](#)提供在 AWS IoT Greengrass 核心上執行的機器學習 (ML) 推論服務。此本機推論服務會使用 SageMaker AI Neo 深度學習編譯器編譯的物件偵測模型來執行物件偵測。支援兩種類型的物件偵測模型：Single Shot Multibox Detector (SSD) 和 You Only Look Once (YOLO) v3。如需詳細資訊，請參閱[物件偵測模型需求](#)。

使用者定義的 Lambda 函數使用 AWS IoT Greengrass Machine Learning SDK 將推論請求提交至本機推論服務。此服務會在輸入影像上執行本機推論，並針對影像中偵測到的每個物件傳回預測清單。每個預測都包含一個物件類別、預測可信度分數，以及指定預測物件之週框方塊的像素座標。

AWS IoT Greengrass 為多個平台提供 ML 物件偵測連接器：

連接器	描述和 ARN
ML 物件偵測 Aarch64 JTX2	適用於 NVIDIA Jetson TX2 的物件偵測推論服務。支援 GPU 加速。 ARN : arn:aws:greengrass: <i>region</i>::/connectors/ObjectDetectionAarch64JTX2/versions/1
ML 物件偵測 x86_64	適用於 x86_64 平台的物件偵測推論服務。 ARN : arn:aws:greengrass: <i>region</i>::/connectors/ObjectDetectionx86-64/versions/1
ML 物件偵測 ARMv7	適用於 ARMv7 平台的物件偵測推論服務。

連接器	描述和 ARN
	ARN : arn:aws:greengrass: <i>region</i> ::/connectors/ObjectDetectionARMv7/versions/1

要求

這些連接器有下列要求：

- AWS IoT Greengrass 核心軟體 1.9.3 版或更新版本。
- 安裝在核心裝置上並新增至 PATH 環境變數的 [Python](#) 3.7 或 3.8 版。

Note

若要使用 Python 3.8，請執行下列命令，從預設 Python 3.7 安裝資料夾建立符號連結，以連接至已安裝的 Python 3.8 二進位檔。

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

這會設定您的裝置以符合 AWS IoT Greengrass 的 Python 需求。

- 核心裝置上安裝的 SageMaker AI Neo 深度學習執行期的相依性。如需詳細資訊，請參閱 [the section called “安裝 Neo 深度學習執行時間相依性”](#)。
- Greengrass 群組中的 [ML 資源](#)。ML 資源必須參考包含物件偵測模型的 Amazon S3 儲存貯體。如需詳細資訊，請參閱 [Amazon S3 模型來源](#)。

Note

模型必須是 Single Shot Multibox Detector 或 You Only Look Once v3 物件偵測模型類型。它必須使用 SageMaker AI Neo 深度學習編譯器進行編譯。如需詳細資訊，請參閱 [物件偵測模型需求](#)。

- [ML 意見回饋連接器](#) 已新增至 Greengrass 群組並設定。只有在您想要使用連接器上傳模型輸入資料，並將預測發佈至 MQTT 主題時，才需要此項目。
- 需要 [AWS IoT Greengrass Machine Learning SDK](#) 1.1.0 版才能與此連接器互動。

物件偵測模型需求

ML 物件偵測連接器支援單一鏡頭多盒偵測器 (SSD) 和僅監看一次 (YOLO) v3 物件偵測模型類型。您可以使用 [GluonCV](#) 提供的物件偵測元件，搭配您自己的資料集來訓練模型。或者，您可以使用來自 GluonCV Model Zoo 的預先訓練模型：

- [預先訓練的 SSD 模型](#)
- [預先訓練的 YOLO v3 模型](#)

您的物件偵測模型必須使用 512 x 512 輸入影像進行訓練。來自 GluonCV Model Zoo 的預先訓練模型已符合此需求。

訓練過的物件偵測模型必須使用 SageMaker AI Neo 深度學習編譯器進行編譯。編譯時，請確定目標硬體符合 Greengrass 核心裝置的硬體。如需詳細資訊，請參閱《Amazon [SageMaker AI 開發人員指南](#)》中的 [SageMaker AI Neo](#)。Amazon SageMaker

編譯的模型必須新增為 ML 資源 ([Amazon S3 模型來源](#)) 到與連接器相同的 Greengrass 群組。

連接器參數

這些連接器提供下列參數。

MLModelDestinationPath

Amazon S3 儲存貯體的絕對路徑，其中包含 Neo 相容 ML 模型。這是指定給 ML 模型資源的目的地路徑。

AWS IoT 主控台顯示名稱：模型目的地路徑

必要：true

類型：string

有效模式：.+

MLModelResourceId

參考來源模型的機器學習資源 ID。

在 AWS IoT 主控台中顯示名稱：Greengrass 群組 ML 資源

必要：true

類型：S3MachineLearningModelResource

有效模式：`^[a-zA-Z0-9:_-]+$`

LocalInferenceServiceName

本機推論服務的名稱。使用者定義的 Lambda 函數透過將名稱傳遞至 AWS IoT Greengrass Machine Learning SDK 的 `invoke_inference_service` 函數來叫用服務。如需範例，請參閱「[the section called “使用範例”](#)」。

在 AWS IoT 主控台中顯示名稱：本機推論服務名稱

必要：true

類型：string

有效模式：`^[a-zA-Z0-9][a-zA-Z0-9-]{1,62}$`

LocalInferenceServiceTimeoutSeconds

推論請求終止之前的時間 (以秒為單位)。最小值為 1。預設值為 10。

AWS IoT 主控台顯示名稱：逾時 (秒)

必要：true

類型：string

有效模式：`^[1-9][0-9]*$`

LocalInferenceServiceMemoryLimitKB

服務可存取的記憶體數量 (KB)。最小值為 1。

AWS IoT 主控台顯示名稱：記憶體限制

必要：true

類型：string

有效模式：`^[1-9][0-9]*$`

GPUAcceleration

CPU 或 GPU (加速) 運算環境。此屬性僅適用於 ML Image Classification Aarch64 JTX2 連接器。

AWS IoT 主控台中的顯示名稱：GPU 加速

必要：true

類型：string

有效值：CPU 或 GPU

MLFeedbackConnectorConfigId

用來上傳模型輸入資料的意見回饋組態 ID。這必須符合針對 [ML 意見回饋連接器](#) 定義的意見回饋組態 ID。

只有在您想要使用 ML 意見回饋連接器上傳模型輸入資料，並將預測發佈至 MQTT 主題時，才需要此參數。

AWS IoT 主控台中的顯示名稱：ML Feedback 連接器組態 ID

必要：false

類型：string

有效模式：`^[a-zA-Z0-9][a-zA-Z0-9-]{1,62}$`

建立範例連接器 (AWS CLI)

下列 CLI 命令會使用包含 ML 物件偵測連接器的 ConnectorDefinition 初始版本建立。此範例會建立 ML 物件偵測 ARMv7I 連接器的執行個體。

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-  
version '{  
  "Connectors": [  
    {  
      "Id": "MyObjectDetectionConnector",  
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/  
ObjectDetectionARMv7/versions/1",  
      "Parameters": {  
        "MLModelDestinationPath": "/path-to-model",  
        "MLModelResourceId": "my-ml-resource",  
        "LocalInferenceServiceName": "objectDetection",  
        "LocalInferenceServiceTimeoutSeconds": "10",  
        "LocalInferenceServiceMemoryLimitKB": "500000",  
        "MLFeedbackConnectorConfigId" : "object-detector-random-sampling"  
      }  
    ]  
  }  
}
```

```
}
]
}'
```

Note

這些連接器中的 Lambda 函數具有[長期生命週期](#)。

在 AWS IoT Greengrass 主控台中，您可以從群組的連接器頁面新增連接器。如需詳細資訊，請參閱[the section called “連接器入門 \(主控台\)”](#)。

輸入資料

這些連接器接受影像檔做為輸入。輸入影像檔必須採用 jpeg 或 png 格式。如需詳細資訊，請參閱[the section called “使用範例”](#)。

這些連接器不接受 MQTT 訊息做為輸入資料。

輸出資料

這些連接器會傳回格式化的清單，列出輸入影像中所識別物件的預測結果：

```
{
  "prediction": [
    [
      14,
      0.9384938478469849,
      0.37763649225234985,
      0.5110225081443787,
      0.6697432398796082,
      0.8544386029243469
    ],
    [
      14,
      0.8859519958496094,
      0,
      0.43536216020584106,
      0.3314110040664673,
      0.9538808465003967
    ],
    [
```

```

        12,
        0.04128098487854004,
        0.5976729989051819,
        0.5747185945510864,
        0.704264223575592,
        0.857937216758728
    ],
    ...
]
}

```

清單中的每個預測都包含在方括號中，並包含六個值：

- 第一個值代表所識別物件的預測物件類別。在 Neo 深度學習編譯器中訓練物件偵測機器學習模型時，會決定物件類別及其對應值。
- 第二個值是物件類別預測的可信度分數。這代表預測正確的機率。
- 最後四個值對應至像素維度，代表影像中所預測物件的週框方塊。

這些連接器不會將 MQTT 訊息發佈為輸出資料。

使用範例

下列範例 Lambda 函數使用 [AWS IoT Greengrass Machine Learning 開發套件](#) 與 ML 物件偵測連接器互動。

Note

您可以從 [AWS IoT Greengrass Machine Learning 開發套件下載頁面下載開發套件](#)。

範例初始化軟體開發套件用戶端，並同步呼叫軟體開發套件的 `invoke_inference_service` 函數，以叫用本機推論服務。它會傳入演算法類型、服務名稱、映像類型和映像內容。然後，範例會剖析服務回應以獲得機率結果 (預測)。

```

import logging
from threading import Timer

import numpy as np

import greengrass_machine_learning_sdk as ml

```

```
# We assume the inference input image is provided as a local file
# to this inference client Lambda function.
with open('/test_img/test.jpg', 'rb') as f:
    content = bytearray(f.read())

client = ml.client('inference')

def infer():
    logging.info('invoking Greengrass ML Inference service')

    try:
        resp = client.invoke_inference_service(
            AlgoType='object-detection',
            ServiceName='objectDetection',
            ContentType='image/jpeg',
            Body=content
        )
    except ml.GreengrassInferenceException as e:
        logging.info('inference exception {}'.format(e.__class__.__name__, e))
        return
    except ml.GreengrassDependencyException as e:
        logging.info('dependency exception {}'.format(e.__class__.__name__, e))
        return

    logging.info('resp: {}'.format(resp))
    predictions = resp['Body'].read().decode("utf-8")
    logging.info('predictions: {}'.format(predictions))
    predictions = eval(predictions)

    # Perform business logic that relies on the predictions.

    # Schedule the infer() function to run again in ten second.
    Timer(10, infer).start()
    return

infer()

def function_handler(event, context):
    return
```

AWS IoT Greengrass Machine Learning開發套件中的 `invoke_inference_service` 函數接受下列引數。

引數	描述
AlgoType	用於推論之演算法類型的名稱。目前僅支援 <code>object-detection</code>。 必要：true 類型：string 有效值：object-detection
ServiceName	本機推論服務的名稱。使用您設定連接器時為 <code>LocalInferenceServiceName</code> 參數指定的名稱。 必要：true 類型：string
ContentType	輸入映像的 mime 類型。 必要：true 類型：string 有效值：image/jpeg, image/png
Body	輸入映像檔的內容。 必要：true 類型：binary

在 AWS IoT Greengrass 核心上安裝 Neo 深度學習執行期相依性

ML 物件偵測連接器隨附於 SageMaker AI Neo 深度學習執行期 (DLR)。連接器使用執行時間來提供 ML 模型。若要使用這些連接器，您必須在核心裝置上安裝 DLR 的相依性。

在安裝 DLR 相依性之前，請確保所需的[系統程式庫](#) (使用指定的最低版本) 存在於裝置上。

NVIDIA Jetson TX2

1. 安裝 CUDA Toolkit 9.0 和 cuDNN 7.0。您可以按照入門教學課程的[the section called “設定其他裝置”](#)中的指示。
2. 啟用世界各地的儲存庫，讓連接器可以安裝社群維護的開放式軟體。如需詳細資訊，請參閱 Ubuntu 文件中的[儲存庫/Ubuntu](#)。
 - a. 開啟 `/etc/apt/sources.list` 檔案。
 - b. 務必註銷以下幾行。

```
deb http://ports.ubuntu.com/ubuntu-ports/ xenial universe
deb-src http://ports.ubuntu.com/ubuntu-ports/ xenial universe
deb http://ports.ubuntu.com/ubuntu-ports/ xenial-updates universe
deb-src http://ports.ubuntu.com/ubuntu-ports/ xenial-updates universe
```

3. 在核心裝置上將以下安裝指令碼的複本儲存成名為 `nvidiajtx2.sh` 的檔案。

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."
echo 'Assuming that universe repos are enabled and checking dependencies...'
apt-get -y update
apt-get -y dist-upgrade
apt-get install -y liblapack3 libopenblas-dev liblapack-dev libatlas-base-dev
apt-get install -y python3.7 python3.7-dev

python3.7 -m pip install --upgrade pip
python3.7 -m pip install numpy==1.15.0
python3.7 -m pip install opencv-python || echo 'Error: Unable to install OpenCV
with pip on this platform. Try building the latest OpenCV from source (https://
github.com/opencv/opencv).'

echo 'Dependency installation/upgrade complete.'
```

Note

如果 [OpenCV](#) 未使用此指令碼成功安裝，您可以嘗試從來源建置。如需詳細資訊，請參閱 OpenCV 文件中的 [Installation in Linux](#)，或參閱適用於您平台的其他線上資源。

4. 從您儲存檔案的目錄中，執行下列命令：


```
sudo nvidiajtx2.sh
```

x86_64 (Ubuntu or Amazon Linux)

1. 在核心裝置上將以下安裝指令碼的複本儲存成名為 x86_64.sh 的檔案。

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."

release=$(awk -F= '/^NAME/{print $2}' /etc/os-release)

if [ "$release" == '"Ubuntu"' ]; then
    # Ubuntu. Supports EC2 and DeepLens. DeepLens has all the dependencies
    # installed, so
    # this is mostly to prepare dependencies on Ubuntu EC2 instance.
    apt-get -y update
    apt-get -y dist-upgrade

    apt-get install -y libgfortran3 libsm6 libxext6 libxrender1
    apt-get install -y python3.7 python3.7-dev
elif [ "$release" == '"Amazon Linux"' ]; then
    # Amazon Linux. Expect python to be installed already
    yum -y update
    yum -y upgrade

    yum install -y compat-gcc-48-libgfortran libSM libXrender libXext
else
    echo "OS Release not supported: $release"
    exit 1
fi

python3.7 -m pip install --upgrade pip
python3.7 -m pip install numpy==1.15.0
python3.7 -m pip install opencv-python || echo 'Error: Unable to install OpenCV
with pip on this platform. Try building the latest OpenCV from source (https://
github.com/opencv/opencv).'

echo 'Dependency installation/upgrade complete.'
```

 Note

如果 [OpenCV](#) 未使用此指令碼成功安裝，您可以嘗試從來源建置。如需詳細資訊，請參閱 OpenCV 文件中的 [Installation in Linux](#)，或參閱適用於您平台的其他線上資源。

2. 從您儲存檔案的目錄中，執行下列命令：

```
sudo x86_64.sh
```

ARMv7 (Raspberry Pi)

1. 在核心裝置上將以下安裝指令碼的複本儲存成名為 `armv7l.sh` 的檔案。

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."

apt-get update
apt-get -y upgrade

apt-get install -y liblapack3 libopenblas-dev liblapack-dev
apt-get install -y python3.7 python3.7-dev

python3.7 -m pip install --upgrade pip
python3.7 -m pip install numpy==1.15.0
python3.7 -m pip install opencv-python || echo 'Error: Unable to install OpenCV
with pip on this platform. Try building the latest OpenCV from source (https://
github.com/opencv/opencv).'

echo 'Dependency installation/upgrade complete.'
```

 Note

如果 [OpenCV](#) 未使用此指令碼成功安裝，您可以嘗試從來源建置。如需詳細資訊，請參閱 OpenCV 文件中的 [Installation in Linux](#)，或參閱適用於您平台的其他線上資源。

2. 從您儲存檔案的目錄中，執行下列命令：

```
sudo bash armv7l.sh
```

Note

在 Raspberry Pi 上，使用 pip 來安裝機器學習相依性是一種記憶體密集型操作，可能會導致裝置記憶體不足、無法回應。若要解決這項問題，您可以暫時增加交換大小。在 `/etc/dphys-swapfile` 中，增加 `CONF_SWAPSIZE` 變數的值，然後執行下列命令來重新啟動 `dphys-swapfile`。

```
/etc/init.d/dphys-swapfile restart
```

記錄和故障診斷

根據您的群組設定，事件和錯誤日誌會寫入 CloudWatch Logs、本機檔案系統或兩者。來自這個連接器的日誌使用字首 `LocalInferenceServiceName`。如果連接器發生非預期的行為，請檢查連接器的日誌。日誌通常會包含有用的偵錯資訊，例如遺失機器學習程式庫相依性或連接器啟動失敗的原因。

如果 AWS IoT Greengrass 群組設定為寫入本機日誌，連接器會將日誌檔案寫入 `greengrass-root/ggc/var/log/user/region/aws/`。如需 Greengrass 記錄的詳細資訊，請參閱 [the section called “使用 AWS IoT Greengrass 日誌進行監控”](#)。

使用下列資訊，以協助針對 ML 物件偵測連接器的問題進行故障診斷。

必要系統程式庫

下列索引標籤列出每個 ML 物件偵測連接器所需的系統程式庫。

ML Object Detection Aarch64 JTX2

程式庫	最低版本
ld-linux-aarch64.so.1	GLIBC_2.17
libc.so.6	GLIBC_2.17
libcublas.so.9.0	不適用

程式庫	最低版本
libcudart.so.9.0	不適用
libcudnn.so.7	不適用
libcufft.so.9.0	不適用
libcurand.so.9.0	不適用
libcusolver.so.9.0	不適用
libgcc_s.so.1	GCC_4.2.0
libgomp.so.1	GOMP_4.0, OMP_1.0
libm.so.6	GLIBC_2.23
libnvinfer.so.4	不適用
libnvm_gpu.so	不適用
libnvm.so	不適用
libnvidia-fatbinaryloader.so.28.2.1	不適用
libnvos.so	不適用
libpthread.so.0	GLIBC_2.17
librt.so.1	GLIBC_2.17
libstdc++.so.6	GLIBCXX_3.4.21, CXXABI_1.3.8

ML Object Detection x86_64

程式庫	最低版本
ld-linux-x86-64.so.2	GCC_4.0.0

程式庫	最低版本
libc.so.6	GLIBC_2.4
libgfortran.so.3	GFORTRAN_1.0
libm.so.6	GLIBC_2.23
libpthread.so.0	GLIBC_2.2.5
librt.so.1	GLIBC_2.2.5
libstdc++.so.6	CXXABI_1.3.8, GLIBCXX_3.4.21

ML Object Detection ARMv7

程式庫	最低版本
ld-linux-armhf.so.3	GLIBC_2.4
libc.so.6	GLIBC_2.7
libgcc_s.so.1	GCC_4.0.0
libgfortran.so.3	GFORTRAN_1.0
libm.so.6	GLIBC_2.4
libpthread.so.0	GLIBC_2.4
librt.so.1	GLIBC_2.4
libstdc++.so.6	CXXABI_1.3.8、CXXABI_ARM_1.3.3、GLIBCXX_3.4.20

問題

徵狀	解決方案
在 Raspberry Pi 上，下列錯誤訊息已記錄，並且您不是使用相機：Failed to initialize libdc1394	執行下列命令以停用驅動程式： <pre>sudo ln /dev/null /dev/raw1394</pre> 此為是暫時性的操作。符號連結會在您重新啟動後消失。請參閱您作業系統版本的手冊，了解如何在重新啟動時自動建立連結。

授權

ML 物件偵測連接器包含下列第三方軟體/授權：

- [適用於 Python \(Boto3\) 的 AWS SDK](#)/Apache License 2.0
- [botocore](#)/Apache License 2.0
- [dateutil](#)/PSF License
- [docutils](#)/BSD 授權、GNU 一般公有授權 (GPL)、Python 軟體基金會授權、公有網域
- [jmespath](#)/MIT License
- [s3transfer](#)/Apache License 2.0
- [urllib3](#)/MIT License

- [深度學習執行時間](#)/Apache License 2.0
- [six](#)/MIT

此連接器根據 [Greengrass 核心軟體授權合約](#) 發行。

另請參閱

- [使用連接器整合服務和通訊協定](#)
- [the section called “連接器入門 \(主控台\)”](#)
- [the section called “連接器入門 \(CLI\)”](#)
- [執行機器學習推論](#)
- 《Amazon SageMaker AI 開發人員指南》中的 [物件偵測演算法](#)

Modbus-RTU 通訊協定轉接器連接器

Modbus-RTU 通訊協定轉接器[連接器](#)會從 AWS IoT Greengrass 群組中的 Modbus RTU 裝置輪詢資訊。

此連接器會從使用者定義的 Lambda 函數接收 Modbus RTU 請求的參數。它會傳送對應的請求，然後從目標裝置發佈回應做為 MQTT 訊息。

此連接器具有下列版本。

版本	ARN
3	arn:aws:greengrass: <i>region</i> ::/connectors/ModbusRTUProtocolAdapter/versions/3
2	arn:aws:greengrass: <i>region</i> ::/connectors/ModbusRTUProtocolAdapter/versions/2
1	arn:aws:greengrass: <i>region</i> ::/connectors/ModbusRTUProtocolAdapter/versions/1

如需版本變更的詳細資訊，請參閱 [Changelog](#)。

要求

此連接器有下列要求：

Version 3

- AWS IoT Greengrass 核心軟體 1.9.3 版或更新版本。
- 安裝在核心裝置上並新增至 PATH 環境變數的 [Python](#) 3.7 或 3.8 版。

Note

若要使用 Python 3.8，請執行下列命令，從預設 Python 3.7 安裝資料夾建立符號連結，以連接至已安裝的 Python 3.8 二進位檔。

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

這會設定您的裝置以符合 AWS IoT Greengrass 的 Python 需求。

- AWS IoT Greengrass 核心和 Modbus 裝置之間的實體連線。核心必須透過序列埠 (例如 USB 連接埠) 實際連接到 RTU Modbus 網路。
- Greengrass 群組中的[本機裝置資源](#)，指向實體 Modbus 序列埠。
- 使用者定義的 Lambda 函數，會將 Modbus RTU 請求參數傳送至此連接器。此請求參數必須符合預期模式，且包含 Modbus RTU 網路上目標裝置的 ID 和地址。如需詳細資訊，請參閱[the section called “輸入資料”](#)。

Versions 1 - 2

- AWS IoT Greengrass 核心軟體 1.7 版或更新版本。
- 核心裝置上安裝並新增至 PATH 環境變數的 [Python 2.7](#) 版。
- AWS IoT Greengrass 核心和 Modbus 裝置之間的實體連線。核心必須透過序列埠 (例如 USB 連接埠) 實際連接到 RTU Modbus 網路。
- Greengrass 群組中的[本機裝置資源](#)，指向實體 Modbus 序列埠。
- 使用者定義的 Lambda 函數，會將 Modbus RTU 請求參數傳送至此連接器。此請求參數必須符合預期模式，且包含 Modbus RTU 網路上目標裝置的 ID 和地址。如需詳細資訊，請參閱[the section called “輸入資料”](#)。

連接器參數

此連接器支援下列參數：

ModbusSerialPort-ResourceId

代表實體 Modbus 序列埠的本機裝置資源 ID。

Note

此連接器已授予資源的讀寫存取權。

AWS IoT 主控台中的顯示名稱：Modbus 序列連接埠資源

必要：true

類型：string

有效模式：.+

ModbusSerialPort

裝置上實體 Modbus 序列埠的絕對路徑。這是指定給 Modbus 本機裝置資源的來源路徑。

AWS IoT 主控台顯示名稱：Modbus 序列連接埠資源的來源路徑

必要：true

類型：string

有效模式：.+

建立範例連接器 (AWS CLI)

下列 CLI 命令 `ConnectorDefinition` 會使用包含 Modbus-RTU 通訊協定轉接器連接器的初始版本來建立。

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
  "Connectors": [
    {
      "Id": "MyModbusRTUProtocolAdapterConnector",
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/
ModbusRTUProtocolAdapter/versions/3",
      "Parameters": {
        "ModbusSerialPort-ResourceId": "MyLocalModbusSerialPort",
        "ModbusSerialPort": "/path-to-port"
      }
    }
  ]
}'
```

Note

此連接器中的 Lambda 函數具有[長期生命週期](#)。

在 AWS IoT Greengrass 主控台中，您可以從群組的連接器頁面新增連接器。如需詳細資訊，請參閱 [the section called “連接器入門 \(主控台\)”](#)。

Note

部署 Modbus-RTU 通訊協定轉接器連接器之後，您可以使用 AWS IoT Things Graph 來協調群組中裝置之間的互動。如需詳細資訊，請參閱 AWS IoT Things Graph 《使用者指南》中的 [Modbus](#)。

輸入資料

此連接器接受 MQTT 主題上使用使用者定義 Lambda 函數的 Modbus RTU 請求參數。輸入訊息必須是 JSON 格式。

訂閱中的主題篩選條件

```
modbus/adapters/request
```

訊息屬性

請求訊息隨著其所代表的 Modbus RTU 請求類型而不同。所有請求需要下列屬性：

- 在 request 物件中：
 - operation。要執行的操作名稱。例如，指定 "operation": "ReadCoilsRequest" 以讀取線圈。這個值必須是 Unicode 字串。如需支援的作業，請參閱 [the section called “Modbus RTU 請求和回應”](#)。
 - device。請求的目標裝置。此值必須介於 0 - 247 之間。
 - id 屬性。請求的 ID。這個值用於重複資料刪除，並在所有回應的 id 屬性中依現狀傳回，包括錯誤回應。這個值必須是 Unicode 字串。

Note

如果您的請求中包含地址欄位，則必須將該值指定為整數。例如："address": 1。

要包含在請求中的其他參數取決於操作。除了 CRC (另外處理)，所有請求參數都是必要。如需範例，請參閱 [the section called “範例請求和回應”](#)。

範例輸入：讀取線圈請求

```
{
  "request": {
    "operation": "ReadCoilsRequest",
    "device": 1,
    "address": 1,
    "count": 1
  },
  "id": "TestRequest"
}
```

輸出資料

此連接器將回應發佈到傳入 Modbus RTU 請求。

訂閱中的主題篩選條件

modbus/adapter/response

訊息屬性

回應訊息的格式根據對應的請求和回應狀態而不同。如需範例，請參閱 [the section called “範例請求和回應”](#)。

Note

寫入操作的回應只是請求的回響。雖然寫入回應沒有傳回有意義的資訊，建議檢查回應的狀態。

每個回應含有以下屬性：

- 在 response 物件中：
 - status。請求的狀態。狀態可以是下列其中一個值：
 - Success。請求有效、傳送至 Modbus RTU 網路，並傳回回應。
 - Exception。請求有效，已傳送至 Modbus RTU 網路，並傳回例外狀況回應。如需詳細資訊，請參閱[the section called “回應狀態：例外”](#)。
 - No Response。請求無效，連接器在透過 Modbus RTU 網路傳送請求之前發現錯誤。如需詳細資訊，請參閱[the section called “回應狀態：沒有回應”](#)。

- device。傳送請求的裝置。
- operation。傳送的請求類型。
- payload。傳回的回應內容。如果 status 是 No Response，此物件只包含 error 屬性和錯誤描述 (例如，"error": "[Input/Output] No Response received from the remote unit")。
- id 屬性。用於資料重複刪除的請求 ID。

範例輸出：成功

```
{
  "response" : {
    "status" : "success",
    "device": 1,
    "operation": "ReadCoilsRequest",
    "payload": {
      "function_code": 1,
      "bits": [1]
    }
  },
  "id" : "TestRequest"
}
```

範例輸出：失敗

```
{
  "response" : {
    "status" : "fail",
    "error_message": "Internal Error",
    "error": "Exception",
    "device": 1,
    "operation": "ReadCoilsRequest",
    "payload": {
      "function_code": 129,
      "exception_code": 2
    }
  },
  "id" : "TestRequest"
}
```

如需更多範例，請參閱[the section called “範例請求和回應”](#)。

Modbus RTU 請求和回應

此連接器接受 Modbus RTU 請求參數做為[輸入資料](#)，並發佈回應做為[輸出資料](#)。

以下是支援的常見操作。

請求中的作業名稱	回應中的函數代碼
ReadCoilsRequest	01
ReadDiscreteInputsRequest	02
ReadHoldingRegistersRequest	03
ReadInputRegistersRequest	04
WriteSingleCoilRequest	05
WriteSingleRegisterRequest	06
WriteMultipleCoilsRequest	15
WriteMultipleRegistersRequest	16
MaskWriteRegisterRequest	22
ReadWriteMultipleRegistersRequest	23

範例請求和回應

以下是受支援操作的範例請求和回應。

讀取線圈

請求範例：

```
{
  "request": {
    "operation": "ReadCoilsRequest",
    "device": 1,
    "address": 1,
    "count": 1
  }
}
```

```
    },  
    "id": "TestRequest"  
  }  
}
```

回應範例：

```
{  
  "response": {  
    "status": "success",  
    "device": 1,  
    "operation": "ReadCoilsRequest",  
    "payload": {  
      "function_code": 1,  
      "bits": [1]  
    }  
  },  
  "id" : "TestRequest"  
}
```

讀取離散輸入

請求範例：

```
{  
  "request": {  
    "operation": "ReadDiscreteInputsRequest",  
    "device": 1,  
    "address": 1,  
    "count": 1  
  },  
  "id": "TestRequest"  
}
```

回應範例：

```
{  
  "response": {  
    "status": "success",  
    "device": 1,  
    "operation": "ReadDiscreteInputsRequest",  
    "payload": {  
      "function_code": 2,  

```

```
        "bits": [1]
      }
    },
    "id" : "TestRequest"
  }
```

讀取保留暫存器

請求範例：

```
{
  "request": {
    "operation": "ReadHoldingRegistersRequest",
    "device": 1,
    "address": 1,
    "count": 1
  },
  "id": "TestRequest"
}
```

回應範例：

```
{
  "response": {
    "status": "success",
    "device": 1,
    "operation": "ReadHoldingRegistersRequest",
    "payload": {
      "function_code": 3,
      "registers": [20,30]
    }
  },
  "id" : "TestRequest"
}
```

讀取輸入暫存器

請求範例：

```
{
  "request": {
    "operation": "ReadInputRegistersRequest",
    "device": 1,
```

```
    "address": 1,  
    "value": 1  
  },  
  "id": "TestRequest"  
}
```

寫入單一線圈

請求範例：

```
{  
  "request": {  
    "operation": "WriteSingleCoilRequest",  
    "device": 1,  
    "address": 1,  
    "value": 1  
  },  
  "id": "TestRequest"  
}
```

回應範例：

```
{  
  "response": {  
    "status": "success",  
    "device": 1,  
    "operation": "WriteSingleCoilRequest",  
    "payload": {  
      "function_code": 5,  
      "address": 1,  
      "value": true  
    }  
  },  
  "id" : "TestRequest"
```

寫入單一暫存器

請求範例：

```
{  
  "request": {  
    "operation": "WriteSingleRegisterRequest",  
    "device": 1,
```



```
    "address": 1,  
    "value": 1  
  },  
  "id": "TestRequest"  
}
```

寫入多個線圈

請求範例：

```
{  
  "request": {  
    "operation": "WriteMultipleCoilsRequest",  
    "device": 1,  
    "address": 1,  
    "values": [1,0,0,1]  
  },  
  "id": "TestRequest"  
}
```

回應範例：

```
{  
  "response": {  
    "status": "success",  
    "device": 1,  
    "operation": "WriteMultipleCoilsRequest",  
    "payload": {  
      "function_code": 15,  
      "address": 1,  
      "count": 4  
    }  
  },  
  "id" : "TestRequest"  
}
```

寫入多個暫存器

請求範例：

```
{  
  "request": {  
    "operation": "WriteMultipleRegistersRequest",
```

```
    "device": 1,  
    "address": 1,  
    "values": [20,30,10]  
  },  
  "id": "TestRequest"  
}
```

回應範例：

```
{  
  "response": {  
    "status": "success",  
    "device": 1,  
    "operation": "WriteMultipleRegistersRequest",  
    "payload": {  
      "function_code": 23,  
      "address": 1,  
      "count": 3  
    }  
  },  
  "id" : "TestRequest"  
}
```

遮罩寫入暫存器

請求範例：

```
{  
  "request": {  
    "operation": "MaskWriteRegisterRequest",  
    "device": 1,  
    "address": 1,  
    "and_mask": 175,  
    "or_mask": 1  
  },  
  "id": "TestRequest"  
}
```

回應範例：

```
{  
  "response": {
```

```
    "status": "success",
    "device": 1,
    "operation": "MaskWriteRegisterRequest",
    "payload": {
      "function_code": 22,
      "and_mask": 0,
      "or_mask": 8
    },
    "id" : "TestRequest"
  }
```

讀寫多個暫存器

請求範例：

```
{
  "request": {
    "operation": "ReadWriteMultipleRegistersRequest",
    "device": 1,
    "read_address": 1,
    "read_count": 2,
    "write_address": 3,
    "write_registers": [20,30,40]
  },
  "id": "TestRequest"
}
```

回應範例：

```
{
  "response": {
    "status": "success",
    "device": 1,
    "operation": "ReadWriteMultipleRegistersRequest",
    "payload": {
      "function_code": 23,
      "registers": [10,20,10,20]
    }
  },
  "id" : "TestRequest"
}
```

 Note

這個回應中傳回的暫存器是要讀取的暫存器。

回應狀態：例外

當請求格式有效但請求未順利完成時會發生例外。在此情況下，回應包含下列資訊：

- status 設定為 Exception。
- function_code 等於請求的函數碼 + 128。
- exception_code 包含例外碼。如需詳細資訊，請參閱 Modbus 例外碼。

範例：

```
{
  "response" : {
    "status" : "fail",
    "error_message": "Internal Error",
    "error": "Exception",
    "device": 1,
    "operation": "ReadCoilsRequest",
    "payload": {
      "function_code": 129,
      "exception_code": 2
    }
  },
  "id" : "TestRequest"
}
```

回應狀態：沒有回應

此連接器會在 Modbus 請求上執行驗證檢查。例如，它檢查是否格式無效和遺漏欄位。如果驗證失敗，連接器不會傳送請求。而是傳回包含以下資訊的回應：

- status 設定為 No Response。
- error 包含錯誤的原因。
- error_message 包含錯誤訊息。

範例：

```
{
  "response" : {
    "status" : "fail",
    "error_message": "Invalid address field. Expected <type 'int'>, got <type 'str'>",
    "error": "No Response",
    "device": 1,
    "operation": "ReadCoilsRequest",
    "payload": {
      "error": "Invalid address field. Expected <type 'int'>, got <type 'str'>"
    }
  },
  "id" : "TestRequest"
}
```

如果請求的目標是不存在的裝置，或 Modbus RTU 網路沒有作用，您可能會收到 `ModbusIOException` (使用「沒有回應」格式)。

```
{
  "response" : {
    "status" : "fail",
    "error_message": "[Input/Output] No Response received from the remote unit",
    "error": "No Response",
    "device": 1,
    "operation": "ReadCoilsRequest",
    "payload": {
      "error": "[Input/Output] No Response received from the remote unit"
    }
  },
  "id" : "TestRequest"
}
```

使用範例

使用下列高階步驟來設定範例 Python 3.7 Lambda 函數，您可以用來嘗試連接器。

Note

- 如果您使用其他 Python 執行期，則可以建立從 Python3.x 到 Python 3.7 的符號連結。

- [連接器入門 \(主控台\)](#) 和 [連接器入門 \(CLI\)](#) 主題包含詳細步驟，說明如何設定和部署範例 Twilio 通知連接器。

1. 確定您符合連接器的[要求](#)。
2. 建立並發佈 Lambda 函數，將輸入資料傳送至連接器。

將範例程式碼儲存為 PY 檔案。下載並解壓縮[AWS IoT Greengrass 適用於 Python 的核心 SDK](#)。然後，建立在根層級包含 PY 檔案和 greengrasssdk 資料夾的 zip 套件。此 zip 套件是您上傳至的部署套件 AWS Lambda。

建立 Python 3.7 Lambda 函數後，發佈函數版本並建立別名。

3. 設定 Greengrass 群組。
 - a. 依別名新增 Lambda 函數（建議）。將 Lambda 生命週期設定為長期（或在 CLI "Pinned": true 中）。
 - b. 新增必要的本機裝置資源，並授予 Lambda 函數的讀取/寫入存取權。
 - c. 新增連接器並設定其[參數](#)。
 - d. 新增訂閱，允許連接器在支援主題篩選條件上接收[輸入資料](#)並傳送[輸出資料](#)。
 - 將 Lambda 函數設定為來源，將連接器設定為目標，並使用支援的輸入主題篩選條件。
 - 將連接器設為來源、將 AWS IoT Core 設為目標，並使用支援的輸出主題篩選條件。您可以使用此訂閱在 AWS IoT 主控台中檢視狀態訊息。
4. 部署群組。
5. 在 AWS IoT 主控台的測試頁面上，訂閱輸出資料主題，以檢視連接器的狀態訊息。Lambda 函數範例為長期函數，並在部署群組後立即開始傳送訊息。

完成測試後，您可以將 Lambda 生命週期設定為隨需（或在 CLI "Pinned": false 中）並部署群組。這會讓函數停止傳送訊息。

範例

下列範例 Lambda 函數會將輸入訊息傳送至連接器。

```
import greengrasssdk
import json
```

```
TOPIC_REQUEST = 'modbus/adapter/request'

# Creating a greengrass core sdk client
iot_client = greengrasssdk.client('iot-data')

def create_read_coils_request():
    request = {
        "request": {
            "operation": "ReadCoilsRequest",
            "device": 1,
            "address": 1,
            "count": 1
        },
        "id": "TestRequest"
    }
    return request

def publish_basic_request():
    iot_client.publish(payload=json.dumps(create_read_coils_request()),
        topic=TOPIC_REQUEST)

publish_basic_request()

def lambda_handler(event, context):
    return
```

授權

Modbus-RTU 通訊協定轉接器連接器包含下列第三方軟體/授權：

- [pymodbus](#)/BSD
- [pyserial](#)/BSD

此連接器根據 [Greengrass 核心軟體授權合約](#) 發行。

變更記錄

下表說明連接器每個版本中的變更。

版本	變更
3	將 Lambda 執行時間升級至 Python 3.7，這會變更執行時間需求。
2	更新連接器 ARN 以取得 AWS 區域支援。 改善錯誤記錄。
1	初始版本。

Greengrass 群組一次只能包含一個版本的連接器。若要取得有關升級連接器版本的資訊，請參閱[the section called “升級連接器版本”](#)。

另請參閱

- [使用連接器整合服務和通訊協定](#)
- [the section called “連接器入門 \(主控台\)”](#)
- [the section called “連接器入門 \(CLI\)”](#)

Modbus-TCP 通訊協定轉接器連接器

Modbus-TCP 通訊協定轉接器[連接器](#)會透過 ModbusTCP 通訊協定從本機裝置收集資料，並將其發佈至選取的StreamManager串流。

您也可以將此連接器與 IoT SiteWise 連接器和 IoT SiteWise 閘道搭配使用。您的閘道必須提供連接器的組態。如需詳細資訊，請參閱 IoT SiteWise 使用者指南中的[設定 Modbus TCP 來源](#)。

Note

此連接器以[無容器](#)隔離模式執行，因此您可以將其部署到在 Docker 容器中執行的 AWS IoT Greengrass 群組。

此連接器具有下列版本。

版本	ARN
3	arn:aws:greengrass: <i>region</i> ::/connectors/ModbusTCPConnector/versions/3
2	arn:aws:greengrass: <i>region</i> ::/connectors/ModbusTCPConnector/versions/2
1	arn:aws:greengrass: <i>region</i> ::/connectors/ModbusTCPConnector/versions/1

如需版本變更的詳細資訊，請參閱 [Changelog](#)。

要求

此連接器有下列要求：

Version 1 - 3

- AWS IoT Greengrass 核心軟體 1.10.2 版或更新版本。
- AWS IoT Greengrass 群組上已啟用串流管理員。
- 安裝在核心裝置上並新增至PATH環境變數的 Java 8。

Note

此連接器僅適用於下列區域：

- ap-southeast-1
- ap-southeast-2
- eu-central-1
- eu-west-1
- us-east-1
- us-west-2

- cn-north-1

連接器參數

此連接器支援下列參數：

LocalStoragePath

IoT SiteWise 連接器可寫入持久性資料的 AWS IoT Greengrass 主機目錄。預設目錄為 `/var/sitewise`。

AWS IoT 主控台顯示名稱：本機儲存路徑

必要：false

類型：string

有效模式：`^\s*$|\/`

MaximumBufferSize

IoT SiteWise 磁碟用量的大小上限，以 GB 為單位。預設大小為 10GB。

AWS IoT 主控台顯示名稱：磁碟緩衝區大小上限

必要：false

類型：string

有效模式：`^\s*$|[0-9]+`

CapabilityConfiguration

連接器從收集資料並連線的一組 Modbus TCP 收集器組態。

AWS IoT 主控台顯示名稱：CapabilityConfiguration

必要：false

類型：格式正確的 JSON 字串，定義一組支援的意見回饋組態。

以下是的範例CapabilityConfiguration：

```
{
```

```

"sources": [
  {
    "type": "ModBusTCPSource",
    "name": "SourceName1",
    "measurementDataStreamPrefix": "SourceName1_Prefix",
    "destination": {
      "type": "StreamManager",
      "streamName": "SiteWise_Stream_1",
      "streamBufferSize": 8
    },
    "endpoint": {
      "ipAddress": "127.0.0.1",
      "port": 8081,
      "unitId": 1
    },
    "propertyGroups": [
      {
        "name": "GroupName",
        "tagPathDefinitions": [
          {
            "type": "ModBusTCPAddress",
            "tag": "TT-001",
            "address": "30001",
            "size": 2,
            "srcDataType": "float",
            "transformation": "byteWordSwap",
            "dstDataType": "double"
          }
        ],
        "scanMode": {
          "type": "POLL",
          "rate": 100
        }
      }
    ]
  }
]
}

```

建立範例連接器 (AWS CLI)

下列 CLI 命令 `ConnectorDefinition` 會使用包含 Modbus-TCP 通訊協定轉接器連接器的初始版本來建立。

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '
{
  "Connectors": [
    {
      "Id": "MyModbusTCPConnector",
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/ModbusTCP/
versions/3",
      "Parameters": {
        "capability_configuration": "{\"version\":1,\"namespace\":
\"iotsitewise:modbuscollector:1\", \"configuration\": {\"sources\": [{\"type
\": \"ModBusTCPSource\", \"name\": \"SourceName1\", \"measurementDataStreamPrefix
\": \"\", \"endpoint\": {\"ipAddress\": \"127.0.0.1\", \"port\": 8081, \"unitId\": 1},
\"propertyGroups\": [{\"name\": \"PropertyGroupName\", \"tagPathDefinitions\": [{\"type
\": \"ModBusTCPAddress\", \"tag\": \"TT-001\", \"address\": \"30001\", \"size\": 2,
\"srcDataType\": \"hexdump\", \"transformation\": \"noSwap\", \"dstDataType\": \"string
\"}], \"scanMode\": {\"rate\": 200, \"type\": \"POLL\"}], \"destination\": {\"type\":
\"StreamManager\", \"streamName\": \"SiteWise_Stream\", \"streamBufferSize\": 10},
\"minimumInterRequestDuration\": 200}]}\""}
      }
    ]
  }
}
```

Note

此連接器中的 Lambda 函數具有[長期生命週期](#)。

輸入資料

此連接器不接受 MQTT 訊息做為輸入資料。

輸出資料

此連接器會將資料發佈至 `StreamManager`。您必須設定目的地訊息串流。輸出訊息的結構如下：

```
{
  "alias": "string",
  "messages": [
    {
      "name": "string",
      "value": boolean|double|integer|string,
      "timestamp": number,
      "quality": "string"
    }
  ]
}
```

授權

Modbus-TCP 通訊協定轉接器連接器包含下列第三方軟體/授權：

- [數位 Petri Modbus](#)

此連接器根據 [Greengrass 核心軟體授權合約](#) 發行。

變更記錄

下表說明連接器每個版本中的變更。

版本	改變	日期
3 (建議)	此版本包含錯誤修正。	2021 年 12 月 22 日
2	新增對 ASCII、UTF8 和 ISO8859 編碼來源字串的支援。 。	2021 年 5 月 24 日
1	初始版本。	2020 年 12 月 15 日

Greengrass 群組一次只能包含一個版本的連接器。若要取得有關升級連接器版本的資訊，請參閱[the section called “升級連接器版本”](#)。

另請參閱

- [使用連接器整合服務和通訊協定](#)

- [the section called “連接器入門 \(主控台\)”](#)
- [the section called “連接器入門 \(CLI\)”](#)

Raspberry Pi GPIO 連接器

Warning

此連接器已進入延長的生命週期階段，AWS IoT Greengrass 且不會發佈提供現有功能、增強功能、安全修補程式或錯誤修正的更新。如需詳細資訊，請參閱[AWS IoT Greengrass Version 1 維護政策](#)。

Raspberry Pi GPIO [連接器](#)控制 Raspberry Pi 核心裝置上的一般用途輸入/輸出 (GPIO) 接腳。

此連接器以指定的間隔輪詢輸入接腳，並將狀態變更發佈到 MQTT 主題。它也接受來自使用者定義 Lambda 函數的讀取和寫入請求做為 MQTT 訊息。寫入請求用來將接腳設為高或低電壓。

連接器提供參數讓您用來指定輸入和輸出接腳。此行為是在群組部署之前設定。無法在執行時間變更。

- 輸入接腳可用於從周邊裝置接收資料。
- 輸出接腳可用於控制周邊設備，或將資料傳送到周邊設備。

您可以在許多案例中使用此連接器，例如：

- 控制紅綠燈的綠色、黃色及紅色 LED 燈。
- 根據濕度感應器的資料來控制風扇 (連接到繼電器)。
- 在零售店當客戶按下按鈕時通知員工。
- 使用智慧型電燈開關來控制其他 IoT 裝置。

Note

此連接器不適用於有即時需求的應用程式。期間較短的活動可能會被遺漏。

此連接器具有下列版本。

版本	ARN
3	arn:aws:greengrass: <i>region</i> ::/connectors/RaspberryPiGPIO/versions/3
2	arn:aws:greengrass: <i>region</i> ::/connectors/RaspberryPiGPIO/versions/2
1	arn:aws:greengrass: <i>region</i> ::/connectors/RaspberryPiGPIO/versions/1

如需版本變更的詳細資訊，請參閱 [Changelog](#)。

要求

此連接器有下列要求：

Version 3

- AWS IoT Greengrass 核心軟體 1.9.3 版或更新版本。
- 安裝在核心裝置上並新增至 PATH 環境變數的 [Python](#) 3.7 版。
- Raspberry Pi 4 模型 B，或 Raspberry Pi 3 模型 B/B+。您必須了解 Raspberry Pi 的接腳順序。如需詳細資訊，請參閱[the section called “GPIO 接腳順序”](#)。
- Greengrass 群組中的[本機裝置資源](#)，指向 Raspberry Pi /dev/gpiomem 上的。如果您在 主控台中建立資源，則必須選取擁有資源選項之 Linux 群組的自動新增作業系統群組許可。在 API 中，將 GroupOwnerSetting.AutoAddGroupOwner 屬性設定為 true。
- 安裝在 Raspberry Pi 的 [RPi.GPIO](#) 模組。在 Raspbian 中，預設已安裝這個模組。您可以使用下列命令來重新加以安裝：

```
sudo pip install RPi.GPIO
```

Versions 1 - 2

- AWS IoT Greengrass 核心軟體 1.7 版或更新版本。
- [Python](#) 2.7 版安裝在核心裝置上，並新增至 PATH 環境變數。
- Raspberry Pi 4 模型 B，或 Raspberry Pi 3 模型 B/B+。您必須了解 Raspberry Pi 的接腳順序。如需詳細資訊，請參閱[the section called “GPIO 接腳順序”](#)。
- Greengrass 群組中的[本機裝置資源](#)，指向 Raspberry Pi /dev/gpiomem 上的。如果您在 主控台中建立資源，則必須選取擁有資源選項之 Linux 群組的自動新增作業系統群組許可。在 API 中，將 GroupOwnerSetting.AutoAddGroupOwner 屬性設定為 true。
- 安裝在 Raspberry Pi 的 [RPi.GPIO](#) 模組。在 Raspbian 中，預設已安裝這個模組。您可以使用下列命令來重新加以安裝：

```
sudo pip install RPi.GPIO
```

GPIO 接腳順序

Raspberry Pi GPIO 連接器透過晶片上基礎系統 (SoC) 的編號方案來參考 GPIO 接腳，而不是 GPIO 接腳的實體配置。接腳的實體順序可能因 Raspberry Pi 版本而異。如需詳細資訊，請參閱 Raspberry Pi 文件中的 [GPIO](#)。

連接器無法驗證您設定的輸入和輸出接腳是否正確映射到 Raspberry Pi 的基礎硬體。如果接腳組態無效，連接器嘗試在裝置上啟動時會傳回執行時間錯誤。若要解決這個問題，請重新設定連接器，然後重新部署。

Note

請確定 GPIO 接腳的周邊設備已正確接線，以防止元件損壞。

連接器參數

此連接器提供下列參數：

InputGpios

要設定為輸入的 GPIO 接腳號碼 (逗號分隔清單)。您也可以選擇附加 U 來設定接腳的上拉電阻，或附加 D 來設定下拉電阻。範例："5,6U,7D"。

在 AWS IoT 主控台中顯示名稱：輸入 GPIO 接腳

必要：false。您必須指定輸入接腳、輸出接腳或兩者。

類型：string

有效模式：`^$|^([0-9]+[UD])?([0-9]+[UD])?$`

InputPollPeriod

每個輪詢操作的間隔 (毫秒)，依此檢查輸入 GPIO 接腳的狀態是否變更。最小值為 1。

這個值取決於您的案例和輪詢的裝置類型。例如，值 50 的速度應該足夠偵測到按鈕按下。

在 AWS IoT 主控台中顯示名稱：輸入 GPIO 輪詢期間

必要：false

類型：string

有效模式：`^$|^([1-9][0-9])?$`

OutputGpios

要設定為輸出的 GPIO 接腳號碼 (逗號分隔清單)。您也可以選擇附加 H 來設定高狀態 (1)，或附加 L 來設定低狀態 (0)。範例："8H,9,27L"。

在 AWS IoT 主控台中顯示名稱：輸出 GPIO 接腳

必要：false。您必須指定輸入接腳、輸出接腳或兩者。

類型：string

有效模式：`^$|^([0-9]+[HL])?([0-9]+[HL])?$`

GpioMem-ResourceId

代表 /dev/gpiomem 的本機裝置資源的 ID。

Note

此連接器已授予資源的讀寫存取權。

AWS IoT 主控台顯示名稱：/dev/gpiomem 裝置的資源

必要：true

類型：string

有效模式：.+

建立範例連接器 (AWS CLI)

下列 CLI 命令會使用包含 Raspberry Pi GPIO 連接器的ConnectorDefinition初始版本建立。

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-  
version '{  
  "Connectors": [  
    {  
      "Id": "MyRaspberryPiGPIOConnector",  
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/RaspberryPiGPIO/  
versions/3",  
      "Parameters": {  
        "GpioMem-ResourceId": "my-gpio-resource",  
        "InputGpios": "5,6U,7D",  
        "InputPollPeriod": 50,  
        "OutputGpios": "8H,9,27L"  
      }  
    }  
  ]  
}'
```

Note

此連接器中的 Lambda 函數具有[長期生命週期](#)。

在 AWS IoT Greengrass 主控台中，您可以從群組的連接器頁面新增連接器。如需詳細資訊，請參閱[the section called “連接器入門 \(主控台\)”](#)。

輸入資料

此連接器接受兩個 MQTT 主題上對 GPIO 接腳的讀取或寫入請求。

- gpio/+ / + / read 主題上的讀取請求。

- `gpio/+/+write` 主題上的寫入請求。

若要發佈到這些主題，請將 + 萬用字元分別換成核心物件名稱和目標接腳號碼。例如：

```
gpio/core-thing-name/gpio-number/read
```

Note

目前，當您建立使用 Raspberry Pi GPIO 連接器的訂閱時，您必須在主題中指定至少一個 + 萬用字元的值。

主題篩選條件：`gpio/+/+read`

使用此主題來引導連接器讀取主題中指定之 GPIO 接腳的狀態。

連接器將回應發佈到對應的輸出主題 (例如，`gpio/core-thing-name/gpio-number/state`)。

訊息屬性

無。忽略傳送到這個主題的訊息。

主題篩選條件：`gpio/+/+write`

使用此主題將寫入請求傳送到 GPIO 接腳。這引導連接器將主題中指定的 GPIO 接腳設定為低或高電壓。

- 0 將接腳設定為低電壓。
- 1 將接腳設定為高電壓。

連接器將回應發佈到對應的輸出 `/state` 主題 (例如，`gpio/core-thing-name/gpio-number/state`)。

訊息屬性

值 0 或 1，做為整數或字串。

範例輸入

```
0
```

輸出資料

此連接器將資料發佈到兩個主題：

- `gpio/+/+/state` 主題上高或低狀態變更。
- `gpio/+/error` 主題的相關錯誤。

主題篩選條件：`gpio/+/+/state`

使用此主題監聽輸入接腳的狀態變更和讀取請求的回應。如果接腳處於低狀態，連接器會傳回字串 "0"，或者，如果是高狀態，則傳回 "1"。

發佈到此主題時，連接器會將 + 萬用字元分別換成核心物件名稱和目標接腳。例如：

```
gpio/core-thing-name/gpio-number/state
```

Note

目前，當您建立使用 Raspberry Pi GPIO 連接器的訂閱時，您必須在主題中指定至少一個 + 萬用字元的值。

範例輸出

```
0
```

主題篩選條件：`gpio/+/error`

使用此主題監聽錯誤。連接器會因為請求無效而發佈到本主題 (例如，在輸入接腳上請求狀態變更時)。

發佈到此主題時，連接器會將 + 萬用字元換成核心物件名稱。

範例輸出

```
{
  "topic": "gpio/my-core-thing/22/write",
  "error": "Invalid GPIO operation",
  "long_description": "GPIO 22 is configured as an INPUT GPIO. Write operations are not permitted."
}
```

使用範例

使用下列高階步驟來設定範例 Python 3.7 Lambda 函數，您可以用來嘗試連接器。

Note

- 如果您使用其他 Python 執行期，則可以建立從 Python3.x 到 Python 3.7 的符號連結。
- [連接器入門 \(主控台\)](#) 和 [連接器入門 \(CLI\)](#) 主題包含詳細步驟，說明如何設定和部署範例 Twilio 通知連接器。

1. 確定您符合連接器的[要求](#)。
2. 建立並發佈 Lambda 函數，將輸入資料傳送至連接器。

將範例程式碼儲存為 PY 檔案。下載並解壓縮[AWS IoT Greengrass 適用於 Python 的核心 SDK](#)。然後，建立在根層級包含 PY 檔案和 greengrasssdk 資料夾的 zip 套件。此 zip 套件是您上傳至的部署套件 AWS Lambda。

建立 Python 3.7 Lambda 函數之後，請發佈函數版本並建立別名。

3. 設定 Greengrass 群組。
 - a. 依別名新增 Lambda 函數（建議）。將 Lambda 生命週期設定為長期（或在 CLI "Pinned": true 中）。
 - b. 新增必要的本機裝置資源，並授予 Lambda 函數的讀取/寫入存取權。
 - c. 新增連接器並設定其[參數](#)。
 - d. 新增訂閱，允許連接器在支援主題篩選條件上接收[輸入資料](#)並傳送[輸出資料](#)。
 - 將 Lambda 函數設定為來源，將連接器設定為目標，並使用支援的輸入主題篩選條件。
 - 將連接器設為來源、將 AWS IoT Core 設為目標，並使用支援的輸出主題篩選條件。您可以使用此訂閱在 AWS IoT 主控台中檢視狀態訊息。
4. 部署群組。
5. 在 AWS IoT 主控台的測試頁面上，訂閱輸出資料主題，以檢視連接器的狀態訊息。Lambda 函數範例為長期函數，並在部署群組後立即開始傳送訊息。

完成測試後，您可以將 Lambda 生命週期設定為隨需（或在 CLI "Pinned": false 中）並部署群組。這會讓函數停止傳送訊息。

範例

下列範例 Lambda 函數會將輸入訊息傳送至連接器。此範例會傳送一組輸入 GPIO 接腳的讀取請求。它會說明如何使用核心物件名稱和 PIN 碼來建構主題。

```
import greengrasssdk
import json
import os

iot_client = greengrasssdk.client('iot-data')
INPUT_GPIOS = [6, 17, 22]

thingName = os.environ['AWS_IOT_THING_NAME']

def get_read_topic(gpio_num):
    return '/'.join(['gpio', thingName, str(gpio_num), 'read'])

def get_write_topic(gpio_num):
    return '/'.join(['gpio', thingName, str(gpio_num), 'write'])

def send_message_to_connector(topic, message=''):
    iot_client.publish(topic=topic, payload=str(message))

def set_gpio_state(gpio, state):
    send_message_to_connector(get_write_topic(gpio), str(state))

def read_gpio_state(gpio):
    send_message_to_connector(get_read_topic(gpio))

def publish_basic_message():
    for i in INPUT_GPIOS:
        read_gpio_state(i)

publish_basic_message()

def lambda_handler(event, context):
    return
```

授權

Raspberry Pi GPIO；連接器包含下列第三方軟體/授權：

- [RPi.GPIO](#)/MIT

此連接器根據 [Greengrass 核心軟體授權合約](#) 發行。

變更記錄

下表說明每個連接器版本中的變更。

版本	變更
3	將 Lambda 執行時間升級至 Python 3.7，這會變更執行時間需求。
2	更新連接器 ARN 以取得 AWS 區域 支援。
1	初始版本。

Greengrass 群組一次只能包含一個版本的連接器。若要取得有關升級連接器版本的資訊，請參閱[the section called “升級連接器版本”](#)。

另請參閱

- [使用連接器整合服務和通訊協定](#)
- [the section called “連接器入門 \(主控台\)”](#)
- [the section called “連接器入門 \(CLI\)”](#)
- Raspberry Pi 文件中的 [GPIO](#)

序列串流連接器

Warning

此連接器已進入延長的生命週期階段，AWS IoT Greengrass 且不會發佈提供現有功能、增強功能、安全修補程式或錯誤修正的更新。如需詳細資訊，請參閱[AWS IoT Greengrass Version 1 維護政策](#)。

序列串流[連接器](#)會讀取和寫入 AWS IoT Greengrass 核心裝置上的序列連接埠。

此連接器支援兩種操作模式：

- 隨需讀取。接收 MQTT 主題上的讀取和寫入請求，並發佈讀取操作的回應，或寫入操作的狀態。
- 輪詢讀取。定期從序列埠讀取。這個模式還支援隨需讀取請求。

Note

讀取請求的最大讀取長度限制為 63994 位元組。寫入請求的最大資料長度限制為 128000 位元組。

此連接器具有下列版本。

版本	ARN
3	arn:aws:greengrass: <i>region</i> ::/connectors/SerialStream/versions/3
2	arn:aws:greengrass: <i>region</i> ::/connectors/SerialStream/versions/2
1	arn:aws:greengrass: <i>region</i> ::/connectors/SerialStream/versions/1

如需版本變更的詳細資訊，請參閱 [Changelog](#)。

要求

此連接器有下列要求：

Version 3

- AWS IoT Greengrass 核心軟體 1.9.3 版或更新版本。
- 安裝在核心裝置上並新增至 PATH 環境變數的 [Python](#) 3.7 或 3.8 版。

Note

若要使用 Python 3.8，請執行下列命令，從預設 Python 3.7 安裝資料夾建立符號連結，以連接至已安裝的 Python 3.8 二進位檔。

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

這會設定您的裝置以符合 AWS IoT Greengrass 的 Python 需求。

- Greengrass 群組中的[本機裝置資源](#)，指向目標序列埠。

Note

在您部署此連接器之前，建議您設定序列埠並驗證是否可對該序列埠進行讀取和寫入。

Versions 1 - 2

- AWS IoT Greengrass 核心軟體 1.7 版或更新版本。
- [Python](#) 2.7 版安裝在核心裝置上，並新增至 PATH 環境變數。
- Greengrass 群組中的[本機裝置資源](#)，指向目標序列埠。

Note

在您部署此連接器之前，建議您設定序列埠並驗證是否可對該序列埠進行讀取和寫入。

連接器參數

此連接器提供下列參數：

BaudRate

序列連線的傳輸速率。

AWS IoT 主控台顯示名稱：傳輸速率

必要：true

類型：string

有效值：110, 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 56000, 57600, 115200, 230400

有效模式：`^110$|^300$|^600$|^1200$|^2400$|^4800$|^9600$|^14400$|^19200$|^28800$|^38400$|^56000$|^57600$|^115200$|^230400$`

Timeout

讀取操作的逾時 (以秒為單位)。

AWS IoT 主控台顯示名稱：逾時

必要：true

類型：string

有效值：1 - 59

有效模式：`^([1-9]|[1-5][0-9])$`

SerialPort

裝置上實體序列埠的絕對路徑。這是指定給本機裝置資源的來源路徑。

AWS IoT 主控台顯示名稱：序列連接埠

必要：true

類型：string

有效模式：`[/a-zA-Z0-9_-]+`

SerialPort-ResourceId

代表實體序列埠的本機裝置資源 ID。

Note

此連接器已授予資源的讀寫存取權。

AWS IoT 主控台顯示名稱：序列連接埠資源

必要：true

類型：string

有效模式：[a-zA-Z0-9_-]+

PollingRead

設定讀取模式：輪詢讀取或隨需讀取。

- 對於輪詢讀取模式，指定 true。在此模式中，需要 PollingInterval、PollingReadType 和 PollingReadLength 屬性。
- 對於隨需讀取模式，指定 false。在這個模式下，類型和長度值是在讀取請求中指定。

AWS IoT 主控台顯示名稱：讀取模式

必要：true

類型：string

有效值：true, false

有效模式：^([Tt][Rr][Uu][Ee]|[Ff][Aa][Ll][Ss][Ee])\$

PollingReadLength

在每個輪詢讀取操作中讀取的資料長度 (以位元組為單位)。這只適用於使用輪詢讀取模式時。

在 AWS IoT 主控台中顯示名稱：輪詢讀取長度

必要：false。當 PollingRead 為 true 時，此為必要屬性。

類型：string

有效模式：^([1-9][0-9]{0,3}|[1-5][0-9]{4}|6[0-2][0-9]{3}|63[0-8][0-9]{2}|639[0-8][0-9]|6399[0-4])\$

PollingReadInterval

輪詢讀取的間隔 (以秒為單位)。這只適用於使用輪詢讀取模式時。

在 AWS IoT 主控台中顯示名稱：輪詢讀取間隔

必要：false。當 PollingRead 為 true 時，此為必要屬性。

類型：string

有效值：1 - 999

有效模式：^([1-9]|[1-9][0-9]|[1-9][0-9][0-9])\$

PollingReadType

輪詢執行緒讀取的資料類型。這只適用於使用輪詢讀取模式時。

在 AWS IoT 主控台中顯示名稱：輪詢讀取類型

必要：false。當 PollingRead 為 true 時，此為必要屬性。

類型：string

有效值：ascii, hex

有效模式：`^(|[Aa][Ss][Cc][Ii][Ii]|[Hh][Ee][Xx])$`

RtsCts

指出是否啟用 RTS/CTS 流程控制。預設值為 false。如需詳細資訊，請參閱 [RTS、CTS 及 RTR](#)。

AWS IoT 主控台顯示名稱：RTS/CTS 流程控制

必要：false

類型：string

有效值：true, false

有效模式：`^(|[Tt][Rr][Uu][Ee]|[Ff][Aa][Ll][Ss][Ee])$`

XonXoff

指出是否啟用軟體流程控制。預設值為 false。如需詳細資訊，請參閱 [軟體流程控制](#)。

AWS IoT 主控台顯示名稱：軟體流程控制

必要：false

類型：string

有效值：true, false

有效模式：`^(|[Tt][Rr][Uu][Ee]|[Ff][Aa][Ll][Ss][Ee])$`

Parity

序列埠的同位。預設值為 N。如需詳細資訊，請參閱 [同位](#)。

AWS IoT 主控台顯示名稱：序列連接埠同位

必要：false

類型：string

有效值：N, E, O, S, M

有效模式：`^([NEOSMneosm])$`

建立範例連接器 (AWS CLI)

下列 CLI 命令 `ConnectorDefinition` 會使用包含序列串流連接器的初始版本來建立。它將連接器設定為輪詢讀取模式。

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
  "Connectors": [
    {
      "Id": "MySerialStreamConnector",
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/SerialStream/
versions/3",
      "Parameters": {
        "BaudRate" : "9600",
        "Timeout" : "25",
        "SerialPort" : "/dev/serial1",
        "SerialPort-ResourceId" : "my-serial-port-resource",
        "PollingRead" : "true",
        "PollingReadLength" : "30",
        "PollingReadInterval" : "30",
        "PollingReadType" : "hex"
      }
    }
  ]
}'
```

在 AWS IoT Greengrass 主控台中，您可以從群組的連接器頁面新增連接器。如需詳細資訊，請參閱 [the section called “連接器入門 \(主控台\)”](#)。

輸入資料

此連接器接受兩個 MQTT 主題上序列連接埠的讀取或寫入請求。輸入訊息必須是 JSON 格式。

- `serial/+/read/#` 主題上的讀取請求。

- `serial/+ /write/#` 主題上的寫入請求。

若要發佈到這些主題，請將 `+` 萬用字元換成核心物件名稱，將 `#` 萬用字元換成序列埠的路徑。例如：

```
serial/core-thing-name/read/dev/serial-port
```

主題篩選條件：`serial/+ /read/#`

使用此主題將隨需讀取請求傳送到序列接腳。讀取請求的最大讀取長度限制為 63994 位元組。

訊息屬性

`readLength`

從序列埠讀取的資料長度。

必要：`true`

類型：`string`

有效模式：`^[1-9][0-9]*$`

`type`

要讀取的資料類型。

必要：`true`

類型：`string`

有效值：`ascii, hex`

有效模式：`(?i)^(ascii|hex)$`

`id`

請求的任意 ID。此屬性用於將輸入請求映射到輸出回應。

必要：`false`

類型：`string`

有效模式：`.+`

範例輸入

```
{
```

```
"readLength": "30",  
"type": "ascii",  
"id": "abc123"  
}
```

主題篩選條件：serial/+/write/#

使用此主題將寫入請求傳送到序列接腳。寫入請求的最大資料長度限制為 128000 位元組。

訊息屬性

data

要寫入序列埠的字串。

必要：true

類型：string

有效模式：`^[1-9][0-9]*$`

type

要讀取的資料類型。

必要：true

類型：string

有效值：ascii, hex

有效模式：`^(ascii|hex|ASCII|HEX)$`

id

請求的任意 ID。此屬性用於將輸入請求映射到輸出回應。

必要：false

類型：string

有效模式：`.+`

範例輸入：ASCII 請求

```
{  
  "data": "random serial data",  
}
```

```
"type": "ascii",  
"id": "abc123"  
}
```

範例輸入：十六進位請求

```
{  
  "data": "base64 encoded data",  
  "type": "hex",  
  "id": "abc123"  
}
```

輸出資料

連接器將輸出資料發佈在兩個主題：

- 在 serial/+/status/# 主題上來自連接器的狀態資訊。
- 在 serial/+/read_response/# 主題上來自讀取請求的回應。

發佈到此主題時，連接器會將 + 萬用字元換成核心物件名稱，將 # 萬用字元換成序列埠的路徑。例如：

```
serial/core-thing-name/status/dev/serial-port
```

主題篩選條件：serial/+/status/#

使用此主題監聽讀取和寫入請求的狀態。如果 id 屬性包含在請求中，則會在回應中傳回。

範例輸出：成功

```
{  
  "response": {  
    "status": "success"  
  },  
  "id": "abc123"  
}
```

範例輸出：失敗

失敗回應包含 error_message 屬性，描述執行讀取或寫入操作時發生的錯誤或逾時。


```
{
  "response": {
    "status": "fail",
    "error_message": "Could not write to port"
  },
  "id": "abc123"
}
```

主題篩選條件：`serial/+/read_response/#`

使用此主題，以接收來自讀取操作的回應資料。如果類型為 hex，則回應資料為 Base64 編碼。

範例輸出

```
{
  "data": "output of serial read operation"
  "id": "abc123"
}
```

使用範例

使用下列高階步驟來設定範例 Python 3.7 Lambda 函數，您可以用來嘗試連接器。

Note

- 如果您使用其他 Python 執行期，則可以建立從 Python3.x 到 Python 3.7 的符號連結。
- [連接器入門 \(主控台\)](#) 和 [連接器入門 \(CLI\)](#) 主題包含詳細步驟，說明如何設定和部署範例 Twilio 通知連接器。

1. 確定您符合連接器的[要求](#)。
2. 建立並發佈 Lambda 函數，將輸入資料傳送至連接器。

將範例程式碼儲存為 PY 檔案。下載並解壓縮[AWS IoT Greengrass 適用於 Python 的核心 SDK](#)。然後，建立在根層級包含 PY 檔案和 greengrasssdk 資料夾的 zip 套件。此 zip 套件是您上傳至的部署套件 AWS Lambda。

建立 Python 3.7 Lambda 函數之後，請發佈函數版本並建立別名。

3. 設定 Greengrass 群組。

- a. 依別名新增 Lambda 函數（建議）。將 Lambda 生命週期設定為長期（或在 CLI "Pinned": true 中）。
 - b. 新增必要的本機裝置資源，並授予 Lambda 函數的讀取/寫入存取權。
 - c. 將連接器新增至群組並設定其[參數](#)。
 - d. 將訂閱新增至群組，允許連接器在支援主題篩選條件上接收[輸入資料](#)並傳送[輸出資料](#)。
 - 將 Lambda 函數設定為來源，將連接器設定為目標，並使用支援的輸入主題篩選條件。
 - 將連接器設為來源、將 AWS IoT Core 設為目標，並使用支援的輸出主題篩選條件。您可以使用此訂閱在 AWS IoT 主控台中檢視狀態訊息。
4. 部署群組。
 5. 在 AWS IoT 主控台的測試頁面上，訂閱輸出資料主題，以檢視連接器的狀態訊息。Lambda 函數範例為長期函數，並在部署群組後立即開始傳送訊息。

完成測試後，您可以將 Lambda 生命週期設定為隨需（或在 CLI "Pinned": false 中）並部署群組。這會讓函數停止傳送訊息。

範例

下列範例 Lambda 函數會將輸入訊息傳送至連接器。

```
import greengrasssdk
import json

TOPIC_REQUEST = 'serial/CORE_THING_NAME/write/dev/serial1'

# Creating a greengrass core sdk client
iot_client = greengrasssdk.client('iot-data')

def create_serial_stream_request():
    request = {
        "data": "TEST",
        "type": "ascii",
        "id": "abc123"
    }
    return request

def publish_basic_request():
    iot_client.publish(payload=json.dumps(create_serial_stream_request()),
        topic=TOPIC_REQUEST)
```

```
publish_basic_request()

def lambda_handler(event, context):
    return
```

授權

序列串流連接器包含下列第三方軟體/授權：

- [pyserial](#)/BSD

此連接器根據 [Greengrass 核心軟體授權合約](#) 發行。

變更記錄

下表說明每個連接器版本中的變更。

版本	變更
3	將 Lambda 執行時間升級至 Python 3.7，這會變更執行時間需求。
2	更新連接器 ARN 以取得 AWS 區域 支援。
1	初始版本。

Greengrass 群組一次只能包含一個版本的連接器。若要取得有關升級連接器版本的資訊，請參閱[the section called “升級連接器版本”](#)。

另請參閱

- [使用連接器整合服務和通訊協定](#)
- [the section called “連接器入門 \(主控台\)”](#)
- [the section called “連接器入門 \(CLI\)”](#)

ServiceNow MetricBase 整合連接器

Warning

此連接器已進入延長生命週期階段，AWS IoT Greengrass 且不會發佈提供現有功能、增強功能、安全修補程式或錯誤修正的更新。如需詳細資訊，請參閱[AWS IoT Greengrass Version 1 維護政策](#)。

ServiceNow MetricBase 整合[連接器](#)會將時間序列指標從 Greengrass 裝置發佈至 ServiceNow MetricBase。這可讓您存放、分析和視覺化來自 Greengrass 核心環境的時間序列資料，並對本機事件採取行動。

此連接器會接收 MQTT 主題上的時間序列資料，並定期將資料發佈到 ServiceNow API。

您可以使用此連接器來支援如下案例：

- 根據從 Greengrass 裝置收集的時間序列資料，建立以閾值為基礎的提醒和警示。
- 使用來自 Greengrass 裝置的時間服務資料，搭配在 ServiceNow 平台上建構的自訂應用程式。

此連接器具有下列版本。

版本	ARN
4	arn:aws:greengrass: <i>region</i> ::/connectors/ServiceNowMetricBaseIntegration/versions/4
3	arn:aws:greengrass: <i>region</i> ::/connectors/ServiceNowMetricBaseIntegration/versions/3
2	arn:aws:greengrass: <i>region</i> ::/connectors/ServiceNowMetricBaseIntegration/versions/2

版本	ARN
1	arn:aws:greengrass: <i>region</i> ::/connectors/ServiceNowMetricBaseIntegration/versions/1

如需版本變更的詳細資訊，請參閱 [Changelog](#)。

要求

此連接器有下列要求：

Version 3 - 4

- AWS IoT Greengrass 核心軟體 v1.9.3 或更新版本。AWS IoT Greengrass 必須設定為支援本機秘密，如[秘密要求](#)中所述。

Note

此要求包括允許存取您的 Secrets Manager 秘密。如果您使用的是預設 Greengrass 服務角色，Greengrass 具有許可，可取得名稱開頭為 greengrass- 的秘密值。

- 安裝在核心裝置上並新增至 PATH 環境變數的 [Python](#) 3.7 或 3.8 版。

Note

若要使用 Python 3.8，請執行下列命令，從預設 Python 3.7 安裝資料夾建立符號連結，以連接至已安裝的 Python 3.8 二進位檔。

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

這會設定您的裝置以符合 AWS IoT Greengrass 的 Python 需求。

- 具有 MetricBase 有效訂閱的 ServiceNow 帳戶。此外，帳戶中必須建立指標和指標資料表。如需詳細資訊，請參閱 ServiceNow 文件中的 [MetricBase](#)。
- 在 中 AWS Secrets Manager 存放使用者名稱和密碼的文字類型秘密，以使用基本身分驗證登入您的 ServiceNow 執行個體。此私密必須包含「使用者」和「密碼」金鑰與對應的值。如需詳細資訊，請參閱 AWS Secrets Manager 《使用者指南》中的[建立基本秘密](#)。

- Greengrass 群組中參考 Secrets Manager 秘密的秘密資源。如需詳細資訊，請參閱[將私密部署至核心](#)。

Versions 1 - 2

- AWS IoT Greengrass 核心軟體 v1.7 或更新版本。AWS IoT Greengrass 必須設定為支援本機秘密，如[秘密要求](#)中所述。

Note

此要求包括允許存取您的 Secrets Manager 秘密。如果您使用預設 Greengrass 服務角色，Greengrass 具有許可，可取得名稱開頭為 greengrass- 的秘密值。

- [Python](#) 2.7 版安裝在核心裝置上，並新增至 PATH 環境變數。
- 具有 MetricBase 有效訂閱的 ServiceNow 帳戶。此外，帳戶中必須建立指標和指標資料表。如需詳細資訊，請參閱 ServiceNow 文件中的 [MetricBase](#)。
- 在中 AWS Secrets Manager 存放使用者名稱和密碼的文字類型秘密，以使用基本身分驗證登入您的 ServiceNow 執行個體。此私密必須包含「使用者」和「密碼」金鑰與對應的值。如需詳細資訊，請參閱 AWS Secrets Manager 《使用者指南》中的[建立基本秘密](#)。
- Greengrass 群組中參考 Secrets Manager 秘密的秘密資源。如需詳細資訊，請參閱[將私密部署至核心](#)。

連接器參數

此連接器提供下列參數：

Version 4

PublishInterval

將事件發佈到 ServiceNow 之間等待的秒數上限。最大值為 900。

連接器 PublishBatchSize 會在達到或 PublishInterval 過期時發佈至 ServiceNow。

在 AWS IoT 主控台中顯示名稱：以秒為單位發佈間隔

必要：true

類型：string

有效值：1 - 900

有效模式：[1-9]|[1-9]\d|[1-9]\d\d|900

PublishBatchSize

發佈到 ServiceNow 之前可批次處理的指標值數量上限。

連接器 PublishBatchSize 會在達到或 PublishInterval 過期時發佈至 ServiceNow。

在 AWS IoT 主控台中顯示名稱：發佈批次大小

必要：true

類型：string

有效模式：^[0-9]+\$

InstanceName

用於連接到 ServiceNow 的執行個體名稱。

在 AWS IoT 主控台中顯示名稱：ServiceNow 執行個體的名稱

必要：true

類型：string

有效模式：.+

DefaultTableName

資料表的名稱，其中包含與時間序列 MetricBase 資料庫相關聯的 GlideRecord。輸入訊息承載中的 table 屬性可用來覆寫這個值。

在 AWS IoT 主控台中顯示名稱：要包含指標的資料表名稱

必要：true

類型：string

有效模式：.+

MaxMetricsToRetain

由新指標取代之之前在記憶體中儲存的指標數目上限。

在沒有網際網路連線且連接器開始緩衝指標來稍後發佈時，就受此限制。當緩衝區已滿時，新指標會取代最舊指標。

 Note

如果連接器的主機程序中斷，則不會儲存指標。例如，在群組部署或裝置重新啟動期間就可能發生這種情況。

這個值應該大於批次大小，且根據 MQTT 訊息的傳入速率，大小足夠保留訊息。

在 AWS IoT 主控台中顯示名稱：記憶體中保留的指標上限

必要：true

類型：string

有效模式：`^[0-9]+$`

AuthSecretArn

中 AWS Secrets Manager 存放 ServiceNow 使用者名稱和密碼的秘密。這必須是文字類型私密。此私密必須包含「使用者」和「密碼」金鑰與對應的值。

在 AWS IoT 主控台中顯示名稱：身分驗證秘密的 ARN

必要：true

類型：string

有效模式：`arn:aws:secretsmanager:[a-z0-9\-\+]:[0-9]{12}:secret:([a-zA-Z0-9\-\+\/)*[a-zA-Z0-9/_+=,.\@-\+]-[a-zA-Z0-9]+`

AuthSecretArn-ResourceId

群組中參考 ServiceNow 登入資料的 Secrets Manager 秘密的秘密資源。

AWS IoT 主控台顯示名稱：Auth 字符資源

必要：true

類型：string

有效模式：`.\+`

IsolationMode

此連接器的[容器化](#)模式。預設值為 `GreengrassContainer`，這表示連接器在 AWS IoT Greengrass 容器內的隔離執行時間環境中執行。

Note

群組的預設容器化設定不會套用至連接器。

在 AWS IoT 主控台中顯示名稱：容器隔離模式

必要：false

類型：string

有效值：GreengrassContainer 或 NoContainer

有效模式：`^NoContainer$|^GreengrassContainer$`

Version 1 - 3

PublishInterval

將事件發佈到 ServiceNow 之間等待的秒數上限。最大值為 900。

連接器 `PublishBatchSize` 會在達到或 `PublishInterval` 過期時發佈至 ServiceNow。

在 AWS IoT 主控台中顯示名稱：以秒為單位發佈間隔

必要：true

類型：string

有效值：1 - 900

有效模式：`[1-9]|[1-9]\d|[1-9]\d\d|900`

PublishBatchSize

發佈到 ServiceNow 之前可批次處理的指標值數量上限。

連接器 `PublishBatchSize` 會在達到或 `PublishInterval` 過期時發佈至 ServiceNow。

在 AWS IoT 主控台中顯示名稱：發佈批次大小

必要：true

類型：string

有效模式：`^[0-9]+$`

InstanceName

用於連接到 ServiceNow 的執行個體名稱。

在 AWS IoT 主控台中顯示名稱：ServiceNow 執行個體的名稱

必要：true

類型：string

有效模式：`.+`

DefaultTableName

資料表的名稱，其中包含與時間序列 MetricBase 資料庫相關聯的 GlideRecord。輸入訊息承載中的 table 屬性可用來覆寫這個值。

在 AWS IoT 主控台中顯示名稱：要包含指標的資料表名稱

必要：true

類型：string

有效模式：`.+`

MaxMetricsToRetain

由新指標取代之之前在記憶體中儲存的指標數目上限。

在沒有網際網路連線且連接器開始緩衝指標來稍後發佈時，就受此限制。當緩衝區已滿時，新指標會取代最舊指標。

Note

如果連接器的主機程序中斷，則不會儲存指標。例如，在群組部署或裝置重新啟動期間就可能發生這種情況。

這個值應該大於批次大小，且根據 MQTT 訊息的傳入速率，大小足夠保留訊息。

在 AWS IoT 主控台中顯示名稱：記憶體中保留的指標上限

必要：true

類型：string

有效模式：`^[0-9]+$`

AuthSecretArn

中 AWS Secrets Manager 存放 ServiceNow 使用者名稱和密碼的秘密。這必須是文字類型私密。此私密必須包含「使用者」和「密碼」金鑰與對應的值。

在 AWS IoT 主控台中顯示名稱：身分驗證秘密的 ARN

必要：true

類型：string

有效模式：`arn:aws:secretsmanager:[a-z0-9\-\+]:[0-9]{12}:secret:(\[a-zA-Z0-9\-\+\/\]*[a-zA-Z0-9/_+=,.\@-\+]-[a-zA-Z0-9]+`

AuthSecretArn-ResourceId

群組中參考 ServiceNow 登入資料的 Secrets Manager 秘密的秘密資源。

在 AWS IoT 主控台中顯示名稱：Auth 字符資源

必要：true

類型：string

有效模式：`.\+`

建立範例連接器 (AWS CLI)

下列 CLI 命令 `ConnectorDefinition` 會使用包含 ServiceNow MetricBase 整合連接器的初始版本來建立。

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
  "Connectors": [
    {
```

```

      "Id": "MyServiceNowMetricBaseIntegrationConnector",
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/ServiceNowMetricBaseIntegration/versions/4",
      "Parameters": {
        "PublishInterval" : "10",
        "PublishBatchSize" : "50",
        "InstanceName" : "myinstance",
        "DefaultTableName" : "u_greengrass_app",
        "MaxMetricsToRetain" : "20000",
        "AuthSecretArn" : "arn:aws:secretsmanager:region:account-id:secret:greengrass-secret-hash",
        "AuthSecretArn-ResourceId" : "MySecretResource",
        "IsolationMode" : "GreengrassContainer"
      }
    }
  ]
}'

```

Note

此連接器中的 Lambda 函數具有[長期生命週期](#)。

在 AWS IoT Greengrass 主控台中，您可以從群組的連接器頁面新增連接器。如需詳細資訊，請參閱[the section called “連接器入門 \(主控台\)”](#)。

輸入資料

此連接器接受 MQTT 主題上的時間序列指標，並將指標發佈至 ServiceNow。輸入訊息必須是 JSON 格式。

訂閱中的主題篩選條件

servicenow/metricbase/metric

訊息屬性

request

資料表、記錄和指標的相關資訊。這個請求代表時間序列 POST 請求中的 seriesRef 物件。如需詳細資訊，請參閱 [Clotho 時間序列 API - POST](#)。

必要：true

類型：object 包含下列屬性：

subject

資料表中特定記錄的 sys_id。

必要：true

類型：string

metric_name

指標欄位名稱。

必要：true

類型：string

table

用於存放記錄的資料表名稱。指定此值以覆寫 DefaultTableName 參數。

必要：false

類型：string

value

個別資料點的值。

必要：true

類型：float

timestamp

個別資料點的時間戳記。預設值是目前時間。

必要：false

類型：string

範例輸入

```
{
  "request": {
    "subject": "ef43c6d40a0a0b5700c77f9bf387afe3",
    "metric_name": "u_count",
    "table": "u_greengrass_app"
```

```
    "value": 1.0,  
    "timestamp": "2018-10-14T10:30:00"  
  }  
}
```

輸出資料

這個連接器會將狀態資訊發佈為輸出資料，且主題為 MQTT。

訂閱中的主題篩選條件

`servicenow/metricbase/metric/status`

範例輸出：成功

```
{  
  "response": {  
    "metric_name": "Errors",  
    "table_name": "GliderProd",  
    "processed_on": "2018-10-14T10:35:00",  
    "response_id": "khjKSkj132qwr23fcba",  
    "status": "success",  
    "values": [  
      {  
        "timestamp": "2016-10-14T10:30:00",  
        "value": 1.0  
      },  
      {  
        "timestamp": "2016-10-14T10:31:00",  
        "value": 1.1  
      }  
    ]  
  }  
}
```

範例輸出：失敗

```
{  
  "response": {  
    "error": "InvalidInputException",  
    "error_message": "metric value is invalid",  
    "status": "fail"  
  }  
}
```

```
}
```

Note

如果連接器偵測到可重試的錯誤（例如連線錯誤），則會在下一個批次中重試發佈。

使用範例

使用下列高階步驟來設定範例 Python 3.7 Lambda 函數，您可以用來嘗試連接器。

Note

- 如果您使用其他 Python 執行期，則可以建立從 Python3.x 到 Python 3.7 的符號連結。
- [連接器入門 \(主控台\)](#) 和 [連接器入門 \(CLI\)](#) 主題包含詳細步驟，說明如何設定和部署範例 Twilio 通知連接器。

1. 確定您符合連接器的[要求](#)。
2. 建立並發佈 Lambda 函數，將輸入資料傳送至連接器。

將[範例程式碼](#)儲存為 PY 檔案。下載並解壓縮[AWS IoT Greengrass 適用於 Python 的核心 SDK](#)。然後，建立在根層級包含 PY 檔案和 greengrasssdk 資料夾的 zip 套件。此 zip 套件是您上傳至的部署套件 AWS Lambda。

建立 Python 3.7 Lambda 函數後，發佈函數版本並建立別名。

3. 設定 Greengrass 群組。
 - a. 依別名新增 Lambda 函數（建議）。將 Lambda 生命週期設定為長期（或在 CLI "Pinned": true 中）。
 - b. 新增必要的秘密資源，並授予 Lambda 函數的讀取存取權。
 - c. 新增連接器並設定其[參數](#)。
 - d. 新增訂閱，允許連接器在支援主題篩選條件上接收[輸入資料](#)並傳送[輸出資料](#)。
 - 將 Lambda 函數設定為來源，將連接器設定為目標，並使用支援的輸入主題篩選條件。
 - 將連接器設為來源、將 AWS IoT Core 設為目標，並使用支援的輸出主題篩選條件。您可以使用此訂閱在 AWS IoT 主控台中檢視狀態訊息。

4. 部署群組。
5. 在 AWS IoT 主控台的測試頁面上，訂閱輸出資料主題，以檢視連接器的狀態訊息。Lambda 函數範例為長期函數，並在部署群組後立即開始傳送訊息。

完成測試後，您可以將 Lambda 生命週期設定為隨需（或在 CLI "Pinned": false 中）並部署群組。這會讓函數停止傳送訊息。

範例

下列範例 Lambda 函數會將輸入訊息傳送至連接器。

```
import greengrasssdk
import json

iot_client = greengrasssdk.client('iot-data')
SEND_TOPIC = 'servicenow/metricbase/metric'

def create_request_with_all_fields():
    return {
        "request": {
            "subject": '2efdf6badbd523803acfae441b961961',
            "metric_name": 'u_count',
            "value": 1234,
            "timestamp": '2018-10-20T20:22:20',
            "table": 'u_greengrass_metricbase_test'
        }
    }

def publish_basic_message():
    messageToPublish = create_request_with_all_fields()
    print("Message To Publish: ", messageToPublish)
    iot_client.publish(topic=SEND_TOPIC,
                      payload=json.dumps(messageToPublish))

publish_basic_message()

def lambda_handler(event, context):
    return
```

授權

ServiceNow MetricBase 整合連接器包含下列第三方軟體/授權：

- [pysnow/MIT](#)

此連接器根據 [Greengrass 核心軟體授權合約](#) 發行。

變更記錄

下表說明每個版本的連接器的變更。

版本	變更
4	已新增 <code>IsolationMode</code> 參數，以設定連接器的容器化模式。
3	將 Lambda 執行時間升級至 Python 3.7，這會變更執行時間需求。
2	可減少過多記錄的修正。
1	初始版本。

Greengrass 群組一次只能包含一個版本的連接器。若要取得有關升級連接器版本的資訊，請參閱[the section called “升級連接器版本”](#)。

另請參閱

- [使用連接器整合服務和通訊協定](#)
- [the section called “連接器入門 \(主控台\)”](#)
- [the section called “連接器入門 \(CLI\)”](#)

SNS 連接器

SNS [連接器](#) 會將訊息發佈至 Amazon SNS 主題。這可讓 Web 伺服器、電子郵件地址和其他訊息訂閱者回應 Greengrass 群組中的事件。

此連接器接收 MQTT 主題上的 SNS 訊息資訊，然後將訊息傳送到指定的 SNS 主題。您可以選擇使用自訂 Lambda 函數，在訊息發佈至此連接器之前，對訊息實作篩選或格式化邏輯。

此連接器具有下列版本。

版本	ARN
4	arn:aws:greengrass: <i>region</i> ::/connectors/SNS/versions/4
3	arn:aws:greengrass: <i>region</i> ::/connectors/SNS/versions/3
2	arn:aws:greengrass: <i>region</i> ::/connectors/SNS/versions/2
1	arn:aws:greengrass: <i>region</i> ::/connectors/SNS/versions/1

如需版本變更的詳細資訊，請參閱 [Changelog](#)。

要求

此連接器有下列要求：

Version 3 - 4

- AWS IoT Greengrass 核心軟體 v1.9.3 或更新版本。
- 安裝在核心裝置上並新增至 PATH 環境變數的 [Python](#) 3.7 或 3.8 版。

Note

若要使用 Python 3.8，請執行下列命令來建立從預設 Python 3.7 安裝資料夾到已安裝 Python 3.8 二進位檔的符號連結。

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

這會設定您的裝置以符合 AWS IoT Greengrass 的 Python 需求。

- 設定的 SNS 主題。如需詳細資訊，請參閱《Amazon Simple Notification Service 開發人員指南》中的 [建立 Amazon SNS 主題](#)。
- 設定為允許對目標 Amazon SNS topic sns:Publish 執行動作的 [Greengrass 群組角色](#)，如下列範例 IAM 政策所示。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1528133056761",
      "Action": [
        "sns:Publish"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:sns:us-east-1:123456789012:topic-name"
      ]
    }
  ]
}
```

此連接器可讓您動態覆寫輸入訊息承載中的預設主題。如果您的實作使用此功能，IAM 政策必須允許所有目標主題的 `sns:Publish` 許可。您可以為資源授予細微或條件式存取 (例如，使用萬用字元 * 命名配置)。

針對群組角色要求，您必須設定角色以授與必要的許可，並確認已將角色新增至群組。如需詳細資訊，請參閱 [the section called “管理群組角色 \(主控台\)”](#) 或 [the section called “管理群組角色 \(CLI\)”](#)。

Versions 1 - 2

- AWS IoT Greengrass 核心軟體 v1.7 或更新版本。
- 安裝在核心裝置上並新增至 PATH 環境變數的 [Python 2.7](#) 版。
- 設定的 SNS 主題。如需詳細資訊，請參閱《Amazon Simple Notification Service 開發人員指南》中的 [建立 Amazon SNS 主題](#)。
- 設定為允許對目標 Amazon SNS topic `sns:Publish` 執行動作的 [Greengrass 群組角色](#)，如下列範例 IAM 政策所示。

JSON

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

```
"Statement": [
  {
    "Sid": "Stmt1528133056761",
    "Action": [
      "sns:Publish"
    ],
    "Effect": "Allow",
    "Resource": [
      "arn:aws:sns:us-east-1:123456789012:topic-name"
    ]
  }
]
```

此連接器可讓您動態覆寫輸入訊息承載中的預設主題。如果您的實作使用此功能，IAM 政策必須允許所有目標主題的 `sns:Publish` 許可。您可以為資源授予細微或條件式存取 (例如，使用萬用字元 * 命名配置)。

針對群組角色要求，您必須設定角色以授與必要的許可，並確認已將角色新增至群組。如需詳細資訊，請參閱 [the section called “管理群組角色 \(主控台\)”](#) 或 [the section called “管理群組角色 \(CLI\)”](#)。

連接器參數

此連接器提供下列參數：

Version 4

DefaultSNSArn

訊息發佈到的預設 SNS 主題 ARN。輸入訊息承載中的 `sns_topic_arn` 屬性可以覆寫目的地主題。

Note

群組角色必須允許所有目標主題的 `sns:Publish` 許可。如需詳細資訊，請參閱[the section called “要求”](#)。

在 AWS IoT 主控台中顯示名稱：預設 SNS 主題 ARN

必要：true

類型：string

有效模式：arn:aws:sns:([a-z]{2}-[a-z]+\-\d{1}):(\d{12}):([a-zA-Z0-9-_\+]*)\$

IsolationMode

此連接器的[容器化](#)模式。預設值為 GreengrassContainer，這表示連接器會在 AWS IoT Greengrass 容器內的隔離執行時間環境中執行。

Note

群組的預設容器化設定不會套用至連接器。

AWS IoT 主控台中的顯示名稱：容器隔離模式

必要：false

類型：string

有效值：GreengrassContainer 或 NoContainer

有效模式：^NoContainer\$|^GreengrassContainer\$

Versions 1 - 3

DefaultSNSArn

訊息發佈到的預設 SNS 主題 ARN。輸入訊息承載中的 sns_topic_arn 屬性可以覆寫目的地主題。

Note

群組角色必須允許所有目標主題的 sns:Publish 許可。如需詳細資訊，請參閱[the section called “要求”](#)。

在 AWS IoT 主控台中顯示名稱：預設 SNS 主題 ARN

必要：true

類型：string

有效模式：arn:aws:sns:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):([a-zA-Z0-9-_\+]*)\$

建立範例連接器 (AWS CLI)

下列 CLI 命令ConnectorDefinition會使用包含 SNS 連接器的初始版本來建立。

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
  "Connectors": [
    {
      "Id": "MySNSConnector",
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/SNS/versions/4",
      "Parameters": {
        "DefaultSNSArn": "arn:aws:sns:region:account-id:topic-name",
        "IsolationMode" : "GreengrassContainer"
      }
    }
  ]
}'
```

在 AWS IoT Greengrass 主控台中，您可以從群組的連接器頁面新增連接器。如需詳細資訊，請參閱[the section called “連接器入門 \(主控台\)”](#)。

輸入資料

此連接器接受 MQTT 主題的 SNS 訊息資訊，然後將訊息依原樣發佈至目標 SNS 主題。輸入訊息必須是 JSON 格式。

訂閱中的主題篩選條件

sns/message

訊息屬性

request

要傳送到 SNS 主題之訊息的相關資訊。

必要：true

類型：object 包含下列屬性：

message

做為字串或 JSON 格式的訊息內容。如需範例，請參閱[範例輸入](#)。

若要傳送 JSON，message_structure 屬性必須設定為 json，並且訊息必須是字串編碼的 JSON 物件，其中包含 default 鍵。

必要：true

類型：string

有效模式：.*

subject

訊息的主旨。

必要：false

類型：ASCII 文字，最多 100 個字元。這必須以字母、數字或標點符號開頭。這不可包含換行或控制字元。

有效模式：.*

sns_topic_arn

將訊息發佈到其中的 SNS 主題的 ARN。如果指定，連接器會發佈到這個主題，而非預設主題。

 Note

群組角色必須允許任何目標主題的 sns:Publish 許可。如需詳細資訊，請參閱[the section called “要求”](#)。

必要：false

類型：string

有效模式：arn:aws:sns:([a-z]{2}-[a-z]+\d{1}):(\d{12}):([a-zA-Z0-9-_\+]*)\$

message_structure

訊息的結構。

必要：false。這必須指定於傳送 JSON 訊息。

類型：string

有效值：json

id

請求的任意 ID。此屬性用於將輸入請求映射到輸出回應。當指定時，回應物件中的 id 屬性會設為這個值。如果您不使用此功能，您可以省略此屬性或指定空白字串。

必要：false

類型：string

有效模式：.*

限制

訊息大小以 SNS 訊息大小上限 256 KB 為限。

範例輸入：字串訊息

此範例會傳送字串訊息。會指定選用 sns_topic_arn 屬性，此屬性會覆寫預設目的地主題。

```
{
  "request": {
    "subject": "Message subject",
    "message": "Message data",
    "sns_topic_arn": "arn:aws:sns:region:account-id:topic2-name"
  },
  "id": "request123"
}
```

範例輸入：JSON 訊息

此範例會傳送做為字串編碼 JSON 物件的訊息，其中包含 default 鍵。

```
{
  "request": {
    "subject": "Message subject",
    "message": "{ \"default\": \"Message data\" }",
  }
}
```



```
    "message_structure": "json"
  },
  "id": "request123"
}
```

輸出資料

這個連接器會將狀態資訊發佈為輸出資料，且主題為 MQTT。

訂閱中的主題篩選條件

sns/message/status

範例輸出：成功

```
{
  "response": {
    "sns_message_id": "f80a81bc-f44c-56f2-a0f0-d5af6a727c8a",
    "status": "success"
  },
  "id": "request123"
}
```

範例輸出：失敗

```
{
  "response" : {
    "error": "InvalidInputException",
    "error_message": "SNS Topic Arn is invalid",
    "status": "fail"
  },
  "id": "request123"
}
```

用量範例

使用下列高階步驟來設定範例 Python 3.7 Lambda 函數，您可以用來嘗試連接器。

Note

- 如果您使用其他 Python 執行期，則可以建立從 Python3.x 到 Python 3.7 的符號連結。

- [連接器入門 \(主控台\)](#) 和 [連接器入門 \(CLI\)](#) 主題包含詳細步驟，說明如何設定和部署範例 Twilio 通知連接器。

1. 確定您符合連接器的[要求](#)。

針對群組角色要求，您必須設定角色以授與必要的許可，並確認已將角色新增至群組。如需詳細資訊，請參閱 [the section called “管理群組角色 \(主控台\)”](#) 或 [the section called “管理群組角色 \(CLI\)”](#)。

2. 建立並發佈 Lambda 函數，將輸入資料傳送至連接器。

將範例程式碼儲存為 PY 檔案。下載並解壓縮[AWS IoT Greengrass 適用於 Python 的 Core SDK](#)。然後，建立在根層級包含 PY 檔案和 greengrasssdk 資料夾的 zip 套件。此 zip 套件是您上傳至的部署套件 AWS Lambda。

建立 Python 3.7 Lambda 函數後，發佈函數版本並建立別名。

3. 設定 Greengrass 群組。

- 依別名新增 Lambda 函數（建議）。將 Lambda 生命週期設定為長期（或在 CLI "Pinned": true 中）。
- 新增連接器並設定其[參數](#)。
- 新增訂閱，允許連接器在支援主題篩選條件上接收[輸入資料](#)並傳送[輸出資料](#)。
 - 將 Lambda 函數設定為來源，將連接器設定為目標，並使用支援的輸入主題篩選條件。
 - 將連接器設為來源、將 AWS IoT Core 設為目標，並使用支援的輸出主題篩選條件。您可以使用此訂閱在 AWS IoT 主控台中檢視狀態訊息。

4. 部署群組。

5. 在 AWS IoT 主控台的測試頁面上，訂閱輸出資料主題，以檢視連接器的狀態訊息。Lambda 函數範例為長期函數，並在部署群組後立即開始傳送訊息。

完成測試後，您可以將 Lambda 生命週期設定為隨需（或在 CLI "Pinned": false 中）並部署群組。這會讓函數停止傳送訊息。

範例

下列範例 Lambda 函數會將輸入訊息傳送至連接器。

```
import greengrasssdk
import time
import json

iot_client = greengrasssdk.client('iot-data')
send_topic = 'sns/message'

def create_request_with_all_fields():
    return {
        "request": {
            "message": "Message from SNS Connector Test"
        },
        "id" : "req_123"
    }

def publish_basic_message():
    messageToPublish = create_request_with_all_fields()
    print("Message To Publish: ", messageToPublish)
    iot_client.publish(topic=send_topic,
        payload=json.dumps(messageToPublish))

publish_basic_message()

def lambda_handler(event, context):
    return
```

授權

SNS 連接器包含下列第三方軟體/授權：

- [適用於 Python \(Boto3\) 的 AWS SDK](#)/Apache License 2.0
- [botocore](#)/Apache License 2.0
- [dateutil](#)/PSF License
- [docutils](#)/BSD 授權、GNU 一般公有授權 (GPL)、Python 軟體基金會授權、公有網域
- [jmespath](#)/MIT License
- [s3transfer](#)/Apache License 2.0
- [urllib3](#)/MIT License

此連接器根據 [Greengrass 核心軟體授權合約](#) 發行。

變更記錄

下表說明每個版本連接器的變更。

版本	變更
4	已新增 <code>IsolationMode</code> 參數，以設定連接器的容器化模式。
3	將 Lambda 執行時間升級至 Python 3.7，這會變更執行時間需求。
2	可減少過多記錄的修正。
1	初始版本。

Greengrass 群組一次只能包含一個版本的連接器。若要取得有關升級連接器版本的資訊，請參閱[the section called “升級連接器版本”](#)。

另請參閱

- [使用連接器整合服務和通訊協定](#)
- [the section called “連接器入門 \(主控台\)”](#)
- [the section called “連接器入門 \(CLI\)”](#)
- Boto 3 文件中的[發佈動作](#)
- 《Amazon Simple Notification Service 開發人員指南》中的[什麼是 Amazon Simple Notification Service ?](#)

Splunk 整合連接器

Warning

此連接器已進入延長的生命週期階段，AWS IoT Greengrass 且不會發佈提供現有功能、增強功能、安全修補程式或錯誤修正的更新。如需詳細資訊，請參閱[AWS IoT Greengrass Version 1 維護政策](#)。

Splunk Integration [連接器](#) 會將資料從 Greengrass 裝置發佈至 Splunk。這可讓您使用 Splunk 來監控和分析 Greengrass 核心環境，並對本機事件採取行動。連接器與 HTTP 事件收集器 (HEC) 整合。如需詳細資訊，請參閱 Splunk 文件中的 [Splunk HTTP 事件收集器簡介](#)。

此連接器會接收 MQTT 主題上的記錄和事件資料，並將資料依現狀發佈到 Splunk API。

您可以使用此連接器來支援工業案例，例如：

- 運算子可以使用來自傳動器和感應器的定期資料（例如溫度、壓力和水讀數），在值超過特定閾值時啟動警示。
- 開發人員使用從工業機械收集的資料，以建置機器學習模型來監控設備的潛在問題。

此連接器具有下列版本。

版本	ARN
4	arn:aws:greengrass: <i>region</i> ::/connectors/SplunkIntegration/versions/4
3	arn:aws:greengrass: <i>region</i> ::/connectors/SplunkIntegration/versions/3
2	arn:aws:greengrass: <i>region</i> ::/connectors/SplunkIntegration/versions/2
1	arn:aws:greengrass: <i>region</i> ::/connectors/SplunkIntegration/versions/1

如需版本變更的詳細資訊，請參閱 [Changelog](#)。

要求

此連接器有下列要求：

Version 3 - 4

- AWS IoT Greengrass 核心軟體 v1.9.3 或更新版本。AWS IoT Greengrass 必須設定為支援本機秘密，如[秘密要求](#)中所述。

 Note

此要求包括允許存取您的 Secrets Manager 秘密。如果您使用的是預設 Greengrass 服務角色，Greengrass 具有許可，可取得名稱開頭為 greengrass- 的秘密值。

- 安裝在核心裝置上並新增至 PATH 環境變數的 [Python](#) 3.7 或 3.8 版。

 Note

若要使用 Python 3.8，請執行下列命令，從預設 Python 3.7 安裝資料夾建立符號連結，以連接至已安裝的 Python 3.8 二進位檔。

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

這會設定您的裝置以符合 AWS IoT Greengrass 的 Python 需求。

- Splunk 中必須啟用 HTTP 事件收集器功能。如需詳細資訊，請參閱 Splunk 文件中的[在 Splunk Web 中設定和使用 HTTP 事件收集器](#)。
- 在中 AWS Secrets Manager 存放 Splunk HTTP 事件收集器字符的文字類型秘密。如需詳細資訊，請參閱 Splunk 文件中的[關於事件收集器權杖](#)和 AWS Secrets Manager 《使用者指南》中的[建立基本秘密](#)。

 Note

若要在 Secrets Manager 主控台中建立秘密，請在純文字索引標籤上輸入您的字符。請勿包含引號或其他格式。在 API 中，指定字符做為 SecretString 屬性的值。

- Greengrass 群組中參考 Secrets Manager 秘密的秘密資源。如需詳細資訊，請參閱[將私密部署至核心](#)。

Versions 1 - 2

- AWS IoT Greengrass 核心軟體 v1.7 或更新版本。AWS IoT Greengrass 必須設定為支援本機秘密，如[秘密要求](#)中所述。

Note

此要求包括允許存取您的 Secrets Manager 秘密。如果您使用的是預設 Greengrass 服務角色，Greengrass 具有許可，可取得名稱開頭為 greengrass- 的秘密值。

- [Python](#) 2.7 版安裝在核心裝置上，並新增至 PATH 環境變數。
- Splunk 中必須啟用 HTTP 事件收集器功能。如需詳細資訊，請參閱 Splunk 文件中的[在 Splunk Web 中設定和使用 HTTP 事件收集器](#)。
- 在中 AWS Secrets Manager 存放 Splunk HTTP 事件收集器字符的文字類型秘密。如需詳細資訊，請參閱 Splunk 文件中的[關於事件收集器權杖](#)和AWS Secrets Manager 《使用者指南》中的[建立基本秘密](#)。

Note

若要在 Secrets Manager 主控台中建立秘密，請在純文字索引標籤上輸入您的字符。請勿包含引號或其他格式。在 API 中，指定字符做為 SecretString 屬性的值。

- Greengrass 群組中參考 Secrets Manager 秘密的秘密資源。如需詳細資訊，請參閱[將私密部署至核心](#)。

連接器參數

此連接器提供下列參數：

Version 4

SplunkEndpoint

Splunk 執行個體的端點。這個值必須包含通訊協定、主機名稱和連接埠。

AWS IoT 主控台顯示名稱：Splunk 端點

必要：true

類型：string

有效模式：`^(http:\\\\|https:\\\\)?[a-z0-9]+([-.]?{1}[a-z0-9]+)*.[a-z]{2,5}(:[0-9]{1,5})?(\\/.*)?$`

MemorySize

配置給連接器的記憶體數量 (KB)。

AWS IoT 主控台顯示名稱：記憶體大小

必要：`true`

類型：`string`

有效模式：`^[0-9]+$`

SplunkQueueSize

提交或捨棄項目之前在記憶體中儲存的項目數上限。達到此限制時，佇列中最舊項目會由較新項目取代。此限制通常適用於未連接到網際網路時。

在 AWS IoT 主控台顯示名稱：要保留的項目上限

必要：`true`

類型：`string`

有效模式：`^[0-9]+$`

SplunkFlushIntervalSeconds

將收到的資料發佈到 Splunk HEC 的間隔 (以秒為單位)。最大值為 900。若要將連接器設定為一收到項目就立刻將其發佈 (不批次處理)，請指定 0。

AWS IoT 主控台顯示名稱：Splunk 發佈間隔

必要：`true`

類型：`string`

有效模式：`[0-9]|[1-9]\d|[1-9]\d\d|900`

SplunkTokenSecretArn

中 AWS Secrets Manager 存放 Splunk 字符的秘密。這必須是文字類型私密。

在 AWS IoT 主控台顯示名稱：Splunk 驗證字符秘密的 ARN

必要：`true`

類型：string

有效模式：arn:aws:secretsmanager:[a-z]{2}-[a-z]+-\d{1}:\d{12}?:secret:[a-zA-Z0-9-_\d]{1,64}

SplunkTokenSecretArn-ResourceId

Greengrass 群組中參考 Splunk 私密的私密資源。

AWS IoT 主控台顯示名稱：Splunk 驗證字符資源

必要：true

類型：string

有效模式：.+

SplunkCustomCALocation

Splunk 自訂憑證授權單位 (CA) 的檔案路徑 (例如 /etc/ssl/certs/splunk.crt)。

AWS IoT 主控台顯示名稱：Splunk 自訂憑證授權單位位置

必要：false

類型：string

有效模式：^\$|/.*

IsolationMode

此連接器的[容器化](#)模式。預設值為 GreengrassContainer，這表示連接器在 AWS IoT Greengrass 容器內的隔離執行時間環境中執行。

 Note

群組的預設容器化設定不會套用至連接器。

在 AWS IoT 主控台顯示名稱：容器隔離模式

必要：false

類型：string

有效值：GreengrassContainer 或 NoContainer

有效模式：`^NoContainer$|^GreengrassContainer$`

Version 1 - 3

SplunkEndpoint

Splunk 執行個體的端點。這個值必須包含通訊協定、主機名稱和連接埠。

AWS IoT 主控台顯示名稱：Splunk 端點

必要：true

類型：string

有效模式：`^(http:\\\\|https:\\\\)?[a-z0-9]+(\\.[a-z0-9]+)*\\. [a-z]{2,5}(:[0-9]{1,5})?(\\/.*)?$`

MemorySize

配置給連接器的記憶體數量 (KB)。

AWS IoT 主控台顯示名稱：記憶體大小

必要：true

類型：string

有效模式：`^[0-9]+$`

SplunkQueueSize

提交或捨棄項目之前在記憶體中儲存的項目數上限。達到此限制時，佇列中最舊項目會由較新項目取代。此限制通常適用於未連接到網際網路時。

在 AWS IoT 主控台顯示名稱：要保留的項目上限

必要：true

類型：string

有效模式：`^[0-9]+$`

SplunkFlushIntervalSeconds

將收到的資料發佈到 Splunk HEC 的間隔 (以秒為單位)。最大值為 900。若要將連接器設定為一收到項目就立刻將其發佈 (不批次處理)，請指定 0。

AWS IoT 主控台顯示名稱：Splunk 發佈間隔

必要：true

類型：string

有效模式：`[0-9]|[1-9]\d|[1-9]\d\d|900`

SplunkTokenSecretArn

中 AWS Secrets Manager 存放 Splunk 字符的秘密。這必須是文字類型私密。

在 AWS IoT 主控台中顯示名稱：Splunk 驗證字符秘密的 ARN

必要：true

類型：string

有效模式：`arn:aws:secretsmanager:[a-z]{2}-[a-z]+-\d{1}:\d{12}?:secret:[a-zA-Z0-9-_-]+-[a-zA-Z0-9-_-]+`

SplunkTokenSecretArn-ResourceId

Greengrass 群組中參考 Splunk 私密的私密資源。

AWS IoT 主控台顯示名稱：Splunk 驗證字符資源

必要：true

類型：string

有效模式：`.+`

SplunkCustomCALocation

Splunk 自訂憑證授權單位 (CA) 的檔案路徑 (例如 `/etc/ssl/certs/splunk.crt`)。

AWS IoT 主控台顯示名稱：Splunk 自訂憑證授權單位位置

必要：false

類型：string

有效模式：`^$|/.*`

建立範例連接器 (AWS CLI)

下列 CLI 命令ConnectorDefinition會使用包含 Splunk 整合連接器的初始版本來建立。

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-  
version '{  
  "Connectors": [  
    {  
      "Id": "MySplunkIntegrationConnector",  
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/SplunkIntegration/  
versions/4",  
      "Parameters": {  
        "SplunkEndpoint": "https://myinstance.cloud.splunk.com:8088",  
        "MemorySize": 200000,  
        "SplunkQueueSize": 10000,  
        "SplunkFlushIntervalSeconds": 5,  
        "SplunkTokenSecretArn": "arn:aws:secretsmanager:region:account-  
id:secret:greengrass-secret-hash",  
        "SplunkTokenSecretArn-ResourceId": "MySplunkResource",  
        "IsolationMode" : "GreengrassContainer"  
      }  
    }  
  ]  
}'
```

Note

此連接器中的 Lambda 函數具有[長期生命週期](#)。

在 AWS IoT Greengrass 主控台中，您可以從群組的連接器頁面新增連接器。如需詳細資訊，請參閱[the section called “連接器入門 \(主控台\)”](#)。

輸入資料

此連接器接受 MQTT 主題上的記錄和事件資料，並將收到的資料依原樣發佈至 Splunk API。輸入訊息必須是 JSON 格式。

訂閱中的主題篩選條件

splunk/logs/put

訊息屬性

request

要傳送到 Splunk API 的事件資料。事件必須符合 [服務/收集器](#) API 的規格。

必要：true

類型：object。只需要 event 屬性。

id

請求的任意 ID。此屬性用於將輸入請求映射到輸出狀態。

必要：false

類型：string

限制

Splunk API 實施的所有限制，在使用此連接器時都適用。如需詳細資訊，請參閱[服務/收集器](#)。

範例輸入

```
{
  "request": {
    "event": "some event",
    "fields": {
      "severity": "INFO",
      "category": [
        "value1",
        "value2"
      ]
    }
  },
  "id": "request123"
}
```

輸出資料

此連接器將輸出資料發佈在兩個主題：

- splunk/logs/put/status 主題上的狀態資訊。
- splunk/logs/put/error 主題的相關錯誤。

主題篩選條件：splunk/logs/put/status

使用此主題監聽請求的狀態。每次連接器將一批收到的資料傳送到 Splunk API 時，就會發佈成功請求和失敗請求的 ID 清單。

範例輸出

```
{
  "response": {
    "succeeded": [
      "request123",
      ...
    ],
    "failed": [
      "request789",
      ...
    ]
  }
}
```

主題篩選條件：splunk/logs/put/error

使用此主題監聽來自連接器的錯誤。error_message 屬性，描述處理請求時發生的錯誤或逾時。

範例輸出

```
{
  "response": {
    "error": "UnauthorizedException",
    "error_message": "invalid splunk token",
    "status": "fail"
  }
}
```

Note

如果連接器偵測到可重試的錯誤（例如連線錯誤），則會在下一個批次中重試發佈。

使用範例

使用下列高階步驟來設定範例 Python 3.7 Lambda 函數，您可以用來嘗試連接器。

 Note

- 如果您使用其他 Python 執行期，則可以建立從 Python3.x 到 Python 3.7 的符號連結。
- [連接器入門 \(主控台\)](#) 和 [連接器入門 \(CLI\)](#) 主題包含詳細步驟，說明如何設定和部署範例 Twilio 通知連接器。

1. 確定您符合連接器的[要求](#)。
2. 建立並發佈 Lambda 函數，將輸入資料傳送至連接器。

將範例程式碼儲存為 PY 檔案。下載並解壓縮[AWS IoT Greengrass 適用於 Python 的核心 SDK](#)。然後，建立在根層級包含 PY 檔案和 greengrasssdk 資料夾的 zip 套件。此 zip 套件是您上傳至的部署套件 AWS Lambda。

建立 Python 3.7 Lambda 函數之後，請發佈函數版本並建立別名。

3. 設定 Greengrass 群組。
 - a. 依別名新增 Lambda 函數（建議）。將 Lambda 生命週期設定為長期（或在 CLI "Pinned": true 中）。
 - b. 新增必要的秘密資源，並授予 Lambda 函數的讀取存取權。
 - c. 新增連接器並設定其[參數](#)。
 - d. 新增訂閱，允許連接器在支援主題篩選條件上接收[輸入資料](#)並傳送[輸出資料](#)。
 - 將 Lambda 函數設定為來源，將連接器設定為目標，並使用支援的輸入主題篩選條件。
 - 將連接器設為來源、將 AWS IoT Core 設為目標，並使用支援的輸出主題篩選條件。您可以使用此訂閱在 AWS IoT 主控台中檢視狀態訊息。
4. 部署群組。
5. 在 AWS IoT 主控台的測試頁面上，訂閱輸出資料主題，以檢視連接器的狀態訊息。Lambda 函數範例為長期函數，並在部署群組後立即開始傳送訊息。

完成測試後，您可以將 Lambda 生命週期設定為隨需（或在 CLI "Pinned": false 中）並部署群組。這會讓函數停止傳送訊息。

範例

下列範例 Lambda 函數會將輸入訊息傳送至連接器。

```
import greengrasssdk
import time
import json

iot_client = greengrasssdk.client('iot-data')
send_topic = 'splunk/logs/put'

def create_request_with_all_fields():
    return {
        "request": {
            "event": "Access log test message."
        },
        "id" : "req_123"
    }

def publish_basic_message():
    messageToPublish = create_request_with_all_fields()
    print("Message To Publish: ", messageToPublish)
    iot_client.publish(topic=send_topic,
        payload=json.dumps(messageToPublish))

publish_basic_message()

def lambda_handler(event, context):
    return
```

授權

此連接器根據 [Greengrass 核心軟體授權合約](#) 發行。

變更記錄

下表說明每個連接器版本中的變更。

版本	變更
4	已新增 <code>IsolationMode</code> 參數，以設定連接器的容器化模式。
3	將 Lambda 執行時間升級至 Python 3.7，這會變更執行時間需求。

版本	變更
2	可減少過多記錄的修正。
1	初始版本。

Greengrass 群組一次只能包含一個版本的連接器。若要取得有關升級連接器版本的資訊，請參閱[the section called “升級連接器版本”](#)。

另請參閱

- [使用連接器整合服務和通訊協定](#)
- [the section called “連接器入門 \(主控台\)”](#)
- [the section called “連接器入門 \(CLI\)”](#)

Twilio 通知連接器

Warning

此連接器已進入延長的生命週期階段，AWS IoT Greengrass 且不會發佈提供現有功能、增強功能、安全修補程式或錯誤修正的更新。如需詳細資訊，請參閱[AWS IoT Greengrass Version 1 維護政策](#)。

Twilio Notifications [連接器](#)會透過 Twilio 進行自動電話呼叫或傳送文字訊息。您可以使用此連接器傳送通知，以回應 Greengrass 群組中的事件。對於電話呼叫，連接器可以將語音訊息轉發給收件人。

此連接器會接收 MQTT 主題上的 Twilio 訊息，然後觸發 Twilio 通知。

Note

如需示範如何使用 Twilio Notifications 連接器的教學課程，請參閱 [the section called “連接器入門 \(主控台\)”](#)或 [the section called “連接器入門 \(CLI\)”](#)。

此連接器具有下列版本。

版本	ARN
5	arn:aws:greengrass: <i>region</i> ::/connectors/TwilioNotifications/versions/5
4	arn:aws:greengrass: <i>region</i> ::/connectors/TwilioNotifications/versions/4
3	arn:aws:greengrass: <i>region</i> ::/connectors/TwilioNotifications/versions/3
2	arn:aws:greengrass: <i>region</i> ::/connectors/TwilioNotifications/versions/2
1	arn:aws:greengrass: <i>region</i> ::/connectors/TwilioNotifications/versions/1

如需版本變更的詳細資訊，請參閱 [Changelog](#)。

要求

此連接器有下列要求：

Version 4 - 5

- AWS IoT Greengrass 核心軟體 v1.9.3 或更新版本。AWS IoT Greengrass 必須設定為支援本機秘密，如[秘密要求](#)中所述。

Note

此要求包括允許存取您的 Secrets Manager 秘密。如果您使用的是預設 Greengrass 服務角色，Greengrass 具有許可，可取得名稱開頭為 greengrass- 的秘密值。

- 安裝在核心裝置上並新增至 PATH 環境變數的 [Python](#) 3.7 或 3.8 版。

 Note

若要使用 Python 3.8，請執行下列命令，從預設 Python 3.7 安裝資料夾建立符號連結，以連接至已安裝的 Python 3.8 二進位檔。

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

這會設定您的裝置以符合 AWS IoT Greengrass 的 Python 需求。

- Twilio 帳戶 SID、驗證字符和已啟用 Twilio 的電話號碼。在您建立 Twilio 專案後，您就可在專案儀表板上使用這些值。

 Note

您可以使用 Twilio 試用帳戶。如果您使用的是試驗帳戶，則必須將非 Twilio 收件人的電話號碼新增至已驗證的電話號碼清單。如需詳細資訊，請參閱[如何使用您的免費 Twilio 試用帳戶](#)。

- 在 AWS Secrets Manager 中存放 Twilio 驗證字符的文字類型秘密。如需詳細資訊，請參閱 AWS Secrets Manager 《使用者指南》中的[建立基本秘密](#)。

 Note

若要在 Secrets Manager 主控台中建立秘密，請在純文字索引標籤上輸入您的字符。請勿包含引號或其他格式。在 API 中，指定字符做為 SecretString 屬性的值。

- Greengrass 群組中參考 Secrets Manager 秘密的秘密資源。如需詳細資訊，請參閱[將私密部署至核心](#)。

Versions 1 - 3

- AWS IoT Greengrass 核心軟體 v1.7 或更新版本。AWS IoT Greengrass 必須設定為支援本機秘密，如[秘密要求](#)中所述。

Note

此要求包括允許存取您的 Secrets Manager 秘密。如果您使用的是預設 Greengrass 服務角色，Greengrass 具有許可，可取得名稱開頭為 greengrass- 的秘密值。

- 安裝在核心裝置上並新增至 PATH 環境變數的 [Python 2.7](#) 版。
- Twilio 帳戶 SID、驗證字符和已啟用 Twilio 的電話號碼。在您建立 Twilio 專案後，您就可在專案儀表板上使用這些值。

Note

您可以使用 Twilio 試用帳戶。如果您使用的是試驗帳戶，則必須將非 Twilio 收件人的電話號碼新增至已驗證的電話號碼清單。如需詳細資訊，請參閱[如何使用您的免費 Twilio 試用帳戶](#)。

- 在 AWS Secrets Manager 存放 Twilio 驗證字符的文字類型秘密。如需詳細資訊，請參閱AWS Secrets Manager 《使用者指南》中的[建立基本秘密](#)。

Note

若要在 Secrets Manager 主控台中建立秘密，請在純文字索引標籤上輸入您的字符。請勿包含引號或其他格式。在 API 中，指定字符做為 SecretString 屬性的值。

- Greengrass 群組中參考 Secrets Manager 秘密的秘密資源。如需詳細資訊，請參閱[將私密部署至核心](#)。

連接器參數

此連接器提供下列參數。

Version 5

TWILIO_ACCOUNT_SID

用於叫用 Twilio API 的 Twilio 帳戶 SID。

AWS IoT 主控台顯示名稱：Twilio 帳戶 SID

必要：true

類型：string

有效模式：.+

TwilioAuthTokenSecretArn

存放 Twilio 驗證字符的 Secrets Manager 秘密的 ARN。

 Note

這用於存取核心上的本機私密值。

AWS IoT 主控台中的顯示名稱：Twilio 驗證字符秘密的 ARN

必要：true

類型：string

有效模式：arn:aws:secretsmanager:[a-z0-9\-\+]:[0-9]{12}:secret:([a-zA-Z0-9\-\+\/*\[\]\+\-])*\[a-zA-Z0-9/_+=,.\@-\]+\-[a-zA-Z0-9]+\

TwilioAuthTokenSecretArn-ResourceId

Greengrass 群組中參考 Twilio 驗證字符私密的私密資源的 ID。

AWS IoT 主控台中的顯示名稱：Twilio 驗證字符資源

必要：true

類型：string

有效模式：.+

DefaultFromPhoneNumber

預設已啟用 Twilio 的電話號碼，供 Twilio 用來傳送訊息。Twilio 使用此號碼來起始文字或通話。

- 如果您不設定預設電話號碼，則必須在輸入訊息本文的 from_number 屬性中指定電話號碼。
- 如果您設定預設電話號碼，您可以選擇在輸入訊息本文中指定 from_number 屬性，以覆寫預設值。

AWS IoT 主控台中的顯示名稱：電話號碼的預設

必要：false

類型：string

有效模式：`^\$|\+[0-9]+`

IsolationMode

此連接器的[容器化](#)模式。預設值為 `GreengrassContainer`，這表示連接器在 AWS IoT Greengrass 容器內的隔離執行時間環境中執行。

 Note

群組的預設容器化設定不會套用至連接器。

AWS IoT 主控台中的顯示名稱：容器隔離模式

必要：false

類型：string

有效值：`GreengrassContainer` 或 `NoContainer`

有效模式：`^NoContainer$|^GreengrassContainer$`

Version 1 - 4

TWILIO_ACCOUNT_SID

用於叫用 Twilio API 的 Twilio 帳戶 SID。

AWS IoT 主控台中的顯示名稱：Twilio 帳戶 SID

必要：true

類型：string

有效模式：`.+`

TwilioAuthTokenSecretArn

存放 Twilio 驗證字符的 Secrets Manager 秘密的 ARN。

Note

這用於存取核心上的本機私密值。

AWS IoT 主控台中的顯示名稱：Twilio 驗證字符秘密的 ARN

必要：true

類型：string

有效模式：arn:aws:secretsmanager:[a-z0-9\-\+]:[0-9]{12}:secret:([a-zA-Z0-9\-\+\/]*[a-zA-Z0-9/_+=,.\@-\+]-[a-zA-Z0-9]+)

TwilioAuthTokenSecretArn-ResourceId

Greengrass 群組中參考 Twilio 驗證字符私密的私密資源的 ID。

AWS IoT 主控台中的顯示名稱：Twilio 驗證字符資源

必要：true

類型：string

有效模式：.+

DefaultFromPhoneNumber

預設已啟用 Twilio 的電話號碼，供 Twilio 用來傳送訊息。Twilio 使用此號碼來起始文字或通話。

- 如果您不設定預設電話號碼，則必須在輸入訊息本文的 `from_number` 屬性中指定電話號碼。
- 如果您設定預設電話號碼，您可以選擇在輸入訊息本文中指定 `from_number` 屬性，以覆寫預設值。

AWS IoT 主控台中的顯示名稱：電話號碼的預設

必要：false

類型：string

有效模式：`^$|\+[0-9]+`

建立範例連接器 (AWS CLI)

下列範例 CLI 命令 `ConnectorDefinition` 會使用包含 Twilio Notifications 連接器的初始版本來建立。

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-version '{
  "Connectors": [
    {
      "Id": "MyTwilioNotificationsConnector",
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/TwilioNotifications/versions/5",
      "Parameters": {
        "TWILIO_ACCOUNT_SID": "abcd12345xyz",
        "TwilioAuthTokenSecretArn": "arn:aws:secretsmanager:region:account-id:secret:greengrass-secret-hash",
        "TwilioAuthTokenSecretArn-ResourceId": "MyTwilioSecret",
        "DefaultFromPhoneNumber": "+19999999999",
        "IsolationMode" : "GreengrassContainer"
      }
    }
  ]
}'
```

如需示範如何將 Twilio Notifications 連接器新增至群組的教學課程，請參閱 [the section called “連接器入門 \(CLI\)”](#) 和 [the section called “連接器入門 \(主控台\)”](#)。

輸入資料

此連接器接受兩個 MQTT 主題的 Twilio 訊息資訊。輸入訊息必須是 JSON 格式。

- `twilio/txt` 主題上的文字訊息資訊。
- `twilio/call` 主題上的電話訊息資訊。

Note

輸入訊息承載可包含文字訊息 (message) 或語音訊息 (voice_message_location)，但非同時包含兩者。

主題篩選條件：twilio/txt**訊息屬性****request**

Twilio 通知的相關資訊。

必要：true

類型：object 包含下列屬性：

recipient

訊息收件人。僅支援一個收件人。

必要：true

類型：object 包含下列屬性：

name

收件人的名稱。

必要：true

類型：string

有效模式：.*

phone_number

收件人的電話號碼。

必要：true

類型：string

有效模式：\+[1-9]+

message

文字訊息的文字內容。這個主題上僅支援文字訊息。對於語音訊息，請使用 `twilio/call`。

必要：true

類型：string

有效模式：.+

from_number

寄件者的電話號碼。Twilio 使用此電話號碼來起始訊息。如果未設定 DefaultFromPhoneNumber 參數，則此為必要屬性。如果設定 DefaultFromPhoneNumber，您可以使用此屬性來覆寫預設值。

必要：false

類型：string

有效模式：\+[1-9]+

retries

重試次數。預設值為 0。

必要：false

類型：integer

id

請求的任意 ID。此屬性用於將輸入請求映射到輸出回應。

必要：true

類型：string

有效模式：.+

範例輸入

```
{
  "request": {
    "recipient": {
      "name": "Darla",
      "phone_number": "+12345000000",
      "message": "Hello from the edge"
    },
    "from_number": "+19999999999",
    "retries": 3
  },
}
```

```
"id": "request123"
}
```

主題篩選條件：**twilio/call**

訊息屬性

request

Twilio 通知的相關資訊。

必要：true

類型：object 包含下列屬性：

recipient

訊息收件人。僅支援一個收件人。

必要：true

類型：object 包含下列屬性：

name

收件人的名稱。

必要：true

類型：string

有效模式：.+

phone_number

收件人的電話號碼。

必要：true

類型：string

有效模式：\+[1-9]+

voice_message_location

語音訊息的音訊內容 URL。這必須是 TwiML 格式。這個主題上僅支援語音訊息。對於文字訊息，請使用 `twilio/txt`。

必要：true

類型：string

有效模式：.+

from_number

寄件者的電話號碼。Twilio 使用此電話號碼來起始訊息。如果未設定 DefaultFromPhoneNumber 參數，則此為必要屬性。如果設定 DefaultFromPhoneNumber，您可以使用此屬性來覆寫預設值。

必要：false

類型：string

有效模式：\+[1-9]+

retries

重試次數。預設值為 0。

必要：false

類型：integer

id

請求的任意 ID。此屬性用於將輸入請求映射到輸出回應。

必要：true

類型：string

有效模式：.+

範例輸入

```
{
  "request": {
    "recipient": {
      "name": "Darla",
      "phone_number": "+12345000000",
      "voice_message_location": "https://some-public-TwiML"
    },
```

```
    "from_number": "+19999999999",
    "retries": 3
  },
  "id": "request123"
}
```

輸出資料

這個連接器會將狀態資訊發佈為輸出資料，且主題為 MQTT。

訂閱中的主題篩選條件

twilio/message/status

範例輸出：成功

```
{
  "response": {
    "status": "success",
    "payload": {
      "from_number": "+19999999999",
      "messages": {
        "message_status": "queued",
        "to_number": "+12345000000",
        "name": "Darla"
      }
    }
  },
  "id": "request123"
}
```

範例輸出：失敗

```
{
  "response": {
    "status": "fail",
    "error_message": "Recipient name cannot be None",
    "error": "InvalidParameter",
    "payload": None
  },
  "id": "request123"
}
```

```
}
```

輸出的 payload 屬性是傳送訊息時來自 Twilio API 的回應。如果連接器偵測到輸入資料無效 (例如，未指定必要的輸入欄位)，連接器會傳回錯誤，並將值設為 None。下列為範例承載：

```
{
  'from_number': '+19999999999',
  'messages': {
    'name': 'Darla',
    'to_number': '+12345000000',
    'message_status': 'undelivered'
  }
}
```

```
{
  'from_number': '+19999999999',
  'messages': {
    'name': 'Darla',
    'to_number': '+12345000000',
    'message_status': 'queued'
  }
}
```

使用範例

使用下列高階步驟來設定範例 Python 3.7 Lambda 函數，您可以用來嘗試連接器。

Note

[the section called “連接器入門 \(主控台\)”](#) 和 [the section called “連接器入門 \(CLI\)”](#) 主題包含端對端步驟，說明如何設定、部署和測試 Twilio 通知連接器。

1. 確定您符合連接器的[要求](#)。
2. 建立並發佈 Lambda 函數，將輸入資料傳送至連接器。

將範例程式碼儲存為 PY 檔案。下載並解壓縮[AWS IoT Greengrass 適用於 Python 的核心 SDK](#)。然後，建立在根層級包含 PY 檔案和 greengrasssdk 資料夾的 zip 套件。此 zip 套件是您上傳至的部署套件 AWS Lambda。

建立 Python 3.7 Lambda 函數後，發佈函數版本並建立別名。

3. 設定 Greengrass 群組。

- a. 依別名新增 Lambda 函數（建議）。將 Lambda 生命週期設定為長期（或在 CLI "Pinned": true 中）。
- b. 新增必要的秘密資源，並授予 Lambda 函數的讀取存取權。
- c. 新增連接器並設定其參數。
- d. 新增訂閱，允許連接器在支援主題篩選條件上接收[輸入資料](#)並傳送[輸出資料](#)。
 - 將 Lambda 函數設定為來源，將連接器設定為目標，並使用支援的輸入主題篩選條件。
 - 將連接器設為來源、將 AWS IoT Core 設為目標，並使用支援的輸出主題篩選條件。您可以使用此訂閱在 AWS IoT 主控台中檢視狀態訊息。

4. 部署群組。

5. 在 AWS IoT 主控台的測試頁面上，訂閱輸出資料主題，以檢視連接器的狀態訊息。Lambda 函數範例為長期函數，並在部署群組後立即開始傳送訊息。

完成測試後，您可以將 Lambda 生命週期設定為隨需（或在 CLI "Pinned": false 中）並部署群組。這會讓函數停止傳送訊息。

範例

下列範例 Lambda 函數會將輸入訊息傳送至連接器。此範例會觸發文字訊息。

```
import greengrasssdk
import json

iot_client = greengrasssdk.client('iot-data')
TXT_INPUT_TOPIC = 'twilio/txt'
CALL_INPUT_TOPIC = 'twilio/call'

def publish_basic_message():

    txt = {
        "request": {
            "recipient": {
                "name": "Darla",
                "phone_number": "+12345000000",
                "message": 'Hello from the edge'
```

```

        },
        "from_number" : "+19999999999"
    },
    "id" : "request123"
}

print("Message To Publish: ", txt)

client.publish(topic=TXT_INPUT_TOPIC,
               payload=json.dumps(txt))

publish_basic_message()

def lambda_handler(event, context):
    return

```

授權

Twilio Notifications 連接器包含下列第三方軟體/授權：

- [twilio-python](#)/MIT

此連接器根據 [Greengrass 核心軟體授權合約](#) 發行。

變更記錄

下表說明連接器每個版本中的變更。

版本	變更
5	已新增 IsolationMode 參數，以設定連接器的容器化模式。
4	將 Lambda 執行時間升級至 Python 3.7，這會變更執行時間需求。
3	可減少過多記錄的修正。
2	少量错误修复和改进。
1	初始版本。

Greengrass 群組一次只能包含一個版本的連接器。若要取得有關升級連接器版本的資訊，請參閱[the section called “升級連接器版本”](#)。

另請參閱

- [使用連接器整合服務和通訊協定](#)
- [the section called “連接器入門 \(主控台\)”](#)
- [the section called “連接器入門 \(CLI\)”](#)
- [Twilio API 參考](#)

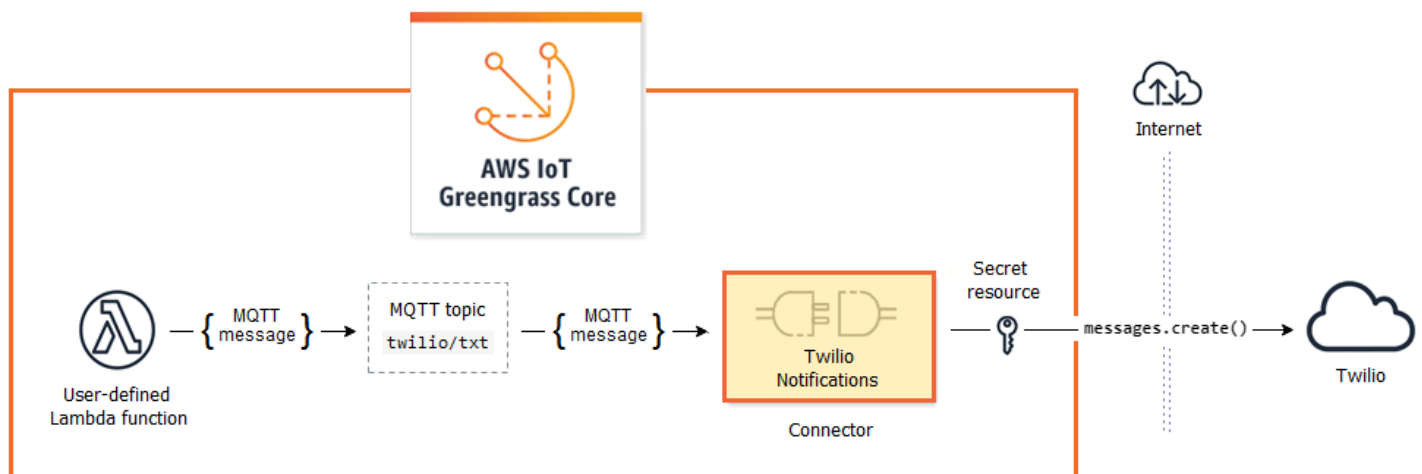
Greengrass 連接器入門 (主控台)

此功能適用於 AWS IoT Greengrass Core v1.7 和更新版本。

本教學課程說明如何使用 AWS 管理主控台 來使用連接器。

使用連接器來加速您的開發生命週期。連接器是預先建置的、可重複使用的模組，可讓您更輕鬆地與服務、通訊協定和資源互動。它們可協助您更快地將商業邏輯部署到 Greengrass 裝置。如需詳細資訊，請參閱[使用連接器整合服務和通訊協定](#)。

在本教學課程中，您會設定和部署 [Twilio Notifications](#) 連接器。連接器接收 Twilio 訊息資訊做為輸入資料，然後觸發 Twilio 文字訊息。下圖顯示資料流程。



設定連接器之後，您可以建立 Lambda 函數和訂閱。

- 此函數會評估來自溫度感應器的模擬資料。它會有條件地將 Twilio 訊息資訊發佈到 MQTT 主題。這是連接器訂閱的主題。
- 該訂閱允許將函數發佈到主題，也允許連接器從主題接收資料。

Twilio Notifications 連接器需要 Twilio 驗證字符才能與 Twilio API 互動。字符是在 中建立 AWS Secrets Manager 並從群組資源參考的文字類型秘密。這 AWS IoT Greengrass 可讓 在 Greengrass 核心上建立秘密的本機副本，並在其中加密並提供給連接器。如需詳細資訊，請參閱[將私密部署至核心](#)。

本教學課程所述以下高階執行步驟：

1. [建立 Secrets Manager 秘密](#)
2. [將私密資源新增至群組](#)
3. [將連接器新增到群組](#)
4. [建立 Lambda 函數部署套件](#)
5. [建立 Lambda 函式](#)
6. [將函數新增到群組](#)
7. [新增訂閱到群組](#)
8. [部署群組](#)
9. [the section called “測試解決方案”](#)

此教學課程需約 20 分鐘完成。

先決條件

為完成此教學課程您需要：

- Greengrass 群組和 Greengrass 核心 (1.9.3 版或更新版本)。若要了解如何建立 Greengrass 群組或核心，請參閱 [入門 AWS IoT Greengrass](#)。入門教學課程也包含 AWS IoT Greengrass Core 軟體的安裝步驟。
- AWS IoT Greengrass 核心裝置上已安裝 Python 3.7。
- AWS IoT Greengrass 必須設定為支援本機秘密，如[秘密要求](#)中所述。

Note

此要求包括允許存取您的 Secrets Manager 秘密。如果您使用預設 Greengrass 服務角色，Greengrass 有權取得名稱開頭為 greengrass- 的秘密值。

- Twilio 帳戶 SID、驗證字符和已啟用 Twilio 的電話號碼。在您建立 Twilio 專案後，您就可在專案儀表板上使用這些值。

Note

您可以使用 Twilio 試用帳戶。如果您使用試驗帳戶，則必須將非 Twilio 收件人的電話號碼新增至已驗證的電話號碼清單。如需詳細資訊，請參閱[如何使用您的 免費 Twilio 試用帳戶](#)。

步驟 1：建立 Secrets Manager 秘密

在此步驟中，您可以使用 AWS Secrets Manager 主控台為 Twilio 驗證字符建立文字類型秘密。

1. 登入 [AWS Secrets Manager 主控台](#)。

Note

如需此程序的詳細資訊，請參閱 AWS Secrets Manager 《使用者指南》中的[步驟 1：建立和存放您的秘密 AWS Secrets Manager](#)。

2. 選擇儲存新機密。
3. 在選擇秘密類型下，選擇其他類型的秘密。
4. 在指定要為此秘密存放的金鑰/值對下，在純文字索引標籤上，輸入您的 Twilio 驗證字符。移除所有的 JSON 格式，只輸入字符值。
5. 保持選取加密金鑰的 aws/secretsmanager，然後選擇下一步。

Note

AWS KMS 如果您使用 Secrets Manager 在帳戶中建立的預設 AWS 受管金鑰，則不會向您收費。

- 針對 Secret name (私密名稱)，輸入 **greengrass-TwilioAuthToken**，然後選擇 Next (下一步)。

 Note

根據預設，Greengrass 服務角色允許 AWS IoT Greengrass 取得名稱開頭為 greengrass- 的秘密值。如需詳細資訊，請參閱[私密需求](#)。

- 本教學課程不需要輪換，因此請選擇停用自動輪換，然後選擇下一步。
- 在 Review (檢閱) 頁面上，檢閱設定，然後選擇 Store (儲存)。

接著，您可以在 Greengrass 群組中建立一個參考私密的私密資源。

步驟 2：將私密資源新增至 Greengrass 群組

在此步驟中，您會將私密資源新增到 Greengrass 群組。此資源會參考您在上一步中建立的私密。

- 在 AWS IoT 主控台導覽窗格的管理下，展開 Greengrass 裝置，然後選擇群組 (V1)。
- 選擇您要將私密資源新增至其中的群組。
- 在群組組態頁面上，選擇資源索引標籤，然後向下捲動至秘密區段。秘密區段會顯示屬於群組的秘密資源。您可以從本節新增、編輯和移除秘密資源。

 Note

或者，主控台可讓您在設定連接器或 Lambda 函數時建立秘密和秘密資源。您可以從連接器的設定參數頁面或 Lambda 函數的資源頁面執行此操作。

- 在秘密區段下選擇新增。
- 在新增秘密資源頁面上，輸入 **MyTwilioAuthToken** 做為資源名稱。
- 針對秘密，選擇 greengrass-TwilioAuthToken。
- 在選取標籤 (選用) 區段中，AWSCURRENT 預備標籤代表秘密的最新版本。此標籤一律包含在私密資源中。

 Note

本教學課程僅需要 AWSCURRENT 標籤。您可以選擇性地包含 Lambda 函數或連接器所需的標籤。

8. 選擇 Add resource (新增資源)。

步驟 3：將連接器新增到 Greengrass 群組

在此步驟中，您會設定 [Twilio Notifications 連接器](#) 的參數，並將其新增至群組。

1. 在群組組態頁面上，選擇 Connectors (連接器)，然後選擇 Add a connector (新增連接器)。
2. 在新增連接器頁面上，選擇 Twilio Notifications。
3. 選擇 版本。
4. 在組態區段中：
 - 對於 Twilio 驗證字符資源，輸入您在上一個步驟中建立的資源。

 Note

當您輸入 資源時，會為您填入 Twilio 驗證字符秘密屬性的 ARN。

- 針對 Default from phone number (預設從電話號碼)，輸入已啟用 Twilio 的電話號碼。
 - 針對 Twilio account SID (Twilio 帳戶 SID)，輸入您的 Twilio 帳戶 SID。
5. 選擇 Add resource (新增資源)。

步驟 4：建立 Lambda 函數部署套件

若要建立 Lambda 函數，您必須先建立包含函數程式碼和相依性的 Lambda 函數部署套件。Greengrass Lambda 函數需要 [AWS IoT Greengrass 核心 SDK](#) 來處理任務，例如與核心環境中的 MQTT 訊息通訊，以及存取本機秘密。本教學課程會建立 Python 函數，因此您可以在部署套件中使用 SDK 的 Python 版本。

1. 從[AWS IoT Greengrass 核心開發套件](#)下載頁面，將適用於 Python AWS IoT Greengrass 的核心開發套件下載至您的電腦。
2. 解壓縮下載的封裝，以取得軟體開發套件。SDK 為 greengrasssdk 資料夾。

3. 將以下 Python 程式碼函數儲存在名為 temp_monitor.py 的本機檔案中。

```
import greengrasssdk
import json
import random

client = greengrasssdk.client('iot-data')

# publish to the Twilio Notifications connector through the twilio/txt topic
def function_handler(event, context):
    temp = event['temperature']

    # check the temperature
    # if greater than 30C, send a notification
    if temp > 30:
        data = build_request(event)
        client.publish(topic='twilio/txt', payload=json.dumps(data))
        print('published:' + str(data))

    print('temperature:' + str(temp))
    return

# build the Twilio request from the input data
def build_request(event):
    to_name = event['to_name']
    to_number = event['to_number']
    temp_report = 'temperature:' + str(event['temperature'])

    return {
        "request": {
            "recipient": {
                "name": to_name,
                "phone_number": to_number,
                "message": temp_report
            }
        },
        "id": "request_" + str(random.randint(1,101))
    }
```

4. 將下列項目壓縮成名為 temp_monitor_python.zip 的檔案。建立 ZIP 檔案時，只包含程式碼及其依存項目，而不包含資料夾。

- temp_monitor.py。應用程式邏輯。

- greengrasssdk。發佈 MQTT 訊息的 Python Greengrass Lambda 函數所需的程式庫。

這是您的 Lambda 函數部署套件。

現在，建立使用部署套件的 Lambda 函數。

步驟 5：在 AWS Lambda 主控台中建立 Lambda 函數

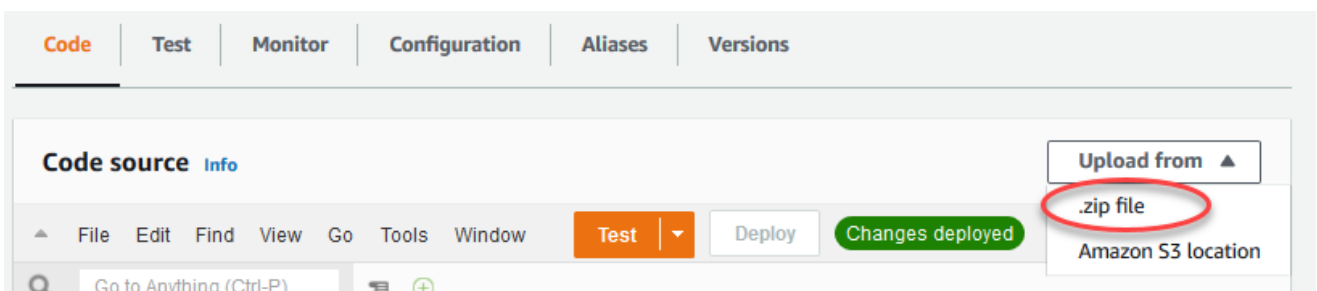
在此步驟中，您會使用 AWS Lambda 主控台來建立 Lambda 函數，並將其設定為使用您的部署套件。然後，您會發佈函數版本和建立別名。

1. 首先，建立 Lambda 函數。

- 在中 AWS 管理主控台，選擇服務，然後開啟 AWS Lambda 主控台。
- 選擇建立函數，然後選擇從頭開始撰寫。
- 在 Basic information (基本資訊) 區段中，使用下列值：
 - 針對函數名稱，請輸入 **TempMonitor**。
 - 針對執行期，選擇 Python 3.7。
 - 對於許可，請保留預設設定。這會建立授予基本 Lambda 許可的執行角色。不會使用此角色 AWS IoT Greengrass。
- 請在頁面底部，選擇建立函式。

2. 接著，註冊處理常式並上傳您的 Lambda 函數部署套件。

- 在程式碼索引標籤的程式碼來源下，選擇上傳來源。從下拉式清單中，選擇 .zip 檔案。



- 選擇上傳，然後選擇您的temp_monitor_python.zip部署套件。然後選擇 Save (儲存)。
- 在函數的程式碼索引標籤的執行時間設定下，選擇編輯，然後輸入下列值。
 - 針對執行期，選擇 Python 3.7。
 - 對於 Handler (處理常式)，輸入 **temp_monitor.function_handler**。

d. 選擇儲存。

 Note

AWS Lambda 主控台上的測試按鈕不適用於此函數。AWS IoT Greengrass 核心 SDK 不包含在 AWS Lambda 主控台中獨立執行 Greengrass Lambda 函數所需的模組。這些模組（例如 `greengrass_common`）會在部署到您的 Greengrass 核心之後提供給函數。

3. 現在，發佈 Lambda 函數的第一個版本，並為[該版本建立別名](#)。

 Note

Greengrass 群組可以依別名（建議）或版本參考 Lambda 函數。使用別名可讓您更輕鬆地管理程式碼更新，因為您不必在更新函數程式碼時變更訂閱資料表或群組定義。反之，您只需將別名指向新的函數版本。

- a. 請從操作功能表中選擇發行新版本。
- b. 針對 Version description (版本描述)，輸入 **First version**，然後選擇 Publish (發佈)。
- c. 在 TempMonitor: 1 組態頁面，從 Actions (動作) 選單中選擇 Create alias (建立別名)。
- d. 在建立警示頁面上使用下列值：
 - 對於名稱，輸入 **GG_TempMonitor**。
 - 對於版本，選擇 1。

 Note

AWS IoT Greengrass 不支援 \$LATEST 版本的 Lambda 別名。

e. 選擇建立。

現在您已準備好將 Lambda 函數新增至 Greengrass 群組。

步驟 6：將 Lambda 函數新增至 Greengrass 群組

在此步驟中，您將 Lambda 函數新增至 群組，然後設定其生命週期和環境變數。如需詳細資訊，請參閱[the section called “控制 Greengrass Lambda 函數執行”](#)。

1. 在群組組態頁面上，選擇 Lambda 函數索引標籤。
2. 在我的 Lambda 函數下，選擇新增。
3. 在新增 Lambda 函數頁面上，為您的 Lambda 函數選擇 TempMonitor。
4. 針對 Lambda 函數版本，選擇別名：GG_TempMonitor。
5. 選擇新增 Lambda 函數。

步驟 7：將訂閱新增到 Greengrass 群組

在此步驟中，您會新增訂閱，讓 Lambda 函數將輸入資料傳送至連接器。連接器定義它所訂閱的 MQTT 主題，所以這個訂閱使用其中一個主題。這是範例函數發佈到的同一個主題。

在本教學課程中，您也會建立訂閱，允許函數接收來自 AWS IoT 的模擬溫度讀數 AWS IoT，並允許從連接器接收狀態資訊。

1. 在群組組態頁面上，選擇訂閱索引標籤，然後選擇新增訂閱。
2. 在建立訂閱頁面上，設定來源和目標，如下所示：
 - a. 針對來源類型，選擇 Lambda 函數，然後選擇 TempMonitor。
 - b. 針對目標類型，選擇連接器，然後選擇 Twilio Notifications。
3. 針對主題篩選條件，選擇 **twilio/txt**。
4. 選擇建立訂閱。
5. 重複步驟 1 - 4 來建立允許 AWS IoT 將訊息發佈至 函數的訂閱。
 - a. 針對來源類型，選擇服務，然後選擇 IoT Cloud。
 - b. 針對選取目標，選擇 Lambda 函數，然後選擇 TempMonitor。
 - c. 針對 Topic filter (主題篩選條件)，輸入 **temperature/input**。
6. 重複步驟 1 - 4，建立允許連接器發佈訊息至 AWS IoT 的訂閱。
 - a. 針對來源類型，選擇連接器，然後選擇 Twilio Notifications。
 - b. 針對目標類型，選擇服務，然後選擇 IoT 雲端。

- c. 針對 Topic filter (主題篩選條件)，已為您輸入 **twilio/message/status**。此為連接器所發佈到的預先定義主題。

步驟 8：部署 Greengrass 群組

將群組部署到核心裝置。

1. 確定 AWS IoT Greengrass 核心正在執行。如果需要，請在您的 Raspberry Pi 終端機執行以下命令。

- a. 檢查精靈是否有在運作：

```
ps aux | grep -E 'greengrass.*daemon'
```

若輸出的 root 含有 `/greengrass/ggc/packages/ggc-version/bin/daemon` 項目，則精靈有在運作。

Note

路徑中的版本取決於核心裝置上安裝的核心 AWS IoT Greengrass 軟體版本。

- b. 若要啟動協助程式：

```
cd /greengrass/ggc/core/  
sudo ./greengrassd start
```

2. 在群組組態頁面上，選擇部署。
3.
 - a. 在 Lambda 函數索引標籤的系統 Lambda 函數區段下，選取 IP 偵測器，然後選擇編輯。
 - b. 在編輯 IP 偵測器設定對話方塊中，選取自動偵測和覆寫 MQTT 代理程式端點。
 - c. 選擇儲存。

這可讓裝置自動取得核心的連接資訊，例如 IP 位址、DNS、連接埠編號。建議自動偵測，但 AWS IoT Greengrass 也支援手動指定的端點。只會在第一次部署群組時收到復原方法的提示。

Note

如果出現提示，請授予許可來建立 [Greengrass 服務角色](#)，並將其與目前 AWS 帳戶中的 建立關聯 AWS 區域。此角色允許 AWS IoT Greengrass 存取 AWS 服務中的資源。

此部署頁面會顯示部署時間戳記、版本 ID 和狀態。完成後，部署顯示的狀態應為已完成。

如需故障診斷協助，請參閱[疑難排解](#)。

Note

Greengrass 群組一次只能包含一個版本的連接器。若要取得有關升級連接器版本的資訊，請參閱[the section called “升級連接器版本”](#)。

測試解決方案

1. 在 AWS IoT 主控台首頁上，選擇測試。
2. 對於訂閱主題，請使用下列值，然後選擇訂閱。Twilio Notifications 連接器會將狀態資訊發佈至此主題。

屬性	Value
訂閱主題	twilio/message/status
MQTT 承載顯示	將承載顯示為字串

3. 針對發佈至主題，請使用下列值，然後選擇發佈以叫用函數。

屬性	Value
主題	temperature/input
訊息	將 <i>recipient-name</i> 換成名稱，將 <i>recipient-phone-number</i> 換成文字

屬性	Value
	訊息收件人的電話號碼。範例：+12345000000 <pre>{ "to_name": " <i>recipient-name</i> ", "to_number": " <i>recipient-phone-number</i> ", "temperature": 31 }</pre> 如果您使用試驗帳戶，則必須將非 Twilio 收件人的電話號碼新增至已驗證的電話號碼清單。如需詳細資訊，請參閱驗證您的個人電話號碼。

如果成功，收件人會收到文字訊息，而主控台會顯示來自[輸出資料](#)success的狀態。

現在，將輸入訊息中的 temperature 變更為 **29** 並發佈。由於此值小於 30，TempMonitor 函數不會觸發 Twilio 訊息。

另請參閱

- [使用連接器整合服務和通訊協定](#)
- [the section called “AWS提供的 Greengrass 連接器”](#)

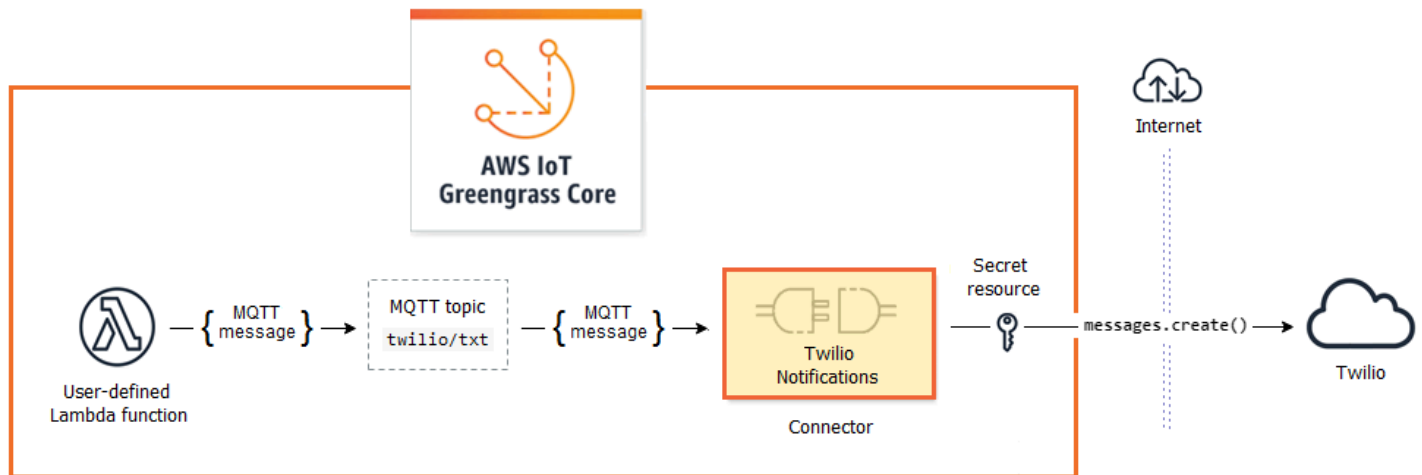
Greengrass 連接器入門 (CLI)

此功能適用於 AWS IoT Greengrass Core v1.7 和更新版本。

本教學課程說明如何使用 AWS CLI 來使用連接器。

使用連接器來加速您的開發生命週期。連接器是預先建置的、可重複使用的模組，可讓您更輕鬆地與服務、通訊協定和資源互動。它們可協助您更快地將商業邏輯部署到 Greengrass 裝置。如需詳細資訊，請參閱[使用連接器整合服務和通訊協定](#)。

在本教學課程中，您會設定和部署 [Twilio Notifications](#) 連接器。連接器接收 Twilio 訊息資訊做為輸入資料，然後觸發 Twilio 文字訊息。下圖顯示資料流程。



設定連接器之後，您可以建立 Lambda 函數和訂閱。

- 此函數會評估來自溫度感應器的模擬資料。它會有條件地將 Twilio 訊息資訊發佈到 MQTT 主題。這是連接器訂閱的主題。
- 該訂閱允許將函數發佈到主題，也允許連接器從主題接收資料。

Twilio Notifications 連接器需要 Twilio 驗證字符才能與 Twilio API 互動。字符是在中建立 AWS Secrets Manager 並從群組資源參考的文字類型秘密。這 AWS IoT Greengrass 可讓在 Greengrass 核心上建立秘密的本機副本，並在其中加密並提供給連接器。如需詳細資訊，請參閱[將私密部署至核心](#)。

本教學課程所述以下高階執行步驟：

1. [建立 Secrets Manager 秘密](#)
2. [建立資源定義和版本](#)
3. [建立連接器定義和版本](#)
4. [建立 Lambda 函數部署套件](#)
5. [建立 Lambda 函式](#)
6. [建立函數定義和版本](#)
7. [建立訂閱定義和版本](#)
8. [建立群組版本](#)

9. 建立部署

10 [the section called “測試解決方案”](#)

此教學課程需約 30 分鐘完成。

使用 AWS IoT Greengrass API

當您使用 Greengrass 群組和群組元件 (例如群組中的連接器、函數和資源) 時，了解下列模式會對您有幫助。

- 在階層的頂端，元件有定義物件，此物件是版本物件的容器。而版本是連接器、函數或其他元件類型的容器。
- 當您部署到 Greengrass 核心時，您會部署特定的群組版本。一個群組版本可以包含每種元件的一個版本。核心是必要的，但其他元件可依需要加入。
- 版本是不可變的，因此想要變更時必須建立新的版本。

Tip

如果您在執行 AWS CLI 命令時收到錯誤，請新增 `--debug` 參數，然後重新執行 命令以取得錯誤的詳細資訊。

AWS IoT Greengrass API 可讓您為元件類型建立多個定義。例如，您可以在每次建立 `FunctionDefinitionVersion` 時建立 `FunctionDefinition` 物件，或者您可以將新版本新增到現有的定義。這種彈性可讓您自訂版本管理系統。

先決條件

為完成此教學課程您需要：

- Greengrass 群組和 Greengrass 核心 (1.9.3 版或更新版本)。若要了解如何建立 Greengrass 群組或核心，請參閱 [入門 AWS IoT Greengrass](#)。入門教學課程也包含 AWS IoT Greengrass Core 軟體的安裝步驟。
- AWS IoT Greengrass 核心裝置上已安裝 Python 3.7。
- AWS IoT Greengrass 必須設定為支援本機秘密，如[秘密要求](#)中所述。

Note

此要求包括允許存取您的 Secrets Manager 秘密。如果您使用預設 Greengrass 服務角色，Greengrass 有權取得名稱開頭為 greengrass- 的秘密值。

- Twilio 帳戶 SID、驗證字符和已啟用 Twilio 的電話號碼。在您建立 Twilio 專案後，您就可在專案儀表板上使用這些值。

Note

您可以使用 Twilio 試用帳戶。如果您使用試驗帳戶，則必須將非 Twilio 收件人的電話號碼新增至已驗證的電話號碼清單。如需詳細資訊，請參閱[如何使用您的 免費 Twilio 試用帳戶](#)。

- AWS CLI 在您的電腦上安裝和設定。如需詳細資訊，請參閱AWS Command Line Interface 《使用者指南》中的[安裝 AWS Command Line Interface](#) 和[設定 AWS CLI](#)。

此教學課程中的範例是針對 Linux 和其他以 Unix 為基礎的系統所撰寫。如果您使用的是 Windows，請參閱[指定的參數值 AWS Command Line Interface](#)，以了解語法的差異。

如果命令包含 JSON 字串，教學課程提供的範例會在單行上有 JSON。在某些系統上，使用此格式也許能讓編輯和執行命令更輕鬆。

步驟 1：建立 Secrets Manager 秘密

在此步驟中，您會使用 AWS Secrets Manager API 為您的 Twilio 驗證字符建立秘密。

1. 首先，建立私密。

- 將 *twilio-auth-token* 換成您的驗證字符。

```
aws secretsmanager create-secret --name greengrass-TwilioAuthToken --secret-string twilio-auth-token
```

Note

根據預設，Greengrass 服務角色允許 AWS IoT Greengrass 取得名稱開頭為 greengrass- 的秘密值。如需詳細資訊，請參閱[私密需求](#)。

2. 從輸出複製私密的 ARN。您可以使用它來建立秘密資源，並設定 Twilio Notifications 連接器。

步驟 2：建立資源定義和版本

在此步驟中，您會使用 AWS IoT Greengrass API 來建立 Secrets Manager 秘密的秘密資源。

1. 建立包含最初版本的資源定義。
 - 將 *secret-arn* 換成您在上一步複製的私密 ARN。

JSON Expanded

```
aws greengrass create-resource-definition --name MyGreengrassResources --
initial-version '{
  "Resources": [
    {
      "Id": "TwilioAuthToken",
      "Name": "MyTwilioAuthToken",
      "ResourceDataContainer": {
        "SecretsManagerSecretResourceData": {
          "ARN": "secret-arn"
        }
      }
    }
  ]
}'
```

JSON Single-line

```
aws greengrass create-resource-definition \
--name MyGreengrassResources \
```



```
--initial-version '{"Resources": [{"Id": "TwilioAuthToken",
  "Name": "MyTwilioAuthToken", "ResourceDataContainer":
  {"SecretsManagerSecretResourceData": {"ARN": "secret-arn"}}}]}'
```

2. 從輸出複製資源定義的 LatestVersionArn。您可以使用這個值，將資源定義版本新增至您部署到核心的群組版本。

步驟 3：建立連接器定義和版本

在此步驟中，您會設定 Twilio Notifications 連接器的參數。

1. 建立含有最初版本的連接器定義。
 - 將 *account-sid* 換成您的 Twilio 帳戶 SID。
 - 使用 *Secrets Manager ### ## secret-arn*。ARN 連接器使用此 ARN 取得本機私密的值。
 - 將 *phone-number* 換成已啟用 Twilio 的電話號碼。Twilio 以此啟動文字訊息。這可在輸入訊息承載中覆寫。使用下列格式：+19999999999。

JSON Expanded

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --
initial-version '{
  "Connectors": [
    {
      "Id": "MyTwilioNotificationsConnector",
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/
TwilioNotifications/versions/4",
      "Parameters": {
        "TWILIO_ACCOUNT_SID": "account-sid",
        "TwilioAuthTokenSecretArn": "secret-arn",
        "TwilioAuthTokenSecretArn-ResourceId": "TwilioAuthToken",
        "DefaultFromPhoneNumber": "phone-number"
      }
    }
  ]
}'
```

JSON Single-line

```
aws greengrass create-connector-definition \  
--name MyGreengrassConnectors \  
--initial-version '{"Connectors": [{"Id": "MyTwilioNotificationsConnector",  
  "ConnectorArn": "arn:aws:greengrass:region::/connectors/TwilioNotifications/  
versions/4", "Parameters": {"TWILIO_ACCOUNT_SID": "account-sid",  
  "TwilioAuthTokenSecretArn": "secret-arn", "TwilioAuthTokenSecretArn-  
ResourceId": "TwilioAuthToken", "DefaultFromPhoneNumber": "phone-number"}]}]'
```

Note

TwilioAuthToken 是您在上一步中用來建立私密資源的 ID。

2. 從輸出複製連接器定義的 LatestVersionArn。您可以使用這個值，將連接器定義版本新增至您部署到核心的群組版本。

步驟 4：建立 Lambda 函數部署套件

若要建立 Lambda 函數，您必須先建立包含函數程式碼和相依性的 Lambda 函數部署套件。Greengrass Lambda 函數需要 [AWS IoT Greengrass 核心 SDK](#) 來處理任務，例如與核心環境中的 MQTT 訊息通訊，以及存取本機秘密。本教學課程會建立 Python 函數，因此您可以在部署套件中使用 SDK 的 Python 版本。

1. 從[AWS IoT Greengrass 核心開發套件](#)下載頁面，將適用於 Python AWS IoT Greengrass 的核心開發套件下載至您的電腦。
2. 解壓縮下載的封裝，以取得軟體開發套件。SDK 為 greengrasssdk 資料夾。
3. 將以下 Python 程式碼函數儲存在名為 temp_monitor.py 的本機檔案中。

```
import greengrasssdk  
import json  
import random  
  
client = greengrasssdk.client('iot-data')  
  
# publish to the Twilio Notifications connector through the twilio/txt topic  
def function_handler(event, context):  
    temp = event['temperature']
```

```
# check the temperature
# if greater than 30C, send a notification
if temp > 30:
    data = build_request(event)
    client.publish(topic='twilio/txt', payload=json.dumps(data))
    print('published:' + str(data))

print('temperature:' + str(temp))
return

# build the Twilio request from the input data
def build_request(event):
    to_name = event['to_name']
    to_number = event['to_number']
    temp_report = 'temperature:' + str(event['temperature'])

    return {
        "request": {
            "recipient": {
                "name": to_name,
                "phone_number": to_number,
                "message": temp_report
            }
        },
        "id": "request_" + str(random.randint(1,101))
    }
```

4. 將下列項目壓縮成名為 `temp_monitor_python.zip` 的檔案。建立 ZIP 檔案時，只包含程式碼及其依存項目，而不包含資料夾。
 - `temp_monitor.py`。應用程式邏輯。
 - `greengrasssdk`。發佈 MQTT 訊息的 Python Greengrass Lambda 函數所需的程式庫。

這是您的 Lambda 函數部署套件。

步驟 5：建立 Lambda 函數

現在，建立使用部署套件的 Lambda 函數。

1. 建立 IAM 角色，以便在建立函數時傳入角色 ARN。

JSON Expanded

```
aws iam create-role --role-name Lambda_empty --assume-role-policy '{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}'
```

JSON Single-line

```
aws iam create-role --role-name Lambda_empty --assume-role-policy '{"Version":
"2012-10-17", "Statement": [{"Effect": "Allow", "Principal": {"Service":
"lambda.amazonaws.com"}}, {"Action": "sts:AssumeRole"}]}'
```

Note

AWS IoT Greengrass 不會使用此角色，因為 Greengrass Lambda 函數的許可是在 Greengrass 群組角色中指定。本教學課程將建立空白角色。

2. 從輸出複製 Arn。
3. 使用 AWS Lambda API 建立 TempMonitor 函數。以下命令假設 zip 檔是在目前的目錄。
 - 將 *role-arn* 取代為您複製 Arn 的。

```
aws lambda create-function \
--function-name TempMonitor \
--zip-file fileb://temp_monitor_python.zip \
--role role-arn \
--handler temp_monitor.function_handler \
--runtime python3.7
```

4. 發佈函數的一個版本。

```
aws lambda publish-version --function-name TempMonitor --description 'First version'
```

5. 建立發佈版本的別名。

Greengrass 群組可以依別名（建議）或版本參考 Lambda 函數。使用別名可讓您更輕鬆地管理程式碼更新，因為您不必在更新函數程式碼時變更訂閱資料表或群組定義。反之，您只需將別名指向新的函數版本。

Note

AWS IoT Greengrass 不支援 \$LATEST 版本的 Lambda 別名。

```
aws lambda create-alias --function-name TempMonitor --name GG_TempMonitor --function-version 1
```

6. 從輸出複製 AliasArn。當您為設定函數時 AWS IoT Greengrass，以及在建立訂閱時，您會使用此值。

現在您已準備好設定的函數 AWS IoT Greengrass。

步驟 6：建立函數定義和版本

若要在 AWS IoT Greengrass 核心上使用 Lambda 函數，您可以建立以別名參考 Lambda 函數的函數定義版本，並定義群組層級組態。如需詳細資訊，請參閱[the section called “控制 Greengrass Lambda 函數執行”](#)。

1. 建立包含最初版本的函數定義。

- 將 *alias-arn* 換成您建立別名時所複製的 AliasArn。

JSON Expanded

```
aws greengrass create-function-definition --name MyGreengrassFunctions --
initial-version '{
  "Functions": [
    {
      "Id": "TempMonitorFunction",
      "FunctionArn": "alias-arn",
      "FunctionConfiguration": {
        "Executable": "temp_monitor.function_handler",
        "MemorySize": 16000,
        "Timeout": 5
      }
    }
  ]
}'
```

JSON Single-line

```
aws greengrass create-function-definition \
--name MyGreengrassFunctions \
--initial-version '{"Functions": [{"Id": "TempMonitorFunction",
  "FunctionArn": "alias-arn", "FunctionConfiguration": {"Executable":
  "temp_monitor.function_handler", "MemorySize": 16000,"Timeout": 5}}]}'
```

2. 從輸出複製 LatestVersionArn。您可以使用這個值，將函數定義版本新增至您部署到核心的群組版本。
3. 從輸出複製 Id。稍後更新函數時，您會使用這個值。

步驟 7：建立訂閱定義和版本

在此步驟中，您會新增訂閱，讓 Lambda 函數將輸入資料傳送至連接器。連接器定義它所訂閱的 MQTT 主題，所以這個訂閱使用其中一個主題。這是範例函數發佈到的同一個主題。

在本教學課程中，您也會建立訂閱，允許函數接收來自 AWS IoT 的模擬溫度讀數 AWS IoT，並允許從連接器接收狀態資訊。

1. 建立包含最初版本的訂閱定義，而此版本包含訂閱。

- 將 *alias-arn* 換成您為函數建立別名時所複製的 AliasArn。對用到它的兩個訂閱都使用此 ARN。

JSON Expanded

```
aws greengrass create-subscription-definition --initial-version '{
  "Subscriptions": [
    {
      "Id": "TriggerNotification",
      "Source": "alias-arn",
      "Subject": "twilio/txt",
      "Target": "arn:aws:greengrass:region::/connectors/
TwilioNotifications/versions/4"
    },
    {
      "Id": "TemperatureInput",
      "Source": "cloud",
      "Subject": "temperature/input",
      "Target": "alias-arn"
    },
    {
      "Id": "OutputStatus",
      "Source": "arn:aws:greengrass:region::/connectors/
TwilioNotifications/versions/4",
      "Subject": "twilio/message/status",
      "Target": "cloud"
    }
  ]
}'
```

JSON Single-line

```
aws greengrass create-subscription-definition \
--initial-version '{"Subscriptions": [{"Id": "TriggerNotification", "Source":
"alias-arn", "Subject": "twilio/txt", "Target": "arn:aws:greengrass:region::/
connectors/TwilioNotifications/versions/4"}, {"Id": "TemperatureInput",
"Source": "cloud", "Subject": "temperature/input", "Target": "alias-arn"},
{"Id": "OutputStatus", "Source": "arn:aws:greengrass:region::/connectors/
```

```
TwilioNotifications/versions/4", "Subject": "twilio/message/status", "Target":  
"cloud"]}]}'
```

2. 從輸出複製 LatestVersionArn。您可以使用這個值，將訂閱定義版本新增至您部署到核心的群組版本。

步驟 8：建立群組版本

現在，您可以開始建立群組版本，其中包含您要部署的所有項目。若要這樣做，您需要建立群組版本來參考每個元件類型的目標版本。

首先，取得核心定義版本的群組 ID 和 ARN。建立群組版本需要這些值。

1. 獲取群組 ID 和最新群組版本：
 - a. 取得目標 Greengrass 群組 ID 和目標群組版本 ID。此程序假設這是最新的群組和群組版本。下列查詢會傳回最近建立的群組。

```
aws greengrass list-groups --query "reverse(sort_by(Groups,  
&CreationTimestamp))[0]"
```

或者，您可以依名稱查詢。群組名稱不需要是唯一名稱，因此可能會傳回多個群組。

```
aws greengrass list-groups --query "Groups[?Name=='MyGroup']"
```

Note

您也可以在此 AWS IoT 主控台中找到這些值。群組 ID 會顯示在群組的 Settings (設定) 頁面上。群組版本 IDs 會顯示在群組的部署索引標籤上。

- b. 從輸出複製目標群組的 Id。部署群組時，您可以使用此 ID 取得核心定義版本。
 - c. 從輸出複製 LatestVersion，這是新增至群組的最新版本 ID。您可以使用此取得核心定義版本。
2. 取得核心定義版本的 ARN：
 - a. 取得群組版本。在此步驟中，我們假設最新的群組版本包含核心定義版本。
 - 將 *group-id* 取代為您為群組所複製的 Id。

- 將 *group-version-id* 換成您為群組所複製的 LatestVersion。

```
aws greengrass get-group-version \  
--group-id group-id \  
--group-version-id group-version-id
```

b. 從輸出複製 CoreDefinitionVersionArn。

3. 建立群組版本。

- 將 *group-id* 取代為您為群組所複製的 Id。
- 將 *core-definition-version-arn* 換成您為核心定義版本所複製的 CoreDefinitionVersionArn。
- 將 *resource-definition-version-arn* 換成您為資源定義所複製的 LatestVersionArn。
- 將 *connector-definition-version-arn* 換成您為連接器定義所複製的 LatestVersionArn。
- 將 *function-definition-version-arn* 換成您為函數定義所複製的 LatestVersionArn。
- 將 *subscription-definition-version-arn* 換成您為訂閱定義所複製的 LatestVersionArn。

```
aws greengrass create-group-version \  
--group-id group-id \  
--core-definition-version-arn core-definition-version-arn \  
--resource-definition-version-arn resource-definition-version-arn \  
--connector-definition-version-arn connector-definition-version-arn \  
--function-definition-version-arn function-definition-version-arn \  
--subscription-definition-version-arn subscription-definition-version-arn
```

4. 複製輸出中的 Version 值。這是群組版本的 ID。您可以使用這個值來部署群組版本。

步驟 9：建立部署

將群組部署到核心裝置。

1. 在核心裝置終端機中，確定 AWS IoT Greengrass 協助程式正在執行。

a. 檢查精靈是否有在運作：

```
ps aux | grep -E 'greengrass.*daemon'
```

若輸出的 root 含有 /greengrass/ggc/packages/1.11.6/bin/daemon 項目，則精靈有在運作。

b. 若要啟動協助程式：

```
cd /greengrass/ggc/core/  
sudo ./greengrassd start
```

2. 建立 部署。

- 將 *group-id* 取代為您為群組所複製的 Id。
- 將 *group-version-id* 取代為您為新群組版本所複製的 Version。

```
aws greengrass create-deployment \  
--deployment-type NewDeployment \  
--group-id group-id \  
--group-version-id group-version-id
```

3. 從輸出複製 DeploymentId。

4. 取得部署狀態。

- 將 *group-id* 取代為您為群組所複製的 Id。
- 將 *deployment-id* 換成您為部署所複製的 DeploymentId。

```
aws greengrass get-deployment-status \  
--group-id group-id \  
--deployment-id deployment-id
```

如果狀態為 Success，則部署成功。如需故障診斷協助，請參閱[疑難排解](#)。

測試解決方案

1. 在 AWS IoT 主控台首頁上，選擇測試。

- 對於訂閱主題，請使用下列值，然後選擇訂閱。Twilio Notifications 連接器會將狀態資訊發佈至此主題。

屬性	Value
訂閱主題	twilio/message/status
MQTT 承載顯示	將承載顯示為字串

- 針對發佈至主題，請使用下列值，然後選擇發佈以叫用函數。

屬性	Value
主題	temperature/input
訊息	將 <i>recipient-name</i> 換成名稱，將 <i>recipient-phone-number</i> 換成文字訊息收件人的電話號碼。範例：+12345000000 <pre>{ "to_name": " <i>recipient-name</i> ", "to_number": " <i>recipient-phone-number</i> ", "temperature": 31 }</pre> 如果您使用試驗帳戶，則必須將非 Twilio 收件人的電話號碼新增至已驗證的電話號碼清單。如需詳細資訊，請參閱驗證您的個人電話號碼。

如果成功，收件人會收到文字訊息，而主控台會顯示來自[輸出資料](#)success的狀態。

現在，將輸入訊息中的 temperature 變更為 **29** 並發佈。由於此值小於 30，TempMonitor 函數不會觸發 Twilio 訊息。

另請參閱

- [使用連接器整合服務和通訊協定](#)
- [the section called “AWS提供的 Greengrass 連接器”](#)
- [the section called “連接器入門 \(主控台\)”](#)
- [AWS Secrets Manager 命令](#) AWS CLI 參考中的 命令
- 《[AWS Identity and Access Management 命令參考](#)》中的 [\(IAM\)](#) AWS CLI 命令
- [AWS Lambda 命令](#) AWS CLI 參考中的 命令
- [AWS IoT Greengrass 命令](#) AWS CLI 參考中的 命令

Greengrass Discovery RESTful API

所有與 AWS IoT Greengrass 核心通訊的用戶端裝置都必須是 Greengrass 群組的成員。每個群組都必須有一個 Greengrass 核心。Discovery API 可讓裝置擷取連線至與用戶端裝置位於相同 Greengrass 群組的 Greengrass 核心所需的資訊。當用戶端裝置第一次上線時，它可以連線到 AWS IoT Greengrass 服務，並使用 Discovery API 來尋找：

- 其所屬的群組。用戶端裝置最多可為 10 個群組的成員。
- 群組中 Greengrass 核心的 IP 位址和連接埠。
- 群組憑證授權機構憑證可用於驗證 Greengrass 核心裝置。

Note

用戶端裝置也可以使用 AWS IoT 裝置 SDKs 來探索 Greengrass 核心的連線資訊。如需詳細資訊，請參閱 [AWS IoT 裝置 SDK](#)。

若要使用此 API，請傳送 HTTP 請求給 Discovery API 端點。例如：

```
https://greengrass-ats.iot.region.amazonaws.com:port/greengrass/discover/thing/thing-name
```

如需 AWS IoT Greengrass Discovery API 支援的 Amazon Web Services 區域和端點清單，請參閱 [AWS IoT Greengrass 中的端點和配額](#) AWS 一般參考。這是一個僅限 API 的資料平面。群組管理和 AWS IoT Core 操作的端點與 Discovery API 端點不同。

請求

此請求包含標準 HTTP 標題並已傳送到 Greengrass Discovery 端點，如下列範例所示。

連接埠號碼取決於核心是否設定為透過連接埠 8443 或連接埠 443 傳送 HTTPS 流量。如需詳細資訊，請參閱 [the section called “連線至連接埠 443 或透過網路代理”](#)。

連接埠 8443

```
HTTP GET https://greengrass-ats.iot.region.amazonaws.com:8443/greengrass/discover/thing/thing-name
```

連接埠 443

```
HTTP GET https://greengrass-ats.iot.region.amazonaws.com:443/greengrass/discover/thing/thing-name
```

在連接埠 443 上連線的用戶端必須實作 [Application Layer Protocol Negotiation \(ALPN\)](#) TLS 延伸模組，並在 ProtocolName 中 x-amzn-http-ca 做為 傳遞 ProtocolNameList。如需詳細資訊，請參閱《AWS IoT 開發人員指南》中的[通訊協定](#)。

Note

這些範例使用 Amazon Trust Services (ATS) 端點，這是用於 ATS 根 CA 憑證 (建議使用)。端點必須符合根 CA 憑證類型。如需詳細資訊，請參閱[the section called “服務端點必須符合憑證類型”](#)。

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
BODY: response document
```

如需詳細資訊，請參閱[範例 Discover 回應文件](#)。

探索授權

擷取連線資訊需要一項允許發起人執行 greengrass:Discover 動作的政策。TLS 用戶端憑證的共同身分驗證僅接受身分驗證的格式。以下為允許發起人執行此動作的範例政策：

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": "greengrass:Discover",
    "Resource": ["arn:aws:iot:us-west-2:123456789012:thing/MyThingName"]
  }]
}
```

}

範例 Discover 回應文件

下列文件顯示用戶端裝置的回應，該用戶端裝置是具有一個 Greengrass 核心、一個端點和一個群組 CA 憑證的群組成員：

```
{
  "GGGroups": [
    {
      "GGGroupId": "gg-group-01-id",
      "Cores": [
        {
          "thingArn": "core-01-thing-arn",
          "Connectivity": [
            {
              "id": "core-01-connection-id",
              "hostAddress": "core-01-address",
              "portNumber": core-01-port,
              "metadata": "core-01-description"
            }
          ]
        }
      ]
    }
  ],
  "CAs": [
    "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----"
  ]
}
}
```

下列文件顯示用戶端裝置的回應，該用戶端裝置是具有一個 Greengrass 核心、多個端點和多個群組 CA 憑證的兩個群組的成員：

```
{
  "GGGroups": [
    {
      "GGGroupId": "gg-group-01-id",
      "Cores": [
        {
          "thingArn": "core-01-thing-arn",
```

```

    "Connectivity": [
      {
        "id": "core-01-connection-id",
        "hostAddress": "core-01-address",
        "portNumber": core-01-port,
        "metadata": "core-01-connection-1-description"
      },
      {
        "id": "core-01-connection-id-2",
        "hostAddress": "core-01-address-2",
        "portNumber": core-01-port-2,
        "metadata": "core-01-connection-2-description"
      }
    ]
  },
  "CAs": [
    "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
    "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
    "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----"
  ]
},
{
  "GGGroupId": "gg-group-02-id",
  "Cores": [
    {
      "thingArn": "core-02-thing-arn",
      "Connectivity" : [
        {
          "id": "core-02-connection-id",
          "hostAddress": "core-02-address",
          "portNumber": core-02-port,
          "metadata": "core-02-connection-1-description"
        }
      ],
      "CAs": [
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----"
      ]
    }
  ]
}
]
}

```



```
}
```

Note

Greengrass 群組必須只定義一個 Greengrass 核心。AWS IoT Greengrass 服務中包含 Greengrass 核心清單的任何回應都只包含一個 Greengrass 核心。

若已安裝 cURL，則可以測試探索請求。例如：

```
$ curl --cert 1a23bc4d56.cert.pem --key 1a23bc4d56.private.key https://greengrass-ats.iot.us-west-2.amazonaws.com:8443/greengrass/discover/thing/MyDevice
{"GGGroups":[{"GGGroupId":"1234a5b6-78cd-901e-2fgh-3i45j6k1789","Cores":[{"thingArn":"arn:aws:iot:us-west-2:123456789012:thing/MyFirstGroup_Core","Connectivity":[{"Id":"AUTOIP_192.168.1.4_1","HostAddress":"192.168.1.5","PortNumber":8883,"Metadata":""}]}],"CAs":["-----BEGIN CERTIFICATE-----\ncert-contents\n-----END CERTIFICATE-----\n"]}]}
```

中的安全性 AWS IoT Greengrass

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，這些架構專為滿足最安全敏感組織的需求而建置。

安全是 AWS 與您之間共同責任。[共同責任模型](#)將其描述為雲端的安全性和雲端中的安全性：

- 雲端的安全性 – AWS 負責保護在 中執行 AWS 服務的基礎設施 AWS 雲端。AWS 也為您提供可安全使用的服務。在[AWS 合規計畫](#)中，第三方稽核人員會定期測試和驗證我們的安全有效性。若要了解適用的合規計劃 AWS IoT Greengrass，請參閱[AWS 合規計劃的服務範圍](#)。
- 雲端內部的安全 – 您的責任取決於所使用的 AWS 服務。您也必須對其他因素負責，包括資料的機密性、您的公司的要求和適用法律和法規。

使用 時 AWS IoT Greengrass，您也必須負責保護裝置、本機網路連線和私有金鑰的安全。

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 Greengrass。下列主題說明如何設定 AWS IoT Greengrass 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 Greengrass 資源。

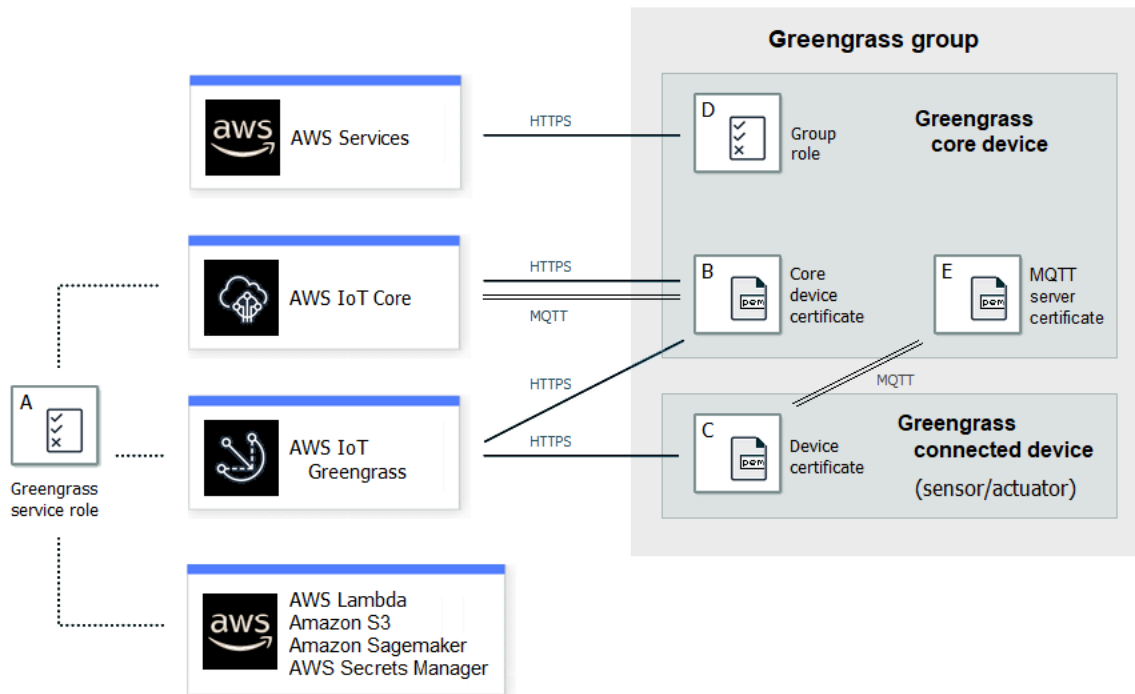
主題

- [安全性概觀 AWS IoT Greengrass](#)
- [中的資料保護 AWS IoT Greengrass](#)
- [的裝置身分驗證和授權 AWS IoT Greengrass](#)
- [的身分和存取管理 AWS IoT Greengrass](#)
- [的合規驗證 AWS IoT Greengrass](#)
- [中的彈性 AWS IoT Greengrass](#)
- [中的基礎設施安全 AWS IoT Greengrass](#)
- [中的組態和漏洞分析 AWS IoT Greengrass](#)
- [AWS IoT Greengrass 和介面 VPC 端點 \(AWS PrivateLink\)](#)
- [的安全最佳實務 AWS IoT Greengrass](#)

安全性概觀 AWS IoT Greengrass

AWS IoT Greengrass 使用 X.509 憑證、AWS IoT 政策和 IAM 政策和角色來保護在本機 Greengrass 環境中的裝置上執行的應用程式。

下圖顯示 AWS IoT Greengrass 安全模型的元件：



A - Greengrass 服務角色

從和其他 AWS 服務存取 AWS 資源 AWS IoT Greengrass 時 AWS IoT Core AWS Lambda，擔任的客戶建立 IAM 角色。如需詳細資訊，請參閱[the section called “Greengrass 服務角色”](#)。

B - 核心裝置憑證

用於使用 AWS IoT Core 和驗證 Greengrass 核心的 X.509 憑證 AWS IoT Greengrass。如需詳細資訊，請參閱[the section called “裝置身分驗證和授權”](#)。

C - 裝置憑證

用來驗證用戶端裝置的 X.509 憑證，也稱為連線裝置，具有 AWS IoT Core 和 AWS IoT Greengrass。如需詳細資訊，請參閱[the section called “裝置身分驗證和授權”](#)。

D - 群組角色

從 Greengrass 核心呼叫 AWS 服務 AWS IoT Greengrass 時，擔任的客戶建立 IAM 角色。

您可以使用此角色來指定使用者定義的 Lambda 函數和連接器存取 AWS 服務所需的存取許可，例如 DynamoDB。您也可以使用它來允許 AWS IoT Greengrass 將串流管理員串流匯出至 AWS 服務，並寫入 CloudWatch Logs。如需詳細資訊，請參閱[the section called “Greengrass 群組角色”](#)。

Note

AWS IoT Greengrass 對於 Lambda 函數的雲端版本，不會使用在 中 AWS Lambda 指定的 Lambda 執行角色。

E-MQTT 伺服器憑證

用於 Greengrass 核心裝置與 Greengrass 群組中用戶端裝置之間的 Transport Layer Security (TLS) 交互身分驗證的憑證。憑證由群組 CA 憑證簽署，該憑證存放在 中 AWS 雲端。

裝置連線工作流程

本節說明用戶端裝置如何連線至 AWS IoT Greengrass 服務和 Greengrass 核心裝置。用戶端裝置是與核心 AWS IoT Core 裝置位於相同 Greengrass 群組中的已註冊裝置。

- Greengrass 核心裝置會使用其裝置憑證、私有金鑰和 AWS IoT Core 根 CA 憑證來連線至 AWS IoT Greengrass 服務。在核心裝置上，[組態檔案中的 crypto](#) 物件會指定這些項目的檔案路徑。
- Greengrass 核心裝置會從 AWS IoT Greengrass 服務下載群組成員資格資訊。
- 當部署於 Greengrass 核心裝置時，裝置憑證管理員 (DCM) 會處理 Greengrass 核心裝置的本機伺服器憑證管理工作。
- 用戶端裝置會使用其裝置憑證、私有金鑰和 AWS IoT Core 根 CA 憑證連線至 AWS IoT Greengrass 服務。進行連線後，用戶端裝置會使用 Greengrass Discovery Service 來尋找其 Greengrass 核心裝置的 IP 地址。用戶端裝置也會下載群組 CA 憑證，用於與 Greengrass 核心裝置進行 TLS 交互身分驗證。
- 用戶端裝置會嘗試連線至 Greengrass 核心裝置，並傳遞其裝置憑證和用戶端 ID。如果用戶端 ID 符合用戶端裝置的物件名稱，且憑證有效 (Greengrass 群組的一部分)，則會進行連線。若否，則終止連線。

用戶端裝置 AWS IoT 的政策必須授予 `greengrass:Discover` 許可，以允許用戶端裝置探索核心的連線資訊。如需有關此政策陳述式的詳細資訊，請參閱[the section called “探索授權”](#)。

設定 AWS IoT Greengrass 安全性

設定您的 Greengrass 應用程式的安全性

1. 為您的 Greengrass 核心裝置建立 AWS IoT Core 物件。

2. 為您的 Greengrass 核心裝置產生金鑰對和裝置憑證。
3. 建立和附加 [AWS IoT 政策](#) 至裝置憑證。憑證和政策允許 Greengrass 核心裝置存取 AWS IoT Core 和服務 AWS IoT Greengrass。如需詳細資訊，請參閱[核心裝置的最低 AWS IoT 政策](#)。

Note

不支援在核心裝置的[政策中使用物件政策變數](#) (`iot:Connection.Thing.*`)。AWS IoT 核心使用相同的裝置憑證進行[多個連線](#)，AWS IoT Core 但連線中的用戶端 ID 可能與核心物件名稱不完全相符。

4. 建立 [Greengrass 服務角色](#) 此 IAM 角色會授權 代表您 AWS IoT Greengrass 從其他 AWS 服務存取資源。這可讓 AWS IoT Greengrass 執行基本任務，例如擷取 AWS Lambda 函數和管理裝置影子。

您可以在 AWS 區域中使用相同的服務角色，但它必須與您使用的每個 AWS 區域 AWS 帳戶 中的相關聯 AWS IoT Greengrass。

5. (選用) 建立 [Greengrass 群組角色](#)。此 IAM 角色授予許可給在 Greengrass 核心上執行的 Lambda 函數和連接器，以呼叫 AWS 服務。例如，[Kinesis Firehose 連接器](#) 需要將記錄寫入 Amazon Data Firehose 交付串流的許可。

您只能連接一個角色到 Greengrass 群組。

6. 為每個連接到 Greengrass 核心的裝置建立 AWS IoT Core 物件。

Note

您也可以使用現有的 AWS IoT Core 物件和憑證。

7. 為連接到 Greengrass 核心的每個裝置建立裝置憑證、金鑰對和 AWS IoT 政策。

AWS IoT Greengrass 核心安全主體

Greengrass 核心使用以下安全主體：AWS IoT 用戶端、本機 MQTT 伺服器和本機秘密管理員。這些主體的組態會存放在 `config.json` 組態檔案的 `crypto` 物件中。如需詳細資訊，請參閱[the section called “AWS IoT Greengrass 核心組態檔案”](#)。

此組態包含主體元件針對身分驗證及加密使用的私有金鑰的路徑。AWS IoT Greengrass 支援兩個模式的私有金鑰儲存：以硬體為基礎或以檔案系統為基礎 (預設值)。如需在硬體安全模組上存放金鑰的詳細資訊，請參閱[the section called “硬體安全整合”](#)。

AWS IoT 用戶端

AWS IoT 用戶端 (IoT 用戶端) 會管理 Greengrass 核心與 之間透過網際網路的通訊 AWS IoT Core。在建立此通訊的 TLS 連線時，AWS IoT Greengrass 會使用具有公有和私有金鑰的 X.509 憑證進行交互身分驗證。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的 [X.509 憑證和 AWS IoT Core](#)。

IoT 用戶端支援 RSA 和 EC 憑證和金鑰。憑證和私有金鑰路徑會在 config.json 中為 IoTCertificate 主體指定。

MQTT 伺服器

本機 MQTT 伺服器會管理群組中 Greengrass 核心和用戶端裝置之間透過本機網路的通訊。建立此通訊的 TLS 連線時，AWS IoT Greengrass 會使用 X.509 憑證搭配公有和私有金鑰進行交互身分驗證。

根據預設，會為您 AWS IoT Greengrass 產生 RSA 私有金鑰。若要將核心設定為使用不同的私有金鑰，您必須在 config.json 中提供 MQTTServerCertificate 主體的金鑰路徑。您必須負責輪換客戶提供的金鑰。

私有金鑰支援

	RSA 金鑰	EC 金鑰
Key type	Supported	Supported
重要參數	Minimum 2048-bit length	NIST P-256 or NIST P-384 curve
磁碟格式	PKCS#1, PKCS#8	SECG1, PKCS#8
最低 GGC 版本	- 使用預設 RSA 金鑰：1.0 - 指定 RSA 金鑰：1.7	指定 EC 金鑰：1.9

私有金鑰的組態會決定相關的程序。如需 Greengrass Core 支援做為伺服器的密碼套件的清單，請參閱 [the section called “TLS 密碼套件支援”](#)。

如果未指定私有金鑰 (預設值)

- AWS IoT Greengrass 會根據您的輪換設定來輪換金鑰。
- 核心會產生 RSA 金鑰，它用來產生憑證。

- MQTT 伺服器憑證具有一個 RSA 公開金鑰和 SHA-256 RSA 簽章。

如果指定 RSA 私有金鑰（需要 GGC 1.7 版或更新版本）

- 您需負責輪換金鑰。
- 核心會使用指定的金鑰來產生憑證。
- RSA 金鑰的長度必須至少為 2048 個位元。
- MQTT 伺服器憑證具有一個 RSA 公開金鑰和 SHA-256 RSA 簽章。

如果指定 EC 私有金鑰（需要 GGC 1.9 版或更新版本）

- 您需負責輪換金鑰。
- 核心會使用指定的金鑰來產生憑證。
- EC 私有金鑰必須使用 NIST P-256 或 NIST P-384 曲線。
- MQTT 伺服器憑證具有一個 EC 公開金鑰和 SHA-256 RSA 簽章。

無論金鑰類型為何，核心呈現的 MQTT 伺服器憑證都有一個 SHA-256 RSA 簽章。因此，用戶端必須支援 SHA-256 RSA 憑證驗證，以與核心建立安全的連線。

Secrets Manager

本機秘密管理員會安全地管理您建立之秘密的本機副本 AWS Secrets Manager。它使用私有金鑰來保護用於加密密碼的資料金鑰。如需詳細資訊，請參閱[將私密部署至核心](#)。

預設會使用 IoT 用戶端私有金鑰，但您可以在 config.json 為 SecretsManager 主體指定不同的私有金鑰。僅支援 RSA 金鑰類型。如需詳細資訊，請參閱[the section called “指定用於秘密加密的私有金鑰”](#)。

Note

目前，僅 AWS IoT Greengrass 支援 [PKCS#1 v1.5](#) 填補機制，在使用硬體型私有金鑰時加密和解密本機秘密。如果您遵循廠商提供的指示來手動產生硬體型私有金鑰，請務必選擇 PKCS#1 v1.5。AWS IoT Greengrass does 不支援最佳非對稱加密填補 (OAEP)。

私有金鑰支援

	RSA 金鑰	EC 金鑰
Key type	Supported	Not supported
重要參數	Minimum 2048-bit length	Not applicable

	RSA 金鑰	EC 金鑰
磁碟格式	PKCS#1, PKCS#8	Not applicable
最低 GGC 版本	1.7	Not applicable

MQTT 簡訊工作流程中的受管訂閱

AWS IoT Greengrass 使用訂閱資料表來定義如何在 Greengrass 群組中的用戶端裝置、函數和連接器，以及與 AWS IoT Core 或本機影子服務之間交換 MQTT 訊息。每個訂閱都會指定傳送或接收訊息的來源、目標和 MQTT 主題（或主旨）。只有在定義對應的訂閱時，才 AWS IoT Greengrass 允許將訊息從來源傳送至目標。

訂閱只會定義單向的訊息流程，從來源到目標。若要支援雙向訊息交換，您必須建立兩個訂閱，每個方向各一個。

TLS 密碼套件支援

AWS IoT Greengrass 使用 AWS IoT Core 傳輸安全模型，透過 [TLS 密碼套件](#) 來加密與雲端的通訊。此外，AWS IoT Greengrass 資料會在靜態時加密（在雲端中）。如需 AWS IoT Core 傳輸安全性和支援密碼套件的詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[傳輸安全性](#)。

支援用於本機網路通訊的密碼套件

相反地 AWS IoT Core，AWS IoT Greengrass 核心支援下列本機網路 TLS 密碼套件進行憑證簽署演算法。於檔案系統上存放私有金鑰時，支援所有這些加密套件。當核心設定為使用硬體安全模組 (HSM) 時，支援一個子集。如需詳細資訊，請參閱 [the section called “安全性主體”](#) 和 [the section called “硬體安全整合”](#)。資料表也包含支援所需的最低版 AWS IoT Greengrass 核心軟體。

	加密	HSM 支援	最低 GGC 版本
TLSv1.2	TLS_ECDHE _RSA_WITH _AES_128_CBC_SHA	Supported	1.0
	TLS_ECDHE _RSA_WITH _AES_256_CBC_SHA	Supported	1.0

	加密	HSM 支援	最低 GGC 版本
TLSv1.1	TLS_ECDHE _RSA_WITH _AES_256_ GCM_SHA384	Supported	1.0
	TLS_RSA_W ITH_AES_1 28_CBC_SHA	Not supported	1.0
	TLS_RSA_W ITH_AES_1 28_GCM_SHA256	Not supported	1.0
	TLS_RSA_W ITH_AES_2 56_CBC_SHA	Not supported	1.0
	TLS_RSA_W ITH_AES_2 56_GCM_SHA384	Not supported	1.0
	TLS_ECDHE _ECDSA_WI TH_AES_12 8_GCM_SHA256	Supported	1.9
	TLS_ECDHE _ECDSA_WI TH_AES_25 6_GCM_SHA384	Supported	1.9
	TLS_ECDHE _RSA_WITH _AES_128_CBC_SHA	Supported	1.0

	加密	HSM 支援	最低 GGC 版本
TLSv1.0	TLS_ECDHE _RSA_WITH _AES_256_CBC_SHA	Supported	1.0
	TLS_RSA_W ITH_AES_1 28_CBC_SHA	Not supported	1.0
	TLS_RSA_W ITH_AES_2 56_CBC_SHA	Not supported	1.0
	TLS_ECDHE _RSA_WITH _AES_128_CBC_SHA	Supported	1.0
	TLS_ECDHE _RSA_WITH _AES_256_CBC_SHA	Supported	1.0
	TLS_RSA_W ITH_AES_1 28_CBC_SHA	Not supported	1.0
	TLS_RSA_W ITH_AES_2 56_CBC_SHA	Not supported	1.0

中的資料保護 AWS IoT Greengrass

AWS [共同責任模型](#)適用於 中的資料保護 AWS IoT Greengrass。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](#)。如需有關歐洲資料保護的相關資訊，請參閱 AWS 安全性部落格上的 [AWS 共同的責任模型和 GDPR](#) 部落格文章。

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 Greengrass 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。

如需在 中保護敏感資訊的詳細資訊 AWS IoT Greengrass，請參閱 [the section called “請勿記錄敏感資訊”](#)。

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。

主題

- [資料加密](#)
- [硬體安全整合](#)

資料加密

AWS IoT Greengrass 使用加密來保護傳輸中（透過網際網路或本機網路）和靜態（存放在 中）的資料 AWS 雲端。

AWS IoT Greengrass 環境中的裝置通常會收集傳送至 AWS 服務的資料，以供進一步處理。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。

主題

- [傳輸中加密](#)
- [靜態加密](#)
- [Greengrass 核心裝置的金鑰管理](#)

傳輸中加密

AWS IoT Greengrass 有三種通訊模式，其中的資料正在傳輸中：

- [the section called “透過網際網路傳輸資料”](#)。Greengrass 核心與 AWS IoT Greengrass 網際網路之間的通訊會加密。
- [the section called “透過區域網路傳輸資料”](#)。透過本機網路的 Greengrass 核心和用戶端裝置之間的通訊會加密。
- [the section called “核心裝置上的資料”](#)。Greengrass 核心裝置上的元件之間的通訊不會加密。

透過網際網路傳輸資料

AWS IoT Greengrass 使用 Transport Layer Security (TLS) 來加密透過網際網路的所有通訊。傳送到的所有資料 AWS 雲端 都會使用 MQTT 或 HTTPS 通訊協定透過 TLS 連線傳送，因此預設是安全的。AWS IoT Greengrass 會使用 AWS IoT 傳輸安全模型。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[傳輸安全性](#)。

透過區域網路傳輸資料

AWS IoT Greengrass 使用 TLS 來加密 Greengrass 核心和用戶端裝置之間透過本機網路進行的所有通訊。如需詳細資訊，請參閱[區域網路通訊支援的密碼套件](#)。

保護區域網路和私有金鑰是您的責任。

對於 Greengrass 核心裝置，您須負責：

- 使用最新的安全性修補程式，持續更新核心。
- 使用最新的安全性修補程式，持續更新系統程式庫。
- 保護私有金鑰。如需詳細資訊，請參閱[the section called “金鑰管理”](#)。

對於用戶端裝置，您有責任：

- 將 TLS 堆疊保持在最新狀態。
- 保護私有金鑰。

核心裝置上的資料

AWS IoT Greengrass 不會加密在 Greengrass 核心裝置上本機交換的資料，因為資料不會離開裝置。這包括使用者定義的 Lambda 函數、連接器、AWS IoT Greengrass 核心 SDK 和系統元件之間的通訊，例如串流管理員。

靜態加密

AWS IoT Greengrass 存放您的資料：

- [the section called “中的靜態資料 AWS 雲端”](#)。此資料已加密。
- [the section called “Greengrass 核心上的靜態資料”](#)。此資料不會加密（秘密的本機副本除外）。

中的靜態資料 AWS 雲端

AWS IoT Greengrass 會加密存放在 中的客戶資料 AWS 雲端。此資料使用 管理的 AWS KMS 金鑰進行保護 AWS IoT Greengrass。

Greengrass 核心上的靜態資料

AWS IoT Greengrass 依賴 Unix 檔案許可和全磁碟加密（如果啟用）來保護核心上的靜態資料。保護檔案系統和裝置是您的責任。

不過，AWS IoT Greengrass 會加密從中擷取之秘密的本機副本 AWS Secrets Manager。如需詳細資訊，請參閱[the section called “秘密金鑰加密”](#)。

Greengrass 核心裝置的金鑰管理

客戶有責任確保安全儲存 Greengrass 核心裝置上的加密 (公有和私有) 金鑰。AWS IoT Greengrass 在下列情況下使用公有金鑰和私有金鑰：

- IoT 用戶端金鑰會搭配 IoT 憑證在 Greengrass 核心連接至 AWS IoT Core 時，驗證 Transport Layer Security (TLS) 交握。如需詳細資訊，請參閱[the section called “裝置身分驗證和授權”](#)。

Note

金鑰和憑證也稱為核心私有金鑰和核心裝置憑證。

- MQTT 伺服器金鑰是使用 MQTT 伺服器憑證來驗證核心和用戶端裝置之間的 TLS 連線。如需詳細資訊，請參閱[the section called “裝置身分驗證和授權”](#)。

- 本機 Secrets Manager 也會使用 IoT 用戶端金鑰來保護用來加密本機秘密的資料金鑰，但您可以提供自己的私有金鑰。如需詳細資訊，請參閱[the section called “秘密金鑰加密”](#)。

Greengrass 核心支援使用檔案系統許可的私有金鑰儲存體、[硬體安全模組](#)，或兩者。如果您使用以檔案系統為基礎的私有金鑰，您必須負責將其安全儲存於核心裝置上。

在 Greengrass 核心上，您私有金鑰的位置會在 config.json 檔案的 crypto 區段中指定。如果您將核心設定為使用客戶提供的 MQTT 伺服器憑證金鑰，則您必須負責輪換金鑰。如需詳細資訊，請參閱[the section called “安全性主體”](#)。

對於用戶端裝置，您有責任將 TLS 堆疊保持在最新狀態並保護私有金鑰。私有金鑰會與裝置憑證搭配使用，以驗證 AWS IoT Greengrass 服務的 TLS 連線。

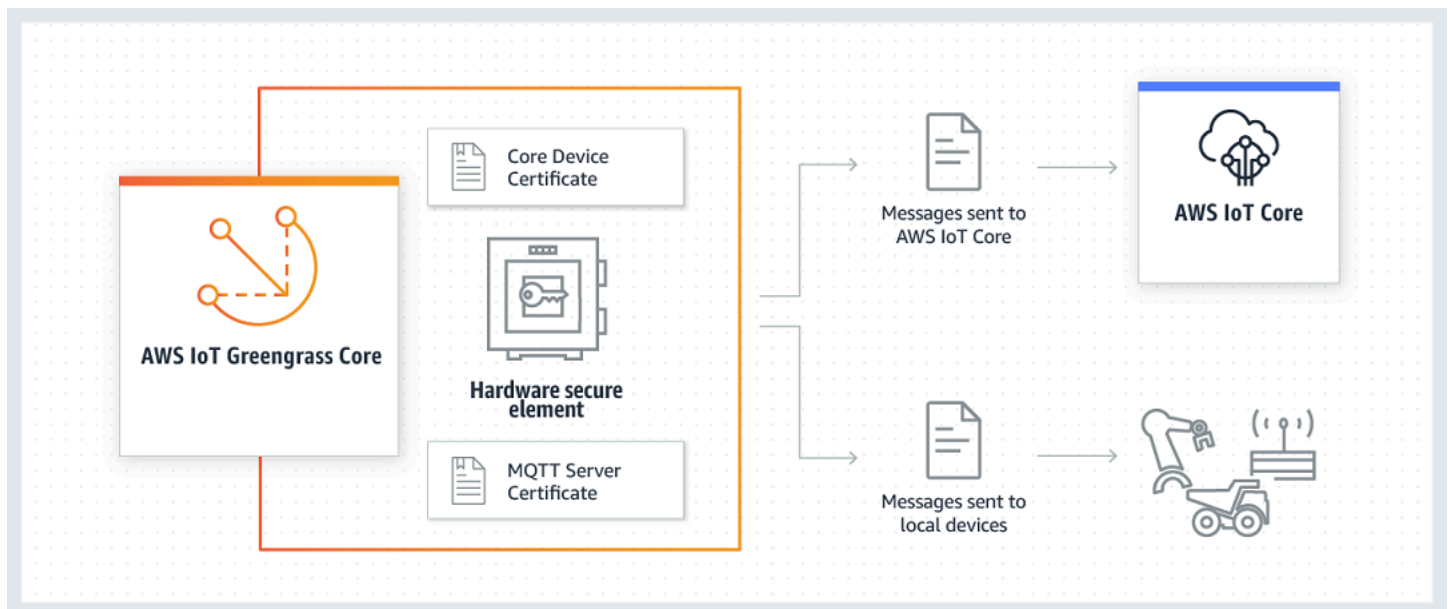
硬體安全整合

此功能適用於 AWS IoT Greengrass Core v1.7 和更新版本。

AWS IoT Greengrass 支援透過 [PKCS#11 介面](#) 使用硬體安全模組 (HSM)，以安全儲存和卸載私有金鑰。這可防止金鑰公開或在軟體中遭到複製。私有金鑰可以安全地存放在硬體模組 (例如 HSM、信賴平台模組 (TPM)) 或其他加密元素上。

在 Device [AWS Partner Catalog](#) 中搜尋符合此功能資格的裝置。

下圖顯示 AWS IoT Greengrass 核心的硬體安全架構。



在標準安裝中，AWS IoT Greengrass 會使用兩個私有金鑰。當 Greengrass 核心連線到時，AWS IoT 用戶端 (IoT 用戶端) 元件會在 Transport Layer Security (TLS) 交握期間使用一個金鑰 AWS IoT Core。(此金鑰也稱為核心私有金鑰)。另一個金鑰則由本機 MQTT 伺服器使用，讓 Greengrass 裝置可與 Greengrass 核心通訊。如果想要針對這兩個元件使用硬體安全，您可以使用共用的私有金鑰或個別的私有金鑰。如需詳細資訊，請參閱[the section called “佈建實務”](#)。

Note

在標準安裝上，本機 Secrets Manager 還會對其加密程序使用 IoT 用戶端金鑰，但您可以使用自己的私有金鑰。它必須是具有的最小長度為 2048 位元的 RSA 金鑰。如需詳細資訊，請參閱[the section called “指定用於秘密加密的私有金鑰”](#)。

需求

在可以設定 Greengrass 核心的硬體安全之前，您必須具有下列各項：

- 硬體安全模組 (HSM)，其支援您針對 IoT 用戶端、本機 MQTT 伺服器和本機 Secrets Manager 元件的目標私有金鑰組態。組態可以包含一個、兩個或三個以硬體為基礎的私有金鑰，取決於您是否將元件設定為共用金鑰。如需有關私有金鑰支援的詳細資訊，請參閱 [the section called “安全性主體”](#)。
- 對於 RSA 金鑰：支援 RSA-2048 金鑰大小 (或更大) 及 [PKCS#1 v1.5](#) 簽章配置。
- 對於 EC 金鑰：NIST P-256 或 NIST P-384 曲線。

Note

在 Device [AWS Partner Catalog](#) 中搜尋符合此功能資格的裝置。

- 可在執行時間載入 (使用 libdl)，並且提供 [PKCS#11](#) 函數的 PKCS#11 提供者程式庫。
- 硬體模組必須可透過插槽標籤根據 PKCS#11 規格中的定義來解析。
- 您必須使用廠商提供的佈建工具，在 HSM 上產生和載入私有金鑰。
- 私有金鑰必須可透過物件標籤來解析。
- 核心裝置憑證。這是對應至私有金鑰的 IoT 用戶端憑證。
- 如果您使用的是 Greengrass OTA 更新代理程式，則必須安裝 [OpenSSL libp11 PKCS#11](#) 包裝函式程式庫。如需詳細資訊，請參閱[the section called “設定 OTA 更新”](#)。

此外，也請確定符合下列條件：

- 與私有金鑰相關聯的 IoT 用戶端憑證會在 中註冊 AWS IoT 並啟用。您可以在 AWS IoT 主控台的管理、展開所有裝置、選擇實物，以及選擇核心實物的憑證索引標籤下進行驗證。
- AWS IoT Greengrass 核心軟體 v1.7 或更新版本安裝在核心裝置上，如入門教學課程的[單元 2](#)所述。使用 MQTT 伺服器的 EC 金鑰需要 1.9 版或更新版本。
- 憑證已附加至 Greengrass 核心。您可以從 AWS IoT 主控台中核心物件的管理頁面進行驗證。

Note

目前，AWS IoT Greengrass 不支援直接從 HSM 載入 CA 憑證或 IoT 用戶端憑證。憑證必須在檔案系統上 Greengrass 可讀取的位置中以純文字檔案載入。

AWS IoT Greengrass 核心的硬體安全組態

硬體安全是在 Greengrass 組態檔中設定。這是位於 `/greengrass-root/config` 目錄中 [config.json](#) 的檔案。

Note

若要逐步進行使用純軟體實作來設定 HSM 組態的程序，請參閱 [the section called “單元 7：模擬硬體安全整合”](#)。

Important

範例中模擬的組態不提供任何安全優勢。其旨在讓您了解 PKCS#11 規格，並在您計劃未來使用硬體型 HSM 時，執行軟體的初始測試。

若要在 中設定硬體安全性 AWS IoT Greengrass，您可以在 中編輯 crypto 物件 `config.json`。

使用硬體安全時，會使用 crypto 物件，針對核心上的 PKCS#11 供應商程式庫指定憑證、私有金鑰和資產的路徑。

```
"crypto": {
  "PKCS11" : {
    "OpenSSLEngine" : "/path-to-p11-openssl-engine",
    "P11Provider" : "/path-to-pkcs11-provider-so",
    "slotLabel" : "crypto-token-name",
```



```

    "slotUserPin" : "crypto-token-user-pin"
  },
  "principals" : {
    "IoTCertificate" : {
      "privateKeyPath" : "pkcs11:object=core-private-key-label;type=private",
      "certificatePath" : "file:///path-to-core-device-certificate"
    },
    "MQTTServerCertificate" : {
      "privateKeyPath" : "pkcs11:object=server-private-key-label;type=private"
    },
    "SecretsManager" : {
      "privateKeyPath": "pkcs11:object=core-private-key-label;type=private"
    }
  },
  "caPath" : "file:///path-to-root-ca"

```

crypto 物件包含下列屬性：

欄位	Description	備註
caPath	AWS IoT 根 CA 的絕對路徑。	必須為此格式的檔案 URI : file:///absolute/ path/to/file 。
 Note 請確定您的端點對應至您的憑證類型。		
PKCS11		
OpenSSLEngine	選用。OpenSSL 引擎 .so 檔案的絕對路徑，此檔案可在 OpenSSL 上啟用 PKCS#11 支援。	必須是檔案系統上的檔案路徑。 如果您使用具有硬體安全性的 Greengrass OTA 更新代理程式，則需要此屬性。如需詳細資訊，請參閱 the section called “設定 OTA 更新” 。

欄位	Description	備註
P11Provider	PKCS#11 實作的 libdl-loadable 程式庫的絕對路徑。	必須是檔案系統上的檔案路徑。
slotLabel	用於識別硬體模組的插槽標籤。	必須符合 PKCS#11 標籤規格。
slotUserPin	用來向模組驗證 Greengrass 核心的使用者 PIN 碼。	必須具有足夠的許可，才能使用設定的私有金鑰執行 C_Sign。
principals		
IoTCertificate	The certificate and private key that the core uses to make requests to AWS IoT.	
IoTCertificate .privateKeyPath	核心私有金鑰的路徑。	如為檔案系統儲存，必須為此格式的檔案 URI : file:///absolute/path/to/file 。 如為 HSM 儲存，必須為指定物件標籤的 RFC 7512 PKCS #11 路徑。
IoTCertificate .certificatePath	核心裝置憑證的絕對路徑。	必須為此格式的檔案 URI : file:///absolute/path/to/file 。
MQTTServerCertificate	選用。核心用於結合憑證做為 MQTT 伺服器或閘道的私有金鑰。	

欄位	Description	備註
MQTTServerCertificate.privateKeyPath	本機 MQTT 伺服器私有金鑰的路徑。	使用此值，為本機 MQTT 伺服器指定您自己的私有金鑰。 如為檔案系統儲存，必須為此格式的檔案 URI : file:///absolute/path/to/file 。 如為 HSM 儲存，必須為指定物件標籤的 RFC 7512 PKCS #11 路徑。 如果省略此屬性，會根據輪換設定 AWS IoT Greengrass 輪換金鑰。若有指定，則由客戶負責輪換金鑰。
SecretsManager	The private key that secures the data key used for encryption. For more information, see 將私密部署至核心 .	
SecretsManager.privateKeyPath	本機 Secrets Manager 私有金鑰的路徑。	僅支援 RSA 金鑰。 如為檔案系統儲存，必須為此格式的檔案 URI : file:///absolute/path/to/file 。 如為 HSM 儲存，必須為指定物件標籤的 RFC 7512 PKCS #11 路徑。必須使用 PKCS#1 v1.5 填補機制產生私密金鑰。

欄位	Description	備註
caPath	AWS IoT 根 CA 的絕對路徑。	必須為此格式的檔案 URI : file:///absolute/path/to/file 。
 Note 請確定您的端點對應至您的憑證類型。		
PKCS11		
OpenSSLEngine	選用。OpenSSL 引擎 .so 檔案的絕對路徑，此檔案可在 OpenSSL 上啟用 PKCS#11 支援。	必須是檔案系統上的檔案路徑。 如果您使用具有硬體安全性的 Greengrass OTA 更新代理程式，則需要此屬性。如需詳細資訊，請參閱 the section called “設定 OTA 更新” 。
P11Provider	PKCS#11 實作的 libdl-loadable 程式庫的絕對路徑。	必須是檔案系統上的檔案路徑。
slotLabel	用於識別硬體模組的插槽標籤。	必須符合 PKCS#11 標籤規格。
slotUserPin	用來向模組驗證 Greengrass 核心的使用者 PIN 碼。	必須具有足夠的許可，才能使用設定的私有金鑰執行 C_Sign。
principals		
IoTCertificate	The certificate and private key that the core uses to make requests to AWS IoT.	
IoTCertificate.privateKeyPath	核心私有金鑰的路徑。	如為檔案系統儲存，必須為此格式的檔案

欄位	Description	備註
		URI : file:///absolute/path/to/file 。 如為 HSM 儲存，必須為指定物件標籤的 RFC 7512 PKCS #11 路徑。
IoTCertificate .certificatePath	核心裝置憑證的絕對路徑。	必須為此格式的檔案 URI : file:///absolute/path/to/file 。
MQTTServerCertificate	選用。核心用於結合憑證做為 MQTT 伺服器或閘道的私有金鑰。	
MQTTServerCertificate .privateKeyPath	本機 MQTT 伺服器私有金鑰的路徑。	使用此值，為本機 MQTT 伺服器指定您自己的私有金鑰。 如為檔案系統儲存，必須為此格式的檔案 URI : file:///absolute/path/to/file 。 如為 HSM 儲存，必須為指定物件標籤的 RFC 7512 PKCS #11 路徑。 如果省略此屬性，會根據輪換設定 AWS IoT Greengrass 輪換金鑰。若有指定，則由客戶負責輪換金鑰。
SecretsManager	The private key that secures the data key used for encryption. For more information, see 將私密部署至核心 .	

欄位	Description	備註
SecretsManager .privateKeyPath	本機 Secrets Manager 私有金鑰的路徑。	僅支援 RSA 金鑰。 如為檔案系統儲存，必須為此格式的檔案 URI : file:///absolute/path/to/file 。 如為 HSM 儲存，必須為指定物件標籤的 RFC 7512 PKCS # 11 路徑。必須使用 PKCS#1 v1.5 填補機制產生私密金鑰。
caPath	AWS IoT 根 CA 的絕對路徑。	必須為此格式的檔案 URI : file:///absolute/path/to/file 。  Note 請確定您的端點對應至您的憑證類型。
PKCS11		
OpenSSLEngine	選用。OpenSSL 引擎 .so 檔案的絕對路徑，此檔案可在 OpenSSL 上啟用 PKCS#11 支援。	必須是檔案系統上的檔案路徑。 如果您使用具有硬體安全性的 Greengrass OTA 更新代理程式，則需要此屬性。如需詳細資訊，請參閱the section called “設定 OTA 更新”。

欄位	Description	備註
P11Provider	PKCS#11 實作的 libdl-loadable 程式庫的絕對路徑。	必須是檔案系統上的檔案路徑。
slotLabel	用於識別硬體模組的插槽標籤。	必須符合 PKCS#11 標籤規格。
slotUserPin	用來向模組驗證 Greengrass 核心的使用者 PIN 碼。	必須具有足夠的許可，才能使用設定的私有金鑰執行 C_Sign。
principals		
IoTCertificate	The certificate and private key that the core uses to make requests to AWS IoT.	
IoTCertificate .privateKeyPath	核心私有金鑰的路徑。	如為檔案系統儲存，必須為此格式的檔案 URI : file:///absolute/path/to/file 。 如為 HSM 儲存，必須為指定物件標籤的 RFC 7512 PKCS #11 路徑。
IoTCertificate .certificatePath	核心裝置憑證的絕對路徑。	必須為此格式的檔案 URI : file:///absolute/path/to/file 。
MQTTServerCertificate	選用。核心用於結合憑證做為 MQTT 伺服器或閘道的私有金鑰。	

欄位	Description	備註
MQTTServerCertificate.privateKeyPath	本機 MQTT 伺服器私有金鑰的路徑。	使用此值，為本機 MQTT 伺服器指定您自己的私有金鑰。 如為檔案系統儲存，必須為此格式的檔案 URI : file:///absolute/path/to/file 。 如為 HSM 儲存，必須為指定物件標籤的 RFC 7512 PKCS #11 路徑。 如果省略此屬性，會根據輪換設定 AWS IoT Greengrass 輪換金鑰。若有指定，則由客戶負責輪換金鑰。
SecretsManager	The private key that secures the data key used for encryption. For more information, see 將私密部署至核心 .	
SecretsManager.privateKeyPath	本機 Secrets Manager 私有金鑰的路徑。	僅支援 RSA 金鑰。 如為檔案系統儲存，必須為此格式的檔案 URI : file:///absolute/path/to/file 。 如為 HSM 儲存，必須為指定物件標籤的 RFC 7512 PKCS #11 路徑。必須使用 PKCS#1 v1.5 填補機制產生私密金鑰。

硬體 AWS IoT Greengrass 安全的佈建實務

以下是安全和效能相關的佈建實務。

安全性

- 使用內部硬體隨機號碼產生器，直接在 HSM 產生私有金鑰。

Note

如果您將私有金鑰設定為搭配此功能使用（遵循硬體廠商提供的指示），請注意，AWS IoT Greengrass 目前僅支援 PKCS1 v1.5 填補機制來加密和解密[本機秘密](#)。AWS IoT Greengrass 不支援最佳非對稱加密填補 (OAEP)。

- 設定私有金鑰來禁止匯出。
- 使用硬體廠商提供的佈建工具，使用受硬體保護的私有金鑰產生憑證簽署請求 (CSR)，然後使用 AWS IoT 主控台產生用戶端憑證。

Note

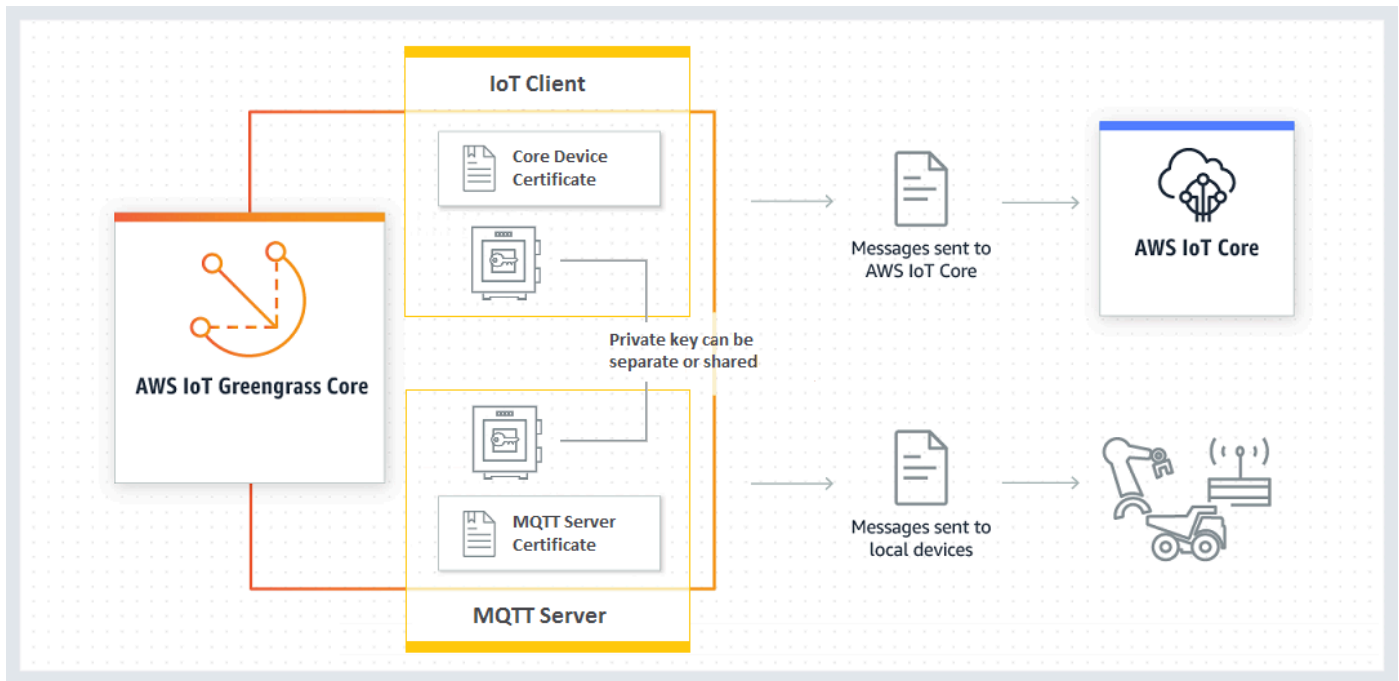
在 HSM 上產生私有金鑰時，輪換金鑰的實務不適用。

效能

下圖顯示 AWS IoT Greengrass 核心上的 IoT 用戶端元件和本機 MQTT 伺服器。如果想要針對這兩個元件使用 HSM 組態，您可以使用相同的私有金鑰或個別的私有金鑰。如果您使用不同的金鑰，則必須將它們存放在相同插槽。

Note

AWS IoT Greengrass 不會對您在 HSM 上存放的金鑰數量施加任何限制，因此您可以存放 IoT 用戶端、MQTT 伺服器和秘密管理員元件的私有金鑰。不過，一些 HSM 廠商可能會對您可以在插槽中存放的金鑰數量施行一些限制。



一般而言，IoT 用戶端金鑰的使用頻率不高，因為 AWS IoT Greengrass Core 軟體會維持與雲端的長期連線。不過，每次 Greengrass 裝置連接到核心，就會使用 MQTT 伺服器金鑰。這些互動直接影響效能。

當 MQTT 伺服器金鑰存放在 HSM 上時，裝置可以連接的速率取決於 HSM 每秒可以執行的 RSA 簽章操作數目。例如，如果 HSM 需要 300 毫秒，在 RSA-2048 私有金鑰上執行 RSASSA-PKCS1-v1.5 簽章，則每秒只有三個裝置可以連接到 Greengrass 核心。在建立連線之後，便不再使用 HSM，並且會套用標準 [AWS IoT Greengrass 的配額](#)。

為了減輕效能瓶頸，您可以將 MQTT 伺服器的私有金鑰存放在檔案系統上，而不是存放在 HSM 上。使用此組態時，MQTT 伺服器的行為如同未啟用硬體安全一般。

AWS IoT Greengrass 支援 IoT 用戶端和 MQTT 伺服器元件的多個金鑰儲存組態，因此您可以針對安全性和效能需求進行最佳化。下表包含範例組態。

Configuration	IoT 金鑰	MQTT 金鑰	效能
HSM 共用金鑰	HSM：金鑰 A	HSM：金鑰 A	受到 HSM 或 CPU 限制
HSM 獨立金鑰	HSM：金鑰 A	HSM：金鑰 B	受到 HSM 或 CPU 限制

Configuration	IoT 金鑰	MQTT 金鑰	效能
僅限 IoT 的 HSM	HSM：金鑰 A	檔案系統：金鑰 B	受到 CPU 限制
傳統	檔案系統：金鑰 A	檔案系統：金鑰 B	受到 CPU 限制

若要將 Greengrass 核心設定為使用 MQTT 伺服器的檔案系統型金鑰，請省略的 `principals.MQTTServerCertificate` 區段 `config.json` (或者，如果您不是使用產生的預設金鑰，請指定金鑰的檔案型路徑 AWS IoT Greengrass)。產生的 `crypto` 物件看起來像這樣：

```
"crypto": {
  "PKCS11": {
    "OpenSSLEngine": "...",
    "P11Provider": "...",
    "slotLabel": "...",
    "slotUserPin": "..."
  },
  "principals": {
    "IoTCertificate": {
      "privateKeyPath": "...",
      "certificatePath": "..."
    },
    "SecretsManager": {
      "privateKeyPath": "..."
    }
  },
  "caPath" : "..."
}
```

硬體安全整合支援的密碼套件

AWS IoT Greengrass 當核心設定為硬體安全性時，支援一組密碼套件。這是核心設定為使用檔案型安全時支援之密碼套件的子集。如需詳細資訊，請參閱[the section called “TLS 密碼套件支援”](#)。

Note

從 Greengrass 裝置透過本機網路連接到 Greengrass 核心時，請務必使用其中一個支援的密碼套件來建立 TLS 連線。

設定無線更新的支援

若要在使用硬體安全性時啟用 AWS IoT Greengrass 核心軟體的over-the-air (OTA) 更新，您必須安裝 OpenSC libp11 [PKCS#11 包裝函式程式庫](#)並編輯 Greengrass 組態檔案。如需 OTA 更新的詳細資訊，請參閱 [AWS IoT Greengrass Core 軟體的 OTA 更新](#)。

1. 停止 Greengrass 協助程式。

```
cd /greengrass-root/ggc/core/  
sudo ./greengrassd stop
```

Note

greengrass-root 代表 Core AWS IoT Greengrass 軟體安裝在您裝置上的路徑。通常，這是 /greengrass 目錄。

2. 安裝 OpenSSL 引擎。支援 OpenSSL 1.0 或 1.1 版本。

```
sudo apt-get install libengine-pkcs11-openssl
```

3. 在您的系統上尋找 OpenSSL 引擎 (libpkcs11.so) 的路徑：

- a. 取得程式庫的已安裝套件清單。

```
sudo dpkg -L libengine-pkcs11-openssl
```

libpkcs11.so 檔案位於 engines 目錄中。

- b. 複製檔案的完整路徑 (例如 /usr/lib/ssl/engines/libpkcs11.so)。

4. 開啟 Greengrass 組態檔。這是 /*greengrass-root*/config目錄中[config.json](#)的檔案。
5. 對於 OpenSSLEngine 屬性，輸入 libpkcs11.so 檔案的路徑。

```
{  
  "crypto": {  
    "caPath" : "file:///path-to-root-ca",  
    "PKCS11" : {  
      "OpenSSLEngine" : "/path-to-p11-openssl-engine",  
      "P11Provider" : "/path-to-pkcs11-provider-so",  
      "slotLabel" : "crypto-token-name",  
      "slotUserPin" : "crypto-token-user-pin"
```

```
    },  
    ...  
  }  
  ...  
}
```

Note

如果 `OpenSSL Engine` 屬性不存在於 `PKCS11` 物件中，則新增它。

6. 啟動 Greengrass 協助程式。

```
cd /greengrass-root/ggc/core/  
sudo ./greengrassd start
```

與舊版 AWS IoT Greengrass 核心軟體的回溯相容性

具有硬體安全支援 AWS IoT Greengrass 的核心軟體與 v1.6 及更早版本產生的 `config.json` 檔案完全回溯相容。如果 `crypto` 物件不存在於 `config.json` 組態檔案中，則 AWS IoT Greengrass 會使用以檔案為基礎的 `coreThing.certPath`、`coreThing.keyPath` 和 `coreThing.caPath` 屬性。此回溯相容性適用於 Greengrass OTA 更新，不會覆蓋 `config.json` 中指定的檔案型組態。

沒有 PKCS#11 支援的硬體

PKCS#11 程式庫通常是硬體廠商提供，或是開放原始碼。例如，使用標準相容硬體 (例如 TPM1.2) 時，可以使用現有的開放原始碼軟體。不過，如果您的硬體沒有對應的 PKCS#11 程式庫實作，或者您想要撰寫自訂 PKCS#11 供應商，您應該聯絡 AWS 企業支援代表，詢問整合相關問題。

另請參閱

- PKCS #11 Cryptographic Token Interface Usage Guide 2.40 版。編輯者為 John Leiseboer 和 Robert Griffin。2014 年 11 月 16 日。OASIS Committee Note 02。 <http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/cn02/pkcs11-ug-v2.40-cn02.html>。最新版本： <http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/pkcs11-ug-v2.40.html>。
- [RFC 7512](#)
- [PKCS #1: RSA Encryption Version 1.5](#)

的裝置身分驗證和授權 AWS IoT Greengrass

AWS IoT Greengrass 環境中的裝置使用 X.509 憑證進行身分驗證，並使用 AWS IoT 政策進行授權。憑證和政策可讓裝置彼此安全地連線 AWS IoT Core，以及 AWS IoT Greengrass。

X.509 憑證為數位憑證，採用 X.509 公有金鑰基礎架構標準，將公有金鑰與憑證內含的身分建立關聯。X.509 憑證是由稱為憑證授權機構 (CA) 的受信任實體所發行。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。僅憑證授權機構可存取憑證授權機構憑證。

AWS IoT 政策定義 AWS IoT 裝置允許的一組操作。具體而言，它們允許和拒絕對 AWS IoT Core 和 AWS IoT Greengrass 資料平面操作的存取，例如發佈 MQTT 訊息和擷取裝置影子。

所有裝置都需要 AWS IoT Core 登錄檔中的項目，以及已連接 AWS IoT 政策的已啟用 X.509 憑證。裝置可分為兩類：

- Greengrass 核心 Greengrass 核心裝置使用憑證和 AWS IoT 政策來連線 AWS IoT Core。憑證和政策也允許 AWS IoT Greengrass 將組態資訊、Lambda 函數、連接器和受管訂閱部署至核心裝置。
- 用戶端裝置。用戶端裝置（也稱為連線裝置、Greengrass 裝置或裝置）是透過 MQTT 連接到 Greengrass 核心的裝置。它們使用憑證和政策來連線至 AWS IoT Core 和服務 AWS IoT Greengrass。這可讓用戶端裝置使用 AWS IoT Greengrass Discovery Service 尋找並連線至核心裝置。用戶端裝置使用相同的憑證來連線至 AWS IoT Core 裝置閘道和核心裝置。用戶端裝置也會使用探索資訊與核心裝置進行交互身分驗證。如需詳細資訊，請參閱[the section called “裝置連線工作流程”](#)及[the section called “使用 Greengrass 核心管理裝置身分驗證”](#)。

X.509 憑證

必須驗證核心和用戶端裝置之間的通訊，以及裝置和 AWS IoT Core 或 AWS IoT Greengrass 之間的通訊。此相互驗證是根據已註冊的 X.509 裝置憑證和加密金鑰。

在 AWS IoT Greengrass 環境中，裝置使用具有公有和私有金鑰的憑證進行下列 Transport Layer Security (TLS) 連線：

- Greengrass 核心上的 AWS IoT 用戶端元件 AWS IoT Greengrass 會透過網際網路連線至 AWS IoT Core 和。
- 連線至 AWS IoT Greengrass 的用戶端裝置可透過網際網路取得核心探索資訊。
- Greengrass 核心上的 MQTT 伺服器元件會透過本機網路連線至 群組中的用戶端裝置。

AWS IoT Greengrass 核心裝置會將憑證存放在兩個位置：

- `/greengrass-root/certs` 中的核心裝置憑證。核心裝置憑證通常命名為 `hash.cert.pem` (例如，`86c84488a5.cert.pem`)。當核心連線至和 AWS IoT Core AWS IoT Greengrass 服務時，AWS IoT 用戶端會使用此憑證進行交互身分驗證。
- `/greengrass-root/ggc/var/state/server` 中的 MQTT 伺服器憑證。MQTT 伺服器憑證的名稱為 `server.crt`。此憑證用於本機 MQTT 伺服器 (位於 Greengrass 核心) 和 Greengrass 裝置之間的相互驗證。

Note

`greengrass-root` 代表 Core AWS IoT Greengrass 軟體安裝在您裝置上的路徑。通常，這是 `/greengrass` 目錄。

如需詳細資訊，請參閱[the section called “安全性主體”](#)。

憑證授權機構 (CA) 憑證

核心裝置和用戶端裝置會下載根 CA 憑證，用於使用 AWS IoT Core 和 AWS IoT Greengrass 服務進行身分驗證。我們建議您使用 Amazon Trust Service (ATS) 根憑證授權機構憑證，例如 [Amazon 根 CA 1](#)。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[伺服器身分驗證的 CA 憑證](#)。

Note

您的根 CA 憑證類型必須符合您的端點。使用 ATS 根 CA 憑證搭配 ATS 端點 (偏好)，或使用 VeriSign 根 CA 憑證搭配舊式端點。只有部分 Amazon Web Services 區域支援舊版端點。如需詳細資訊，請參閱[the section called “服務端點必須符合憑證類型”](#)。

用戶端裝置也會下載 Greengrass 群組 CA 憑證。這是在相互驗證期間用來驗證 Greengrass 核心上的 MQTT 伺服器憑證。如需詳細資訊，請參閱[the section called “裝置連線工作流程”](#)。MQTT 伺服器憑證的預設到期日為七天。

本機 MQTT 伺服器上的憑證輪換

用戶端裝置使用本機 MQTT 伺服器憑證與 Greengrass 核心裝置進行交互身分驗證。根據預設，此憑證會在七天後到期。此有限期間是依據安全最佳實務。MQTT 伺服器憑證是由儲存在雲端的群組憑證授權機構憑證簽署的。

若要進行憑證輪換，您的 Greengrass 核心裝置必須上線，並且能夠定期直接存取 AWS IoT Greengrass 服務。當憑證過期時，核心裝置會嘗試連線到 AWS IoT Greengrass 服務以取得新的憑證。如果連線成功，核心裝置會下載新的 MQTT 伺服器憑證並重新啟動本機 MQTT 服務。此時，所有連接到核心的用戶端裝置都會中斷連線。如果核心裝置在過期時離線，則不會收到替代憑證。任何連接至核心裝置的新嘗試都會遭到拒絕。現有的連線不受影響。在還原 AWS IoT Greengrass 與服務的連線並下載新的 MQTT 伺服器憑證之前，用戶端裝置無法連線至核心裝置。

您可以根據需求將過期日設為 7 到 30 天。更頻繁的輪換也需要更頻繁的雲端連線。不常輪換可能會造成安全問題。如果您想要將憑證過期設定為大於 30 天的值，請聯絡支援。

在 AWS IoT 主控台中，您可以在群組的設定頁面上管理憑證。在 AWS IoT Greengrass API 中，您可以使用 [UpdateGroupCertificateConfiguration](#) 動作。

當 MQTT 伺服器憑證過期，驗證憑證都會失敗。用戶端裝置必須能夠偵測故障並終止連線。

AWS IoT 資料平面操作的 政策

使用 AWS IoT 政策來授權存取 AWS IoT Core 和 AWS IoT Greengrass 資料平面。AWS IoT Core 資料平面向包含裝置、使用者和應用程式的操作，例如連線至 AWS IoT Core 和訂閱主題。AWS IoT Greengrass 資料平面向包含 Greengrass 裝置的操作，例如擷取部署和更新連線資訊。

AWS IoT 政策是類似 [IAM 政策](#) 的 JSON 文件。它包含一或多個指定下列屬性的政策陳述式：

- Effect。存取模式，可以是 Allow 或 Deny。
- Action。政策允許或拒絕的動作清單。
- Resource。允許或拒絕動作的資源清單。

AWS IoT 政策支援 * 萬用字元，並將 MQTT 萬用字元 (+ 和 #) 視為文字字串。如需 * 萬用字元的詳細資訊，請參閱 AWS Identity and Access Management 《使用者指南》中的 [在資源 ARNs 中使用萬用字元](#)。

如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的 [AWS IoT 政策和 AWS IoT 政策動作](#)。

Note

AWS IoT Core 可讓您將 AWS IoT 政策連接至物件群組，以定義裝置群組的許可。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。

AWS IoT Greengrass 政策動作

Greengrass 核心動作

AWS IoT Greengrass 定義了 Greengrass 核心裝置可在政策中使用的下列 AWS IoT 政策動作：

`greengrass:AssumeRoleForGroup`

Greengrass 核心裝置使用 Token Exchange Service (TES) 系統 Lambda 函數擷取登入資料的許可。繫結到擷取的登入資料的許可是依據已連接到設定的群組角色的政策。

當 Greengrass 核心裝置嘗試擷取登入資料 (假設登入資料未快取在本機) 時，會檢查此許可。

`greengrass:CreateCertificate`

讓 Greengrass 核心裝置建立自身伺服器憑證的許可。

當 Greengrass 核心裝置建立憑證時，會檢查此許可。Greengrass 核心裝置會在第一次執行、核心的連線資訊有所變更時及指定的輪換期間，嘗試建立伺服器憑證。

`greengrass:GetConnectivityInfo`

讓 Greengrass 核心裝置擷取自身連線資訊的許可。

當 Greengrass 核心裝置嘗試從中擷取其連線資訊時，會檢查此許可 AWS IoT Core。

`greengrass:GetDeployment`

讓 Greengrass 核心裝置擷取部署的許可。

當 Greengrass 核心裝置嘗試從雲端擷取部署和部署狀態時，會檢查此許可。

`greengrass:GetDeploymentArtifacts`

Greengrass 核心裝置擷取部署成品的許可，例如群組資訊或 Lambda 函數。

當 Greengrass 核心裝置收到部署，然後嘗試擷取部署成品時，會檢查此許可。

`greengrass:UpdateConnectivityInfo`

讓 Greengrass 核心裝置使用 IP 或主機名稱資訊更新自身連線資訊的許可。

當 Greengrass 核心裝置嘗試在雲端更新其連線資訊時，會檢查此許可。

`greengrass:UpdateCoreDeploymentStatus`

讓 Greengrass 核心裝置更新部署狀態的許可。

當 Greengrass 核心裝置收到部署，然後嘗試更新部署狀態時，會檢查此許可。

Greengrass 裝置動作

AWS IoT Greengrass 定義下列政策動作，用戶端裝置可在政策中使用 AWS IoT：

`greengrass:Discover`

用戶端裝置使用 [Discovery API](#) 擷取其群組核心連線資訊和群組憑證授權單位的許可。

當用戶端裝置使用 TLS 交互身分驗證呼叫 Discovery API 時，會檢查此許可。

AWS IoT Greengrass 核心裝置的最低 AWS IoT 政策

以下範例政策包含支援核心裝置基本 Greengrass 功能所需的最少動作組合。

- 政策會列出可供核心裝置發佈訊息、訂閱和接收訊息的 MQTT 主題和主題篩選條件，包括用於陰影狀態的主題。若要支援 Greengrass 群組中的 AWS IoT Core Lambda 函數、連接器和用戶端裝置之間的訊息交換，請指定您要允許的主題和主題篩選條件。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[發佈/訂閱政策範例](#)。
- 此政策包含一個區段，AWS IoT Core 允許取得、更新和刪除核心裝置的影子。若要允許 Greengrass 群組中用戶端裝置的陰影同步，請在Resource清單中指定目標 Amazon Resource Name (ARNs) (例如，`arn:aws:iot:region:account-id:thing/device-name`)。
- 不支援在核心裝置的[政策中使用物件政策變數](#) (`iot:Connection.Thing.*`)。AWS IoT 核心使用相同的裝置憑證進行[多個連線](#)，AWS IoT Core 但連線中的用戶端 ID 可能不完全符合核心物件名稱。
- 針對 `greengrass:UpdateCoreDeploymentStatus` 許可，Resource ARN 中的最終區段為核心裝置的 URL 編碼 ARN。

JSON

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

```

```

        "Resource": [
            "arn:aws:iot:us-east-1:123456789012:client/core-name-"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "iot:Publish",
            "iot:Receive"
        ],
        "Resource": [
            "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/core-name-
        *"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "iot:Subscribe"
        ],
        "Resource": [
            "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/core-
name - *"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "iot:GetThingShadow",
            "iot:UpdateThingShadow",
            "iot>DeleteThingShadow"
        ],
        "Resource": [
            "arn:aws:iot:us-east-1:123456789012:thing/core-name - *"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "greengrass:AssumeRoleForGroup",
            "greengrass:CreateCertificate"
        ],
        "Resource": [
            "*"
        ]
    }
]

```

```

    ]
  },
  {
    "Effect": "Allow",
    "Action": [
      "greengrass:GetDeployment"
    ],
    "Resource": [
      "arn:aws:greengrass:us-east-1:123456789012:/greengrass/
groups/group-id/deployments/*"
    ]
  },
  {
    "Effect": "Allow",
    "Action": [
      "greengrass:GetDeploymentArtifacts"
    ],
    "Resource": [
      "arn:aws:greengrass:us-east-1:123456789012:/greengrass/
groups/group-id/deployments/*"
    ]
  },
  {
    "Effect": "Allow",
    "Action": [
      "greengrass:UpdateCoreDeploymentStatus"
    ],
    "Resource": [
      "arn:aws:greengrass:us-east-1:123456789012:/
greengrass/groups/group-id/deployments/*/cores/arn%3Aaws%3Aiot
%3Aregion%3A123456789012%3Athing%2Fcore-name"
    ]
  },
  {
    "Effect": "Allow",
    "Action": [
      "greengrass:GetConnectivityInfo",
      "greengrass:UpdateConnectivityInfo"
    ],
    "Resource": [
      "arn:aws:iot:us-east-1:123456789012:thing/core-name - *"
    ]
  }
]

```

```
}
```

Note

AWS IoT 用戶端裝置的政策通常需要 `iot:Connect`、`iot:Receive`、`iot:Publish` 和 `iot:Subscribe` 動作的類似許可。

若要允許用戶端裝置自動偵測裝置所屬 Greengrass 群組中核心的連線資訊，用戶端裝置 AWS IoT 的政策必須包含 `greengrass:Discover` 動作。在 `Resource` 區段中，指定用戶端裝置的 ARN，而不是 Greengrass 核心裝置的 ARN。例如：

```
{
  "Effect": "Allow",
  "Action": [
    "greengrass:Discover"
  ],
  "Resource": [
    "arn:aws:iot:region:account-id:thing/device-name"
  ]
}
```

用戶端裝置的 AWS IoT 政策通常不需要 `iot:GetThingShadow`、`iot:UpdateThingShadow` 或 `iot:DeleteThingShadow` 動作的許可，因為 Greengrass 核心會處理用戶端裝置的陰影同步操作。在此情況下，請確定核心 AWS IoT 政策中陰影動作的 `Resource` 區段包含用戶端裝置的 ARNs。

在 AWS IoT 主控台中，您可以檢視和編輯連接到核心憑證的政策。

1. 在導覽窗格的管理下，展開所有裝置，然後選擇物件。
2. 選擇您的核心。
3. 在核心的組態頁面上，選擇憑證索引標籤。
4. 在憑證索引標籤中，選擇您的憑證。
5. 在憑證的組態頁面上，選擇 Policies (政策)，然後選擇政策。

如果您想要編輯政策，請選擇編輯作用中版本。

6. 檢閱政策，並視需要新增、移除或編輯許可。

7. 若要將新的政策版本設定為作用中版本，請在政策版本狀態下，選取將編輯的版本設定為此政策的作用中版本。
8. 選擇儲存為新版本。

的身分和存取管理 AWS IoT Greengrass

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驗證（登入）和授權（具有許可）來使用 AWS IoT Greengrass 資源。IAM 是您可以免費使用 AWS 服務的。

Note

本主題說明 IAM 概念和功能。如需支援的 IAM 功能的相關資訊 AWS IoT Greengrass，請參閱 [the section called “AWS IoT Greengrass 如何使用 IAM”](#)。

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 [對的身分和存取問題進行故障診斷 AWS IoT Greengrass](#))
- 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 Greengrass 如何使用 IAM](#))
- IAM 管理員 — 撰寫政策以管理存取控制 (請參閱 [的身分型政策範例 AWS IoT Greengrass](#))

使用身分驗證

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，請參閱《IAM 使用者指南》中的[工作階段政策](#)。

多種政策類型

當多種類型的政策套用到請求時，產生的許可會更複雜而無法理解。若要了解如何 AWS 在涉及多種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](#)。

另請參閱

- [the section called “AWS IoT Greengrass 如何使用 IAM”](#)
- [the section called “身分型政策範例”](#)
- [the section called “針對識別和存取問題進行故障診斷”](#)

AWS IoT Greengrass 如何使用 IAM

在您使用 IAM 管理對的存取之前 AWS IoT Greengrass，您應該了解可與搭配使用的 IAM 功能 AWS IoT Greengrass。

IAM 功能	受到 Greengrass 支援？
具有資源層級許可的身分型政策	是
資源型政策	否
存取控制清單 (ACL)	否
標籤型授權	否
臨時憑證	是
服務連結角色	否
服務角色	是

如需其他 AWS 服務如何與 IAM 搭配使用的高階檢視，請參閱 [《AWS IAM 使用者指南》中的與 IAM 搭配使用的服務](#)。

的身分型政策 AWS IoT Greengrass

透過 IAM 身分型政策，您可以指定允許或拒絕的動作和資源，以及允許或拒絕動作的條件。AWS IoT Greengrass 支援特定動作、資源和條件索引鍵。若要了解您在政策中使用的所有元素，請參閱 [《IAM 使用者指南》中的 IAM 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。

的政策動作會在動作之前 AWS IoT Greengrass 使用 `greengrass:` 字首。例如，若要允許某人使用 `ListGroups` API 操作來列出群組中的群組 AWS 帳戶，請在其政策中包含 `greengrass:ListGroups` 動作。政策陳述式必須包含 Action 或 NotAction 元素。AWS IoT Greengrass 會定義一組自己的動作，來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，請在括號 ([]) 之間列出它們，並以逗號分隔，如下所示：

```
"Action": [  
    "greengrass:action1",  
    "greengrass:action2",  
    "greengrass:action3"  
]
```

您可以使用萬用字元 (*) 來指定多個動作。例如，若要指定開頭是 `List` 文字的所有動作，請包含以下動作：

```
"Action": "greengrass:List*"
```

Note

建議您避免使用萬用字元來指定服務的所有可用動作。最佳實務是，您應該在政策中授予最低權限和範圍較窄的許可。如需詳細資訊，請參閱[the section called “盡可能授予最低的許可”](#)。

如需 AWS IoT Greengrass 動作的完整清單，請參閱《IAM 使用者指南》中的 [定義的動作 AWS IoT Greengrass](#)。

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": "*"

```

下表包含可在政策陳述式 Resource 元素中使用的 AWS IoT Greengrass 資源 ARNs。如需 AWS IoT Greengrass 動作支援的資源層級許可映射，請參閱《IAM 使用者指南》中的 [定義的動作 AWS IoT Greengrass](#)。

資源	ARN
Group	arn:\${Partition}:greengrass:\${Region}:\${Account}:/ greengrass/groups/\${GroupId}
GroupVersion	arn:\${Partition}:greengrass:\${Region}:\${Account}:/ greengrass/groups/\${GroupId}/versions/\${VersionId}
CertificateAuthority	arn:\${Partition}:greengrass:\${Region}:\${Account}:/ greengrass/groups/\${GroupId}/certificateauthorities/\${CertificateAuthorityId}
Deployment	arn:\${Partition}:greengrass:\${Region}:\${Account}:/ greengrass/groups/\${GroupId}/deployments/\${DeploymentId}
BulkDeployment	arn:\${Partition}:greengrass:\${Region}:\${Account}:/ greengrass/bulk/deployments/\${BulkDeploymentId}
ConnectorDefinition	arn:\${Partition}:greengrass:\${Region}:\${Account}:/ greengrass/definition/connectors/\${ConnectorDefinitionId}
ConnectorDefinitionVersion	arn:\${Partition}:greengrass:\${Region}:\${Account}:/ greengrass/definition/connectors/\${ConnectorDefinitionId}/versions/\${VersionId}
CoreDefinition	arn:\${Partition}:greengrass:\${Region}:\${Account}:/ greengrass/definition/cores/\${CoreDefinitionId}

資源	ARN
CoreDefinitionVersion	arn:\${Partition}:greengrass:\${Region}:\${Account}:/greengrass/definition/cores/\${CoreDefinitionId}/versions/\${VersionId}
DeviceDefinition	arn:\${Partition}:greengrass:\${Region}:\${Account}:/greengrass/definition/devices/\${DeviceDefinitionId}
DeviceDefinitionVersion	arn:\${Partition}:greengrass:\${Region}:\${Account}:/greengrass/definition/devices/\${DeviceDefinitionId}/versions/\${VersionId}
FunctionDefinition	arn:\${Partition}:greengrass:\${Region}:\${Account}:/greengrass/definition/functions/\${FunctionDefinitionId}
FunctionDefinitionVersion	arn:\${Partition}:greengrass:\${Region}:\${Account}:/greengrass/definition/functions/\${FunctionDefinitionId}/versions/\${VersionId}
LoggerDefinition	arn:\${Partition}:greengrass:\${Region}:\${Account}:/greengrass/definition/loggers/\${LoggerDefinitionId}
LoggerDefinitionVersion	arn:\${Partition}:greengrass:\${Region}:\${Account}:/greengrass/definition/loggers/\${LoggerDefinitionId}/versions/\${VersionId}
ResourceDefinition	arn:\${Partition}:greengrass:\${Region}:\${Account}:/greengrass/definition/resources/\${ResourceDefinitionId}
ResourceDefinitionVersion	arn:\${Partition}:greengrass:\${Region}:\${Account}:/greengrass/definition/resources/\${ResourceDefinitionId}/versions/\${VersionId}

資源	ARN
SubscriptionDefinition	arn:\${Partition}:greengrass:\${Region}:\${Account}:/ greengrass/definition/subscriptions/\${SubscriptionDefinitionId}
SubscriptionDefinitionVersion	arn:\${Partition}:greengrass:\${Region}:\${Account}:/ greengrass/definition/subscriptions/\${SubscriptionDefinitionId}/versions/\${VersionId}
ConnectivityInfo	arn:\${Partition}:greengrass:\${Region}:\${Account}:/ greengrass/things/\${ThingName}/connectivityInfo

下列範例Resource元素指定 中美國西部（奧勒岡）區域中群組的 ARN AWS 帳戶 123456789012：

```
"Resource": "arn:aws:greengrass:us-west-2:123456789012:/greengrass/groups/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
```

或者，若要指定屬於特定 AWS 帳戶 的所有群組 AWS 區域，請使用 萬用字元取代群組 ID：

```
"Resource": "arn:aws:greengrass:us-west-2:123456789012:/greengrass/groups/*"
```

某些 AWS IoT Greengrass 動作（例如，某些清單操作）無法對特定資源執行。在這些情況下，您必須使用萬用字元 (*)。

```
"Resource": ""
```

若要在陳述式中指定多個資源 ARNs，請在括號 ([]) 之間列出它們，並以逗號分隔，如下所示：

```
"Resource": [
  "resource-arn1",
  "resource-arn2",
  "resource-arn3"
]
```

如需 ARN 格式的詳細資訊，請參閱 中的 [Amazon Resource Name \(ARNs\) AWS 和服務命名空間](#) Amazon Web Services 一般參考。

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 Greengrass 支援下列全域條件金鑰。

金鑰	Description
aws:CurrentTime	在日期/時間條件中檢查目前日期和時間，以篩選存取權。
aws:EpochTime	在日期/時間條件中檢查以 epoch 或 Unix 時間表示的目前日期和時間，以篩選存取權。
aws:MultiFactorAuthAge	檢查多久以前 (以秒為單位) 在請求中使用 MFA 發出由多重驗證 (MFA) 來驗證的安全登入資料，以篩選存取權。
aws:MultiFactorAuthPresent	檢查是否使用多重驗證 (MFA) 來驗證發出目前請求的臨時安全登入資料，以篩選存取權。
aws:RequestTag/\${TagKey}	根據每個強制標籤的允許值集來篩選建立請求。
aws:ResourceTag/\${TagKey}	根據與資源相關聯的標籤值來篩選動作。
aws:SecureTransport	檢查是否使用 SSL 傳送請求，以篩選存取權。
aws:TagKeys	根據請求中是否存在強制標籤來篩選建立請求。
aws:UserAgent	依請求者的用戶端應用程式來篩選存取權。

如需詳細資訊，請參閱《IAM 使用者指南》中的[AWS 全域條件內容金鑰](#)。

範例

若要檢視 AWS IoT Greengrass 身分型政策的範例，請參閱 [the section called “身分型政策範例”](#)。

的資源型政策 AWS IoT Greengrass

AWS IoT Greengrass 不支援[以資源為基礎的政策](#)。

存取控制清單 (ACL)

AWS IoT Greengrass 不支援 [ACLs](#)。

以 AWS IoT Greengrass 標籤為基礎的授權

AWS IoT Greengrass 不支援標籤型授權。

的 IAM 角色 AWS IoT Greengrass

[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 Greengrass

臨時登入資料用於使用聯合身分登入、擔任 IAM 角色，或擔任跨帳戶角色。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。

在 Greengrass 核心上，[群組角色](#)的臨時登入資料可供使用者定義的 Lambda 函數和連接器使用。如果您的 Lambda 函數使用 AWS SDK，您不需要新增邏輯來取得登入資料，因為 AWS SDK 會為您執行此操作。

服務連結角色

AWS IoT Greengrass 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 Greengrass 使用服務角色代表您存取您的部分 AWS 資源。如需詳細資訊，請參閱[the section called “Greengrass 服務角色”](#)。

在 AWS IoT Greengrass 主控台中選擇 IAM 角色

在 AWS IoT Greengrass 主控台中，您可能需要從帳戶中的 IAM 角色清單中選擇 Greengrass 服務角色或 Greengrass 群組角色。

- Greengrass 服務角色允許 代表您 AWS IoT Greengrass 存取其他 服務中的 AWS 資源。一般而言，您不需要選擇服務角色，因為主控台可以為您建立和設定服務角色。如需詳細資訊，請參閱[the section called “Greengrass 服務角色”](#)。
- Greengrass 群組角色用於允許群組中的 Greengrass Lambda 函數和連接器存取您的 AWS 資源。它也可以授予 AWS IoT Greengrass 許可，將串流匯出至 AWS 服務並寫入 CloudWatch 日誌。如需詳細資訊，請參閱[the section called “Greengrass 群組角色”](#)。

Greengrass 服務角色

Greengrass 服務角色是 AWS Identity and Access Management (IAM) 服務角色，授權 代表您 AWS IoT Greengrass 存取來自 AWS 服務的資源。這可讓 AWS IoT Greengrass 執行基本任務，例如擷取 AWS Lambda 函數和管理 AWS IoT 陰影。

若要允許 AWS IoT Greengrass 存取您的 資源，Greengrass 服務角色必須與 相關聯，AWS 帳戶 並指定 AWS IoT Greengrass 為信任的實體。此角色必須包含 [AWSGreengrassResourceAccessRolePolicy](#) 受管政策或自訂政策，以定義您所使用的 AWS IoT Greengrass 功能的同等許可。此政策由 AWS 維護，並定義 AWS IoT Greengrass 用於存取 AWS 資源的一組許可。

您可以在 之間重複使用相同的 Greengrass AWS 區域服務角色，但您必須在您使用的每個 AWS 區域中將其與您的帳戶建立關聯 AWS IoT Greengrass。如果目前 AWS 帳戶 和 區域中不存在服務角色，則群組部署會失敗。

下列各節說明如何在 AWS 管理主控台 或 中建立和管理 Greengrass 服務角色 AWS CLI。

- [管理服務角色 \(主控台\)](#)
- [管理服務角色 \(CLI\)](#)

Note

除了授權服務層級存取的服務角色之外，您還可以將群組角色指派給 AWS IoT Greengrass 群組。群組角色是獨立的 IAM 角色，可控制群組中的 Greengrass Lambda 函數和連接器如何存取 AWS 服務。

管理 Greengrass 服務角色 (主控台)

AWS IoT 主控台可讓您輕鬆管理您的 Greengrass 服務角色。例如，當您建立或部署 Greengrass 群組時，主控台會檢查是否 AWS 帳戶 連接到主控台中目前選取的 中的 Greengrass AWS 區域 服務角色。如果未連接，主控台可以為您建立和設定服務角色。如需詳細資訊，請參閱[the section called “建立 Greengrass 服務角色”](#)。

您可以使用 AWS IoT 主控台執行下列角色管理任務：

- [尋找您的 Greengrass 服務角色](#)
- [建立 Greengrass 服務角色](#)
- [變更 Greengrass 服務角色](#)
- [分離 Greengrass 服務角色](#)

Note

登入主控台的使用者必須擁有可檢視、建立或變更服務角色的許可。

尋找您的 Greengrass 服務角色 (主控台)

使用下列步驟來尋找 AWS IoT Greengrass 目前 中使用的服務角色 AWS 區域。

1. 從[AWS IoT 主控台](#)導覽窗格中，選擇設定。
2. 捲動至 Greengrass service role (Greengrass 服務角色) 區段，查看您的服務角色及其政策。

如果您沒有看到服務角色，您可以讓主控台為您建立或設定服務角色。如需詳細資訊，請參閱[建立 Greengrass 服務角色](#)。

建立 Greengrass 服務角色 (主控台)

主控台可以為您建立和設定預設的 Greengrass 服務角色。這個角色具有以下屬性：

屬性	Value
名稱	Greengrass_ServiceRole
信任實體	AWS service: greengrass
政策	AWSGreengrassResourceAccessRolePolicy (AWSGreengrassResourceAccessRolePolicy)

Note

如果 [Greengrass 裝置安裝程式](#) 建立了服務角色，則角色名稱為 GreengrassServiceRole_*random-string*。

當您從 AWS IoT 主控台建立或部署 Greengrass 群組時，主控台會在目前在主控台中選取的 AWS 帳戶 AWS 區域中，檢查 Greengrass 服務角色是否與相關聯。如果沒有，主控台會提示您允許代表您 AWS IoT Greengrass 讀取和寫入 AWS 服務。

如果您授予許可，主控台會檢查 Greengrass_ServiceRole 中是否存在名為的角色 AWS 帳戶。

- 如果角色存在，主控台會將服務角色連接到目前 AWS 帳戶中的 AWS 區域。
- 如果角色不存在，主控台會建立預設 Greengrass 服務角色，並將其連接到目前 AWS 帳戶中的 AWS 區域。

Note

如果您想要使用自訂角色政策建立服務角色，請使用 IAM 主控台來建立或修改角色。如需詳細資訊，請參閱 [《IAM 使用者指南》中的建立角色以將許可委派給 AWS 服務或修改角色](#)。請確定角色會授與與您所使用功能和資源之 AWSGreengrassResourceAccessRolePolicy 受管政策相同的許可。我們建議您也在信任政策中包含 `aws:SourceArn` 和 `aws:SourceAccount` 全域條件內容金鑰，以協助防止混淆代理人安全問題。條件內容索引鍵會限制存取，只允許來自指定帳戶和 Greengrass 工作區的這些請求。如需有關混淆代理人問題的詳細資訊，請參閱 [預防跨服務混淆代理人](#)。

如果您建立服務角色，請返回 AWS IoT 主控台並將角色連接到群組。您可以在群組的設定頁面上的 Greengrass 服務角色下執行此操作。

變更 Greengrass 服務角色 (主控台)

使用下列程序選擇不同的 Greengrass 服務角色，以在主控台 AWS 區域 目前選取的 AWS 帳戶 中連接至您的 。

1. 從[AWS IoT 主控台](#)導覽窗格中，選擇設定。
2. 在 Greengrass 服務角色下，選擇變更角色。

更新 Greengrass 服務角色對話方塊隨即開啟，並顯示 中定義為 AWS IoT Greengrass 信任實體 AWS 帳戶 的 IAM 角色。

3. 選擇要連接的 Greengrass 服務角色。
4. 選擇連接角色。

Note

若要允許主控台為您建立預設的 Greengrass 服務角色，請選擇 Create role for me (為我建立角色)，而不要從清單中選擇角色。如果名為 的角色位於您的 中，則不會顯示為我建立角色連結 AWS 帳戶。Greengrass_ServiceRole

分離 Greengrass 服務角色 (主控台)

使用下列程序，在主控台 AWS 區域 目前選取的 AWS 帳戶 中，從 分離 Greengrass 服務角色。這會撤銷 AWS IoT Greengrass 存取目前 AWS 服務的權限 AWS 區域。

Important

分離服務角色可能會中斷作用中的操作。

1. 從[AWS IoT 主控台](#)導覽窗格中，選擇設定。
2. 在 Greengrass 服務角色下，選擇分離角色。
3. 在確認對話方塊中，選擇 Detach (分離)。

Note

如果您不再需要該角色，您可以在 IAM 主控台中刪除該角色。如需詳細資訊，請參閱 IAM 使用者指南中的[刪除角色或執行個體描述檔](#)。

其他角色可能允許 AWS IoT Greengrass 存取您的資源。若要尋找允許代表您 AWS IoT Greengrass 擔任許可的所有角色，請在 IAM 主控台的角色頁面上，尋找包含 AWS 服務的角色：信任實體欄中的 greengrass。

管理 Greengrass 服務角色 (CLI)

在下列程序中，我們假設 AWS CLI 已安裝並設定為使用您的 AWS 帳戶 ID。如需詳細資訊，請參閱 AWS Command Line Interface 《使用者指南》中的[安裝 AWS 命令列界面](#)和[設定 AWS CLI](#)。

您可以使用 AWS CLI 執行下列角色管理任務：

- [取得您的 Greengrass 服務角色](#)
- [建立 Greengrass 服務角色](#)
- [移除 Greengrass 服務角色](#)

取得 Greengrass 服務角色 (CLI)

使用下列程序來了解 Greengrass 服務角色是否與 AWS 帳戶中的相關聯 AWS 區域。

- 取得服務角色。將##取代為您的 AWS 區域（例如，us-west-2）。

```
aws Greengrass get-service-role-for-account --region region
```

如果 Greengrass 服務角色已與您的帳戶關聯，會傳回下列的角色中繼資料。

```
{
  "AssociatedAt": "timestamp",
  "RoleArn": "arn:aws:iam::account-id:role/path/role-name"
}
```

如果未傳回任何角色中繼資料，則必須建立服務角色（如果不存在），並將其與中的帳戶建立關聯 AWS 區域。

建立 Greengrass 服務角色 (CLI)

使用下列步驟來建立角色，並將其與您的 建立關聯 AWS 帳戶。

使用 IAM 建立服務角色

1. 使用允許 AWS IoT Greengrass 擔任角色的信任政策來建立角色。此範例會建立名為 Greengrass_ServiceRole 的角色，但您可以使用不同的名稱。我們建議您也在信任政策中包含 `aws:SourceArn` 和 `aws:SourceAccount` 全域條件內容金鑰，以協助防止混淆代理人安全問題。條件內容索引鍵會限制存取，只允許來自指定帳戶和 Greengrass 工作區的這些請求。如需有關混淆代理人問題的詳細資訊，請參閱 [預防跨服務混淆代理人](#)。

Linux, macOS, or Unix

```
aws iam create-role --role-name Greengrass_ServiceRole --assume-role-policy-document '{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "greengrass.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "account-id"
        },
        "ArnLike": {
          "aws:SourceArn": "arn:aws:greengrass:region:account-id:*"
        }
      }
    }
  ]
}'
```

Windows command prompt

```
aws iam create-role --role-name Greengrass_ServiceRole --assume-role-policy-document "{ \"Version\": \"2012-10-17\", \"Statement\": [{ \"Effect
```

```
\":\\"Allow\\",\\"Principal\\":{\\"Service\\":\\"greengrass.amazonaws.com\\"},
\\"Action\\":\\"sts:AssumeRole\\",\\"Condition\\":{\\"ArnLike\\":{\\"aws:SourceArn
\\":\\"arn:aws:greengrass:region:account-id:*\\"},\\"StringEquals\\":
{\\"aws:SourceAccount\\":\\"account-id\\"}}}}]"
```

2. 從輸出中的角色中繼資料，複製角色 ARN。您使用 ARN 將角色與您的帳戶相關聯。
3. 將 AWSGreengrassResourceAccessRolePolicy 政策連接到角色。

```
aws iam attach-role-policy --role-name Greengrass_ServiceRole --policy-arn
arn:aws:iam::aws:policy/service-role/AWSGreengrassResourceAccessRolePolicy
```

將服務角色與 建立關聯 AWS 帳戶

- 將角色與您的帳戶相關聯。將 *role-arn* 取代為服務角色 ARN，並將 *region* 取代為 your AWS 區域（例如，us-west-2）。

```
aws greengrass associate-service-role-to-account --role-arn role-arn --
region region
```

如果成功，會傳回下列回應。

```
{
  "AssociatedAt": "timestamp"
}
```

移除 Greengrass 服務角色 (CLI)

使用下列步驟取消 Greengrass 服務角色與 的關聯 AWS 帳戶。

- 將服務角色與您的帳戶取消關聯。將 *##* 取代為您的 AWS 區域（例如，us-west-2）。

```
aws greengrass disassociate-service-role-from-account --region region
```

如果成功，會傳回下列回應。

```
{
  "DisassociatedAt": "timestamp"
}
```

```
}
```

Note

如果您未在任何 中使用服務角色，您應該將其刪除 AWS 區域。首先，使用 [delete-role-policy](#) 將 AWSGreengrassResourceAccessRolePolicy 受管政策從角色中分離，然後使用 [delete-role](#) 來刪除角色。如需詳細資訊，請參閱 IAM 使用者指南中的[刪除角色或執行個體描述檔](#)。

另請參閱

- 《IAM 使用者指南》中的[建立角色以將許可委派給 AWS 服務](#)
- IAM 使用者指南中的[修改角色](#)
- 《IAM 使用者指南》中的[刪除角色或執行個體描述檔](#)
- AWS IoT Greengrass 命令AWS CLI 參考中的 命令
 - [associate-service-role-to-account](#)
 - [disassociate-service-role-from-account](#)
 - [get-service-role-for-account](#)
- 《命令AWS CLI 參考》中的 IAM 命令
 - [attach-role-policy](#)
 - [create-role](#)
 - [delete-role](#)
 - [delete-role-policy](#)

Greengrass 群組角色

Greengrass 群組角色是一種 IAM 角色，可授權在 Greengrass 核心上執行的程式碼存取您的 AWS 資源。您可以在 AWS Identity and Access Management (IAM) 中建立角色和管理許可，並將角色連接到 Greengrass 群組。Greengrass 群組有一個群組角色。若要新增或變更許可，您可以連接不同的角色或變更連接到角色的 IAM 政策。

角色必須 AWS IoT Greengrass 定義為信任的實體。根據您的業務案例，群組角色可能包含定義下列項目的 IAM 政策：

- 使用者定義 [Lambda 函數](#) 存取 AWS 服務的許可。
- [連接器](#) 存取 AWS 服務的許可。
- [串流管理員](#) 將串流匯出至 AWS IoT Analytics 和 Kinesis Data Streams 的許可。
- 允許 [CloudWatch 記錄](#) 的許可。

下列各節說明如何在 AWS 管理主控台 或 中連接或分離 Greengrass 群組角色 AWS CLI。

- [管理群組角色 \(主控台\)](#)
- [管理群組角色 \(CLI\)](#)

 Note

除了授權從 Greengrass 核心存取的群組角色之外，您還可以指派允許 代表您 AWS IoT Greengrass 存取 AWS 資源的 [Greengrass 服務角色](#)。

管理 Greengrass 群組角色 (主控台)

您可以使用 AWS IoT 主控台執行下列角色管理任務：

- [尋找您的 Greengrass 群組角色](#)
- [新增或變更 Greengrass 群組角色](#)
- [移除 Greengrass 群組角色](#)

 Note

登入主控台的使用者必須具有管理角色的許可。

尋找您的 Greengrass 群組角色 (主控台)

請依照下列步驟尋找連接到 Greengrass 群組的角色。

1. 在 AWS IoT 主控台導覽窗格的管理下，展開 Greengrass 裝置，然後選擇群組 (V1)。

2. 選擇目標群組。
3. 在群組組態頁面上，選擇檢視設定。

如果角色已連接至群組，則會顯示在群組角色下。

新增或變更 Greengrass 群組角色 (主控台)

請依照下列步驟，從 中選擇 AWS 帳戶 要新增至 Greengrass 群組的 IAM 角色。

群組角色具有下列需求：

- AWS IoT Greengrass 定義為信任的實體。
- 連接到角色的許可政策必須將許可授予群組中 Lambda 函數和連接器以及 Greengrass 系統元件所需的 AWS 資源。

Note

我們建議您也在信任政策中包含 `aws:SourceArn` 和 `aws:SourceAccount` 全域條件內容金鑰，以協助防止混淆代理人安全問題。條件內容索引鍵會限制存取，只允許來自指定帳戶和 Greengrass 工作區的請求。如需有關混淆代理人問題的詳細資訊，請參閱 [預防跨服務混淆代理人](#)。

使用 IAM 主控台來建立和設定角色及其許可。如需建立允許存取 Amazon DynamoDB 資料表的範例角色的步驟，請參閱 [the section called “設定群組角色”](#)。如需一般步驟，請參閱《IAM 使用者指南》中的 [為 AWS 服務建立角色 \(主控台\)](#)。

設定角色之後，請使用 AWS IoT 主控台將角色新增至群組。

Note

只有為群組選擇角色時，才需要此程序。變更目前選取之群組角色的許可後，不需要此程序。

1. 在 AWS IoT 主控台導覽窗格的管理下，展開 Greengrass 裝置，然後選擇群組 (V1)。

2. 選擇目標群組。
3. 在群組組態頁面上，選擇檢視設定。
4. 在群組角色下，選擇新增或變更角色：
 - 若要新增角色，請選擇關聯角色，然後從角色清單中選擇您的角色。這些是您 中 AWS IoT Greengrass 將 AWS 帳戶 定義為信任實體的角色。
 - 若要選擇不同的角色，請選擇編輯角色，然後從角色清單中選擇您的角色。
5. 選擇儲存。

移除 Greengrass 群組角色 (主控台)

請依照下列步驟來分開 Greengrass 群組的角色。

1. 在 AWS IoT 主控台導覽窗格的管理下，展開 Greengrass 裝置，然後選擇群組 (V1)。
2. 選擇目標群組。
3. 在群組組態頁面上，選擇檢視設定。
4. 在群組角色下，選擇取消關聯角色。
5. 在確認對話方塊中，選擇取消關聯角色。此步驟會從群組移除角色，但不會刪除角色。如果您想要刪除角色，請使用 IAM 主控台。

管理 Greengrass 群組角色 (CLI)

您可以使用 AWS CLI 執行下列角色管理任務：

- [取得您的 Greengrass 群組角色](#)
- [建立 Greengrass 群組角色](#)
- [移除 Greengrass 群組角色](#)

取得 Greengrass 群組角色 (CLI)

請依照下列步驟來了解 Greengrass 群組是否具有相關聯的角色。

1. 從群組清單中取得目標群組的 ID。

```
aws greengrass list-groups
```

以下是 `list-groups` 回應範例。回應中的每個群組都包含 `Id` 屬性。當中包含群組 ID。

```
{
  "Groups": [
    {
      "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/groups/00dedaaa-ac16-484d-ad77-c3eedEXAMPLE/versions/4cbc3f07-fc5e-48c4-a50e-7d356EXAMPLE",
      "Name": "MyFirstGroup",
      "LastUpdatedTimestamp": "2019-11-11T05:47:31.435Z",
      "LatestVersion": "4cbc3f07-fc5e-48c4-a50e-7d356EXAMPLE",
      "CreationTimestamp": "2019-11-11T05:47:31.435Z",
      "Id": "00dedaaa-ac16-484d-ad77-c3eedEXAMPLE",
      "Arn": "arn:aws:us-west-2:123456789012:/greengrass/groups/00dedaaa-ac16-484d-ad77-c3eedEXAMPLE"
    },
    {
      "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE/versions/8fe9e8ec-64d1-4647-b0b0-01dc8EXAMPLE",
      "Name": "GreenhouseSensors",
      "LastUpdatedTimestamp": "2020-01-07T19:58:36.774Z",
      "LatestVersion": "8fe9e8ec-64d1-4647-b0b0-01dc8EXAMPLE",
      "CreationTimestamp": "2020-01-07T19:58:36.774Z",
      "Id": "036ceaf9-9319-4716-ba2a-237f9EXAMPLE",
      "Arn": "arn:aws:us-west-2:123456789012:/greengrass/groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE"
    },
    ...
  ]
}
```

如需詳細資訊，包括使用 `query` 選項篩選結果的範例，請參閱[the section called “取得群組 ID”](#)。

2. 從輸出複製目標群組的 `Id`。
3. 取得群組角色。以目標群組的 ID 取代 *group-id*。

```
aws greengrass get-associated-role --group-id group-id
```

如果角色與您的 Greengrass 群組相關聯，則會傳回下列角色中繼資料。

```
{
  "AssociatedAt": "timestamp",
  "RoleArn": "arn:aws:iam::account-id:role/path/role-name"
}
```

如果您的群組沒有相關聯的角色，則會傳回下列錯誤。

```
An error occurred (404) when calling the GetAssociatedRole operation: You need to
attach an IAM role to this deployment group.
```

建立 Greengrass 群組角色 (CLI)

請依照下列步驟建立角色，並將其與 Greengrass 群組產生關聯。

使用 IAM 建立群組角色

1. 使用允許 AWS IoT Greengrass 擔任角色的信任政策來建立角色。此範例會建立名為 MyGreengrassGroupRole 的角色，但您可以使用不同的名稱。我們建議您也在信任政策中包含 aws:SourceArn 和 aws:SourceAccount 全域條件內容金鑰，以協助防止混淆代理人安全問題。條件內容索引鍵會限制存取，只允許來自指定帳戶和 Greengrass 工作區的請求。如需有關混淆代理人問題的詳細資訊，請參閱 [預防跨服務混淆代理人](#)。

Linux, macOS, or Unix

```
aws iam create-role --role-name MyGreengrassGroupRole --assume-role-policy-
document '{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "greengrass.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
```

```

        "aws:SourceAccount": "account-id"
      },
      "ArnLike": {
        "aws:SourceArn": "arn:aws:greengrass:region:account-id:/greengrass/
groups/group-id"
      }
    }
  ]
}'

```

Windows command prompt

```

aws iam create-role --role-name MyGreengrassGroupRole --assume-role-policy-
document "{\"Version\":\"2012-10-17\", \"Statement\": [{\"Effect
\": \"Allow\", \"Principal\": {\"Service\": \"greengrass.amazonaws.com\"},
\"Action\": \"sts:AssumeRole\", \"Condition\": {\"ArnLike\": {\"aws:SourceArn
\": \"arn:aws:greengrass:region:account-id:/greengrass/groups/group-id\"},
\"StringEquals\": {\"aws:SourceAccount\": \"account-id\"}}}]}"

```

2. 從輸出中的角色中繼資料，複製角色 ARN。您使用 ARN 將角色與您的帳戶相關聯。
3. 將受管或內嵌政策連接至角色，以支援您的商業案例。例如，如果使用者定義的 Lambda 函數從 Amazon S3 讀取，您可以將 AmazonS3ReadOnlyAccess 受管政策連接至角色。

```

aws iam attach-role-policy --role-name MyGreengrassGroupRole --policy-arn
arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess

```

如果成功，則不會傳回任何回應。

將角色與您的 Greengrass 群組建立關聯

1. 從群組清單中取得目標群組的 ID。

```

aws greengrass list-groups

```

以下是 list-groups 回應範例。回應中的每個群組都包含 Id 屬性。當中包含群組 ID。

```

{

```

```

    "Groups": [
      {
        "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/00dedaaa-ac16-484d-ad77-c3eedEXAMPLE/versions/4cbc3f07-fc5e-48c4-
a50e-7d356EXAMPLE",
        "Name": "MyFirstGroup",
        "LastUpdatedTimestamp": "2019-11-11T05:47:31.435Z",
        "LatestVersion": "4cbc3f07-fc5e-48c4-a50e-7d356EXAMPLE",
        "CreationTimestamp": "2019-11-11T05:47:31.435Z",
        "Id": "00dedaaa-ac16-484d-ad77-c3eedEXAMPLE",
        "Arn": "arn:aws:us-west-2:123456789012:/greengrass/groups/00dedaaa-
ac16-484d-ad77-c3eedEXAMPLE"
      },
      {
        "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE/versions/8fe9e8ec-64d1-4647-
b0b0-01dc8EXAMPLE",
        "Name": "GreenhouseSensors",
        "LastUpdatedTimestamp": "2020-01-07T19:58:36.774Z",
        "LatestVersion": "8fe9e8ec-64d1-4647-b0b0-01dc8EXAMPLE",
        "CreationTimestamp": "2020-01-07T19:58:36.774Z",
        "Id": "036ceaf9-9319-4716-ba2a-237f9EXAMPLE",
        "Arn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE"
      },
      ...
    ]
  }

```

如需詳細資訊，包括使用 query 選項篩選結果的範例，請參閱[the section called “取得群組 ID”](#)。

2. 從輸出複製目標群組的 Id。
3. 將角色與叢集相關聯。以目標群組的 ID 取代 *group-id* 並以群組角色的 ARN 取代 *role-arn*。

```
aws greengrass associate-role-to-group --group-id group-id --role-arn role-arn
```

如果成功，會傳回下列回應。

```

{
  "AssociatedAt": "timestamp"
}

```

移除 Greengrass 群組角色 (CLI)

請依照下列步驟來取消群組角色與 Greengrass 群組的關聯。

1. 從群組清單中取得目標群組的 ID。

```
aws greengrass list-groups
```

以下是 list-groups 回應範例。回應中的每個群組都包含 Id 屬性。當中包含群組 ID。

```
{
  "Groups": [
    {
      "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/00dedaaa-ac16-484d-ad77-c3eedEXAMPLE/versions/4cbc3f07-fc5e-48c4-
a50e-7d356EXAMPLE",
      "Name": "MyFirstGroup",
      "LastUpdatedTimestamp": "2019-11-11T05:47:31.435Z",
      "LatestVersion": "4cbc3f07-fc5e-48c4-a50e-7d356EXAMPLE",
      "CreationTimestamp": "2019-11-11T05:47:31.435Z",
      "Id": "00dedaaa-ac16-484d-ad77-c3eedEXAMPLE",
      "Arn": "arn:aws:us-west-2:123456789012:/greengrass/groups/00dedaaa-
ac16-484d-ad77-c3eedEXAMPLE"
    },
    {
      "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE/versions/8fe9e8ec-64d1-4647-
b0b0-01dc8EXAMPLE",
      "Name": "GreenhouseSensors",
      "LastUpdatedTimestamp": "2020-01-07T19:58:36.774Z",
      "LatestVersion": "8fe9e8ec-64d1-4647-b0b0-01dc8EXAMPLE",
      "CreationTimestamp": "2020-01-07T19:58:36.774Z",
      "Id": "036ceaf9-9319-4716-ba2a-237f9EXAMPLE",
      "Arn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE"
    },
    ...
  ]
}
```

如需詳細資訊，包括使用 `query` 選項篩選結果的範例，請參閱[the section called “取得群組 ID”](#)。

2. 從輸出複製目標群組的 Id。
3. 取消角色與群組的關聯。以目標群組的 ID 取代 *group-id*。

```
aws greengrass disassociate-role-from-group --group-id group-id
```

如果成功，會傳回下列回應。

```
{
  "DisassociatedAt": "timestamp"
}
```

Note

如果您不使用群組角色，可以將其刪除。首先，使用 [delete-role-policy](#) 將每個受管政策從角色中分開，然後使用 [delete-role](#) 來刪除角色。如需詳細資訊，請參閱 IAM 使用者指南中的[刪除角色或執行個體描述檔](#)。

另請參閱

- IAM 使用者指南中的相關主題
 - [建立角色以將許可委派給 AWS 服務](#)
 - [修改角色](#)
 - [新增和移除 IAM 身分許可](#)
 - [刪除角色或執行個體描述檔](#)
- AWS IoT Greengrass 命令AWS CLI 參考中的 命令
 - [list-groups](#)
 - [associate-role-to-group](#)
 - [disassociate-role-from-group](#)
 - [get-associated-role](#)
- 《命令AWS CLI 參考》中的 IAM 命令
 - [attach-role-policy](#)
 - [create-role](#)

- [delete-role](#)
- [delete-role-policy](#)

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許可的實體執行該動作。在 AWS 中，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。

我們建議在資源政策中使用 [aws:SourceArn](#) 和 [aws:SourceAccount](#) 全域條件內容索引鍵，以限制將另一個服務 AWS IoT Greengrass 提供給資源的許可。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。

的值 `aws:SourceArn` 必須是與 `sts:AssumeRole` 請求相關聯的 Greengrass 客戶資源。

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:greengrass:region:account-id:*`。

如需使用 `aws:SourceArn` 和 `aws:SourceAccount` 全域條件內容索引鍵的政策範例，請參閱下列主題：

- [建立 Greengrass 服務角色](#)
- [建立 Greengrass 群組角色](#)
- [為大量部署建立和設定 IAM 執行角色](#)

的身分型政策範例 AWS IoT Greengrass

根據預設，IAM 使用者和角色不具備建立或修改 AWS IoT Greengrass 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 作業的所需許可。管理員接著必須將這些政策連接至需要這些許可的 IAM 使用者或群組。

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 Greengrass 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](#)。

AWS 的 受管政策 AWS IoT Greengrass

AWS IoT Greengrass 會維護下列 AWS 受管政策，您可以使用這些政策將許可授予 IAM 使用者和角色。

政策	Description
AWSGreengrassFullAccess (AWSGreengrassFullAccess)	允許所有 AWS 資源的所有 AWS IoT Greengrass 動作。建議將此政策用於 AWS IoT Greengrass 服務管理員 或測試用途。

政策	Description
AWSGreengrassReadOnlyAccess	允許所有 AWS 資源的 List 和 Get AWS IoT Greengrass 動作。
AWSGreengrassResourceAccessRolePolicy (AWSGreengrassResourceAccessRolePolicy)	允許從 AWS 服務存取 資源，包括 AWS Lambda 和 AWS IoT Device Shadow。這是用於 Greengrass 服務角色 的預設政策。這項政策是為了方便存取而設計。您可以定義更嚴格的自訂政策。
GreengrassOTAUpdateArtifactAccess	允許唯讀存取所有 中 AWS IoT Greengrass Core 軟體的 over-the-air (OTA) AWS 區域更新成品。

政策範例

下列客戶定義的政策範例會授予常見案例的許可。

範例

- [允許使用者檢視他們自己的許可](#)

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 標籤上建立政策](#)。

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": "*"
}
]
}

```

對的身分和存取問題進行故障診斷 AWS IoT Greengrass

使用以下資訊來協助您診斷和修正使用 AWS IoT Greengrass 和 IAM 時可能遇到的常見問題。

問題

- [我未獲授權在中執行動作 AWS IoT Greengrass](#)
- [錯誤：Greengrass 未獲授權，無法擔任與此帳戶關聯的服務角色，或錯誤：失敗：TES 服務角色並未與此帳戶關聯。](#)
- [錯誤：嘗試使用角色 `arn:aws:iam::<account-id>:role/<role-name>` 存取 s3 url `https://<region>-greengrass-updates.s3.<region>.amazonaws.com/core/<architecture>/greengrass-core-<distribution-version>.tar.gz` 時，許可遭拒。](#)
- [裝置陰影與雲端不同步。](#)
- [我未獲授權執行 `iam:PassRole`](#)
- [我是管理員，想要允許其他人存取 AWS IoT Greengrass](#)
- [我想要允許以外的人員 AWS 帳戶 存取我的 AWS IoT Greengrass 資源](#)

如需一般的故障診斷協助，請參閱 [疑難排解](#)。

我未獲授權在 中執行動作 AWS IoT Greengrass

若您收到指出您未獲授權執行動作的錯誤，您必須聯絡管理員以取得協助。您的管理員是提供您使用者名稱和密碼的人員。

當 IAM mateojackson 使用者嘗試檢視核心定義版本的詳細資訊，但沒有 greengrass:GetCoreDefinitionVersion 許可時，會發生下列範例錯誤。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
greengrass:GetCoreDefinitionVersion on resource: resource: arn:aws:greengrass:us-
west-2:123456789012:/greengrass/definition/cores/78cd17f3-bc68-ee18-47bd-5bda5EXAMPLE/
versions/368e9ffa-4939-6c75-859c-0bd4cEXAMPLE
```

在此情況下，Mateo 會請求管理員更新他的政策，允許他使用 arn:aws:greengrass:us-west-2:123456789012:/greengrass/definition/cores/78cd17f3-bc68-ee18-47bd-5bda5EXAMPLE/versions/368e9ffa-4939-6c75-859c-0bd4cEXAMPLE 動作存取 greengrass:GetCoreDefinitionVersion 資源。

錯誤：Greengrass 未獲授權，無法擔任與此帳戶關聯的服務角色，或**錯誤：**失敗：TES 服務角色並未與此帳戶關聯。

解決方案：部署失敗時，您可能會看到此錯誤。檢查 Greengrass 服務角色是否與目前 AWS 帳戶中的相關聯 AWS 區域。如需詳細資訊，請參閱 [the section called “管理服務角色 \(CLI\)”](#) 或 [the section called “管理服務角色 \(主控台\)”](#)。

錯誤：嘗試使用角色 arn:aws:iam::<account-id>:role/<role-name> 存取 s3 url https://<region>-greengrass-updates.s3.<region>.amazonaws.com/core/<architecture>/greengrass-core-<distribution-version>.tar.gz 時，許可遭拒。

解決方案：您可以在無線 (OTA) 更新失敗時查看此錯誤。在簽署者角色政策中，將目標新增 AWS 區域為 Resource。此簽署者角色用於預先簽署 S3 URL 以進行 AWS IoT Greengrass 軟體更新。如需詳細資訊，請參閱 [S3 URL 簽署者角色](#)。

裝置陰影與雲端不同步。

解決方案：確定 AWS IoT Greengrass 具有 [Greengrass 服務角色](#) 中 iot:UpdateThingShadow 和 iot:GetThingShadow 動作的許可。如果服務角色使用 AWSGreengrassResourceAccessRolePolicy 受管政策，預設會包含這些權限。

請參閱 [陰影同步逾時問題故障診斷](#)。

以下是使用時可能遇到的一般 IAM 問題 AWS IoT Greengrass。

我未獲授權執行 iam:PassRole

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 Greengrass。

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 Greengrass 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。

我是管理員，想要允許其他人存取 AWS IoT Greengrass

若要允許其他人存取 AWS IoT Greengrass，您必須將許可授予需要存取的人員或應用程式。如果您使用 AWS IAM Identity Center 管理人員和應用程式，您可以將許可集指派給使用者或群組，以定義其存取層級。許可集會自動建立 IAM 政策，並將其指派給與該人員或應用程式相關聯的 IAM 角色。如需詳細資訊，請參閱 AWS IAM Identity Center 《使用者指南》中的 [許可集](#)。

如果您不是使用 IAM Identity Center，則必須為需要存取的人員或應用程式建立 IAM 實體（使用者或角色）。您接著必須將政策連接到實體，在 AWS IoT Greengrass 中授予他們正確的許可。授予許可後，請將登入資料提供給使用者或應用程式開發人員。他們將使用這些登入資料來存取 AWS。若要進一步了解如何建立 IAM 使用者、群組、政策和許可，請參閱《IAM [使用者指南](#)》中的 [IAM 身分和政策](#)和 [許可](#)。

我想要允許以外的人員 AWS 帳戶 存取我的 AWS IoT Greengrass 資源

您可以建立 IAM 角色，讓其他帳戶中的使用者或您組織外部的人員可用來存取您的 AWS 資源。您可以指定要允許哪些信任對象擔任該角色。如需詳細資訊，請參閱《IAM [使用者指南](#)》中的 [提供存取權給您擁有 AWS 帳戶 的另一個 中的 IAM 使用者](#)，以及 [提供存取權給第三方擁有的 Amazon Web Services 帳戶](#)。

AWS IoT Greengrass 不支援以資源型政策或存取控制清單 (ACLs) 為基礎的跨帳戶存取。

的合規驗證 AWS IoT Greengrass

若要了解 AWS 服務 是否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[下載報告 in 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](#)。

中的彈性 AWS IoT Greengrass

AWS 全球基礎設施是以 Amazon Web Services 區域和可用區域為基礎。每個 AWS 區域 提供多個實體分隔和隔離的可用區域，這些可用區域以低延遲、高輸送量和高度備援的聯網連接。透過可用區域，您可以設計與操作的應用程式和資料庫，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、容錯能力和擴展能力，均較單一或多個資料中心的傳統基礎設施還高。

如需 Amazon Web Services 區域和可用區域的詳細資訊，請參閱 [AWS 全球基礎設施](#)。

除了 AWS 全球基礎設施之外，AWS IoT Greengrass 還提供數種功能，以協助支援您的資料彈性和備份需求。

- 如果核心失去網際網路連線，用戶端裝置可以繼續透過本機網路進行通訊。
- 您可以設定核心，將目標為 AWS 雲端 目標的未處理訊息存放在本機儲存快取中，而非記憶體內儲存。本機儲存快取可以在整個核心重新啟動期間持續存在（例如，在群組部署或裝置重新啟動之後），因此 AWS IoT Greengrass 可以繼續處理目標為 的訊息 AWS IoT Core。如需詳細資訊，請參閱[the section called “MQTT 訊息佇列”](#)。
- 您可以設定核心，以與 AWS IoT Core 訊息中介裝置建立持久性工作階段。這可讓核心在核心離線時接收傳送的訊息。如需詳細資訊，請參閱[the section called “使用的 MQTT 持久性工作階段 AWS IoT Core”](#)。
- 您可以設定 Greengrass 群組，將日誌寫入本機檔案系統和 CloudWatch Logs。如果核心失去連線，本機記錄可以繼續，但 CloudWatch 日誌的傳送重試次數有限。用完重試次數之後，會刪除事件。您也應該注意[記錄限制](#)。
- 您可以編寫 Lambda 函數，讀取[串流管理員](#)串流並將資料傳送至本機儲存目的地。

中的基礎設施安全 AWS IoT Greengrass

作為受管服務，AWS IoT Greengrass 受到 AWS 全球網路安全的保護。如需 AWS 安全服務以及 如何 AWS 保護基礎設施的相關資訊，請參閱[AWS 雲端安全](#)。若要使用基礎設施安全的最佳實務來設計您的 AWS 環境，請參閱安全支柱 AWS Well-Architected Framework 中的[基礎設施保護](#)。

您可以使用 AWS 發佈的 API 呼叫，AWS IoT Greengrass 透過網路存取。使用者端必須支援下列專案：

- Transport Layer Security (TLS)。我們需要 TLS 1.2 並建議使用 TLS 1.3。
- 具備完美轉送私密(PFS)的密碼套件，例如 DHE (Ephemeral Diffie-Hellman)或 ECDHE (Elliptic Curve Ephemeral Diffie-Hellman)。現代系統(如 Java 7 和更新版本)大多會支援這些模式。

在 AWS IoT Greengrass 環境中，裝置會使用 X.509 憑證和密碼編譯金鑰來連接和驗證 AWS 雲端。如需詳細資訊，請參閱[the section called “裝置身分驗證和授權”](#)。

中的組態和漏洞分析 AWS IoT Greengrass

IoT 環境可能由大量裝置組成，各自具有多樣化的功能、長時間在線上，並且散佈在多個地理位置。這些特點使裝置設定複雜且極易出錯。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 裝置成為對駭客極具吸引力的目標，且難以持續保護裝置。

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，以確保裝置遵守安全最佳實務，並偵測裝置上的異常行為。這可讓您在裝置上強制執行一致的安全政策，並在裝置遭到入侵時快速回應。在與的連線中 AWS IoT Core，AWS IoT Greengrass 產生可與 AWS IoT Device Defender 功能搭配使用的[可預測用戶端 IDs](#)。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的 [AWS IoT Device Defender](#)。

在 AWS IoT Greengrass 環境中，您應該注意下列考量事項：

- 您須負責保護您的實體裝置、裝置上的檔案系統以及區域網路。
- AWS IoT Greengrass 不會強制執行使用者定義 Lambda 函數的網路隔離，無論它們是否在 [Greengrass 容器中](#)執行。因此，Lambda 函數可以與系統或網路外部執行的任何其他程序進行通訊。

如果您失去對 Greengrass 核心裝置的控制權，而且想要防止用戶端裝置將資料傳輸到核心，請執行下列動作：

1. 從 Greengrass 群組中移除 Greengrass 核心。
2. 輪換群組憑證授權機構憑證。在 AWS IoT 主控台中，您可以在群組的設定頁面上輪換 CA 憑證。在 AWS IoT Greengrass API 中，您可以使用 [CreateGroupCertificateAuthority](#) 動作。

如果核心裝置的硬碟容易遭竊，我們也建議您使用全磁碟加密。

AWS IoT Greengrass 和介面 VPC 端點 (AWS PrivateLink)

您可以建立介面 VPC 端點，在 VPC 與 AWS IoT Greengrass 控制平面之間建立私有連線。您可以使用此端點來管理 AWS IoT Greengrass 服務中的群組、Lambda 函數、部署和其他資源。介面端點採用 [AWS PrivateLink](#) 技術，可讓您在沒有網際網路閘道、NAT 裝置、VPN 連接或 AWS Direct Connect 連接的情況下私下存取 AWS IoT Greengrass APIs。VPC 中的執行個體不需要公有 IP 地址即可與 AWS IoT Greengrass APIs 通訊。VPC 與之間的流量 AWS IoT Greengrass 不會離開 Amazon 網路。

Note

目前，您無法將 Greengrass 核心裝置設定為在您的 VPC 中完全運作。

每個介面端點都是由您子網路中的一或多個[彈性網路介面](#)表示。

如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的[介面 VPC 端點 \(AWS PrivateLink\)](#)。

主題

- [AWS IoT Greengrass VPC 端點的考量事項](#)
- [建立 AWS IoT Greengrass 控制平面操作的介面 VPC 端點](#)
- [為建立 VPC 端點政策 AWS IoT Greengrass](#)

AWS IoT Greengrass VPC 端點的考量事項

設定介面 VPC 端點之前 AWS IoT Greengrass，請檢閱《Amazon VPC 使用者指南》中的[介面端點屬性和限制](#)。此外，請注意下列考量事項：

- AWS IoT Greengrass 支援從您的 VPC 呼叫其所有控制平面 API 動作。控制平面包含 [CreateDeployment](#) 和 [StartBulkDeployment](#) 等操作。控制平面不包含 [GetDeployment](#) 和 [Discover](#) 等操作，這是資料平面操作。
- 中國區域 AWS IoT Greengrass 目前不支援的 VPC AWS 端點。

建立 AWS IoT Greengrass 控制平面操作的介面 VPC 端點

您可以使用 Amazon VPC 主控台或 AWS Command Line Interface () 為 AWS IoT Greengrass 控制平面建立 VPC 端點 AWS CLI。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的 [建立介面端點](#)。

AWS IoT Greengrass 使用下列服務名稱建立的 VPC 端點：

- `com.amazonaws.region.greengrass`

如果您為端點啟用私有 DNS，則可以 AWS IoT Greengrass 使用區域的預設 DNS 名稱向提出 API 請求，例如 `greengrass.us-east-1.amazonaws.com`。預設為啟用私有 DNS。

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](#)。

為建立 VPC 端點政策 AWS IoT Greengrass

您可以將端點政策連接至 VPC 端點，以控制控制平面操作的 AWS IoT Greengrass 存取。此政策會指定下列資訊：

- 可執行動作的主體。
- 委託人可以執行的動作。
- 委託人可以對其執行動作的資源。

如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的 [使用 VPC 端點控制對服務的存取](#)。

Example 範例：AWS IoT Greengrass 動作的 VPC 端點政策

以下是端點政策的範例 AWS IoT Greengrass。連接到端點時，此政策會授予所有資源上所有委託人的所列 AWS IoT Greengrass 動作的存取權。

```
{
  "Statement": [
    {
```

```
    "Principal": "*",
    "Effect": "Allow",
    "Action": [
        "greengrass:CreateDeployment",
        "greengrass:StartBulkDeployment"
    ],
    "Resource": "*"
  }
]
```

的安全最佳實務 AWS IoT Greengrass

本主題包含 的安全最佳實務 AWS IoT Greengrass。

盡可能授予最低的許可

在 IAM 角色中使用最低權限集，以遵循最低權限原則。限制對 IAM 政策中的 Action 和 Resource 屬性使用 * 萬用字元。相反地，在可能的情況下，宣告一組有限的動作和資源。如需最低權限和其他原則最佳實務的詳細資訊，請參閱 [the section called “政策最佳實務”](#)。

最低權限最佳實務也適用於您連接到 Greengrass 核心和用戶端裝置的 AWS IoT 政策。

請勿在 Lambda 函數中硬式編碼登入資料

請勿在使用者定義的 Lambda 函數中硬式編碼登入資料。為了更妥善地保護您的登入資料：

- 若要與服務互動 AWS，請定義 [Greengrass 群組角色](#) 中特定動作和資源的許可。
- 使用 [本機秘密](#) 來儲存您的登入資料。或者，如果函數使用 AWS SDK，請使用預設登入資料提供者鏈結中的登入資料。

請勿記錄敏感資訊

您應該防止記錄登入資料和其他個人識別資訊 (PII)。即使存取核心裝置上的本機日誌需要根權限，而且存取 CloudWatch Logs 需要 IAM 許可，仍建議您實作下列保護措施。

- 請勿在 MQTT 主題路徑中使用敏感資訊。
- 請勿在 AWS IoT Core 登錄檔的裝置（物件）名稱、類型和屬性中使用敏感資訊。

- 請勿在使用者定義的 Lambda 函數中記錄敏感資訊。
- 請勿在 Greengrass 資源的名稱和 ID 中使用敏感資訊：
 - 連接器
 - 核心
 - 裝置
 - 函數
 - 群組
 - Loggers
 - 資源 (本機、機器學習和秘密)
 - 訂閱

建立目標訂閱

訂閱透過定義 服務、裝置和 Lambda 函數之間的訊息交換方式，來控制 Greengrass 群組中的資訊流程。若要確保應用程式只能執行其預期執行的動作，您的訂閱應允許發行者只將訊息傳送至特定主題，並限制訂閱者只接收來自其職能所需的主題的訊息。

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。裝置時鐘可能會隨著時間而偏移，或是電池可能會放電。

如需詳細資訊，請參閱《[開發人員指南](#)》中的[保持裝置的時鐘同步](#)最佳實務。AWS IoT Core

使用 Greengrass 核心管理裝置身分驗證

用戶端裝置可以執行 [FreeRTOS](#) 或使用 [AWS IoT 裝置 SDK](#) 或 [AWS IoT Greengrass 探索 API](#)，取得用來與相同 Greengrass 群組中核心進行連線和驗證的探索資訊。探索資訊包括：

- 與用戶端裝置位於相同 Greengrass 群組的 Greengrass 核心連線資訊。此資訊包括核心裝置每個端點的主機地址和連接埠號碼。
- 用來簽署本機 MQTT 伺服器憑證的群組憑證授權機構憑證。用戶端裝置使用群組 CA 憑證來驗證核心呈現的 MQTT 伺服器憑證。

以下是用戶端裝置使用 Greengrass 核心管理交互身分驗證的最佳實務。如果您的核心裝置遭到入侵，這些做法有助於降低風險。

驗證每個連線的本機 MQTT 伺服器憑證。

用戶端裝置每次與核心建立連線時，都應驗證核心提供的 MQTT 伺服器憑證。此驗證是核心裝置與用戶端裝置之間交互身分驗證的用戶端裝置端。用戶端裝置必須能夠偵測故障並終止連線。

請勿硬式編碼探索資訊。

用戶端裝置應該依賴探索操作來取得核心連線資訊和群組 CA 憑證，即使核心使用靜態 IP 地址也是如此。用戶端裝置不應硬式編碼此探索資訊。

定期更新探索資訊。

用戶端裝置應定期執行探索，以更新核心連線資訊和群組 CA 憑證。我們建議用戶端裝置在與核心建立連線之前更新此資訊。由於探索操作之間的持續時間較短，可將潛在的暴露時間降至最低，因此我們建議用戶端裝置定期中斷連線並重新連線，以觸發更新。

如果您失去對 Greengrass 核心裝置的控制權，並且想要防止用戶端裝置將資料傳輸到核心，請執行下列動作：

1. 從 Greengrass 群組中移除 Greengrass 核心。
2. 輪換群組憑證授權機構憑證。在 AWS IoT 主控台中，您可以在群組的設定頁面上輪換 CA 憑證。在 AWS IoT Greengrass API 中，您可以使用 [CreateGroupCertificateAuthority](#) 動作。

如果核心裝置的硬碟容易遭竊，我們也建議您使用全磁碟加密。

如需詳細資訊，請參閱[the section called “裝置身分驗證和授權”](#)。

另請參閱

- 《AWS IoT 開發人員指南》[中的 中的安全最佳實務 AWS IoT Core](#)
- AWS 官方部落格上[工業 IoT 解決方案的十項安全黃金規則](#)

在 中記錄和監控 AWS IoT Greengrass

監控是維護和 AWS 解決方案的可靠性、可用性 AWS IoT Greengrass 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。開始監控之前 AWS IoT Greengrass，您應該建立監控計畫，其中包含下列問題的答案：

- 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？

監控工具

AWS 提供可用來監控的工具 AWS IoT Greengrass。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。

您可以使用下列自動化監控工具來監控 AWS IoT Greengrass 和報告問題：

- Amazon CloudWatch Logs：監控、存放及存取來自 AWS CloudTrail 或其他來源的日誌檔案。如需詳細資訊，請參閱《Amazon CloudWatch 使用者指南》中的[監控日誌檔案](#)。
- 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 AWS CloudTrail 使用者指南中的[使用 CloudTrail 日誌檔案](#)。
- Amazon EventBridge – 使用 EventBridge 事件規則來取得有關使用 CloudTrail 記錄的 Greengrass 群組部署或 API 呼叫狀態變更的通知。如需詳細資訊，請參閱《Amazon EventBridge 使用者指南》中的[the section called “取得部署通知”](#)或什麼是 Amazon EventBridge？。 [EventBridge](#) EventBridge
- Greengrass 系統運作狀態遙測 – 訂閱以接收從 Greengrass 核心傳送的遙測資料。如需詳細資訊，請參閱[the section called “收集系統運作狀態遙測資料”](#)。
- 本機運作狀態檢查 – 使用運作狀態 APIs 取得核心裝置上的本機 AWS IoT Greengrass 程序狀態快照。如需詳細資訊，請參閱[the section called “呼叫本機運作狀態檢查 API”](#)。

另請參閱

- [the section called “使用 AWS IoT Greengrass 日誌進行監控”](#)
- [the section called “使用 記錄 AWS IoT Greengrass API 呼叫 AWS CloudTrail”](#)
- [the section called “取得部署通知”](#)

使用 AWS IoT Greengrass 日誌監控

AWS IoT Greengrass 包含 雲端服務和 AWS IoT Greengrass 核心軟體。AWS IoT Greengrass 核心軟體可以將日誌寫入 Amazon CloudWatch 和核心裝置的本機檔案系統。在核心上執行的 Lambda 函數和連接器也可以將日誌寫入 CloudWatch Logs 和本機檔案系統。您可以使用日誌來監控事件和排解疑難問題。所有 AWS IoT Greengrass 日誌項目都包含時間戳記、日誌層級和事件的相關資訊。對記錄設定所做的變更會在部署群組後生效。

於群組層級中設定日誌記錄。如需顯示如何設定 Greengrass 群組記錄日誌的步驟，請參閱[the section called “設定的記錄 AWS IoT Greengrass”](#)。

存取 CloudWatch Logs

如果您設定 CloudWatch 記錄，您可以在 Amazon CloudWatch 主控台的日誌頁面上檢視日誌。日誌的 AWS IoT Greengrass 日誌群組使用以下命名慣例：

```
/aws/greengrass/GreengrassSystem/greengrass-system-component-name  
/aws/greengrass/Lambda/aws-region/account-id/lambda-function-name
```

每個日誌群組都包含使用以下命名慣例的日誌串流：

```
date/account-id/greengrass-group-id/name-of-core-that-generated-log
```

當您使用 CloudWatch Logs 時，適用下列考量：

- 如果沒有網際網路連線，日誌會傳送至重試次數有限的 CloudWatch Logs。用完重試次數之後，會刪除事件。
- 交易、記憶體及其他限制應用。如需詳細資訊，請參閱[the section called “記錄限制”](#)。
- 您的 Greengrass 群組角色必須允許 AWS IoT Greengrass 寫入 CloudWatch Logs。若要授與許可，請以您的群組角色[內嵌以下內嵌政策](#)。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams"
      ],
      "Resource": [
        "arn:aws:logs:*:*:*"
      ]
    }
  ]
}
```

Note

您可以授與您的日誌資源的更精細存取權。如需詳細資訊，請參閱《Amazon [CloudWatch 使用者指南](#)》中的針對 [CloudWatch Logs 使用身分型政策 \(IAM 政策\)](#)。Amazon CloudWatch

群組角色是您建立並連接至 Greengrass 群組的 IAM 角色。您可以使用 主控台或 AWS IoT Greengrass API 來管理群組角色。

使用主控台

1. 在 AWS IoT 主控台導覽窗格的管理下，展開 Greengrass 裝置，然後選擇群組 (V1)。
2. 選擇目標群組。
3. 選擇檢視設定。在群組角色下，您可以檢視、關聯或取消群組角色的關聯。

如需說明如何連接群組角色的步驟，請參閱[群組角色](#)。

使用 CLI

- 若要尋找群組角色，請使用 [get-associated-role](#) 命令。
- 若要連接群組角色，請使用 [associate-role-to-group](#) 命令。
- 若要移除群組角色，請使用 [disassociate-role-from-group](#) 命令。

若要了解如何取得群組 ID 以與這些指令搭配使用，請參閱 [the section called “取得群組 ID”](#)。

存取檔案系統日誌

如果設定檔案系統記錄，則日誌檔會存放於核心裝置的 *greengrass-root*/ggc/var/log 之下。以下是高階目錄結構：

```
greengrass-root/ggc/var/log
- crash.log
- system
  - log files for each Greengrass system component
- user
  - region
    - account-id
      - log files generated by each user-defined Lambda function
  - aws
    - log files generated by each connector
```

Note

在預設情況下，*greengrass-root* 即為 /greengrass 目錄。如果已設定好 [寫入目錄](#)，那麼日誌就會位在該目錄中。

使用檔案系統日誌時有下列考量：

- 在檔案系統上讀取 AWS IoT Greengrass 日誌需要根許可。
- AWS IoT Greengrass 當日誌資料量接近設定的限制時，支援以大小為基礎的輪換和自動清除。
- crash.log 檔案僅於檔案系統日誌中提供。此日誌不會寫入 CloudWatch Logs。

- 磁碟使用量限制應用。如需詳細資訊，請參閱[the section called “記錄限制”](#)。

 Note

AWS IoT Greengrass Core 軟體 v1.0 的日誌存放在 *greengrass-root*/var/log 目錄下。

預設日誌記錄組態

如果未明確設定記錄設定，會在第一個群組部署後 AWS IoT Greengrass 使用以下預設記錄組態。

AWS IoT Greengrass 系統元件

- 類型 - FileSystem
- 元件 - GreengrassSystem
- 等級 - INFO
- 空格 - 128 KB

使用者定義的 Lambda 函數

- 類型 - FileSystem
- 元件 - Lambda
- 等級 - INFO
- 空格 - 128 KB

 Note

在第一次部署之前，只有系統元件會將日誌寫入檔案系統，因為不會部署使用者定義的 Lambda 函數。

設定 的記錄 AWS IoT Greengrass

您可以使用 AWS IoT 主控台或 [AWS IoT Greengrass APIs](#) 來設定 AWS IoT Greengrass 記錄。

Note

若要允許 AWS IoT Greengrass 將日誌寫入 CloudWatch Logs，您的群組角色必須允許[必要的 CloudWatch Logs 動作](#)。

設定日誌記錄 (主控台)

您可以在群組的設定頁面設定日誌記錄。

1. 在 AWS IoT 主控台導覽窗格的管理下，展開 Greengrass 裝置，然後選擇群組 (V1)。
2. 選擇您要設定日誌記錄的群組。
3. 在群組組態頁面上，選擇日誌索引標籤。
4. 選擇記錄的位置，如下所示：
 - 若要設定 CloudWatch 記錄，請在 CloudWatch 日誌組態中選擇編輯。
 - 若要設定本機日誌組態的檔案系統記錄，請選擇編輯。

您可以同時設定兩個記錄的位置或單選。

5. 在編輯日誌組態模式中，選取 Greengrass 系統日誌層級或 User Lambda 函數日誌層級。您可以同時選取兩個元件或單選。
6. 選擇您想要記錄事件日誌的最低等級。將會過濾低於此閾值的事件且不將其儲存。
7. 選擇儲存。變更會在部署群組之後生效。

設定日誌記錄 (API)

您可以使用 AWS IoT Greengrass 記錄器 APIs，以程式設計方式設定記錄。例如，使用 [CreateLoggerDefinition](#) 動作根據 [LoggerDefinitionVersion](#) 承載建立記錄器定義，其會使用以下語法：

```
{
  "Loggers": [
    {
      "Id": "string",
      "Type": "FileSystem|AWSCloudWatch",
      "Component": "GreengrassSystem|Lambda",
      "Level": "DEBUG|INFO|WARN|ERROR|FATAL",
    }
  ]
}
```

```
    "Space": "integer"
  },
  {
    "Id": "string",
    ...
  }
]
```

LoggerDefinitionVersion 是一個陣列，其中一個或多個 [Logger](#) 物件具有下列屬性：

Id

記錄器的識別符。

Type

日誌事件的儲存機制。使用 AWSCloudWatch 時，日誌事件會傳送到 CloudWatch Logs。使用 FileSystem 時，會儲存日誌事件於本機檔案系統中。

有效值：AWSCloudWatch、FileSystem

Component

日誌事件的來源。使用 GreengrassSystem 時，會記錄來自 Greengrass 系統元件的事件。使用 Lambda 時，會記錄來自使用者定義的 Lambda 函數之事件。

有效值：GreengrassSystem、Lambda

Level

日誌等級閾值。將會過濾低於此閾值的日誌事件，且不將其儲存。

有效值：DEBUG、INFO (建議)、WARN、ERROR、FATAL

Space

用來存放日誌的最大本機儲存容量 (KB)。僅當 Type 設為 FileSystem 時才會套用此範圍設定。

組態範例

以下 LoggerDefinitionVersion 範例指定記錄組態：

- 開啟 AWS IoT Greengrass 系統元件的檔案系統和ERROR以上記錄。

- 開啟使用者定義 Lambda 函數的檔案系統 INFO (及更高版本) 記錄。
- 開啟使用者定義 Lambda 函數的 CloudWatch INFO (及更高版本) 記錄。

```
{
  "Name": "LoggingExample",
  "InitialVersion": {
    "Loggers": [
      {
        "Id": "1",
        "Component": "GreengrassSystem",
        "Level": "ERROR",
        "Space": 10240,
        "Type": "FileSystem"
      },
      {
        "Id": "2",
        "Component": "Lambda",
        "Level": "INFO",
        "Space": 10240,
        "Type": "FileSystem"
      },
      {
        "Id": "3",
        "Component": "Lambda",
        "Level": "INFO",
        "Type": "AWSCloudWatch"
      }
    ]
  }
}
```

在您建立記錄器定義版本後，您可以使用其版本 ARN 在[部署群組](#)前建立群組。

記錄限制

AWS IoT Greengrass 有下列記錄限制。

每秒交易數

啟用記錄到 CloudWatch 時，記錄元件會在本機批次處理記錄事件，再將事件傳送到 CloudWatch，因此您可以以高於每個日誌串流每秒 5 個請求的速率進行記錄。

記憶體

如果 AWS IoT Greengrass 設定為將日誌傳送至 CloudWatch，且 Lambda 函數長時間記錄超過 5 MB/秒，則內部處理管道最終會填滿。理論上最壞的情況是每個 Lambda 函數 6 MB。

時鐘誤差

啟用記錄到 CloudWatch 時，記錄元件會使用一般 Signature 第 4 版簽署程序來簽署對 CloudWatch 的請求。如果 AWS IoT Greengrass 核心裝置上的系統時間不同步超過 [15 分鐘](#)，則會拒絕請求。

硬碟使用量

使用下列公式計算磁碟要記錄的最大使用總量。

```
greengrass-system-component-space * 8      // 7 if automatic IP detection is disabled
+ 128KB                                       // the internal log for the local logging
component
+ lambda-space * lambda-count           // different versions of a Lambda function are
treated as one
```

其中：

`greengrass-system-component-space`

AWS IoT Greengrass 系統元件日誌的本機儲存容量上限。

`lambda-space`

Lambda 函數日誌的本機儲存體數量上限。

`lambda-count`

部署的 Lambda 函數數量。

日誌受損

如果您的 AWS IoT Greengrass 核心裝置設定為僅記錄到 CloudWatch，而且沒有網際網路連線，則您無法擷取目前記憶體中的日誌。

當 Lambda 函數終止時（例如，在部署期間），不會將數秒的日誌寫入 CloudWatch。

CloudTrail 日誌

AWS IoT Greengrass 使用執行 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 Greengrass。如需詳細資訊，請參閱[the section called “使用記錄 AWS IoT Greengrass API 呼叫 AWS CloudTrail”](#)。

使用記錄 AWS IoT Greengrass API 呼叫 AWS CloudTrail

AWS IoT Greengrass 已與服務整合 AWS CloudTrail，此服務可提供使用者、角色或 AWS 服務在其中採取之動作的記錄 AWS IoT Greengrass。CloudTrail 會將 AWS IoT Greengrass 的所有 API 呼叫擷取為事件。擷取的呼叫包括來自 AWS IoT Greengrass 主控台的呼叫，以及對 AWS IoT Greengrass API 操作的程式碼呼叫。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 Greengrass。即使您未設定追蹤，依然可以透過 CloudTrail 主控台內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 Greengrass、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 Greengrass CloudTrail 中的資訊

當您建立帳戶 AWS 帳戶時，您的上會啟用 CloudTrail。當活動在發生時 AWS IoT Greengrass，該活動會與事件歷史記錄中的其他服務 AWS 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](#)。

若要求續記錄中的事件 AWS 帳戶，包括的事件 AWS IoT Greengrass，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 分割區中所有區域的事件，並將日誌檔案傳送到您指定的 Amazon S3 儲存貯體。此外，您可以設定其他 AWS 服務，以進一步分析和處理 CloudTrail 日誌中所收集的事件資料。如需詳細資訊，請參閱下列內容：

- [建立追蹤的概觀](#)
- [CloudTrail 支援的服務和整合](#)
- [設定 CloudTrail 的 Amazon SNS 通知](#)
- [接收多個區域的 CloudTrail 日誌檔案和接收多個帳戶的 CloudTrail 日誌檔案](#)

CloudTrail 會記錄所有 AWS IoT Greengrass 動作，並記錄在 [AWS IoT Greengrass API 參考](#)中。例如，對 AssociateServiceRoleToAccount、GetGroupVersion、GetConnectivityInfo 以及 CreateFunctionDefinition 動作發出的呼叫會在 CloudTrail 日誌檔案中產生項目。

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：

- 是否使用根或 AWS Identity and Access Management (IAM) 使用者登入資料提出請求。
- 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](#)。

了解 AWS IoT Greengrass 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 日誌檔並非依公有 API 呼叫的堆疊追蹤排序，因此不會以任何特定順序出現。

以下範例顯示的是展示 AssociateServiceRoleToAccount 動作的 CloudTrail 日誌項目。

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE",
    "arn": "arn:aws:iam::123456789012:user/Mary_Major",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "Mary_Major"
  },
  "eventTime": "2018-10-17T17:04:02Z",
  "eventSource": "greengrass.amazonaws.com",
  "eventName": "AssociateServiceRoleToAccount",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "203.0.113.12",
  "userAgent": "apimanager.amazonaws.com",
  "errorCode": "BadRequestException",
  "requestParameters": null,
  "responseElements": {
    "Message": "That role ARN is invalid."
  },
  "requestID": "a5990ec6-d22e-11e8-8ae5-c7d2eEXAMPLE",
  "eventID": "b9070ce2-0238-451a-a9db-2dbf1EXAMPLE",
  "readOnly": false,
```



```
"eventType": "AwsApiCall",
"recipientAccountId": "123456789012"
}
```

以下範例顯示的是展示 GetGroupVersion 動作的 CloudTrail 日誌項目。

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE",
    "arn": "arn:aws:iam::123456789012:user/Mary_Major",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "Mary_Major",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2018-10-17T18:14:57Z"
      }
    }
  },
  "invokedBy": "apimanager.amazonaws.com",
},
"eventTime": "2018-10-17T18:15:11Z",
"eventSource": "greengrass.amazonaws.com",
"eventName": "GetGroupVersion",
"awsRegion": "us-east-1",
"sourceIPAddress": "203.0.113.12",
"userAgent": "apimanager.amazonaws.com",
"requestParameters": {
  "GroupVersionId": "6c477753-dbf2-4cb8-acc3-5ba4eEXAMPLE",
  "GroupId": "90fcf6df-413c-4515-93a8-00056EXAMPLE"
},
"responseElements": null,
"requestID": "95dcffce-d238-11e8-9240-a3993EXAMPLE",
"eventID": "8a608034-82ed-431b-b5e0-87fbdEXAMPLE",
"readOnly": true,
"eventType": "AwsApiCall",
"recipientAccountId": "123456789012"
}
```

以下範例顯示的是展示 GetConnectivityInfo 動作的 CloudTrail 日誌項目。

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE",
    "arn": "arn:aws:iam::123456789012:user/Mary_Major",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "Mary_Major"
  },
  "eventTime": "2018-10-17T17:02:12Z",
  "eventSource": "greengrass.amazonaws.com",
  "eventName": "GetConnectivityInfo",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "203.0.113.12",
  "userAgent": "apimanager.amazonaws.com",
  "requestParameters": {
    "ThingName": "us-east-1_CIS_1539795000000_"
  },
  "responseElements": null,
  "requestID": "63e3ebe3-d22e-11e8-9ddd-5baf3EXAMPLE",
  "eventID": "db2260d1-a8cc-4a65-b92a-13f65EXAMPLE",
  "readOnly": true,
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}
```

以下範例顯示的是展示 CreateFunctionDefinition 動作的 CloudTrail 日誌項目。

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE",
    "arn": "arn:aws:iam::123456789012:user/Mary_Major",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "Mary_Major"
  },
  "eventTime": "2018-10-17T18:01:11Z",
  "eventSource": "greengrass.amazonaws.com",
  "eventName": "CreateFunctionDefinition",
  "awsRegion": "us-east-1",
```

```
{
  "sourceIPAddress": "203.0.113.12",
  "userAgent": "apimanager.amazonaws.com",
  "requestParameters": {
    "InitialVersion": "****"
  },
  "responseElements": {
    "CreationTimestamp": "2018-10-17T18:01:11.449Z",
    "LatestVersion": "dae06a61-c32c-41e9-b983-ee5cfEXAMPLE",
    "LatestVersionArn": "arn:aws:greengrass:us-east-1:123456789012:/greengrass/definition/functions/7a94847d-d4d2-406c-9796-a3529EXAMPLE/versions/dae06a61-c32c-41e9-b983-ee5cfEXAMPLE",
    "LastUpdatedTimestamp": "2018-10-17T18:01:11.449Z",
    "Id": "7a94847d-d4d2-406c-9796-a3529EXAMPLE",
    "Arn": "arn:aws:greengrass:us-east-1:123456789012:/greengrass/definition/functions/7a94847d-d4d2-406c-9796-a3529EXAMPLE"
  },
  "requestID": "a17d4b96-d236-11e8-a74e-3db27EXAMPLE",
  "eventID": "bdbf6677-a47a-4c78-b227-c5f64EXAMPLE",
  "readOnly": false,
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}
```

另請參閱

- AWS CloudTrail 《使用者指南》中的[什麼是 AWS CloudTrail？](#)
- 《Amazon [EventBridge 使用者指南](#)》中的[使用 CloudTrail 在 AWS API 呼叫上建立觸發的 EventBridge 規則](#) EventBridge
- [AWS IoT Greengrass API 參考](#)

從 AWS IoT Greengrass 核心裝置收集系統運作狀態遙測資料

系統運作狀態遙測資料是診斷資料，可協助您監控 Greengrass 核心裝置上關鍵操作的效能。Greengrass 核心上的遙測代理程式會收集本機遙測資料，並將其發佈至 Amazon EventBridge，而不需要任何客戶互動。核心裝置會盡最大努力將遙測資料發佈至 EventBridge。例如，核心裝置可能無法在離線時交付遙測資料。

Note

Amazon EventBridge 是一種事件匯流排服務，可用來將應用程式與各種來源的資料連線，例如 Greengrass 核心裝置和[部署通知](#)。如需詳細資訊，請參閱《Amazon EventBridge 使用者指南》中的[什麼是 Amazon EventBridge？](#)

您可以建立專案和應用程式，從邊緣裝置擷取、分析、轉換和報告遙測資料。程序工程師等網域專家可以使用這些應用程式來深入了解機群運作狀態。

為了確保 Greengrass 邊緣元件正常運作，會將資料 AWS IoT Greengrass 用於開發和品質改善目的。此功能也有助於通知新的和增強的邊緣功能。最多 AWS IoT Greengrass 只會保留七天的遙測資料。

此功能可在 AWS IoT Greengrass Core 軟體 1.11.0 版中使用，並預設為所有 Greengrass 核心啟用，包括現有的核心。只要升級至 AWS IoT Greengrass Core 軟體 v1.11.0 或更新版本，就會自動開始接收資料。

如需如何存取或管理已發佈遙測資料的資訊，請參閱 [the section called “訂閱以接收遙測資料”](#)。

遙測代理程式會收集並發佈下列系統指標。

遙測指標

名稱	描述	來源
SystemMemUsage	Greengrass 核心裝置上所有應用程式目前使用的記憶體數量，包括作業系統。	系統
CpuUsage	Greengrass 核心裝置上所有應用程式目前使用的 CPU 數量，包括作業系統。	系統
TotalNumberOfFDs	Greengrass 核心裝置的作業系統存放的檔案描述項數量。一個檔案描述項可唯一識別一個開啟的檔案。	系統

名稱	描述	來源
LambdaOutOfMemory	導致 Lambda 函數記憶體不足的執行次數。	系統
DroppedMessageCount	目的地為的捨棄訊息數量 AWS IoT Core。	GGCloudSpooler 系統元件
LambdaTimeout	執行使用者定義 Lambda 函數的逾時次數。	使用者定義的 Lambda 函數 AWS 雲端和系統
LambdaUngracefully Killed	使用者定義的 Lambda 函數無法完成的執行次數。	使用者定義的 Lambda 函數 AWS 雲端和系統
LambdaError	導致使用者定義 Lambda 函數寫入錯誤日誌的執行次數。	使用者定義的 Lambda 函數 AWS 雲端和系統
BytesAppended	附加至串流管理員的資料位元組數。	GGStreamManager 系統元件
BytesUploadedToIoT Analytics	串流管理員匯出至頻道的資料位元組數 AWS IoT Analytics。	GGStreamManager 系統元件
BytesUploadedToKinesis	串流管理員匯出至 Amazon Kinesis Data Streams 中串流的資料位元組數。	GGStreamManager 系統元件
BytesUploadedToIoT SiteWise	串流管理員匯出至資產屬性的資料位元組數 AWS IoT SiteWise。	GGStreamManager 系統元件
BytesUploadedToS3ExportTaskExecutor	串流管理員匯出至 Amazon S3 物件的資料位元組數。	GGStreamManager 系統元件
BytesUploadedToHTTP	串流管理員匯出至 HTTP 的資料位元組數。	GGStreamManager 系統元件

設定遙測設定

Greengrass 遙測使用下列設定：

- 遙測代理程式每小時彙總一次遙測資料。
- 遙測代理程式每 24 小時發佈一次遙測訊息。

Note

這些設定無法變更。

您可以啟用或停用 Greengrass 核心裝置的遙測功能。AWS IoT Greengrass 會使用陰影來管理遙測組態。<https://docs.aws.amazon.com/iot/latest/developerguide/iot-device-shadows.html>當核心有連線時，您的變更會立即生效 AWS IoT Core。

遙測代理程式會使用服務品質 (QoS) 層級為 0 的 MQTT 通訊協定來發佈資料。這表示它不會確認交付或重試發佈嘗試。遙測訊息與目標訂閱的其他訊息共用 MQTT 連線 AWS IoT Core。

除了您的資料連結成本之外，從核心到 的資料傳輸 AWS IoT Core 是免費的。這是因為代理程式發佈到 AWS 預留主題。不過，視您的使用案例而定，您可能會在接收或處理資料時產生成本。

要求

當您設定遙測設定時，適用下列要求：

- 您必須使用 AWS IoT Greengrass Core 軟體 v1.11.0 或更新版本。

Note

如果您執行的是舊版，而且不想使用遙測，則不需要執行任何動作。

- 您必須提供 IAM 許可來更新核心（物件）影子，並在更新遙測設定之前呼叫組態 APIs。

下列範例 IAM 政策可讓您管理特定核心的影子和執行時間組態：

JSON

```
{
```

```

"Version": "2012-10-17",
"Statement": [
  {
    "Sid": "AllowManageShadow",
    "Effect": "Allow",
    "Action": [
      "iot:GetThingShadow",
      "iot:UpdateThingShadow",
      "iot:DeleteThingShadow",
      "iot:DescribeThing"
    ],
    "Resource": [
      "arn:aws:iot:us-east-1:123456789012:thing/core-name-*"
    ]
  },
  {
    "Sid": "AllowManageRuntimeConfig",
    "Effect": "Allow",
    "Action": [
      "greengrass:GetCoreRuntimeConfiguration",
      "greengrass:UpdateCoreRuntimeConfiguration"
    ],
    "Resource": [
      "arn:aws:iot:us-east-1:123456789012:thing/core-name"
    ]
  }
]
}

```

您可以使用萬用字元*命名機制，授予對資源的精細或條件式存取。如需詳細資訊，請參閱 [《IAM 使用者指南》](#) 中的 [新增和移除 IAM 政策](#)。

設定遙測設定（主控台）

以下說明如何更新 AWS IoT Greengrass 主控台中 Greengrass 核心的遙測設定。

1. 在 AWS IoT 主控台導覽窗格的管理下，展開 Greengrass 裝置，然後選擇群組 (V1)。
2. 在 Greengrass 群組下，選擇您的目標群組。
3. 在群組組態頁面的概觀區段中，選擇您的 Greengrass 核心。
4. 在核心的組態頁面上，選擇遙測索引標籤。
5. 在系統運作狀態遙測區段中，選擇設定。

6. 在設定遙測中，選取遙測以啟用或停用遙測狀態。

Important

預設會針對 AWS IoT Greengrass Core 軟體 v1.11.0 或更新版本啟用遙測功能。

變更會在執行時間生效。您不需要部署 群組。

設定遙測設定 (CLI)

在 AWS IoT Greengrass API 中，TelemetryConfiguration 物件代表 Greengrass 核心的遙測設定。此物件是與核心相關聯之RuntimeConfiguration物件的一部分。您可以使用 AWS IoT Greengrass API AWS CLI或 AWS SDK 來管理 Greengrass 遙測。本節中的範例使用 AWS CLI。

檢查遙測設定

下列命令會取得 Greengrass 核心的遙測設定。

- 將 *core-thing-name* 取代為目標核心的名稱。

若要取得物件名稱，請使用 [get-core-definition-version](#) 命令。命令會傳回包含物件名稱之物件的 ARN。

```
aws greengrass get-thing-runtime-configuration --thing-name core-thing-name
```

命令會在 JSON 回應中傳回GetCoreRuntimeConfigurationResponse物件。例如：

```
{
  "RuntimeConfiguration": {
    "TelemetryConfiguration": {
      "ConfigurationSyncStatus": "OutOfSync",
      "Telemetry": "On"
    }
  }
}
```

設定遙測設定

下列命令會更新 Greengrass 核心的遙測設定。

- 將 *core-thing-name* 取代為目標核心的名稱。

若要取得物件名稱，請使用 [get-core-definition-version](#) 命令。命令會傳回包含物件名稱之物件的 ARN。

JSON expanded

```
aws greengrass update-thing-runtime-configuration --thing-name core-thing-name --
telemetry-configuration '{
  "RuntimeConfiguration": {
    "TelemetryConfiguration": {
      "ConfigurationSyncStatus": "InSync",
      "Telemetry": "Off"
    }
  }
}
```

JSON single-line

```
aws greengrass update-thing-runtime-configuration --thing-name core-thing-name --
telemetry-configuration "{\"TelemetryConfiguration\":{\"ConfigurationSyncStatus\":"
\": \"InSync\", \"Telemetry\": \"Off\"}}"
```

如果 ConfigurationSyncStatus 是 `InSync`，則已套用對遙測設定的變更 InSync。變更會在執行時間生效。您不需要部署 群組。

TelemetryConfiguration 物件

TelemetryConfiguration 物件具有下列屬性：

ConfigurationSyncStatus

檢查遙測設定是否同步。您可能不會變更此屬性。

類型：字串

有效值：InSync 或 OutOfSync

Telemetry

開啟或關閉遙測。預設值為 On。

類型：字串

有效值：On 或 Off

訂閱以接收遙測資料

您可以在 Amazon EventBridge 中建立規則，定義如何處理從 Greengrass 核心裝置發佈的遙測資料。EventBridge 收到資料時，會叫用規則中定義的目標動作。例如，您可以建立事件規則來傳送通知、存放事件資訊、採取修正動作或叫用其他事件。

遙測事件

部署狀態變更的事件，包括遙測資料使用下列格式：

```
{
  "version": "0",
  "id": "f70f943b-9ae2-e7a5-fec4-4c22178a3e6a",
  "detail-type": "Greengrass Telemetry Data",
  "source": "aws.greengrass",
  "account": "123456789012",
  "time": "2020-07-28T20:45:53Z",
  "region": "us-west-1",
  "resources": [],
  "detail": {
    "ThingName": "CoolThing",
    "Schema": "2020-06-30",
    "ADP": [
      {
        "TS": 123231546,
        "NS": "StreamManager",
        "M": [
          {
            "N": "BytesAppended|BytesUploadedToKinesis",
            "Sum": 11,
            "U": "Bytes"
          }
        ]
      },
      {
        "TS": 123231546,
        "NS": "StreamManager",
        "M": [
          {
            "N": "BytesAppended|BytesUploadedToS3ExportTaskExecutor",
```

```

        "Sum": 11,
        "U": "Bytes"
    }
]
},
{
    "TS": 123231546,
    "NS": "StreamManager",
    "M": [
        {
            "N": "BytesAppended|BytesUploadedToHTTP",
            "Sum": 11,
            "U": "Bytes"
        }
    ]
},
{
    "TS": 123231546,
    "NS": "StreamManager",
    "M": [
        {
            "N": "BytesAppended|BytesUploadedToIoTAnalytics",
            "Sum": 11,
            "U": "Bytes"
        }
    ]
},
{
    "TS": 123231546,
    "NS": "StreamManager",
    "M": [
        {
            "N": "BytesAppended|BytesUploadedToIoTSiteWise",
            "Sum": 11,
            "U": "Bytes"
        }
    ]
},
{
    "TS": 123231546,
    "NS": "arn:aws:lambda:us-west-1:123456789012:function:my-function",
    "M": [
        {
            "N": "LambdaTimeout",

```

```

        "Sum": 15,
        "U": "Count"
    }
]
},
{
    "TS": 123231546,
    "NS": "CloudSpooler",
    "M": [
        {
            "N": "DroppedMessageCount",
            "Sum": 15,
            "U": "Count"
        }
    ]
},
{
    "TS": 1593727692,
    "NS": "SystemMetrics",
    "M": [
        {
            "N": "SystemMemUsage",
            "Sum": 11.23,
            "U": "Megabytes"
        },
        {
            "N": "CpuUsage",
            "Sum": 35.63,
            "U": "Percent"
        },
        {
            "N": "TotalNumberOfFDs",
            "Sum": 416,
            "U": "Count"
        }
    ]
},
{
    "TS": 1593727692,
    "NS": "arn:aws:lambda:us-west-1:123456789012:function:my-function",
    "M": [
        {
            "N": "LambdaOutOfMemory",
            "Sum": 12,

```

```

        "U": "Count"
      },
      {
        "N": "LambdaUngracefullyKilled",
        "Sum": 100,
        "U": "Count"
      },
      {
        "N": "LambdaError",
        "Sum": 7,
        "U": "Count"
      }
    ]
  }
}

```

ADP 陣列包含具有下列屬性的彙總資料點清單：

TS

必要. 彙總資料時的時間戳記。

NS

必要. 系統的命名空間。

M

必要. 指標清單。指標包含下列屬性：

N

[指標](#)的名稱。

Sum

彙總的指標值。遙測代理程式會將新值新增至先前的總計，因此總和為不斷增加的值。您可以使用時間戳記來尋找特定彙總的值。例如，若要尋找最新的彙總值，請從最新的時間戳記值中減去先前的時間戳記值。

U

指標值的單位。

ThingName

必要。您目標的物件裝置名稱。

建立 EventBridge 規則的先決條件

為建立 EventBridge 規則之前 AWS IoT Greengrass，您應該執行下列動作：

- 熟悉 Eventbridge 中的事件、規則和目標。
- 建立和設定 EventBridge 規則調用的[目標](#)。規則可以調用許多類型的目標，例如 Amazon Kinesis 串流、AWS Lambda 函數、Amazon SNS 主題和 Amazon SQS 佇列。

您的 EventBridge 規則和相關聯的目標必須位於您建立 Greengrass 資源的 AWS 區域中。如需詳細資訊，請參閱《AWS 一般參考》中的[服務端點和配額](#)。

如需詳細資訊，請參閱《Amazon EventBridge 使用者指南》中的[什麼是 Amazon EventBridge？](#)和[Amazon EventBridge 入門](#)。

建立事件規則以取得遙測資料（主控台）

使用下列步驟來使用 AWS 管理主控台 建立 EventBridge 規則，以接收由 Greengrass 核心發佈的遙測資料。藉由這個規則，Web 伺服器、電子郵件地址和其他主題訂閱者將能回應事件。如需詳細資訊，請參閱《Amazon [EventBridge 使用者指南](#)》中的[從 AWS 資源建立事件觸發的 EventBridge 規則](#)。

1. 開啟 [Amazon EventBridge 主控台](#)，然後選擇建立規則。
2. 在 Name and description (名稱和描述)，輸入規則的名稱和描述。
3. 選擇事件匯流排-，並在選取的事件匯流排上啟用規則。
4. 選取規則類型，然後選擇具有事件模式的規則。
5. 選擇下一步。
6. 在事件來源欄位中，選擇 AWS 事件或 EventBridge 合作夥伴事件。
7. 針對範例事件，選擇AWS 事件，然後選取 Greengrass 遙測資料。
8. 在事件模式中，進行下列選擇：
 - a. 在事件來源欄位中，選擇 AWS 服務。
 - b. 針對AWS 服務，選擇 Greengrass。
 - c. 針對事件類型，選擇 Greengrass 遙測資料。

9. 選擇下一步。
10. 針對目標 1，選擇 AWS 服務。
11. 針對選取目標，選擇 SQS 佇列。
12. 針對佇列，選擇您的 函數。

建立事件規則以取得遙測資料 (CLI)

使用下列步驟來使用 AWS CLI 建立 EventBridge 規則，以接收由 Greengrass 核心發佈的遙測資料。藉由這個規則，Web 伺服器、電子郵件地址和其他主題訂閱者將能回應事件。

1. 建立 規則。
 - 以核心的物件名稱取代 *thing-name*。

若要取得物件名稱，請使用 [get-core-definition-version](#) 命令。命令會傳回包含物件名稱之物件的 ARN。

```
aws events put-rule \  
  --name TestRule \  
  --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail\": {\"ThingName\":  
    [\"thing-name\"]}}\"
```

模式省略的屬性會遭到忽略。

2. 新增主題作為規則目標。下列範例使用 Amazon SQS，但您可以設定其他目標類型。
 - 將 *queue-arn* 取代為 Amazon SQS 佇列的 ARN。

```
aws events put-targets \  
  --rule TestRule \  
  --targets "Id"="1", "Arn"="queue-arn"
```

Note

若要允許 Amazon EventBridge 叫用您的目標佇列，您必須將資源型政策新增至主題。如需詳細資訊，請參閱 [《Amazon EventBridge 使用者指南》中的 Amazon SQS 許可](#)。EventBridge

如需詳細資訊，請參閱《Amazon EventBridge 使用者指南》中的 [EventBridge 中的事件和事件模式](#)。

對 AWS IoT Greengrass 遙測進行故障診斷

使用以下資訊來協助疑難排解設定 AWS IoT Greengrass 遙測的問題。

錯誤：回應包含執行 get-thing-runtime-configuration 命令後的 "ConfigurationStatus" : "OutOfSync"

解決方案：

- AWS IoT Device Shadow 服務需要一些時間來處理執行期組態更新，並將更新交付至 Greengrass 核心裝置。您可以稍後等待並檢查遙測設定是否同步。
- 請確定您的核心裝置在線上。
- 在 [中啟用 Amazon CloudWatch Logs AWS IoT Core](#) 以監控影子。
- 使用 [AWS IoT 指標](#) 來監控您的物件。

呼叫本機運作狀態檢查 API

AWS IoT Greengrass 包含本機 HTTP API，可提供由 啟動之本機工作者程序目前狀態的快照 AWS IoT Greengrass。此快照包含使用者定義的 Lambda 函數和系統 Lambda 函數。System Lambda 函數是 AWS IoT Greengrass Core 軟體的一部分。它們在核心裝置上以本機工作者程序執行，並管理訊息路由、本機陰影同步和自動 IP 地址偵測等操作。

運作狀態檢查 API 支援下列請求：

- 傳送 GET 請求 [以取得所有工作者的健康資訊](#)。
- 傳送 POST 請求以取得 [指定工作者的健康資訊](#)。

請求會在裝置本機傳送，不需要網際網路連線。

取得所有工作者的健康資訊

傳送 GET 請求以取得所有執行中工作者的健康資訊。

- 將 ### 取代為 IPC 的連接埠號碼。


```
GET http://localhost:port/2016-11-01/health/workers
```

port

IPC 的连接埠號碼。

值可能介於 1024 和 65535 之間。預設值為 8000。

若要變更此連接埠號碼，您可以更新 config.json 檔案中的 ggDaemonPort 屬性。如需詳細資訊，請參閱[AWS IoT Greengrass 核心組態檔案](#)。

範例請求

下列範例curl請求會取得所有工作者的運作狀態資訊。

```
curl http://localhost:8000/2016-11-01/health/workers
```

JSON 回應

此請求會傳回一系列[工作者運作狀態資訊](#)物件。

回應範例

下列範例回應會列出 啟動之所有工作者程序的運作狀態資訊物件 AWS IoT Greengrass。

```
[
  {
    "FuncArn": "arn:aws:lambda::function:GGShadowService:1",
    "WorkerId" : "65515053-2f70-43dc-7cc0-1712bEXAMPLE",
    "ProcessId": "1234",
    "WorkerState": "Waiting"
  },
  {
    "FuncArn": "arn:aws:lambda::function:GGSecretManager:1",
    "WorkerId": "a9916cc2-1b4d-4f0e-4b12-b1872EXAMPLE",
    "ProcessId": "9798",
    "WorkerState": "Waiting"
  },
  {
    "FuncArn": "arn:aws:lambda:us-west-2:123456789012:function:my-lambda-function:3",
    "WorkerId": "2e6f785e-66a5-42c9-67df-42073EXAMPLE",
```

```
    "ProcessId": "11837",
    "WorkerState": "Waiting"
  },
  ...
]
```

取得指定工作者的運作狀態資訊

傳送POST請求以取得指定工作者的運作狀態資訊。將###取代為 IPC 的連接埠號碼。預設值為 8000。

```
POST http://localhost:port/2016-11-01/health/workers
```

範例請求

下列範例curl請求會取得指定工作者的運作狀態資訊。

```
curl --data "@body.json" http://localhost:8000/2016-11-01/health/workers
```

以下是body.json請求內文的範例：

```
{
  "FuncArns": [
    "arn:aws:lambda::function:GGShadowService:1",
    "arn:aws:lambda:us-west-2:123456789012:function:my-lambda-function:3"
  ]
}
```

請求內文包含FuncArns陣列。

FuncArns

代表目標工作者之 Lambda 函數的 Amazon Resource Name ARNs) 清單。

- 對於使用者定義的 Lambda 函數，指定目前部署版本的 ARN。如果您使用別名 ARN 將 Lambda 函數新增至群組，您可以使用 GET 請求來取得所有工作者，然後選擇您要查詢ARNs。
- 針對系統 Lambda 函數，指定對應 Lambda 函數的 ARN。如需詳細資訊，請參閱[the section called “System Lambda 函數”](#)。

類型：字串的陣列

長度下限：1

長度上限：核心裝置上由 啟動 AWS IoT Greengrass 的工作者總數。

JSON 回應

此請求會傳回Workers陣列和InvalidArns陣列。

Workers

指定工作者的運作狀態資訊物件清單。

類型：[運作狀態資訊物件](#)陣列

InvalidArns

無效的函數 ARNs 清單，包括沒有相關聯工作者ARNs。

類型：字串的陣列

回應範例

下列範例回應會列出指定工作者[的運作狀態資訊物件](#)。

```
{
  "Workers": [
    {
      "FuncArn": "arn:aws:lambda::function:GGShadowService:1",
      "WorkerId" : "65515053-2f70-43dc-7cc0-1712bEXAMPLE",
      "ProcessId": "1234",
      "WorkerState": "Waiting"
    },
    {
      "FuncArn": "arn:aws:lambda:us-west-2:123456789012:function:my-lambda-function:3",
      "WorkerId": "2e6f785e-66a5-42c9-67df-42073ESAMPLE",
      "ProcessId": "11837",
      "WorkerState": "Waiting"
    }
  ],
  "InvalidArns" : [
    "some-malformed-arn",
    "arn:aws:lambda:us-west-2:123456789012:function:some-unknown-function:1"
  ]
}
```

```
}
```

此請求會傳回下列錯誤：

400 無效的請求

請求內文格式錯誤。若要解決此問題，請使用下列格式並重新傳送請求：

```
{"FuncArns":["function-1-arn","function-2-arn"]}
```

400 請求超過工作者數量上限

FuncArns 陣列中指定的 ARNs 數目超過工作者數目。

工作者運作狀態資訊

運作狀態資訊物件包含下列屬性：

FuncArn

代表工作者之系統 Lambda 函數的 ARN。

類型：string

WorkerId

工作者的 ID。此屬性對於偵錯非常有用。runtime.log 檔案和 Lambda 函數日誌包含工作者 ID，因此此屬性對於偵錯啟動多個執行個體的隨需 Lambda 函數特別有用。

類型：string

ProcessId

工作者程序的程序 ID (PID)。

類型：int

WorkerState

工作者的狀態。

類型：string

以下是可能的工作者狀態：

Working

處理訊息。

Waiting

等待訊息。適用於做為協助程式或獨立程序執行的長期 Lambda 函數。

Starting

轉身，入門。

FailedInitialization

無法初始化。

Terminated

停止 Greengrass 協助程式

NotStarted

無法啟動，再次嘗試啟動。

Initialized

初始化成功。

System Lambda 函數

您可以請求下列系統 Lambda 函數的運作狀態資訊：

GGCloudSpooler

管理以 AWS IoT Core 做為來源或目標的 MQTT 訊息佇列。

ARN : `arn:aws:lambda::function:GGCloudSpooler:1`

GGConnManager

在 Greengrass 核心和用戶端裝置之間路由 MQTT 訊息。

ARN : `arn:aws:lambda::function:GGConnManager`

GGDeviceCertificateManager

接聽 AWS IoT 影子中核心 IP 端點的變更，並產生 GGConnManager 用於交互身分驗證的伺服器端憑證。

ARN : `arn:aws:lambda:::function:GGDeviceCertificateManager`

GGIPDetector

管理自動 IP 地址偵測，讓 Greengrass 群組中的裝置探索 Greengrass 核心裝置。當您手動提供 IP 地址時，不適用此服務。

ARN : `arn:aws:lambda:::function:GGIPDetector:1`

GGSecretManager

管理本機秘密的安全儲存，並依使用者定義的 Lambda 和連接器存取。

ARN : `arn:aws:lambda:::function:GGSecretManager:1`

GGShadowService

管理用戶端裝置的本機陰影。

ARN : `arn:aws:lambda:::function:GGShadowService`

GGShadowSyncManager

如果裝置的 `syncShadow` 屬性設定為 `true`，AWS 雲端 則同步本機影子與核心裝置和用戶端裝置的 `true`。

ARN : `arn:aws:lambda:::function:GGShadowSyncManager`

GGStreamManager

在本機處理資料串流，並執行自動匯出至 AWS 雲端。

ARN : `arn:aws:lambda:::function:GGStreamManager:1`

GGTES

擷取在本機程式碼用來存取服務之 Greengrass 群組角色中定義的 IAM 憑證的本機字符交換 AWS 服務。

ARN : `arn:aws:lambda:::function:GGTES`

標記您的 AWS IoT Greengrass 資源

標籤可協助您組織和管理 AWS IoT Greengrass 群組。您可以使用標籤來指派中繼資料至群組、大量部署和核心、裝置，以及加入至群組的其他資源。標籤也可用於 IAM 政策，以定義對 Greengrass 資源的條件式存取。

Note

目前，AWS IoT 帳單群組或成本分配報告不支援 Greengrass 資源標籤。

標籤基本概念

標籤可讓您將 AWS IoT Greengrass 資源分類，例如依用途、擁有者和環境。當您有許多相同類型的資源時，您可以依據先前附加的標籤，快速識別資源。每個標籤皆包含由您定義的一個標籤鍵和選用值。我們建議您為每種資源類型設計一組標籤金鑰。使用一致的標籤金鑰組可讓您更輕鬆管理您的資源。例如，您可以為您的群組定義一組標籤，幫助您追蹤核心裝置的工廠位置。如需詳細資訊，請參閱 [AWS 標記策略](#)。

AWS IoT 主控台支援的標記支援

您可以在 AWS IoT 主控台中建立、檢視和管理 Greengrass Group 資源的標籤。在建立標籤之前，請注意下列標籤限制。如需詳細資訊，請參閱《》中的 [標籤命名和使用慣例](#) Amazon Web Services 一般參考。

在建立群組時指派標籤

您可以在建立群組時將標籤指派給群組。選擇標籤區段下的新增標籤，以顯示標記輸入欄位。

從群組組態頁面檢視和管理標籤

您可以選擇檢視設定，從群組組態頁面檢視和管理標籤。在群組的標籤區段中，選擇管理標籤以新增、編輯或移除群組標籤。

AWS IoT Greengrass API 中的標記支援

您可以使用 AWS IoT Greengrass API 來建立、列出和管理支援標記之 AWS IoT Greengrass 資源的標籤。在建立標籤之前，請注意下列標籤限制。如需詳細資訊，請參閱《》中的 [標籤命名和使用慣例](#) Amazon Web Services 一般參考。

- 若要在資源建立時新增標籤，請在資源的 tags 屬性定義這些標籤。
- 若要在建立資源後新增標籤，或更新標籤值，請使用 TagResource 動作。
- 若要從資源移除標籤，請使用 UntagResource 動作。
- 若要擷取與一項資源相關聯的標籤，請使用 ListTagsForResource 動作，或取得資源並檢查其 tags 屬性。

下表列出您可以在 AWS IoT Greengrass API 中標記的資源及其對應的 Create 和 Get 動作。

資源	建立	取得
Group	CreateGroup	GetGroup
ConnectorDefinition	CreateConnectorDefinition	GetConnectorDefinition
CoreDefinition	CreateCoreDefinition	GetCoreDefinition
DeviceDefinition	CreateDeviceDefinition	GetDeviceDefinition
FunctionDefinition	CreateFunctionDefinition	GetFunctionDefinition
LoggerDefinition	CreateLoggerDefinition	GetLoggerDefinition
ResourceDefinition	CreateResourceDefinition	GetResourceDefinition
SubscriptionDefinition	CreateSubscriptionDefinition	GetSubscriptionDefinition
BulkDeployment	StartBulkDeployment	GetBulkDeploymentsStatus

使用下列動作列出並管理支援標記之資源的標籤：

- [TagResource](#)。將標籤新增至資源。也用來改變標籤鍵值對的數值。

- [ListTagsForResource](#)。列出資源的標籤。
- [UntagResource](#)。從資源移除標籤。

您可以隨時新增或移除資源的標籤。若要變更標籤金鑰的值，請在資源加入定義相同金鑰和新數值的標籤。新的數值會覆寫舊的數值。您可以將數值設為空白字串，但您無法將數值設為 null。

您刪除資源時，也會刪除與該資源相關聯的標籤。

Note

請勿將資源標籤與您可以指派給 AWS IoT 物件的屬性混淆。雖然 Greengrass 核心是 AWS IoT 實物，但本主題中所述的資源標籤會連接到 CoreDefinition，而不是核心實物。

搭配 IAM 政策使用標籤

在您的 IAM 政策中，您可以使用資源標籤來控制使用者存取和許可。例如，政策可讓使用者僅能建立具有特定標籤的資源。政策也可以限制使用者建立或修改具有特定標籤的資源。您可以在建立資源時進行標記 (稱為建立時套用標籤)，如此一來，您就不用再執行自訂標記指令碼。使用標籤啟動新環境時，會自動套用對應的 IAM 許可。

下列條件內容鍵和值可用於政策的 Condition 元素 (又稱為 Condition 區塊)。

`greengrass:ResourceTag/tag-key: tag-value`

允許或拒絕資源使用者對具有特定標籤之資源的動作。

`aws:RequestTag/tag-key: tag-value`

要求提出 API 請求，以建立或修改可標記的資源上之標籤時，必須使用 (或未使用) 特定的標籤。

`aws:TagKeys: [tag-key, ...]`

要求提出 API 請求，以建立或修改可標記的資源上之標籤時，必須使用 (或未使用) 特定的一組標籤鍵。

條件內容索引鍵和值只能用於對可標記資源採取動作 AWS IoT Greengrass 的動作。這些動作將資源視為必要參數。例如，您可以設定對 `GetGroupVersion` 的條件式存取。您不能設定 `AssociateServiceRoleToAccount` 的有條件存取，因為請求中沒有參考可標記的資源 (例如群組、核心定義或裝置定義)。

如需詳細資訊，請參閱 [《IAM 使用者指南》中的使用標籤和 IAM JSON 政策參考控制存取](#)。JSON 政策參考包含 IAM 中 JSON 政策的元素、變數和評估邏輯的詳細語法、描述和範例。

範例 IAM 政策

以下範例政策套用標籤式許可，限制試用使用者只能在試用資源上進行動作。

- 第一個陳述式允許 IAM 使用者僅對具有 env=beta 標籤的資源採取行動。
- 第二個陳述式可防止 IAM 使用者從資源中移除 env=beta 標籤。這樣可防止使用者移除自己的存取。

Note

如果您使用標籤控制資源存取，您也應該管理許可，讓使用者能從同樣的資源新增或移除標籤。否則，有時使用者可能會透過修改標籤來避開您的限制，並取得資源的存取。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "greengrass:*",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/env": "beta"
        }
      }
    },
    {
      "Effect": "Deny",
      "Action": "greengrass:UntagResource",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/env": "beta"
        }
      }
    }
  ]
}
```

```
    }
  ]
}
```

若要允許使用者建立資源時套用標籤，您必須提供適當的許可。以下範例政策包含 `greengrass:TagResource` 和 `greengrass:CreateGroup` 行動的 `"aws:RequestTag/env": "beta"` 條件，可讓使用者僅建立標記為 `env=beta` 的群組。這可有效強制使用者標記新群組。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "greengrass:TagResource",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/env": "beta"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": "greengrass:CreateGroup",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/env": "beta"
        }
      }
    }
  ]
}
```

下列指令碼片段說明您可如何將用於標籤鍵的多組標籤值包含在清單中：

```
"StringEquals" : {
  "greengrass:ResourceTag/env" : ["dev", "test"]
}
```

```
}
```

另請參閱

- 中的 [標記 AWS 資源](#) Amazon Web Services 一般參考

CloudFormation 支援 AWS IoT Greengrass

CloudFormation 是一項服務，可協助您建立、管理和複寫 AWS 資源。您可以使用 CloudFormation 範本來定義 AWS IoT Greengrass 群組和用戶端裝置、訂閱，以及您要部署的其他元件。如需範例，請參閱「[the section called “範例 範本”](#)」。

您從範本所產生的資源和基礎設施稱為堆疊。您可以將所有資源定義在一個範本中，或參考其他堆疊的資源。如需 CloudFormation 範本和功能的詳細資訊，請參閱AWS CloudFormation 《使用者指南》中的[什麼是 CloudFormation ?](#)。

建立 資源。

CloudFormation 範本是描述 AWS 資源屬性和關係的 JSON 或 YAML 文件。支援下列 AWS IoT Greengrass 資源：

- 群組
- 核心
- 用戶端裝置（裝置）
- Lambda 函數
- 連接器
- 資源 (本機、機器學習和秘密)
- 訂閱
- 記錄器 (記錄組態)

在 CloudFormation 範本中，Greengrass 資源的結構和語法是以 AWS IoT Greengrass API 為基礎。例如，[範例範本](#)會將頂層DeviceDefinition與DeviceDefinitionVersion包含個別用戶端裝置的 建立關聯。如需詳細資訊，請參閱[the section called “群組物件模型概觀”](#)。

AWS CloudFormation 使用者指南中的[AWS IoT Greengrass 資源類型參考](#)說明您可以使用的 Greengrass 資源 CloudFormation。當您使用 CloudFormation 範本建立 Greengrass 資源時，建議您僅從中管理它們 CloudFormation。例如，如果您想要新增、變更或移除裝置（而不是使用 AWS IoT Greengrass API 或 AWS IoT 主控台），您應該更新範本。這可讓您使用轉返和其他 CloudFormation 變更管理功能。如需使用 CloudFormation 建立和管理資源和堆疊的詳細資訊，請參閱AWS CloudFormation 《使用者指南》中的[使用堆疊](#)。

如需示範如何在 CloudFormation 範本中建立和部署 AWS IoT Greengrass 資源的逐步解說，請參閱 AWS 官方部落格上的[使用 自動化 AWS IoT Greengrass 設定 CloudFormation](#)。

部署資源

建立包含 群組版本的 CloudFormation 堆疊之後，您可以使用 AWS CLI 或 AWS IoT 主控台來部署它。

Note

若要部署群組，您必須擁有與 相關聯的 Greengrass 服務角色 AWS 帳戶。服務角色允許 AWS IoT Greengrass 存取 AWS Lambda 和其他 AWS 服務中的資源。如果您已在目前的 部署 Greengrass 群組，則此角色應該存在 AWS 區域。如需詳細資訊，請參閱[the section called “Greengrass 服務角色”](#)。

部署群組 (AWS CLI)

- 執行 [create-deployment](#) 命令。

```
aws greengrass create-deployment --group-id GroupId --group-version-id GroupVersionId --deployment-type NewDeployment
```

Note

[範例範本](#)中的 CommandToDeployGroup 陳述式示範如何在建立堆疊時，以您的群組和群組版本 IDs 輸出命令。

部署群組 (主控台)

1. 在 AWS IoT 主控台導覽窗格的管理下，展開 Greengrass 裝置，然後選擇群組 (V1)。
2. 選擇群組。
3. 在群組組態頁面上，選擇部署。

範例 範本

下列範例範本會建立 Greengrass 群組，其中包含核心、用戶端裝置、函數、記錄器、訂閱和兩個資源。為此，範本遵循 AWS IoT Greengrass API 的物件模型。例如，您要新增至群組的用戶端裝置會包含在資源中，該 `DeviceDefinitionVersion` 資源與 `DeviceDefinition` 資源相關聯。若要新增裝置到群組，該群組的版本會參考 `DeviceDefinitionVersion` 的 ARN。

範本包含參數，可讓您指定核心和裝置的憑證 ARNs，以及來源 Lambda 函數的版本 ARN（這是 AWS Lambda 資源）。它使用 `Ref` 和 `GetAtt` 內部函數來參考建立 Greengrass 資源所需的 ID、ARN 和其他屬性。

範本也會定義兩個 AWS IoT 裝置（物件），代表新增至 Greengrass 群組的核心和用戶端裝置。

使用 Greengrass 資源建立堆疊之後，您可以使用 AWS CLI 或 AWS IoT 主控台來[部署群組](#)。

Note

範例中的 `CommandToDeployGroup` 陳述式說明如何輸出完整的 `create-deployment` CLI 命令，讓您用來部署群組。

JSON

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Description": "AWS IoT Greengrass example template that creates a group version with a core, device, function, logger, subscription, and resources.",
  "Parameters": {
    "CoreCertificateArn": {
      "Type": "String"
    },
    "DeviceCertificateArn": {
      "Type": "String"
    },
    "LambdaVersionArn": {
      "Type": "String"
    }
  },
  "Resources": {
    "TestCore1": {
      "Type": "AWS::IoT::Thing",
      "Properties": {
```

```

        "ThingName": "TestCore1"
    }
},
"TestCoreDefinition": {
    "Type": "AWS::Greengrass::CoreDefinition",
    "Properties": {
        "Name": "DemoTestCoreDefinition"
    }
},
"TestCoreDefinitionVersion": {
    "Type": "AWS::Greengrass::CoreDefinitionVersion",
    "Properties": {
        "CoreDefinitionId": {
            "Ref": "TestCoreDefinition"
        },
        "Cores": [
            {
                "Id": "TestCore1",
                "CertificateArn": {
                    "Ref": "CoreCertificateArn"
                },
                "SyncShadow": "false",
                "ThingArn": {
                    "Fn::Join": [
                        ":",
                        [
                            "arn:aws:iot",
                            {
                                "Ref": "AWS::Region"
                            },
                            {
                                "Ref": "AWS::AccountId"
                            },
                            "thing/TestCore1"
                        ]
                    ]
                }
            }
        ]
    }
},
"TestClientDevice1": {
    "Type": "AWS::IoT::Thing",
    "Properties": {

```



```

    "ThingName": "TestClientDevice1"
  },
  "TestDeviceDefinition": {
    "Type": "AWS::Greengrass::DeviceDefinition",
    "Properties": {
      "Name": "DemoTestDeviceDefinition"
    }
  },
  "TestDeviceDefinitionVersion": {
    "Type": "AWS::Greengrass::DeviceDefinitionVersion",
    "Properties": {
      "DeviceDefinitionId": {
        "Fn::GetAtt": [
          "TestDeviceDefinition",
          "Id"
        ]
      },
      "Devices": [
        {
          "Id": "TestClientDevice1",
          "CertificateArn": {
            "Ref": "DeviceCertificateArn"
          },
          "SyncShadow": "true",
          "ThingArn": {
            "Fn::Join": [
              ":",
              [
                "arn:aws:iot",
                {
                  "Ref": "AWS::Region"
                },
                {
                  "Ref": "AWS::AccountId"
                },
                "thing/TestClientDevice1"
              ]
            ]
          }
        }
      ]
    }
  }
}

```

```
"TestFunctionDefinition": {
  "Type": "AWS::Greengrass::FunctionDefinition",
  "Properties": {
    "Name": "DemoTestFunctionDefinition"
  }
},
"TestFunctionDefinitionVersion": {
  "Type": "AWS::Greengrass::FunctionDefinitionVersion",
  "Properties": {
    "FunctionDefinitionId": {
      "Fn::GetAtt": [
        "TestFunctionDefinition",
        "Id"
      ]
    },
    "DefaultConfig": {
      "Execution": {
        "IsolationMode": "GreengrassContainer"
      }
    },
    "Functions": [
      {
        "Id": "TestLambda1",
        "FunctionArn": {
          "Ref": "LambdaVersionArn"
        },
        "FunctionConfiguration": {
          "Pinned": "true",
          "Executable": "run.exe",
          "ExecArgs": "argument1",
          "MemorySize": "512",
          "Timeout": "2000",
          "EncodingType": "binary",
          "Environment": {
            "Variables": {
              "variable1": "value1"
            }
          },
          "ResourceAccessPolicies": [
            {
              "ResourceId": "ResourceId1",
              "Permission": "ro"
            },
            {
              "ResourceId": "ResourceId2",
```

```

        "Permission": "rw"
    },
    ],
    "AccessSysfs": "false",
    "Execution": {
        "IsolationMode": "GreengrassContainer",
        "RunAs": {
            "Uid": "1",
            "Gid": "10"
        }
    }
}

}

}

}

}

}

},
    "TestLoggerDefinition": {
        "Type": "AWS::Greengrass::LoggerDefinition",
        "Properties": {
            "Name": "DemoTestLoggerDefinition"
        }
    },
    "TestLoggerDefinitionVersion": {
        "Type": "AWS::Greengrass::LoggerDefinitionVersion",
        "Properties": {
            "LoggerDefinitionId": {
                "Ref": "TestLoggerDefinition"
            },
            "Loggers": [
                {
                    "Id": "TestLogger1",
                    "Type": "AWS::CloudWatch",
                    "Component": "GreengrassSystem",
                    "Level": "INFO"
                }
            ]
        }
    },
    "TestResourceDefinition": {
        "Type": "AWS::Greengrass::ResourceDefinition",
        "Properties": {
            "Name": "DemoTestResourceDefinition"
        }
    }
}

```

```

},
"TestResourceDefinitionVersion": {
  "Type": "AWS::Greengrass::ResourceDefinitionVersion",
  "Properties": {
    "ResourceDefinitionId": {
      "Ref": "TestResourceDefinition"
    },
    "Resources": [
      {
        "Id": "ResourceId1",
        "Name": "LocalDeviceResource",
        "ResourceDataContainer": {
          "LocalDeviceResourceData": {
            "SourcePath": "/dev/TestSourcePath1",
            "GroupOwnerSetting": {
              "AutoAddGroupOwner": "false",
              "GroupOwner": "TestOwner"
            }
          }
        }
      },
      {
        "Id": "ResourceId2",
        "Name": "LocalVolumeResourceData",
        "ResourceDataContainer": {
          "LocalVolumeResourceData": {
            "SourcePath": "/dev/TestSourcePath2",
            "DestinationPath": "/volumes/TestDestinationPath2",
            "GroupOwnerSetting": {
              "AutoAddGroupOwner": "false",
              "GroupOwner": "TestOwner"
            }
          }
        }
      }
    ]
  }
},
"TestSubscriptionDefinition": {
  "Type": "AWS::Greengrass::SubscriptionDefinition",
  "Properties": {
    "Name": "DemoTestSubscriptionDefinition"
  }
},

```

```

    "TestSubscriptionDefinitionVersion": {
      "Type": "AWS::Greengrass::SubscriptionDefinitionVersion",
      "Properties": {
        "SubscriptionDefinitionId": {
          "Ref": "TestSubscriptionDefinition"
        },
        "Subscriptions": [
          {
            "Id": "TestSubscription1",
            "Source": {
              "Fn::Join": [
                ":",
                [
                  "arn:aws:iot",
                  {
                    "Ref": "AWS::Region"
                  },
                  {
                    "Ref": "AWS::AccountId"
                  },
                  "thing/TestClientDevice1"
                ]
              ]
            },
            "Subject": "TestSubjectUpdated",
            "Target": {
              "Ref": "LambdaVersionArn"
            }
          }
        ]
      }
    },
    "TestGroup": {
      "Type": "AWS::Greengrass::Group",
      "Properties": {
        "Name": "DemoTestGroupNewName",
        "RoleArn": {
          "Fn::Join": [
            ":",
            [
              "arn:aws:iam:",
              {
                "Ref": "AWS::AccountId"
              }
            ]
          ]
        }
      }
    }
  }
}

```

```

        "role/TestUser"
    ]
]
},
"InitialVersion": {
    "CoreDefinitionVersionArn": {
        "Ref": "TestCoreDefinitionVersion"
    },
    "DeviceDefinitionVersionArn": {
        "Ref": "TestDeviceDefinitionVersion"
    },
    "FunctionDefinitionVersionArn": {
        "Ref": "TestFunctionDefinitionVersion"
    },
    "SubscriptionDefinitionVersionArn": {
        "Ref": "TestSubscriptionDefinitionVersion"
    },
    "LoggerDefinitionVersionArn": {
        "Ref": "TestLoggerDefinitionVersion"
    },
    "ResourceDefinitionVersionArn": {
        "Ref": "TestResourceDefinitionVersion"
    }
},
"Tags": {
    "KeyName0": "value",
    "KeyName1": "value",
    "KeyName2": "value"
}
}
},
"Outputs": {
    "CommandToDeployGroup": {
        "Value": {
            "Fn::Join": [
                " ",
                [
                    "groupVersion=$(cut -d'/' -f6 <<<",
                    {
                        "Fn::GetAtt": [
                            "TestGroup",
                            "LatestVersionArn"
                        ]
                    }
                ]
            ]
        }
    }
}
}

```

```
    },  
    ");",  
    "aws --region",  
    {  
        "Ref": "AWS::Region"  
    },  
    "greengrass create-deployment --group-id",  
    {  
        "Ref": "TestGroup"  
    },  
    "--deployment-type NewDeployment --group-version-id",  
    "$groupVersion"  
]  
  
}  
  
}
```

YAML

```

AWSTemplateFormatVersion: 2010-09-09

Description: >-
  AWS IoT Greengrass example template that creates a group version with a core,
  device, function, logger, subscription, and resources.

Parameters:
  CoreCertificateArn:
    Type: String
  DeviceCertificateArn:
    Type: String
  LambdaVersionArn:
    Type: String

Resources:
  TestCore1:
    Type: 'AWS::IoT::Thing'
    Properties:
      ThingName: TestCore1
  TestCoreDefinition:
    Type: 'AWS::Greengrass::CoreDefinition'
    Properties:
      Name: DemoTestCoreDefinition
  TestCoreDefinitionVersion:
    Type: 'AWS::Greengrass::CoreDefinitionVersion'

```

```

Properties:
  CoreDefinitionId: !Ref TestCoreDefinition
  Cores:
    - Id: TestCore1
      CertificateArn: !Ref CoreCertificateArn
      SyncShadow: 'false'
      ThingArn: !Join
        - ':'
        - - 'arn:aws:iot'
          - !Ref 'AWS::Region'
          - !Ref 'AWS::AccountId'
          - thing/TestCore1
TestClientDevice1:
  Type: 'AWS::IoT::Thing'
  Properties:
    ThingName: TestClientDevice1
TestDeviceDefinition:
  Type: 'AWS::Greengrass::DeviceDefinition'
  Properties:
    Name: DemoTestDeviceDefinition
TestDeviceDefinitionVersion:
  Type: 'AWS::Greengrass::DeviceDefinitionVersion'
  Properties:
    DeviceDefinitionId: !GetAtt
      - TestDeviceDefinition
      - Id
    Devices:
      - Id: TestClientDevice1
        CertificateArn: !Ref DeviceCertificateArn
        SyncShadow: 'true'
        ThingArn: !Join
          - ':'
          - - 'arn:aws:iot'
            - !Ref 'AWS::Region'
            - !Ref 'AWS::AccountId'
            - thing/TestClientDevice1
TestFunctionDefinition:
  Type: 'AWS::Greengrass::FunctionDefinition'
  Properties:
    Name: DemoTestFunctionDefinition
TestFunctionDefinitionVersion:
  Type: 'AWS::Greengrass::FunctionDefinitionVersion'
  Properties:
    FunctionDefinitionId: !GetAtt

```



```

    - TestFunctionDefinition
    - Id
  DefaultConfig:
    Execution:
      IsolationMode: GreengrassContainer
  Functions:
    - Id: TestLambda1
      FunctionArn: !Ref LambdaVersionArn
      FunctionConfiguration:
        Pinned: 'true'
        Executable: run.exe
        ExecArgs: argument1
        MemorySize: '512'
        Timeout: '2000'
        EncodingType: binary
        Environment:
          Variables:
            variable1: value1
          ResourceAccessPolicies:
            - ResourceId: ResourceId1
              Permission: ro
            - ResourceId: ResourceId2
              Permission: rw
        AccessSysfs: 'false'
        Execution:
          IsolationMode: GreengrassContainer
          RunAs:
            Uid: '1'
            Gid: '10'
  TestLoggerDefinition:
    Type: 'AWS::Greengrass::LoggerDefinition'
    Properties:
      Name: DemoTestLoggerDefinition
  TestLoggerDefinitionVersion:
    Type: 'AWS::Greengrass::LoggerDefinitionVersion'
    Properties:
      LoggerDefinitionId: !Ref TestLoggerDefinition
      Loggers:
        - Id: TestLogger1
          Type: AWSCloudWatch
          Component: GreengrassSystem
          Level: INFO
  TestResourceDefinition:
    Type: 'AWS::Greengrass::ResourceDefinition'

```

```

Properties:
  Name: DemoTestResourceDefinition
TestResourceDefinitionVersion:
  Type: 'AWS::Greengrass::ResourceDefinitionVersion'
Properties:
  ResourceDefinitionId: !Ref TestResourceDefinition
Resources:
  - Id: ResourceId1
    Name: LocalDeviceResource
    ResourceDataContainer:
      LocalDeviceResourceData:
        SourcePath: /dev/TestSourcePath1
        GroupOwnerSetting:
          AutoAddGroupOwner: 'false'
          GroupOwner: TestOwner
  - Id: ResourceId2
    Name: LocalVolumeResourceData
    ResourceDataContainer:
      LocalVolumeResourceData:
        SourcePath: /dev/TestSourcePath2
        DestinationPath: /volumes/TestDestinationPath2
        GroupOwnerSetting:
          AutoAddGroupOwner: 'false'
          GroupOwner: TestOwner
TestSubscriptionDefinition:
  Type: 'AWS::Greengrass::SubscriptionDefinition'
Properties:
  Name: DemoTestSubscriptionDefinition
TestSubscriptionDefinitionVersion:
  Type: 'AWS::Greengrass::SubscriptionDefinitionVersion'
Properties:
  SubscriptionDefinitionId: !Ref TestSubscriptionDefinition
Subscriptions:
  - Id: TestSubscription1
    Source: !Join
      - ':'
      - - 'arn:aws:iot'
        - !Ref 'AWS::Region'
        - !Ref 'AWS::AccountId'
        - thing/TestClientDevice1
    Subject: TestSubjectUpdated
    Target: !Ref LambdaVersionArn
TestGroup:
  Type: 'AWS::Greengrass::Group'

```

```

Properties:
  Name: DemoTestGroupNewName
  RoleArn: !Join
    - ':'
    - - 'arn:aws:iam:'
      - !Ref 'AWS::AccountId'
      - role/TestUser
  InitialVersion:
    CoreDefinitionVersionArn: !Ref TestCoreDefinitionVersion
    DeviceDefinitionVersionArn: !Ref TestDeviceDefinitionVersion
    FunctionDefinitionVersionArn: !Ref TestFunctionDefinitionVersion
    SubscriptionDefinitionVersionArn: !Ref TestSubscriptionDefinitionVersion
    LoggerDefinitionVersionArn: !Ref TestLoggerDefinitionVersion
    ResourceDefinitionVersionArn: !Ref TestResourceDefinitionVersion
  Tags:
    KeyName0: value
    KeyName1: value
    KeyName2: value
Outputs:
  CommandToDeployGroup:
    Value: !Join
      - ' '
      - - groupVersion=$(cut -d'/' -f6 <<<
        - !GetAtt
          - TestGroup
          - LatestVersionArn
        - );
        - aws --region
        - !Ref 'AWS::Region'
        - greengrass create-deployment --group-id
        - !Ref TestGroup
        - '--deployment-type NewDeployment --group-version-id'
        - $groupVersion

```

支援的 AWS 區域

目前，您只能在下列 [AWS 區域](#) 中建立和管理 AWS IoT Greengrass 資源：

- 美國東部 (俄亥俄)
- 美國東部 (維吉尼亞北部)
- 美國西部 (奧勒岡)

- 亞太區域 (孟買)
- 亞太區域 (首爾)
- 亞太區域 (新加坡)
- 亞太區域 (雪梨)
- 亞太區域 (東京)
- 中國 (北京)
- 歐洲 (法蘭克福)
- 歐洲 (愛爾蘭)
- 歐洲 (倫敦)
- AWS GovCloud (美國西部)

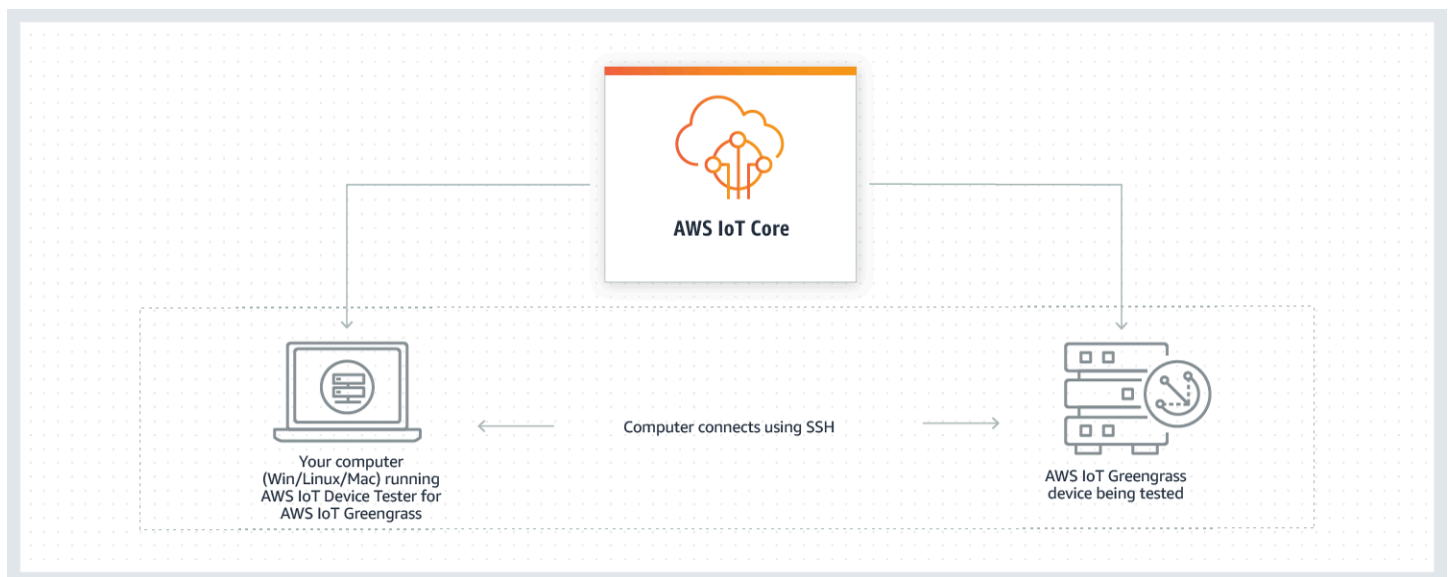
使用 AWS IoT Device Tester for AWS IoT Greengrass V1

AWS IoT Device Tester (IDT) 是可下載的測試架構，可讓您驗證 IoT 裝置。因為 AWS IoT Greengrass Version 1 已移至[維護模式](#)，所以 IDT for AWS IoT Greengrass V1 不再產生已簽署的資格報告。您將無法再透過 AWS IoT Greengrass V1 裝置資格計劃，讓新裝置符合[AWS Partner 在裝置目錄中](#)列出的[AWS 資格](#)。不過，您可以繼續使用 IDT for AWS IoT Greengrass V1 來測試您的 Greengrass V1 裝置。我們建議您使用 [IDT for AWS IoT Greengrass V2](#) 在 Device [AWS Partner Catalog](#) 中[限定和列出 Greengrass 裝置](#)。

IDT for 會在連接到要測試之裝置的主機電腦 (Windows、macOS 或 Linux) AWS IoT Greengrass 上執行。它會執行測試並彙總結果。它還提供命令列界面來管理測試程序。

AWS IoT Greengrass 資格套件

使用 IDT for AWS IoT Greengrass 來驗證 AWS IoT Greengrass Core 軟體是否在您的硬體上執行，並且可以與 通訊 AWS 雲端。它也會使用 end-to-end 測試 AWS IoT Core。例如，它會驗證您的裝置是否可以傳送和接收 MQTT 訊息，並正確處理它們。



AWS IoT Device Tester for 使用測試套件和測試群組的概念來 AWS IoT Greengrass 組織測試。

- 測試套件是一組測試群組，用來驗證裝置是否適用於特定版本的 AWS IoT Greengrass。
- 測試群組是一組與特定功能相關的個別測試，例如 Greengrass 群組部署和 MQTT 簡訊。

如需詳細資訊，請參閱[使用 IDT AWS IoT Greengrass 執行資格套件](#)。

自訂測試套件

從 IDT v4.0.0 開始，IDT for AWS IoT Greengrass 結合了標準化的組態設定和結果格式，以及測試套件環境，可讓您為裝置和裝置軟體開發自訂測試套件。您可以為自己的內部驗證新增自訂測試，或將其提供給客戶進行裝置驗證。

測試寫入器如何設定自訂測試套件，決定執行自訂測試套件所需的設定組態。如需詳細資訊，請參閱[使用 IDT 來開發和執行您自己的測試套件](#)。

AWS IoT Device Tester for AWS IoT Greengrass V1 的支援版本

因為 AWS IoT Greengrass Version 1 已移至[維護模式](#)，所以 IDT for AWS IoT Greengrass V1 不再產生已簽署的資格報告。我們建議您使用 [IDT for AWS IoT Greengrass V2](#)。

如需有關 IDT for AWS IoT Greengrass V2 的資訊，請參閱《AWS IoT Greengrass V2 開發人員指南》中的[使用 AWS IoT Device Tester for AWS IoT Greengrass V2](#)。

Note

如果 IDT for 與您 AWS IoT Greengrass 正在使用的 版本 AWS IoT Greengrass 不相容，您會在啟動測試執行時收到通知。

下載軟體即表示您同意 [AWS IoT Device Tester 授權合約](#)。

for 不支援的 IDT 版本 AWS IoT Greengrass

本主題列出不支援的 IDT for 版本 AWS IoT Greengrass。不支援的版本不會收到錯誤修正或更新。如需詳細資訊，請參閱[the section called “AWS IoT Device Tester for 的支援政策 AWS IoT Greengrass V1”](#)。

IDT v4.4.1 適用於 AWS IoT Greengrass v1.11.6、v1.10.5

版本備註：

- 可讓您驗證並限定執行 AWS IoT Greengrass 核心軟體 v1.11.6 和 v1.10.5 的裝置。
- 包含次要錯誤修正。

測試套件版本：

GGQ_1.3.1

- 已發佈 2021.12.20

IDT v4.1.0 適用於 AWS IoT Greengrass v1.11.4、v1.10.4

版本備註：

- 可讓您驗證和限定執行 AWS IoT Greengrass 核心軟體 v1.11.4 和 v1.10.4 的裝置。
- 修正導致測試執行期間顯示的日誌使用備援標籤的問題。

測試套件版本：

GGQ_1.3.0

- 已發佈 2021.06.23
- 新增對 Lambda、IAM 的 API 呼叫重試，並改善調節或伺服器問題的 AWS STS 處理。
- 將對 Python 3.8 的支援新增至 ML 和 Docker 測試案例。

適用於 v1.11.1、v1.11.0、v1.10.3 AWS IoT Greengrass 版本 IDT v4.0.2

版本備註：

- 修正導致 IDT 遮罩硬體安全整合 (HSI) 錯誤的問題。
- 可讓您使用 AWS IoT Device Tester for 開發和執行自訂測試套件 AWS IoT Greengrass。如需詳細資訊，請參閱[使用 IDT 來開發和執行您自己的測試套件](#)。
- 提供 macOS 和 Windows 的程式碼簽署 IDT 應用程式。在 macOS 中，如果顯示安全警告訊息，您可能需要授予 IDT 的安全例外狀況。如需詳細資訊，請參閱[macOS 上的安全例外狀況](#)。

 Note

AWS IoT Greengrass 不提供核心軟體 1.11.1 版的 AWS IoT Greengrass Dockerfile 或 Docker 映像。若要測試您的裝置是否符合 Docker 資格，請使用舊版 AWS IoT Greengrass 的核心軟體。

IDT v3.2.0 適用於 AWS IoT Greengrass v1.11.0、v1.10.1、v1.10.0

版本備註：

- 根據預設，IDT 只會執行資格所需的測試。若要符合其他功能的資格，您可以修改 [device.json](#) 檔案。

- 在 中新增了連接埠號碼 `device.json`，您可以為 SSH 連線設定該號碼。
- Docker 僅支援[串流管理員](#)和機器學習 (ML)，無需容器化。容器、Docker 和硬體安全整合 (HSI) 不適用於 Docker 裝置。
- 我們將 `device-ml.json` 和 合併 `device-hsm.json` 到 `device.json`。

IDT v3.1.3 適用於 AWS IoT Greengrass 版本：v1.10.x、v1.9.x、v1.8.x

版本備註：

- 新增對 v1 AWS IoT Greengrass .10.x 和 v1.9.x 的 ML 功能資格的支援。您現在可以使用 IDT 來驗證您的裝置是否可以使用在雲端中儲存並訓練的模型，在本機執行 ML 推論。
- 已為 `run-suite` 命令新增 `--stop-on-first-failure`。您可以使用此選項來設定 IDT 在第一次失敗時停止執行。我們建議在測試群組層級的偵錯階段使用此選項。
- 新增 MQTT 測試的時鐘偏離檢查，以確保測試中的裝置使用正確的系統時間。使用的時間必須在可接受的時間範圍內。
- 已為 `run-suite` 命令新增 `--update-idt`。您可以使用此選項來設定提示更新 IDT 的回應。
- 已為 `run-suite` 命令新增 `--update-managed-policy`。您可以使用此選項來設定提示的回應，以更新受管政策。
- 新增 IDT 測試套件版本的自動更新錯誤修正。此修正可確保 IDT 可以執行適用於您 AWS IoT Greengrass 版本的最新測試套件。

適用於的 IDT v3.0.1 AWS IoT Greengrass

版本備註：

- 新增對 AWS IoT Greengrass v1.10.1 的支援。
- IDT 測試套件版本的自動更新。IDT 可以下載適用於您 AWS IoT Greengrass 版本的最新測試套件。此功能包括：
 - 測試套件會使用 *major.minor.patch* 格式進行版本化。初始測試套件版本為 GGQ_1.0.0。
 - 您可以在命令列介面中以互動方式下載新的測試套件，或在啟動 IDT 時設定 `upgrade-test-suite` 旗標。

如需詳細資訊，請參閱[the section called “測試套件版本”](#)。

- 新增了 `list-supported-products`。您可以使用此命令來列出 AWS IoT Greengrass 並測試 IDT 已安裝版本所支援的套件版本。

- 新增了 `list-test-cases`。您可以使用此命令來列出測試群組中可用的測試案例。
- 已為 `run-suite` 命令新增 `test-id`。您可以使用此選項來執行測試群組中的個別測試案例。

適用於 v1.10、v1.9.x 和 v1 AWS IoT Greengrass .8.x 的 IDT v2.3.0

在實體裝置上測試時，支援 AWS IoT Greengrass v1.10、v1.9.x 和 v1.8.x。

在 Docker 容器中測試時，支援 AWS IoT Greengrass v1.10 和 v1.9.x。

版本備註：

- 新增了對 [the section called “ AWS IoT Greengrass 在 Docker 容器中執行”](#) 的支援。您現在可以使用 IDT 來限定和驗證您的裝置是否可以 AWS IoT Greengrass 在 Docker 容器中執行。
- 新增 [AWS 受管政策](#) (`AWSIoTDeviceTesterForGreengrassFullAccess`)，定義執行 AWS IoT Device Tester 所需的許可。如果新版本需要額外許可，會將它們 AWS 新增至此受管政策，因此您不需要更新 IAM 許可。
- 在您執行測試案例之前，先採用檢查來驗證是否已正確設定環境 (例如，裝置連線能力和網際網路連線能力)。
- 改善 IDT 中的 Greengrass 相依性檢查工具，使其在檢查裝置上的 libc 時更加靈活。

適用於 v1.10、v1.9.x 和 v1 AWS IoT Greengrass .8.x 的 IDT v2.2.0

版本備註：

- 新增對 AWS IoT Greengrass v1.10 的支援。
- 新增對 [Greengrass Docker 應用程式部署](#) 連接器的支援。
- 新增對 AWS IoT Greengrass [串流管理員](#) 的支援。
- 已新增 AWS IoT Greengrass 對中國（北京）區域的支援。

適用於 v1.9.x、v1.8.x 和 v1 AWS IoT Greengrass .7.x 的 IDT v2.1.0

版本備註：

- 新增對 AWS IoT Greengrass v1.9.4 的支援。
- 新增對 Linux-ARMv6l 裝置的支援。

適用於 v1.9.3、v1 AWS IoT Greengrass .9.2、v.1.9.1、v1.9.0、v1.8.4、v1.8.3 和 v1.8.2 的 IDT v2.0.0

版本備註：

- 已移除待測裝置對 Python 的相依性。
- 測試套件的執行時間減少 50% 以上，使得資格程序更快。
- 可執行大小減少 50% 以上，使得下載和安裝更快。
- 已改善所有測試案例的[逾時乘數支援](#)。
- 增強診斷後訊息，以更快地對錯誤進行故障診斷。
- 更新執行 IDT 所需的許可政策範本。
- 新增對 AWS IoT Greengrass v1.9.3 的支援。

適用於 v1.9.2、v1 AWS IoT Greengrass .9.1、v1.9.0、v1.8.3 和 v1.8.2 的 IDT v1.3.3

版本備註：

- 新增對 Greengrass 1.9.2 版與 v1.8.3 版的支援。
- 新增對 Greengrass OpenWRT 版的支援。
- 新增 SSH 使用者名稱和密碼裝置登入。
- 新增 OpenWrt-ARMv7l 平台的原生測試錯誤修正。

適用於 v1.8.1 的 IDT AWS IoT Greengrass v1.2

版本備註：

- 新增可設定的逾時乘數來處理並解決逾時問題 (例如，低頻寬連線)。

適用於 v1.8.0 的 IDT AWS IoT Greengrass v1.1

版本備註：

- 新增 AWS IoT Greengrass 硬體安全整合 (HSI) 的支援。
- 已新增對 AWS IoT Greengrass 容器的支援，且沒有 容器。

- 新增自動 AWS IoT Greengrass 建立服務角色。
- 改善測試資源清理。
- 新增測試執行摘要報告。

適用於 v1.7.1 的 IDT AWS IoT Greengrass v1.1

版本備註：

- 新增 AWS IoT Greengrass 硬體安全整合 (HSI) 的支援。
- 已新增對 AWS IoT Greengrass 容器的支援，且沒有 容器。
- 新增自動 AWS IoT Greengrass 建立服務角色。
- 改善測試資源清理。
- 新增測試執行摘要報告。

適用於 v1.6.1 的 IDT AWS IoT Greengrass v1.0

版本備註：

- 新增了 OTA 測試錯誤修正，以確保未來 AWS IoT Greengrass 版本相容性。

Note

如果您使用適用於 v1.6.1 的 IDT v1 AWS IoT Greengrass .0，則必須建立 [Greengrass 服務角色](#)。在更新的版本中，IDT 會為您建立服務角色。

使用 IDT AWS IoT Greengrass 執行資格套件

您可以使用 AWS IoT Device Tester (IDT) for AWS IoT Greengrass 來驗證 AWS IoT Greengrass Core 軟體是否在您的硬體上執行，並且可以與 通訊 AWS 雲端。也會使用 AWS IoT Core 執行端對端測試。例如，它會驗證您的裝置是否可以傳送和接收 MQTT 訊息，並正確處理它們。

因為 AWS IoT Greengrass Version 1 已移至[維護模式](#)，所以 IDT for AWS IoT Greengrass V1 不再產生已簽署的資格報告。如果您想要將硬體新增至 AWS Partner Device Catalog，請執行 AWS IoT Greengrass V2 資格套件來產生可提交的測試報告 AWS IoT。如需詳細資訊，請參閱 [AWS 裝置資格計劃](#)和 [IDT for 的支援版本 AWS IoT Greengrass V2](#)。

除了測試裝置之外，IDT for 還會在您的 中 AWS IoT Greengrass 建立資源（例如 AWS IoT 物件、AWS IoT Greengrass 群組、Lambda 函數等）AWS 帳戶，以促進資格程序。

若要建立這些資源，IDT for AWS IoT Greengrass 會使用 `config.json` 檔案中設定的 AWS 登入資料，代表您進行 API 呼叫。系統會在測試期間的不同時間點內佈建這些資源。

當您使用 IDT for AWS IoT Greengrass AWS IoT Greengrass 執行資格套件時，IDT 會執行下列步驟：

1. 載入並驗證您的裝置和登入資料組態。
2. 對所需的本機和雲端資源執行選取的測試。
3. 清除本機和雲端資源。
4. 產生測試報告，指出您的裝置是否通過資格所需的測試。

測試套件版本

IDT for 會將測試 AWS IoT Greengrass 組織成測試套件和測試群組。

- 測試套件是一組測試群組，用來驗證裝置是否適用於特定版本的 AWS IoT Greengrass。
- 測試群組是一組與特定功能相關的個別測試，例如 Greengrass 群組部署和 MQTT 簡訊。

從 IDT v3.0.0 開始，測試套件使用 *major.minor.patch* 格式進行版本化，例如，GGQ_1.0.0。當您下載 IDT 時，套件會包含最新的測試套件版本。

Important

IDT 支援三種最新的測試套件版本，以符合裝置資格。如需詳細資訊，請參閱[the section called “AWS IoT Device Tester for 的支援政策 AWS IoT Greengrass V1”](#)。

您可以執行 `list-supported-products` 來列出目前 IDT 版本支援的 AWS IoT Greengrass 和 測試套件版本。來自不支援的測試套件版本的測試不符合裝置資格。IDT 不會列印不支援版本的資格報告。

IDT 組態設定的更新

新測試可能會引入新的 IDT 組態設定。

- 如果這些是選用設定，IDT 會繼續執行測試。

- 如果這些是必要設定，IDT 會通知您並停止執行。設定組態之後，請重新啟動測試回合。

組態設定位於 `<device-tester-extract-location>/configs` 資料夾中。如需詳細資訊，請參閱[the section called “設定 IDT 設定”](#)。

如果更新的測試套件版本新增了組態設定，IDT 會在 `<device-tester-extract-location>/configs` 中建立原始組態檔的副本。

測試群組描述

IDT v2.0.0 and later

核心資格的必要測試群組

這些測試群組是讓您的 AWS IoT Greengrass 裝置符合 AWS Partner Device Catalog 資格的必要項目。

AWS IoT Greengrass 核心相依性

驗證您的裝置是否符合 AWS IoT Greengrass Core 軟體的所有軟體和硬體需求。

在 [Docker 容器中](#) Software Packages Dependencies 測試時，此測試群組中的測試案例不適用。

部署

驗證 Lambda 函數是否可以部署在您的裝置上。

MQTT

檢查 Greengrass 核心和用戶端裝置之間的本機通訊，以驗證 AWS IoT Greengrass 訊息路由功能，這些裝置是本機 IoT 裝置。

無線 (OTA)

驗證您的裝置是否可以成功執行 AWS IoT Greengrass 核心軟體的 OTA 更新。

在 [Docker 容器](#) 中進行測試時，此測試群組不適用。

版本

檢查 AWS IoT Greengrass 提供的版本是否與您正在使用的 AWS IoT Device Tester 版本相容。

選用測試群組

這些測試群組是選用的。如果您選擇符合選用測試的資格，您的裝置會在 AWS Partner Device Catalog 中列出其他功能。

容器相依性

驗證裝置是否符合在 Greengrass 核心上以容器模式執行 Lambda 函數的所有軟體和硬體需求。

在 [Docker 容器](#) 中進行測試時，此測試群組不適用。

部署容器

驗證 Lambda 函數是否可以部署在裝置上，並在 Greengrass 核心上以容器模式執行。

在 [Docker 容器](#) 中進行測試時，此測試群組不適用。

Docker 相依性 (對 IDT v2.2.0 及更新版本的支援)

驗證裝置是否符合所有必要的技術相依性，以使用 Greengrass Docker 應用程式部署連接器來執行容器

在 [Docker 容器](#) 中進行測試時，此測試群組不適用。

硬體安全整合 (HSI)

驗證提供的 HSI 共用程式庫是否可以與硬體安全模組 (HSM) 溝通，並正確實作所需的 PKCS#11 API。HSM 和共用程式庫必須能夠簽署 CSR、執行 TLS 操作並提供正確的金鑰長度和公開金鑰演算法。

串流管理員相依性 (支援 IDT v2.2.0 及更新版本)

驗證裝置是否符合執行 AWS IoT Greengrass 串流管理員所需的所有技術相依性。

機器學習相依性 (IDT v3.1.0 及更新版本支援)

驗證裝置符合所有必要的技術相依性，以便在本機執行 ML 推論。

機器學習推論測試 (IDT v3.1.0 及更新版本支援)

驗證 ML 推斷可以在給定受測裝置上執行。如需詳細資訊，請參閱 [the section called “選用：設定您的裝置以取得 ML 資格”](#)。

機器學習推論容器測試 (IDT v3.1.0 及更新版本支援)

驗證 ML 推斷可以在給定受測裝置上執行，並在 Greengrass 核心的容器模式上執行。如需詳細資訊，請參閱 [the section called “選用：設定您的裝置以取得 ML 資格”](#)。

IDT v1.3.3 and earlier

核心資格的必要測試群組

需要這些測試才能讓您的 AWS IoT Greengrass 裝置符合 AWS Partner Device Catalog 的資格。

AWS IoT Greengrass 核心相依性

驗證您的裝置是否符合 AWS IoT Greengrass Core 軟體的所有軟體和硬體需求。

組合 (裝置安全互動)

在雲端變更 Greengrass 群組的連線資訊，以驗證 Greengrass 核心裝置上憑證管理員和 IP 偵測的功能。測試群組會輪換 AWS IoT Greengrass 伺服器憑證並驗證 是否 AWS IoT Greengrass 允許連線。

部署 (IDT 第 1.2 版及更早版本的必要項目)

驗證 Lambda 函數是否可以部署在您的裝置上。

Device Certificate Manager (DCM)

驗證 AWS IoT Greengrass 裝置憑證管理員是否可以在啟動時產生伺服器憑證，並在憑證即將過期時輪換憑證。

IP 偵測 (IPD)

Greengrass 核心裝置中的 IP 地址變更時，會驗證核心連線資訊是否更新。如需詳細資訊，請參閱[啟動自動 IP 偵測](#)。

日誌

驗證 AWS IoT Greengrass 記錄服務是否可以使用以 Python 撰寫的使用者 Lambda 函數寫入日誌檔案。

MQTT

透過傳送路由到兩個 Lambda 函數之主題上的訊息，來驗證 AWS IoT Greengrass 訊息路由器功能。

原生

驗證 AWS IoT Greengrass 可以執行原生（編譯）Lambda 函數。

無線 (OTA)

驗證您的裝置是否可以成功執行 AWS IoT Greengrass 核心軟體的 OTA 更新。

滲透

如果未啟用硬連結/軟連結保護和 [seccomp](#)，則驗證 AWS IoT Greengrass Core 軟體是否無法啟動。它也用來驗證其他安全相關功能。

影子

驗證本機陰影和陰影雲端同步功能。

多工緩衝處理區域

驗證 MQTT 訊息是否依預設的多工緩衝處理區域組態佇列。

字符交換服務 (TES)

驗證 AWS IoT Greengrass 是否可以交換其核心憑證以取得有效的 AWS 憑證。

版本

檢查 AWS IoT Greengrass 提供的 版本是否與您正在使用的 AWS IoT Device Tester 版本相容。

選用測試群組

這些測試是選用的。如果您選擇符合選用測試的資格，您的裝置會在 AWS Partner Device Catalog 中列出其他功能。

容器相依性

檢查裝置是否符合在容器模式下執行 Lambda 函數所需的所有必要相依性。

硬體安全整合 (HSI)

驗證提供的 HSI 共用程式庫是否可以與硬體安全模組 (HSM) 溝通，並正確實作所需的 PKCS#11 API。HSM 和共用程式庫必須能夠簽署 CSR、執行 TLS 操作並提供正確的金鑰長度和公開金鑰演算法。

本機資源存取

透過透過 LRA API，將各種 Linux 使用者和群組擁有的本機檔案和目錄的存取權提供給容器化 Lambda 函數，AWS IoT Greengrass 以驗證 的本機資源存取 (AWS IoT Greengrass LRA) 功能。APIs 應允許或拒絕 Lambda 函數根據本機資源存取組態存取本機資源。

網路

驗證是否可以從 Lambda 函數建立通訊端連線。這些通訊端連線應該根據 Greengrass 核心組態而允許或拒絕。

執行 AWS IoT Greengrass 資格套件的先決條件

本節說明使用 AWS IoT Device Tester (IDT) for AWS IoT Greengrass AWS IoT Greengrass 執行資格套件的先決條件。

下載適用於 的最新版 AWS IoT Device Tester AWS IoT Greengrass

下載[最新版本](#)的 IDT，並將軟體解壓縮到檔案系統上具有讀取和寫入許可的位置。

Note

IDT 不支援由多位使用者從共用位置執行，例如 NFS 目錄或 Windows 網路共用資料夾。我們建議您將 IDT 套件解壓縮到本機磁碟機，並在本機工作站上執行 IDT 二進位檔。Windows 的路徑長度限制為 260 個字元。如果您使用的是 Windows，請將 IDT 解壓縮到根目錄，例如 C:\ 或 D:\，使路徑保持在 260 個字元的限制以下。

建立和設定 AWS 帳戶

您必須先執行下列步驟 AWS IoT Greengrass，才能使用 IDT for：

1. [建立 AWS 帳戶](#)。如果您已有 AWS 帳戶，請跳到步驟 2。
2. [設定 IDT 的許可](#)。

這些帳戶許可允許 IDT 代表您存取 AWS 服務和建立 AWS 資源，例如 AWS IoT 物件、Greengrass 群組和 Lambda 函數。

若要建立這些資源，IDT for AWS IoT Greengrass 會使用 config.json 檔案中設定的 AWS 登入資料來代表您進行 API 呼叫。系統會在測試期間的不同時間點內佈建這些資源。

Note

雖然大多數測試都符合 [Amazon Web Services 免費方案](#) 的資格，但您必須在註冊時提供信用卡 AWS 帳戶。如需詳細資訊，請參閱[如果我的帳戶適用於免費方案，為何需要付款方式？](#)。

步驟 1：建立 AWS 帳戶

在此步驟中，建立和設定 AWS 帳戶。如果您已有 AWS 帳戶，請跳至 [the section called “步驟 2：設定 IDT 的許可”](#)。

註冊 AWS 帳戶

如果您沒有 AWS 帳戶，請完成下列步驟來建立一個。

註冊 AWS 帳戶

1. 開啟 <https://portal.aws.amazon.com/billing/signup>。
2. 請遵循線上指示進行。

部分註冊程序需接收來電或簡訊，並在電話鍵盤輸入驗證碼。

當您註冊時 AWS 帳戶，AWS 帳戶根使用者會建立。根使用者有權存取該帳戶中的所有 AWS 服務和資源。作為安全最佳實務，請將管理存取權指派給使用者，並且僅使用根使用者來執行[需要根使用者存取權的任務](#)。

AWS 會在註冊程序完成後傳送確認電子郵件給您。您可以隨時登錄 <https://aws.amazon.com/> 並選擇我的帳戶，以檢視您目前的帳戶活動並管理帳戶。

建立具有管理存取權的使用者

註冊後 AWS 帳戶，請保護 AWS 帳戶根使用者、啟用 AWS IAM Identity Center 和建立管理使用者，以免將根使用者用於日常任務。

保護您的 AWS 帳戶根使用者

1. 選擇根使用者並輸入 AWS 帳戶您的電子郵件地址，以帳戶擁有者[AWS 管理主控台](#)身分登入。在下一頁中，輸入您的密碼。

如需使用根使用者登入的說明，請參閱 AWS 登入 使用者指南中的[以根使用者身分登入](#)。

2. 若要在您的根使用者帳戶上啟用多重要素驗證 (MFA)。

如需說明，請參閱《IAM 使用者指南》中的[為您的 AWS 帳戶根使用者（主控台）啟用虛擬 MFA 裝置](#)。

建立具有管理存取權的使用者

1. 啟用 IAM Identity Center。

如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的[啟用 AWS IAM Identity Center](#)。

2. 在 IAM Identity Center 中，將管理存取權授予使用者。

如需使用 IAM Identity Center 目錄 做為身分來源的教學課程，請參閱AWS IAM Identity Center 《使用者指南》中的[使用預設值設定使用者存取 IAM Identity Center 目錄](#)。

以具有管理存取權的使用者身分登入

- 若要使用您的 IAM Identity Center 使用者簽署，請使用建立 IAM Identity Center 使用者時傳送至您電子郵件地址的簽署 URL。

如需使用 IAM Identity Center 使用者登入的說明，請參閱AWS 登入 《使用者指南》中的[登入 AWS 存取入口網站](#)。

指派存取權給其他使用者

- 在 IAM Identity Center 中，建立一個許可集來遵循套用最低權限的最佳實務。

如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的[建立許可集](#)。

- 將使用者指派至群組，然後對該群組指派單一登入存取權。

如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的[新增群組](#)。

步驟 2：設定 IDT 的許可

在此步驟中，設定 IDT for AWS IoT Greengrass 用來執行測試和收集 IDT 用量資料的許可。您可以使用 AWS 管理主控台 或 AWS Command Line Interface (AWS CLI) 來建立 IAM 政策和 IDT 的測試使用者，然後將政策連接到使用者。如果您已經為 IDT 建立測試使用者，請跳至[the section called “設定您的裝置以執行 IDT 測試”](#) 或 [the section called “選用：設定您的 Docker 容器”](#)。

- [設定 IDT \(主控台\) 的許可](#)
- [步驟 2：設定 IDT \(AWS CLI\) 的許可](#)

設定 IDT (主控台) 的許可

請依照下列步驟使用主控台來設定 IDT for AWS IoT Greengrass的許可。

- 登入 [IAM 主控台](#)。
- 建立客戶受管政策，該政策授與建立具有特定許可之角色的許可。

- a. 在導覽窗格中，選擇政策，然後選擇建立政策。
- b. 在 JSON 標籤上，用以下政策取代預留位置內容。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ManageRolePoliciesForIDTGreengrass",
      "Effect": "Allow",
      "Action": [
        "iam:DetachRolePolicy",
        "iam:AttachRolePolicy"
      ],
      "Resource": [
        "arn:aws:iam::*:role/idt-*",
        "arn:aws:iam::*:role/GreengrassServiceRole"
      ],
      "Condition": {
        "ArnEquals": {
          "iam:PolicyARN": [
            "arn:aws:iam::aws:policy/service-role/AWSGreengrassResourceAccessRolePolicy",
            "arn:aws:iam::aws:policy/service-role/GreengrassOTAUpdateArtifactAccess",
            "arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole"
          ]
        }
      }
    },
    {
      "Sid": "ManageRolesForIDTGreengrass",
      "Effect": "Allow",
      "Action": [
        "iam:CreateRole",
        "iam>DeleteRole",
        "iam:PassRole",
        "iam:GetRole"
      ],
    }
  ]
}
```

```
        "Resource": [
            "arn:aws:iam::123456789012:role/idt-*",
            "arn:aws:iam::123456789012:role/GreengrassServiceRole"
        ]
    }
}
```

 Important

下列政策會授予許可來建立和管理 IDT for AWS IoT Greengrass所需的角色。這包括連接下列 AWS 受管政策的許可：

- [AWSGreengrassResourceAccessRolePolicy \(AWSGreengrassResourceAccessRolePolicy\)](#)
- [GreengrassOTAUpdateArtifactAccess](#)
- [AWSLambdaBasicExecutionRole](#)

- c. 選擇下一步：標籤。
 - d. 選擇下一步：檢閱。
 - e. 對於名稱，輸入 **IDTGreengrassIAMPermissions**。在 Summary (摘要) 下，檢閱您的政策所授與的許可。
 - f. 選擇建立政策。
3. 建立 IAM 使用者並連接 IDT for 所需的許可 AWS IoT Greengrass。
 - a. 建立 IAM 使用者。請遵循《IAM 使用者指南》中[建立 IAM 使用者（主控台）](#)的步驟 1 到 5。
 - b. 將許可連接至您的 IAM 使用者：
 - i. 在設定許可頁面上，選擇直接連接現有政策。
 - ii. 搜尋您在上一個步驟中建立的 IDTGreengrassIAMPermissions 政策。選取核取方塊。
 - iii. 搜尋 AWSIoTDeviceTesterForGreengrassFullAccess 政策。選取核取方塊。

Note

[AWSIoTDeviceTesterForGreengrassFullAccess](#) 是一種 AWS 受管政策，定義 IDT 建立和存取用於測試 AWS 的資源所需的許可。如需詳細資訊，請參閱[the section called “AWS IDT 的 受管政策”](#)。

- c. 選擇下一步：標籤。
 - d. 選擇 Next: Review (下一步：檢閱) 以檢視選擇的摘要。
 - e. 選擇 Create user (建立使用者)。
 - f. 若要檢視使用者的存取金鑰 (存取金鑰 ID 和私密存取金鑰)，請選擇密碼和存取金鑰旁的 Show (顯示)。若要儲存存取金鑰，請選擇 Download.csv，並將檔案儲存到安全的位置。您稍後會使用此資訊來設定您的 AWS 登入資料檔案。
4. 下一步：設定您的[實體裝置](#)。

步驟 2：設定 IDT (AWS CLI) 的許可

請依照下列步驟使用 AWS CLI 來設定 IDT for 的許可 AWS IoT Greengrass。如果您已在主控台中設定許可，請跳至[the section called “設定您的裝置以執行 IDT 測試”](#)或[the section called “選用：設定您的 Docker 容器”](#)。

1. 如果尚未安裝，請在電腦上安裝和設定 AWS CLI。請遵循AWS Command Line Interface 《使用者指南》中的[安裝 AWS CLI](#) 中的步驟。

Note

AWS CLI 是開放原始碼工具，可用來與 AWS 命令列 Shell 中的 服務互動。

2. 建立客戶受管政策，以授予管理 IDT 和 AWS IoT Greengrass 角色的許可。

Linux, macOS, or Unix

```
aws iam create-policy --policy-name IDTGreengrassIAMPermissions --policy-document '{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```

        "Sid": "ManageRolePoliciesForIDTGreengrass",
        "Effect": "Allow",
        "Action": [
            "iam:DetachRolePolicy",
            "iam:AttachRolePolicy"
        ],
        "Resource": [
            "arn:aws:iam::*:role/idt-*",
            "arn:aws:iam::*:role/GreengrassServiceRole"
        ],
        "Condition": {
            "ArnEquals": {
                "iam:PolicyARN": [
                    "arn:aws:iam::aws:policy/service-role/
AWSGreengrassResourceAccessRolePolicy",
                    "arn:aws:iam::aws:policy/service-role/
GreengrassOTAUpdateArtifactAccess",
                    "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole"
                ]
            }
        }
    },
    {
        "Sid": "ManageRolesForIDTGreengrass",
        "Effect": "Allow",
        "Action": [
            "iam:CreateRole",
            "iam>DeleteRole",
            "iam:PassRole",
            "iam:GetRole"
        ],
        "Resource": [
            "arn:aws:iam::123456789012:role/idt-*",
            "arn:aws:iam::123456789012:role/GreengrassServiceRole"
        ]
    }
]
}'

```

Windows command prompt

```
aws iam create-policy --policy-name IDTGreengrassIAMPermissions --  
policy-document '{\"Version\": \"2012-10-17\", \"Statement\":  
  [{\"Sid\": \"ManageRolePoliciesForIDTGreengrass\", \"Effect\": \"Allow  
\", \"Action\": [\"iam:DetachRolePolicy\", \"iam:AttachRolePolicy\"],  
  \"Resource\": [\"arn:aws:iam::*:role/idt-*\", \"arn:aws:iam::*:role/  
GreengrassServiceRole\"], \"Condition\": {\"ArnEquals\": {\"iam:PolicyARN\":  
  [\"arn:aws:iam::aws:policy/service-role/AWSGreengrassResourceAccessRolePolicy  
\", \"arn:aws:iam::aws:policy/service-role/GreengrassOTAUpdateArtifactAccess  
\", \"arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole\"]}}},  
  {\"Sid\": \"ManageRolesForIDTGreengrass\", \"Effect\": \"Allow\", \"Action\":  
  [\"iam:CreateRole\", \"iam>DeleteRole\", \"iam:PassRole\", \"iam:GetRole  
\"], \"Resource\": [\"arn:aws:iam::*:role/idt-*\", \"arn:aws:iam::*:role/  
GreengrassServiceRole\"]}]}'
```

Note

此步驟包含 Windows 命令提示字元範例，因為它使用的 JSON 語法與 Linux、macOS 或 Unix 終端機命令不同。

3. 建立 IAM 使用者並連接 IDT for 所需的許可 AWS IoT Greengrass。

a. 建立 IAM 使用者。在此範例設定中，使用者命名為 IDTGreengrassUser。

```
aws iam create-user --user-name IDTGreengrassUser
```

b. 將您在步驟 2 中建立 IDTGreengrassIAMPermissions 的政策連接到您的 IAM 使用者。使用的 ID 取代命令中的 *<account-id>* AWS 帳戶。

```
aws iam attach-user-policy --user-name IDTGreengrassUser --policy-arn  
arn:aws:iam::<account-id>:policy/IDTGreengrassIAMPermissions
```

c. 將 AWSIoTDeviceTesterForGreengrassFullAccess 政策連接至您的 IAM 使用者。

```
aws iam attach-user-policy --user-name IDTGreengrassUser --policy-arn  
arn:aws:iam::aws:policy/AWSIoTDeviceTesterForGreengrassFullAccess
```


 Note

[AWSIoTDeviceTesterForGreengrassFullAccess](#) 是一種 AWS 受管政策，定義 IDT 建立和存取用於測試 AWS 的資源所需的許可。如需詳細資訊，請參閱[the section called “AWS IDT 的 受管政策”](#)。

4. 為使用者建立私密存取金鑰。

```
aws iam create-access-key --user-name IDTGreengrassUser
```

將輸出儲存在安全的位置。您稍後會使用此資訊來設定您的 AWS 登入資料檔案。

5. 下一步：設定您的[實體裝置](#)。

AWS AWS IoT Device Tester 的 受管政策

[AWSIoTDeviceTesterForGreengrassFullAccess](#) 受管政策允許 IDT 執行操作和收集用量指標。此政策可授予下列 IDT 許可：

- `iot-device-tester:CheckVersion`。檢查一組 AWS IoT Greengrass、測試套件和 IDT 版本是否相容。
- `iot-device-tester:DownloadTestSuite`。下載測試套件。
- `iot-device-tester:LatestIdt`。取得可供下載的最新 IDT 版本相關資訊。
- `iot-device-tester:SendMetrics`。發佈 IDT 收集的有關測試的使用資料。
- `iot-device-tester:SupportedVersion`。取得 IDT 支援的 AWS IoT Greengrass 和 測試套件版本清單。此資訊會顯示在命令列視窗中。

設定您的裝置以執行 IDT 測試

若要設定您的裝置，您必須安裝 AWS IoT Greengrass 相依性、設定 AWS IoT Greengrass Core 軟體、設定主機電腦以存取您的裝置，以及在裝置上設定使用者許可。

驗證待測裝置的 AWS IoT Greengrass 相依性

在 IDT for AWS IoT Greengrass 可以測試您的裝置之前，請確定您已如 [入門 AWS IoT Greengrass](#) 中所述設定您的裝置。如需受支援平台的相關資訊，請參閱[支援的平台](#)。

設定 AWS IoT Greengrass 軟體

IDT for 會 AWS IoT Greengrass 測試您的裝置與特定 版本的相容性 AWS IoT Greengrass。IDT 提供兩種 AWS IoT Greengrass 在裝置上進行測試的選項：

- 下載並使用某個版本的 [AWS IoT Greengrass Core 軟體](#)。IDT 會為您安裝此軟體。
- 使用您裝置上已安裝的 AWS IoT Greengrass Core 軟體版本。

Note

的每個 版本 AWS IoT Greengrass 都有對應的 IDT 版本。您必須下載對應 AWS IoT Greengrass 至您正在使用之 版本的 IDT 版本。

下列各節描述這些選項。您只需要執行一個選項。

選項 1：下載 AWS IoT Greengrass Core 軟體並設定 AWS IoT Device Tester 使用它

您可以從 AWS IoT Greengrass 核心軟體下載頁面下載[AWS IoT Greengrass 核心軟體](#)。

1. 尋找正確的架構和 Linux 發行版本，然後選擇 Download (下載)。
2. 將 tar.gz 檔案複製到 `<device-tester-extract-location>/products/greengrass/ggc`。

Note

請勿變更 AWS IoT Greengrass tar.gz 檔案的名稱。請勿將相同作業系統和架構的多個檔案放在這個目錄中。例如，將 `greengrass-linux-armv7l-1.7.1.tar.gz` 和 `greengrass-linux-armv7l-1.8.1.tar.gz` 檔案放在該目錄中將導致測試失敗。

選項 2：AWS IoT Greengrass 搭配 AWS IoT Device Tester 使用 的現有安裝

設定 IDT 將 `greengrassLocation` 屬性新增至 `<device-tester-extract-location>/configs` 資料夾中的 `device.json` 檔案，以測試安裝在裝置上的 AWS IoT Greengrass Core 軟體。例如：

```
"greengrassLocation" : "<path-to-greengrass-on-device>"
```

如需 `device.json` 詳細資訊，請參閱 [設定 device.json](#)。

在 Linux 裝置上，AWS IoT Greengrass 核心軟體的預設位置為 `/greengrass`。

Note

您的裝置應安裝尚未啟動的 AWS IoT Greengrass Core 軟體。

請確定您已在裝置上新增 `ggc_user` 使用者和 `ggc_group`。如需詳細資訊，請參閱 [AWS IoT Greengrass 的環境設定](#)。

設定主機電腦以存取待測裝置

IDT 是在您的主機電腦上執行，而且必須能夠使用 SSH 連線到您的裝置。有兩個選項允許 IDT 取得待測裝置的 SSH 存取權：

1. 依照此處的指示來建立 SSH 金鑰對，並授權您的金鑰可以登入待測裝置，無需指定密碼。
2. 提供 `device.json` 檔案中每個裝置的使用者名稱和密碼。如需詳細資訊，請參閱 [設定 device.json](#)。

您可以使用任何 SSL 實作來建立 SSH 金鑰。以下指示展示如何使用 [SSH-KEYGEN](#) 或 [PuTTYgen](#) (適用於 Windows)。如果您使用的是另一個 SSL 實作，請參閱該實作的文件。

IDT 使用 SSH 金鑰向待測裝置進行驗證。

使用 SSH-KEYGEN 建立 SSH 金鑰

1. 建立 SSH 金鑰。

您可以使用 Open SSH `ssh-keygen` 命令建立 SSH 金鑰對。如果您的主機電腦上已有 SSH 金鑰對，則最佳實務是特別為 IDT 建立 SSH 金鑰對。如此一來，在您完成測試之後，若沒有輸入密碼，主機電腦再也無法連接至您的裝置。它還可讓您限制只有需要遠端裝置的人，才能存取該裝置。

Note

Windows 沒有安裝的 SSH 用戶端。如需在 Windows 上安裝 SSH 用戶端的詳細資訊，請參閱 [下載 SSH 用戶端軟體](#)。

ssh-keygen 命令會提示您提供金鑰對的存放名稱和路徑。根據預設，該金鑰對檔案會命名為 id_rsa (私有金鑰) 和 id_rsa.pub (公有金鑰)。在 macOS 和 Linux 上，這些檔案的預設位置是 ~/.ssh/。在 Windows 上，預設位置為 C:\Users*<user-name>*\.ssh。

出現提示時，請輸入金鑰字詞來保護您的 SSH 金鑰。如需詳細資訊，請參閱[產生新的 SSH 金鑰](#)。

2. 將授權的 SSH 金鑰新增至待測裝置。

IDT 必須使用您的 SSH 私有金鑰登入待測裝置。請從您的主機電腦使用 ssh-copy-id 命令，授權您的 SSH 私有金鑰登入待測裝置。此命令會將您的公有金鑰新增至待測裝置上的 ~/.ssh/authorized_keys 檔案。例如：

```
$ ssh-copy-id <remote-ssh-user>@<remote-device-ip>
```

remote-ssh-user 是用來登入測試裝置的使用者名稱，而 *remote-device-ip* 是要執行測試的測試裝置 IP 地址。例如：

```
ssh-copy-id pi@192.168.1.5
```

出現提示時，請輸入您在 ssh-copy-id 命令中指定的使用者名稱密碼。

ssh-copy-id 假設公有金鑰名為 id_rsa.pub，並存放在預設位置 (macOS 和 Linux 為 ~/.ssh/，Windows 為 C:\Users*<user-name>*\.ssh)。如果您給公有金鑰不同的名稱，或將其存放在不同的位置中，則必須在 ssh-copy-id 中使用 -i 選項，以指定 SSH 公有金鑰的完整路徑 (例如，ssh-copy-id -i ~/my/path/myKey.pub)。如需有關建立 SSH 金鑰和複製公有金鑰的詳細資訊，請參閱[SSH-COPY-ID](#)。

使用 PuTTYgen 建立 SSH 金鑰 (僅限 Windows)

1. 確定您的待測裝置上已安裝 OpenSSH 伺服器 and 用戶端。如需詳細資訊，請參閱[OpenSSH](#)。
2. 在您的待測裝置上安裝 [PuTTYgen](#)。
3. 開啟 PuTTYgen。
4. 選擇 Generate (產生)，並將滑鼠游標移到方塊內以產生私有金鑰。
5. 從 Conversions (轉換) 功能表中，選擇 Export OpenSSH key (匯出 OpenSSH 金鑰)，然後以 .pem 副檔名儲存私有金鑰。
6. 將公有金鑰新增至待測裝置上的 /home/*<user>*/.ssh/authorized_keys 檔案。

- a. 從 PuTTYgen 視窗複製公有金鑰文字。
 - b. 使用 PuTTY 在您的待測裝置上建立工作階段。
 - i. 從命令提示字元或 Windows Powershell 視窗中，執行下列命令：

```
C:/<path-to-putty>/putty.exe -ssh <user>@<dut-ip-address>
```
 - ii. 出現提示時，請輸入您裝置的密碼。
 - iii. 使用 vi 或其他文字編輯器，將公有金鑰附加到待測裝置上的 `/home/<user>/.ssh/authorized_keys` 檔案。
7. 使用您的使用者名稱、IP 地址，以及私有金鑰檔案的路徑 (您剛針對待測裝置將該檔案儲存在主機電腦上) 來更新 `device.json` 檔案。如需詳細資訊，請參閱[the section called “設定 device.json”](#)。請務必提供私有金鑰的完整路徑和檔案名稱，並使用正斜線 (/)。例如，若為 Windows 路徑 `C:\DT\privatekey.pem`，請在 `device.json` 檔案中使用 `C:/DT/privatekey.pem`。

在您的裝置上設定使用者許可

IDT 會在待測裝置的各種目錄和檔案上執行操作。其中某些操作需要較高的許可 (使用 `sudo`)。若要自動化這些操作，IDT for AWS IoT Greengrass 必須能夠使用 `sudo` 執行命令，而不會提示您輸入密碼。

在待測裝置上依照以下步驟，在不提示輸入密碼的情況下允許 `sudo` 存取。

Note

`username` 是指 IDT 存取待測裝置時所使用的 SSH 使用者。

將使用者新增至 `sudo` 群組

1. 在待測裝置上，執行 `sudo usermod -aG sudo <username>`。
2. 登出後再重新登入，以使變更生效。
3. 若要驗證是否已成功新增您的使用者名稱，請執行 `sudo echo test`。如果未提示您輸入密碼，表示已正確設定您的使用者。
4. 開啟 `/etc/sudoers` 檔案，然後在檔案結尾處新增以下一行：

```
<ssh-username> ALL=(ALL) NOPASSWD: ALL
```

設定您的裝置以測試選用功能

下列主題說明如何設定您的裝置以針對選用功能執行 IDT 測試。只有在您想要測試這些功能時，才需遵循這些設定步驟。否則，請繼續進行[the section called “設定 IDT 設定”](#)。

主題

- [選用：設定 IDT for 的 Docker 容器 AWS IoT Greengrass](#)
- [選用：設定您的裝置以取得 ML 資格](#)

選用：設定 IDT for 的 Docker 容器 AWS IoT Greengrass

AWS IoT Greengrass 提供 Docker 映像和 Dockerfile，可讓您更輕鬆地在 Docker 容器中執行 AWS IoT Greengrass Core 軟體。設定 AWS IoT Greengrass 容器之後，您可以執行 IDT 測試。目前，僅支援 x86_64 Docker 架構來執行 IDT for AWS IoT Greengrass。

此功能需要 IDT v2.3.0 或更新版本。

設定 Docker 容器以執行 IDT 測試的程序取決於您使用的是提供的 Docker 映像或 Dockerfile AWS IoT Greengrass。

- [使用 Docker 映像](#)。Docker 映像檔已安裝 AWS IoT Greengrass 核心軟體和相依性。
- [使用 Dockerfile](#)。Dockerfile 包含可用來建置自訂 AWS IoT Greengrass 容器映像的原始碼。您可以修改映像，以在不同平台架構上執行或減少映像大小。

Note

AWS IoT Greengrass 不提供 AWS IoT Greengrass 核心軟體版本 1.11.1 的 Dockerfile 或 Docker 映像。若要在您自己的自訂容器映像上執行 IDT 測試，映像必須包含提供的 Dockerfile 中定義的相依性 AWS IoT Greengrass。

當您 AWS IoT Greengrass 在 Docker 容器中執行時，無法使用下列功能：

- 在 Greengrass 容器模式下執行[的連接器](#)。若要在 Docker 容器中執行連接器，連接器必須以 No container (無容器) 模式執行。若要尋找支援 No container (無容器) 模式的連接器，請參閱 [the section called “AWS提供的 Greengrass 連接器”](#)。其中一些連接器具有隔離模式參數，您必須設定為 No container (無容器)。

- [本機裝置和磁碟區資源](#)。在 Docker 容器中執行的使用者定義 Lambda 函數必須直接存取核心上的裝置和磁碟區。

設定 提供的 Docker 映像 AWS IoT Greengrass

請依照下列步驟設定 AWS IoT Greengrass Docker 映像以執行 IDT 測試。

先決條件

開始本教學課程之前，您必須執行下列動作。

- 您必須根據您選擇的 AWS Command Line Interface (AWS CLI) 版本，在主機電腦上安裝下列軟體和版本。

AWS CLI version 2

- [Docker](#) 18.09 版或更新版本。舊版可能也適用，但我們建議使用 18.09 或更新版本。
- AWS CLI 2.0.0 版或更新版本。
 - 若要安裝第 2 AWS CLI 版，請參閱[安裝第 2 AWS CLI 版](#)。
 - 若要設定 AWS CLI，請參閱[設定 AWS CLI](#)。

 Note

若要在 Windows 電腦上升級至更新 AWS CLI 版本 2，您必須重複 [MSI 安裝](#) 程序。

AWS CLI version 1

- [Docker](#) 18.09 版或更新版本。舊版可能也適用，但我們建議使用 18.09 或更新版本。
- [Python](#) 3.6 版或更新版本。
- [pip](#) 版本 18.1 或更新版本。
- AWS CLI 版本 - 第 1.17.10 版或更新版本
 - 若要安裝第 1 AWS CLI 版，請參閱[安裝第 1 AWS CLI 版](#)。
 - 若要設定 AWS CLI，請參閱[設定 AWS CLI](#)。
 - 若要升級至最新版本的 第 1 AWS CLI 版，請執行下列命令。

```
pip install awscli --upgrade --user
```


Note

如果您在 Windows 上使用 AWS CLI 版本 1 的 [MSI 安裝](#)，請注意下列事項：

- 如果第 1 AWS CLI 版安裝無法安裝 botocore，請嘗試使用 [Python 和 pip 安裝](#)。
- 若要升級到更新 AWS CLI 版本 1，您必須重複 MSI 安裝程序。

- 若要存取 Amazon Elastic Container Registry (Amazon ECR) 資源，您必須授予下列許可。
- Amazon ECR 要求使用者透過 AWS Identity and Access Management (IAM) 政策授予 `ecr:GetAuthorizationToken` 許可，才能向登錄檔進行身分驗證，並從 Amazon ECR 儲存庫推送或提取映像。如需詳細資訊，請參閱《[Amazon Elastic Container Registry 使用者指南](#)》中的 [Amazon ECR 儲存庫政策範例](#)和[存取一個 Amazon ECR 儲存庫](#)。

1. 下載 Docker 映像並設定容器。您可以從 [Docker Hub](#) 或 [Amazon Elastic Container Registry](#) (Amazon ECR) 下載預先建置的映像，並在 Windows、macOS 和 Linux (x86_64) 平台上執行。

若要從 Amazon ECR 下載 Docker 映像，請完成 中的所有步驟 [the section called “從 Amazon ECR 取得 AWS IoT Greengrass 容器映像”](#)。然後，返回此主題以繼續進行設定。

2. 僅限 Linux 使用者：確定執行 IDT 的使用者具備執行 Docker 命令的許可。如需詳細資訊，請參閱 Docker 文件中的 [以非根使用者身分管理 Docker](#)。
3. 若要執行 AWS IoT Greengrass 容器，請針對您的作業系統使用 命令：

Linux

```
docker run --rm --init -it -d --name aws-iot-greengrass \
-p 8883:8883 \
-v <host-path-to-kernel-config-file>:<container-path> \
<image-repository>:<tag>
```

- 將 `<host-path-to-kernel-config-file>` 替換為主機上核心組態檔案的路徑，並將 `<container-path>` 替換為容器中裝載磁碟區的路徑。

主機上的核心組態圖通常位於 `/proc/config.gz` 或 `/boot/config-<kernel-release-date>`。您可以執行 `uname -r` 以尋找 `<kernel-release-date>` 值。

範例：若要從 `/boot/config-<kernel-release-date>` 裝載組態檔


```
-v /boot/config-4.15.0-74-generic:/boot/config-4.15.0-74-generic \
```

範例：若要從 `proc/config.gz` 裝載組態檔

```
-v /proc/config.gz:/proc/config.gz \
```

- 將命令中的 `<image-repository>:<tag>` 替換為儲存庫的名稱和目標映像的標籤。

範例：指向最新版的 AWS IoT Greengrass Core 軟體

```
216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

若要取得 AWS IoT Greengrass Docker 映像的清單，請執行下列命令。

```
aws ecr list-images --region us-west-2 --registry-id 216483018798 --  
repository-name aws-iot-greengrass
```

macOS

```
docker run --rm --init -it -d --name aws-iot-greengrass \  
-p 8883:8883 \  
<image-repository>:<tag>
```

- 將命令中的 `<image-repository>:<tag>` 替換為儲存庫的名稱和目標映像的標籤。

範例：指向最新版的 AWS IoT Greengrass Core 軟體

```
216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

若要取得 AWS IoT Greengrass Docker 映像的清單，請執行下列命令：

```
aws ecr list-images --region us-west-2 --registry-id 216483018798 --  
repository-name aws-iot-greengrass
```

Windows

```
docker run --rm --init -it -d --name aws-iot-greengrass \  
-p 8883:8883 \  
<image-repository>:<tag>
```

- 將命令中的 `<image-repository>:<tag>` 替換為儲存庫的名稱和目標映像的標籤。

範例：指向最新版的 AWS IoT Greengrass Core 軟體

```
216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

若要取得 AWS IoT Greengrass Docker 映像的清單，請執行下列命令：

```
aws ecr list-images --region us-west-2 --registry-id 216483018798 --  
repository-name aws-iot-greengrass
```

Important

使用 IDT 進行測試時，請勿包含用來執行映像以供一般 AWS IoT Greengrass 使用的 `--entrypoint /greengrass-entrypoint.sh` 引數。

- 下一步：[設定您的 AWS 登入資料和device.json檔案](#)。

設定 提供的 dockerfile AWS IoT Greengrass

請依照下列步驟設定從 Dockerfile 建置的 AWS IoT Greengrass Docker 映像檔，以執行 IDT 測試。

- 從 [the section called “AWS IoT Greengrass Docker 軟體”](#) 中，將 Dockerfile 套件下載到您的主機電腦，並將其解壓縮。
- 打開 README.md。接下來的三個步驟，請參閱此檔案中的章節。
- 請確定您符合先決條件一節中的需求。
- 僅限 Linux 使用者：完成啟用 Symlink 和 Hardlink 保護和啟用 IPv4 網路轉送步驟。
- 若要建置 Docker 映像，請完成步驟 1 中的所有步驟。建置 AWS IoT Greengrass Docker 映像。然後，返回此主題以繼續進行設定。

6. 若要執行 AWS IoT Greengrass 容器，請針對您的作業系統使用 命令：

Linux

```
docker run --rm --init -it -d --name aws-iot-greengrass \  
-p 8883:8883 \  
-v <host-path-to-kernel-config-file>:<container-path> \  
<image-repository>:<tag>
```

- 將 *<host-path-to-kernel-config-file>* 替換為主機上核心組態檔案的路徑，並將 *<container-path>* 替換為容器中裝載磁碟區的路徑。

主機上的核心組態圖通常位於 `/proc/config.gz` 或 `/boot/config-<kernel-release-date>`。您可以執行 `uname -r` 以尋找 *<kernel-release-date>* 值。

範例：若要從 `/boot/config-<kernel-release-date>` 裝載組態檔

```
-v /boot/config-4.15.0-74-generic:/boot/config-4.15.0-74-generic \
```

範例：若要從 `proc/config.gz` 裝載組態檔

```
-v /proc/config.gz:/proc/config.gz \
```

- 將命令中的 *<image-repository>:<tag>* 替換為儲存庫的名稱和目標映像的標籤。

範例：指向最新版的 AWS IoT Greengrass Core 軟體

```
216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

若要取得 AWS IoT Greengrass Docker 映像的清單，請執行下列命令。

```
aws ecr list-images --region us-west-2 --registry-id 216483018798 --  
repository-name aws-iot-greengrass
```

macOS

```
docker run --rm --init -it -d --name aws-iot-greengrass \  
-p 8883:8883 \
```

```
<image-repository>:<tag>
```

- 將命令中的 `<image-repository>:<tag>` 替換為儲存庫的名稱和目標映像的標籤。

範例：指向最新版的 AWS IoT Greengrass Core 軟體

```
216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

若要取得 AWS IoT Greengrass Docker 映像的清單，請執行下列命令：

```
aws ecr list-images --region us-west-2 --registry-id 216483018798 --  
repository-name aws-iot-greengrass
```

Windows

```
docker run --rm --init -it -d --name aws-iot-greengrass \  
-p 8883:8883 \  
<image-repository>:<tag>
```

- 將命令中的 `<image-repository>:<tag>` 替換為儲存庫的名稱和目標映像的標籤。

範例：指向最新版的 AWS IoT Greengrass Core 軟體

```
216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

若要取得 AWS IoT Greengrass Docker 映像的清單，請執行下列命令：

```
aws ecr list-images --region us-west-2 --registry-id 216483018798 --  
repository-name aws-iot-greengrass
```

Important

使用 IDT 進行測試時，請勿包含用來執行映像以供一般 AWS IoT Greengrass 使用的 `--entrypoint /greengrass-entrypoint.sh` 引數。

7. 下一步：[設定您的 AWS 登入資料和device.json檔案。](#)

對 IDT for 的 Docker 容器設定進行故障診斷 AWS IoT Greengrass

使用下列資訊來協助疑難排解執行 Docker 容器進行 IDT AWS IoT Greengrass 測試的問題。

警告：載入 config file:/home/user/.docker/config.json - stat /home/<user>/.docker/config.json 時發生錯誤：權限遭拒

如果在 Linux 上執行 docker 命令時發生錯誤，請執行下列命令。使用執行 IDT 的使用者取代下列命令中的 **<user>**。

```
sudo chown <user>:<user> /home/<user>/.docker -R
sudo chmod g+rx /home/<user>/.docker -R
```

選用：設定您的裝置以取得 ML 資格

IDT for AWS IoT Greengrass 提供機器學習 (ML) 資格測試，以驗證您的裝置是否可以使用雲端訓練模型在本機執行 ML 推論。

若要執行 ML 資格測試，您必須先按照[the section called “設定您的裝置以執行 IDT 測試”](#)中所述設定裝置。然後，依照本主題中的步驟，安裝您想要執行之 ML 框架的相依性。

需要 IDT v3.1.0 或更高版本才能執行 ML 資格的測試。

安裝 ML 框架相依性

所有 ML 框架相依性必須安裝在 /usr/local/lib/python3.x/site-packages 目錄下。為了確保它們安裝在正確的目錄下，我們建議您在安裝相依性時使用 sudo root 許可。資格測試不支援虛擬環境。

Note

如果您要測試使用[容器化](#)（在 Greengrass 容器模式下）執行的 Lambda 函數，/usr/local/lib/python3.x則不支援在下建立 Python 程式庫的符號連結。若要避免錯誤，您必須在正確的目錄下安裝相依性。

按照下列步驟，安裝目標框架的相依性：

- [安裝 MXNet 相依性](#)
- [the section called “安裝 TensorFlow 相依性”](#)
- [安裝 DLR 相依性](#)

安裝 Apache MXNet 相依性

此框架的 IDT 資格測試具有以下相依性：

- Python 3.6 或 Python 3.7。

Note

如果您使用的是 Python 3.6，則必須建立從 Python 3.7 到 Python 3.6 二進位檔的符號連結。這會設定您的裝置以符合 AWS IoT Greengrass 的 Python 需求。例如：

```
sudo ln -s path-to-python-3.6/python3.6 path-to-python-3.7/python3.7
```

- Apache MXNet v1.2.1 或更高版本。
- NumPy. 該版本必須與您的 MxNet 版本相容。

安裝 MXNet

請遵循 MxNet 文件中的指示來[安裝 MxNet](#)。

Note

如果您的裝置上同時安裝了 Python 2.x 和 Python 3.x，則請在您執行的安裝相依性命令中使用 Python 3.x。

驗證 MxNet 安裝

選擇下列其中一個選項來驗證 MXNet 安裝。

選項 1：使用 SSH 連接到您的裝置並執行指令碼

1. 使用 SSH 連接到您的裝置。
2. 執行下列指令碼，確認相依性已正確安裝。

```
sudo python3.7 -c "import mxnet; print(mxnet.__version__)"
```

```
sudo python3.7 -c "import numpy; print(numpy.__version__)"
```

輸出會列印版本號碼，指令碼應該會沒有錯誤地退出。

選項 2：執行 IDT 相依性測試

1. 確定已針對 ML 資格設定 device.json。如需詳細資訊，請參閱[the section called “設定 device.json 以取得 ML 資格”](#)。
2. 執行框架的相依性測試。

```
devicetester_[linux | mac | win_x86-64] run-suite --group-id mldependencies --test-id mxnet_dependency_check
```

測試摘要會顯示 mldependencies 的 PASSED 結果。

安裝 TensorFlow 相依性

此框架的 IDT 資格測試具有以下相依性：

- Python 3.6 或 Python 3.7。

Note

如果您使用的是 Python 3.6，則必須建立從 Python 3.7 到 Python 3.6 二進位檔的符號連結。這會設定您的裝置以符合 AWS IoT Greengrass 的 Python 需求。例如：

```
sudo ln -s path-to-python-3.6/python3.6 path-to-python-3.7/python3.7
```

- TensorFlow 1.x.

安裝 TensorFlow

按照 TensorFlow 文件中的說明，[使用 pip](#) 或 [從來源](#) 安裝 TensorFlow 1.x。

 Note

如果您的裝置上同時安裝了 Python 2.x 和 Python 3.x，則請在您執行的安裝相依性命令中使用 Python 3.x。

驗證 TensorFlow 安裝

選擇下列其中一個選項來驗證 TensorFlow 安裝。

選項 1：使用 SSH 連接到您的裝置並執行指令碼

1. 使用 SSH 連接到您的裝置。
2. 執行下列指令碼，確認相依性已正確安裝。

```
sudo python3.7 -c "import tensorflow; print(tensorflow.__version__)"
```

輸出會列印版本號碼，指令碼應該會沒有錯誤地退出。

選項 2：執行 IDT 相依性測試

1. 確定已針對 ML 資格設定 device.json。如需詳細資訊，請參閱[the section called “設定 device.json 以取得 ML 資格”](#)。
2. 執行框架的相依性測試。

```
devicetester_[linux | mac | win_x86-64] run-suite --group-id mldependencies --test-id tensorflow_dependency_check
```

測試摘要會顯示 mldependencies 的 PASSED 結果。

安裝 Amazon SageMaker AI Neo Deep Learning Runtime (DLR) 相依性

此框架的 IDT 資格測試具有以下相依性：

- Python 3.6 或 Python 3.7。

Note

如果您使用的是 Python 3.6，則必須建立從 Python 3.7 到 Python 3.6 二進位檔的符號連結。這會設定您的裝置以符合 AWS IoT Greengrass 的 Python 需求。例如：

```
sudo ln -s path-to-python-3.6/python3.6 path-to-python-3.7/python3.7
```

- SageMaker AI Neo DLR。
- numpy。

安裝 DLR 測試相依性之後，您必須[編譯模型](#)。

安裝 DLR

請依照 DLR 文件中的指示[安裝 Neo DLR](#)。

Note

如果您的裝置上同時安裝了 Python 2.x 和 Python 3.x，則請在您執行的安裝相依性命令中使用 Python 3.x。

驗證 DLR 安裝

選擇下列其中一個選項來驗證 DLR 安裝。

選項 1：使用 SSH 連接到您的裝置並執行指令碼

1. 使用 SSH 連接到您的裝置。
2. 執行下列指令碼，確認相依性已正確安裝。

```
sudo python3.7 -c "import dlr; print(dlr.__version__)"
```

```
sudo python3.7 -c "import numpy; print(numpy.__version__)"
```

輸出會列印版本號碼，指令碼應該會沒有錯誤地退出。

選項 2：執行 IDT 相依性測試

1. 確定已針對 ML 資格設定 device.json。如需詳細資訊，請參閱[the section called “設定 device.json 以取得 ML 資格”](#)。
2. 執行框架的相依性測試。

```
devicetester_[linux | mac | win_x86-64] run-suite --group-id mldependencies --test-id dlr_dependency_check
```

測試摘要會顯示 mldependencies 的 PASSED 結果。

編譯 DLR 模型

您必須先編譯 DLR 模型，才能將其用於 ML 資格測試。針對步驟，選擇以下其中一個選項：

選項 1：使用 Amazon SageMaker AI 編譯模型

請依照下列步驟使用 SageMaker AI 編譯 IDT 提供的 ML 模型。此模式已使用 Apache MXNET 預先訓練。

1. 確認 SageMaker AI 支援您的裝置類型。如需詳細資訊，請參閱 Amazon SageMaker AI API 參考中的[目標裝置選項](#)。如果 SageMaker AI 目前不支援您的裝置類型，請遵循中的步驟[the section called “選項 2：使用 TVM 編譯 DLR 模型”](#)。

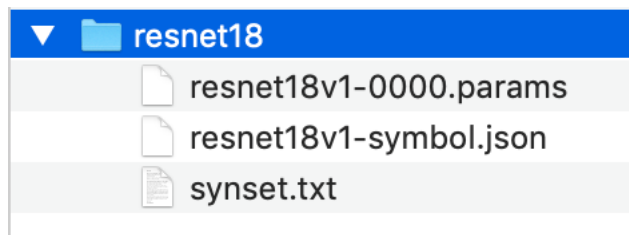
Note

使用 SageMaker AI 編譯的模型執行 DLR 測試可能需要 4 或 5 分鐘。在這段時間內不要停止 IDT。

2. 下載包含適用於 DLR 之未編譯、預先訓練 MxNet 模型的 tarball 檔案：

- [dlr-noncompiled-model-1.0.tar.gz](https://d1r-noncompiled-model-1.0.tar.gz)

3. 解壓縮 tarball。此命令會產生以下目錄結構。



4. 將 `synset.txt` 移出 `resnet18` 目錄。記下新位置。您稍後會將此檔案複製到編譯的模型目錄中。
5. 壓縮 `resnet18` 目錄的內容。

```
tar cvfz model.tar.gz resnet18v1-symbol.json resnet18v1-0000.params
```

6. 將壓縮檔案上傳到中的 Amazon S3 儲存貯體 AWS 帳戶，然後遵循[編譯模型（主控台）](#)中的步驟來建立編譯任務。
 - a. 對於 Input configuration (輸入組態)，請使用下列值：
 - 對於 Data input configuration (資料輸入組態)，請輸入 `{"data": [1, 3, 224, 224]}`。
 - 對於 Machine learning framework (機器學習框架)，請選擇 MXNet。
 - b. 對於 Output configuration (輸出組態)，請使用下列值：
 - 針對 S3 輸出位置，輸入您要存放已編譯模型之 Amazon S3 儲存貯體或資料夾的路徑。
 - 對於 Target device (目標裝置)，選擇您的裝置類型。
7. 從您指定的輸出位置下載已編譯的模型，然後解壓縮檔案。
8. 將 `synset.txt` 複製到已編譯的模型目錄中。
9. 將已編譯模型目錄的名稱變更為 `resnet18`。

您編譯的模型目錄必須具有以下目錄結構。



選項 2：使用 TVM 編譯 DLR 模型

請依照下列步驟，使用 TVM 編譯 IDT 所提供的 ML 模型。此模型已使用 Apache MxNet 進行預先訓練，因此您必須在編譯模型的電腦或裝置上安裝 MxNet。若要安裝 MxNet，請依照 [MxNet 文件](#) 中的指示進行。

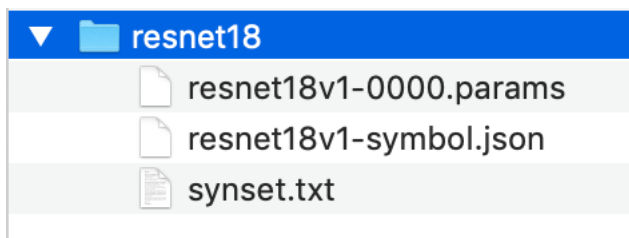
Note

我們建議您在目標裝置上編譯模型。這種做法是選用的，但它可以協助確保相容性並減輕潛在問題。

1. 下載包含適用於 DLR 之未編譯、預先訓練 MxNet 模型的 tarball 檔案：

- dlr-noncompiled-model-1.0.tar.gz

2. 解壓縮 tarball。此命令會產生以下目錄結構。



3. 遵循 TVM 文件中的指示，[從您平台的來源建置和安裝 TVM](#)。
4. 建置 TVM 之後，請針對 resnet18 模型執行 TVM 編譯。以下步驟是以 TVM 文件中的[編譯深度學習模型的快速入門教學](#)為基礎。

- a. 從複製的 TVM 儲存庫中開啟 relay_quick_start.py 檔案。
- b. 更新[在轉送中定義神經網路](#)的程式碼。您可以使用下列其中一個選項：

- 選項 1：使用 mxnet.gluon.model_zoo.vision.get_model 取得轉送模組和參數：

```
from mxnet.gluon.model_zoo.vision import get_model
block = get_model('resnet18_v1', pretrained=True)
mod, params = relay.frontend.from_mxnet(block, {"data": data_shape})
```

- 選項 2：從您在步驟 1 中下載的未編譯模型，將下列檔案複製到與 relay_quick_start.py 檔案相同的目錄中。這些檔案包含轉送模組和參數。

- resnet18v1-symbol.json
- resnet18v1-0000.params

- c. 更新[儲存並載入已編譯模組](#)的程式碼，以使用下列程式碼。

```
from tvn.contrib import util
```

```
path_lib = "deploy_lib.so"
# Export the model library based on your device architecture
lib.export_library("deploy_lib.so", cc="aarch64-linux-gnu-g++")
with open("deploy_graph.json", "w") as fo:
    fo.write(graph)
with open("deploy_param.params", "wb") as fo:
    fo.write(relay.save_param_dict(params))
```

d. 建置模型：

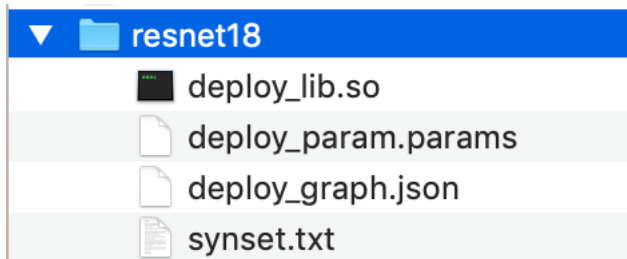
```
python3 tutorials/relay_quick_start.py --build-dir ./model
```

此命令會產生下列檔案。

- deploy_graph.json
- deploy_lib.so
- deploy_param.params

5. 將產生的模型檔案複製到名為 resnet18 的目錄中。這是您編譯的模型目錄。
6. 將編譯後的模型目錄複製到您的主機電腦。然後從您在步驟 1 中下載的未編譯模型，將 synset.txt 複製到已編譯的模型目錄中。

您編譯的模型目錄必須具有以下目錄結構。



接著，[設定您的 AWS 登入資料和 device.json 檔案](#)。

設定 IDT AWS IoT Greengrass 設定以執行資格套件

執行測試之前，您必須在主機電腦上設定 AWS 登入資料和裝置的設定。

設定您的 AWS 登入資料

您必須在 `<device-tester-extract-location> /configs/config.json` 檔案中設定 IAM 使用者登入資料。使用在 中建立之 IDT for AWS IoT Greengrass 使用者的登入資料[the section called “建立和設定 AWS 帳戶”](#)。您可以使用下列兩種方式的其中之一指定登入資料：

- 憑證檔案
- 環境變數

使用 AWS 登入資料檔案設定登入資料

IDT 會使用與 AWS CLI 相同的登入資料檔案。如需詳細資訊，請參閱[組態與登入資料檔案](#)。

登入資料檔案的位置會有所不同，取決於您使用的作業系統：

- macOS, Linux: `~/.aws/credentials`
- Windows : `C:\Users\UserName\.aws\credentials`

以下列格式將您的 AWS 登入資料新增至 `credentials` 檔案：

```
[default]
aws_access_key_id = <your_access_key_id>
aws_secret_access_key = <your_secret_access_key>
```

若要 AWS IoT Greengrass 將 IDT for 設定為使用 `credentials` 檔案中的 AWS 登入資料，請編輯 `config.json` 檔案，如下所示：

```
{
  "awsRegion": "us-west-2",
  "auth": {
    "method": "file",
    "credentials": {
      "profile": "default"
    }
  }
}
```

 Note

如果您不使用 default AWS 設定檔，請務必變更 config.json 檔案中的設定檔名稱。如需詳細資訊，請參閱 [具名描述檔](#)。

使用環境變數設定 AWS 登入資料

環境變數是由作業系統維護且由系統命令使用的變數。如果您關閉 SSH 工作階段，則不會儲存它們。IDT for AWS IoT Greengrass 可以使用 AWS_ACCESS_KEY_ID 和 AWS_SECRET_ACCESS_KEY 環境變數來存放您的 AWS 登入資料。

若要在 Linux、macOS 或 Unix 上設定這些變數，請使用 export：

```
export AWS_ACCESS_KEY_ID=<your_access_key_id>
export AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

若要在 Windows 上設定這些變數，請使用 set：

```
set AWS_ACCESS_KEY_ID=<your_access_key_id>
set AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

若要設定 IDT 來使用環境變數，請在 config.json 檔案中編輯 auth 區段。請見此處範例：

```
{
  "awsRegion": "us-west-2",
  "auth": {
    "method": "environment"
  }
}
```

設定 device.json

除了 AWS 登入資料之外，IDT for AWS IoT Greengrass 還需要測試執行所在裝置的相關資訊（例如 IP 地址、登入資訊、作業系統和 CPU 架構）。

您必須使用位於 `<device_tester_extract_location>/configs/device.json` 中的 device.json 範本提供此資訊：

Physical device

```
[
  {
    "id": "<pool-id>",
    "sku": "<sku>",
    "features": [
      {
        "name": "os",
        "value": "linux | ubuntu | openwrt"
      },
      {
        "name": "arch",
        "value": "x86_64 | armv6l | armv7l | aarch64"
      },
      {
        "name": "container",
        "value": "yes | no"
      },
      {
        "name": "docker",
        "value": "yes | no"
      },
      {
        "name": "streamManagement",
        "value": "yes | no"
      },
      {
        "name": "hsi",
        "value": "yes | no"
      },
      {
        "name": "ml",
        "value": "mxnet | tensorflow | dlr | mxnet,dlr,tensorflow | no"
      },
      ***** Remove the section below if the device is not qualifying for ML
      *****
      {
        "name": "mlLambdaContainerizationMode",
        "value": "container | process | both"
      },
      {
        "name": "processor",
        "value": "cpu | gpu"
      }
    ]
  }
]
```



```

    },

*****

    ],
    ***** Remove the section below if the device is not qualifying for HSI
    *****
    "hsm": {
        "p11Provider": "/path/to/pkcs11ProviderLibrary",
        "slotLabel": "<slot_label>",
        "slotUserPin": "<slot_pin>",
        "privateKeyLabel": "<key_label>",
        "openSSLEngine": "/path/to/openssl/engine"
    },

*****

    ***** Remove the section below if the device is not qualifying for ML
    *****
    "machineLearning": {
        "dlrModelPath": "/path/to/compiled/dlr/model",
        "environmentVariables": [
            {
                "key": "<environment-variable-name>",
                "value": "<Path:$PATH>"
            }
        ],
        "deviceResources": [
            {
                "name": "<resource-name>",
                "path": "<resource-path>",
                "type": "device | volume"
            }
        ]
    },

*****

    "kernelConfigLocation": "",
    "greengrassLocation": "",
    "devices": [
        {
            "id": "<device-id>",
            "connectivity": {
                "protocol": "ssh",
                "ip": "<ip-address>",
                "port": 22,

```

```
    "auth": {
      "method": "pki | password",
      "credentials": {
        "user": "<user-name>",
        "privKeyPath": "/path/to/private/key",
        "password": "<password>"
      }
    }
  }
}
]
```

Note

如果 method 是設定為 pki，則指定 privKeyPath
如果 method 是設定為 password，則指定 password

Docker container

```
[
  {
    "id": "<pool-id>",
    "sku": "<sku>",
    "features": [
      {
        "name": "os",
        "value": "linux | ubuntu | openwrt"
      },
      {
        "name": "arch",
        "value": "x86_64"
      },
      {
        "name": "container",
        "value": "no"
      },
      {
        "name": "docker",
        "value": "no"
      }
    ]
  }
]
```

```

    },
    {
      "name": "streamManagement",
      "value": "yes | no"
    },
    {
      "name": "hsi",
      "value": "no"
    },
    {
      "name": "ml",
      "value": "mxnet | tensorflow | dlr | mxnet,dlr,tensorflow | no"
    },
    ***** Remove the section below if the device is not qualifying for ML
    *****
    {
      "name": "mlLambdaContainerizationMode",
      "value": "process"
    },
    {
      "name": "processor",
      "value": "cpu | gpu"
    },
    },
    *****
  ],
  ***** Remove the section below if the device is not qualifying for ML
  *****
  "machineLearning": {
    "dlrModelPath": "/path/to/compiled/dlr/model",
    "environmentVariables": [
      {
        "key": "<environment-variable-name>",
        "value": "<Path:$PATH>"
      }
    ],
    "deviceResources": [
      {
        "name": "<resource-name>",
        "path": "<resource-path>",
        "type": "device | volume"
      }
    ]
  },

```

```
*****
  "kernelConfigLocation": "",
  "greengrassLocation": "",
  "devices": [
    {
      "id": "<device-id>",
      "connectivity": {
        "protocol": "docker",
        "containerId": "<container-name | container-id>",
        "containerUser": "<user>"
      }
    }
  ]
}
```

如下所述，包含值的所有欄位皆為必要：

id

使用者定義的英數字元 ID，可唯一識別裝置的集合，稱為「裝置集區」。屬於同一個集區的裝置必須有相同的硬體。當您執行測試套件，集區中的裝置會用來將工作負載平行化。多個裝置用來執行不同的測試。

sku

可唯一識別測試裝置的英數字元值。SKU 用來追蹤合格的主機板。

Note

如果您想要在 AWS Partner Device Catalog 中列出您的電路板，您在此處指定的 SKU 必須符合您在列出程序中使用的 SKU。

features

包含裝置支援功能的陣列。所有功能都是必要的。

os 和 arch

支援的作業系統 (OS) 和架構組合：

- linux, x86_64
- linux, armv6l
- linux, armv7l
- linux, aarch64
- ubuntu, x86_64
- openwrt, armv7l
- openwrt, aarch64

 Note

如果您使用 IDT 來測試在 Docker 容器中 AWS IoT Greengrass 執行，則僅支援 x86_64 Docker 架構。

container

驗證裝置是否符合在 Greengrass 核心上以容器模式執行 Lambda 函數的所有軟體和硬體需求。

有效值為 yes 或 no。

docker

驗證裝置是否符合所有必要的技術相依性，以使用 Greengrass Docker 應用程式部署連接器來執行容器

有效值為 yes 或 no。

streamManagement

驗證裝置是否符合執行 AWS IoT Greengrass 串流管理員所需的所有技術相依性。

有效值為 yes 或 no。

hsi

驗證提供的 HSI 共用程式庫是否可以與硬體安全模組 (HSM) 溝通，並正確實作所需的 PKCS#11 API。HSM 和共用程式庫必須能夠簽署 CSR、執行 TLS 操作並提供正確的金鑰長度 and 公開金鑰演算法。

有效值為 yes 或 no。

ml

驗證裝置符合所有必要的技術相依性，以便在本機執行 ML 推論。

有效值可以是 mxnet、tensorflow、dlr 和 的任意組合 no (例如 mxnet、mxnet,tensorflow、mxnet,tensorflow,dlr 或 no)。

mlLambdaContainerizationMode

驗證裝置是否符合在 Greengrass 裝置上以容器模式執行 ML 推論所需的所有技術相依性。

有效值為 container、process 或 both。

processor

驗證裝置是否符合指定處理器類型的所有硬體需求。

有效值為 cpu 或 gpu。

Note

如果您不想使用 container、docker、hsi、streamManager 或 ml 功能，您可以將對應的 設定為 value no。

Docker 僅支援 streamManagement 和 的功能資格 ml。

machineLearning

選用。ML 資格測試的組態資訊。如需詳細資訊，請參閱 [the section called “設定 device.json 以取得 ML 資格”](#)。

hsm

選用。使用 AWS IoT Greengrass 硬體安全模組 (HSM) 進行測試的組態資訊。否則，應省略 hsm 屬性。如需詳細資訊，請參閱 [硬體安全整合](#)。

只有當 connectivity.protocol 設為 ssh 時，才會套用此屬性。

hsm.p11Provider

PKCS#11 實作的 libdl-loadable 程式庫的絕對路徑。

hsm.slotLabel

用於識別硬體模組的插槽標籤。

`hsm.slotUserPin`

用來向模組驗證 AWS IoT Greengrass 核心的使用者 PIN 碼。

`hsm.privateKeyLabel`

用於識別硬體模組中之金鑰的標籤。

`hsm.openSSLEngine`

OpenSSL 引擎 .so 檔案的絕對路徑，此檔案可在 OpenSSL 上啟用 PKCS#11 支援。由 OTA AWS IoT Greengrass 更新代理程式使用。

`devices.id`

使用者定義的唯一識別符，用於識別要測試的裝置。

`connectivity.protocol`

用來與此裝置通訊的通訊協定。目前，唯一支援的值是實體裝置的 `ssh`，以及 Docker 容器的 `docker`。

`connectivity.ip`

要測試之裝置的 IP 位址。

只有當 `connectivity.protocol` 設為 `ssh` 時，才會套用此屬性。

`connectivity.containerId`

要測試之 Docker 容器的容器 ID 或名稱。

只有當 `connectivity.protocol` 設為 `docker` 時，才會套用此屬性。

`connectivity.auth`

連線的驗證資訊。

只有當 `connectivity.protocol` 設為 `ssh` 時，才會套用此屬性。

`connectivity.auth.method`

用來透過指定的連線通訊協定存取裝置的驗證方法。

支援的值如下：

- `pki`
- `password`

`connectivity.auth.credentials`

用於驗證的登入資料。

`connectivity.auth.credentials.password`

用於登入要測試裝置的密碼。

只有當 `connectivity.auth.method` 設為 `password` 時，才會套用此值。

`connectivity.auth.credentials.privKeyPath`

用來登入待測裝置之私有金鑰的完整路徑。

只有當 `connectivity.auth.method` 設為 `pki` 時，才會套用此值。

`connectivity.auth.credentials.user`

登入要測試之裝置的使用者名稱。

`connectivity.auth.credentials.privKeyPath`

用來登入要測試之裝置的私有金鑰的完整路徑。

`connectivity.port`

選用。用於 SSH 連線的連接埠號碼。

預設值為 22。

此屬性僅適用於 `connectivity.protocol` 設為 `ssh` 的情況。

`greengrassLocation`

您裝置上 AWS IoT Greengrass Core 軟體的位置。

對於實體裝置，只有在您使用的現有安裝時，才會使用此值 AWS IoT Greengrass。使用此屬性指示 IDT 使用安裝在您裝置上的 AWS IoT Greengrass Core 軟體版本。

從 Docker 映像或提供的 Dockerfile 在 Docker 容器中執行測試時 AWS IoT Greengrass，請將此值設定為 `/greengrass`。

`kernelConfigLocation`

選用。核心組態檔案的路徑。AWS IoT Device Tester 使用此檔案來檢查裝置是否已啟用必要的核心功能。如果未指定，IDT 會使用下列路徑來搜尋核心組態檔案：`/proc/config.gz` 和 `/boot/config-<kernel-version>`。AWS IoT Device Tester 會使用其找到的第一個路徑。

設定 device.json 以取得 ML 資格

本節說明裝置組態檔案中適用於 ML 資格的選用屬性。如果您計劃針對 ML 資格執行測試，則必須定義適用於您的使用案例的屬性。

您可以使用 device-ml.json 範本來定義裝置的組態設定。此範本包含選用的 ML 屬性。您也可以使用 device.json 並新增 ML 資格屬性。這些檔案位於 `<device-tester-extract-location>/configs` 中，並包含 ML 資格屬性。如果您使用 device-ml.json，您必須在執行 IDT 測試之前將檔案重新命名為 device.json。

如需不適用於 ML 資格之裝置組態屬性的相關資訊，請參閱 [the section called “設定 device.json”](#)。

features 陣列中的 ml

您的主機板支援的 ML 框架。此屬性需要 IDT v3.1.0 或更新版本。

- 如果您的主機板只支援一個框架，請指定該框架。例如：

```
{
  "name": "ml",
  "value": "mxnet"
}
```

- 如果您的主機板支援多個框架，請以逗號分隔的清單指定框架。例如：

```
{
  "name": "ml",
  "value": "mxnet,tensorflow"
}
```

features 陣列中的 mlLambdaContainerizationMode

您想要測試的[容器化模式](#)。此屬性需要 IDT v3.1.0 或更新版本。

- 選擇使用非容器化 Lambda 函數 process 執行 ML 推論程式碼。此選項需要 v1 AWS IoT Greengrass .10.x 或更新版本。
- 選擇使用容器化 Lambda 函數 container 執行 ML 推論程式碼。
- 選擇 both 以兩種模式執行 ML 推論程式碼。此選項需要 v1 AWS IoT Greengrass .10.x 或更新版本。

features 陣列中的 processor

指出主機板支援的硬體加速器。此屬性需要 IDT v3.1.0 或更新版本。

- 如果您的主機板使用 CPU 做為處理器，請選擇 `cpu`。
- 如果您的主機板使用 GPU 做為處理器，請選擇 `gpu`。

machineLearning

選用。ML 資格測試的組態資訊。此屬性需要 IDT v3.1.0 或更新版本。

dlrModelPath

使用 dlr 框架時需要。DLR 編譯模型目錄的絕對路徑，必須命名為 `resnet18`。如需詳細資訊，請參閱[the section called “編譯 DLR 模型”](#)。

Note

以下是 macOS 上的範例路徑：`/Users/<user>/Downloads/resnet18`。

environmentVariables

鍵值組的陣列，可以動態地將設定傳遞給 ML 推論測試。對於 CPU 裝置為選用。您可以使用此區段來新增裝置類型所需的框架特定環境變數。如需這些需求的相關資訊，請參閱框架或裝置的官方網站。例如，若要在某些裝置上執行 MxNet 推論測試，可能需要下列環境變數。

```
"environmentVariables": [  
  ...  
  {  
    "key": "PYTHONPATH",  
    "value": "$MXNET_HOME/python:$PYTHONPATH"  
  },  
  {  
    "key": "MXNET_HOME",  
    "value": "$HOME/mxnet/"  
  },  
  ...  
]
```

Note

`value` 欄位可能會根據您的 MxNet 安裝而有所不同。

如果您要測試在 GPU 裝置上使用[容器化](#)執行的 Lambda 函數，請新增 GPU 程式庫的環境變數。這使得 GPU 可以執行計算。若要使用不同的 GPU 程式庫，請參閱程式庫或裝置的官方文件。

 Note

如果 `m1LambdaContainerizationMode` 功能設定為 `container` 或 `both`，請設定下列鍵。

```
"environmentVariables": [  
  {  
    "key": "PATH",  
    "value": "<path/to/software/bin>:$PATH"  
  },  
  {  
    "key": "LD_LIBRARY_PATH",  
    "value": "<path/to/ld/lib>"  
  },  
  ...  
]
```

deviceResources

GPU 裝置所需。包含可由 Lambda 函數存取的[本機資源](#)。使用此區段可新增本機裝置和磁碟區資源。

- 對於裝置資源，請指定 `"type": "device"`。對於 GPU 裝置，裝置資源應該是 `/dev` 下的 GPU 相關裝置檔案。

 Note

`/dev/shm` 目錄是例外狀況。它僅能設為磁碟區資源。

- 對於磁碟區資源，請指定 `"type": "volume"`。

執行 AWS IoT Greengrass 資格套件

在您[設定必要組態](#)之後，便可開始測試。完整測試套件的執行時間取決於您的硬體。做為參考，在 Raspberry Pi 3B 上完成整個測試套件約需 30 分鐘。

下列範例 `run-suite` 命令示範如何執行裝置集區的資格測試。裝置集區是一組相同的裝置。

IDT v3.0.0 and later

執行指定的測試套件中的所有測試群組。

```
devicetester_[linux | mac | win_x86-64] run-suite --suite-id GGQ_1.0.0 --pool-id <pool-id>
```

使用 `list-suites` 命令列出 `tests` 資料夾中的測試套件。

執行測試套件中的特定測試群組。

```
devicetester_[linux | mac | win_x86-64] run-suite --suite-id GGQ_1.0.0 --group-id <group-id> --pool-id <pool-id>
```

使用 `list-groups` 命令列出測試套件中的測試群組。

執行測試群組中的特定測試案例。

```
devicetester_[linux | mac | win_x86-64] run-suite --group-id <group-id> --test-id <test-id>
```

執行測試群組中的多個測試案例。

```
devicetester_[linux | mac | win_x86-64] run-suite --group-id <group-id> --test-id <test-id1>,<test-id2>
```

列出測試群組中的測試案例。

```
devicetester_[linux | mac | win_x86-64] list-test-cases --group-id <group-id>
```

`run-suite` 命令的選項是選擇性的。例如，如果 `device.json` 檔案中只定義一個裝置集區，則可以省略 `pool-id`。或者，如果您想要執行 `tests` 資料夾中最新的測試套件版本，則可以省略 `suite-id`。

Note

如果線上有較新的測試套件版本，IDT 會提示您。如需詳細資訊，請參閱[the section called “設定預設更新行為”](#)。

如需 `run-suite` 和其他 IDT 命令的詳細資訊，請參閱 [the section called “IDT 命令”](#)。

IDT v2.3.0 and earlier

執行指定套件中的所有測試群組。

```
devicetester_[linux | mac | win_x86-64] run-suite --suite-id GGQ_1 --pool-id <pool-id>
```

執行特定的測試群組。

```
devicetester_[linux | mac | win_x86-64] run-suite --suite-id GGQ_1 --group-id <group-id> --pool-id <pool-id>
```

如果您在單一裝置集區上執行單一測試套件，則 `suite-id` 和 `pool-id` 是選擇性的。這表示您的 `device.json` 檔案中只定義了一個裝置集區。

檢查 Greengrass 相依性

我們建議您在執行相關測試群組之前，先執行相依性檢查程式測試群組，以確定所有 Greengrass 相依性都已安裝。例如：

- 在執行核心資格測試組之前執行 `ggcdependencies`。
- 在執行容器特定測試群組之前執行 `containerdependencies`。
- 在執行 Docker 特定測試群組之前執行 `dockerdependencies`。
- 在執行串流管理員特定測試群組之前執行 `ggcstreammanagementdependencies`。

設定預設更新行為

當您開始測試回合時，IDT 會在線上檢查是否有較新的測試套件版本。如果有新版本，IDT 會提示您更新至最新的可用版本。您可以設定 `upgrade-test-suite` (或 `u`) 旗標來控制預設的更新行為。有效的值如下：

- `y`。IDT 會下載並使用最新的可用版本。
- `n` (預設)。IDT 使用 `suite-id` 選項中指定的版本。如果 `suite-id` 未指定，IDT 會在 `tests` 資料夾中使用最新版本。

如果您未包含 `upgrade-test-suite` 旗標，IDT 會在有可用的更新時提示您，並等待 30 秒供您輸入 (y 或 n)。如果未輸入項目，它會預設為 n 並繼續執行測試。

下列範例顯示此功能的常用案例：

自動使用測試群組可用的最新測試。

```
devicetester_linux run-suite -u y --group-id mqtt --pool-id DevicePool1
```

在特定測試套件版本中執行測試。

```
devicetester_linux run-suite -u n --suite-id GGQ_1.0.0 --group-id mqtt --pool-id DevicePool1
```

在執行時間提示更新。

```
devicetester_linux run-suite --pool-id DevicePool1
```

IDT for AWS IoT Greengrass 命令

IDT 命令位於 *<device-tester-extract-location>*/bin 目錄中。將它們用於下列操作：

IDT v3.0.0 and later

`help`

列出所指定命令的相關資訊。

`list-groups`

列出指定測試套件中的群組。

`list-suites`

列出可用的測試套件。

`list-supported-products`

列出支援的 產品，在此案例中為 AWS IoT Greengrass IDT 版本，以及測試套件版本。

`list-test-cases`

列出特定測試群組中的測試案例。支援下列選項：

- `group-id`。要搜尋的測試群組。此選項為必要選項，且必須指定單一群組。

`run-suite`

在裝置集區上執行測試套件。以下是一些支援的選項：

- `suite-id`。要執行的測試套件版本。如果未指定，IDT 會使用 `tests` 資料夾中的最新版本。
- `group-id`。要執行的測試群組，以逗號分隔的清單。如果未指定，IDT 會執行測試套件中的所有測試群組。
- `test-id`。要執行的測試案例，以逗號分隔的清單。指定時，`group-id` 必須指定單一群組。
- `pool-id`。要測試的裝置集區。如果 `device.json` 檔案中已定義多個裝置集區，則必須指定集區。
- `upgrade-test-suite`。控制測試套件版本更新的處理方式。從 IDT v3.0.0 開始，IDT 會在線上檢查是否有更新的測試套件版本。如需詳細資訊，請參閱[the section called “測試套件版本”](#)。
- `stop-on-first-failure`。設定 IDT 在第一次失敗時停止執行。此選項應與 `group-id` 搭配使用以偵錯指定的測試群組。執行完整測試套件產生資格報告時，請勿使用此選項。
- `update-idt`。設定提示更新 IDT 的回應。Y 當輸入在 IDT 偵測到有較新的版本時停止測試執行。N 當輸入繼續測試執行時。
- `update-managed-policy`。輸入 Y 會在 IDT 偵測到使用者的受管政策未更新時停止測試執行。輸入 N 則繼續測試執行。

如需 `run-suite` 選項的詳細資訊，請使用下列 `help` 選項：

```
devicetester_[linux | mac | win_x86-64] run-suite -h
```

IDT v2.3.0 and earlier

`help`

列出所指定命令的相關資訊。

`list-groups`

列出指定測試套件中的群組。

list-suites

列出可用的測試套件。

run-suite

在裝置集區上執行測試套件。

如需 run-suite 選項的詳細資訊，請使用下列 help 選項：

```
devicetester_[linux | mac | win_x86-64] run-suite -h
```

了解結果和日誌

本節說明如何檢視和解譯 IDT 結果報告與日誌。

檢視結果

執行期間，IDT 會將錯誤寫入主控台、日誌檔和測試報告。IDT 完成資格測試套件後會產生兩份測試報告。您可以在 *<device-tester-extract-location>/results/<execution-id>/* 中找到這些報告。這兩份報告都會從資格測試套件執行擷取結果。

awsiotdevicetester_report.xml 是您提交 AWS 至 以在 Device AWS Partner Catalog 中列出裝置的資格測試報告。該報告包含下列元素：

- IDT 版本。
- 已測試的 AWS IoT Greengrass 版本。
- device.json 檔案中指定的 SKU 和裝置集區名稱。
- device.json 檔案中指定的裝置集區的功能。
- 測試結果的彙總摘要。
- 依照基於裝置功能 (例如，本機資源存取、影子、MQTT 等) 而測試的程式庫來分類的測試結果。

GGQ_Result.xml 報告採用 [JUnit XML 格式](#)。您可以將它整合到持續整合和部署平台，例如 [Jenkins](#)、[Bamboo](#) 等。該報告包含下列元素：

- 測試結果的彙總摘要。
- 依已測試 AWS IoT Greengrass 的功能分類測試結果。

解譯 IDT 報告

awsiotdevicetester_report.xml 或 awsiotdevicetester_report.xml 的報告區段會列出已執行的測試及結果。

第一個 XML 標籤 <testsuites> 包含測試執行的摘要。例如：

```
<testsuites name="GGQ results" time="2299" tests="28" failures="0" errors="0" disabled="0">
```

<testsuites> 標籤中使用的屬性

name

測試套件的名稱。

time

執行資格套件所花費的時間 (以秒為單位)。

tests

執行的測試次數。

failures

已執行但未通過的測試次數。

errors

IDT 無法執行的測試次數。

disabled

此屬性未使用，可忽略。

awsiotdevicetester_report.xml 檔案包含 <awsproduct> 標籤，其中包含關於受測產品和經過一系列測試驗證後之產品功能的資訊。

<awsproduct> 標籤中使用的屬性

name

受測產品名稱。

version

受測產品版本。

features

驗證的功能。標記為 `required` 的功能為提交主機板獲得資格時所需。以下片段顯示此資訊如何出現在 `awsiotdevicetester_report.xml` 檔案中。

```
<feature name="aws-iot-greengrass-no-container" value="supported" type="required"></feature>
```

標記為 `optional` 的功能不需要進行資格測試。以下程式碼片段顯示選用功能。

```
<feature name="aws-iot-greengrass-container" value="supported" type="optional"></feature>

<feature name="aws-iot-greengrass-hsi" value="not-supported" type="optional"></feature>
```

如果必要功能沒有測試失敗或錯誤，您的裝置符合執行的技術需求，AWS IoT Greengrass 並且可以與 AWS IoT 服務互通。如果您想要在 Device AWS Partner Catalog 中列出您的裝置，您可以使用此報告做為資格證據。

如果測試發生失敗或錯誤，您可以檢閱 `<testsuites>` XML 標籤來識別失敗的測試。`<testsuites>` 標籤內的 `<testsuite>` XML 標籤會顯示測試群組的測試結果摘要。例如：

```
<testsuite name="combination" package="" tests="1" failures="0" time="161" disabled="0" errors="0" skipped="0">
```

其格式類似於 `<testsuites>` 標籤，但有不使用且可忽略的 `skipped` 屬性。在每個 `<testsuite>` XML 標籤內，測試群組每個執行的測試都有 `<testcase>` 標籤。例如：

```
<testcase classname="Security Combination (IPD + DCM) Test Context" name="Security Combination IP Change Tests sec4_test_1: Should rotate server cert when IPD disabled and following changes are made:Add CIS conn info and Add another CIS conn info" attempts="1"></testcase>>
```

<testcase> 標籤中使用的屬性

name

測試的名稱。

attempts

IDT 執行測試案例的次數。

當測試案例失敗或發生錯誤時，系統就會將 <failure> 或 <error> 標籤新增至 <testcase> 標籤，其中附有相關資訊以利故障診斷。例如：

```
<testcase classname="mcu.Full_MQTT" name="AFQP_MQTT_Connect_HappyCase" attempts="1">
  <failure type="Failure">Reason for the test failure</failure>
  <error>Reason for the test execution error</error>
</testcase>
```

檢視日誌

IDT 會在 *<devicetester-extract-location>/results/<execution-id>/logs* 中從測試執行產生日誌。該工具會產生兩組日誌：

test_manager.log

從 AWS IoT Device Tester 的測試管理員元件產生的日誌（例如，與組態、測試排序和報告產生相關的日誌）。

<test_case_id>.log (for example, ota.log)

測試群組的日誌，包括來自待測裝置的日誌。測試失敗時，將會建立 tar.gz 檔案 (例如，ota_prod_test_1_ggc_logs.tar.gz)，內含所建立測試的待測裝置日誌。

如需詳細資訊，請參閱[IDT 故障 AWS IoT Greengrass 診斷](#)。

使用 IDT 來開發和執行您自己的測試套件

從 IDT v4.0.0 開始，IDT for AWS IoT Greengrass 結合了標準化的組態設定和結果格式與測試套件環境，可讓您為裝置和裝置軟體開發自訂測試套件。您可以為自己的內部驗證新增自訂測試，或將其提供給客戶進行裝置驗證。

使用 IDT 來開發和執行自訂測試套件，如下所示：

開發自訂測試套件

- 為您要測試的 Greengrass 裝置建立具有自訂測試邏輯的測試套件。
- 提供 IDT 您的自訂測試套件來測試執行器。包含測試套件特定設定組態的相關資訊。

執行自訂測試套件

- 設定您要測試的裝置。
- 實作您想要使用的測試套件所需的設定組態。
- 使用 IDT 執行您的自訂測試套件。
- 檢視 IDT 所執行測試的測試結果和執行日誌。

下載適用於 的最新版 AWS IoT Device Tester AWS IoT Greengrass

下載[最新版本](#)的 IDT，並將軟體解壓縮到檔案系統上具有讀取和寫入許可的位置。

Note

IDT 不支援由多位使用者從共用位置執行，例如 NFS 目錄或 Windows 網路共用資料夾。我們建議您將 IDT 套件解壓縮到本機磁碟機，並在本機工作站上執行 IDT 二進位檔。

Windows 的路徑長度限制為 260 個字元。如果您使用的是 Windows，請將 IDT 解壓縮到根目錄，例如 C:\ 或 D:\，使路徑保持在 260 個字元的限制以下。

測試套件建立工作流程

測試套件由三種類型的檔案組成：

- 為 IDT 提供如何執行測試套件相關資訊的 JSON 組態檔案。
- 測試 IDT 用來執行測試案例的可執行檔。
- 執行測試所需的其他檔案。

完成下列基本步驟以建立自訂 IDT 測試：

1. 為您的測試套件[建立 JSON 組態檔案](#)。
2. [建立包含測試套件測試邏輯的測試案例可執行檔](#)。
3. 驗證並記錄[測試執行器執行測試套件所需的組態資訊](#)。

4. 確認 IDT 可以執行您的測試套件，並如預期產生[測試結果](#)。

若要快速建置範例自訂套件並進行執行，請遵循 中的指示[教學課程：建置並執行範例 IDT 測試套件](#)。

若要開始在 Python 中建立自訂測試套件，請參閱 [教學課程：開發簡單的 IDT 測試套件](#)。

教學課程：建置並執行範例 IDT 測試套件

AWS IoT Device Tester 下載包含範例測試套件的原始程式碼。您可以完成此教學課程來建置和執行範例測試套件，以了解如何使用 AWS IoT Device Tester for AWS IoT Greengrass 來執行自訂測試套件。

在本教學課程中，您將完成下列步驟：

1. [建置範例測試套件](#)
2. [使用 IDT 執行範例測試套件](#)

先決條件

為了完成本教學，您需要以下項目：

- 主機電腦需求
 - AWS IoT Device Tester 的最新版本
 - [Python](#) 3.7 或更新版本

若要檢查安裝在電腦上的 Python 版本，請執行下列命令：

```
python3 --version
```

在 Windows 上，如果使用此命令傳回錯誤，請將 `python3` 改用 `python`。如果傳回的版本編號為 3.7 或更新版本，請在 Powershell 終端機中執行下列命令，將 `python3` 設定為 `python` 命令的別名。

```
Set-Alias -Name "python3" -Value "python"
```

如果未傳回版本資訊或版本編號小於 3.7，請遵循[下載 Python](#) 中的指示安裝 Python 3.7+。如需詳細資訊，請參閱 [Python 文件](#)。

- [urllib3](#)

若要驗證 `urllib3` 是否正確安裝，請執行下列命令：

```
python3 -c 'import urllib3'
```

如果 `urllib3` 未安裝，請執行下列命令來安裝它：

```
python3 -m pip install urllib3
```

- 裝置要求
- 具有 Linux 作業系統和與主機電腦相同網路連線的裝置。

我們建議您將 [Raspberry Pi](#) 與 Raspberry Pi 作業系統搭配使用。請務必在 Raspberry Pi 上設定 [SSH](#) 以遠端連線到它。

設定 IDT 的裝置資訊

設定您的裝置資訊，讓 IDT 執行測試。您必須使用下列資訊更新位於 `<device-tester-extract-location>/configs` 資料夾的 `device.json` 範本。

```
[
  {
    "id": "pool",
    "sku": "N/A",
    "devices": [
      {
        "id": "<device-id>",
        "connectivity": {
          "protocol": "ssh",
          "ip": "<ip-address>",
          "port": "<port>",
          "auth": {
            "method": "pki | password",
            "credentials": {
              "user": "<user-name>",
              "privKeyPath": "/path/to/private/key",
              "password": "<password>"
            }
          }
        }
      }
    ]
  }
]
```

```
]
}
]
```

在 `devices` 物件中，提供下列資訊：

`id`

您裝置的使用者定義唯一識別符。

`connectivity.ip`

您裝置的 IP 地址。

`connectivity.port`

選用。用於 SSH 連線至您裝置的連接埠號碼。

`connectivity.auth`

連線的驗證資訊。

只有當 `connectivity.protocol` 設為 `ssh` 時，才會套用此屬性。

`connectivity.auth.method`

用來透過指定的連線通訊協定存取裝置的驗證方法。

支援的值如下：

- `pki`
- `password`

`connectivity.auth.credentials`

用於驗證的登入資料。

`connectivity.auth.credentials.user`

用來登入裝置的使用者名稱。

`connectivity.auth.credentials.privKeyPath`

用來登入您裝置之私有金鑰的完整路徑。

只有當 `connectivity.auth.method` 設為 `pki` 時，才會套用此值。

`devices.connectivity.auth.credentials.password`

用於登入您裝置的密碼。

只有當 `connectivity.auth.method` 設為 `password` 時，才會套用此值。

Note

如果 `method` 是設定為 `pki`，則指定 `privKeyPath`

如果 `method` 是設定為 `password`，則指定 `password`

建置範例測試套件

`<device-tester-extract-location>/samples/python` 資料夾包含範例組態檔案、原始程式碼和 IDT 用戶端 SDK，您可以使用提供的建置指令碼將其合併為測試套件。下列目錄樹狀目錄顯示這些範例檔案的位置：

```
<device-tester-extract-location>
### ...
### tests
### samples
#   ### ...
#   ### python
#       ### configuration
#       ### src
#       ### build-scripts
#           ### build.sh
#           ### build.ps1
### sdks
### ...
### python
### idt_client
```

若要建置測試套件，請在主機電腦上執行下列命令：

Windows

```
cd <device-tester-extract-location>/samples/python/build-scripts
./build.ps1
```


Linux, macOS, or UNIX

```
cd <device-tester-extract-location>/samples/python/build-scripts
./build.sh
```

這會在 IDTSampleSuitePython_1.0.0 資料夾內的 *<device-tester-extract-location>/tests* 資料夾中建立範例測試套件。檢閱 IDTSampleSuitePython_1.0.0 資料夾中的檔案，以了解範例測試套件的結構，並查看各種測試案例可執行檔和測試組態 JSON 檔案的範例。

下一步：使用 IDT 執行[您建立的範例測試套件](#)。

使用 IDT 執行範例測試套件

若要執行範例測試套件，請在主機電腦上執行下列命令：

```
cd <device-tester-extract-location>/bin
./devicetester_[linux | mac | win_x86-64] run-suite --suite-id IDTSampleSuitePython
```

IDT 會執行範例測試套件，並將結果串流至主控台。當測試完成執行時，您會看到下列資訊：

```
===== Test Summary =====
Execution Time:           5s
Tests Completed:          4
Tests Passed:             4
Tests Failed:             0
Tests Skipped:            0
-----
Test Groups:
    sample_group:         PASSED
-----
Path to IoT Device Tester Report: /path/to/devicetester/
results/87e673c6-1226-11eb-9269-8c8590419f30/awsiotdevicetester_report.xml
Path to Test Execution Logs: /path/to/devicetester/
results/87e673c6-1226-11eb-9269-8c8590419f30/logs
Path to Aggregated JUnit Report: /path/to/devicetester/
results/87e673c6-1226-11eb-9269-8c8590419f30/IDTSampleSuitePython_Report.xml
```

疑難排解

使用以下資訊來協助解決完成教學課程的任何問題。

測試案例未成功執行

如果測試未成功執行，IDT 會將錯誤日誌串流至主控台，以協助您對測試執行進行疑難排解。請確定您符合本教學課程的所有[先決條件](#)。

無法連線至待測裝置

請確認下列內容：

- 您的 `device.json` 檔案包含正確的 IP 地址、連接埠和身分驗證資訊。
- 您可以從主機電腦透過 SSH 連線至裝置。

教學課程：開發簡單的 IDT 測試套件

測試套件結合了下列項目：

- 包含測試邏輯的測試可執行檔
- 描述測試套件的 JSON 組態檔案

本教學課程說明如何使用 IDT for AWS IoT Greengrass 來開發包含單一測試案例的 Python 測試套件。在本教學課程中，您將完成下列步驟：

1. [建立測試套件目錄](#)
2. [建立 JSON 組態檔案](#)
3. [建立測試案例可執行檔](#)
4. [執行測試套件](#)

先決條件

為了完成本教學，您需要以下項目：

- 主機電腦需求
 - AWS IoT Device Tester 的最新版本
 - [Python](#) 3.7 或更新版本

若要檢查安裝在電腦上的 Python 版本，請執行下列命令：

```
python3 --version
```

在 Windows 上，如果使用此命令傳回錯誤，請將 `python3` 改為 `python`。如果傳回的版本編號為 3.7 或更新版本，請在 Powershell 終端機中執行下列命令，將 `python3` 設定為 `python` 命令的別名。

```
Set-Alias -Name "python3" -Value "python"
```

如果未傳回版本資訊或版本編號小於 3.7，請遵循[下載 Python](#) 中的指示安裝 Python 3.7+。如需詳細資訊，請參閱 [Python 文件](#)。

- [urllib3](#)

若要驗證 `urllib3` 是否正確安裝，請執行下列命令：

```
python3 -c 'import urllib3'
```

如果 `urllib3` 未安裝，請執行下列命令來安裝它：

```
python3 -m pip install urllib3
```

- 裝置要求

- 具有 Linux 作業系統和與主機電腦相同網路連線的裝置。

我們建議您將 [Raspberry Pi](#) 與 Raspberry Pi 作業系統搭配使用。請務必在 Raspberry Pi 上設定 [SSH](#) 以遠端連線到它。

建立測試套件目錄

IDT 會在邏輯上將測試案例分成每個測試套件內的測試群組。每個測試案例都必須位於測試群組內。在本教學課程中，請建立名為 `MyTestSuite_1.0.0` 的資料夾，並在此資料夾中建立下列目錄樹狀目錄：

```
MyTestSuite_1.0.0
### suite
    ### myTestGroup
        ### myTestCase
```

建立 JSON 組態檔案

您的測試套件必須包含下列必要的 [JSON 組態檔案](#)：

必要的 JSON 檔案

suite.json

包含測試套件的相關資訊。請參閱 [設定 suite.json](#)。

group.json

包含測試群組的相關資訊。您必須為測試套件中的每個測試群組建立group.json檔案。請參閱 [設定 group.json](#)。

test.json

包含測試案例的相關資訊。您必須為測試套件中的每個測試案例建立test.json檔案。請參閱 [設定 test.json](#)。

1. 在 MyTestSuite_1.0.0/suite資料夾中，建立具有下列結構suite.json的檔案：

```
{
  "id": "MyTestSuite_1.0.0",
  "title": "My Test Suite",
  "details": "This is my test suite.",
  "userDataRequired": false
}
```

2. 在 MyTestSuite_1.0.0/myTestGroup資料夾中，建立具有下列結構group.json的檔案：

```
{
  "id": "MyTestGroup",
  "title": "My Test Group",
  "details": "This is my test group.",
  "optional": false
}
```

3. 在 MyTestSuite_1.0.0/myTestGroup/myTestCase資料夾中，建立具有下列結構test.json的檔案：

```
{
  "id": "MyTestCase",
```

```
{
  "title": "My Test Case",
  "details": "This is my test case.",
  "execution": {
    "timeout": 300000,
    "linux": {
      "cmd": "python3",
      "args": [
        "myTestCase.py"
      ]
    },
    "mac": {
      "cmd": "python3",
      "args": [
        "myTestCase.py"
      ]
    },
    "win": {
      "cmd": "python3",
      "args": [
        "myTestCase.py"
      ]
    }
  }
}
```

資料夾的目錄樹狀目錄現在MyTestSuite_1.0.0看起來應該如下所示：

```
MyTestSuite_1.0.0
### suite
### suite.json
### myTestGroup
### group.json
### myTestCase
### test.json
```

取得 IDT 用戶端 SDK

您可以使用 [IDT 用戶端 SDK](#) 讓 IDT 與待測裝置互動，並報告測試結果。在本教學課程中，您將使用開發套件的 Python 版本。

從 *<device-tester-extract-location>/sdks/python/* 資料夾，將 `idt_client` 資料夾複製到您的 `MyTestSuite_1.0.0/suite/myTestGroup/myTestCase` 資料夾。

若要確認軟體開發套件已成功複製，請執行下列命令。

```
cd MyTestSuite_1.0.0/suite/myTestGroup/myTestCase
python3 -c 'import idt_client'
```

建立測試案例可執行檔

測試案例可執行檔包含您要執行的測試邏輯。測試套件可以包含多個測試案例可執行檔。在本教學課程中，您只會建立一個測試案例可執行檔。

1. 建立測試套件檔案。

在 `MyTestSuite_1.0.0/suite/myTestGroup/myTestCase` 資料夾中，建立包含下列內容 `myTestCase.py` 的檔案：

```
from idt_client import *

def main():
    # Use the client SDK to communicate with IDT
    client = Client()

if __name__ == "__main__":
    main()
```

2. 使用用戶端 SDK 函數將下列測試邏輯新增至您的 `myTestCase.py` 檔案：

a. 在待測裝置上執行 SSH 命令。

```
from idt_client import *

def main():
    # Use the client SDK to communicate with IDT
    client = Client()

    # Create an execute on device request
    exec_req = ExecuteOnDeviceRequest(ExecuteOnDeviceCommand("echo 'hello world'"))

    # Run the command
    exec_resp = client.execute_on_device(exec_req)

    # Print the standard output
```

```
print(exec_resp.stdout)

if __name__ == "__main__":
    main()
```

- b. 將測試結果傳送至 IDT。

```
from idt_client import *

def main():
    # Use the client SDK to communicate with IDT
    client = Client()

    # Create an execute on device request
    exec_req = ExecuteOnDeviceRequest(ExecuteOnDeviceCommand("echo 'hello
world'"))

    # Run the command
    exec_resp = client.execute_on_device(exec_req)

    # Print the standard output
    print(exec_resp.stdout)

    # Create a send result request
    sr_req = SendResultRequest(TestResult(passed=True))

    # Send the result
    client.send_result(sr_req)

if __name__ == "__main__":
    main()
```

設定 IDT 的裝置資訊

設定您的裝置資訊，讓 IDT 執行測試。您必須使用下列資訊更新位於 *<device-tester-extract-location>*/configs 資料夾的 device.json 範本。

```
[
  {
    "id": "pool",
    "sku": "N/A",
    "devices": [
```

```
{
  "id": "<device-id>",
  "connectivity": {
    "protocol": "ssh",
    "ip": "<ip-address>",
    "port": "<port>",
    "auth": {
      "method": "pki | password",
      "credentials": {
        "user": "<user-name>",
        "privKeyPath": "/path/to/private/key",
        "password": "<password>"
      }
    }
  }
}
```

在 devices 物件中，提供下列資訊：

id

您裝置的使用者定義唯一識別符。

connectivity.ip

您裝置的 IP 地址。

connectivity.port

選用。用於 SSH 連線至您裝置的連接埠號碼。

connectivity.auth

連線的驗證資訊。

只有當 connectivity.protocol 設為 ssh 時，才會套用此屬性。

connectivity.auth.method

用來透過指定的連線通訊協定存取裝置的驗證方法。

支援的值如下：

- pki

- password

`connectivity.auth.credentials`

用於驗證的登入資料。

`connectivity.auth.credentials.user`

用來登入裝置的使用者名稱。

`connectivity.auth.credentials.privKeyPath`

用來登入您裝置之私有金鑰的完整路徑。

只有當 `connectivity.auth.method` 設為 `pki` 時，才會套用此值。

`devices.connectivity.auth.credentials.password`

用於登入您裝置的密碼。

只有當 `connectivity.auth.method` 設為 `password` 時，才會套用此值。

Note

如果 `method` 是設定為 `pki`，則指定 `privKeyPath`

如果 `method` 是設定為 `password`，則指定 `password`

執行測試套件

建立測試套件之後，您想要確保其如預期般運作。完成下列步驟，以使用您現有的裝置集區執行測試套件。

1. 將您的 `MyTestSuite_1.0.0` 資料夾複製到 `<device-tester-extract-location>/tests`。
2. 執行下列命令：

```
cd <device-tester-extract-location>/bin
./devicetester_[linux | mac | win_x86-64] run-suite --suite-id MyTestSuite
```

IDT 會執行您的測試套件，並將結果串流至主控台。當測試完成執行時，您會看到下列資訊：

```
time="2020-10-19T09:24:47-07:00" level=info msg=Using pool: pool
time="2020-10-19T09:24:47-07:00" level=info msg=Using test suite "MyTestSuite_1.0.0"
  for execution
time="2020-10-19T09:24:47-07:00" level=info msg=b'hello world\n'
  suiteId=MyTestSuite groupId=myTestGroup testCaseId=myTestCase deviceId=my-device
  executionId=9a52f362-1227-11eb-86c9-8c8590419f30
time="2020-10-19T09:24:47-07:00" level=info msg=All tests finished.
  executionId=9a52f362-1227-11eb-86c9-8c8590419f30
time="2020-10-19T09:24:48-07:00" level=info msg=

===== Test Summary =====
Execution Time:          1s
Tests Completed:         1
Tests Passed:            1
Tests Failed:            0
Tests Skipped:           0
-----
Test Groups:
  myTestGroup:           PASSED
-----
Path to IoT Device Tester Report: /path/to/devicetester/
results/9a52f362-1227-11eb-86c9-8c8590419f30/awsiotdevicetester_report.xml
Path to Test Execution Logs: /path/to/devicetester/
results/9a52f362-1227-11eb-86c9-8c8590419f30/logs
Path to Aggregated JUnit Report: /path/to/devicetester/
results/9a52f362-1227-11eb-86c9-8c8590419f30/MyTestSuite_Report.xml
```

疑難排解

使用以下資訊來協助解決完成教學課程的任何問題。

測試案例未成功執行

如果測試未成功執行，IDT 會將錯誤日誌串流至主控台，以協助您對測試執行進行疑難排解。檢查錯誤日誌之前，請確認下列事項：

- IDT 用戶端 SDK 位於正確的資料夾中，如[此步驟](#)所述。
- 您符合本教學課程的所有[先決條件](#)。

無法連線至待測裝置

請確認下列內容：

- 您的 `device.json` 檔案包含正確的 IP 地址、連接埠和身分驗證資訊。
- 您可以從主機電腦透過 SSH 連線至裝置。

建立 IDT 測試套件組態檔案

本節說明您在撰寫自訂測試套件時，建立 JSON 組態檔案的格式。

必要的 JSON 檔案

`suite.json`

包含測試套件的相關資訊。請參閱 [設定 `suite.json`](#)。

`group.json`

包含測試群組的相關資訊。您必須為測試套件中的每個測試群組建立 `group.json` 檔案。請參閱 [設定 `group.json`](#)。

`test.json`

包含測試案例的相關資訊。您必須為測試套件中的每個測試案例建立 `test.json` 檔案。請參閱 [設定 `test.json`](#)。

選用 JSON 檔案

`state_machine.json`

定義 IDT 執行測試套件時如何執行測試。請參閱 [設定 `state\_machine.json`](#)。

`userdata_schema.json`

定義測試執行器可以包含在其設定組態中的 [userdata.json 檔案結構描述](#)。 `userdata.json` 檔案會用於執行測試所需的任何其他組態資訊，但不會出現在 `device.json` 檔案中。請參閱 [設定 `userdata\_schema.json`](#)。

JSON 組態檔案會放置在中 `<custom-test-suite-folder>`，如下所示。

```
<custom-test-suite-folder>
### suite
  ### suite.json
  ### state_machine.json
  ### userdata_schema.json
```

```
### <test-group-folder>
### group.json
### <test-case-folder>
### test.json
```

設定 suite.json

suite.json 檔案會設定環境變數，並判斷是否需要使用者資料才能執行測試套件。使用下列範本來設定您的 *<custom-test-suite-folder>*/suite/suite.json 檔案：

```
{
  "id": "<suite-name>_<suite-version>",
  "title": "<suite-title>",
  "details": "<suite-details>",
  "userDataRequired": true | false,
  "environmentVariables": [
    {
      "key": "<name>",
      "value": "<value>",
    },
    ...
    {
      "key": "<name>",
      "value": "<value>",
    }
  ]
}
```

如下所述，包含值的所有欄位皆為必要：

id

測試套件的唯一使用者定義 ID。的值id必須符合suite.json檔案所在的測試套件資料夾名稱。套件名稱和套件版本也必須符合下列要求：

- *<suite-name>* 不能包含底線。
- *<suite-version>* 表示為 *x.x.x*，其中 x是數字。

ID 會顯示在 IDT 產生的測試報告中。

title

此測試套件所測試之產品或功能的使用者定義名稱。名稱會顯示在測試執行器的 IDT CLI 中。

details

測試套件用途的簡短描述。

userDataRequired

定義測試執行器是否需要在 `userdata.json` 檔案中包含自訂資訊。如果您將此值設定為 `true`，您還必須在測試套件資料夾中包含 [userdata_schema.json 檔案](#)。

environmentVariables

選用。要為此測試套件設定的環境變數陣列。

environmentVariables.key

環境變數的名稱。

environmentVariables.value

環境變數的值。

設定 group.json

`group.json` 檔案會定義測試群組是必要或選用。使用下列範本來設定 *<custom-test-suite-folder>/suite/<test-group>/group.json* 檔案：

```
{
  "id": "<group-id>",
  "title": "<group-title>",
  "details": "<group-details>",
  "optional": true | false,
}
```

如下所述，包含值的所有欄位皆為必要：

id

測試群組的唯一使用者定義 ID。的值 `id` 必須符合 `group.json` 檔案所在的測試群組資料夾名稱，且不得包含底線 (`_`)。ID 用於 IDT 產生的測試報告。

title

測試群組的描述性名稱。名稱會顯示在測試執行器的 IDT CLI 中。

details

測試群組用途的簡短描述。

optional

選用。設定為 true 以在 IDT 完成執行所需的測試後，將此測試群組顯示為選用群組。預設值為 false。

設定 test.json

test.json 檔案會決定測試案例可執行檔和測試案例所使用的環境變數。如需建立測試案例可執行檔的詳細資訊，請參閱 [建立 IDT 測試案例可執行檔](#)。

使用下列範本來設定 `<custom-test-suite-folder>/suite/<test-group>/<test-case>/` test.json 檔案：

```
{
  "id": "<test-id>",
  "title": "<test-title>",
  "details": "<test-details>",
  "requireDUT": true | false,
  "requiredResources": [
    {
      "name": "<resource-name>",
      "features": [
        {
          "name": "<feature-name>",
          "version": "<feature-version>",
          "jobSlots": <job-slots>
        }
      ]
    }
  ],
  "execution": {
    "timeout": <timeout>,
    "mac": {
      "cmd": "/path/to/executable",
      "args": [
        "<argument>"
      ]
    }
  },
}
```

```
    "linux": {
      "cmd": "/path/to/executable",
      "args": [
        "<argument>"
      ],
    },
    "win": {
      "cmd": "/path/to/executable",
      "args": [
        "<argument>"
      ]
    }
  },
  "environmentVariables": [
    {
      "key": "<name>",
      "value": "<value>",
    }
  ]
}
```

如下所述，包含值的所有欄位皆為必要：

id

測試案例的唯一使用者定義 ID。的值id必須符合test.json檔案所在的測試案例資料夾名稱，且不得包含底線(_)。ID 用於 IDT 產生的測試報告。

title

測試案例的描述性名稱。名稱會顯示在測試執行器的 IDT CLI 中。

details

測試案例用途的簡短描述。

requireDUT

選用。true 如果需要裝置執行此測試，請將 設定為 true，否則將 設定為 false。預設值為 true。測試執行器會設定其用來在device.json檔案中執行測試的裝置。

requiredResources

選用。陣列，提供執行此測試所需的資源裝置相關資訊。

`requiredResources.name`

執行此測試時提供給資源裝置的唯一名稱。

`requiredResources.features`

使用者定義資源裝置功能的陣列。

`requiredResources.features.name`

功能的名稱。您要使用此裝置的裝置功能。此名稱會與 `resource.json` 檔案中測試執行器所提供的功能名稱相符。

`requiredResources.features.version`

選用。功能的版本。此值會與 `resource.json` 檔案中測試執行器所提供的功能版本相符。如果未提供版本，則不會檢查該功能。如果功能不需要版本號碼，請將此欄位保留空白。

`requiredResources.features.jobSlots`

選用。此功能可支援的同時測試數量。預設值為 1。如果您希望 IDT 對個別功能使用不同的裝置，建議您將此值設定為 1。

`execution.timeout`

IDT 等待測試完成執行的時間量（以毫秒為單位）。如需設定此值的詳細資訊，請參閱 [建立 IDT 測試案例可執行檔](#)。

`execution.os`

要根據執行 IDT 之主機電腦的作業系統執行的測試案例可執行檔。支援的值為 `linux`、`mac` 和 `win`。

`execution.os.cmd`

您要針對指定作業系統執行的測試案例可執行檔路徑。此位置必須位於系統路徑中。

`execution.os.args`

選用。用來執行測試案例可執行檔的引數。

`environmentVariables`

選用。此測試案例設定的環境變數陣列。

`environmentVariables.key`

環境變數的名稱。

`environmentVariables.value`

環境變數的值。

 Note

如果您在 `test.json` 檔案和 `test_suite.json` 檔案中指定相同的環境變數 `test_suite.json`，則 `test.json` 檔案的值優先。

設定 `state_machine.json`

狀態機器是一種控制測試套件執行流程的建構。它決定測試套件的開始狀態、根據使用者定義的規則管理狀態轉換，並繼續轉換這些狀態，直到達到結束狀態為止。

如果您的測試套件不包含使用者定義的狀態機器，IDT 會為您產生狀態機器。預設狀態機器會執行下列函數：

- 讓測試執行器能夠選取和執行特定測試群組，而不是整個測試套件。
- 如果未選取特定測試群組，會以隨機順序執行測試套件中的每個測試群組。
- 產生報告並列印主控台摘要，以顯示每個測試群組和測試案例的測試結果。

如需 IDT 狀態機器如何運作的詳細資訊，請參閱 [設定 IDT 狀態機器](#)。

設定 `userdata_schema.json`

`userdata_schema.json` 檔案會決定測試執行器提供使用者資料的結構描述。如果您的測試套件需要 `device.json` 檔案中不存在的資訊，則需要使用者資料。例如，您的測試可能需要 Wi-Fi 網路登入資料、特定開放連接埠或使用者必須提供的憑證。此資訊可以做為名為 `userdata` 的輸入參數提供給 IDT，這是使用者在其 `<device-tester-extract-location>/config` 資料夾中建立 `userdata.json` 的檔案 `userdata` 值。`userdata.json` 檔案格式是根據您在測試套件中包含的 `userdata_schema.json` 檔案。

若要指出測試執行器必須提供 `userdata.json` 檔案：

1. 在 `suite.json` 檔案中，將 `userDataRequired` 設定為 `true`。
2. 在您的 `<custom-test-suite-folder>` 中，建立 `userdata_schema.json` 檔案。
3. 編輯 `userdata_schema.json` 檔案以建立有效的 [IETF 草稿 v4 JSON 結構描述](#)。

當 IDT 執行您的測試套件時，它會自動讀取結構描述，並使用它來驗證測試執行器所提供的 `userdata.json` 檔案。如果有效，`userdata.json` 檔案的內容可在 [IDT 內容](#) 和 [狀態機器內容](#) 中使用。

設定 IDT 狀態機器

狀態機器是一種控制測試套件執行流程的建構。它決定測試套件的開始狀態、根據使用者定義的規則管理狀態轉換，並繼續轉換這些狀態，直到達到結束狀態為止。

如果您的測試套件不包含使用者定義的狀態機器，IDT 會為您產生狀態機器。預設狀態機器會執行下列函數：

- 讓測試執行器能夠選取和執行特定測試群組，而不是整個測試套件。
- 如果未選取特定測試群組，會以隨機順序執行測試套件中的每個測試群組。
- 產生報告並列印主控台摘要，顯示每個測試群組和測試案例的測試結果。

IDT 測試套件的狀態機器必須符合下列條件：

- 每個狀態對應於 IDT 要採取的動作，例如執行測試群組或製作報告檔案的產品。
- 轉換為 狀態會執行與 狀態相關聯的動作。
- 每個狀態都會定義下一個狀態的轉換規則。
- 結束狀態必須為 Succeed 或 Fail。

狀態機器格式

您可以使用下列範本來設定自己的 `<custom-test-suite-folder>/suite/state_machine.json` 檔案：

```
{
  "Comment": "<description>",
  "StartAt": "<state-name>",
  "States": {
    "<state-name>": {
      "Type": "<state-type>",
      // Additional state configuration
    }

    // Required states
  }
}
```

```
"Succeed": {
  "Type": "Succeed"
},
"Fail": {
  "Type": "Fail"
}
}
```

如下所述，包含值的所有欄位皆為必要：

Comment

狀態機器的描述。

StartAt

IDT 開始執行測試套件的狀態名稱。的值StartAt必須設定為 States 物件中列出的其中一個狀態。

States

將使用者定義狀態名稱映射至有效 IDT 狀態的物件。每個 States.*state-name* 物件都包含映射至 *state-name* 的有效狀態定義。

States 物件必須包含 Succeed和 Fail 狀態。如需有效狀態的資訊，請參閱 [有效狀態和狀態定義](#)。

有效狀態和狀態定義

本節說明可在 IDT 狀態機器中使用的所有有效狀態的狀態定義。下列某些狀態支援測試案例層級的組態。不過，建議您在測試群組層級設定狀態轉換規則，而非測試案例層級，除非絕對必要。

狀態定義

- [RunTask](#)
- [Choice](#)
- [平行](#)
- [AddProductFeatures](#)
- [報告](#)
- [LogMessage](#)

- [SelectGroup](#)
- [失敗](#)
- [Succeed](#)

RunTask

RunTask 狀態會從測試套件中定義的測試群組執行測試案例。

```
{
  "Type": "RunTask",
  "Next": "<state-name>",
  "TestGroup": "<group-id>",
  "TestCases": [
    "<test-id>"
  ],
  "ResultVar": "<result-name>"
}
```

如下所述，包含值的所有欄位皆為必要：

Next

在目前狀態下執行動作後，要轉換為的狀態名稱。

TestGroup

選用。要執行的測試群組 ID。如果未指定此值，IDT 會執行測試執行器選取的測試群組。

TestCases

選用。來自 中指定群組的測試案例 IDs 陣列TestGroup。根據 TestGroup和 的值TestCases，IDT 會決定測試執行行為，如下所示：

- 同時指定 TestCases TestGroup和 時，IDT 會從測試群組執行指定的測試案例。
- TestCases 指定TestGroup但未指定 時，IDT 會執行指定的測試案例。
- 指定 TestCases 但未指定 TestGroup 時，IDT 會執行指定測試群組中的所有測試案例。
- 未指定 TestCases TestGroup或 時，IDT 會從測試執行器從 IDT CLI 中選取的測試群組執行所有測試案例。若要啟用測試執行器的群組選擇，您必須在 statemachine.json 檔案中同時包含 RunTask和 Choice 狀態。如需如何運作的範例，請參閱[範例狀態機器：執行使用者選取的測試群組](#)。

如需為測試執行器啟用 IDT CLI 命令的詳細資訊，請參閱 [the section called “啟用 IDT CLI 命令”](#)。

ResultVar

使用測試執行結果設定的內容變數名稱。如果您未指定的值，請勿指定此值TestGroup。IDT false會根據下列項目，將您在 中定義的變數值設定為 ResultVartrue或：

- 如果變數名稱的格式為 `text_text_passed`，則值會設定為是否通過或略過第一個測試群組中的所有測試。
- 在所有其他情況下，該值會設定為是否通過或略過所有測試群組中的所有測試。

一般而言，您將使用 RunTask 狀態來指定測試群組 ID，而不指定個別測試案例 IDs，以便 IDT 執行指定測試群組中的所有測試案例。此狀態執行的所有測試案例都會以隨機順序平行執行。不過，如果所有測試案例都需要裝置才能執行，而且只有單一裝置可用，則測試案例將改為依序執行。

錯誤處理

如果任何指定的測試群組或測試案例 IDs 無效，則此狀態會發出RunTaskError執行錯誤。如果狀態遇到執行錯誤，則也會將狀態機器內容中的hasExecutionError變數設定為 true。

Choice

Choice 狀態可讓您動態設定下一個狀態，以根據使用者定義的條件轉換為。

```
{
  "Type": "Choice",
  "Default": "<state-name>",
  "FallthroughOnError": true | false,
  "Choices": [
    {
      "Expression": "<expression>",
      "Next": "<state-name>"
    }
  ]
}
```

如下所述，包含值的所有欄位皆為必要：

Default

如果 中定義的任何表達式Choices都無法評估為 ，則要轉換為的預設狀態true。

FallthroughOnError

選用。指定狀態在評估表達式時發生錯誤時的行為。`true` 如果您想要在評估導致錯誤時略過表達式，請將 設定為 `true`。如果沒有相符的表達式，則狀態機器會轉換為 `Default` 狀態。如果未指定 `FallthroughOnError` 值，則預設為 `false`。

Choices

運算式和狀態的陣列，用於判斷在目前狀態下執行動作後要轉換到哪個狀態。

Choices.Expression

評估為布林值的表達式字串。如果表達式評估為 `true`，則狀態機器會轉換為 中定義的狀態 `Choices.Next`。表達式字串會從狀態機器內容擷取值，然後對其執行操作以到達布林值。如需存取狀態機器內容的資訊，請參閱 [狀態機器內容](#)。

Choices.Next

如果 中定義的表達式 `Choices.Expression` 評估為 `true`，則要轉換為的狀態名稱 `true`。

錯誤處理

在下列情況下，`Choice` 狀態可能需要錯誤處理：

- 選擇表達式中的某些變數不存在於狀態機器內容中。
- 表達式的結果不是布林值。
- JSON 查詢的結果不是字串、數字或布林值。

您無法使用 `Catch` 區塊來處理此狀態的錯誤。如果您想要在遇到錯誤時停止執行狀態機器，您必須將 `FallthroughOnError` 設定為 `false`。不過，我們建議您將 `FallthroughOnError` 設定為 `true`，並根據您的使用案例執行下列其中一項操作：

- 如果您存取的變數預期在某些情況下不存在，請使用 值 `Default` 和其他 `Choices` 區塊來指定下一個狀態。
- 如果您存取的變數應一律存在，請將 `Default` 狀態設定為 `Fail`。

平行

`Parallel` 狀態可讓您彼此平行定義和執行新的狀態機器。

```
{
```

```
"Type": "Parallel",
"Next": "<state-name>",
"Branches": [
    <state-machine-definition>
]
}
```

如下所述，包含值的所有欄位皆為必要：

Next

在目前狀態下執行動作後，要轉換為的狀態名稱。

Branches

要執行的狀態機器定義陣列。每個狀態機器定義都必須包含自己的 StartAt、Succeed 和 Fail 狀態。此陣列中的狀態機器定義無法參考其定義以外的狀態。

Note

由於每個分支狀態機器共用相同的狀態機器內容，因此在一個分支中設定變數，然後從另一個分支讀取這些變數可能會導致意外行為。

Parallel 狀態只有在執行所有分支狀態機器之後才會移至下一個狀態。需要裝置的每個狀態都會等待執行，直到裝置可用為止。如果有多個裝置可用，此狀態會從多個群組平行執行測試案例。如果沒有足夠的裝置可用，則測試案例將依序執行。由於測試案例在平行執行時以隨機順序執行，因此可能會使用不同的裝置從相同的測試群組執行測試。

錯誤處理

請確定分支狀態機器和父系狀態機器都會轉換為 Fail 狀態，以處理執行錯誤。

由於分支狀態機器不會將執行錯誤傳輸至父系狀態機器，因此您無法使用 Catch 區塊來處理分支狀態機器中的執行錯誤。反之，請在共用狀態機器內容中使用 hasExecutionErrors 值。如需如何運作的範例，請參閱 [狀態機器範例：平行執行兩個測試群組](#)。

AddProductFeatures

AddProductFeatures 狀態可讓您將產品功能新增至 IDT 產生的 awsiotdevicetester_report.xml 檔案。

產品功能是有關裝置可能符合的特定條件的使用者定義資訊。例如，MQTT 產品功能可以指定裝置正確發佈 MQTT 訊息。在報告中，產品功能會根據是否通過指定的測試，設定為 supported、not-supported 或自訂值。

Note

AddProductFeatures 狀態本身不會產生報告。此狀態必須轉換為 [Report 狀態](#) 才能產生報告。

```
{
  "Type": "Parallel",
  "Next": "<state-name>",
  "Features": [
    {
      "Feature": "<feature-name>",
      "Groups": [
        "<group-id>"
      ],
      "OneOfGroups": [
        "<group-id>"
      ],
      "TestCases": [
        "<test-id>"
      ],
      "IsRequired": true | false,
      "ExecutionMethods": [
        "<execution-method>"
      ]
    }
  ]
}
```

如下所述，包含值的所有欄位皆為必要：

Next

在目前狀態下執行動作後，要轉換為的狀態名稱。

Features

要在 awsiotdevicetester_report.xml 檔案中顯示的產品功能陣列。

Feature

功能的名稱

FeatureValue

選用。要在報告中使用的自訂值，而非 supported。如果未指定此值，則根據測試結果，功能值會設為 supported 或 not-supported。

如果您使用的自訂值 FeatureValue，則可以使用不同的條件測試相同的功能，IDT 會串連支援條件的功能值。例如，以下摘錄顯示具有兩個不同 MyFeature 特徵值的功能：

```
...
{
  "Feature": "MyFeature",
  "FeatureValue": "first-feature-supported",
  "Groups": ["first-feature-group"]
},
{
  "Feature": "MyFeature",
  "FeatureValue": "second-feature-supported",
  "Groups": ["second-feature-group"]
},
...
```

如果兩個測試群組都通過，則功能值會設為 first-feature-supported, second-feature-supported。

Groups

選用。測試群組 IDs 的陣列。每個指定測試群組中的所有測試都必須通過，才能支援此功能。

OneOfGroups

選用。測試群組 IDs 的陣列。至少一個指定測試群組內的所有測試都必須通過，才能支援此功能。

TestCases

選用。測試案例 IDs 的陣列。如果您指定此值，則適用下列條件：

- 所有指定的測試案例都必須通過，才能支援此功能。
- Groups 只能包含一個測試群組 ID。
- OneOfGroups 不得指定。

IsRequired

選用。設定為 `false` 以在報告中將此功能標記為選用功能。預設值為 `true`。

ExecutionMethods

選用。符合 `device.json` 檔案中指定 `protocol` 值的執行方法陣列。如果指定此值，則測試執行器必須指定 `protocol` 符合此陣列中其中一個值的值，以在報告中包含 功能。如果未指定此值，則功能一律會包含在報告中。

若要使用 `AddProductFeatures` 狀態，您必須將 `RunTask` 狀態 `ResultVar` 中的 值設定為下列其中一個值：

- 如果您指定個別測試案例 IDs，請將 `ResultVar` 設定為 `group-id_test-id_passed`。
- 如果您未指定個別測試案例 IDs，請將 `ResultVar` 設定為 `group-id_passed`。

`AddProductFeatures` 狀態會以下列方式檢查測試結果：

- 如果您未指定任何測試案例 IDs，則每個測試群組的結果取決於狀態機器內容中的 `group-id_passed` 變數值。
- 如果您確實指定測試案例 IDs，則每個測試的結果取決於狀態機器內容中的 `group-id_test-id_passed` 變數值。

錯誤處理

如果在此狀態下提供的群組 ID 不是有效的群組 ID，則此狀態會導致 `AddProductFeaturesError` 執行錯誤。如果狀態遇到執行錯誤，則也會將狀態機器內容中的 `hasExecutionErrors` 變數設定為 `true`。

報告

`Report` 狀態會產生 `suite-name_Report.xml` 和 `awsiotdevicetester_report.xml` 檔案。此狀態也會將報告串流至 主控台。

```
{
  "Type": "Report",
  "Next": "<state-name>"
}
```

如下所述，包含值的所有欄位皆為必要：

Next

在目前狀態下執行動作後，要轉換為的狀態名稱。

您應該一律在測試執行流程結束時轉換為 Report 狀態，以便測試執行器可以檢視測試結果。一般而言，此狀態之後的下一個狀態為 Succeed。

錯誤處理

如果此狀態在產生報告時遇到問題，則會發出 ReportError 執行錯誤。

LogMessage

LogMessage 狀態會產生 test_manager.log 檔案，並將日誌訊息串流至 主控台。

```
{
  "Type": "LogMessage",
  "Next": "<state-name>"
  "Level": "info | warn | error"
  "Message": "<message>"
}
```

如下所述，包含值的所有欄位皆為必要：

Next

在目前狀態下執行動作後，要轉換為的狀態名稱。

Level

建立日誌訊息的錯誤層級。如果您指定的關卡無效，此狀態會產生錯誤訊息並予以捨棄。

Message

要記錄的訊息。

SelectGroup

SelectGroup 狀態會更新狀態機器內容，以指出已選取哪些群組。此狀態設定的值會由任何後續 Choice 狀態使用。

```
{
```

```
"Type": "SelectGroup",
"Next": "<state-name>"
"TestGroups": [
    <group-id>
]
```

如下所述，包含值的所有欄位皆為必要：

Next

在目前狀態下執行動作後，要轉換為的狀態名稱。

TestGroups

將標記為選取的測試群組陣列。對於此陣列中的每個測試群組 ID，*group-id_selected*變數會在內容true中設定為。請務必提供有效的測試群組 IDs因為 IDT 不會驗證指定的群組是否存在。

失敗

Fail 狀態表示狀態機器未正確執行。這是狀態機器的結束狀態，每個狀態機器定義都必須包含此狀態。

```
{
  "Type": "Fail"
}
```

Succeed

Succeed 狀態表示狀態機器已正確執行。這是狀態機器的結束狀態，每個狀態機器定義都必須包含此狀態。

```
{
  "Type": "Succeed"
}
```

狀態機器內容

狀態機器內容是唯讀 JSON 文件，其中包含可在執行期間提供給狀態機器的資料。狀態機器內容只能從狀態機器存取，並包含決定測試流程的資訊。例如，您可以使用 `userdata.json` 檔案中測試執行器設定的資訊來判斷是否需要執行特定測試。

狀態機器內容使用下列格式：

```
{
  "pool": {
    <device-json-pool-element>
  },
  "userData": {
    <userdata-json-content>
  },
  "config": {
    <config-json-content>
  },
  "suiteFailed": true | false,
  "specificTestGroups": [
    "<group-id>"
  ],
  "specificTestCases": [
    "<test-id>"
  ],
  "hasExecutionErrors": true
}
```

pool

為測試執行選取之裝置集區的相關資訊。對於選取的裝置集區，此資訊會從 device.json 檔案中定義的對應頂層裝置集區陣列元素擷取。

userData

userdata.json 檔案中的資訊。

config

資訊鎖定 config.json 檔案。

suiteFailed

狀態機器啟動 false 時，此值會設為。如果測試群組在 RunTask 狀態失敗，則此值會在狀態機器執行的剩餘持續時間內設定為 true。

specificTestGroups

如果測試執行器選取要執行的特定測試群組，而不是整個測試套件，則會建立此金鑰並包含特定測試群組 IDs 的清單。

specificTestCases

如果測試執行器選取要執行的特定測試案例，而不是整個測試套件，則會建立此金鑰並包含特定測試案例 IDs 的清單。

hasExecutionErrors

狀態機器啟動時不會結束。如果任何狀態遇到執行錯誤，則會建立此變數 `true`，並在狀態機器執行的剩餘持續時間內設定為 `false`。

您可以使用 JSONPath 表示法查詢內容。狀態定義中 JSONPath 查詢的語法為 `{{$.query}}`。您可以在某些狀態下使用 JSONPath 查詢做為預留位置字串。IDT 會將預留位置字串取代為內容中評估 JSONPath 查詢的值。您可以針對下列值使用預留位置：

- 狀態 `TestCases` 的值 `RunTask`。
- `Expression` 值 `Choice` 狀態。

當您從狀態機器內容存取資料時，請確定符合下列條件：

- 您的 JSON 路徑必須以開頭 `$`。
- 每個值都必須評估為字串、數字或布林值。

如需使用 JSONPath 表示法從內容存取資料的詳細資訊，請參閱 [使用 IDT 內容](#)。

執行錯誤

執行錯誤是狀態機器在執行狀態時遇到的狀態機器定義錯誤。IDT 會記錄 `test_manager.log` 檔案中每個錯誤的相關資訊，並將日誌訊息串流至主控台。

您可以使用下列方法來處理執行錯誤：

- 在狀態定義中新增 [Catch 區塊](#)。
- 在狀態機器內容中檢查 [hasExecutionErrors](#) 值的值。

擷取

若要使用 `Catch`，請將下列項目新增至您的狀態定義：

```
"Catch": [
```

```
{
  "ErrorEquals": [
    "<error-type>"
  ]
  "Next": "<state-name>"
}
```

如下所述，包含值的所有欄位皆為必要：

Catch.ErrorEquals

要擷取的錯誤類型陣列。如果執行錯誤符合其中一個指定的值，則狀態機器會轉換為 中指定的狀態Catch.Next。如需其產生的錯誤類型的相關資訊，請參閱每個狀態定義。

Catch.Next

如果目前狀態遇到符合 中指定其中一個值的執行錯誤，則會轉換為下一個狀態Catch.ErrorEquals。

擷取區塊會依序處理，直到一個相符為止。如果沒有錯誤符合 Catch 區塊中列出的錯誤，則狀態機器會繼續執行。由於執行錯誤是不正確狀態定義的結果，我們建議您在狀態遇到執行錯誤時轉換為失敗狀態。

hasExecutionError

當某些狀態遇到執行錯誤時，除了發出錯誤之外，也會在狀態機器內容true中將 hasExecutionError 值設定為 。您可以使用此值來偵測何時發生錯誤，然後使用 Choice 狀態將狀態機器轉換為 Fail 狀態。

此方法具有下列特性。

- 狀態機器不會以指派給 的任何值開始hasExecutionError，且此值在特定狀態設定之前無法使用。這表示您必須false針對存取此值Choice的狀態，明確FallthroughOnError將 設定為 ，以防止狀態機器在未發生執行錯誤時停止。
- 一旦設定為 true，hasExecutionError 永遠不會設定為 false 或從內容中移除。這表示此值只有在第一次設定為 時才有用true，而且對於所有後續狀態，它不提供有意義的值。
- 該hasExecutionError值會與 Parallel 狀態的所有分支狀態機器共用，這可能會導致非預期的結果，具體取決於存取它的順序。

由於這些特性，如果您可以改用 Catch 區塊，我們不建議您使用此方法。

狀態機器範例

本節提供一些範例狀態機器組態。

範例

- [狀態機器範例：執行單一測試群組](#)
- [狀態機器範例：執行使用者選取的測試群組](#)
- [狀態機器範例：使用產品功能執行單一測試群組](#)
- [狀態機器範例：平行執行兩個測試群組](#)

狀態機器範例：執行單一測試群組

此狀態機器：

- 執行 ID 為 的測試群組GroupA，該群組必須存在於 group.json 檔案的 套件中。
- 檢查執行錯誤，Fail如果找到任何，則轉換為。
- Succeed 如果沒有錯誤，則產生報告並轉換為，Fail否則。

```
{
  "Comment": "Runs a single group and then generates a report.",
  "StartAt": "RunGroupA",
  "States": {
    "RunGroupA": {
      "Type": "RunTask",
      "Next": "Report",
      "TestGroup": "GroupA",
      "Catch": [
        {
          "ErrorEquals": [
            "RunTaskError"
          ],
          "Next": "Fail"
        }
      ]
    },
    "Report": {
      "Type": "Report",
```



```

        "Next": "Succeed",
        "Catch": [
            {
                "ErrorEquals": [
                    "ReportError"
                ],
                "Next": "Fail"
            }
        ],
        "Succeed": {
            "Type": "Succeed"
        },
        "Fail": {
            "Type": "Fail"
        }
    }
}

```

狀態機器範例：執行使用者選取的測試群組

此狀態機器：

- 檢查測試執行器是否已選取特定測試群組。狀態機器不會檢查特定測試案例，因為測試執行器無法在未同時選取測試群組的情況下選取測試案例。
- 如果選取測試群組：
 - 在選取的測試群組中執行測試案例。若要這樣做，狀態機器不會明確指定 RunTask 狀態中的任何測試群組或測試案例。
 - 執行所有測試並結束之後產生報告。
- 如果未選取測試群組：
 - 在測試群組 中執行測試GroupA。
 - 產生報告並結束。

```

{
    "Comment": "Runs specific groups if the test runner chose to do that, otherwise runs GroupA.",
    "StartAt": "SpecificGroupsCheck",
    "States": {
        "SpecificGroupsCheck": {

```

```

        "Type": "Choice",
        "Default": "RunGroupA",
        "FallthroughOnError": true,
        "Choices": [
            {
                "Expression": "{{$.specificTestGroups[0]}} != ''",
                "Next": "RunSpecificGroups"
            }
        ]
    },
    "RunSpecificGroups": {
        "Type": "RunTask",
        "Next": "Report",
        "Catch": [
            {
                "ErrorEquals": [
                    "RunTaskError"
                ],
                "Next": "Fail"
            }
        ]
    },
    "RunGroupA": {
        "Type": "RunTask",
        "Next": "Report",
        "TestGroup": "GroupA",
        "Catch": [
            {
                "ErrorEquals": [
                    "RunTaskError"
                ],
                "Next": "Fail"
            }
        ]
    },
    "Report": {
        "Type": "Report",
        "Next": "Succeed",
        "Catch": [
            {
                "ErrorEquals": [
                    "ReportError"
                ],
                "Next": "Fail"
            }
        ]
    }
}

```

```

        }
    ]
},
"Succeed": {
    "Type": "Succeed"
},
"Fail": {
    "Type": "Fail"
}
}
}

```

狀態機器範例：使用產品功能執行單一測試群組

此狀態機器：

- 執行測試群組 GroupA。
- 檢查執行錯誤，Fail如果找到任何，則轉換為。
- 將 FeatureThatDependsOnGroupA功能新增至 awsiotdevicetester_report.xml 檔案：
 - 如果GroupA通過，則功能會設定為 supported。
 - 該功能在報告中不會標記為選用。
- Succeed 如果沒有錯誤，則產生報告並轉換為，Fail否則

```

{
  "Comment": "Runs GroupA and adds product features based on GroupA",
  "StartAt": "RunGroupA",
  "States": {
    "RunGroupA": {
      "Type": "RunTask",
      "Next": "AddProductFeatures",
      "TestGroup": "GroupA",
      "ResultVar": "GroupA_passed",
      "Catch": [
        {
          "ErrorEquals": [
            "RunTaskError"
          ],
          "Next": "Fail"
        }
      ]
    }
  ]
}

```

```

    },
    "AddProductFeatures": {
      "Type": "AddProductFeatures",
      "Next": "Report",
      "Features": [
        {
          "Feature": "FeatureThatDependsOnGroupA",
          "Groups": [
            "GroupA"
          ],
          "IsRequired": true
        }
      ]
    },
    "Report": {
      "Type": "Report",
      "Next": "Succeed",
      "Catch": [
        {
          "ErrorEquals": [
            "ReportError"
          ],
          "Next": "Fail"
        }
      ]
    },
    "Succeed": {
      "Type": "Succeed"
    },
    "Fail": {
      "Type": "Fail"
    }
  }
}

```

狀態機器範例：平行執行兩個測試群組

此狀態機器：

- 平行執行 GroupA 和 GroupB 測試群組。分支狀態機器中 RunTask 狀態存放在內容中的 ResultVar 變數可供 AddProductFeatures 狀態使用。
- 檢查執行錯誤，Fail 如果找到任何，則轉換為。此狀態機器不會使用 Catch 區塊，因為該方法不會偵測分支狀態機器中的執行錯誤。

- 根據傳遞的群組，將功能新增至 `awsiotdevicetester_report.xml` 檔案
 - 如果 GroupA 通過，則功能會設定為 `supported`。
 - 該功能在報告中不會標記為選用。
- Succeed 如果沒有錯誤，則產生報告並轉換為 `Succeed`，Fail 否則

如果在裝置集區中設定了兩個裝置，則 GroupA 和 GroupB 可以同時執行。不過，如果 GroupA 或有 GroupB 多個測試，則兩個裝置可能會配置到這些測試。如果只設定一個裝置，測試群組將依序執行。

```
{
  "Comment": "Runs GroupA and GroupB in parallel",
  "StartAt": "RunGroupAAndB",
  "States": {
    "RunGroupAAndB": {
      "Type": "Parallel",
      "Next": "CheckForErrors",
      "Branches": [
        {
          "Comment": "Run GroupA state machine",
          "StartAt": "RunGroupA",
          "States": {
            "RunGroupA": {
              "Type": "RunTask",
              "Next": "Succeed",
              "TestGroup": "GroupA",
              "ResultVar": "GroupA_passed",
              "Catch": [
                {
                  "ErrorEquals": [
                    "RunTaskError"
                  ],
                  "Next": "Fail"
                }
              ]
            },
            "Succeed": {
              "Type": "Succeed"
            },
            "Fail": {
              "Type": "Fail"
            }
          }
        }
      ]
    }
  }
}
```

```

        }
    },
    {
        "Comment": "Run GroupB state machine",
        "StartAt": "RunGroupB",
        "States": {
            "RunGroupA": {
                "Type": "RunTask",
                "Next": "Succeed",
                "TestGroup": "GroupB",
                "ResultVar": "GroupB_passed",
                "Catch": [
                    {
                        "ErrorEquals": [
                            "RunTaskError"
                        ],
                        "Next": "Fail"
                    }
                ]
            },
            "Succeed": {
                "Type": "Succeed"
            },
            "Fail": {
                "Type": "Fail"
            }
        }
    }
]
},
"CheckForErrors": {
    "Type": "Choice",
    "Default": "AddProductFeatures",
    "FallthroughOnError": true,
    "Choices": [
        {
            "Expression": "{{$.hasExecutionErrors}} == true",
            "Next": "Fail"
        }
    ]
},
"AddProductFeatures": {
    "Type": "AddProductFeatures",
    "Next": "Report",

```

```
    "Features": [
      {
        "Feature": "FeatureThatDependsOnGroupA",
        "Groups": [
          "GroupA"
        ],
        "IsRequired": true
      },
      {
        "Feature": "FeatureThatDependsOnGroupB",
        "Groups": [
          "GroupB"
        ],
        "IsRequired": true
      }
    ],
    "Report": {
      "Type": "Report",
      "Next": "Succeed",
      "Catch": [
        {
          "ErrorEquals": [
            "ReportError"
          ],
          "Next": "Fail"
        }
      ]
    },
    "Succeed": {
      "Type": "Succeed"
    },
    "Fail": {
      "Type": "Fail"
    }
  }
}
```

建立 IDT 測試案例可執行檔

您可以透過下列方式，在測試套件資料夾中建立和放置測試案例可執行檔：

- 對於使用 `test.json` 檔案中的引數或環境變數來決定要執行哪些測試的測試套件，您可以為整個測試套件建立單一測試案例可執行檔，或為測試套件中的每個測試群組建立測試可執行檔。
- 對於您想要根據指定命令執行特定測試的測試套件，您可以為測試套件中的每個測試案例建立一個測試案例可執行檔。

身為測試寫入器，您可以判斷哪種方法適合您使用案例，並相應地建構您的測試案例可執行檔。請確定您的 在每個`test.json`檔案中提供正確的測試案例可執行路徑，並且指定的可執行檔正確執行。

當所有裝置都準備好執行測試案例時，IDT 會讀取下列檔案：

- 所選測試案例`test.json`的 會決定要啟動的程序，以及要設定的環境變數。
- 測試套件`suite.json`的 決定要設定的環境變數。

IDT 會根據 `test.json` 檔案中指定的命令和引數啟動必要的測試可排除程序，並將必要的環境變數傳遞給程序。

使用 IDT 用戶端 SDK

IDT 用戶端 SDKs 可讓您使用 API 命令，簡化在測試可執行檔中撰寫測試邏輯的方式，您可以使用這些命令與 IDT 和待測裝置互動。IDT 目前提供下列 SDKs：

- 適用於 Python 的 IDT 用戶端 SDK
- IDT 用戶端 SDK for Go

這些 SDKs 位於 `<device-tester-extract-location>/sdks` 資料夾中。建立新的測試案例可執行檔時，您必須將要使用的開發套件複製到包含測試案例可執行檔的資料夾，並在程式碼中參考開發套件。本節提供可在測試案例可執行檔中使用的可用 API 命令的簡短描述。

本節內容

- [裝置互動](#)
- [IDT 互動](#)
- [主機互動](#)

裝置互動

下列命令可讓您與待測裝置通訊，而不必實作任何其他裝置互動和連線管理功能。

ExecuteOnDevice

允許測試套件在支援 SSH 或 Docker shell 連線的裝置上執行 Shell 命令。

CopyToDevice

允許測試套件將本機檔案從執行 IDT 的主機機器複製到支援 SSH 或 Docker shell 連線之裝置上的指定位置。

ReadFromDevice

允許測試套件從支援 UART 連線的裝置序列連接埠讀取。

Note

由於 IDT 不會管理從內容使用裝置存取資訊所建立之裝置的直接連線，因此建議您在測試案例可執行檔中使用這些裝置互動 API 命令。不過，如果這些命令不符合您的測試案例需求，則您可以從 IDT 內容擷取裝置存取資訊，並使用它從測試套件直接連線至裝置。

若要進行直接連線，請分別擷取 `device.connectivity` 以及待測裝置和資源裝置 `resource.devices.connectivity` 的欄位。如需使用 IDT 內容的詳細資訊，請參閱 [使用 IDT 內容](#)。

IDT 互動

下列命令可讓您的測試套件與 IDT 通訊。

PollForNotifications

允許測試套件檢查來自 IDT 的通知。

GetContextValue 和 GetContextString

允許測試套件從 IDT 內容擷取值。如需詳細資訊，請參閱 [使用 IDT 內容](#)。

SendResult

允許測試套件向 IDT 報告測試案例結果。此命令必須在測試套件中每個測試案例的結尾呼叫。

主機互動

下列命令可讓您的測試套件與主機通訊。

PollForNotifications

允許測試套件檢查來自 IDT 的通知。

GetContextValue 和 GetContextString

允許測試套件從 IDT 內容擷取值。如需詳細資訊，請參閱[使用 IDT 內容](#)。

ExecuteOnHost

允許測試套件在本機電腦上執行命令，並讓 IDT 管理測試案例可執行的生命週期。

啟用 IDT CLI 命令

`run-suite` 命令 IDT CLI 提供數個選項，可讓測試執行器自訂測試執行。若要允許測試執行器使用這些選項來執行自訂測試套件，請實作對 IDT CLI 的支援。如果您未實作支援，測試執行器仍然可以執行測試，但某些 CLI 選項將無法正常運作。為了提供理想的客戶體驗，建議您在 IDT CLI 中為 `run-suite` 命令實作下列引數的支援：

timeout-multiplier

指定大於 1.0 的值，該值將在執行測試時套用至所有逾時。

測試執行器可以使用此引數來增加想要執行的測試案例逾時。當測試執行器在其 `run-suite` 命令中指定此引數時，IDT 會使用它來計算 `IDT_TEST_TIMEOUT` 環境變數的值，並在 IDT 內容中設定 `config.timeoutMultiplier` 欄位。若要支援此引數，您必須執行下列動作：

- 讀取 `IDT_TEST_TIMEOUT` 環境變數以取得正確計算的逾時值，而不是直接使用 `test.json` 檔案的逾時值。
- 從 IDT 內容擷取 `config.timeoutMultiplier` 值，並將其套用至長時間執行的逾時。

如需因逾時事件而提早結束的詳細資訊，請參閱 [指定結束行為](#)。

stop-on-first-failure

指定如果 IDT 遇到失敗，應停止執行所有測試。

當測試執行器在其 `run-suite` 命令中指定此引數時，IDT 會在遇到失敗時立即停止執行測試。不過，如果測試案例平行執行，則這可能會導致非預期的結果。若要實作支援，請確定如果 IDT 遇到此事件，您的測試邏輯會指示所有執行中的測試案例停止、清除臨時資源，並向 IDT 報告測試結果。如需在故障時提早結束的詳細資訊，請參閱 [指定結束行為](#)。

group-id 和 test-id

指定 IDT 應僅執行選取的測試群組或測試案例。

測試執行器可以將這些引數與其 `run-suite` 命令搭配使用，以指定下列測試執行行為：

- 在指定的測試群組內執行所有測試。
- 從指定的測試群組內執行一系列測試。

若要支援這些引數，測試套件的狀態機器必須在狀態機器中包含一組特定的 `RunTask` 和 `Choice` 狀態。如果您不是使用自訂狀態機器，則預設的 IDT 狀態機器會包含所需的狀態，您不需要採取其他動作。不過，如果您使用的是自訂狀態機器，則使用 [狀態機器範例：執行使用者選取的測試群組](#) 做為範例，在狀態機器中新增所需的狀態。

如需 IDT CLI 命令的詳細資訊，請參閱 [偵錯並執行自訂測試套件](#)。

寫入事件日誌

測試執行時，您會將資料傳送至 `stderr` `stdout`，並將事件日誌和錯誤訊息寫入 主控台。如需主控台訊息格式的資訊，請參閱 [主控台訊息格式](#)。

當 IDT 完成執行測試套件時，此資訊也可用於位於 `<device-tester-extract-location>/results/<execution-id>/logs` 資料夾的 `test_manager.log` 檔案。

您可以設定每個測試案例，將日誌從其測試執行寫入至位於 `<device-tester-extract-location>/results/<execution-id>/logs` 資料夾 `<group-id>_<test-id>` 的檔案，包括來自待測裝置的日誌。若要執行此操作，請從 IDT 內容中擷取日誌檔案的路徑 `testData.logFilePath`，在該路徑建立檔案，然後寫入您想要的內容。IDT 會根據正在執行的測試案例自動更新路徑。如果您選擇不建立測試案例的日誌檔案，則不會為該測試案例產生檔案。

您也可以設定文字可執行檔，視需要在 `<device-tester-extract-location>/logs` 資料夾中建立其他日誌檔案。建議您為日誌檔案名稱指定唯一的字首，以免您的檔案遭到覆寫。

向 IDT 報告結果

IDT 會將測試結果寫入 `awsiotdevicetester_report.xml` 和 `suite-name_report.xml` 檔案。這些報告檔案位於 `<device-tester-extract-location>/results/<execution-id>/` 兩個報告都會擷取測試套件執行的結果。如需 IDT 用於這些報告的結構描述的詳細資訊，請參閱 [檢閱 IDT 測試結果和日誌](#)

若要填入 `suite-name_report.xml` 檔案的內容，您必須在測試執行完成之前，使用 `SendResult` 命令向 IDT 報告測試結果。如果 IDT 找不到測試的結果，則會發出測試案例的錯誤。下列 Python 摘錄顯示將測試結果傳送至 IDT 的命令：

```
request-variable = SendResultRequest(TestResult(result))  
client.send_result(request-variable)
```

如果您未透過 API 報告結果，IDT 會在測試成品資料夾中尋找測試結果。此資料夾的路徑會存放在 IDT 內容中 `testData.testArtifactsPath` 存檔的中。在此資料夾中，IDT 會使用第一個依字母順序排序的 XML 檔案，將其定位為測試結果。

如果您的測試邏輯產生 JUnit XML 結果，您可以將測試結果寫入成品資料夾中的 XML 檔案，以直接將結果提供給 IDT，而不是剖析結果，然後使用 API 將結果提交給 IDT。

如果您使用此方法，請確定您的測試邏輯準確摘要測試結果，並以與檔案相同的格式格式化結果 `suite-name_report.xml` 檔案。IDT 不會對您提供的資料執行任何驗證，但以下情況除外：

- IDT 會忽略 `testsuites` 標籤的所有屬性。反之，它會從其他報告的測試群組結果計算標籤屬性。
- 中至少必須存在一個 `testsuite` 標籤 `testsuites`。

由於 IDT 對所有測試案例使用相同的成品資料夾，而且不會在測試執行之間刪除結果檔案，因此如果 IDT 讀取不正確的檔案，此方法也可能導致錯誤報告。建議您在所有測試案例中使用與產生的 XML 結果檔案相同的名稱，以覆寫每個測試案例的結果，並確保 IDT 可以使用正確的結果。雖然您可以使用混合方法在測試套件中報告，也就是說，針對某些測試案例使用 XML 結果檔案，並透過 API 為其他人提交結果，但不建議使用此方法。

指定結束行為

將您的文字可執行檔設定為一律以結束碼 0 結束，即使測試案例報告失敗或錯誤結果。僅使用非零結束代碼來表示測試案例未執行，或測試案例可執行檔無法將任何結果傳達給 IDT。當 IDT 收到非零的結束碼時，它會標記測試案例發生錯誤，導致無法執行。

IDT 可能會請求或預期測試案例在下列事件完成之前停止執行。使用此資訊來設定您的測試案例可執行檔，以偵測測試案例中的每個事件：

Timeout (逾時)

當測試案例執行的時間超過 `test.json` 檔案中指定的逾時值時，便會發生。如果測試執行器使用 `timeout-multiplier` 引數指定逾時乘數，則 IDT 會使用乘數計算逾時值。

若要偵測此事件，請使用 `IDT_TEST_TIMEOUT` 環境變數。當測試執行器啟動測試時，IDT 會將 `IDT_TEST_TIMEOUT` 環境變數的值設定為計算的逾時值（以秒為單位），並將變數傳遞至測試案例可執行檔。您可以讀取變數值來設定適當的計時器。

中斷

當測試執行器中斷 IDT 時發生。例如，按 Ctrl+C。

由於終端機會將訊號傳播到所有子程序，因此您只需在測試案例中設定訊號處理常式，即可偵測中斷訊號。

或者，您可以定期輪詢 API，以檢查 `PollForNotifications API CancellationRequested` 回應中的布林值。當 IDT `CancellationRequested` 收到中斷訊號時，會將布林值設定為 `true`。

第一次失敗時停止

當與目前測試案例平行執行的測試案例失敗，且測試執行器使用 `stop-on-first-failure` 引數指定 IDT 應在遇到任何失敗時停止時，便會發生。

若要偵測此事件，您可以定期輪詢 API，以檢查 `PollForNotifications API CancellationRequested` 回應中的布林值。當 IDT 遇到故障，且設定為在第一次失敗時停止時，它會將 `CancellationRequested` 布林值設定為 `true`。

當這些事件發生時，IDT 會等待 5 分鐘，讓任何目前正在執行的測試案例完成執行。如果所有執行中的測試案例未在 5 分鐘內結束，IDT 會強制停止其每個程序。如果 IDT 在程序結束之前尚未收到測試結果，則會將測試案例標記為逾時。最佳實務是，您應該確保測試案例在遇到其中一個事件時執行下列動作：

1. 停止執行正常測試邏輯。
2. 清除任何臨時資源，例如待測裝置上的測試成品。
3. 向 IDT 報告測試結果，例如測試失敗或錯誤。
4. 結束。

使用 IDT 內容

當 IDT 執行測試套件時，測試套件可以存取一組資料，用於判斷每個測試的執行方式。此資料稱為 IDT 內容。例如，在 IDT 內容中，測試執行器在 `userdata.json` 檔案中提供的使用者資料組態可用於測試套件。

IDT 內容可視為唯讀 JSON 文件。測試套件可以使用物件、陣列、數字等標準 JSON 資料類型，從擷取資料並將資料寫入內容。

內容結構描述

IDT 內容使用下列格式：

```
{
  "config": {
    <config-json-content>
    "timeoutMultiplier": timeout-multiplier
  },
  "device": {
    <device-json-device-element>
  },
  "devicePool": {
    <device-json-pool-element>
  },
  "resource": {
    "devices": [
      {
        <resource-json-device-element>
        "name": "<resource-name>"
      }
    ]
  },
  "testData": {
    "awsCredentials": {
      "awsAccessKeyId": "<access-key-id>",
      "awsSecretAccessKey": "<secret-access-key>",
      "awsSessionToken": "<session-token>"
    },
    "logFilePath": "/path/to/log/file"
  },
  "userData": {
    <userdata-json-content>
  }
}
```

config

來自 [config.json 檔案](#) 的資訊。config 欄位也包含下列其他欄位：

`config.timeoutMultiplier`

測試套件所使用的任何逾時值的乘數。此值由 IDT CLI 的測試執行器指定。預設值為 1。

device

為測試執行選取之裝置的相關資訊。此資訊等同於所選裝置 [device.json 檔案中](#)的devices陣列元素。

devicePool

為測試執行選取之裝置集區的相關資訊。此資訊等同於所選裝置集區device.json檔案中定義的最上層裝置集區陣列元素。

resource

resource.json 檔案中資源裝置的相關資訊。

resource.devices

此資訊等同於 resource.json 檔案中定義的devices陣列。每個devices元素都包含下列額外欄位：

resource.device.name

資源裝置的名稱。此值設定為 test.json 檔案中的 requiredResource.name值。

testData.awsCredentials

測試用來連線至 AWS 雲端的 AWS 登入資料。此資訊是從 config.json 檔案取得。

testData.logFilePath

測試案例寫入日誌訊息的日誌檔案路徑。如果此檔案不存在，測試套件會建立此檔案。

userData

[userdata.json 檔案中](#)測試執行器提供的資訊。

存取內容中的資料

您可以使用 JSON 檔案的 JSONPath 標記法，以及使用 GetContextValue和 GetContextString APIs 的可執行文字來查詢內容。用於存取 IDT 內容的 JSONPath 字串語法會有所不同，如下所示：

- 在 suite.json和中test.json，您可以使用 `{{query}}`。也就是說，請勿使用根元素\$。來啟動表達式。
- 在中statemachine.json，您可以使用 `{{$.query}}`。
- 在 API 命令中，您可以使用 `query`或 `{{$.query}}`，視命令而定。如需詳細資訊，請參閱 SDKs 中的內嵌文件。

下表說明典型 JSONPath 表達式中的運算子：

Operator	Description
\$	The root element. Because the top-level context value for IDT is an object, you will typically use \$. to start your queries.
.childName	Accesses the child element with name childName from an object. If applied to an array, yields a new array with this operator applied to each element. The element name is case sensitive. For example, the query to access the awsRegion value in the config object is \$.config.awsRegion .
#####	Filters elements from an array, retrieving items beginning from the ## index and going up to the end index, both inclusive.
#index1# index2#...# indexN#	Filters elements from an array, retrieving items from only the specified indices.
##(expr)#	Filters elements from an array using the expr expression. This expression must evaluate to a boolean value.

若要建立篩選條件表達式，請使用下列語法：

```
<jsonpath> | <value> operator <jsonpath> | <value>
```

在此語法中：

- jsonpath 是使用標準 JSON 語法的 JSONPath。
- value 是使用標準 JSON 語法的任何自訂值。
- operator 是下列其中一個運算子：
 - < (小於)

- `<=` (小於或等於)
- `==` (等於)

如果表達式中的 JSONPath 或 值是陣列、布林值或物件值，則這是您可以唯一支援的二進位運算子。

- `>=` (大於或等於)
- `>` (大於)
- `=~` (規則表達式比對)。若要在篩選條件表達式中使用此運算子，表達式左側的 JSONPath 或值必須評估為字串，而右側必須是遵循 [RE2 語法](#) 的模式值。

您可以使用格式為 `{{query}}` 的 JSONPath 查詢，做為 `test.json` 檔案中 `args` 和 `environmentVariables` 欄位以及 `suite.json` 檔案中 `environmentVariables` 欄位內的預留位置字串。IDT 會執行內容查詢，並將查詢的評估值填入欄位。例如，在 `suite.json` 檔案中，您可以使用預留位置字串來指定每個測試案例變更的環境變數值，IDT 會將每個測試案例的正確值填入環境變數。不過，當您在 `test.json` 和 `suite.json` 檔案中使用預留位置字串時，下列考量適用於您的查詢：

- 您必須在所有小寫的查詢中每次出現 `devicePool` 金鑰。也就是說，請 `devicepool` 改用。
- 對於陣列，您只能使用字串陣列。此外，陣列使用非標準 `item1, item2, ..., itemN` 格式。如果陣列只包含一個元素，則會將其序列化為 `item`，使其與字串欄位無法區分。
- 您無法使用預留位置從內容擷取物件。

由於這些考量，我們建議您盡可能使用 API 來存取測試邏輯中的內容，而不是 `test.json` 和 `suite.json` 檔案中的預留位置字串。不過，在某些情況下，使用 JSONPath 預留位置來擷取單一字串以設定為環境變數可能會更方便。

設定測試執行器的設定

若要執行自訂測試套件，測試執行器必須根據他們想要執行的測試套件來設定其設定。根據位於 `<device-tester-extract-location>/configs/` 資料夾中的 JSON 組態檔案範本來指定設定。如有需要，測試執行器也必須設定 IDT 用來連線至 AWS 雲端的 AWS 登入資料。

身為測試寫入器，您需要設定這些檔案來[偵錯您的測試套件](#)。您必須提供測試執行器的說明，以便他們可以視需要設定下列設定來執行您的測試套件。

設定 device.json

device.json 檔案包含執行測試之裝置的相關資訊（例如 IP 地址、登入資訊、作業系統和 CPU 架構）。

測試執行器可以使用位於 *<device-tester-extract-location>/configs/* 資料夾中的下列範本 device.json 檔案提供此資訊。

```
[
  {
    "id": "<pool-id>",
    "sku": "<pool-sku>",
    "features": [
      {
        "name": "<feature-name>",
        "value": "<feature-value>",
        "configs": [
          {
            "name": "<config-name>",
            "value": "<config-value>"
          }
        ]
      }
    ],
    "devices": [
      {
        "id": "<device-id>",
        "connectivity": {
          "protocol": "ssh | uart | docker",
          // ssh
          "ip": "<ip-address>",
          "port": "<port-number>",
          "auth": {
            "method": "pki | password",
            "credentials": {
              "user": "<user-name>",
              // pki
              "privKeyPath": "/path/to/private/key",

              // password
              "password": "<password>",
            }
          }
        }
      }
    ]
  }
]
```

```

        // uart
        "serialPort": "<serial-port>",

        // docker
        "containerId": "<container-id>",
        "containerUser": "<container-user-name>",
    }
}
]
}
]

```

如下所述，包含值的所有欄位皆為必要：

id

使用者定義的英數字元 ID，可唯一識別裝置的集合，稱為「裝置集區」。屬於同一個集區的裝置必須有相同的硬體。當您執行測試套件，集區中的裝置會用來將工作負載平行化。多個裝置用來執行不同的測試。

sku

可唯一識別測試裝置的英數字元值。SKU 用於追蹤合格裝置。

Note

如果您想要在 AWS Partner Device Catalog 中列出您的電路板，您在此處指定的 SKU 必須符合您在列出程序中使用的 SKU。

features

選用。包含裝置支援功能的陣列。裝置功能是您測試套件中設定的使用者定義值。您必須提供測試執行器有關要包含在 device.json 檔案中的功能名稱和值的資訊。例如，如果您想要測試可做為其他裝置的 MQTT 伺服器的裝置，則可以設定測試邏輯來驗證名為 之功能的特定支援層級 MQTT_QOS。測試執行器提供此功能名稱，並將功能值設定為其裝置支援的 QOS 層級。您可以使用 devicePool.features 查詢從 [IDT 內容](#) 擷取提供的資訊，或使用 pool.features 查詢從 [狀態機器內容](#) 擷取提供的資訊。

features.name

功能的名稱。

`features.value`

支援的功能值。

`features.configs`

如有需要，此功能的組態設定。

`features.config.name`

組態設定的名稱。

`features.config.value`

支援的設定值。

`devices`

要測試之集區中的裝置陣列。至少需要一個裝置。

`devices.id`

使用者定義的唯一識別符，用於識別要測試的裝置。

`connectivity.protocol`

用來與此裝置通訊的通訊協定。集區中的每個裝置都必須使用相同的通訊協定。

目前，實體裝置ssh和 以及 uart Docker 容器唯一支援的值為 docker 和 。

`connectivity.ip`

要測試之裝置的 IP 位址。

只有當 `connectivity.protocol` 設為 ssh 時，才會套用此屬性。

`connectivity.port`

選用。用於 SSH 連線的連接埠號碼。

預設值為 22。

只有當 `connectivity.protocol` 設為 ssh 時，才會套用此屬性。

`connectivity.auth`

連線的驗證資訊。

只有當 `connectivity.protocol` 設為 `ssh` 時，才會套用此屬性。

`connectivity.auth.method`

用來透過指定的連線通訊協定存取裝置的驗證方法。

支援的值如下：

- `pki`
- `password`

`connectivity.auth.credentials`

用於驗證的登入資料。

`connectivity.auth.credentials.password`

用於登入要測試裝置的密碼。

只有當 `connectivity.auth.method` 設為 `password` 時，才會套用此值。

`connectivity.auth.credentials.privKeyPath`

用來登入待測裝置之私有金鑰的完整路徑。

只有當 `connectivity.auth.method` 設為 `pki` 時，才會套用此值。

`connectivity.auth.credentials.user`

登入要測試之裝置的使用者名稱。

`connectivity.serialPort`

選用。裝置所連接的序列連接埠。

只有當 `connectivity.protocol` 設為 `uart` 時，才會套用此屬性。

`connectivity.containerId`

要測試之 Docker 容器的容器 ID 或名稱。

只有當 `connectivity.protocol` 設為 `docker` 時，才會套用此屬性。

`connectivity.containerUser`

選用。容器內使用者的名稱。預設值是 Dockerfile 中提供的使用者。

預設值為 22。

只有當 `connectivity.protocol` 設為 `docker` 時，才會套用此屬性。

Note

若要檢查測試執行器是否為測試設定不正確的裝置連線，您可以從 `pool.Devices[0].Connectivity.Protocol` 狀態機器內容擷取，並將其與 `Choice` 狀態中的預期值進行比較。如果使用不正確的通訊協定，請使用 `LogMessage` 狀態列印訊息並轉換為 `Fail` 狀態。或者，您可以使用錯誤處理程式碼來報告不正確裝置類型的測試失敗。

(選用) 設定 `userdata.json`

`userdata.json` 檔案包含測試套件所需的任何其他資訊，但未在 `device.json` 檔案中指定。此檔案的格式取決於測試套件中定義的 [userdata_scheme.json 檔案](#)。如果您是測試寫入器，請務必將此資訊提供給將執行您所寫入測試套件的使用者。

(選用) 設定 `resource.json`

`resource.json` 檔案包含將用作資源裝置的任何裝置的相關資訊。資源裝置是測試待測裝置特定功能所需的裝置。例如，若要測試裝置的藍牙功能，您可以使用資源裝置來測試裝置是否可以成功連線到該裝置。資源裝置是選用的，您可以視需要要求任意數量的資源裝置。身為測試寫入器，您可以使用 [test.json 檔案](#) 來定義測試所需的資源裝置功能。然後，測試執行器會使用 `resource.json` 檔案來提供具有所需功能的資源裝置集區。請務必將此資訊提供給將執行您撰寫之測試套件的使用者。

測試執行器可以使用位於 `<device-tester-extract-location>/configs/` 資料夾中的下列範本 `resource.json` 檔案提供此資訊。

```
[
  {
    "id": "<pool-id>",
    "features": [
      {
        "name": "<feature-name>",
        "version": "<feature-value>",
        "jobSlots": <job-slots>
      }
    ]
  },
]
```

```

    "devices": [
      {
        "id": "<device-id>",
        "connectivity": {
          "protocol": "ssh | uart | docker",
          // ssh
          "ip": "<ip-address>",
          "port": <port-number>,
          "auth": {
            "method": "pki | password",
            "credentials": {
              "user": "<user-name>",
              // pki
              "privKeyPath": "/path/to/private/key",

              // password
              "password": "<password>",
            }
          },
          // uart
          "serialPort": "<serial-port>",

          // docker
          "containerId": "<container-id>",
          "containerUser": "<container-user-name>",
        }
      }
    ]
  }
}
]

```

如下所述，包含值的所有欄位皆為必要：

id

使用者定義的英數字元 ID，可唯一識別裝置的集合，稱為「裝置集區」。屬於同一個集區的裝置必須有相同的硬體。當您執行測試套件，集區中的裝置會用來將工作負載平行化。多個裝置用來執行不同的測試。

features

選用。包含裝置支援功能的陣列。此欄位所需的資訊是在測試套件的 [test.json 檔案中](#) 定義，並決定要執行哪些測試，以及如何執行這些測試。如果測試套件不需要任何功能，則不需要此欄位。

`features.name`

功能的名稱。

`features.version`

功能版本。

`features.jobSlots`

設定 以指示有多少測試可以同時使用裝置。預設值為 1。

`devices`

要測試之集區中的裝置陣列。至少需要一個裝置。

`devices.id`

使用者定義的唯一識別符，用於識別要測試的裝置。

`connectivity.protocol`

用來與此裝置通訊的通訊協定。集區中的每個裝置都必須使用相同的通訊協定。

目前，實體裝置ssh和 以及 uart Docker 容器唯一支援的值為 docker 和 。

`connectivity.ip`

要測試之裝置的 IP 位址。

只有當 `connectivity.protocol` 設為 ssh 時，才會套用此屬性。

`connectivity.port`

選用。用於 SSH 連線的連接埠號碼。

預設值為 22。

只有當 `connectivity.protocol` 設為 ssh 時，才會套用此屬性。

`connectivity.auth`

連線的驗證資訊。

只有當 `connectivity.protocol` 設為 ssh 時，才會套用此屬性。

`connectivity.auth.method`

用來透過指定的連線通訊協定存取裝置的驗證方法。

支援的值如下：

- `pki`
- `password`

`connectivity.auth.credentials`

用於驗證的登入資料。

`connectivity.auth.credentials.password`

用於登入要測試裝置的密碼。

只有當 `connectivity.auth.method` 設為 `password` 時，才會套用此值。

`connectivity.auth.credentials.privKeyPath`

用來登入待測裝置之私有金鑰的完整路徑。

只有當 `connectivity.auth.method` 設為 `pki` 時，才會套用此值。

`connectivity.auth.credentials.user`

登入要測試之裝置的使用者名稱。

`connectivity.serialPort`

選用。裝置所連接的序列連接埠。

只有當 `connectivity.protocol` 設為 `uart` 時，才會套用此屬性。

`connectivity.containerId`

要測試之 Docker 容器的容器 ID 或名稱。

只有當 `connectivity.protocol` 設為 `docker` 時，才會套用此屬性。

`connectivity.containerUser`

選用。容器內使用者的名稱。預設值是 Dockerfile 中提供的使用者。

預設值為 22。

只有當 `connectivity.protocol` 設為 `docker` 時，才會套用此屬性。

(選用) 設定 config.json

`config.json` 檔案包含 IDT 的組態資訊。一般而言，測試執行器不需要修改此檔案，除了提供其 IDT AWS 使用者登入資料，以及選擇性 AWS 的區域。如果 AWS 提供具有必要許可的登入資料 AWS IoT，則 Device Tester 會收集用量指標並將其提交至其中 AWS。這是選擇加入功能，用於改善 IDT 功能。如需詳細資訊，請參閱[IDT 用量指標](#)。

測試執行器可以透過下列其中一種方式設定其 AWS 登入資料：

- 憑證檔案

IDT 會使用與 AWS CLI 相同的登入資料檔案。如需詳細資訊，請參閱[組態與登入資料檔案](#)。

登入資料檔案的位置會有所不同，取決於您使用的作業系統：

- macOS, Linux: `~/.aws/credentials`
- Windows : `C:\Users\UserName\.aws\credentials`
- 環境變數

環境變數是由作業系統維護且由系統命令使用的變數。在 SSH 工作階段期間定義的變數在該工作階段關閉後無法使用。IDT 可以使用 `AWS_ACCESS_KEY_ID` 和 `AWS_SECRET_ACCESS_KEY` 環境變數來存放 AWS 登入資料

若要在 Linux、macOS 或 Unix 上設定這些變數，請使用 `export`：

```
export AWS_ACCESS_KEY_ID=<your_access_key_id>
export AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

若要在 Windows 上設定這些變數，請使用 `set`：

```
set AWS_ACCESS_KEY_ID=<your_access_key_id>
set AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

若要設定 IDT 的 AWS 登入資料，測試執行器會編輯位於 `<device-tester-extract-location>/configs/` 資料夾之 `config.json` 檔案中的 `auth` 區段。

```
{
```

```
"log": {
  "location": "logs"
},
"configFiles": {
  "root": "configs",
  "device": "configs/device.json"
},
"testPath": "tests",
"reportPath": "results",
"awsRegion": "<region>",
"auth": {
  "method": "file | environment",
  "credentials": {
    "profile": "<profile-name>"
  }
}
}
```

如下所述，包含值的所有欄位皆為必要：

 Note

此檔案中的所有路徑都是相對於 *<device-tester-extract-location>* 定義。

log.location

<device-tester-extract-location> 中日誌資料夾的路徑。

configFiles.root

包含組態檔案之資料夾的路徑。

configFiles.device

device.json 檔案的路徑。

testPath

包含測試套件的資料夾路徑。

reportPath

IDT 執行測試套件後，將包含測試結果的資料夾路徑。

awsRegion

選用。測試套件將使用 AWS 的區域。如果未設定，則測試套件將使用每個測試套件中指定的預設區域。

auth.method

IDT 用來擷取 AWS 登入資料的方法。支援的值是從登入資料檔案file擷取登入資料，以及使用環境變數environment擷取登入資料。

auth.credentials.profile

從登入資料檔案使用的登入資料設定檔。只有當 auth.method 設為 file 時，才會套用此屬性。

偵錯並執行自訂測試套件

設定[必要的組態](#)後，IDT 可以執行您的測試套件。完整測試套件的執行時間取決於硬體和測試套件的組成。如需參考，在 Raspberry Pi 3B AWS IoT Greengrass 上完成完整資格測試套件大約需要 30 分鐘。3B

撰寫測試套件時，您可以使用 IDT 在偵錯模式下執行測試套件，在執行程式碼之前先檢查程式碼，或將其提供給測試執行器。

在偵錯模式下執行 IDT

由於測試套件依賴 IDT 與裝置互動、提供內容並接收結果，因此您無法在沒有任何 IDT 互動的情況下，在 IDE 中對測試套件進行偵錯。若要這樣做，IDT CLI 會提供 debug-test-suite命令，可讓您在偵錯模式下執行 IDT。執行下列命令以檢視 的可用選項debug-test-suite：

```
devicetester_[linux | mac | win_x86-64] debug-test-suite -h
```

當您在偵錯模式下執行 IDT 時，IDT 實際上不會啟動測試套件或執行狀態機器；而是與您的 IDE 互動，以回應從 IDE 中執行的測試套件發出的請求，並將日誌列印到主控台。IDT 不會逾時，並等待結束，直到手動中斷為止。在偵錯模式中，IDT 也不會執行狀態機器，也不會產生任何報告檔案。若要偵錯您的測試套件，您必須使用 IDE 來提供 IDT 通常從組態 JSON 檔案取得的一些資訊。請務必提供下列資訊：

- 每個測試的環境變數和引數。IDT 不會從 test.json或 讀取此資訊suite.json。
- 選取資源裝置的引數。IDT 不會從 讀取此資訊test.json。

若要偵錯您的測試套件，請完成下列步驟：

1. 建立執行測試套件所需的設定組態檔案。例如，如果您的測試套件需要 `device.json`、`resource.json` 和 `user data.json`，請務必視需要設定所有測試套件。
2. 執行下列命令，將 IDT 置於偵錯模式，然後選取執行測試所需的任何裝置。

```
devicetester_[linux | mac | win_x86-64] debug-test-suite [options]
```

執行此命令後，IDT 會等待測試套件的請求，然後回應這些請求。IDT 也會產生 IDT 用戶端 SDK 案例程序所需的環境變數。

3. 在 IDE 中，使用 `run` 或 `debug` 組態來執行下列動作：
 - a. 設定 IDT 產生環境變數的值。
 - b. 設定您在 `test.json` 和 `suite.json` 檔案中指定的任何環境變數或引數的值。
 - c. 視需要設定中斷點。
4. 在 IDE 中執行測試套件。

您可以視需要多次偵錯並重新執行測試套件。IDT 不會在偵錯模式下逾時。

5. 完成偵錯後，請中斷 IDT 以結束偵錯模式。

用於執行測試的 IDT CLI 命令

下節說明 IDT CLI 命令：

IDT v4.0.0

`help`

列出所指定命令的相關資訊。

`list-groups`

列出指定測試套件中的群組。

`list-suites`

列出可用的測試套件。

list-supported-products

列出目前 IDT 版本可用的 IDT AWS IoT Greengrass 版本和 AWS IoT Greengrass 資格測試套件版本。

list-test-cases

列出特定測試群組中的測試案例。支援下列選項：

- `group-id`。要搜尋的測試群組。此選項為必要選項，且必須指定單一群組。

run-suite

在裝置集區上執行測試套件。以下是一些常用的選項：

- `suite-id`。要執行的測試套件版本。如果未指定，IDT 會使用 `tests` 資料夾中的最新版本。
- `group-id`。要執行的測試群組，以逗號分隔的清單。如果未指定，IDT 會執行測試套件中的所有測試群組。
- `test-id`。要執行的測試案例，以逗號分隔的清單。指定時，`group-id` 必須指定單一群組。
- `pool-id`。要測試的裝置集區。如果測試執行器在您的 `device.json` 檔案中定義了多個裝置集區，則必須指定集區。
- `timeout-multiplier`。設定 IDT 修改具有使用者定義乘數之測試的 `test.json` 檔案中指定的測試執行逾時。
- `stop-on-first-failure`。設定 IDT 在第一次失敗時停止執行。此選項應與 `group-id` 搭配使用以偵錯指定的測試群組。
- `userdata`。設定 檔案，其中包含執行測試套件所需的使用者資料資訊。只有在測試套件的 `suite.json` 檔案中 `userdataRequired` 設定為 `true` 時，才需要這樣做。

如需 `run-suite` 選項的詳細資訊，請使用下列 `help` 選項：

```
devicetester_[linux | mac | win_x86-64] run-suite -h
```

debug-test-suite

在除錯模式下執行測試套件。如需詳細資訊，請參閱[在偵錯模式下執行 IDT](#)。

檢閱 IDT 測試結果和日誌

本節說明 IDT 產生主控台日誌和測試報告的格式。

主控台訊息格式

AWS IoT Device Tester 會在啟動測試套件時，使用標準格式將訊息列印至主控台。以下摘錄顯示 IDT 產生的主控台訊息範例。

```
time="2000-01-02T03:04:05-07:00" level=info msg=Using suite: MyTestSuite_1.0.0
executionId=9a52f362-1227-11eb-86c9-8c8590419f30
```

大多數主控台訊息包含下列欄位：

time

已記錄事件的完整 ISO 8601 時間戳記。

level

記錄事件的訊息層級。一般而言，記錄的訊息層級是 info、warn 或 error。如果 IDT 遇到導致它提早結束的預期事件，則會發出 fatal 或 panic 訊息。

msg

記錄的訊息。

executionId

目前 IDT 程序的唯一 ID 字串。此 ID 用於區分個別的 IDT 執行。

從測試套件產生的主控台訊息提供有關待測裝置以及 IDT 執行的測試套件、測試群組和測試案例的其他資訊。以下摘錄顯示從測試套件產生的主控台訊息範例。

```
time="2000-01-02T03:04:05-07:00" level=info msg=Hello world! suiteId=MyTestSuite
groupId=myTestGroup testCaseId=myTestCase deviceId=my-device
executionId=9a52f362-1227-11eb-86c9-8c8590419f30
```

主控台訊息的測試套件特定部分包含下列欄位：

suiteId

目前正在執行的測試套件名稱。

groupId

目前正在執行的測試群組 ID。

testCaseId

目前執行的測試案例 ID。

deviceId

目前測試案例使用的待測裝置 ID。

若要在 IDT 完成執行測試時將測試摘要列印至主控台，您必須在狀態機器中包含 [Report 狀態](#)。測試摘要包含測試套件的相關資訊、執行的每個群組的測試結果，以及產生的日誌和報告檔案的位置。下列範例顯示測試摘要訊息。

```
===== Test Summary =====
Execution Time:      5m00s
Tests Completed:     4
Tests Passed:        3
Tests Failed:        1
Tests Skipped:       0
-----
Test Groups:
  GroupA:            PASSED
  GroupB:            FAILED
-----
Failed Tests:
  Group Name: GroupB
    Test Name: TestB1
      Reason: Something bad happened
-----
Path to IoT Device Tester Report: /path/to/awsiotdevicetester_report.xml
Path to Test Execution Logs: /path/to/logs
Path to Aggregated JUnit Report: /path/to/MyTestSuite_Report.xml
```

AWS IoT Device Tester 報告結構描述

awsiotdevicetester_report.xml 是已簽署的報告，其中包含下列資訊：

- IDT 版本。
- 測試套件版本。
- 用於簽署報告的報告簽章和金鑰。
- device.json 檔案中指定的裝置 SKU 和裝置集區名稱。

- 已測試的產品版本和裝置功能。
- 測試結果的彙總摘要。此資訊與 *suite-name_report.xml* 檔案中包含的資訊相同。

```
<apnreport>
  <awsiotdevicetesterversion>idt-version</awsiotdevicetesterversion>
  <testsuiteversion>test-suite-version</testsuiteversion>
  <signature>signature</signature>
  <keyname>keyname</keyname>
  <session>
    <testsession>execution-id</testsession>
    <starttime>start-time</starttime>
    <endtime>end-time</endtime>
  </session>
  <awsproduct>
    <name>product-name</name>
    <version>product-version</version>
    <features>
      <feature name="<feature-name>" value="supported | not-supported | <feature-value>" type="optional | required"/>
    </features>
  </awsproduct>
  <device>
    <sku>device-sku</sku>
    <name>device-name</name>
    <features>
      <feature name="<feature-name>" value="<feature-value>"/>
    </features>
    <executionMethod>ssh | uart | docker</executionMethod>
  </device>
  <devenvironment>
    <os name="<os-name>"/>
  </devenvironment>
  <report>
    <suite-name-report-content>
  </report>
</apnreport>
```

awsiotdevicetester_report.xml 檔案包含 <awsproduct> 標籤，其中包含關於受測產品和經過一系列測試驗證後之產品功能的資訊。

<awsproduct> 標籤中使用的屬性

name

受測產品名稱。

version

受測產品版本。

features

驗證的功能。測試套件required需要標記為 的功能，才能驗證裝置。以下片段顯示此資訊如何出現在 awsiotdevicetester_report.xml 檔案中。

```
<feature name="ssh" value="supported" type="required"></feature>
```

驗證optional不需要標記為 的功能。以下程式碼片段顯示選用功能。

```
<feature name="hsi" value="supported" type="optional"></feature>

<feature name="mqtt" value="not-supported" type="optional"></feature>
```

測試套件報告結構描述

*suite-name*_Result.xml 報告採用 [JUnit XML 格式](#)。您可以將它整合到持續整合和部署平台，例如 [Jenkins](#)、[Bamboo](#) 等。報告包含測試結果的彙總摘要。

```
<testsuites name="<suite-name> results" time="<run-duration>" tests="<number-of-test>"
failures="<number-of-tests>" skipped="<number-of-tests>" errors="<number-of-tests>"
disabled="0">
  <testsuite name="<test-group-id>" package="" tests="<number-of-tests>"
failures="<number-of-tests>" skipped="<number-of-tests>" errors="<number-of-tests>"
disabled="0">
    <!--success-->
    <testcase classname="<classname>" name="<name>" time="<run-duration>"/>
    <!--failure-->
    <testcase classname="<classname>" name="<name>" time="<run-duration>">
      <failure type="<failure-type>">
        reason
      </failure>
    </testcase>
    <!--skipped-->
```

```
<testcase classname="<classname>" name="<name>" time="<run-duration>">
  <skipped>
    reason
  </skipped>
</testcase>
<!--error-->
<testcase classname="<classname>" name="<name>" time="<run-duration>">
  <error>
    reason
  </error>
</testcase>
</testsuite>
</testsuites>
```

awsiotdevicetester_report.xml 或 中的報告區段會 *suite-name*_report.xml 列出已執行的測試和結果。

第一個 XML 標籤 <testsuites> 包含測試執行的摘要。例如：

```
<testsuites name="MyTestSuite results" time="2299" tests="28" failures="0" errors="0"
  disabled="0">
```

<testsuites> 標籤中使用的屬性

name

測試套件的名稱。

time

執行測試套件所需的時間，以秒為單位。

tests

執行的測試次數。

failures

已執行但未通過的測試次數。

errors

IDT 無法執行的測試次數。

disabled

此屬性未使用，可忽略。

如果測試發生失敗或錯誤，您可以檢閱 `<testsuites>` XML 標籤來識別失敗的測試。`<testsuites>` 標籤內的 `<testsuite>` XML 標籤會顯示測試群組的測試結果摘要。例如：

```
<testsuite name="combination" package="" tests="1" failures="0" time="161" disabled="0" errors="0" skipped="0">
```

其格式類似於 `<testsuites>` 標籤，但有不使用且可忽略的 `skipped` 屬性。在每個 `<testsuite>` XML 標籤內，測試群組每個執行的測試都有 `<testcase>` 標籤。例如：

```
<testcase classname="Security Test" name="IP Change Tests" attempts="1"></testcase>>
```

`<testcase>` 標籤中使用的屬性

`name`

測試的名稱。

`attempts`

IDT 執行測試案例的次數。

當測試案例失敗或發生錯誤時，系統就會將 `<failure>` 或 `<error>` 標籤新增至 `<testcase>` 標籤，其中附有相關資訊以利故障診斷。例如：

```
<testcase classname="mcu.Full_MQTT" name="MQTT_TestCase" attempts="1">
  <failure type="Failure">Reason for the test failure</failure>
  <error>Reason for the test execution error</error>
</testcase>
```

IDT 用量指標

如果您提供具有必要許可的 AWS 登入資料，AWS IoT Device Tester 會收集用量指標並將其提交至 AWS。這是選擇加入功能，用於改善 IDT 功能。IDT 會收集下列資訊：

- 用來執行 IDT 的 AWS 帳戶 ID
- 用於執行測試的 IDT CLI 命令
- 執行的測試套件
- *`<device-tester-extract-location>`* 資料夾的測試套件
- 在裝置集區中設定的裝置數量

- 測試案例名稱和執行時間
- 測試結果資訊，例如測試是否通過、失敗、發生錯誤或被略過
- 測試的產品功能
- IDT 結束行為，例如非預期或提早結束

IDT 傳送的所有資訊也會記錄到 `<device-tester-extract-location>/results/<execution-id>/` 資料夾中 `metrics.log` 的檔案。您可以檢視日誌檔案，以查看在測試執行期間收集的資訊。只有在您選擇收集用量指標時，才會產生此檔案。

若要停用指標集合，您不需要採取其他動作。只要不要存放您的 AWS 登入資料，而且如果您有存放的 AWS 登入資料，請勿設定 `config.json` 檔案來存取它們。

設定您的 AWS 登入資料

如果您還沒有 AWS 帳戶，則必須[建立一個](#)。如果您已經有 AWS 帳戶，您只需要為您的帳戶[設定必要的許可](#)，以允許 IDT AWS 代表您將用量指標傳送至。

步驟 1：建立 AWS 帳戶

在此步驟中，建立和設定 AWS 帳戶。如果您已有 AWS 帳戶，請跳至 [the section called “步驟 2：設定 IDT 的許可”](#)。

註冊 AWS 帳戶

如果您沒有 AWS 帳戶，請完成下列步驟來建立一個。

註冊 AWS 帳戶

1. 開啟 <https://portal.aws.amazon.com/billing/signup>。
2. 請遵循線上指示進行。

部分註冊程序需接收來電或簡訊，並在電話鍵盤輸入驗證碼。

當您註冊時 AWS 帳戶，AWS 帳戶根使用者會建立。根使用者有權存取該帳戶中的所有 AWS 服務和資源。作為安全最佳實務，請將管理存取權指派給使用者，並且僅使用根使用者來執行[需要根使用者存取權的任務](#)。

AWS 會在註冊程序完成後傳送確認電子郵件給您。您可以隨時登錄 <https://aws.amazon.com/> 並選擇我的帳戶，以檢視您目前的帳戶活動並管理帳戶。

建立具有管理存取權的使用者

註冊後 AWS 帳戶，請保護 AWS 帳戶根使用者、啟用 AWS IAM Identity Center 和建立管理使用者，以免將根使用者用於日常任務。

保護您的 AWS 帳戶根使用者

1. 選擇根使用者並輸入 AWS 帳戶 您的電子郵件地址，以帳戶擁有者 [AWS 管理主控台](#) 身分登入。在下一頁中，輸入您的密碼。

如需使用根使用者登入的說明，請參閱 AWS 登入 使用者指南中的 [以根使用者身分登入](#)。

2. 若要在您的根使用者帳戶上啟用多重要素驗證 (MFA)。

如需說明，請參閱《IAM 使用者指南》中的 [為您的 AWS 帳戶 根使用者（主控台）啟用虛擬 MFA 裝置](#)。

建立具有管理存取權的使用者

1. 啟用 IAM Identity Center。

如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的 [啟用 AWS IAM Identity Center](#)。

2. 在 IAM Identity Center 中，將管理存取權授予使用者。

如需使用 IAM Identity Center 目錄 做為身分來源的教學課程，請參閱 AWS IAM Identity Center 《使用者指南》中的 [使用預設值設定使用者存取 IAM Identity Center 目錄](#)。

以具有管理存取權的使用者身分登入

- 若要使用您的 IAM Identity Center 使用者簽署，請使用建立 IAM Identity Center 使用者時傳送至您電子郵件地址的簽署 URL。

如需使用 IAM Identity Center 使用者登入的說明，請參閱 AWS 登入 《使用者指南》中的 [登入 AWS 存取入口網站](#)。

指派存取權給其他使用者

1. 在 IAM Identity Center 中，建立一個許可集來遵循套用最低權限的最佳實務。

如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的 [建立許可集](#)。

2. 將使用者指派至群組，然後對該群組指派單一登入存取權。

如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的[新增群組](#)。

步驟 2：設定 IDT 的許可

在此步驟中，設定 IDT 用來執行測試和收集 IDT 用量資料的許可。您可以使用 AWS 管理主控台 或 AWS Command Line Interface (AWS CLI) 來建立 IAM 政策和 IDT 的使用者，然後將政策連接到使用者。

- [設定 IDT \(主控台\) 的許可](#)
- [步驟 2：設定 IDT \(AWS CLI\) 的許可](#)

設定 IDT (主控台) 的許可

請依照下列步驟使用主控台來設定 IDT for AWS IoT Greengrass 的許可。

1. 登入 [IAM 主控台](#)。
2. 建立客戶受管政策，該政策授與建立具有特定許可之角色的許可。
 - a. 在導覽窗格中，選擇政策，然後選擇建立政策。
 - b. 在 JSON 標籤上，用以下政策取代預留位置內容。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot-device-tester:SendMetrics"
      ],
      "Resource": "*"
    }
  ]
}
```

- c. 選擇下一步：標籤。

- d. 選擇下一步：檢閱。
 - e. 對於名稱，輸入 **IDTUsageMetricsIAMPermissions**。在 Summary (摘要) 下，檢閱您的政策所授與的許可。
 - f. 選擇建立政策。
3. 建立 IAM 使用者並將許可連接到使用者。
- a. 建立 IAM 使用者。請遵循《IAM 使用者指南》中[建立 IAM 使用者（主控台）](#)的步驟 1 到 5。如果您已建立 IAM 使用者，請跳至下一個步驟。
 - b. 將許可連接至您的 IAM 使用者：
 - i. 在設定許可頁面上，選擇直接連接現有政策。
 - ii. 搜尋您在上一個步驟中建立的 IDTUsageMetricsIAMPermissions 政策。選取核取方塊。
 - c. 選擇下一步：標籤。
 - d. 選擇 Next: Review (下一步：檢閱) 以檢視選擇的摘要。
 - e. 選擇 Create user (建立使用者)。
 - f. 若要檢視使用者的存取金鑰 (存取金鑰 ID 和私密存取金鑰)，請選擇密碼和存取金鑰旁的 Show (顯示)。若要儲存存取金鑰，請選擇 Download.csv，並將檔案儲存到安全的位置。您稍後會使用此資訊來設定您的 AWS 登入資料檔案。

步驟 2：設定 IDT (AWS CLI) 的許可

請依照下列步驟使用 AWS CLI 來設定 IDT for 的許可 AWS IoT Greengrass。如果您已在主控台中設定許可，請跳至[the section called “設定您的裝置以執行 IDT 測試”](#)或[the section called “選用：設定您的 Docker 容器”](#)。

1. 如果尚未安裝，請在電腦上安裝和設定 AWS CLI。請遵循AWS Command Line Interface 《使用者指南》中的[安裝 AWS CLI](#) 中的步驟。

Note

AWS CLI 是開放原始碼工具，可用來與命令列 Shell 中的 AWS 服務互動。

2. 建立下列客戶受管政策，授予管理 IDT 和 AWS IoT Greengrass 角色的許可。

Linux, macOS, or Unix

```
aws iam create-policy --policy-name IDTUsageMetricsIAMPermissions --policy-  
document '{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": [  
        "iot-device-tester:SendMetrics"  
      ],  
      "Resource": "*"   
    }  
  ]  
}'
```

Windows command prompt

```
aws iam create-policy --policy-name IDTUsageMetricsIAMPermissions --policy-  
document '{\"Version\": \"2012-10-17\",  
  \"Statement\": [{\"Effect\": \"Allow\", \"Action\": [\"iot-device-  
tester:SendMetrics\"], \"Resource\": \"*\"]}]'
```

Note

此步驟包含 Windows 命令提示字元範例，因為它使用的 JSON 語法與 Linux、macOS 或 Unix 終端機命令不同。

3. 建立 IAM 使用者並連接 IDT for 所需的許可 AWS IoT Greengrass。

a. 建立 IAM 使用者。

```
aws iam create-user --user-name user-name
```

b. 將您建立IDTUsageMetricsIAMPermissions的政策連接至 IAM 使用者。將 *user-name* 取代為您的 IAM 使用者名稱，並將 命令中的 *<account-id>* 取代為您的 ID AWS 帳戶。

```
aws iam attach-user-policy --user-name user-name --policy-arn  
arn:aws:iam::<account-id>:policy/IDTGreengrassIAMPermissions
```

4. 為使用者建立私密存取金鑰。

```
aws iam create-access-key --user-name user-name
```

將輸出儲存在安全的位置。您稍後會使用此資訊來設定您的 AWS 登入資料檔案。

提供 AWS 登入資料給 IDT

若要允許 IDT 存取您的 AWS 登入資料並提交指標 AWS，請執行下列動作：

1. 將 IAM 使用者的 AWS 登入資料儲存為環境變數或儲存在登入資料檔案中：

a. 若要使用環境變數，請執行下列命令：

```
AWS_ACCESS_KEY_ID=access-key  
AWS_SECRET_ACCESS_KEY=secret-access-key
```

b. 若要使用登入資料檔案，請將下列資訊新增至 `.aws/credentials` file：

```
[profile-name]  
aws_access_key_id=access-key  
aws_secret_access_key=secret-access-key
```

2. 設定 `config.json` 檔案的 `auth` 區段。如需詳細資訊，請參閱 [\(選用\) 設定 config.json](#)。

IDT 故障 AWS IoT Greengrass 診斷

IDT for 會根據錯誤類型將這些錯誤 AWS IoT Greengrass 寫入各個位置。錯誤會寫入主控台、日誌檔和測試報告。

錯誤代碼

下表列出由 IDT for AWS IoT Greengrass 產生的錯誤代碼。

錯誤碼	錯誤代碼名稱	可能的根本原因	故障診斷
101	InternalError	發生內部錯誤。	檢查 <code><device-tester-extract-location></code> / results 目錄下的日誌。如果您無法偵錯問題，請聯絡 AWS 開發人員支援 。
102	TimeoutError	無法在限定的時間範圍內完成測試。可能發生這種情況的原因： - 測試機器和裝置之間的網路連線緩慢 (例如，若您使用的是 VPN 網路)。 - 緩慢的網路會延遲裝置與雲端之間的通訊。 - 錯誤修改了測試組態檔案 (test.json) 中的 timeout 欄位。	- 檢查網路連線和速度。 - 確認您未在 /test 目錄下修改任何檔案。 - 嘗試利用 "--group-id" 旗標手動執行故障測試群組。 - 嘗試延長測試逾時來執行測試套件。如需詳細資訊，請參閱逾時錯誤。
103	PlatformNotSupport Error	在 device.json 中指定了不正確的作業系統/架構組合。	變更組態為其中一項支援的組合： - Linux , x86_64 - Linux , ARMv6l

錯誤碼	錯誤代碼名稱	可能的根本原因	故障診斷
			Linux , ARMv7l • Linux , AArch64 • Ubuntu , x86_64 • OpenWRT , ARMv7l • OpenWRT , AArch64 如需詳細資訊，請參閱設定 device.json。

錯誤碼	錯誤代碼名稱	可能的根本原因	故障診斷
104	VersionNotSupportError	您使用的 IDT 版本不支援 AWS IoT Greengrass 核心軟體版本。	使用 <code>device_tester_bin version</code> 命令來尋找 AWS IoT Greengrass 核心軟體的支援版本。例如，如果您使用的是 macOS，請使用 <code>./devicetester_mac_x86_64 version</code>。 若要尋找您正在使用 AWS IoT Greengrass 的核心軟體版本： - 如果您使用預先安裝的 AWS IoT Greengrass Core 軟體執行測試，請使用 SSH 連線到您的 AWS IoT Greengrass 核心裝置並執行 <code><path-to-preinstalled-green-grass-location> /greengrass/ggc/core/greengrassd --version</code> - 如果您使用不同版本的 AWS IoT Greengrass Core 軟體執行測試，請前往 <code>devicetester_green</code>

錯誤碼	錯誤代碼名稱	可能的根本原因	故障診斷
			grass_ <os>/products /greengra ss/gcc 目 錄。Core AWS IoT Greengrass 軟體版 本是 .zip 檔案名稱 的一部分。 您可以測試不同 版本的 AWS IoT Greengrass 核心軟 體。如需詳細資訊， 請參閱入門 AWS IoT Greengrass。

錯誤碼	錯誤代碼名稱	可能的根本原因	故障診斷
105	LanguageNotSupport Error	IDT 僅支援 Python for AWS IoT Greengrass 程式庫和 SDKs。	請確定： • <code>devicetester_greengrass_<os>/products/greengrass/ggsdk</code> 下的軟體開發套件是 Python 軟體開發套件。 • <code>devicetester_greengrass_<os>/tests/GGQ_1.0.0/suite/resources/runtimefarm/bin</code> 下的 <code>bin</code> 資料夾內容未變更。

錯誤碼	錯誤代碼名稱	可能的根本原因	故障診斷
106	ValidationError	device.json 或 config.json 中的某些欄位無效。	請查看報告中錯誤代碼右側的錯誤訊息。 - 裝置的驗證類型無效：請指定正確的方法來連接到您的裝置。如需詳細資訊，請參閱the section called “設定 device.json”。 - 無效的私密金鑰路徑：請指定正確的私密金鑰路徑。如需詳細資訊，請參閱設定 device.js on。 - 無效 AWS 區域：AWS 區域 在您的 config.json 檔案中指定有效的。如需詳細資訊，請參閱AWS 服務端點。 - AWS 登入資料：在您的測試機器上設定有效的 AWS 登入資料（使用環境變數或 credentials 檔案）。確認 auth 欄位已正確設定。如需詳細

錯誤碼	錯誤代碼名稱	可能的根本原因	故障診斷
			資訊，請參閱the section called “建立和設定 AWS 帳戶”。 無效的 HSM 輸入：檢查在 device.json 中的 p11Provider、privateKeyLabel、slotLabel、slotUserPin 和 openSSLEngine 欄位。
107	SSHConnectionFailed	測試機器無法連接到設定的裝置。	確認 device.json 檔案中的以下欄位正確： - ip - user - privKeyPath - password 如需詳細資訊，請參閱設定 device.json。

錯誤碼	錯誤代碼名稱	可能的根本原因	故障診斷
108	RunCommandError	測試無法在待測裝置上執行命令。	確認允許 <code>device.json</code> 中的設定使用者進行根存取。 執行具有根存取的命令時，某些裝置必須輸入密碼。確定允許在沒有密碼的情況下進行根存取。如需更多詳細資訊，請參閱您的裝置文件。 在裝置上嘗試手動執行失敗的命令，以查看是否發生錯誤。
109	PermissionDeniedError	無根存取。	在裝置上為設定的使用者設定根存取。
110	CreateFileError	無法建立檔案。	檢查裝置的磁碟空間和目錄許可。
111	CreateDirError	無法建立目錄。	檢查裝置的磁碟空間和目錄許可。
112	InvalidPathError	Core AWS IoT Greengrass 軟體的路徑不正確。	請確認錯誤訊息中的路徑有效。請勿編輯 <code>devicetester_greengrass_<os></code> 目錄下的任何檔案。
113	InvalidFileError	檔案無效。	確認錯誤訊息中的檔案有效。

錯誤碼	錯誤代碼名稱	可能的根本原因	故障診斷
114	ReadFileError	無法讀取指定的檔案。 。	請確認下列內容： - 檔案許可正確。 <code>limits.config</code> 允許開啟足夠的檔案。 - 錯誤訊息中指定的檔案存在且有效。 如果您是在 MacOS 上測試，請提高開啟檔案限制。預設限制為 256，這對測試而言已足夠。

錯誤碼	錯誤代碼名稱	可能的根本原因	故障診斷
115	FileNotFoundError	找不到需要的檔案。	請確認下列內容： • 壓縮的 Greengrass 檔案存在於 <code>devicetes ter_green grass_ <os>/ products/ greengrass/ ggc</code> 下。您可以從 AWS IoT Greengrass 核心 AWS IoT Greengrass 軟體下載頁面 下載核心 tar 檔案。 • 軟體開發套件存在於 <code>devicetes ter_green grass_ <os>/ products/ greengrass/ ggsdk</code> 下。 • <code>devicetes ter_green grass_ <os>/ tests</code> 下的檔案未修改。

錯誤碼	錯誤代碼名稱	可能的根本原因	故障診斷
116	OpenFileFailed	無法開啟指定的檔案。 。	請確認下列內容： - 錯誤訊息中指定的檔案存在且有效。 <code>limits.config</code> 允許開啟足夠的檔案。 如果您是在 MacOS 上測試，請提高開啟檔案限制。預設限制為 256，這對測試而言已足夠。
117	WriteFileFailed	無法寫入檔案 (可以是 DUT 或測試機器)。	確認錯誤訊息中指定的目錄存在，而且您具有寫入許可。
118	FileCleanUpError	測試無法移除指定檔案或目錄，或無法在遠端裝置上卸載指定檔案。	如果二進位檔案仍在執行，檔案可能會鎖定。結束程序並刪除指定檔案。
119	InvalidInputError	無效的組態。	確認您的 <code>suite.json</code> 檔案是否有效。

錯誤碼	錯誤代碼名稱	可能的根本原因	故障診斷
120	InvalidCredentialError	無效的 AWS 登入資料。	- 驗證您的 AWS 登入資料。如需詳細資訊，請參閱the section called “設定您的 AWS 登入資料”。 - 請檢查網路連線並重新執行測試群組。網路問題也可能導致這個錯誤。
121	AWSSessionError	無法建立 AWS 工作階段。	如果 AWS 登入資料無效或網際網路連線不穩定，可能會發生此錯誤。嘗試使用 AWS CLI 呼叫 AWS API 操作。
122	AWSApiCallError	發生 AWS API 錯誤。	此錯誤可能是網路問題造成的。請檢查您的網路，然後再重試測試群組。

錯誤碼	錯誤代碼名稱	可能的根本原因	故障診斷
123	IpNotExistError	IP 地址未包含於連線資訊。	請檢查您的網際網路連線。您可以使用 AWS IoT Greengrass 主控台來檢查測試所使用之 AWS IoT Greengrass 核心物件的連線資訊。如果連線資訊中包含 10 個端點，您可以移除一些或全部，再重新執行測試。如需詳細資訊，請參閱 連線資訊 。
124	OTAJobNotCompleteError	OTA 任務未完成。	請檢查網際網路連線並重試 OTA 測試群組。
125	CreateGreengrassServiceRoleError	發生以下其中一項： • 建立角色時發生錯誤。 • 將政策連接至 AWS IoT Greengrass 服務角色時發生錯誤。 • 與服務角色關聯的政策無效。 • 將角色與 建立關聯時發生錯誤 AWS 帳戶。	設定 AWS IoT Greengrass 服務角色。如需詳細資訊，請參閱 the section called “Greengrass 服務角色” 。

錯誤碼	錯誤代碼名稱	可能的根本原因	故障診斷
126	DependenciesNotPresentError	特定測試所需的一或多個相依性不存在裝置上。	檢查測試日誌以查看裝置上缺少哪些相依性： <i><device-tester-extension> /results/<execution-id>/logs/<test-case-name.log></i>
127	InvalidHSMConfiguration	提供的 HSM/PKCS 組態不正確。	在您的 device.json 檔案中，使用 PKCS#11 提供與 HSM 互動所需的正確組態。

錯誤碼	錯誤代碼名稱	可能的根本原因	故障診斷
128	OTAJobNotSucceededError	OTA 任務未成功。	- 如果您個別執行 ota 測試群組，請執行 ggcdependencies 測試群組以確認所有相依性 (例如 wget) 都存在。然後重試 ota 測試群組。 - 檢閱 <code><device-tester-extract-location> / results/ <execution-id>/logs/</code> 下的詳細日誌，以取得故障診斷和錯誤資訊。特別是，請檢查下列日誌： - 主控台日誌 (test_manager.log) - OTA 測試案例日誌 (ota_test.log) - GGC 協助程式日誌 (ota_test_ggc_logs.tar.gz)

錯誤碼	錯誤代碼名稱	可能的根本原因	故障診斷
			OTA 代理程式日誌 (ota_test_ota_logs.tar.gz) - 請檢查網際網路連線並重試 ota 測試群組。 - 如果問題仍然存在，請聯絡 AWS 開發人員支援。
129	NoConnectivityError	主機代理程式無法連線到網際網路。	檢查您的網路連線和防火牆設定。解決連線問題之後，請重試測試群組。
130	NoPermissionError	您用來執行 IDT for 的 IAM 使用者 AWS IoT Greengrass 沒有建立執行 IDT 所需 AWS 資源的許可。	請參閱 許可政策範本 ，了解授予執行 IDT for 所需許可的政策範本 AWS IoT Greengrass。

錯誤碼	錯誤代碼名稱	可能的根本原因	故障診斷
131	LeftoverAgentExistError	當您嘗試啟動 IDT for 時，您的裝置正在執行 AWS IoT Greengrass 程序 AWS IoT Greengrass。	請確定沒有現有的 Greengrass 協助程式正在您的裝置上執行。 - 您可以使用此命令來停止協助程式：<code>sudo ./<absolute-path-to-greengrass-daemon> /greengrassd stop。</code> - 您也可以透過 PID 終止 Greengrass 協助程式。 Note 如果您使用的是現有的安裝 AWS IoT Greengrass，設定為在重新啟動後自動啟動，您必須在重新啟動後和執行測試套件之前停止協助程式。
132	DeviceTimeOffsetError	裝置的時間不正確。	請將裝置設定為正確的時間。

錯誤碼	錯誤代碼名稱	可能的根本原因	故障診斷
133	InvalidMLConfiguration	提供的 ML 組態不正確。	在您的 <code>device.json</code> 檔案中，提供執行 ML 推論測試所需的正確組態。如需詳細資訊，請參閱 the section called “選用：設定您的裝置以取得 ML 資格” 。

解決 IDT AWS IoT Greengrass 錯誤

使用 IDT 時，您必須先取得正確的組態檔案，才能執行 IDT for AWS IoT Greengrass。如果您遇到剖析和組態錯誤，您的第一個步驟是找到並使用適合您的環境的組態範本。

如果您仍然有問題，請參閱下列除錯程序。

主題

- [在哪裡尋找錯誤？](#)
- [剖析錯誤](#)
- [遺漏必要參數錯誤](#)
- [無法開始測試錯誤](#)
- [未授權存取資源錯誤](#)
- [拒絕許可錯誤](#)
- [SSH 連線錯誤](#)
- [逾時錯誤](#)
- [測試時發生找不到錯誤命令](#)
- [macOS 上的安全例外狀況](#)

在哪裡尋找錯誤？

執行期間會在主控台顯示高階錯誤，當所有測試完成時，將會顯示含有錯誤的失敗測試的摘要。`awsiotdevicetester_report.xml` 包含造成測試失敗的所有錯誤的摘要。每次執行完測試的日誌檔會存放在以測試執行的 UUID (在測試執行期間顯示於主控台) 所命名的目錄中。

測試日誌目錄位於 `<device-tester-extract-location>/results/<execution-id>/logs/`。此目錄包含下列檔案，有助於除錯。

檔案	描述
<code>test_manager.log</code>	測試執行期間寫入主控台的所有日誌。結果摘要位於這個檔案的尾端，其中包含失敗測試的清單。 這個檔案中的警告和錯誤日誌提供有關失敗的一些資訊。
<code><test-group-id> __<test-name> .log</code>	特定測試的詳細日誌。
<code><test-name> _ggc_logs.tar.gz</code>	測試期間產生之 AWS IoT Greengrass 核心協助程式的所有日誌壓縮集合。如需詳細資訊，請參閱 疑難排解 AWS IoT Greengrass 。
<code><test-name> _ota_logs.tar.gz</code>	測試期間 OTA AWS IoT Greengrass 代理程式產生的日誌壓縮集合。僅適用於 OTA 測試。
<code><test-name> _basic_assertion_publisher_ggad_logs.tar.gz</code>	AWS IoT 發佈者裝置在測試期間所產生之日誌的壓縮集合。
<code><test-name> _basic_assertion_subscriber_ggad_logs.tar.gz</code>	AWS IoT 訂閱者裝置在測試期間所產生之日誌的壓縮集合。

剖析錯誤

JSON 組態中的錯字有時會導致剖析錯誤。在大部分的情況下，問題是出在 JSON 檔案中省略了括弧、逗號或引號。IDT 會執行 JSON 驗證並列印除錯資訊。該工具會印出發生錯誤的行、行號和語法錯誤的欄號。此資訊應該足以協助您修正錯誤，但如果仍找不到錯誤，則您可以在 IDE、文字編輯器 (例如 Atom 或 Sublime) 或透過線上工具 (如 JSONLint) 手動執行驗證。

遺漏必要參數錯誤

由於新功能會新增到 IDT，可能需要變更組態檔案。使用舊的組態檔案可能會破壞組態。如果發生這種情況，`/results/<execution-id>/logs` 下的 `<test_case_id>.log` 檔案會明確列出所有遺漏的參數。IDT 也會驗證您的 JSON 組態檔案結構描述，以確保使用最新支援的版本。

無法開始測試錯誤

您可能遇到錯誤，指向測試開始期間發生的失敗。有數種可能的原因，因此，請執行下列動作：

- 確定您在執行命令中包含的集區名稱實際存在。集區名稱是直接從您的 `device.json` 檔案參考。
- 確保您集區中的裝置都有正確的組態參數。

未授權存取資源錯誤

您可能會在終端機輸出或 `/results/<execution-id>/logs` 下方的 `test_manager.log` 檔案中看到 `<user or role> is not authorized to access this resource` 錯誤訊息。若要解決這個問題，請將 `AWSIoTDeviceTesterForGreengrassFullAccess` 受管政策連接到您的測試使用者。如需詳細資訊，請參閱[the section called “建立和設定 AWS 帳戶”](#)。

拒絕許可錯誤

IDT 會在待測裝置的各種目錄和檔案上執行操作。這些操作中有些需要根存取。IDT 必須能夠在不輸入密碼的情況下使用 `sudo` 執行命令，才能自動化這些操作。

依照以下步驟，在不輸入密碼的情況下允許 `sudo` 存取。

Note

`user` 和 `username` 是指 IDT 存取待測裝置時所使用的 SSH 使用者。

1. 使用 `sudo usermod -aG sudo <ssh-username>` 將您的 SSH 使用者新增至 `sudo` 群組。
2. 登出後再登入，以使變更生效。
3. 開啟 `/etc/sudoers` 檔案，然後在檔案結尾處新增以下一行：`<ssh-username> ALL=(ALL) NOPASSWD: ALL`

 Note

最佳實務為建議您在編輯 `/etc/sudoers` 時使用 `sudo visudo`。

SSH 連線錯誤

當 IDT 無法連接到待測裝置時，會在 `/results/<execution-id>/logs/<test-case-id>.log` 中記錄連線失敗。SSH 失敗訊息會出現在此日誌檔的頂端，因為連線到待測裝置是 IDT 最先執行的其中一個操作。

大多數的 Windows 設定會使用 PuTTY 終端機應用程式連線到 Linux 主機。此應用程式要求標準的 PEM 私有金鑰檔案轉換為專屬的 Windows 格式，稱為 PPK。在 `device.json` 檔案中設定 IDT，請只使用 PEM 檔案。如果您使用 PPK 檔案，IDT 無法與 AWS IoT Greengrass 裝置建立 SSH 連線，也無法執行測試。

逾時錯誤

您可以指定逾時乘數 (套用到每次測試的逾時預設值)，以延長每次測試的逾時。此旗標設定的任何值必須大於或等於 1.0。

若要使用逾時乘數，請在執行測試時使用旗標 `--timeout-multiplier`。例如：

```
./devicetester_linux run-suite --suite-id GGQ_1.0.0 --pool-id DevicePool1 --timeout-multiplier 2.5
```

如需詳細資訊，請執行 `run-suite --help`。

測試時發生找不到錯誤命令

您需要較舊版本的 OpenSSL 程式庫 (libssl1.0.0) 才能在 AWS IoT Greengrass 裝置上執行測試。目前的 Linux 發行版本大多使用 libssl 版本 1.0.2 或更新版本 (v1.1.0)。

例如，在 Raspberry Pi 中執行以下命令，安裝所需的 libssl 版本：

1.

```
wget http://ftp.us.debian.org/debian/pool/main/o/openssl/libssl1.0.0_1.0.2l-1~bpo8+1_armhf.deb
```
2.

```
sudo dpkg -i libssl1.0.0_1.0.2l-1~bpo8+1_armhf.deb
```

macOS 上的安全例外狀況

當您在使用 macOS 10.15 的主機電腦上執行 IDT 時，無法正確偵測到 IDT 的公證票證，且 IDT 會遭到封鎖而無法執行。若要執行 IDT，您需要將安全性例外狀況授予 `devicetester_mac_x86-64` 可執行檔。

將安全性例外狀況授予 IDT 可執行檔

1. 從 Apple 選單啟動系統偏好設定。
2. 選擇安全與隱私權，然後在一般索引標籤上，按一下鎖定圖示以變更安全設定。
3. 尋找訊息 "`devicetester_mac_x86-64`" was blocked from use because it is not from an identified developer.，然後選擇允許任何方式。
4. 接受安全警告。

如果您對 IDT 支援政策有任何疑問，請聯絡 [AWS 客戶支援](#)。

AWS IoT Device Tester for 的支援政策 AWS IoT Greengrass V1

AWS IoT Device Tester (IDT) for AWS IoT Greengrass 是可下載的測試架構，可讓您驗證 AWS IoT Greengrass 裝置並將其納入 [AWS Partner Device Catalog](#) 中。我們建議您使用最新版本的 AWS IoT Greengrass 和 IDT 來測試或限定您的裝置。如需詳細資訊，請參閱 [《開發人員指南》中的 IDT for 的支援版本 AWS IoT Greengrass V2](#)。AWS IoT Greengrass Version 2

您也可以使用任何支援的 AWS IoT Greengrass 和 IDT 版本來測試或限定您的裝置。雖然您可以繼續使用 [不支援的 IDT 版本](#)，但這些版本不會收到錯誤修正或更新。

Important

截至 2022 年 4 月 4 日，適用於的 AWS IoT Device Tester (IDT) AWS IoT Greengrass V1 不再產生已簽章的資格報告。您無法再透過 AWS IoT Greengrass V1 裝置資格計劃，讓新裝置符合 [AWS Partner 在裝置目錄中](#) 列出的資格。雖然您無法限定 Greengrass V1 裝置的資格，但您可以繼續使用 IDT for AWS IoT Greengrass V1 來測試 Greengrass V1 裝置。我們建議您使用 [IDT for AWS IoT Greengrass V2](#) 在 Device [AWS Partner Catalog](#) 中限定和列出 [Greengrass 裝置](#)。

如果您對支援政策有任何疑問，請聯絡 [AWS 客戶支援](#)。

故障診斷 AWS IoT Greengrass

本節提供疑難排解資訊和可能的解決方案，以協助解決的問題 AWS IoT Greengrass。

如需配額 AWS IoT Greengrass（限制）的詳細資訊，請參閱《》中的 [Service Quotas](#) Amazon Web Services 一般參考。

AWS IoT Greengrass 核心問題

如果 AWS IoT Greengrass 核心軟體未啟動，請嘗試下列一般疑難排解步驟：

- 請確定您安裝的二進位檔適用於您的架構。如需詳細資訊，請參閱 [AWS IoT Greengrass 核心軟體](#)。
- 確保核心裝置有可用的本機儲存空間。如需詳細資訊，請參閱 [the section called “對儲存體問題進行故障診斷”](#)。
- 檢查 runtime.log 和 crash.log 是否有錯誤訊息。如需詳細資訊，請參閱 [the section called “日誌故障診斷”](#)。

搜尋下列症狀和錯誤，尋找有助於疑難排解 AWS IoT Greengrass 核心問題的資訊。

問題

- [錯誤：組態檔案缺少 CaPath、CertPath 或 KeyPath。Greengrass 協助程式在 \[pid = <pid>\] 結束狀態下處理。](#)
- [錯誤：無法剖析 /<greengrass-root>/config/config.json。](#)
- [錯誤：產生 TLS 組態時發生錯誤：ErrUnknownURIScheme](#)
- [錯誤：執行時間無法開始：無法啟動工作者：容器測試逾時。](#)
- [錯誤：無法在本機 Cloudwatch 叫用 PutLogEvents，logGroup: /GreengrassSystem/connection_manager，錯誤：RequestError：傳送請求失敗，原因為：Post http://<path>/cloudwatch/logs/: dial tcp <address>: getsockopt：連線遭拒，回應：{ }。](#)
- [錯誤：無法建立伺服器，因為：無法載入群組：chmod /<greengrass-root>/ggc/deployment/lambda/arn:aws:lambda:<region>:<account-id>:function:<function-name>:<version>/<file-name>：沒有此類檔案或目錄。](#)
- [AWS IoT Greengrass 核心軟體不會在您從沒有容器化的執行變更為在 Greengrass 容器中執行後啟動。](#)
- [錯誤：多工緩衝處理應至少為 262144 位元組。](#)

- 錯誤：[ERROR]-雲端傳訊錯誤：嘗試發佈訊息時發生錯誤。{"errorString": "操作逾時"}
- 錯誤：container_linux.go:344：啟動容器程序造成 "process_linux.go:424：容器 init 造成 \"rootfs_linux.go:64：將 \"/greengrass/ggc/socket/greengrass_ipc.sock\" 裝載至 rootfs \"/greengrass/ggc/packages/<version>/rootfs/merged\" (位於 \"/greengrass_ipc.sock\") 造成 \"stat /greengrass/ggc/socket/greengrass_ipc.sock: permission denied\"\"\"。
- 錯誤：具 PID：<process-id> 的執行中 Greengrass 協助程式。有些系統元件無法啟動。檢查 'runtime.log' 是否有錯誤。
- 裝置陰影與雲端不同步。
- 錯誤：無法接受 TCP 連線。接受 tcp [::]:8000: accept4: 太多開啟的檔案。
- 錯誤：執行時間執行錯誤：無法啟動 lambda 容器。container_linux.go:259：啟動容器程序造成 "process_linux.go:345：容器 init 造成 \"rootfs_linux.go:50：準備 rootfs 造成 \"permission denied\"\"\"。
- 警告：[WARN]-[5]GK 遠端：擷取公有金鑰資料發生錯誤：ErrPrincipalNotConfigured：未設定 MqttCertificate 的私有金鑰。
- 錯誤：嘗試使用角色 arn：aws：iam：：<account-id>：role/<role-name> 存取 s3 url https://<region>-greengrass-updates.s3.<region>.amazonaws.com/core/<architecture>/greengrass-core-<distribution-version>.tar.gz 時，許可遭拒。
- AWS IoT Greengrass 核心設定為使用網路代理，您的 Lambda 函數無法進行傳出連線。
- 此核心位於連線/中斷連線的無限迴圈。runtime.log 檔案包含連線和中斷連線項目的連續系列。
- 錯誤：無法啟動 lambda 容器。container_linux.go: 259: 啟動容器程序導致「process_linux.go: 345: container init 導致「rootfs_linux.go: 62: 將「proc」掛載到 rootfs」
- 【ERROR】-執行時間執行錯誤：無法啟動 lambda 容器。{"errorString": "無法初始化容器掛載：無法在浮水印上方 dir 中遮罩 greengrass 根目錄：無法在目錄 <ggc-path>：檔案存在"} 建立遮罩裝置
- 【ERROR】-Deployment 失敗。{"deploymentId": "<deployment-id>", "errorString": "container test process with pid <pid> failed: container process status: exit status 1"}
- 錯誤：[ERROR]-執行時間執行錯誤：無法啟動 lambda 容器。{"errorString": "無法初始化容器掛載：無法為容器建立覆蓋 fs：正在將覆蓋掛載於 /greengrass/ggc/ packages/<ggc-version>/rootfs/merged 失敗：無法利用 args 來源進行掛載=\"no_source\" dest=\"/greengrass/ggc/packages/<ggc-version>/rootfs/merged\" fstype=\"overlay\" flags=\"0\" data=\"lowerdir=/greengrass/ggc/packages/<ggc-version>/dns:/,upperdir=/greengrass/ggc/packages/<ggc-version>/rootfs/upper,workdir=/greengrass/ggc/packages/<ggc-version>/rootfs/work\": too many levels of symbolic links"}
- 錯誤：[DEBUG]-無法取得路由。捨棄訊息。

- 錯誤:[Errno 24] 太多開啟的 <lambda-function> , [Errno 24] 太多開啟的檔案
- 錯誤：ds 伺服器無法開始接聽通訊端：接聽 unix <ggc-path>/ggc/socket/greengrass_ipc.sock : bind : 無效引數
- 【INFO】 (Copier) aws.greengrass.StreamManager : stdout。原因：
[com.fasterxml.jackson.databind.JsonMappingException](#)：立即超過最小或最大立即
- GPG error: <https://dnw9lb6lzp2d8.cloudfront.net/stable/InRelease>: The following signatures were invalid: EXPKEYSIG 68D644ABD2327D47 AWS Greengrass Master Key

錯誤：組態檔案缺少 CaPath、CertPath 或 KeyPath。Greengrass 協助程式在 [pid = <pid>] 結束狀態下處理。

解決方案：當 AWS IoT Greengrass 核心軟體未啟動 crash.log 時，您可能會在 中看到此錯誤。如果您執行的是 1.6 版或更早的版本，也可能會發生這種情況。執行以下任意一項：

- 升級至 v1.7 或更新版本。我們建議您一律執行最新版本的 AWS IoT Greengrass Core 軟體。如需下載資訊，請參閱 [AWS IoT Greengrass 核心軟體](#)。
- 使用 AWS IoT Greengrass Core 軟體版本的正確 config.json 格式。如需詳細資訊，請參閱 [the section called “AWS IoT Greengrass 核心組態檔案”](#)。

Note

若要尋找核心裝置上已安裝的 AWS IoT Greengrass 核心軟體版本，請在裝置終端機中執行下列命令。

```
cd /greengrass-root/ggc/core/  
sudo ./greengrassd --version
```

錯誤：無法剖析 /<greengrass-root>/config/config.json。

解決方案：您可以在 AWS IoT Greengrass 核心軟體沒有啟動時查看此錯誤。請確定 [Greengrass 組態檔案](#) 使用有效的 JSON 格式。

開啟 config.json (位於 `/greengrass-root/config`) 並驗證 JSON 格式。例如，確保已正確使用逗號。

錯誤：產生 TLS 組態時發生錯誤：ErrUnknownURIScheme

解決方案：您可以在 AWS IoT Greengrass 核心軟體沒有啟動時查看此錯誤。請確定 Greengrass 組態檔案中的 [crypto \(加密\)](#) 區段中的屬性有效。這個錯誤訊息應提供更多資訊。

開啟 config.json (位於 `/greengrass-root/config`)，並檢查 crypto 區段。例如，憑證和金鑰路徑必須使用正確的 URI 格式，並指向正確的位置。

錯誤：執行時間無法開始：無法啟動工作者：容器測試逾時。

解決方案：您可以在 AWS IoT Greengrass 核心軟體沒有啟動時查看此錯誤。在 [Greengrass 組態檔案中](#) 設定 postStartHealthCheckTimeout 屬性。此選用屬性會設定 Greengrass 協助程式等候後置開始健康狀況檢查完成的時間量 (毫秒)。預設值為 30 秒 (30000 毫秒)。

開啟 config.json (位於 `/greengrass-root/config`)。在 runtime 物件上，新增 postStartHealthCheckTimeout 屬性並將值設為大於 30000。視需要插入逗號以建立有效的 JSON 文件。例如：

```
...
"runtime" : {
  "cgroup" : {
    "useSystemd" : "yes"
  },
  "postStartHealthCheckTimeout" : 40000
},
...
```

錯誤：無法在本機 Cloudwatch 叫用 PutLogEvents，logGroup: /GreengrassSystem/connection_manager，錯誤：RequestError：傳送請求失敗，原因為：Post http://<path>/cloudwatch/logs/: dial tcp <address>: getsockopt：連線遭拒，回應：{ }。

解決方案：您可以在 AWS IoT Greengrass 核心軟體沒有啟動時查看此錯誤。如果您在 Raspberry Pi AWS IoT Greengrass 上執行，且所需的記憶體設定尚未完成，就會發生這種情況。如需有關此步驟的詳細資訊，請參閱[此步驟](#)。

錯誤：無法建立伺服器，因為：無法載入群組：chmod /<greengrass-root>/ggc/deployment/lambda/arn:aws:lambda:<region>:<account-id>:function:<function-name>:<version>/<file-name>：沒有此類檔案或目錄。

解決方案：您可以在 AWS IoT Greengrass 核心軟體沒有啟動時查看此錯誤。如果您已將 [Lambda 可執行檔](#) 部署至核心，請檢查 group.json 檔案中的函數Handler屬性（位於 */greengrass-root/ggc/deployment/group*）。若處理常式不是您所編譯可執行檔的完全相符名稱，則請以空白 JSON 物件 ({}) 取代 group.json 檔案的內容，並執行以下命令來啟動 AWS IoT Greengrass：

```
cd /greengrass/ggc/core/  
sudo ./greengrassd start
```

然後，使用 [AWS Lambda API](#) 更新函數組態的 handler 參數，發佈新功能版本和更新別名。如需詳細資訊，請參閱 [AWS Lambda 函數版本控制和別名](#)。

假設您已透過別名 (建議) 新增函數到 Greengrass，那麼您現在可以重新部署您的群組。(若否，您必須在您的群組定義和訂閱中指向新的函數版本或別名，才能部署群組。)

AWS IoT Greengrass 核心軟體不會在您從沒有容器化的執行變更為在 Greengrass 容器中執行後啟動。

解決方案：檢查您沒有遺漏任何容器相依性。

錯誤：多工緩衝處理應至少為 262144 位元組。

解決方案：您可以在 AWS IoT Greengrass 核心軟體沒有啟動時查看此錯誤。開啟 `group.json` 檔案 (位於 `/greengrass-root/ggc/deployment/group`)、以空的 JSON 物件 (`{}`) 取代檔案的內容，並執行下列命令來啟動 AWS IoT Greengrass：

```
cd /greengrass/ggc/core/  
sudo ./greengrassd start
```

然後依照 [the section called “在本機儲存快取訊息”](#) 程序中的步驟操作。對於 `GGCloudSpooler` 函數，請務必指定大於或等於 262144 的 `GG_CONFIG_MAX_SIZE_BYTES` 值。

錯誤：[ERROR]-雲端傳訊錯誤：嘗試發佈訊息時發生錯誤。{"errorString": "操作逾時"}

解決方案：當 Greengrass 核心無法傳送 MQTT 訊息給 AWS IoT Core 時，您可能會在 `GGCloudSpooler.log` 中看到此錯誤。如果核心環境的頻寬有限又有高延遲，就可能發生這種情況。如果您執行的是 AWS IoT Greengrass v1.10.2 或更新版本，請嘗試增加 [config.json](#) 檔案中 `mqttOperationTimeout` 的值。如果該屬性不存在，請將其新增到 `coreThing` 物件。例如：

```
{  
  "coreThing": {  
    "mqttOperationTimeout": 10,  
    "caPath": "root-ca.pem",  
    "certPath": "hash.cert.pem",  
    "keyPath": "hash.private.key",  
    ...  
  },  
  ...  
}
```

預設值為 5，最小值為 5。

錯誤： container_linux.go:344：啟動容器程序造成 "process_linux.go:424：容器 init 造成 \"rootfs_linux.go:64：將 \"/greengrass/ggc/socket/greengrass_ipc.sock\" 裝載至 rootfs \"/greengrass/ggc/packages/<version>/rootfs/merged\" (位於 \"/greengrass_ipc.sock\")) 造成 \"stat /greengrass/ggc/socket/greengrass_ipc.sock: permission denied\"。"

解決方案： 當 AWS IoT Greengrass 核心軟體未啟動 runtime.log 時，您可能會在 中看到此錯誤。如果您的 umask 高於 0022，就會發生此情形。若要解決這個問題，您必須將 umask 設定為 0022 或更低。數值 0022 預設會授予所有人讀取新檔案的許可。

錯誤： 具 PID：<process-id> 的執行中 Greengrass 協助程式。有些系統元件無法啟動。檢查 'runtime.log' 是否有錯誤。

解決方案： 您可以在 AWS IoT Greengrass 核心軟體沒有啟動時查看此錯誤。檢查 runtime.log 和 crash.log 是否有特定的錯誤資訊。如需詳細資訊，請參閱 [the section called “日誌故障診斷”](#)。

裝置陰影與雲端不同步。

解決方案： 確定 AWS IoT Greengrass 具有 [Greengrass 服務角色](#) 中 iot:UpdateThingShadow 和 iot:GetThingShadow 動作的許可。如果服務角色使用 AWSGreengrassResourceAccessRolePolicy 受管政策，預設會包含這些權限。

請參閱 [陰影同步逾時問題故障診斷](#)。

錯誤： 無法接受 TCP 連線。接受 tcp [::]:8000: accept4: 太多開啟的檔案。

解決方案： 您可以在 greengrassd 指令碼輸出中查看此錯誤。如果 AWS IoT Greengrass Core 軟體的檔案描述項限制已達到閾值且必須增加，就會發生這種情況。

使用以下命令，然後重新啟動 AWS IoT Greengrass Core 軟體。

```
ulimit -n 2048
```

錯誤： container_linux.go:344：啟動容器程序造成 "process_linux.go:424：容器 init 造成 \"rootfs_linux.go:64：將 \"/greengrass/ggc/socket/greengrass_ipc.sock\" 裝載至 rootfs \"/greengrass/ggc/packages/<version>/rootfs/merged\" (位於 \"/greengrass_ipc.sock\")) 造成 \"stat /greengrass/ggc/socket/greengrass_ipc.sock: permission denied\"。"

 Note

在此範例中，將提高限制到 2048。請為您使所用的案例選擇適當的數值。

錯誤：執行時間執行錯誤：無法啟動 lambda 容器。container_linux.go:259：啟動容器程序造成 "process_linux.go:345：容器 init 造成 \"rootfs_linux.go:50：準備 rootfs 造成 \\\"permission denied\\\"\"。"

解決方案：AWS IoT Greengrass 直接在根目錄下安裝，或確保安裝 AWS IoT Greengrass Core 軟體的目錄及其父目錄具有所有人的execute許可。

警告：[WARN]-[5]GK 遠端：擷取公有金鑰資料發生錯誤：ErrPrincipalNotConfigured：未設定 MqttCertificate 的私有金鑰。

解決方案：AWS IoT Greengrass 使用通用處理常式來驗證所有安全主體的屬性。runtime.log 中的這個警告是預期的，除非您已為本機 MQTT 伺服器指定自訂私有金鑰。如需詳細資訊，請參閱[the section called “安全性主體”](#)。

錯誤：嘗試使用角色 arn : aws : iam : : <account-id> : role/<role-name> 存取 s3 url https://<region>-greengrass-updates.s3.<region>.amazonaws.com/core/<architecture>/greengrass-core-<distribution-version>.tar.gz 時，許可遭拒。

解決方案：您可以在無線 (OTA) 更新失敗時查看此錯誤。在簽署者角色政策中，將目標新增 AWS 區域為 Resource。此簽署者角色用於預先簽署 S3 URL 以進行 AWS IoT Greengrass 軟體更新。如需詳細資訊，請參閱 [S3 URL 簽署者角色](#)。

AWS IoT Greengrass 核心設定為使用[網路代理](#)，您的 Lambda 函數無法進行傳出連線。

解決方案：根據您的執行期和 Lambda 函數用來建立連線的可執行檔，您可能也會收到連線逾時錯誤。確保您的 Lambda 函數使用適當的代理組態透過網路 Proxy 進行連線。會透過 `http_proxy`、`https_proxy` 和 `no_proxy` 環境變數，將代理組態 AWS IoT Greengrass 傳遞給使用者定義的 Lambda 函數。存取方式如下面 Python 程式碼片段所示。

```
import os
print(os.environ['http_proxy'])
```

大小寫請與環境中定義的變數相同，例如全部小寫的 `http_proxy` 或全部大寫的 `HTTP_PROXY`。對於這些變數，AWS IoT Greengrass 支援兩者。

Note

大多數用來連線的程式庫 (例如 boto3 或 cURL 和 python requests 套件)，預設會使用這些環境變數。

此核心位於連線/中斷連線的無限迴圈。runtime.log 檔案包含連線和中斷連線項目的連續系列。

解決方案：如果另一個裝置以硬式編碼為使用核心物件名稱做為對 AWS IoT 之 MQTT 連線的用戶端 ID，可能會發生這種情形。相同中的同時連線 AWS 區域，且 AWS 帳戶必須使用唯一的用戶端 IDs。在預設情況下，核心使用核心物件名稱做為這些連線的用戶端 ID。

若要解決這個問題，您可以變更其他裝置用於連線的用戶端 ID (建議使用)，或覆寫核心的預設值。

覆寫核心裝置的預設用戶端 ID

1. 執行下列命令來停止 Greengrass 協助程式：

```
cd /greengrass-root/ggc/core/
sudo ./greengrassd stop
```

2. 開放 `greengrass-root/config/config.json` 由 su 使用者編輯。
3. 在 `coreThing` 物件中，新增 `coreClientId` 屬性，並將值設定為您的自訂用戶端 ID。值必須介於 1 到 128 個字元之間。它在的目前 AWS 區域中必須是唯一的 AWS 帳戶。

```
"coreClientId": "MyCustomClientId"
```

4. 啟動協助程式。

```
cd /greengrass-root/ggc/core/  
sudo ./greengrassd start
```

錯誤：無法啟動 lambda 容器。container_linux.go: 259: 啟動容器程序導致「process_linux.go: 345: container init 導致「rootfs_linux.go: 62: 將「proc」掛載到 rootfs」

解決方案：在某些平台上，當 AWS IoT Greengrass 嘗試掛載 `/proc` 檔案系統以建立 Lambda 容器 runtime.log 時，您可能會在 中看到此錯誤。或可能出現類似錯誤，例如 `operation not permitted` 或 `EPERM`。即使依相依性檢查程式指令碼在平台上執行的測試完全通過，這些錯誤也可能會發生。

請嘗試下列其中一種可能的解決方案：

- 在 Linux 核心中啟用 `CONFIG_DEVPTS_MULTIPLE_INSTANCES` 選項。
- 主機上的 `/proc` 掛載選項只能設為 `rw,relatim`。
- 將 Linux 核心升級至 4.9 版或更新版本。

Note

這個問題與掛載 `/proc` 以提供本機資源存取無關。

【ERROR】-執行時間執行錯誤：無法啟動 lambda 容器。{"errorString": "無法初始化容器掛載：無法在浮水印上方 dir 中遮罩 greengrass 根目錄：無法在目錄 <ggc-path>：檔案存在"} 建立遮罩裝置

解決方案：當部署失敗時，您可能會在 runtime.log 中看到此錯誤。如果 AWS IoT Greengrass 群組中的 Lambda 函數無法存取核心檔案系統中的 /usr 目錄，就會發生此錯誤。

若要解決這個問題，請將本機磁碟區資源新增至群組，然後部署群組。此資源必須：

- 指定 /usr 作為來源路徑和目的地路徑。
- 為擁有資源的 Linux 群組自動新增作業系統群組許可。
- 隸屬於 Lambda 函數，並允許唯讀存取。

【ERROR】-Deployment 失敗。{"deploymentId": "<deployment-id>", "errorString": "container test process with pid <pid> failed : container process status : exit status 1"}

解決方案：當部署失敗時，您可能會在 runtime.log 中看到此錯誤。如果 AWS IoT Greengrass 群組中的 Lambda 函數無法存取核心檔案系統中的 /usr 目錄，就會發生此錯誤。

您可以檢查 GGCanary.log 其他錯誤來確認確實如此。如果 Lambda 函數無法存取 /usr 目錄，GGCanary.log 將包含下列錯誤：

```
[ERROR]-standard_init_linux.go:207: exec user process caused "no such file or directory"
```

若要解決這個問題，請將本機磁碟區資源新增至群組，然後部署群組。此資源必須：

- 指定 /usr 作為來源路徑和目的地路徑。
- 為擁有資源的 Linux 群組自動新增作業系統群組許可。
- 隸屬於 Lambda 函數，並允許唯讀存取。

錯誤：[ERROR -執行時間執行錯誤：無法啟動 lambda 容器。

```
{"errorString": "無法初始化容器掛載：無法為容器建立覆蓋 fs：正在將  
覆蓋掛載於 /greengrass/ggc/ packages/<ggc-version>/rootfs/merged 失敗：無法利用 args 來源進行掛載=\"no_source\" dest=\"/greengrass/ggc/  
packages/<ggc-version>/rootfs/merged\" fstype=\"overlay\" flags=\"0\" data=  
\"lowerdir=/greengrass/ggc/packages/<ggc-version>/dns:/,upperdir=/greengr  
ass/ggc/packages/<ggc-version>/rootfs/upper,workdir=/greengrass/ggc/  
packages/<ggc-version>/rootfs/work\": too many levels of symbolic links"}
```

解決方案：當 AWS IoT Greengrass 核心軟體未啟動時，您可能會在 `runtime.log` 檔案中看到此錯誤。這個問題在 Debian 作業系統上可能比較常見。

要解決此問題，請依照下列步驟：

1. 將 AWS IoT Greengrass Core 軟體升級至 v1.9.3 或更新版本。這應該會自動解決此問題。
2. 如果您在升級 AWS IoT Greengrass Core 軟體後仍收到此錯誤，請在 `configtrue.json` 檔案中將 `system.useOverlayWithTmpfs` 屬性設定為 `???`

Example範例

```
{
  "system": {
    "useOverlayWithTmpfs": true
  },
  "coreThing": {
    "caPath": "root-ca.pem",
    "certPath": "cloud.pem.crt",
    "keyPath": "cloud.pem.key",
    ...
  },
  ...
}
```

Note

您的 AWS IoT Greengrass 核心軟體版本會顯示在錯誤訊息中。若要尋找您的 Linux 核心版本，請執行 `uname -r`。

錯誤：[DEBUG]-無法取得路由。捨棄訊息。

解決方案：檢查您群組中的訂閱，並確認該訂閱有列在 [DEBUG] 訊息中。

錯誤：[Errno 24] 太多開啟的 <lambda-function>，[Errno 24] 太多開啟的檔案

解決方案：如果函數在函數處理常式中執行個體化，您可能會在 `StreamManagerClient` 在 Lambda 函數日誌檔案中看到此錯誤。我們建議您將用戶端建立處理常式之外。如需詳細資訊，請參閱 [the section called “使用 StreamManagerClient 搭配串流”](#)。

錯誤：ds 伺服器無法開始接聽通訊端：接聽 unix <ggc-path>/ggc/socket/greengrass_ipc.sock：bind：無效引數

解決方案：當核心軟體未啟動時，AWS IoT Greengrass 您可能會看到此錯誤。當 AWS IoT Greengrass Core 軟體安裝到具有長檔案路徑的資料夾時，會發生此錯誤。如果您不使用 [寫入目錄](#)，請將 AWS IoT Greengrass Core 軟體重新安裝至檔案路徑少於 79 個位元組的資料夾，或者如果您使用 [寫入目錄](#)，則重新安裝 83 個位元組。

【INFO】 (Copier) aws.greengrass.StreamManager：stdout。原因：
`com.fasterxml.jackson.databind.JsonMappingException`：立即超過最小或最大立即

當您將 AWS IoT Greengrass 核心軟體升級至 v1.11.3 時，如果串流管理員無法啟動，您可能會在串流管理員日誌中看到下列錯誤。

```
2021-07-16T00:54:58.568Z [INFO] (Copier) aws.greengrass.StreamManager:
stdout. Caused by: com.fasterxml.jackson.databind.JsonMappingException:
```

```
Instant exceeds minimum or maximum instant (through reference chain:
com.amazonaws.iot.greengrass.streammanager.export.PersistedSuccessExportStatesV1["lastExportTi
{scriptName=services.aws.greengrass.StreamManager.lifecycle.startup.script,
serviceName=aws.greengrass.StreamManager, currentState=STARTING}
2021-07-16T00:54:58.579Z [INFO] (Copier) aws.greengrass.StreamManager: stdout.
Caused by: java.time.DateTimeException: Instant exceeds minimum or maximum instant.
{scriptName=services.aws.greengrass.StreamManager.lifecycle.startup.script,
serviceName=aws.greengrass.StreamManager, currentState=STARTING}
```

如果您使用的是比 v1.11.3 更舊 AWS IoT Greengrass 的核心軟體版本，而且想要升級至較新的版本，請使用 OTA 更新升級至 v1.11.4。

GPG error: https://dnw9lb6lzp2d8.cloudfront.net stable InRelease: The following signatures were invalid: EXPKEYSIG 68D644ABD2327D47 AWS Greengrass Master Key

當您 apt update 在 [從 APT 儲存庫安裝 AWS IoT Greengrass 核心軟體](#) 的裝置上執行時，您可能會看到下列錯誤。

```
Err:4 https://dnw9lb6lzp2d8.cloudfront.net stable InRelease
  The following signatures were invalid: EXPKEYSIG 68D644ABD2327D47 AWS Greengrass
  Master Key
Reading package lists... Done
W: GPG error: https://dnw9lb6lzp2d8.cloudfront.net stable InRelease: The following
  signatures were invalid: EXPKEYSIG 68D644ABD2327D47 AWS Greengrass Master Key
```

發生此錯誤是因為 AWS IoT Greengrass 不再提供從 APT 儲存庫安裝或更新 AWS IoT Greengrass 核心軟體的選項。若要成功執行 apt update，請從裝置的來源清單中移除儲存 AWS IoT Greengrass 庫。

```
sudo rm /etc/apt/sources.list.d/greengrass.list
sudo apt update
```

部署問題

使用以下資訊協助您排解部署的故障問題。

問題

- [您目前的部署無法運作，因此您想要回復到之前可正常運作的部署。](#)

- 您會在日誌中看到部署 403 Forbidden (403 禁止) 錯誤。
- 當您第一次執行 create-deployment 命令時，就會發生 ConcurrentDeployment 錯誤。
- 錯誤：Greengrass 未獲授權，無法擔任與此帳戶關聯的服務角色，或錯誤：失敗：TES 服務角色並未與此帳戶關聯。
- 錯誤：無法在部署中執行下載步驟，下載時發生錯誤：下載群組定義檔案時發生錯誤：... x509：憑證已過期或無效
- 部署尚未完成。
- 錯誤：找不到 java 或 java8 可執行檔，或錯誤：適用於群組 <group-id> 的類型 NewDeployment 部署 <deployment-id> 失敗 錯誤：<worker-id> 的背景工作者無法初始化，原因是安裝的 Java 版本必須大於或等於 8
- 此部署沒有完成，且 runtime.log 含有多個「等待 1 秒讓容器停止」項目。
- 部署未完成，且 runtime.log 包含 "[ERROR]-Greengrass 部署錯誤：無法將部署狀態回報給雲端 {"deploymentId": "<deployment-id>", "errorString": "無法初始化 PUT，端點: https://<deployment-status>", 錯誤: Put https://<deployment-status>: proxyconnect tcp: x509: 未知授權機構簽署的憑證"}"
- 錯誤：適用於群組 <guid> 的類型 NewDeployment 部署 <deployment-id> 失敗錯誤：處理時發生錯誤。群組設定無效：112 或 [119 0] 在檔案上沒有 rw 許可：<path>。
- 錯誤：<list-of-function-arns> 設定為以根執行，但 Greengrass 未設定為以根許可執行 Lambda 函數。
- 錯誤：適用於群組 <group-id> 的類型 NewDeployment 部署 <deployment-id> 失敗錯誤：Greengrass 部署錯誤：無法在部署中執行下載步驟。處理時發生錯誤：無法載入下載的群組檔案：無法根據使用者名稱 userName 找到 UID：userName：ggc_user：使用者：不明的使用者 ggc_user。
- 錯誤：[ERROR] 執行時間執行錯誤：無法啟動 lambda 容器。{"errorString": "無法初始化容器掛載：無法在覆蓋上層目錄中遮罩 Greengrass 根目錄：無法在目錄 <ggc-path> 上建立遮罩裝置：檔案存在"}"
- 錯誤：適用於群組 <group-id> 類型 NewDeployment 的部署 <deployment-id> 失敗錯誤：程序開始失敗：container_linux.go:259：啟動容器程序造成 "process_linux.go:250：為 init 執行 exec setns 程序造成 \"等待：沒有子程序\""
- 錯誤：【WARN】-MQTT【client】撥號 tcp：lookup <host-prefix>-ats.iot.<region>.amazonaws.com：無此類主機... 【ERROR】-Greengrass 部署錯誤：無法將部署狀態回報給雲端 ... net/http：請求在等待連線時取消（在等待標頭時超過 Client.Timeout)

您目前的部署無法運作，因此您想要回復到之前可正常運作的部署。

解決方案：使用 AWS IoT 主控台或 AWS IoT Greengrass API 重新部署先前的工作部署。這會將對應的群組版本部署至您的核心裝置。

重新部署 (主控台)

1. 在群組組態頁面上，選擇部署索引標籤。此頁面會顯示群組的部署歷史記錄，包括日期和時間、群組版本，以及每個部署嘗試的狀態。
2. 尋找您要重新部署的資料列。選取您要重新部署的部署，然後選擇重新部署。

Deployments	Group history overview		
Subscriptions	By deployment		
Cores	Deployed	Version	Status
Devices	Jul 1, 2019 1:56:49 PM -0700	8dd1d899-4ac9-4f5d-afe4-22de086efc62	Successfully complet... ⋮
Lambdas	Jul 1, 2019 1:41:47 PM -0700	4ad66e5d-3808-446b-940a-b1a788898382	Successfully complet... ⋮
Resources	Jun 18, 2019 8:16:02 AM -0700	1f3870b6-850e-4c97-8018-c872e17b235b	Failed ⋮
Connectors			

重新部署 (CLI)

1. 使用 [ListDeployments](#) 尋找您要重新部署的部署 ID。例如：

```
aws greengrass list-deployments --group-id 74d0b623-c2f2-4cad-9acc-ef92f61fcaf7
```

命令會傳回群組的部署清單。

```
{
  "Deployments": [
    {
      "DeploymentId": "8d179428-f617-4a77-8a0c-3d61fb8446a6",
      "DeploymentType": "NewDeployment",
      "GroupArn": "arn:aws:greengrass:us-west-2:123456789012:/greengrass/groups/74d0b623-c2f2-4cad-9acc-ef92f61fcaf7/versions/8dd1d899-4ac9-4f5d-afe4-22de086efc62",
      "CreatedAt": "2019-07-01T20:56:49.641Z"
    },
    {
      "DeploymentId": "f8e4c455-8ac4-453a-8252-512dc3e9c596",
```



```

        "DeploymentType": "NewDeployment",
        "GroupArn": "arn:aws:greengrass:us-west-2::123456789012:/greengrass/
groups/74d0b623-c2f2-4cad-9acc-ef92f61fcaf7/versions/4ad66e5d-3808-446b-940a-
b1a788898382",
        "CreatedAt": "2019-07-01T20:41:47.048Z"
    },
    {
        "DeploymentId": "e4aca044-bbd8-41b4-b697-930ca7c40f3e",
        "DeploymentType": "NewDeployment",
        "GroupArn": "arn:aws:greengrass:us-west-2::123456789012:/greengrass/
groups/74d0b623-c2f2-4cad-9acc-ef92f61fcaf7/versions/1f3870b6-850e-4c97-8018-
c872e17b235b",
        "CreatedAt": "2019-06-18T15:16:02.965Z"
    }
]
}

```

Note

這些 AWS CLI 命令會使用群組和部署 ID 的範例值。當您執行命令時，請務必取代範例值。

2. 使用 [CreateDeployment](#) 來重新部署目標部署。將部署類型設定為 Redeployment。例如：

```

aws greengrass create-deployment --deployment-type Redeployment \
  --group-id 74d0b623-c2f2-4cad-9acc-ef92f61fcaf7 \
  --deployment-id f8e4c455-8ac4-453a-8252-512dc3e9c596

```

命令會傳回新部署的 ARN 和 ID。

```

{
    "DeploymentId": "f9ed02b7-c28e-4df6-83b1-e9553ddd0fc2",
    "DeploymentArn": "arn:aws:greengrass:us-west-2::123456789012:/greengrass/
groups/74d0b623-c2f2-4cad-9acc-ef92f61fcaf7/deployments/f9ed02b7-c28e-4df6-83b1-
e9553ddd0fc2"
}

```

3. 使用 [GetDeploymentStatus](#) 來取得部署的狀態。

您會在日誌中看到部署 403 Forbidden (403 禁止) 錯誤。

解決方案：確定雲端中 AWS IoT Greengrass 核心的政策包含 "greengrass:*" 作為允許的動作。

當您第一次執行 create-deployment 命令時，就會發生 ConcurrentDeployment 錯誤。

解決方案：部署可能在進行中。您可以執行 [get-deployment-status](#) 來查看是否已建立部署。如果尚未建立，可以再建立一次部署。

錯誤：Greengrass 未獲授權，無法擔任與此帳戶關聯的服務角色，或錯誤：失敗：TES 服務角色並未與此帳戶關聯。

解決方案：部署失敗時，您可能會看到此錯誤。檢查 Greengrass 服務角色是否與目前 AWS 帳戶中的相關聯 AWS 區域。如需詳細資訊，請參閱 [the section called “管理服務角色 \(CLI\)”](#) 或 [the section called “管理服務角色 \(主控台\)”](#)。

錯誤：無法在部署中執行下載步驟，下載時發生錯誤：下載群組定義檔案時發生錯誤：... x509：憑證已過期或無效

解決方案：您可以在部署失敗時，在 runtime.log 查看此錯誤。如果您收到 Deployment failed 錯誤，其包含訊息 x509: certificate has expired or is not yet valid，請檢查裝置時鐘。TLS 和 X.509 憑證提供安全基礎以便建置 IoT 系統，但是這些憑證在伺服器 and 用戶端上需要準確的時間。IoT 裝置在嘗試連線至 AWS IoT Greengrass 或其他使用伺服器憑證的 TLS 服務之前，應該有正確的時間 (15 分鐘內)。如需詳細資訊，請參閱 [使用裝置時間驗證官方部落格上的物聯網 AWS IoT 伺服器憑證](#)。AWS

部署尚未完成。

解決方案：執行下列動作：

- 請確定 AWS IoT Greengrass 協助程式正在您的核心裝置上執行。在核心裝置終端機中，執行下列命令來檢查協助程式是否正在執行，並視需要啟動它。

1. 檢查精靈是否有在運作：

```
ps aux | grep -E 'greengrass.*daemon'
```

若輸出的 root 含有 /greengrass/ggc/packages/1.11.6/bin/daemon 項目，則精靈有在運作。

路徑中的版本取決於核心裝置上安裝的核心 AWS IoT Greengrass 軟體版本。

2. 若要啟動協助程式：

```
cd /greengrass/ggc/core/  
sudo ./greengrassd start
```

- 確保您的核心裝置已連線，且核心連線端點的設定都正確。

錯誤：找不到 java 或 java8 可執行檔，或錯誤：適用於群組 <group-id> 的類型 NewDeployment 部署 <deployment-id> 失敗 錯誤：<worker-id> 的背景工作者無法初始化，原因是安裝的 Java 版本必須大於或等於 8

解決方案：如果 AWS IoT Greengrass 核心已啟用串流管理員，您必須在核心裝置上安裝 Java 8 執行時間，才能部署 群組。如需詳細資訊，請參閱串流管理員的[需求](#)。當您使用 AWS IoT 主控台預設群組建立工作流程來建立群組時，預設會啟用串流管理員。

或者，可以停用串流管理員，然後部署群組。如需詳細資訊，請參閱[the section called “進行設定 \(主控台\)”](#)。

此部署沒有完成，且 runtime.log 含有多個「等待 1 秒讓容器停止」項目。

解決方案：在核心裝置終端機中執行下列命令，以重新啟動 AWS IoT Greengrass 協助程式。

```
cd /greengrass/ggc/core/
```

```
sudo ./greengrassd stop
sudo ./greengrassd start
```

部署未完成，且 **runtime.log** 包含 "[ERROR]-Greengrass 部署錯誤：無法將部署狀態回報給雲端 {"deploymentId": "<deployment-id>", "errorString": "無法初始化 PUT，端點: https://<deployment-status>，錯誤: Put https://<deployment-status>: proxyconnect tcp: x509: 未知授權機構簽署的憑證"}"

解決方案：當 Greengrass 核心設定為使用 HTTPS Proxy 連線，且系統上的 Proxy 伺服器憑證鏈結不受信任時，您可能會在 runtime.log 檔案中看到此錯誤。若要嘗試解決這個問題，請將憑證鏈結新增至根 CA 憑證。Greengrass 核心會將此檔案的憑證新增至 HTTPS 和 MQTT 與 AWS IoT Greengrass 的連線中用於 TLS 驗證的憑證集區。

下列範例顯示新增至根 CA 憑證檔案的 Proxy 伺服器 CA 憑證：

```
# My proxy CA
-----BEGIN CERTIFICATE-----
MIIEFTCCA v2gAwIQWgIVAMHSAzWG/5YVRYtRQ0xXUTEpHuEmApzGCSqGSIb3DQEK
\ nCwUAhuL9MQswCQwJVUzEPMAVUzEYMBYGA1UECgwP1hem9uLmNvbSBjb3BmMuMRww
... content of proxy CA certificate ...
+vHIR1t0e5JAm5\ noTIZGoFbK82A0/ n07f/ t5PSIDAim9V3Gc3pSXxCCAQoFYnui
GaPU1Gk1gCE84a0X\ n7Rp/ lND/ PuMZ/ s8Yj1kY2NmYmNjMCAXDTE5MTEyN2cM216
gJMIADggEPADf2/ m45hzEXAMPLE=
-----END CERTIFICATE-----

# Amazon Root CA 1
-----BEGIN CERTIFICATE-----
MIIDQTCCAimGF6AwIBAgITBmyfz/ 5mjAo54vB4ikPmljZKyjANJmApzyMZFo6qBg
ADA5MQswCQYDVQQGEwJVUzEPMA0tMVT8QtpHRh8jrdkGA1UEChMGDV3QQDExBBKW
... content of root CA certificate ...
o/ufQJQWUCyziar1hem9uMRkwFwYVPSHCb2XV4cdFyQzR1KldZwgJcIQ6XUDgHaa
5MsI+ yMRQ+ hDaXJiobl dXgjUka642M4UwtBV8oK2xJNDd2ZhwLnoQdeXeGADKkpy
rqXRfKoQnoZsG4q5WTP46EXAMPLE
-----END CERTIFICATE-----
```

根 CA 憑證檔案預設位於 **/greengrass-root/certs/root.ca.pem** 中。若要尋找核心裝置上的位置，請檢查 [config.json](#) 中的 crypto.caPath 屬性。

 Note

greengrass-root 代表 Core AWS IoT Greengrass 軟體安裝在您裝置上的路徑。通常，這是 /greengrass 目錄。

錯誤：適用於群組 <guid> 的類型 NewDeployment 部署 <deployment-id> 失敗
錯誤：處理時發生錯誤。群組設定無效：112 或 [119 0] 在檔案上沒有 rw 許可：<path>。

解決方案：確保 *<path>* 目錄的擁有者群組具有讀取目錄和寫入目錄許可。

錯誤：<list-of-function-arns> 設定為以根執行，但 Greengrass 未設定為以根許可執行 Lambda 函數。

解決方案：您可以在部署失敗時，在 runtime.log 查看此錯誤。請確定您已設定 AWS IoT Greengrass 允許 Lambda 函數使用根許可執行。將 allowFunctionsToRunAsRoot 中的值變更為 *greengrass_root/config/config.json* 中的 *yes* 或將 Lambda 函數變更為以另一個使用者/群組身分執行。如需詳細資訊，請參閱 [the section called “執行 Lambda 函數做為根”](#)。

錯誤：適用於群組 <group-id> 的類型 NewDeployment 部署 <deployment-id> 失敗
錯誤：Greengrass 部署錯誤：無法在部署中執行下載步驟。處理時發生錯誤：無法載入下載的群組檔案：無法根據使用者名稱 userName 找到 UID：userName：ggc_user：使用者：不明的使用者 ggc_user。

解決方案：如果 AWS IoT Greengrass 群組 [的預設存取身分](#) 使用標準系統帳戶，則 ggc_user 使用者和 ggc_group 群組必須存在於裝置上。如需展示如何新增使用者和群組的指示，請參閱此 [步驟](#)。請務必輸入完全如所示的名稱。

錯誤：[ERROR] 執行時間執行錯誤：無法啟動 lambda 容器。

{"errorString": "無法初始化容器掛載：無法在覆蓋上層目錄中遮罩

Greengrass 根目錄：無法在目錄 <ggc-path> 上建立遮罩裝置：檔案存在"}

解決方案：您可以在部署失敗時，在 `runtime.log` 查看此錯誤。如果 Greengrass 群組中的 Lambda 函數無法存取核心檔案系統中的 `/usr` 目錄，就會發生此錯誤。若要解決這個問題，請將[本機磁碟區資源](#)新增至群組，然後部署群組。資源必須：

- 指定 `/usr` 作為來源路徑和目的地路徑。
- 為擁有資源的 Linux 群組自動新增作業系統群組許可。
- 隸屬於 Lambda 函數，並允許唯讀存取。

錯誤：適用於群組 <group-id> 類型 NewDeployment 的部署 <deployment-id> 失敗錯誤：程序開始失敗：container_linux.go:259：啟動容器程序造成 "process_linux.go:250：為 init 執行 exec setns 程序造成 \"等待：沒有子程序\"。"

解決方案：部署失敗時，您可能會看到此錯誤。重新嘗試部署。

錯誤：【WARN】-MQTT【client】撥號 tcp：lookup <host-prefix>-ats.iot.<region>.amazonaws.com：無此類主機... 【ERROR】-Greengrass 部署錯誤：無法將部署狀態回報給雲端 ... net/http：請求在等待連線時取消（在等待標頭時超過 Client.Timeout）

解決方案：如果您使用 `systemd-resolved`，根據預設啟用 DNSSEC 設定，則可能會看到此錯誤。因此，無法辨識許多公有網域。嘗試到達 AWS IoT Greengrass 端點時，找不到主機，因此您的部署會保持在 In Progress 狀態。

您可以使用下列命令和輸出，來測試是否有此問題。將端點中的 `##` 預留位置取代為 AWS 區域。

```
$ ping greengrass-ats.iot.region.amazonaws.com
ping: greengrass-ats.iot.region.amazonaws.com: Name or service not known
```

```
$ systemd-resolve greengrass-ats.iot.region.amazonaws.com
greengrass-ats.iot.region.amazonaws.com: resolve call failed: DNSSEC validation failed:
failed-auxiliary
```

可能的解決方案是停用 DNSSEC。當 DNSSEC 為 false 時，不會進行 DNSSEC 驗證。如需詳細資訊，請參閱的此[已知問題](#)systemd。

1. 將 DNSSEC=false 新增至 /etc/systemd/resolved.conf。
2. 重新啟動 systemd-resolved。

如需 resolved.conf 和 DNSSEC 的詳細資訊，請在終端機中執行 man resolved.conf。

建立群組和建立函數問題

使用以下資訊來協助疑難排解建立 AWS IoT Greengrass 群組或 Greengrass Lambda 函數的問題。

問題

- [錯誤：群組的 'IsolationMode' 設定無效。](#)
- [錯誤：arn 為 <function-arn> 的 'IsolationMode' 設定無效。](#)
- [錯誤：IsolationMode=NoContainer 中不允許 arn 為 <function-arn> 的函數 MemorySize 設定。](#)
- [錯誤：IsolationMode=NoContainer 中不允許 arn 為 <function-arn> 的函數 Access Sysfs 設定。](#)
- [錯誤：IsolationMode=GreengrassContainer 中需要 arn 為 <function-arn> 的函數 MemorySize 設定。](#)
- [錯誤：函數 <function-arn> 會參考 IsolationMode = NoContainer 中不允許的類型 <resource-type> 資源。](#)
- [錯誤：不允許 arn 為 <function-arn> 的函數執行組態。](#)

錯誤：群組的 'IsolationMode' 設定無效。

解決方案：不支援 function-definition-version 的 DefaultConfig 中的 IsolationMode 值時，可能會發生此錯誤。支援的值為 GreengrassContainer 和 NoContainer。

錯誤：arn 為 <function-arn> 的 'IsolationMode' 設定無效。

解決方案：不支援 function-definition-version 的 <function-arn> 中的 IsolationMode 值時，可能會發生此錯誤。支援的值為 GreengrassContainer 和 NoContainer。

錯誤：IsolationMode=NoContainer 中不允許 arn 為 <function-arn> 的函數 MemorySize 設定。

解決方案：當您指定 MemorySize 值並選擇在沒有容器化的情況下執行時，會發生此錯誤。在沒有容器化的情況下執行的 Lambda 函數不能有記憶體限制。您可以移除限制，也可以將 Lambda 函數變更為在 AWS IoT Greengrass 容器中執行。

錯誤：IsolationMode=NoContainer 中不允許 arn 為 <function-arn> 的函數 Access Sysfs 設定。

解決方案：當您 true 為指定 AccessSysfs 並選擇在沒有容器化的情況下執行時，會發生此錯誤。在沒有容器化的情況下執行的 Lambda 函數必須更新其程式碼，才能直接存取檔案系統，且無法使用 AccessSysfs。您可以指定 false 的值，AccessSysfs 也可以將 Lambda 函數變更為在 AWS IoT Greengrass 容器中執行。

錯誤：IsolationMode=GreengrassContainer 中需要 arn 為 <function-arn> 的函數 MemorySize 設定。

解決方案：此錯誤是因為您未為在 AWS IoT Greengrass 容器中執行的 Lambda 函數指定 MemorySize 限制。指定 MemorySize 值以解決錯誤。

錯誤：函數 <function-arn> 會參考 IsolationMode = NoContainer 中不允許的類型 <resource-type> 資源。

解決方案：在沒有容器化的情況下執行 Lambda 函數時，您無法存取 Local.Device、Local.VolumeML_Model.SageMaker.Job、ML_Model.S3_Object、或

`S3_Object.Generic_Archive` 資源類型。如果您需要這些資源類型，您必須在 AWS IoT Greengrass 容器中執行。您也可以執行時直接存取本機裝置，無需容器化，方法是變更 Lambda 函數中的程式碼。

錯誤：不允許 `arn` 為 `<function-arn>` 的函數執行組態。

解決方案：當您使用 `GGIPDetector` 或 建立系統 Lambda 函數，`GGCloudSpooler` 並指定 `IsolationMode` 或 `RunAs` 組態時，就會發生此錯誤。您必須省略此系統 Lambda 函數的 `Execution` 參數。

Discovery 問題

使用以下資訊來協助對 AWS IoT Greengrass Discovery 服務的問題進行故障診斷。

問題

- [錯誤：裝置是太多群組的成員，裝置不得在超過 10 個群組中](#)

錯誤：裝置是太多群組的成員，裝置不得在超過 10 個群組中

解決方案：這是已知的限制。[用戶端裝置](#)最多可為 10 個群組的成員。

機器學習資源問題

使用下列資訊協助疑難排解機器學習資源的問題。

問題

- [InvalidMLModelOwner - GroupOwnerSetting 在 ML 模型資源中提供，但是 GroupOwner 或 GroupPermission 不存在](#)
- [附加 Machine Learning 資源時，NoContainer 函數無法設定權限。<function-arn> 係指在資源存取政策中，具有權限 <ro/rw> 的 Machine Learning 資源 <resource-id>。](#)

- [函數 <function-arn> 係指在資源 <resource-id> ResourceAccessPolicy 和資源 OwnerSetting 中具有遺失權限的 Machine Learning 資源。](#)
- [函數 <function-arn> 係指具有 \"rw\" 權限的 Machine Learning 資源 <resource-id>，而資源擁有者設定 GroupPermission 只允許 \"ro\"。](#)
- [NoContainer 函數 <function-arn> 係指巢狀目的地路徑的資源。](#)
- [Lambda <function-arn> 透過共用相同群組擁有者 ID 來獲得資源 <resource-id> 的存取權](#)

InvalidMLModelOwner - GroupOwnerSetting 在 ML 模型資源中提供，但是 GroupOwner 或 GroupPermission 不存在

解決方案：如果機器學習資源包含 [ResourceDownloadOwnerSetting](#) 物件，但未定義必要的 GroupOwner 或 GroupPermission 屬性，便會收到此錯誤訊息。若要解決此問題，請定義遺失的屬性。

附加 Machine Learning 資源時，NoContainer 函數無法設定權限。<function-arn> 係指在資源存取政策中，具有權限 <ro/rw> 的 Machine Learning 資源 <resource-id>。

解決方案：如果非容器化 Lambda 函數指定機器學習資源的函數層級許可，您會收到此錯誤。非容器化函數必須從機器學習資源上定義的資源擁有者權限繼承權限。若要解決此問題，請選擇[繼承資源擁有者許可](#)（主控台），或從 [Lambda 函數的資源存取政策 \(API\)](#) 移除許可。

函數 <function-arn> 係指在資源 <resource-id> ResourceAccessPolicy 和資源 OwnerSetting 中具有遺失權限的 Machine Learning 資源。

解決方案：如果未針對連接的 Lambda 函數或資源設定機器學習資源的許可，您會收到此錯誤。若要解決此問題，請在 Lambda 函數的 [ResourceAccessPolicy](#) 屬性或資源的 [OwnerSetting](#) 屬性中設定許可。

函數 <function-arn> 係指具有 \"rw\" 權限的 Machine Learning 資源 <resource-id>，而資源擁有者設定 GroupPermission 只允許 \"ro\"。

解決方案：如果為連接的 Lambda 函數定義的存取許可超過為機器學習資源定義的資源擁有者許可，您會收到此錯誤。若要解決此問題，請為 Lambda 函數設定更嚴格的許可，或為資源擁有者設定較不嚴格的許可。

NoContainer 函數 <function-arn> 係指巢狀目的地路徑的資源。

解決方案：如果連接到非容器化 Lambda 函數的多個機器學習資源使用相同的目的地路徑或巢狀目的地路徑，則會收到此錯誤。若要解決此問題，請為資源指定個別目的地路徑。

Lambda <function-arn> 透過共用相同群組擁有者 ID 來獲得資源 <resource-id> 的存取權

解決方案：runtime.log 如果將相同的作業系統群組指定為 Lambda 函數的 [Run as](#) identity 和機器學習資源的資源擁有者，但資源未連接到 Lambda 函數，則會收到此錯誤。此組態提供 Lambda 函數隱含許可，其可在未經授權的情況下用來存取資源 AWS IoT Greengrass。

若要解決此問題，請針對其中一個屬性使用不同的作業系統群組，或將機器學習資源連接至 Lambda 函數。

AWS IoT Greengrass Docker 問題的核心

使用以下資訊來協助疑難排解在 Docker 容器中執行 AWS IoT Greengrass 核心的問題。

問題

- [錯誤：不明選項：-no-include-email。](#)
- [警告：IPv4 已停用。網路將無法運作。](#)
- [錯誤：防火牆封鎖了 Windows 和容器之間的檔案共用。](#)
- [錯誤：呼叫 GetAuthorizationToken 操作時發生錯誤 \(AccessDeniedException\)：User：arn：aws：iam：：<account-id>：user/<user-name> 未獲授權在資源上執行：ecr：GetAuthorizationToken：*](#)

- 錯誤：無法為服務 greengrass 建立容器：衝突。容器名稱 "/aws-iot-greengrass" 已在使用中。
- 錯誤：[FATAL]-因為意外錯誤，無法重設執行緒的掛載命名空間：「不允許操作」。為了保持一致性，GGC 會當機且需要手動重新啟動。

錯誤：不明選項：-no-include-email。

解決方案：執行 `aws ecr get-login` 命令時，可能會發生此錯誤。請確定您已安裝 AWS CLI 最新版本（例如，執行：`pip install awscli --upgrade --user`）。如果您使用 Windows 並已使用 MSI 安裝程式安裝 CLI，您必須重新啟動安裝程序。如需詳細資訊，請參閱 AWS Command Line Interface 《使用者指南》中的在 [Microsoft Windows AWS Command Line Interface 上安裝](#)。

警告：IPv4 已停用。網路將無法運作。

解決方案：在 Linux AWS IoT Greengrass 電腦上執行時，您可能會收到此警告或類似訊息。如[此步驟](#)所述啟用 IPv4 網路轉送。未啟用 IPv4 轉送時 AWS IoT Greengrass，雲端部署和 MQTT 通訊無法運作。如需詳細資訊，請參閱 Docker 文件中的[在執行時間設定命名空間核心參數 \(sysctls\)](#)。

錯誤：防火牆封鎖了 Windows 和容器之間的檔案共用。

解決方案：在 Windows 電腦上執行 Docker 時，您可能收到此錯誤或 Firewall Detected 訊息。如果您登入虛擬私有網路 (VPN)，而您的網路設定防止掛載共用磁碟機，也可能會發生這個錯誤。在這種情況下，請關閉 VPN 並重新執行 Docker 容器。

錯誤：呼叫 GetAuthorizationToken 操作時發生錯誤

(AccessDeniedException) : User : arn : aws : iam : : <account-id> : user/
<user-name> 未獲授權在資源上執行 : ecr : GetAuthorizationToken : *

如果您沒有足夠的許可來存取 Amazon ECR 儲存庫，則執行 `aws ecr get-login-password` 命令時可能會收到此錯誤。如需詳細資訊，請參閱 [《Amazon ECR 使用者指南》中的 Amazon ECR 儲存庫政策範例和存取一個 Amazon ECR 儲存庫](#)。

錯誤：無法為服務 greengrass 建立容器：衝突。容器名稱 "/aws-iot-greengrass" 已在使用中。

解決方案：當容器名稱由舊版容器使用時，可能會發生這種情況。若要解決這個問題，請執行以下命令來移除舊的 Docker 容器：

```
docker rm -f $(docker ps -a -q -f "name=aws-iot-greengrass")
```

錯誤：[FATAL]-因為意外錯誤，無法重設執行緒的掛載命名空間：「不允許操作」。為了保持一致性，GGC 會當機且需要手動重新啟動。

解決方案：當您嘗試將 GreengrassContainer Lambda 函數部署到 Docker 容器中執行 AWS IoT Greengrass 的核心時，runtime.log 可能會發生此錯誤。目前，只有 NoContainer Lambda 函數可以部署到 Greengrass Docker 容器。

若要解決此問題，[請確定所有 Lambda 函數都處於 NoContainer 模式](#)，並啟動新的部署。然後，啟動容器時，請勿將現有 deployment 目錄繫結掛載至 AWS IoT Greengrass 核心 Docker 容器。而是建立一個空的 deployment 目錄來取代並在 Docker 容器中綁定掛載該目錄。這可讓新的 Docker 容器使用在 NoContainer 模式下執行的 Lambda 函數接收最新的部署。

如需詳細資訊，請參閱[the section called “AWS IoT Greengrass 在 Docker 容器中執行”](#)。

日誌故障診斷

您可以設定 Greengrass 群組的記錄設定，例如是否要將日誌傳送至 CloudWatch Logs、將日誌存放在本機檔案系統上，或兩者。若要在問題故障診斷時取得詳細資訊，您可以暫時將記錄層級變更為 DEBUG。對記錄設定所做的變更會在部署群組時生效。如需詳細資訊，請參閱[the section called “設定的記錄 AWS IoT Greengrass”](#)。

在本機檔案系統上，會將日誌 AWS IoT Greengrass 存放在下列位置。在檔案系統中讀取日誌需要根許可。

greengrass-root/ggc/var/log/crash.log

顯示 AWS IoT Greengrass 核心當機時產生的訊息。

greengrass-root/ggc/var/log/system/runtime.log

顯示元件故障的訊息。

greengrass-root/ggc/var/log/system/

包含來自 AWS IoT Greengrass 系統元件的所有日誌，例如憑證管理員和連線管理員。透過使用 *ggc/var/log/system/* 和 中的訊息 *ggc/var/log/system/runtime.log*，您應該能夠找出 AWS IoT Greengrass 系統元件中發生的錯誤。

greengrass-root/ggc/var/log/system/localwatch/

包含處理將 Greengrass 日誌上傳至 CloudWatch Logs 之 AWS IoT Greengrass 元件的日誌。如果您無法在 CloudWatch 中檢視 Greengrass 日誌，則可以使用這些日誌進行故障診斷。

greengrass-root/ggc/var/log/user/

包含使用者定義 Lambda 函數的所有日誌。檢查此資料夾，以從本機 Lambda 函數尋找錯誤訊息。

Note

在預設情況下，*greengrass-root* 即為 */greengrass* 目錄。如果已設定好[寫入目錄](#)，那麼日誌就會位在該目錄中。

如果日誌設定為存放在雲端，請使用 CloudWatch Logs 來檢視日誌訊息。*crash.log* 只在 AWS IoT Greengrass 核心裝置上的檔案系統日誌中找到。

如果 AWS IoT 設定為將日誌寫入 CloudWatch，請在系統元件嘗試連線時發生連線錯誤時檢查這些日誌 AWS IoT。

如需 AWS IoT Greengrass 記錄的詳細資訊，請參閱 [the section called “使用 AWS IoT Greengrass 日誌進行監控”](#)。

Note

AWS IoT Greengrass Core 軟體 v1.0 的日誌存放在 *greengrass-root*/var/log 目錄下。

對儲存體問題進行故障診斷

當本機檔案儲存空間已滿時，有些元件可能會開始故障：

- 不會更新本機陰影。
- 新的 AWS IoT Greengrass 核心 MQTT 伺服器憑證無法在本機下載。
- 部署失敗。

您應該隨時注意本機中可用的自由空間容量。您可以根據已部署的 Lambda 函數大小、記錄組態（請參閱[the section called “日誌故障診斷”](#)），以及本機存放的影子數量來計算可用空間。

訊息故障診斷

在本機傳送的所有訊息 AWS IoT Greengrass 都會使用 QoS 0 傳送。根據預設，會將訊息 AWS IoT Greengrass 存放在記憶體內佇列中。因此，當 Greengrass 核心重新啟動時（例如在群組部署或裝置重新開機後），未處理的訊息便會遺失。不過，您可以設定 AWS IoT Greengrass (v1.6 或更新版本) 將訊息快取至檔案系統，使其在核心重新啟動期間持續存在。您也可以設定佇列的容量。如果您設定佇列的容量，請確定其值大於或等於 262144 位元組 (256 KB)。否則，AWS IoT Greengrass 可能無法正常啟動。如需詳細資訊，請參閱[the section called “MQTT 訊息佇列”](#)。

Note

使用預設記憶體內佇列，當服務中斷最低時，我們建議您部署群組或重新啟動裝置。

您也可以設定核心以建立 AWS IoT 的持久性工作階段。這可讓核心在核心離線 AWS 雲端時接收從傳送的訊息。如需詳細資訊，請參閱[the section called “使用的 MQTT 持久性工作階段 AWS IoT Core”](#)。

陰影同步逾時問題故障診斷

若 Greengrass 核心裝置和雲端間的通訊有明顯的延遲，則陰影同步可能會因為逾時而失敗。在這種情況下，您應該會看到與以下內容相似的日誌項目：

```
[2017-07-20T10:01:58.006Z][ERROR]-cloud_shadow_client.go:57,Cloud shadow
client error: unable to get cloud shadow what_the_thing_is_named for
synchronization. Get https://1234567890abcd.iot.us-west-2.amazonaws.com:8443/things/
what_the_thing_is_named/shadow: net/http: request canceled (Client.Timeout exceeded
while awaiting headers)
[2017-07-20T10:01:58.006Z][WARN]-sync_manager.go:263,Failed to get cloud
copy: Get https://1234567890abcd.iot.us-west-2.amazonaws.com:8443/things/
```



```
what_the_thing_is_named/shadow: net/http: request canceled (Client.Timeout exceeded
while awaiting headers)
[2017-07-20T10:01:58.006Z][ERROR]-sync_manager.go:375,Failed to execute sync operation
{what_the_thing_is_named VersionDiscontinued []}]"
```

可能的修復方式是設定核心裝置等待主機回應的時間。在 `中` 開啟 [config.json](#) 檔案，`greengrass-root/config` 並以秒為單位新增具有逾時值 `system.shadowSyncTimeout` 的欄位。例如：

```
{
  "system": {
    "shadowSyncTimeout": 10
  },
  "coreThing": {
    "caPath": "root-ca.pem",
    "certPath": "cloud.pem.crt",
    "keyPath": "cloud.pem.key",
    ...
  },
  ...
}
```

若沒有在 `config.json` 中指定任何 `shadowSyncTimeout` 值，則預設為 5 秒鐘。

Note

對於 AWS IoT Greengrass Core 軟體 v1.6 和更早版本，預設值 `shadowSyncTimeout` 為 1 秒。

檢查 AWS re : Post

如果您無法使用本主題中的疑難排解資訊來解決您的問題，您可以搜尋 [疑難排解](#) 或查看 [AWS IoT Greengrass AWS re : Post 上的標籤](#) 是否有相關問題或發佈新問題。AWS IoT Greengrass 團隊成員主動監控 AWS re : Post。

的文件歷史記錄 AWS IoT Greengrass

下表說明 2018 年 6 月之後 AWS IoT Greengrass 開發人員指南的重要變更。如需有關此文件更新的通知，您可以訂閱 RSS 訂閱源。

變更	描述	日期
適用於 的終止支援通知 AWS IoT Greengrass Version 1	支援終止通知：2026 年 10 月 7 日 AWS 將停止 的支援 AWS IoT Greengrass Version 1。2026 年 10 月 7 日之後，您將無法再存取 AWS IoT Greengrass V1 資源。如需詳細資訊，請造訪 從 遷移 AWS IoT Greengrass Version 1 。	2025 年 10 月 7 日
更新至結束對 v1.11.x Snap 的支援	更新 https : //snapcraft.io 上 AWS IoT Greengrass 核心 v 1.11.x Snap 的終止支援資訊。	2023 年 9 月 22 日
終止對 v1.11.x Snap 的支援	新增 AWS IoT Greengrass 核心 v 1.11.x Snap on snapcraft .io 的終止支援資訊。	2023 年 9 月 19 日
適用於 AWS IoT Greengrass v1.11.6 的 Docker 映像	適用於 AWS IoT Greengrass Core 軟體 v1.11.6 的 Docker 映像可在 Amazon Elastic Container Registry (Amazon ECR) 和 Docker Hub 上取得。我們建議您一律執行最新版本。	2022 年 4 月 12 日
AWS IoT 用於 AWS IoT Greengrass V1 取代的 Device Tester (IDT)	IDT for AWS IoT Greengrass V1 將不再產生已簽署的資格報告。	2022 年 4 月 4 日
支援 Device AWS IoT Tester for AWS IoT Greengrass	IDT for 4.4.1 AWS IoT Greengrass 版現在支援使用	2022 年 3 月 24 日

	AWS IoT Greengrass 核心軟體版本 v1.11.6 進行裝置資格。	
AWS IoT Greengrass 1.11.6 版已發行	1.11.6 版的 AWS IoT Greengrass Core 軟體已推出。此版本包含效能改進功能和錯誤修復程式。我們建議您一律執行最新版本。	2022 年 3 月 24 日
IoT SiteWise 連接器版本 12 已發行	提供 IoT SiteWise 連接器的第 12 版。此版本包含錯誤修正。	2021 年 12 月 23 日
適用於 AWS IoT Greengrass v1.11.5 和 v1.10.5 的 Docker 映像	適用於 AWS IoT Greengrass Core 軟體 v1.11.5 和 v1.10.5 的 Docker 映像可在 Amazon Elastic Container Registry (Amazon ECR) 和 Docker Hub 上取得。我們建議您一律執行最新版本。	2021 年 12 月 22 日
AWS IoT Greengrass V1 維護政策	AWS IoT Greengrass V1 維護政策會定義 AWS IoT Greengrass V1 服務和 AWS IoT Greengrass 核心軟體 v1.x 的不同維護和更新層級。	2021 年 12 月 20 日
AWS IoT Device Tester 4.4.1 版已發行	IDT for 4.4.1 AWS IoT Greengrass 版現已推出。此版本包含 AWS IoT Greengrass 資格套件 (GGQ) v1.3.1，並支援使用 AWS IoT Greengrass 核心軟體版本 v1.11.5 和 v1.10.5 進行裝置資格。	2021 年 12 月 20 日

[AWS IoT Greengrass 發行 1.11.5 和 1.10.5 版](#)

1.11.5 和 1.10.5 版的 AWS IoT Greengrass 核心軟體已推出。這些版本包含效能改善和錯誤修正。我們建議您一律執行最新版本。

[重新發佈的 AWS IoT Greengrass v1.11.4 和 v1.10.4 Docker 映像](#)

AWS IoT Greengrass Core 軟體版本 1.11.4 和 1.10.4 的 Docker 映像已重新發佈至 Amazon Elastic Container Registry (Amazon ECR) 和 Docker Hub，以解決 BusyBox 的錯誤修正。若要使用最新的 Docker 映像，請使用 1.11.4-1 或 1.10.4-1 標籤。如需可用標籤的詳細資訊，請參閱 Docker Hub 中的 [amazon/aws-iot-greengrass](https://hub.docker.com/r/amazon/aws-iot-greengrass)。

[CloudWatch Metrics 連接器支 援輸入資料中的重複時間戳記](#)

您現在可以將具有重複時間戳記的輸入資料傳送至此連接器。

[跨服務混淆代理人預防更新](#)

AWS IoT Greengrass 支援在 IAM 資源政策中使用 [aws:SourceArn](#) 和 [aws:SourceAccount](#) 全域條件內容金鑰，以防止混淆代理人問題。

[適用於 AWS IoT Greengrass v1.11.4 的 Docker 映像](#)

適用於 AWS IoT Greengrass Core 軟體 v1.11.4 的 Docker 映像可在 Amazon Elastic Container Registry (Amazon ECR) 和 Docker Hub 上取得。我們建議您一律執行最新版本。

發佈的 AWS IoT Greengrass v1.11.4 快照	1.11.4 版的 AWS IoT Greengrass Snap 已推出。我們建議您一律執行最新版本。	2021 年 8 月 20 日
支援 Device AWS IoT Tester for AWS IoT Greengrass	IDT for 4.1.0 AWS IoT Greengrass 版現在支援使用 AWS IoT Greengrass 核心軟體版本 v1.11.4 來取得裝置資格。	2021 年 8 月 18 日
AWS IoT Greengrass 1.11.4 版已發行	1.11.4 版的 AWS IoT Greengrass Core 軟體已推出。此版本修正串流管理員無法從舊版 AWS IoT Greengrass 核心軟體升級至 v1.11.3 的問題。我們建議您一律執行最新版本。	2021 年 8 月 17 日
VPC 端點 (AWS PrivateLink)	AWS IoT Greengrass 現在支援 AWS IoT Greengrass 控制平面的介面 VPC 端點 (AWS PrivateLink)。您可以在 VPC 與 AWS IoT Greengrass 控制平面之間建立私有連線。	2021 年 8 月 16 日
AWS IoT Device Tester 4.1.0 版已發行	適用於的 AWS IoT Device Tester 4.1.0 版 AWS IoT Greengrass 已推出。此版本支援使用 AWS IoT Greengrass 核心軟體版本 1.11.3 和 1.10.4 進行裝置資格。	2021 年 6 月 23 日
發佈的 AWS IoT Greengrass v1.11.3 快照	1.11.3 版的 AWS IoT Greengrass snap 包含效能改善和錯誤修正。作為最佳實務，建議您一律執行最新版本。	2021 年 6 月 15 日

[發行 v1.11 AWS IoT Greengrass .3 和 v1.10.4 的 Docker 映像](#)

適用於 AWS IoT Greengrass Core 軟體 v1.11.3 和 v1.10.4 的 Docker 映像可在 Amazon Elastic Container Registry (Amazon ECR) 和 Docker Hub 上取得。這些版本的 AWS IoT Greengrass Core 包含效能改善和錯誤修正。我們建議您一律執行最新版本。

2021 年 6 月 15 日

[AWS IoT Greengrass 1.11.3 版已發行](#)

1.11.3 版的 AWS IoT Greengrass Core 軟體已推出。此版本包含效能改進功能和錯誤修復程式。我們建議您一律執行最新版本。

2021 年 6 月 14 日

[AWS IoT Greengrass 1.10.4 版已發行](#)

1.10.4 版的 AWS IoT Greengrass Core 軟體已推出。此版本包含效能改進功能和錯誤修復程式。我們建議您一律執行最新版本。

2021 年 6 月 14 日

[Modbus-TCP 通訊協定轉接器第 2 版已發行](#)

Modbus-TCP 通訊協定轉接器連接器第 2 版已推出。此版本新增了對 ASCII、UTF8 和 ISO8859 編碼來源字串的支援。

2021 年 5 月 24 日

[已發行 Docker 應用程式部署連接器第 7 版](#)

Greengrass Docker 應用程式部署連接器第 7 版已推出。

2021 年 4 月 5 日

[AWS IoT Greengrass 發行 1.11.1 版](#)

1.11.1 版的 AWS IoT Greengrass Core 軟體已推出。此版本包含效能改進功能和錯誤修復程式。我們建議您一律執行最新版本。

2021 年 3 月 29 日

[AWS IoT Device Tester 4.0.2 版已發行](#)

適用於的 AWS IoT Device Tester 4.0.2 版 AWS IoT Greengrass 已推出。此版本取代 IDT v4.0.0，並新增對 AWS IoT Greengrass Core 軟體 1.11.1 版的支援。這也修正了導致 IDT 遮罩硬體安全整合 (HSI) 錯誤的問題。

2021 年 3 月 29 日

[IoT SiteWise 連接器版本 11 已發佈](#)

提供 IoT SiteWise 連接器的第 11 版。這會啟動對包含隱藏或無法列印字元的字串的支援。此版本也包含一般效能改善和錯誤修正。

2021 年 3 月 24 日

[重新發佈的 AWS IoT Greengrass v1.11.0 快照](#)

AWS IoT Greengrass Snap 1.11.0 版已重新發佈在 Snapcraft 上，以解決使用 Python 解譯器時可能發生的錯誤修正和應用程式當機問題。AWS IoT Greengrass 不提供軟體版本 1.10 和 1.9 的 Snap。

2021 年 3 月 19 日

[支援適用於的 AWS IoT Device Tester 更新 AWS IoT Greengrass](#)

IDT for 4.0.0 AWS IoT Greengrass 版現在支援使用 AWS IoT Greengrass 核心軟體版本 v1.10.3 進行裝置資格。

2021 年 3 月 18 日

[重新發佈的 AWS IoT Greengrass v1.8.4 快照](#)

AWS IoT Greengrass Snap 1.8.4 版已重新發佈在 Snapcraft 上，以解決使用 Python 解譯器時可能發生的錯誤修正和應用程式當機問題。

2021 年 3 月 15 日

[針對 Armv7l 重新發佈 v1 AWS IoT Greengrass .11.0 Docker 映像](#)

適用於 Armv7l 平台之 AWS IoT Greengrass Core 軟體版本 1.11.0 的 Docker 映像已重新發佈至 Amazon Elastic Container Registry (Amazon ECR) 和 Docker Hub，以解決使用 Python 解譯器時的錯誤修正和可能的應用程式當機問題。

2021 年 3 月 8 日

[AWS IoT Greengrass 發行 v1.10.3 Docker 映像](#)

適用於 AWS IoT Greengrass Core 軟體版本 1.10.3 的 Docker 映像現在可在 Amazon Elastic Container Registry (Amazon ECR) 和 Docker Hub 上使用。

2021 年 3 月 8 日

[重新發佈的 AWS IoT Greengrass v1.11.0 和 v1.9.4 Docker 映像](#)

適用於 AWS IoT Greengrass Core 軟體版本 1.11.0 和 1.9.4 的 Docker 映像已重新發佈至 Amazon Elastic Container Registry (Amazon ECR) 和 Docker Hub，以解決使用 Python 解譯器時的錯誤修正和可能的應用程式當機問題。Armv7l 的 Docker 映像目前尚未重新發佈。

2021 年 2 月 26 日

[AWS IoT Greengrass 1.10.3 版已發行](#)

1.10.3 版的 AWS IoT Greengrass Core 軟體已推出。此版本新增了 systemComponentAuthTimeout 核心組態屬性，並包含效能改進和錯誤修正。我們建議您一律執行最新版本。

2021 年 2 月 24 日

[IoT SiteWise 連接器版本 10 已發行](#)

提供 IoT SiteWise 連接器的第 10 版。此版本可在連線中斷時解決 StreamManager 用戶端的穩定性問題，並在不存在時改善 OPC-UA SourceTimestamp 值處理。使用 IoT SiteWise 連接器將本機裝置和設備資料傳送至 IoT SiteWise 中的資產屬性。

2021 年 1 月 22 日

[IoT SiteWise 連接器第 9 版已發行](#)

IoT SiteWise 連接器第 9 版已推出。這會啟動對自訂 Greengrass StreamManager 串流目的地、OPC-UA 不動作帶、自訂掃描模式和自訂掃描速率的支援。這也包括在從 IoT SiteWise 閘道進行組態更新期間改善的效能。使用 IoT SiteWise 連接器將本機裝置和設備資料傳送至 IoT SiteWise 中的資產屬性。

2020 年 12 月 15 日

[AWS IoT Device Tester 4.0.0 版已發行](#)

適用於的 AWS IoT Device Tester 4.0.0 版 AWS IoT Greengrass 已推出。此版本可讓您使用 IDT 來開發和執行自訂測試套件，以進行裝置驗證。這也包含 macOS 和 Windows 的程式碼簽署 IDT 應用程式。

2020 年 12 月 15 日

[AWS IoT Greengrass snap 1.11 版](#)

1.11.0 版的 AWS IoT Greengrass snap 支援非容器化 Lambda 函數。作為最佳實務，建議您一律執行最新版本。

2020 年 12 月 6 日

[IoT SiteWise 連接器版本 8 已發行](#)

IoT SiteWise 連接器第 8 版已推出。當連接器遇到間歇性網路連線時，此版本可改善穩定性。使用 IoT SiteWise 連接器將本機裝置和設備資料傳送至 IoT SiteWise 中的資產屬性。

2020 年 11 月 19 日

[Kinesis Firehose 連接器支援無容器模式](#)

您可以使用 IsolationMode 參數來設定連接器的容器化模式。

2020 年 10 月 19 日

[已發行 Docker 應用程式部署連接器第 6 版](#)

Greengrass Docker 應用程式部署連接器第 6 版已推出。

2020 年 9 月 18 日

[AWS IoT Greengrass 1.11.0 版已發行](#)

1.11.0 版的 AWS IoT Greengrass Core 軟體已推出。此版本新增了系統運作狀態遙測功能和本機運作狀態檢查 API。串流管理員現在可以將資料匯出至 Amazon Simple Storage Service (Amazon S3) 和 IoT SiteWise。此版本也包含效能改善和錯誤修正。我們建議您一律執行最新版本。

2020 年 9 月 16 日

[IoT SiteWise 連接器版本 7 已發佈](#)

IoT SiteWise 連接器第 7 版已推出。此版本修正了閘道指標的問題。使用 IoT SiteWise 連接器將本機裝置和設備資料傳送至 IoT SiteWise 中的資產屬性。

2020 年 8 月 14 日

[ServiceNow MetricBase 整合、Splunk 整合和 Twilio Notifications 連接器支援無容器模式](#)

您可以使用 IsolationMode 參數來設定連接器的容器化模式。

2020 年 7 月 30 日

SNS 連接器支援無容器模式	您可以使用 IsolationMode 參數來設定連接器的容器化模式。	2020 年 7 月 6 日
CloudWatch Metrics 連接器支援無容器模式	您可以使用 IsolationMode 參數來設定連接器的容器化模式。	2020 年 6 月 17 日
AWS IoT Greengrass 1.10.2 版已發行	1.10.2 版的 AWS IoT Greengrass Core 軟體已推出。此版本新增了 mqttOperationTimeout 核心組態屬性，並包含效能改進和錯誤修正。我們建議您一律執行最新版本。	2020 年 6 月 8 日
Tensorflow 機器學習安裝程式已棄用	AWS IoT Greengrass Tensorflow 預先封裝的機器學習安裝程式已棄用。機器學習範例已升級至 Python 3.7。	2020 年 5 月 29 日
Chainer 架構支援和 Greengrass 機器學習安裝程式已棄用	AWS IoT Greengrass MXNet 和 DLR 的預先封裝機器學習安裝程式和下載已棄用。Chainer 框架支援和相關聯的下載已廢除。	2020 年 5 月 4 日

[IoT SiteWise 連接器第 6 版已發行](#)

IoT SiteWise 連接器第 6 版已推出。此版本新增對 CloudWatch 指標的支援，並自動探索新的 OPC-UA 標籤。也就是說，您不需要在標籤針對 OPC-UA 來源進行變更時重新啟動閘道。此版本的連接器需要串流管理員和 AWS IoT Greengrass Core 軟體 v1.10.0 或更高版本。使用 IoT SiteWise 連接器將本機裝置和設備資料傳送至 IoT SiteWise 中的資產屬性。

2020 年 4 月 29 日

[連接器升級至 Python 3.7](#)

支援 Python 執行時間的連接器已升級為 Python 3.7。建議您將連接器版本從 Python 2.7 升級至 Python 3.7。

2020 年 4 月 29 日

[Greengrass 裝置設定可以在無提示模式下執行](#)

您可以在無提示模式下執行 Greengrass 裝置設定，這樣指令碼就不會提示您輸入任何值。

2020 年 4 月 27 日

[新的 Docker 基礎映像](#)

您可以下載以 Alpine Linux AWS IoT Greengrass (x86_64、Arm64 或 AArch64) 為基礎映像建置的 Docker 映像。

2020 年 4 月 23 日

[AWS IoT Greengrass 發行 1.10.1 版](#)

1.10.1 版的 AWS IoT Greengrass Core 軟體已推出。此版本包含效能改進功能和錯誤修復程式。我們建議您一律執行最新版本。

2020 年 4 月 16 日

新增安全性章節	AWS IoT Greengrass 安全內容已重新組織，並新增了新資訊。	2020 年 3 月 30 日
使用 APT 套件管理員來安裝 AWS IoT Greengrass	在支援的 Debian 型 Linux 發行版本上，您可以使用在裝置上 apt 安裝 AWS IoT Greengrass Core 軟體。	2020 年 2 月 26 日
IoT SiteWise 連接器版本 5 已發行	IoT SiteWise 連接器第 5 版已推出。此版本修正 AWS IoT Greengrass 了 Core 軟體 v1.9.4 的相容性問題。使用 IoT SiteWise 連接器將本機裝置和設備資料傳送至 IoT SiteWise 中的資產屬性。	2020 年 2 月 12 日
快速設定核心裝置的新指令碼	您可以使用 Greengrass 裝置安裝程式在幾分鐘內設定核心裝置。此外，AWS IoT Greengrass 現在也支援 Node.js 12.x Lambda 函數。	2019 年 12 月 20 日
AWS IoT Greengrass 發行 1.10.0 版	1.10.0 版的 AWS IoT Greengrass Core 軟體已推出。此版本的新功能包括串流管理員、使用 Docker 應用程式部署連接器的容器支援、可存取機器學習資源的非容器化 Lambda 函數、使用支援 MQTT 持久性工作階段 AWS IoT，以及透過指定連接埠支援本機 MQTT 流量。	2019 年 11 月 25 日

主控台支援部署通知	使用 Amazon EventBridge 主控台建立事件規則，在您的 Greengrass 群組部署變更狀態時觸發。	2019 年 11 月 14 日
AWS IoT Greengrass 1.9.4 版已發行	1.9.4 版的 AWS IoT Greengrass Core 軟體已推出。此版本包含效能改進功能和錯誤修復程式。根據最佳實務，我們建議您一律執行最新的版本。	2019 年 10 月 17 日
主控台支援管理 Greengrass 服務角色	在 AWS IoT 主控台中使用新的和改進的功能來管理您的 Greengrass 服務角色。	2019 年 10 月 4 日
管理群組層級標籤的主控台支援	您可以在 AWS IoT 主控台中建立、檢視和管理 Greengrass 群組的標籤。	2019 年 9 月 23 日
新的機器學習連接器	使用 ML 意見回饋連接器來發佈模型輸入和預測，以及使用 ML 物件偵測連接器來執行本機物件偵測推論服務。	2019 年 9 月 19 日
AWS IoT Greengrass 1.9.3 版已發行	1.9.3 版的 AWS IoT Greengrass Core 軟體已推出。此版本可讓您在 Armv6l 架構上的 Raspbian 分佈上安裝 AWS IoT Greengrass Core 軟體、支援使用 ALPN 的連接埠 443 上的 OTA 更新，以及包含從 Python 2.7 Lambda 函數傳送至其他 Lambda 函數之二進位承載的錯誤修正。	2019 年 9 月 12 日

[AWS IoT Greengrass 1.8.4 版已發行](#)

1.8.4 版的 AWS IoT Greengrass Core 軟體已推出。此版本包含效能改進功能和錯誤修復程式。如果您正在執行 1.8.x 版，建議您升級到 1.8.4 版或 1.9.3 版。使用較舊版本時，建議您升級到 1.9.3 版。

2019 年 8 月 30 日

[AWS IoT Greengrass 1.9.2 版發行，支援 OpenWrt](#)

1.9.2 版的 AWS IoT Greengrass Core 軟體已推出。此版本可讓您使用 Armv8 (AArch64) 和 Armv7l 架構在 OpenWrt 分佈上安裝 AWS IoT Greengrass Core 軟體。

2019 年 6 月 20 日

[AWS IoT Greengrass 1.8.3 版已發行](#)

1.8.3 版的 AWS IoT Greengrass Core 軟體已推出。此版本包含一般效能改進功能和錯誤修復程式。如果您正在執行 1.8.x 版，建議您升級到 1.8.3 版或 1.9.2 版。使用較舊版本時，建議您升級到 1.9.2 版。

2019 年 6 月 20 日

[AWS IoT Greengrass 1.9.1 版已發行](#)

1.9.1 版的 AWS IoT Greengrass Core 軟體已推出。此版本包含中 AWS IoT 包含萬用字元之訊息的錯誤修正。

2019 年 5 月 10 日

AWS IoT Greengrass 1.8.2 版已發行	1.8.2 版的 AWS IoT Greengrass Core 軟體已推出。此版本包含一般效能改進功能和錯誤修復程式。如果您正在執行 1.8.x 版，建議您升級到 1.8.2 版或 1.9.0 版。使用較舊版本時，建議您升級到 1.9.0 版。	2019 年 5 月 2 日
AWS IoT Greengrass 1.9.0 版已發行	新功能：支援 Python 3.7 和 Node.js 8.10 Lambda 執行時間、最佳化 MQTT 連線，以及本機 MQTT 伺服器的橢圓曲線 (EC) 金鑰支援。	2019 年 5 月 1 日
AWS IoT Greengrass 1.8.1 版已發行	1.8.1 版的 AWS IoT Greengrass Core 軟體已推出。此版本包含一般效能改進功能和錯誤修復程式。根據最佳實務，我們建議您一律執行最新的版本。	2019 年 4 月 18 日
AWS IoT Greengrass Snapcraft 上可用的 snap	使用 AWS IoT Greengrass Snap Store 應用程式快速設計、測試和部署 Linux 裝置上的軟體 AWS IoT Greengrass。	2019 年 4 月 1 日
支援使用標籤型許可進行更多存取控制	您可以在 AWS Identity and Access Management (IAM) 政策中使用標籤來控制對 AWS IoT Greengrass 資源的存取。	2019 年 3 月 29 日
IoT Analytics 連接器已發行	使用 IoT Analytics 連接器將本機裝置資料 AWS IoT Analytics 傳送至頻道。	2019 年 3 月 15 日

Kinesis Firehose 連接器中的批次支援	Kinesis Firehose 連接器支援以指定的間隔將批次資料記錄傳送至 Amazon Data Firehose。	2019 年 3 月 15 日
CloudFormation 支援 AWS IoT Greengrass 資源	使用 CloudFormation 範本來建立和管理 AWS IoT Greengrass 資源。	2019 年 3 月 15 日
AWS IoT Greengrass 1.8.0 版已發行	新功能：Lambda 函數的可設定預設存取身分、透過連接埠 443 支援 HTTPS 流量，以及 MQTT 連線的可預測命名用戶端 IDs AWS IoT。	2019 年 3 月 7 日
AWS IoT Greengrass 發行 1.7.1 和 1.6.1 版	1.7.1 和 1.6.1 版的 AWS IoT Greengrass 核心軟體已推出。這些版本需要 Linux 核心版本 3.17 或更新版本。我們建議客戶執行任何版本的 Greengrass 核心軟體以立即升級至 1.7.1 版。	2019 年 2 月 11 日
SageMaker AI Neo 深度學習執行期	SageMaker AI Neo 深度學習執行期支援由 SageMaker AI Neo 深度學習編譯器最佳化的機器學習模型。	2018 年 11 月 28 日
AWS IoT Greengrass 在 Docker 容器中執行	您可以將 Greengrass AWS IoT Greengrass 群組設定為在沒有容器化的情況下執行，以在 Docker 容器中執行。	2018 年 11 月 26 日
AWS IoT Greengrass 1.7.0 版已發行	新功能：Greengrass 連接器、本機秘密管理員、Lambda 函數的隔離和許可設定、信任安全性的硬體根、使用 ALPN 或網路代理的連線，以及 Raspbian Stretch 支援。	2018 年 11 月 26 日

AWS IoT Greengrass 軟體下載	AWS IoT Greengrass Core 軟體、AWS IoT Greengrass Core SDK 和 AWS IoT Greengrass Machine Learning SDK 套件可透過 Amazon CloudFront 進行 download。	2018 年 11 月 26 日
AWS IoT Device Tester for AWS IoT Greengrass	使用 AWS IoT Device Tester for AWS IoT Greengrass 驗證您的 CPU 架構、核心組態和驅動程式是否可搭配使用 AWS IoT Greengrass。	2018 年 11 月 26 日
AWS CloudTrail 記錄 AWS IoT Greengrass API 呼叫	AWS IoT Greengrass 已與服務整合 AWS CloudTrail，此服務可提供使用者、角色或 AWS 服務在其中採取之動作的記錄 AWS IoT Greengrass。	2018 年 10 月 29 日
在 NVIDIA Jetson TX2 上支援 TensorFlow 1.10.1 版	AWS IoT Greengrass 提供的適用於 NVIDIA Jetson TX2 的 TensorFlow 預先編譯程式庫現在使用 TensorFlow 1.10.1 版。這支援 Jetpack 3.3 和 CUDA 工具組 9.0。	2018 年 10 月 18 日
支援 MXNet v1.2.1 機器學習資源	AWS IoT Greengrass 支援使用 MXNet v1.2.1 訓練的機器學習模型。	2018 年 8 月 29 日
AWS IoT Greengrass 發行 1.6.0 版	新功能：Lambda 可執行檔、可設定的訊息佇列、可設定的重新連線重試間隔、/proc 下的磁碟區資源，以及可設定的寫入目錄。	2018 年 7 月 26 日

舊版更新

下表說明 2018 年 7 月之前 AWS IoT Greengrass 開發人員指南的重要變更。

變更	描述	Date
AWS IoT Greengrass 1.5.0 版已發行	新功能： - 使用雲端訓練模型執行本機器學習推論。如需詳細資訊，請參閱執行機器學習推論。 - 除了 JSON 之外，Greengrass Lambda 函數還支援二進位輸入資料。 如需詳細資訊，請參閱 AWS IoT Greengrass Core 版本。	2018 年 3 月 29 日
AWS IoT Greengrass 1.3.0 版已發行	新功能： - 無線 (OTA) 更新代理程式能夠處理雲端部署的 Greengrass 更新任務。如需詳細資訊，請參閱AWS IoT Greengrass Core 軟體的 OTA 更新。 - 從 Greengrass Lambda 函數存取本機周邊裝置和資源。如需詳細資訊，請參閱使用 Lambda 函數和連接器存取本機資源。	2017 年 11 月 27 日
AWS IoT Greengrass 1.1.0 版已發行	新功能： - 重設部署的 AWS IoT Greengrass 群組。如需詳細資訊，請參閱重設部署。 - 除了 Python 2.7 之外，還支援 Node.js 6.10 和 Java 8 Lambda 執行時間。	2017 年 9 月 20 日
AWS IoT Greengrass 1.0.0 版已發行	AWS IoT Greengrass 已全面推出。	2017 年 6 月 7 日