aws

開發人員指南，第 2 版

AWS IoT Greengrass

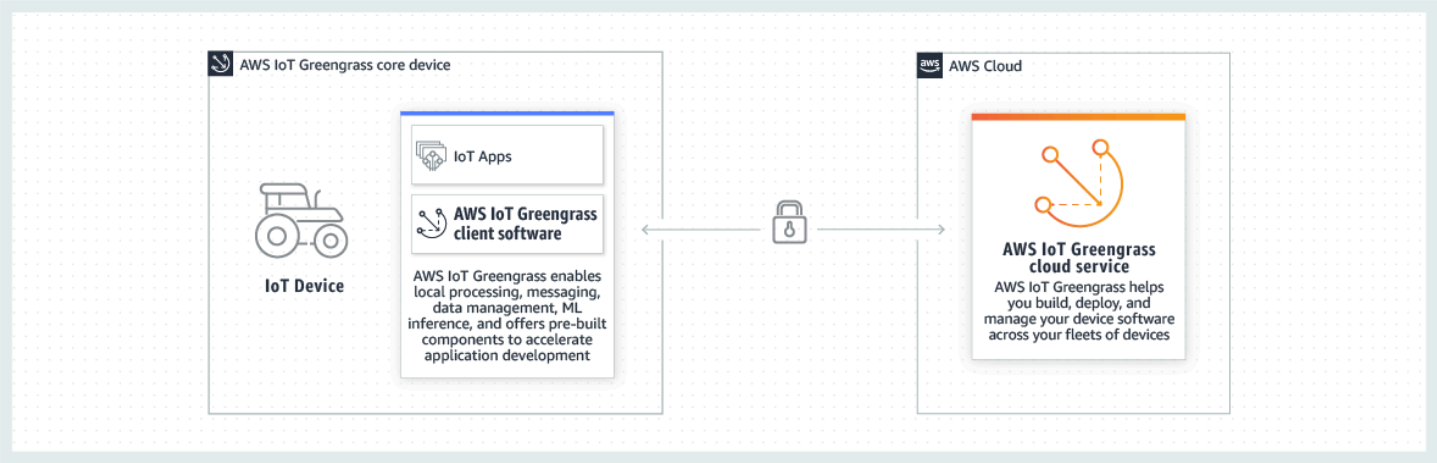
AWS IoT Greengrass: 開發人員指南，第 2 版

Table of Contents

什麼是 AWS IoT Greengrass？

AWS IoT Greengrass是開放原始碼物聯網 (IoT) 邊緣執行階段和雲端服務，可協助您在裝置上建置、部署及管理 IoT 應用程式。您可AWS IoT Greengrass以使用建置可讓裝置根據其產生的資料在本機上採取行動的軟體、根據機器學習模型執行預測，以及篩選和彙總裝置資料。 AWS IoT Greengrass讓您的裝置能夠在更接近資料產生位置的位置收集和分析資料、自動回應本機事件，以及與區域網路上的其他裝置安全通訊。Greengrass 裝置也可以安全地與 IoT 資料通訊，AWS IoT Core並將其匯出到. AWS 雲端 您可以使用預先建置的軟體模組 (元件) 和 AWS IoT Greengrass 來建置邊緣應用程式，這些模組可將邊緣裝置連接至 AWS 服務或第三方服務。您也可以使用 Lambda 函數、Docker 容器、原生作業系統程序或您選擇的自訂執行階段AWS IoT Greengrass來封裝和執行軟體。

下列範例顯示AWS IoT Greengrass裝置如何與. AWS 雲端



新功能

AWS IoT Greengrass V2引入了新功能和改進。以下內容包含有關第 2 版中提供的新功能的詳細資訊。

- 在中有什麼新功能 AWS IoT Greengrass Version 2

對於初次使用的用戶 AWS IoT Greengrass

如果您是新手AWS IoT Greengrass，我們建議您查看以下部分：

- AWS IoT Greengrass 的運作方式

接下來，按照入門教程嘗試的基本功能AWS IoT Greengrass。在本教學課程中，您將在裝置上安裝 AWS IoT Greengrass Core 軟體、開發 Hello World 元件，然後封裝該元件以進行部署。

對於現有用戶 AWS IoT Greengrass

對於目前的使用者AWS IoT Greengrass V1，我們建議您使用下列主題來協助您瞭解 Greengrass 版本 1 和 Greengrass 版本 2 之間的差異，並瞭解如何從版本 1 移至第 2 版：

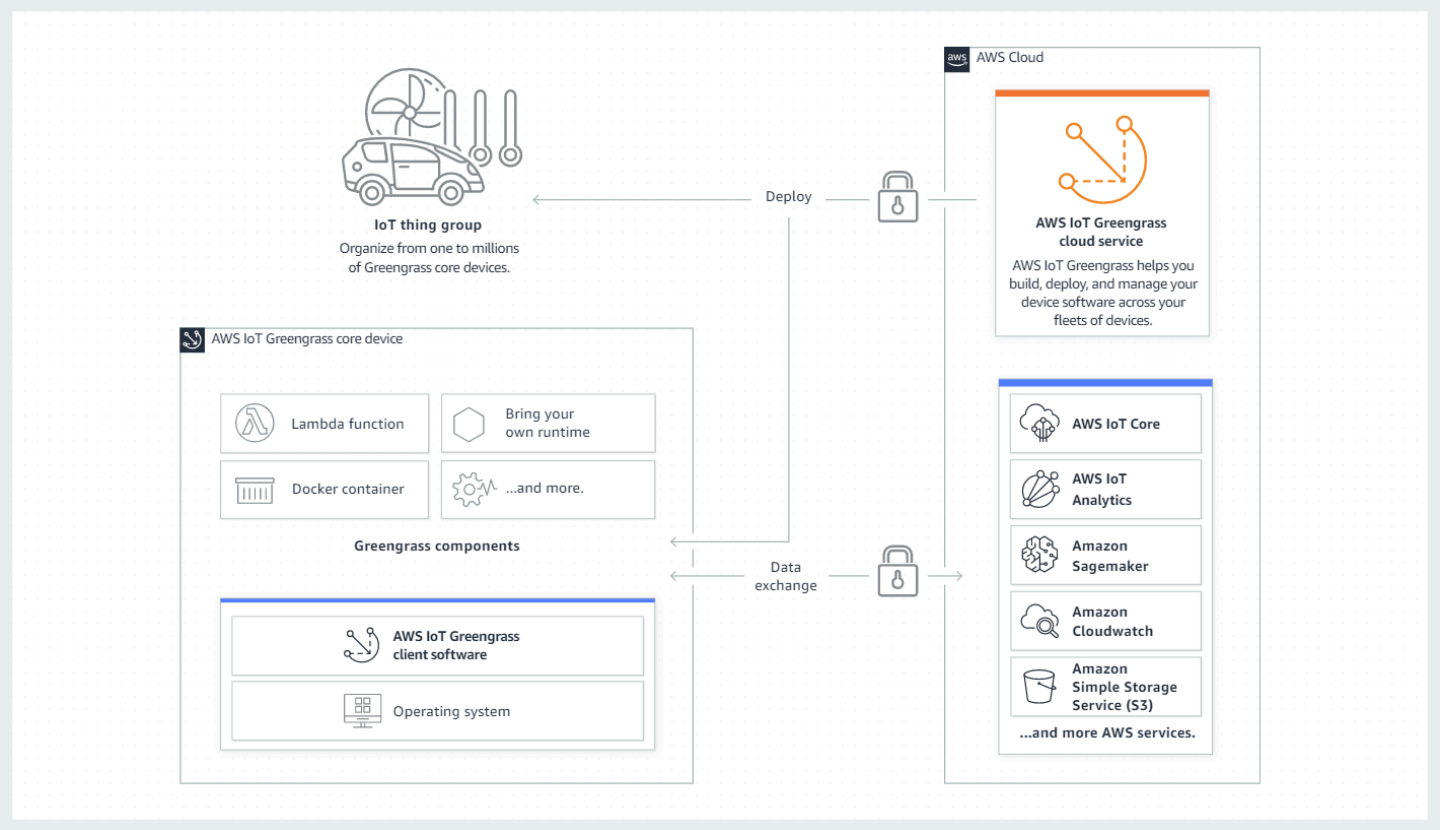
- 從AWS IoT Greengrass版本 1 遷移

AWS IoT Greengrass 的運作方式

AWS IoT Greengrass客戶端軟件，也稱為AWS IoT Greengrass核心軟件，運行在 Windows 和基於 Linux 的發行版，如 Ubuntu 或樹莓派操作系統，用於與 ARM 或 x86 架構的設備。使用時AWS IoT Greengrass，您可以對裝置進行程式設計，以根據其產生的資料在本機上採取行動、根據機器學習模型執行預測，以及篩選和彙總裝置資料。 AWS IoT Greengrass啟用本機執行AWS Lambda函數、Docker 容器、原生作業系統程序或您選擇的自訂執行階段。

AWS IoT Greengrass提供稱為組件的預構建軟件模塊，可讓您輕鬆擴展邊緣設備功能。 AWS IoT Greengrass元件可讓您連線至邊緣的AWS服務和協力廠商應用程式。開發 IoT 應用程式後，AWS IoT Greengrass可讓您在現場的裝置叢集上遠端部署、設定和管理這些應用程式。

下列範例顯示AWS IoT Greengrass裝置如何與中的AWS IoT Greengrass雲端服務和其他服AWS務互動AWS 雲端。



關鍵概念 AWS IoT Greengrass

以下是理解和使用的基本概念AWS IoT Greengrass：

AWS IoT事情

AWS IoT物件是特定裝置或邏輯實體的表示方式。物件的相關資訊會儲存在AWS IoT登錄中。

Greengrass 核心設備

運行AWS IoT Greengrass核心軟件的設備。格 Greengrass 核心設備是 IoT 的東西。AWS您可以將多個核心裝置新增至AWS IoT物件群組，以建立和管理 Greengrass 核心裝置群組。如需詳細資訊，請參閱 設定AWS IoT Greengrass核心裝置。

客戶 Greengrass 設備

透過 MQTT 連接到 Greengrass 核心裝置並與之通訊的裝置。Greengrass 客戶端設備是一回事。AWS IoT核心裝置可以處理、篩選和彙總來自連線至該裝置的用戶端裝置的資料。您可以將核心裝置設定為在用戶端裝置、AWS IoT Core雲端服務和 Greengrass 元件之間轉送 MQTT 訊息。如需詳細資訊，請參閱 與本機 IoT 裝置互動。

用戶端裝置可以執行 FreeRTOS 或使用AWS IoT Device SDK或 Greengrass 探索 API 來取得可連線的核心裝置相關資訊。

Greengrass 組件

部署至 Greengrass 核心裝置並在其上執行的軟體模組。使AWS IoT Greengrass用開發和部署的所有軟體都會建模為元件。 AWS IoT Greengrass提供預先建置的公用元件，提供您可以在應用程式中使用的特性和功能。您也可以在本機裝置或雲端中開發自己的自訂元件。開發自訂元件之後，您可以使用AWS IoT Greengrass雲端服務將其部署到單一或多個核心裝置。您可以建立自訂元件，然後將該元件部署到核心裝置。當您這麼做時，核心裝置會下載下列資源以執行元件：

- 方法：JSON 或 YAML 檔案，透過定義元件詳細資訊、組態和參數來描述軟體模組。
- Artifact：定義將在您裝置上執行之軟體的原始程式碼、二進位檔案或指令碼。您可以從頭開始建立成品，也可以使用 Lambda 函數、Docker 容器或自訂執行階段建立元件。
- 相依性：元件之間的關係，可讓您強制執行相依元件的自動更新或重新啟動。例如，您可以擁有依賴於加密元件的安全郵件處理元件。如此可確保加密元件的任何更新都會自動更新並重新啟動郵件處理元件。

如需詳細資訊，請參閱 AWS-提供的組件 及 開發AWS IoT Greengrass元件。

部署

傳送元件並將所需元件組態套用至目標目標裝置的程序，該裝置可以是單一 Greengrass 核心裝置或一組 Greengrass 核心裝置。部署會自動將任何更新的元件組態套用至目標，並包含定義為相依性的任何其他元件。您也可以複製現有部署，以建立使用相同元件但部署到不同目標的新部署。部署是連續的，這表示您對部署的元件或元件組態所做的任何更新都會自動傳送至所有目的地目標。如需詳細資訊，請參閱 將AWS IoT Greengrass元件部署到裝置。

AWS IoT Greengrass核心軟體

您在核心裝置上安裝的所有AWS IoT Greengrass軟體集。 AWS IoT Greengrass核心軟件包括以下內容：

- 核心：此必要元件提供AWS IoT Greengrass核心軟體的最低功能。核心會管理其他元件的部署、協調和生命週期管理。它還可以促進單個設備上本地AWS IoT Greengrass組件之間的通信。如需詳細資訊，請參閱 Greengrass 核。
- 選用元件：這些可設定元件由您的邊緣裝置提供AWS IoT Greengrass並啟用其他功能。根據您的需求，您可以選擇要部署到裝置的選用元件，例如資料串流、本機機器學習推論或本機命令列介面。如需詳細資訊，請參閱 AWS-提供的組件。

您可以透過將新版本的元件部署到裝置來升級 AWS IoT Greengrass Core 軟體。

AWS IoT Greengrass 的功能

AWS IoT Greengrass Version 2由以下元素組成：

- 软件发行
  - Greengrass 核組件，這是核心軟件的最小安裝。AWS IoT Greengrass此元件管理 Greengrass 元件的部署、協調和生命週期管理。
  - 與服務、通訊協定和軟體整合的其他選AWS用元件。
  - Greengrass 開發工具，您可以使用它來建立、測試、建置、發佈和部署自訂 Greengrass 元件。
  - 其中包含自訂 Greengrass 元件的處理序間通訊 (IPC) 程式庫 AWS IoT Device SDK，以及用戶端裝置的 Greengrass 探索程式庫。
  - 串流管理員 SDK，可用來管理核心裝置上的資料串流。
- 雲端服務
  - AWS IoT Greengrass V2 API
  - AWS IoT Greengrass V2 主控台

AWS IoT Greengrass 核心軟體

您可以使用在邊緣裝置上執行的 AWS IoT Greengrass Core 軟體來執行下列動作：

- 透過自動匯出至AWS雲端處理本機裝置上的資料串流。如需詳細資訊，請參閱 管理核心裝 Greengrass 資料串流。
- Support AWS IoT 和組件之間的 MQTT 消息傳遞。如需詳細資訊，請參閱 發布/訂閱MQTT 訊 AWS IoT Core 息。
- 與透過 MQTT 連線和通訊的本機裝置進行互動。如需詳細資訊，請參閱 與本機 IoT 裝置互動。
- Support 組件之間的本地發布和訂閱消息傳遞。如需詳細資訊，請參閱 發佈/訂閱本地訊息。
- 部署和叫用元件和 Lambda 函數。如需詳細資訊，請參閱 將AWS IoT Greengrass元件部署到裝置。
- 管理元件生命週期，例如支援安裝和執行指令碼。如需詳細資訊，請參閱 AWS IoT Greengrass元件配方參考。
- 執行AWS IoT Greengrass核心軟體和自訂元件的安全 over-the-air (OTA) 軟體更新。如需詳細資訊，請參閱 更新AWS IoT Greengrass核心軟件（OTA） 及 將AWS IoT Greengrass元件部署到裝置。
- 提供本機密碼的安全加密儲存，並由元件控制存取。如需詳細資訊，請參閱 秘密經理。

- 通過設備身份驗證和授權來保護設備和AWS雲端之間的連接。如需詳細資訊，請參閱 AWS IoT Greengrass 的裝置身分驗證和授權。

您可以透過 AWS IoT Greengrass API 來設定和管理 Greengrass 核心裝置，以建立持續的軟體部署。如需詳細資訊，請參閱 將AWS IoT Greengrass元件部署到裝置。

某些功能僅在特定平台上受支援。如需詳細資訊，請參閱 通過操作系統的 Greengrass 功能兼容性。

如需有關支援平台、需求和下載的詳細資訊，請參閱設定AWS IoT Greengrass核心裝置。

下載此軟體，即表示您同意 Greengrass 核心軟體授權合約之規定。

通過操作系統的 Greengrass 功能兼容性

AWS IoT Greengrass 支持運行各種操作系統的設備。某些功能僅支援某些作業系統。使用下表瞭解哪些功能適用於每個受支援的作業系統。如需有關支援的作業系統、需求以及如何設定 Greengrass 核心裝置的詳細資訊，請參閱。設定AWS IoT Greengrass核心裝置

簡訊

功能	Linux	Windows
在和元件之間 AWS IoT 交換 MQTT 訊息	是	是
在元件之間交換本地發布/訂閱消息	是	是
透過 MQTT 與本地 IoT 裝置互動	是	是

功能	Linux	Windows
使用模組-RTU 元件與本機模組-RTU 裝置互動	 是	 否

安全

功能	Linux	Windows
通過設備身份驗證和授權保護連接	 是	 是
從中部署和存取安全、加密的密碼 AWS Secrets Manager	 是	 是
使用硬體安全模組 (HSM) 安全地儲存裝置的私密金鑰和憑證	 是	 否
稽核核心裝置 AWS IoT Device Defender	 是	 是
使用 AWS 認證與 AWS 服務互動	 是	 是

安裝

功能	Linux	Windows
AWS IoT Greengrass 以自動佈建進行安裝	 是	 是
AWS IoT Greengrass 以手動佈建進行安裝	 是	 是
AWS IoT Greengrass 以 AWS IoT 叢集佈建進行安裝	 是	 是
AWS IoT Greengrass 使用自定義配置插件安裝	 是	 是
使用預構建 AWS IoT Greengrass 的 Docker 映像在 Docker 容器中運行	 是	 否

遠端維護與更新

功能	Linux	Windows
執行安全的 over-the-air (OTA) 軟體更新	 是	 是

功能	Linux	Windows
管理核心裝置 AWS Systems Manager	 是	 否
透過 AWS IoT 安全通道 Connect 至核心裝置	 是	 否

機器學習

功能	Linux	Windows
使用 Amazon SageMaker 邊緣管理器執行機器學習推論	 是	 是
使用 Amazon Lookout for Vision 執行機器學習推論	 是	 否
使用 DLR 執行機器學習推論	 是	 是
使用執行機器學習推論 TensorFlow	 是	 是

元件特徵

功能	Linux	Windows
部署和叫用 Lambda 函數	 是	 否
在組件中運行碼頭容器	 是	 是
使用串流管理員處理和匯出大量資料串流	 是	 是
使用生命週期指令碼管理元件生命週期	 是	 是
與裝置陰影互動	 是	 是
將日誌上傳到 Amazon CloudWatch 日誌	 是	 是

功能	Linux	Windows
使用 CloudWatch 指標元件將資料上傳到 Amazon CloudWatch 指標	 是	 是
使用 Amazon SNS 元件將訊息發佈到亞馬遜簡單通知服務	 是	 否
使用串流管理員將資料發佈到 Amazon 資料 Firehose 交付串流	 是	 是
使用 Firehose 元件將資料發佈到 Amazon 資料 Firehose 交付串流	 是	 否
收集即時系統遙測指標並採取行動	 是	 是
設定元件程序的系統資源限制	 是	 否
暫停及繼續元件程序	 是	 否

功能	Linux	Windows
與 AWS IoT SiteWise 使用 AWS IoT SiteWise 組件集成	 是	 是
使用適用於 Kinesis 視訊串流元件的邊緣連接器，將視訊串流發佈到 Amazon Kinesis 影片串流	 是	 否

元件開發

功能	Linux	Windows
在核心裝置上本機開發元件	 是	 是
使用 AWS IoT Greengrass CLI 與核心裝置互動	 是	 是
使用本機偵錯主控台與核心裝置互動	 是	 是
在自訂組件中使用 Python AWS IoT Device SDK	 是	 是

功能	Linux	Windows
在自訂元件中使用 C++ AWS IoT Device SDK	 是	 是
在自訂元件中使用 Java AWS IoT Device SDK	 是	 是

設備認證

功能	Linux	Windows
用 AWS IoT Device Tester 於 AWS IoT Greengrass V2 驗證 IoT 裝置	 是	 是

在中有什麼新功能 AWS IoT Greengrass Version 2

AWS IoT Greengrass Version 2 是介紹下列功 AWS IoT Greengrass 能的主要版本：

- 出版商支援的元件 — AWS IoT Greengrass 現在提供發行商支援的元件。這些組件由第三方供應商開發，提供和服務。如需詳細資訊，請參閱 出版商支援的元件。
- 在 VPC 中操作 Greengrass 設備 — 現在可以在 VPC 中操作 Greengrass 核心設備。這使您可以在沒有公共互聯網訪問的情況下在 VPC 中執行部署。如需詳細資訊，請參閱 在 VPC 中操作AWS IoT Greengrass核心設備。
- Greengrass 測試框架（GTF）-GTF 現已推出。 AWS IoT Greengrass Version 2 GTF 是支援 end-to-end 自動化的建置區塊集合。它可讓 AWS IoT Greengrass Version 2 內部客戶使用與服務團隊相同的測試架構來進行軟體變更、自動接受和品質保證目的。有關更多信息，請參閱 Greengrass itHub 上的綠色測試框架。
- PSA 認證 — AWS IoT Greengrass 核心版本 2.7.0 及更高版本現已通過平台安全性架構 (PSA) 認證。如需詳細資訊，請參閱AWS IoT Greengrass 已通過 PSA 認證。

AWS IoT Greengrass 版本說明提供有關發行版本的詳細資訊， AWS IoT Greengrass 包括新功能、更新與改良功能，以及一般修正。 AWS IoT Greengrass 具有以下類型的發行版本：

- 的新功能發行 AWS IoT Greengrass
- AWS IoT Greengrass 核心軟體更新

本節包含所有 AWS IoT Greengrass V2 版本說明，最新優先，並包含重大功能變更和重大錯誤修正。如需其他次要修正的相關資訊，請參閱上的 aws-greeng rass 組織。 GitHub

版本備註

- 發行版本： AWS IoT Greengrass 核心 v2.12.4 軟體更新於 2024 年 4 月 2 日
- 發行版本： AWS IoT Greengrass 核心 v2.12.3 軟體更新於二零二四年三月二十七日
- 發行版本： AWS IoT Greengrass 核心 v2.12.2 軟體更新將於二零二四年二月十五日
- 發行版本：AWS IoT Greengrass核心 v2.12.1 軟體更新於二零二三年十二月八日
- 發行版本：AWS IoT Greengrass核心 v2.12.0 軟體更新於 2023 年 11 月 7 日
- 發行版本：AWS IoT Greengrass核心 v2.11.3 軟體更新於 2023 年 10 月 18 日
- 發行版本：AWS IoT Greengrass核心 v2.11.2 軟體更新於 2023 年 8 月 9 日
- 發行版本：AWS IoT Greengrass核心 v2.11.1 軟體更新於二零二三年七月二十一日

- 發行版本：AWS IoT Greengrass核心 v2.11.0 軟體更新於二零二三年六月二十八日
- 發行版本：AWS IoT Greengrass核心 v2.10.3 軟體更新於二零二三年六月二十一日
- 發行版本：AWS IoT Greengrass核心 v2.10.2 軟體更新於 2023 年 6 月 5 日
- 發行版本：AWS IoT Greengrass核心 v2.10.1 軟體更新於 2023 年 5 月 11 日
- 發行版本：AWS IoT Greengrass核心 v2.10.0 軟體更新將於 2023 年 5 月 9 日發行
- 發行版本：2023 年 4 月 20 日的AWS IoT Greengrass核心 v2.9.6 軟體更新
- 發行版本：AWS IoT Greengrass核心 v2.9.5 軟體更新於 2023 年 3 月 30 日
- 發行版本：AWS IoT Greengrass2023 年 2 月 24 日核心 v2.9.4 軟體更新
- 發行版本：AWS IoT Greengrass核心 v2.9.3 軟體更新於 2023 年 2 月 1 日
- 發行版本：AWS IoT Greengrass核心 v2.9.2 軟體更新於 2022 年 12 月 22 日
- 發行版本：AWS IoT Greengrass核心 v2.9.1 軟體更新（二零二二年十一月十八日）
- 發行版本：AWS IoT Greengrass核心軟體更新（二零二二年十一月十五日）
- 發行版本：AWS IoT Greengrass核心 v2.8.1 軟體更新（二零二二年十月十三日）
- 發行版本：2022 年 10 月 7 日AWS IoT Greengrass核心軟體更新
- 發行版本：2022 年 7 月 28 日AWS IoT Greengrass核心軟體更新
- 發行版本：2022 年 6 月 27 日AWS IoT Greengrass核心軟體更新
- 發行版本：2022 年 5 月 31 日AWS IoT Greengrass核心軟體更新
- 發行版本：2022 年 4 月 6 日AWS IoT Greengrass核心軟體更新
- 發行版本：AWS IoT Greengrass核心 v2.5.4 軟體更新於 2022 年 3 月 23 日
- 發行版本：AWS IoT Greengrass核心 v2.5.3 軟體更新於 2022 年 1 月 6 日
- 發行版本：AWS IoT Greengrass2021 年 12 月 3 日的核心 v2.5.2 軟體更新
- 發行版本：AWS IoT Greengrass2021 年 11 月 23 日核心 v2.5.1 軟體更新
- 發行版本：AWS IoT Greengrass2021 年 11 月 12 日核心 v2.5.0 軟體更新
- 發行版本：AWS IoT Greengrass2021 年 8 月 3 日核心 v2.4.0 軟體更新
- 發行版本：AWS IoT Greengrass2021 年 6 月 29 日核心 v2.3.0 軟體更新
- 發行版本：AWS IoT Greengrass2021 年 6 月 18 日核心 v2.2.0 軟體更新
- 發行版本：AWS IoT Greengrass2021 年 4 月 26 日核心 v2.1.0 軟體更新
- 發行版本：AWS IoT Greengrass2021 年 3 月 9 日核心 v2.0.5 軟體更新
- 發行版本：AWS IoT Greengrass2021 年 2 月 4 日核心 v2.0.4 軟體更新

發行版本： AWS IoT Greengrass 核心 v2.12.4 軟體更新於 2024 年 4 月 2 日

此發行版本提供 Greengrass 核心元件的 2.12.4 版，以及提供元件的更新。 AWS

發行日期：二零二四年四月二日

版本詳情

- 公用元件更新

公用元件更新

下表列出由提供的元件，其中 AWS 包括新功能和更新的特徵。

> **Important**
>
> 當您部署元件時， AWS IoT Greengrass 會安裝該元件所有相依性的最新受支援版本。因此，如果您將新裝置新增至物件群組，或更新以這些裝置為目標的部署，則 AWS提供之公用元件的新修補程式版本可能會自動部署到核心裝置。某些自動更新 (例如核心更新) 可能會導致裝置意外重新啟動。
> 若要避免對裝置上執行的元件進行意外更新，建議您在建立部署時直接包含該元件的偏好版本。如需有關 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱更新AWS IoT Greengrass核心軟件（OTA）。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.12.4 版本是可用的。 錯誤修復和改進 • 修正在某些 Linux 裝置上啟動時，核心會進入鎖死狀態的問題。

發行版本： AWS IoT Greengrass 核心 v2.12.3 軟體更新於二零二四年三月二十七日

此發行版本提供 Greengrass 核心元件的 2.12.3 版，以及提供元件的更新。 AWS

發行日期：二零二四年三月二十七日

版本詳情

- 公用元件更新

公用元件更新

下表列出由提供的元件，其中 AWS 包括新功能和更新的特徵。

> **Important**
>
> 當您部署元件時， AWS IoT Greengrass 會安裝該元件所有相依性的最新受支援版本。因此，如果您將新裝置新增至物件群組，或更新以這些裝置為目標的部署，則 AWS提供之公用元件的新修補程式版本可能會自動部署到核心裝置。某些自動更新 (例如核心更新) 可能會導致裝置意外重新啟動。
> 若要避免對裝置上執行的元件進行意外更新，建議您在建立部署時直接包含該元件的偏好版本。如需有關 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱更新AWS IoT Greengrass核心軟件（OTA）。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.12.3 版本是可用的。 錯誤修復和改進 • 修復了核重新啟動和元件恢復期間，原子核無法報告正確組件狀態的問題。 • 一般錯誤修正與改進。
陰影管理	陰影管理器組件的 2.3.7 版可用。 錯誤修復和改進 修正陰影管理員在陰影管理員同步處理期間定期記錄`NullPointerException` 錯誤的問題。
叢集佈建	AWS IoT 車隊配置插件的 1.2.1 版可用。

元件	詳細資訊
	錯誤修復和改進 修正了在 Greengrass 核心啟動期間，叢集佈建外掛程式處於離線狀態的問題。車隊佈建外掛程式現在可以無限期地重試 MQTT 連線呼叫。
IP 偵測器	磁碟多工緩衝處理程式元件的 2.1.9 版可用。 錯誤修復和改進 將取得的 IP 步驟調整為僅在偵錯記錄檔層級傳送記錄檔。
轉換 MQTT 3.1.1 代理組件	模特 MQTT 3.1.1 代理組件的版本 2.3.6 是可用的。 錯誤修復和改進 一般錯誤修正與改進。
Lambda 經理	您可以使用 Lambda 管理員元件 2.3.3 版本。 錯誤修復和改進 一般錯誤修正與改進。
本機除錯主控台	本地調試控制台組件的 2.4.2 版本可用。 錯誤修復和改進 一般錯誤修正與改進。

發行版本： AWS IoT Greengrass 核心 v2.12.2 軟體更新將於二零二四年二月十五日

此版本提供 Greengrass 核心元件的 2.12.2 版，以及提供元件的更新。 AWS

發行日期：二零二四年二月十五日

版本詳情

- 公用元件更新

公用元件更新

下表列出由提供的元件，其中 AWS 包括新功能和更新的特徵。

> **Important**
>
> 當您部署元件時， AWS IoT Greengrass 會安裝該元件所有相依性的最新受支援版本。因此，如果您將新裝置新增至物件群組，或更新以這些裝置為目標的部署，則 AWS提供之公用元件的新修補程式版本可能會自動部署到核心裝置。某些自動更新 (例如核心更新) 可能會導致裝置意外重新啟動。
>
> 若要避免對裝置上執行的元件進行意外更新，建議您在建立部署時直接包含該元件的偏好版本。如需有關 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱更新AWS IoT Greengrass核心軟件（OTA）。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.12.2 版本是可用的。 錯誤修復和改進 • 修正舊記錄未正確清理的問題。 • 一般錯誤修正與改進。
陰影管理	陰影管理器組件的 2.3.6 版可用。 錯誤修復和改進 修正當裝置離線時透過 AWS 雲端 更新刪除的陰影屬性在重新取得連線後，仍會繼續存在於本機陰影中的問題。
发 Lambda 器	可以使用 lambda 啟動器組件的 2.0.13 版本。 錯誤修復和改進 一般錯誤修正與改進。
磁碟後臺解決程式	磁盤後台處理程序組件的版本 1.0.3 可用。

元件	詳細資訊
	錯誤修復和改進 透過重複使用資料庫連線來改善效能。

發行版本：AWS IoT Greengrass核心 v2.12.1 軟體更新於二零二三年十二月八日

此版本提供 Greengrass 核心元件的 2.12.1 版，以及提供元件的更新。AWS

發行日期：二零二三年十二月八日

版本詳情

- 公用元件更新

公用元件更新

下表列出由提供的元件，其中AWS包括新功能和更新的特徵。

> **Important**
>
> 當您部署元件時，AWS IoT Greengrass會安裝該元件所有相依性的最新受支援版本。因此，如果您將新裝置新增至物件群組，或更新以這些裝置為目標的部署，則AWS提供之公用元件的新修補程式版本可能會自動部署到核心裝置。某些自動更新 (例如核心更新) 可能會導致裝置意外重新啟動。
> 若要避免對裝置上執行的元件進行意外更新，建議您在建立部署時直接包含該元件的偏好版本。如需有關 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱更新AWS IoT Greengrass核心軟件（OTA）。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.12.1 版本是可用的。

元件	詳細資訊
	錯誤修復和改進 • 修正核心可能會將 MQTT 訂閱複製到部署主題，導致其他記錄和 MQTT 發佈的問題。
用戶端裝置驗證	客戶端設備身份驗證組件的 2.4.5 版可用。 新功能 新增萬用字元前置字元以使用`selectionRule` 參數選取物件名稱的支援。 錯誤修復和改進 修正在某些情況下，憑證不會以新連線資訊更新的問題。
磁碟後臺解決程式	磁盤後台處理程序組件的 1.0.2 版本可用。 錯誤修復和改進 修正 MQTT 郵件格式欄位在特定情況下不會持續存在的問題。
MQTT 大橋	磁碟多工緩衝處理程式元件的 2.3.1 版可用。 錯誤修復和改進 修正本機 MQTT 用戶端進入中斷連線迴圈的問題。
流管理器	流管理器組件的版本 2.1.12 可用。 錯誤修復和改進 更新證明資料的使用順序，讓 Greengrass 證明資料偏好用於服務要求 AWS。

發行版本：AWS IoT Greengrass核心 v2.12.0 軟體更新於 2023 年 11 月 7 日

此版本提供 Greengrass 核心元件的 2.12.0 版，以及提供元件的更新。AWS

發行日期：二○二三年十一月七日

發行亮點

- 引導回滾-AWS IoT Greengrass 現在提供了一個名為 Greengrass 核配置參數。`BootstrapOnRollback`此功能可讓您在復原部署中執行啟動程序生命週期步驟。

版本詳情

- 公用元件更新

公用元件更新

下表列出由提供的元件，其中AWS包括新功能和更新的特徵。

> **Important**
>
> 當您部署元件時，AWS IoT Greengrass會安裝該元件所有相依性的最新受支援版本。因此，如果您將新裝置新增至物件群組，或更新以這些裝置為目標的部署，則AWS提供之公用元件的新修補程式版本可能會自動部署到核心裝置。某些自動更新 (例如核心更新) 可能會導致裝置意外重新啟動。
> 若要避免對裝置上執行的元件進行意外更新，建議您在建立部署時直接包含該元件的偏好版本。如需有關 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱更新AWS IoT Greengrass核心軟件（OTA）。

元件	詳細資訊
Greengrass 核	Greengr ass 核的版本 2.12.0 是可用的。 新功能 • 可讓您執行啟動程序生命週期步驟，做為復原部署的一部分。

發行版本：AWS IoT Greengrass核心 v2.11.3 軟體更新於 2023 年 10 月 18 日

此版本提供了 Greengrass 核成分的 2.11.3 版本。

發佈日期：2023 年 10 月 18 日

版本詳情

- 公用元件更新

公用元件更新

下表列出由提供的元件，其中AWS包括新功能和更新的特徵。

> **Important**
>
> 當您部署元件時，AWS IoT Greengrass會安裝該元件所有相依性的最新受支援版本。因此，如果您將新裝置新增至物件群組，或更新以這些裝置為目標的部署，則AWS提供之公用元件的新修補程式版本可能會自動部署到核心裝置。某些自動更新 (例如核心更新) 可能會導致裝置意外重新啟動。
> 若要避免對裝置上執行的元件進行意外更新，建議您在建立部署時直接包含該元件的偏好版本。如需有關 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱更新AWS IoT Greengrass核心軟件（OTA）。

元件	詳細資訊
Greengrass 核	Greengrass 核的版本 2.11.3 是可用的。 錯誤修復和改進 • 修正核心中的問題，即當元件的相依性失敗時，可能不正確地啟動元件。 新功能 • 添加可配置的 s3 端點類型。
Lambda 經理	您可以使用 Lambda 管理員元件 2.3.1 版。 錯誤修復和改進 • 調整某些錯誤的日誌級別。
本地控制台	您可以使用 Lambda 管理員元件 2.4.0 版。

元件	詳細資訊
	新功能 • 添加流管理器調試控制台。
日誌管理器	日誌管理器組件的 2.3.6 版可用。 錯誤修復和改進 • 調整某些錯誤的日誌級別。
陰影管理員	版本 2.3.4 的陰影管理器組件可用。 錯誤修復和改進 • 添加對空白和空陰影狀態文檔的支持。

發行版本：AWS IoT Greengrass核心 v2.11.2 軟體更新於 2023 年 8 月 9 日

此版本提供了 Greengrass 核成分的 2.11.2 版本。

發行日期：二零二三年八月九日

版本詳情

- 公用元件更新

公用元件更新

下表列出由提供的元件，其中AWS包括新功能和更新的特徵。

> **Important**
>
> 當您部署元件時，AWS IoT Greengrass會安裝該元件所有相依性的最新受支援版本。因此，如果您將新裝置新增至物件群組，或更新以這些裝置為目標的部署，則AWS提供之公用元件的新修補程式版本可能會自動部署到核心裝置。某些自動更新 (例如核心更新) 可能會導致裝置意外重新啟動。

若要避免對裝置上執行的元件進行意外更新，建議您在建立部署時直接包含該元件的偏好版本。如需有關 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱更新AWS IoT Greengrass核心軟件（OTA）。

元件	詳細資訊
Greengrass 核	Greengrass 核的版本 2.11.2 是可用的。 錯誤修復和改進 • 修正核心 MQTT 5 用戶端中的問題，在使用大量 (> 50) 訂閱時，該用戶端可能會顯示為離線狀態。 • 添加 docker 撥號 TCP 失敗的重試。

發行版本：AWS IoT Greengrass核心 v2.11.1 軟體更新於二零二三年七月二十一日

此版本提供了 Greengrass 核成分的 2.11.1 版。

發行日期：二零二三年七月二十一日

版本詳情

- 公用元件更新

公用元件更新

下表列出由提供的元件，其中AWS包括新功能和更新的特徵。

Important

當您部署元件時，AWS IoT Greengrass會安裝該元件所有相依性的最新受支援版本。因此，如果您將新裝置新增至物件群組，或更新以這些裝置為目標的部署，則AWS提供之公用元件的新修補程式版本可能會自動部署到核心裝置。某些自動更新 (例如核心更新) 可能會導致裝置意外重新啟動。

若要避免對裝置上執行的元件進行意外更新，建議您在建立部署時直接包含該元件的偏好版本。如需有關 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱更新AWS IoT Greengrass核心軟件（OTA）。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.11.1 版本是可用的。 錯誤修復和改進 • 修正啟動程序工作失敗且部署中繼資料檔案損毀時，核無法啟動的問題。 • 修正部署狀態更新中未報告隨選 Lambda 元件的問題。 • 添加對重複授權策略 ID 的支援。
Lambda 經理	您可以使用 Lambda 管理員 2.2.11 版。 錯誤修復和改進 • 修正 Lambda LegacySubscriptionRouter 組態變更時組態不會更新的問題。

發行版本：AWS IoT Greengrass核心 v2.11.0 軟體更新於二零二三年六月二十八日

此版本提供了 Greengrass 核成分的 2.11.0 版本。

發行日期：二零二三年六月二十八日

發行亮點

- 永久磁碟多工緩衝處理程式 — AWS IoT Greengrass 現在為 Greengrass 核心裝置多工緩衝處理的訊息提供持久的多工緩衝處理程式實作。AWS IoT Core此元件會將這些輸出郵件儲存在磁碟上。如需詳細資訊，請參閱 磁碟後臺解決程式。
- 本機部署改進 — 您現在可以取消本機部署、設定部署失敗處理原則，以及取得詳細的部署狀態。
- 記錄速度改進 — 已改善記錄管理員元件的記錄檔上傳速度。

版本詳情

公用元件更新

下表列出由提供的元件，其中AWS包括新功能和更新的特徵。

> **Important**
>
> 當您部署元件時，AWS IoT Greengrass會安裝該元件所有相依性的最新受支援版本。因此，如果您將新裝置新增至物件群組，或更新以這些裝置為目標的部署，則AWS提供之公用元件的新修補程式版本可能會自動部署到核心裝置。某些自動更新 (例如核心更新) 可能會導致裝置意外重新啟動。
> 若要避免對裝置上執行的元件進行意外更新，建議您在建立部署時直接包含該元件的偏好版本。如需有關 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱更新AWS IoT Greengrass核心軟件（OTA）。

元件	詳細資訊
Greengrass 核	Greengr ass 核的版本 2.11.0 是可用的。 新功能 • 可讓您取消本機部署。 • 可讓您設定本機部署的失敗處理原則。 • 添加對磁盤後台處理程序插件的支持。
Greengrass CLI	Greengr ass CLI 的 2.11.0 版本是可用的。 新功能 • 可讓您取消本機部署。 • 可讓您設定本機部署的失敗處理原則。 • 改進了詳細的部署狀態報告。
磁碟後臺解決程式	磁盤後台處理程序組件的版本 1.0.0 可用。 • 磁碟多工緩衝處理程式元件提供永久儲存從 Greengrass 核心裝置傳送至的訊息。AWS IoT Core

元件	詳細資訊
日誌管理器	日誌管理器組件的 2.3.5 版可用。 改善項目 提高了日誌上傳速度。

發行版本：AWS IoT Greengrass核心 v2.10.3 軟體更新於二零二三年六月二十一日

此版本提供了 Greengrass 核成分的 2.10.3 版本。

發行日期：二零二三年六月二十一日

版本詳情

- 公用元件更新

公用元件更新

下表列出由提供的元件，其中AWS包括新功能和更新的特徵。

> **Important**
>
> 當您部署元件時，AWS IoT Greengrass會安裝該元件所有相依性的最新受支援版本。因此，如果您將新裝置新增至物件群組，或更新以這些裝置為目標的部署，則AWS提供之公用元件的新修補程式版本可能會自動部署到核心裝置。某些自動更新 (例如核心更新) 可能會導致裝置意外重新啟動。
> 若要避免對裝置上執行的元件進行意外更新，建議您在建立部署時直接包含該元件的偏好版本。如需有關 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱更新AWS IoT Greengrass核心軟件（OTA）。

元件	詳細資訊
Greengrass 核	Greengrass 核的版本 2.10.3 是可用的。

元件	詳細資訊
	錯誤修復和改進 • 修正使用 PKCS #11 提供者時 Greengrass 未訂閱部署通知的問題。

發行版本：AWS IoT Greengrass核心 v2.10.2 軟體更新於 2023 年 6 月 5 日

此版本提供了 Greengrass 核成分的 2.10.2 版。

發行日期：二零二三年六月五日

版本詳情

- 公用元件更新

公用元件更新

下表列出由提供的元件，其中AWS包括新功能和更新的特徵。

> **Important**
>
> 當您部署元件時，AWS IoT Greengrass會安裝該元件所有相依性的最新受支援版本。因此，如果您將新裝置新增至物件群組，或更新以這些裝置為目標的部署，則AWS提供之公用元件的新修補程式版本可能會自動部署到核心裝置。某些自動更新 (例如核心更新) 可能會導致裝置意外重新啟動。
> 若要避免對裝置上執行的元件進行意外更新，建議您在建立部署時直接包含該元件的偏好版本。如需有關 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱更新AWS IoT Greengrass核心軟件（OTA）。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.10.2 版本是可用的。 錯誤修復和改進 • 允許對元件生命週期進行不區分大小寫的剖析。

元件	詳細資訊
	• 修正無法正確重新建立環境 PATH 變數的問題。 • 修復了組件的代理 URI 編碼，包括具有特殊字符的用戶名的流管理器。
用戶端裝置驗證	客戶端設備身份驗證組件的 2.4.2 版本可用。 新功能 添加一個新的`startupTimeoutSeconds` 配置選項。
Lambda 經理	您可以使用 Lambda 管理員元件 2.2.9 版。 錯誤修復和改進 修復了端口號由於時鐘偏斜而損壞的問題。
日誌管理器	日誌管理器組件的 2.3.4 版可用。 錯誤修復和改進 • 添加對將參數設置為分`periodicUploadIntervalSec` 數值的支援。最小值為 1 微秒。 • 修正記錄管理員不遵守 CloudWatch`putLogEvents` 限制的問題。
MQTT 3.1 經紀商（汽車）	MQTT 3.1 代理程式 (Moquette) 元件的 2.3.3 版本可供使用。 新功能 添加一個新的`startupTimeoutSeconds` 配置選項。
MQTT 大橋	MQTT 橋接器元件的 2.2.6 版可供使用。 新功能 添加一個新的`startupTimeoutSeconds` 配置選項。
流管理器	流管理器組件的 2.1.7 版本可用。 錯誤修復和改進 修正串流管理員無法正確讀取 Proxy 組態的問題。

發行版本：AWS IoT Greengrass核心 v2.10.1 軟體更新於 2023 年 5 月 11 日

此版本提供了 Greengrass 核成分的 2.10.1 版本。

發行日期：二零二三年五月十一日

版本詳情

- 公用元件更新

公用元件更新

下表列出由提供的元件，其中AWS包括新功能和更新的特徵。

> **Important**
>
> 當您部署元件時，AWS IoT Greengrass會安裝該元件所有相依性的最新受支援版本。因此，如果您將新裝置新增至物件群組，或更新以這些裝置為目標的部署，則AWS提供之公用元件的新修補程式版本可能會自動部署到核心裝置。某些自動更新 (例如核心更新) 可能會導致裝置意外重新啟動。
> 若要避免對裝置上執行的元件進行意外更新，建議您在建立部署時直接包含該元件的偏好版本。如需有關 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱更新AWS IoT Greengrass核心軟件（OTA）。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.10.1 版本是可用的。 錯誤修復和改進 • 修正了在某些 ARMv8 處理器（包括 Jetson Nano）上啟動時可能導致當機的問題。 • Greengrass 不再關閉中的組件的標準，這會將行為恢復為 2.10.0 之前的行為
流管理器	新的流管理器的 2.1.6 版本可用。

元件	詳細資訊
	錯誤修復和改進 修正了在某些 ARMv8 處理器（包括 Jetson Nano）上啟動時可能導致當機的問題。

發行版本：AWS IoT Greengrass核心 v2.10.0 軟體更新將於 2023 年 5 月 9 日發行

此版本提供 Greengrass 核心元件的 2.10.0 版，以及提供元件的更新。AWS

發行日期：二零二三年五月九日

發行亮點

- MQTT5 支援 — AWS IoT Greengrass 現在支援AWS IoT Core使用 MQTT5 傳送和接收訊息。如需詳細資訊，請參閱發佈 AWS IoT Core MQTT 訊息。

版本詳情

- 公用元件更新

公用元件更新

下表列出由提供的元件，其中AWS包括新功能和更新的特徵。

> **Important**
>
> 當您部署元件時，AWS IoT Greengrass會安裝該元件所有相依性的最新受支援版本。因此，如果您將新裝置新增至物件群組，或更新以這些裝置為目標的部署，則AWS提供之公用元件的新修補程式版本可能會自動部署到核心裝置。某些自動更新 (例如核心更新) 可能會導致裝置意外重新啟動。
> 若要避免對裝置上執行的元件進行意外更新，建議您在建立部署時直接包含該元件的偏好版本。如需有關 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱更新AWS IoT Greengrass核心軟件（OTA）。

元件	詳細資訊
Greengrass 核	Greengr ass 核的版本 2.10.0 是可用的。 新功能 • 添加`interpolateComponentConfiguration` 對空的正則表達式的支持。Greengrass 現在從根配置對象進行插值。 • 添加對 MQTT5 的支持。 • 添加了一種機制，無需掃描即可快速加載插件組件。 • 啟用 Greengrass 通過刪除未使用的 Docker 映像來節省磁盤空間。 錯誤修復和改進 • 修正復原會保留部署中某些組態值的問題。 • 修正 Greengrass 核心驗證自訂非認證和資料端點中AWS網域序列的問題。AWS • 更新多群組相依性解析，以透過AWS 雲端交涉重新解析所有群組相依性，而不是鎖定至使用中版本。此更新也會移除部署錯誤程式碼`INSTALLED_COMPONENT_NOT_FOUND` 。 • 更新 Greengrass 核心，以便在本機已經存在 Docker 映像檔時略過下載。 • 更新 Greengrass 核心，以在逾時到期之前重新啟動元件安裝步驟。 • 其他小修正和改進。
陰影管理員	新的陰影管理器 2.3.2 版可用。 錯誤修復和改進 修正當本機陰影資料庫損毀時，陰影管理員進入`BROKEN`狀態的問題。

發行版本：2023 年 4 月 20 日的AWS IoT Greengrass核心 v2.9.6 軟體更新

此版本提供了 Greengrass 核成分的 2.9.6 版。

發佈日期：2023 年 4 月 20 日

版本詳情

- 公用元件更新

公用元件更新

下表列出由提供的元件，其中AWS包括新功能和更新的特徵。

> **Important**
>
> 當您部署元件時，AWS IoT Greengrass會安裝該元件所有相依性的最新受支援版本。因此，如果您將新裝置新增至物件群組，或更新以這些裝置為目標的部署，則AWS提供之公用元件的新修補程式版本可能會自動部署到核心裝置。某些自動更新 (例如核心更新) 可能會導致裝置意外重新啟動。
> 若要避免對裝置上執行的元件進行意外更新，建議您在建立部署時直接包含該元件的偏好版本。如需有關 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱更新AWS IoT Greengrass核心軟件（OTA）。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.9.6 版本是可用的。 錯誤修復和改進 • 修正了 Greengrass 部署失敗並顯示錯誤發生錯誤，且後續裝置重新開機無法啟動 Greengrass 的問題。當您在需要 Greengrass 重新啟動的部署的多個物件群組之間移動 Greengrass 裝置時，可能會發生此錯誤。

發行版本：AWS IoT Greengrass核心 v2.9.5 軟體更新於 2023 年 3 月 30 日

此版本提供了 Greengrass 核成分的 2.9.5 版本。

發行日期：二零二三年三月三十日

版本詳情

- 公用元件更新

公用元件更新

下表列出由提供的元件，其中AWS包括新功能和更新的特徵。

> **Important**
>
> 當您部署元件時，AWS IoT Greengrass會安裝該元件所有相依性的最新受支援版本。因此，如果您將新裝置新增至物件群組，或更新以這些裝置為目標的部署，則AWS提供之公用元件的新修補程式版本可能會自動部署到核心裝置。某些自動更新 (例如核心更新) 可能會導致裝置意外重新啟動。
>
> 若要避免對裝置上執行的元件進行意外更新，建議您在建立部署時直接包含該元件的偏好版本。如需有關 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱更新AWS IoT Greengrass核心軟件（OTA）。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.9.5 版本是可用的。 新功能 • 添加對 Greengrass 核軟體簽名驗證的支援。 錯誤修復和改進 • 修正當本機方案中繼資料區域與 Greengrass 核心啟動區域不相符時，部署失敗的問題。發生這種情況時，Greengrass 核現在會與雲端重新協商。 • 修正 MQTT 訊息多工緩衝處理程式填滿且永不移除訊息的問題。 • 其他小的修正和改進。

發行版本：AWS IoT Greengrass2023 年 2 月 24 日核心 v2.9.4 軟體更新

此版本提供了 Greengrass 核成分的 2.9.4 版本。

發行日期：二零二三年二月二十四日

版本詳情

公用元件更新

下表列出由提供的元件，其中AWS包括新功能和更新的特徵。

> **Important**
>
> 當您部署元件時，AWS IoT Greengrass會安裝該元件所有相依性的最新受支援版本。因此，如果您將新裝置新增至物件群組，或更新以這些裝置為目標的部署，則AWS提供之公用元件的新修補程式版本可能會自動部署到核心裝置。某些自動更新 (例如核心更新) 可能會導致裝置意外重新啟動。
>
> 若要避免對裝置上執行的元件進行意外更新，建議您在建立部署時直接包含該元件的偏好版本。如需有關 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱更新AWS IoT Greengrass核心軟件（OTA）。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.9.4 版本是可用的。 錯誤修復和改進 • 在丟棄 QOS 0 訊息之前檢查是否有空訊息。 • 如果工作狀態詳細資料值超過 1024 個字元的限制，則截斷它們。 • 如果該路徑包含空格，則更新 Windows 的啟動程序檔，以正確讀取 Greengrass 根路徑。 • 訂閱的更新，以AWS IoT Core便在未傳送訂閱回應時捨棄用戶端訊息。 • 確保當主配置文件損壞或丟失時，核從備份文件加載其配置。

發行版本：AWS IoT Greengrass核心 v2.9.3 軟體更新於 2023 年 2 月 1 日

此版本提供了 Greengrass 核成分的 2.9.3 版。

發行日期：二零二三年二月一日

版本詳情

- 公用元件更新

公用元件更新

下表列出由提供的元件，其中AWS包括新功能和更新的特徵。

> **Important**
>
> 當您部署元件時，AWS IoT Greengrass會安裝該元件所有相依性的最新受支援版本。因此，如果您將新裝置新增至物件群組，或更新以這些裝置為目標的部署，則AWS提供之公用元件的新修補程式版本可能會自動部署到核心裝置。某些自動更新 (例如核心更新) 可能會導致裝置意外重新啟動。
> 若要避免對裝置上執行的元件進行意外更新，建議您在建立部署時直接包含該元件的偏好版本。如需有關 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱更新AWS IoT Greengrass核心軟件（OTA）。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.9.3 版本是可用的。 錯誤修復和改進 • 確保 MQTT 用戶端 ID 不會重複。 • 添加更強大的文件讀取和寫入，以避免損壞和恢復。 • 在特定的網絡相關錯誤上重試 docker 映像提取。 • 添加 MQTT 連接的`noProxyAddresses` 選項。

發行版本：AWS IoT Greengrass核心 v2.9.2 軟體更新於 2022 年 12 月 22 日

此版本提供了 Greengrass 核成分的 2.9.2 版本。

發行日期：二 ○○ 二年十二月二十二

版本詳情

- 公用元件更新

公用元件更新

下表列出AWS提供的元件，其中包括新功能和更新的功能。

> **Important**
>
> 當您部署元件時，AWS IoT Greengrass會安裝該元件所有相依性的最新受支援版本。因此，如果您將新裝置新增至物件群組，或更新以這些裝置為目標的部署，則AWS提供之公用元件的新修補程式版本可能會自動部署到核心裝置。某些自動更新 (例如核心更新) 可能會導致裝置意外重新啟動。
> 若要避免對裝置上執行的元件進行意外更新，建議您在建立部署時直接包含該元件的偏好版本。如需有關 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱更新AWS IoT Greengrass核心軟件（OTA）。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.9.2 版本是可用的。 錯誤修復和改進 • 修正設定`interpolateComponentConfiguration`不適用於進行中部署的問題。 • 使用 OSHI 列出所有子進程。

發行版本：AWS IoT Greengrass核心 v2.9.1 軟體更新（二零二二年十一月十八日）

此版本提供 Greengrass 核心元件的 2.9.1 版，以及提供元件的更新。AWS

版本日期：2022 年 11 月 18 日

發行亮點

- 日誌管理器-日誌管理器現在處理和直接上傳活動的日誌文件，而不是等待新的文件被輪換。這項改善可大幅減少記錄延遲。如需詳細資訊，請參閱記錄管理員

版本詳情

- 公用元件更新

公用元件更新

下表列出AWS提供的元件，其中包括新功能和更新的功能。

> **Important**
>
> 當您部署元件時，AWS IoT Greengrass會安裝該元件所有相依性的最新受支援版本。因此，如果您將新裝置新增至物件群組，或更新以這些裝置為目標的部署，則AWS提供之公用元件的新修補程式版本可能會自動部署到核心裝置。某些自動更新 (例如核心更新) 可能會導致裝置意外重新啟動。
> 若要避免對裝置上執行的元件進行意外更新，建議您在建立部署時直接包含該元件的偏好版本。如需有關 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱更新AWS IoT Greengrass核心軟件（OTA）。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.9.1 版本是可用的。 錯誤修復和改進 • 添加修復，如果部署刪除插件組件，Greengrass 重新啟動。
日誌管理器	新的日誌管理器版本 2.3.0 可用。 **Note** 我們建議您升級至記錄檔管理員 2.3.0 時，升級至 Greengrass 核心 2.9.1。

元件	詳細資訊
	新功能 • 透過處理和直接上傳作用中的記錄檔，而不是等待輪換新檔案來減少記錄延遲。 錯誤修復和改進 • 改進了旋轉具有唯一名稱的文件時對日誌旋轉的支持。 • 其他小修正和改進。

發行版本：AWS IoT Greengrass核心軟體更新（二零二二年十一月十五日）

此發行版本提供 Greengrass 核元件的 2.9.0 版，以及對所提供元件的更新。AWS

發行日期：二 ○○ 二年十一月十五日

發行亮點

- 離線驗證 — AWS IoT Greengrass 現在支援離線驗證。您可以配置AWS IoT Greengrass核心設備，以便客戶端設備可以連接到核心設備，即使核心設備未連接到雲端也是如此。如需詳細資訊，請參閱離線驗證。
- 子部署 — 您現在可以建立子部署。您可以使用子部署來解決不成功的部署。每個子部署都可以在較小的裝置子集上測試未成功部署的不同組態。如需詳細資訊，請參閱建立子部署。

版本詳情

- 公用元件更新

公用元件更新

下表列出AWS提供的元件，其中包括新功能和更新的功能。

> **Important**
>
> 當您部署元件時，AWS IoT Greengrass會安裝該元件所有相依性的最新受支援版本。因此，如果您將新裝置新增至物件群組，或更新以這些裝置為目標的部署，則AWS提供之公用元件的新

修補程式版本可能會自動部署到核心裝置。某些自動更新 (例如核心更新) 可能會導致裝置意外重新啟動。
若要避免對裝置上執行的元件進行意外更新，建議您在建立部署時直接包含該元件的偏好版本。如需有關 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱更新AWS IoT Greengrass核心軟件（OTA）。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.9.0 版本是可用的。 新功能 • 新增建立子部署的功能，以便使用較小的裝置子集重試部署。此功能可提供更有效率的方式來測試和解決失敗的部署。 錯誤修復和改進 • 改善對沒有useradd、groupadd和的系統的支援usermod。 • 其他小修正和改進。
用戶端裝置驗證	客戶端設備身份驗證組件的 2.3.0 版本可用。 新功能 • 添加對客戶端設備的離線身份驗證的支持。透過此功能，當核心裝置未連線到網際網路時，用戶端裝置可以繼續連線至核心裝置。 • 新增對客戶提供的憑證授權單位 (CA) 的支援。您的核心裝置使用客戶提供的 CA 作為根憑證，以產生 MQTT 代理程式憑證。
MQTT 5 經紀商	該MQTT 5 經紀商組件的版本 1.2.0 可用。 新功能 添加對證書鏈的支持。
MQTT 經紀商模式	新的 Moquette MQTT 代理組件的 2.3.0 版本可用。 新功能 添加對證書鏈的支持。

元件	詳細資訊
秘密經理	新的密碼管理器的 2.1.4 版本可用。 錯誤修復和改進 修正部署秘密管理員且 Greengrass 核心重新啟動時，會移除快取密碼的問題。
流管理器	新的流管理器的 2.1.2 版本可用。 錯誤修復和改進 修正 Windows 作業系統上使用非英文語言的問題。

發行版本：AWS IoT Greengrass核心 v2.8.1 軟體更新（二零二二年十月十三日）

此版本提供了 Greengrass 核組件的 2.8.1 版本。

發行日期：二○二二年十月十三日

> **Note**
> 如果您使用的是 Greengrass 核 2.8.0 版，我們強烈建議您升級到 Greengrass 核 2.8.1 版本。

版本詳情

- 公用元件更新

公用元件更新

下表列出AWS提供的元件，其中包括新功能和更新的功能。

> **Important**
> 當您部署元件時，AWS IoT Greengrass會安裝該元件所有相依性的最新受支援版本。因此，如果您將新裝置新增至物件群組，或更新以這些裝置為目標的部署，則AWS提供之公用元件的新

修補程式版本可能會自動部署到核心裝置。某些自動更新 (例如核心更新) 可能會導致裝置意外重新啟動。
若要避免對裝置上執行的元件進行意外更新，建議您在建立部署時直接包含該元件的偏好版本。如需有關 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱更新AWS IoT Greengrass核心軟件（OTA）。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.8.1 版本是可用的。 錯誤修復和改進 • 修正無法從 Greengrass API 錯誤正確產生部署錯誤碼的問題。 • 修正當元件在部署期間達到某個狀態時，叢集狀態更新會傳送不正確資訊的問題。ERRORED • 修正 Greengrass 擁有超過 50 個現有訂閱時，部署無法完成的問題。

發行版本：2022 年 10 月 7 日AWS IoT Greengrass核心軟體更新

此版本提供了綠核元件的 2.8.0 版，以及 MQTT 5 代理程式 (EMQX) 元件的 1.1.0 版。

發行日期：二○二二年十月七日

發行亮點

- 部署錯誤代碼 — Greengrass 核心現在會報告部署健康狀態回應，當元件部署無法完成時，其中包含詳細的錯誤碼。如需詳細資訊，請參閱 詳細的部署錯誤代碼。
- 元件錯誤狀態 — Greengrass 核心現在會報告元件健康狀態回應，其中包含元件進入或狀態時的詳細錯誤狀態。BROKEN ERRORED如需詳細資訊，請參閱 詳細的元件狀態代碼。

版本詳情

- 公用元件更新

公用元件更新

下表列出AWS提供的元件，其中包括新功能和更新的功能。

Important

當您部署元件時，AWS IoT Greengrass會安裝該元件所有相依性的最新受支援版本。因此，如果您將新裝置新增至物件群組，或更新以這些裝置為目標的部署，則AWS提供之公用元件的新修補程式版本可能會自動部署到核心裝置。某些自動更新 (例如核心更新) 可能會導致裝置意外重新啟動。

若要避免對裝置上執行的元件進行意外更新，建議您在建立部署時直接包含該元件的偏好版本。如需有關 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱更新AWS IoT Greengrass核心軟件（OTA）。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.8.0 版本是可用的。 新功能 • 更新 Greengrass 核心以報告部署健康狀態回應，其中包含詳細的錯誤碼，當將元件部署到核心裝置時發生問題。如需詳細資訊，請參閱 詳細的部署錯誤代碼。 • 更新 Greengrass 核心以報告元件健康狀態回應，其中包含元件進入或狀態時的詳細錯誤代碼。BROKEN ERRORED如需詳細資訊，請參閱 詳細的元件狀態代碼。 • 擴展狀態訊息欄位，以改善裝置的雲端可用性資訊。 • 提高車隊狀態服務的穩健性。 錯誤修復和改進 • 允許損毀的元件在組態變更時重新安裝。 • 修正啟動程序部署期間核心重新啟動會導致部署失敗的問題。 • 修正 Windows 中當根路徑包含空格時安裝失敗的問題。 • 修正部署期間關閉元件會使用新版本的關機指令碼的問題。 • 各種關機改進。 • 其他小修正和改進。
MQTT 5 經紀商	該MQTT 5 經紀商組件的 1.1.0 版本可用。

元件	詳細資訊
	新功能 • 添加對 EMQX 配置的支持，包括代理程序選項和插件。 錯誤修復和改進 • 將 EMQX 更新至 4.4.9 版本。

發行版本：2022 年 7 月 28 日AWS IoT Greengrass核心軟體更新

此發行版本提供 Greengrass 核心元件的 2.7.0 版、串流管理員元件的 2.1.0 版，以及 Lambda 管理員元件的 2.2.5 版。

發行日期：2022 年 7 月 28 日

發行亮點

- 串流管理員遙測指標 — Stream Manager 現在會自動將遙測指標傳送至 Amazon EventBridge，以便您建立雲端應用程式，以監控和分析核心裝置上傳的資料量。如需詳細資訊，請參閱 從AWS IoT Greengrass核心裝置收集系統健康狀態遙測資料。
- 自訂憑證授權單位 (CA) — 現在支援由自訂憑證 CA (未註冊 CA) 簽署的用戶端憑證。AWS IoT如需詳細資訊，請參閱 使用私有 CA 簽署的裝置憑證。

版本詳情

- 公用元件更新

公用元件更新

下表列出AWS提供的元件，其中包括新功能和更新的功能。

> **Important**
>
> 當您部署元件時，AWS IoT Greengrass會安裝該元件所有相依性的最新受支援版本。因此，如果您將新裝置新增至物件群組，或更新以這些裝置為目標的部署，則AWS提供之公用元件的新修補程式版本可能會自動部署到核心裝置。某些自動更新 (例如核心更新) 可能會導致裝置意外重新啟動。

若要避免對裝置上執行的元件進行意外更新，建議您在建立部署時直接包含該元件的偏好版本。如需有關 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱更新AWS IoT Greengrass核心軟件（OTA）。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.7.0 版本是可用的。 新功能 • 更新 Greengrass 核心核心，以便在核心裝置套用本機部AWS IoT Greengrass署時，將狀態更新傳送至雲端。 • 添加對由未註冊 CA 的自定義證書頒發機構（CA）簽名的客戶端證書的支持AWS IoT。若要使用此功能，您可以將新的組`greengrassDataPlaneEndpoint` 態選項設定為`iotdata`。如需詳細資訊，請參閱 使用私有 CA 簽署的裝置憑證。 錯誤修復和改進 • 修正當核停止或重新啟動時，Greengrass 核會在特定情況下復原部署的問題。核心現在會在核心重新啟動後繼續部署。 • 當您指定將軟體設定為系統服務時，請更新 Greengrass 安裝程式以遵守`--start`引數。 • 更新的行為，SubscribeToComponentUpdates以在核心更新元件的事件中設定部署 ID。 • 其他小的修正和改進。
流管理器	流管理器組件的版本 2.1.0 可用。 新功能 • 更新此元件以自動將遙測指標傳送至 Amazon EventBridge。如需詳細資訊，請參閱 從AWS IoT Greengrass核心裝置收集系統健康狀態遙測資料。 此功能需要 v2.7.0 或更高版 Greeng rass 核成分。 • 版本更新了 Greengrass 核 2.7.0 版本釋放。
Lambda 經理	您可以使用 Lambda 管理員元件 2.2.5 版本。

元件	詳細資訊
	新功能 • 在您訂閱本機發佈/訂閱訊息的事件來源中新增 MQTT 主題萬用字元的支援。 此功能需要 v2.6.0 或更高版 Greeng rass 核成分。 • 版本更新了 Greengrass 核 2.7.0 版本釋放。

發行版本：2022 年 6 月 27 日AWS IoT Greengrass核心軟體更新

此版本提供 Greengrass 核心元件的 2.6.0 版、新提供的元件，以及AWS提供元件的更新。AWS

發行日期：二 ○○ 二年六月二十七日

發行亮點

- 本機發佈/訂閱主題中的通配符號 — 您現在可以在訂閱本機發佈/訂閱主題時使用 MQTT 萬用字元。如需詳細資訊，請參閱 發佈/訂閱本地訊息 及 SubscribeToTopic。
- 用戶端裝置陰影支援 — 您現在可以與自訂元件中的用戶端裝置陰影互動，並與用戶端裝置陰影同步 AWS IoT Core。如需詳細資訊，請參閱 與用戶端裝置陰影互動並同步。
- 用戶端裝置的本機 MQTT 5 支援 — 您現在可以部署 EMQX MQTT 5 代理程式，在用戶端裝置與核心裝置之間的通訊中使用 MQTT 5 功能。如需詳細資訊，請參閱 MQTT 5 經紀商 及 將用戶端裝置連線至核心裝置。
- 零組件組態中的配方變數 — 您現在可以在元件組態中使用特定的 recipe 變數。當您在方案中定義元件的預設組態或在部署中設定元件時，您可以使用這些方案變數。如需詳細資訊，請參閱 配方變數 及 在合併更新中使用配方變數。
- IPC 授權原則中的萬用字元 — 您現在可以使用*萬用字元來比對處理序間通訊 (IPC) 授權原則中的任何字元組合。此萬用字元可讓您在單一授權原則中允許存取多個資源。如需詳細資訊，請參閱 授權原則中的萬用字元。
- 管理本機部署和元件的 IPC 作業 — 您現在可以開發用於管理本機部署和檢視元件詳細資料的自訂元件。如需詳細資訊，請參閱 IPC：管理本機部署和元件。
- 驗證和授權用戶端裝置的 IPC 作業 — 您現在可以使用這些作業來建立自訂的本機代理程式元件。如需詳細資訊，請參閱 IPC：驗證和授權用戶端裝置。

版本詳情

公用元件更新

下表列出AWS提供的元件，其中包括新功能和更新的功能。

> **Important**
>
> 當您部署元件時，AWS IoT Greengrass會安裝該元件所有相依性的最新受支援版本。因此，如果您將新裝置新增至物件群組，或更新以這些裝置為目標的部署，則AWS提供之公用元件的新修補程式版本可能會自動部署到核心裝置。某些自動更新 (例如核心更新) 可能會導致裝置意外重新啟動。
> 若要避免對裝置上執行的元件進行意外更新，建議您在建立部署時直接包含該元件的偏好版本。如需有關 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱更新AWS IoT Greengrass核心軟件（OTA）。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.6.0 版本是可用的。 新功能 • 當您訂閱本機發佈/訂閱主題時，新增對 MQTT 萬用字元的支援。如需詳細資訊，請參閱 發佈/訂閱本地訊息 及 SubscribeToTopic。 • 添加對組件配置中的配方變量的支持，而不是 *component_dependency_name* :configuration: *json_pointer* recipe 變量。當您在方案中定義元件或在部署DefaultConfiguration 中設定元件時，您可以使用這些方法變數。若要啟用此功能，請將interpolateComponentConfiguration組態選項設定為true。如需詳細資訊，請參閱 配方變數 及 在合併更新中使用配方變數。 • 在進程間通信（IPC）授權策略中添加對通*配符的完全支持。您現在可以指定*資源字串中的字元，以符合任何字元組合。如需詳細資訊，請參閱 授權原則中的萬用字元。 • 添加對自定義組件的支持，以調用 Greengrass CLI 使用的 IPC 操作。您可以使用這些 IPC 作業來管理本機部署、檢視元件詳細資料，以及產生

元件	詳細資訊
	可用來登入本機除錯主控台的密碼。如需詳細資訊，請參閱 IPC：管理本機部署和元件。 錯誤修復和改進 • 修正了相依元件在某些情況下重新啟動其硬相依性或變更狀態時不會反應的問題。 • 改善部署失敗時核心裝置向AWS IoT Greengrass雲端服務報告的錯誤訊息。 • 修正當核心重新啟動時，Greengrass 核在特定案例中套用物件部署兩次的問題。 • 其他小修正和改進。如需詳細資訊，請參閱中的發行版本 GitHub。
MQTT 5 經紀商	新 EMQX MQTT 5 代理程式元件的 1.0.0 版可供使用。 新功能 • 添加對本地 EMQX MQTT 5 代理程式的支援。用戶端裝置可以連線到此 MQTT 代理程式，以便使用 MQTT 5 功能與核心裝置進行通訊。
陰影管理員	陰影管理員元件的 2.2.0 版可用。 新功能 • 通過本地發布/訂閱界面添加對本地陰影服務的支持。您現在可以在陰影 MQTT 主題上與本機發佈/訂閱訊息代理程式進行通訊，以取得、更新和刪除核心裝置上的陰影。此功能可讓您使用 MQTT 橋接器在用戶端裝置與本機發佈/訂閱介面之間轉送陰影主題的訊息，將用戶端裝置連線至本機陰影服務。 此功能需要 v2.6.0 或更高版 Greeng rass 核成分。若要將用戶端裝置連線到本機陰影服務，您也必須使用 v2.2.0 或更新版本的 MQ TT 橋接器元件。 • 新增您可以設定的`direction` 選項，以自訂方向，以便在本機陰影服務與之間同步陰影AWS 雲端。您可以設定此選項以減少頻寬和與AWS 雲端.

元件	詳細資訊
用戶端裝置驗證	客戶端設備身份驗證組件的 2.2.0 版可用。 新功能 • 添加對自定義組件的支持，以調用進程間通信（IPC）操作以驗證和授權客戶端設備。例如，您可以在自訂 MQTT 代理程式元件中使用這些作業。如需詳細資訊，請參閱 IPC：驗證和授權用戶端裝置。 • 新增`maxActiveAuthTokens` 、和`threadPoolSize` 選項`cloudQueueSize` ，您可以設定這些選項來調整此元件的執行方式。
MQTT 大橋	MQTT 橋接器元件的 2.2.0 版本可供使用。 新功能 • 當您指定本機發佈/訂閱做為來源訊息代理程式時，新增對 MQTT 主題萬用字元 (#和+) 的支援。 此功能需要 v2.6.0 或更高版 Greeng rass 核成分。 • 新增選`targetTopicPrefix` 項，您可以指定該選項來設定 MQTT 橋接器，以便在轉送郵件時將首碼新增至目標主題。
Greengrass CLI	Greengrass CLI 的 2.6.0 版本是可用的。 新功能 • 添加對自定義組件的支持，以調用 Greengrass CLI 使用的進程間通信（IPC）操作。您可以使用這些 IPC 作業來管理本機部署、檢視元件詳細資料，以及產生可用來登入本機除錯主控台的密碼。如需詳細資訊，請參閱 IPC：管理本機部署和元件。 錯誤修復和改進 • 其他小修正和改進。

發行版本：2022 年 5 月 31 日AWS IoT Greengrass核心軟體更新

此版本提供 Greengrass 核心元件的 2.5.6 版，以及記錄檔管理員元件的 2.2.4 版。

發行日期：二○二二年五月三十一

版本詳情

- 公用元件更新

公用元件更新

下表列出AWS提供的元件，其中包括新功能和更新的功能。

> **Important**
>
> 當您部署元件時，AWS IoT Greengrass會安裝該元件所有相依性的最新受支援版本。因此，如果您將新裝置新增至物件群組，或更新以這些裝置為目標的部署，則AWS提供之公用元件的新修補程式版本可能會自動部署到核心裝置。某些自動更新 (例如核心更新) 可能會導致裝置意外重新啟動。
> 若要避免對裝置上執行的元件進行意外更新，建議您在建立部署時直接包含該元件的偏好版本。如需有關 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱更新AWS IoT Greengrass核心軟件（OTA）。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.5.6 版本是可用的。 新功能 • 添加對使用 ECC 密鑰的硬件安全模塊的支持。您可以使用硬體安全模組 (HSM) 來安全地儲存裝置的私密金鑰和憑證。如需詳細資訊，請參閱 硬體安全整合。 錯誤修復和改進 • 修正當您在特定案例中部署含有損毀安裝指令碼的元件時，部署永遠不會完成的問題。 • 改善啟動期間的效能。 • 其他小修正和改進。
日誌管理器	日誌管理器組件的 2.2.4 版可用。 錯誤修復和改進 • 改善處理無效組態時的穩定性。

元件	詳細資訊
	• 其他小修正和改進。

發行版本：2022 年 4 月 6 日AWS IoT Greengrass核心軟體更新

此版本提供了 Greengrass 核成分的 2.5.5 版。

發行日期：二 ○ 二二年四月六日

版本詳情

- 公用元件更新

公用元件更新

下表列出AWS提供的元件，其中包括新功能和更新的功能。

> **Important**
>
> 當您部署元件時，AWS IoT Greengrass會安裝該元件所有相依性的最新受支援版本。因此，如果您將新裝置新增至物件群組，或更新以這些裝置為目標的部署，則AWS提供之公用元件的新修補程式版本可能會自動部署到核心裝置。某些自動更新 (例如核心更新) 可能會導致裝置意外重新啟動。
> 若要避免對裝置上執行的元件進行意外更新，建議您在建立部署時直接包含該元件的偏好版本。如需有關 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱更新AWS IoT Greengrass核心軟件（OTA）。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.5.5 版本是可用的。 新功能 • 新增元件的`GG_ROOT_CA_PATH` 環境變數，以便您可以存取自訂元件中的根憑證授權單位 (CA) 憑證。

元件	詳細資訊
	錯誤修復和改進 • 添加對使用英語以外顯示語言的 Windows 設備的支持。 • 更新 Greengrass 核心剖析布林安裝程式引數的方式，以便您可以指定不含布林值的布林引數來指定值。`true`例如，您現在可以指定`--provision` 而不是使用自動資源佈建`--provision true` 進行安裝。 • 修正核心裝置在特定情況下佈建後未向AWS IoT Greengrass雲端服務回報狀態的問題。 • 其他小修正和改進。

發行版本：AWS IoT Greengrass核心 v2.5.4 軟體更新於 2022 年 3 月 23 日

此版本提供了綠核元件的 2.5.4 版，以及 Lambda 啟動器元件的 2.0.10 版。

發行日期：二 ○○ 二年三月二十三日

版本詳情

- 公用元件更新

公用元件更新

下表列出AWS提供的元件，其中包括新功能和更新的功能。

> **Important**
>
> 當您部署元件時，AWS IoT Greengrass會安裝該元件所有相依性的最新受支援版本。因此，如果您將新裝置新增至物件群組，或更新以這些裝置為目標的部署，則AWS提供之公用元件的新修補程式版本可能會自動部署到核心裝置。某些自動更新 (例如核心更新) 可能會導致裝置意外重新啟動。
> 若要避免對裝置上執行的元件進行意外更新，建議您在建立部署時直接包含該元件的偏好版本。如需有關 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱更新AWS IoT Greengrass核心軟件（OTA）。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.5.4 版本是可用的。 錯誤修復和改進 • 一般錯誤修正與改進。
发 Lambda 器	已提供 Lambda 啟動器元件的 2.0.10 版本。 錯誤修復和改進 • 一般錯誤修正與改進。

發行版本：AWS IoT Greengrass核心 v2.5.3 軟體更新於 2022 年 1 月 6 日

此發行版本提供了 Greengrass 核心元件的 2.5.3 版，以及新的 PKCS #11 提供者元件。

發行日期：二 ○ 二二年一月六日

發行亮點

- 硬體安全性整合 — 您現在可以將 AWS IoT Greengrass Core 軟體設定為使用安全性儲存在硬體安全模組 (HSM) 中的私密金鑰和憑證。如需詳細資訊，請參閱 硬體安全整合。

版本詳情

- 公用元件更新

公用元件更新

下表列出AWS提供的元件，其中包括新功能和更新的功能。

> **Important**
>
> 當您部署元件時，AWS IoT Greengrass會安裝該元件所有相依性的最新受支援版本。因此，如果您將新裝置新增至物件群組，或更新以這些裝置為目標的部署，則AWS提供之公用元件的新修補程式版本可能會自動部署到核心裝置。某些自動更新 (例如核心更新) 可能會導致裝置意外重新啟動。

若要避免對裝置上執行的元件進行意外更新，建議您在建立部署時直接包含該元件的偏好版本。如需有關 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱更新AWS IoT Greengrass核心軟件（OTA）。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.5.3 版本是可用的。 新功能 • 添加對硬件安全集成的支持。您可以使用硬體安全模組 (HSM) 來安全地儲存裝置的私密金鑰和憑證。如需詳細資訊，請參閱 硬體安全整合。 錯誤修復和改進 • 修正核心建立 MQTT 連線時執行階段例外狀況的問題。AWS IoT Core
PKCS #11 供應商	PKCS #11 提供者元件的 2.0.0 版可供使用。 新功能 • 添加對硬件安全集成的支持。您可以使用硬體安全模組 (HSM) 來安全地儲存裝置的私密金鑰和憑證。如需更多詳細資訊，請參閱 硬體安全整合。

發行版本：AWS IoT Greengrass2021 年 12 月 3 日的核心 v2.5.2 軟體更新

此版本提供了 Greengrass 核成分的 2.5.2 版本。

發行日期：二零二一年十二月三日

版本詳情

- 公用元件更新

公用元件更新

下表列出AWS提供的元件，其中包括新功能和更新的功能。

> **Important**
>
> 當您部署元件時，AWS IoT Greengrass會安裝該元件所有相依性的最新受支援版本。因此，如果您將新裝置新增至物件群組，或更新以這些裝置為目標的部署，則AWS提供之公用元件的新修補程式版本可能會自動部署到核心裝置。某些自動更新 (例如核心更新) 可能會導致裝置意外重新啟動。
>
> 若要避免對裝置上執行的元件進行意外更新，建議您在建立部署時直接包含該元件的偏好版本。如需有關 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱更新AWS IoT Greengrass核心軟件（OTA）。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.5.2 版本是可用的。 錯誤修復和改進 • 修復了 Greengrass 核心更新後，Windows 服務在您停止或重新啟動設備後無法再次啟動的問題。
AWS IoT Device Defender	該AWS IoT Device Defender組件的版本 3.0.1 可用。 此版本的AWS IoT Device Defender元件需要與版本 2.x 不同的組態參數。如果您對 2.x 版使用非預設組態，並且想要從 v2.x 升級至 v3.x，則必須更新元件的組態。如需詳細資訊，請參閱AWS IoT Device Defender元件組態。 新功能 • 添加對運行 Windows 的核心設備的支持。 • 將元件類型從 Lambda 元件變更為泛型元件。此元件現在不再依賴舊版訂閱路由器元件來建立訂閱。 • 新增可讓您選擇性地停用安裝元件相依性的安裝指令碼的新`UseInstaller` 組態參數。

發行版本：AWS IoT Greengrass2021 年 11 月 23 日核心 v2.5.1 軟體更新

此版本提供了 Greengrass 核成分的 2.5.1 版本。

版本日期：2021 年 11 月 23 日

版本詳情

- 公用元件更新

公用元件更新

下表列出AWS提供的元件，其中包括新功能和更新的功能。

> **Important**
>
> 當您部署元件時，AWS IoT Greengrass會安裝該元件所有相依性的最新受支援版本。因此，如果您將新裝置新增至物件群組，或更新以這些裝置為目標的部署，則AWS提供之公用元件的新修補程式版本可能會自動部署到核心裝置。某些自動更新 (例如核心更新) 可能會導致裝置意外重新啟動。
> 若要避免對裝置上執行的元件進行意外更新，建議您在建立部署時直接包含該元件的偏好版本。如需有關 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱更新AWS IoT Greengrass核心軟件（OTA）。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.5.1 版本是可用的。 錯誤修復和改進 • 添加對 Windows 上 Java 運行時環境（JRE）的 32 位版本的支持。 • 變更AWS IoT原則未授與`greengrass:ListThingGroupsForCoreDevice` 權限之核心裝置的物件群組移除行為。使用此版本時，部署會繼續進行、記錄警告，而且當您從物件群組移除核心裝置時，不會移除元件。如需詳細資訊，請參閱 將AWS IoT Greengrass元件部署到裝置。 • 修正 Greengrass 核可供 Greengrass 元件處理程序使用的系統環境變數問題。您現在可以重新啟動元件，使其使用最新的系統環境變數。

發行版本：AWS IoT Greengrass2021 年 11 月 12 日核心 v2.5.0 軟體更新

此版本提供 Greengrass 核心元件 2.5.0 版、新提供的元件，以及AWS提供元件的更新。AWS

發行日期：二零二一年十一月十二

發行亮點

- Windows 裝置支援 — 您現在可以在執行 Windows 作業系統的裝置上執行AWS IoT Greengrass核心軟體。如需詳細資訊，請參閱 支援平台和需求 及 通過操作系統的 Greengrass 功能兼容性。
- 新增物件群組移除行為 — 您現在可以從物件群組中移除核心裝置，以在下次部署至該裝置時移除該物件群組的元件。

> **Important**
>
> 由於此更改，核心設備的AWS IoT策略必須具有`greengrass:ListThingGroupsForCoreDevice`權限。如果您使用 AWS IoT GreengrassCore 軟體安裝程式來佈建資源，則預設AWS IoT原則允許`greengrass:*`，其中包括此權限。如需詳細資訊，請參閱 AWS IoT Greengrass 的裝置身分驗證和授權。

- 硬體安全性支援 — 您現在可以將 AWS IoT Greengrass Core 軟體設定為使用硬體安全模組 (HSM)，以便安全地儲存裝置的私密金鑰和憑證。如需詳細資訊，請參閱 硬體安全整合。
- HTTPS 代理伺服器支援 — 您現在可以將AWS IoT Greengrass核心軟體設定為透過 HTTPS 代理伺服器進行連線。如需詳細資訊，請參閱 連線至連接埠 443 或透過網路代理。

版本詳情

- 平台支援更新
- 公用元件更新

平台支援更新

平台	詳細資訊
Windows	AWS IoT Greengrass現在支援在以下版本的 Windows 上執行AWS IoT Greengrass核心軟體：

平台	詳細資訊
	• Windows 10 • Windows Server 2019 如需詳細資訊，請參閱 支援平台和需求 及 通過操作系統的 Greengrass 功能兼容性。

公用元件更新

下表列出AWS提供的元件，其中包括新功能和更新的功能。

> **Important**
>
> 當您部署元件時，AWS IoT Greengrass會安裝該元件所有相依性的最新受支援版本。因此，如果您將新裝置新增至物件群組，或更新以這些裝置為目標的部署，則AWS提供之公用元件的新修補程式版本可能會自動部署到核心裝置。某些自動更新 (例如核心更新) 可能會導致裝置意外重新啟動。
> 若要避免對裝置上執行的元件進行意外更新，建議您在建立部署時直接包含該元件的偏好版本。如需有關 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱更新AWS IoT Greengrass核心軟件（OTA）。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.5.0 版本是可用的。 新功能 • 添加對運行 Windows 的核心設備的支持。 • 變更物件群組移除的行為。使用此版本，您可以從物件群組中移除核心裝置，以便在下次部署中解除安裝該物群組的元件。 由於此更改，核心設備的AWS IoT策略必須具有`greengrass:ListThingGroupsForCoreDevice` 權限。如果您使用 AWS IoT GreengrassCore 軟體安裝程式來佈建資源，則預設AWS IoT原則允

元件	詳細資訊
	許`greengrass:*` ，其中包括此權限。如需詳細資訊，請參閱 AWS IoT Greengrass 的裝置身分驗證和授權。 • 添加對 HTTPS 代理配置的支持。如需詳細資訊，請參閱 連線至連接埠 443 或透過網路代理。 • 加入新的`windowsUser` 組態參數。您可以使用此參數來指定用來在 Windows 核心裝置上執行元件的預設使用者。如需詳細資訊，請參閱 設定執行元件的使用者。 • 新增可用來自訂 HTTP 要求逾時的新`httpClient` 組態選項，以改善慢速網路上的效能。如需詳細資訊，請參閱 httpClient 設定參數。 錯誤修復和改進 • 修復了引導生命週期選項，以從組件重新啟動核心設備。 • 添加對配方變量中連字符的支持。 • 修正隨選 Lambda 函數元件的 IPC 授權。 • 改進了日誌消息並將非關鍵日誌從級別更改`INFO`為`DEBUG`級別，因此日誌更有用。 • 當您使`iot:DescribeCertificate` 用自動佈建安裝 AWS IoT Greengrass Core 軟體時，會從 Greengrass 核心建立的預設權杖交換角色中移除權限。Greengrass 核不使用此權限。 • 修正問題，如`iam:CreatePolicy` 果相同原則可使用，則自動佈建指令碼不需要`iam:GetPolicy` 權限。 • 其他小修正和改進。
Greengrass CLI	Greengrass CLI 的 2.5.0 版本是可用的。 新功能 • 添加對運行 Windows 的核心設備的支持。 • 新增可指定的新組`AuthorizedWindowsGroups` 態參數，以授權系統群組在 Windows 裝置上使用 Greengrass CLI。 • 為本機部署新增`windowsUser` 參數。您可以使用此參數指定用來在 Windows 核心裝置上執行元件的使用者。

元件	詳細資訊
CloudWatch 度量	CloudWatch指標元件的 3.0.0 版可用。 此版本的 CloudWatch 指標元件預期的組態參數與版本 2.x 不同。如果您對 2.x 版使用非預設組態，並且想要從 v2.x 升級至 v3.x，則必須更新元件的組態。如需詳細資訊，請參閱CloudWatch測量結果元件組態。 新功能 • 添加對運行 Windows 的核心設備的支持。 • 將元件類型從 Lambda 元件變更為泛型元件。此元件現在不再依賴舊版訂閱路由器元件來建立訂閱。 • 新增`InputTopic` 組態參數，以指定元件訂閱接收訊息的主題。 • 新增`OutputTopic` 組態參數，以指定元件發佈狀態回應的主題。 • 新增`PubSubToIoTCore` 組態參數，以指定是否要發佈和訂閱 AWS IoT Core MQTT 主題。 • 新增可讓您選擇性地停用安裝元件相依性的安裝指令碼的新`UseInstaller` 組態參數。 錯誤修復和改進 添加對輸入數據中重複時間戳的支持。
Lambda 經理	您可以使用 Lambda 管理員元件 2.2.0 版。 錯誤修復和改進 • 修正 Lambda 函數在重新啟動後無法寫入記錄的問題。 • 修正當主題中有萬用字元時，舊版訂閱路由器會傳送重複訊息的問題。 • 修正非釘選 Lambda 函數無法在. AWS IoT Device SDK

發行版本：AWS IoT Greengrass2021 年 8 月 3 日核心 v2.4.0 軟體更新

此版本提供 Greengrass 核心元件 2.4.0 版、新提供的元件，以及AWS提供元件的更新。AWS

發行日期：二零二一年八月三日

發行亮點

- 系統資源限制 — Greengrass 核元件現在支援系統資源限制。您可以設定每個元件的處理序可在核心裝置上使用的 CPU 和 RAM 使用量上限。如需詳細資訊，請參閱 設定元件的系統資源限制。
- 暫停/繼續元件 — Greengrass 核現在支援暫停和恢復元件。您可以使用處理序間通訊 (IPC) 程式庫來開發可暫停和繼續其他元件程序的自訂元件。如需詳細資訊，請參閱 PauseComponent 及 ResumeComponent。
- 透過AWS IoT叢集佈建進行安裝 — 使用新的AWS IoT叢集佈建外掛程式，在連線以佈建所需AWS資源的裝置上安裝 AWS IoT Greengrass Core 軟體。AWS IoT裝置會使用宣告憑證進行佈建。您可以在製造過程中將聲明憑證嵌入裝置上，這樣每個裝置都可以在線上時立即佈建。如需詳細資訊，請參閱 透過 AWS IoT 叢集佈建安裝 AWS IoT Greengrass 核心軟體。
- 使用自訂佈建進行安裝 — 開發自訂佈建外掛程式，以便在裝置上安裝 AWS IoT Greengrass Core 軟體時佈建所需的AWS資源。您可以建立在安裝期間執行的 Java 應用程式，為您的自訂使用案例設定 Greengrass 核心裝置。如需詳細資訊，請參閱 使用自訂資源佈建安裝 AWS IoT Greengrass 核心軟體。

版本詳情

- 公用元件更新

公用元件更新

下表列出AWS提供的元件，其中包括新功能和更新的功能。

> **Important**
>
> 當您部署元件時，AWS IoT Greengrass會安裝該元件所有相依性的最新受支援版本。因此，如果您將新裝置新增至物件群組，或更新以這些裝置為目標的部署，則AWS提供之公用元件的新修補程式版本可能會自動部署到核心裝置。某些自動更新 (例如核心更新) 可能會導致裝置意外重新啟動。
>
> 若要避免對裝置上執行的元件進行意外更新，建議您在建立部署時直接包含該元件的偏好版本。如需有關 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱更新AWS IoT Greengrass核心軟件（OTA）。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.4.0 版本是可用的。 新功能 • 添加對系統資源限制的支援。您可以設定每個元件的處理序可在核心裝置上使用的 CPU 和 RAM 使用量上限。如需詳細資訊，請參閱 設定元件的系統資源限制。 • 添加 IPC 操作以暫停和恢復組件。如需詳細資訊，請參閱 PauseComponent 及 ResumeComponent。 • 添加對配置插件的支持。您可以指定要在安裝期間執行的 JAR 檔案，以佈建 Greengrass 核心裝置所需的AWS資源。Greengrass 核包括一個接口，您可以實現該接口以開發自定義配置插件。如需詳細資訊，請參閱使用自訂資源佈建安裝 AWS IoT Greengrass 核心軟體。 • 將選用`thing-name-policy` 引數新增至 AWS IoT Greengrass Core 軟體安裝程式。當您使用自動資源佈建來安裝 AWS IoT Greengrass Core 軟體時，您可以使用此選項來指定現有或自訂AWS IoT原則。 錯誤修復和改進 • 在啟動時更新記錄配置。這修正了啟動時未套用記錄設定的問題。 • 更新核子加載器符號鏈接，以在安裝期間指向 Greengrass 根文件夾中的組件存儲區。此更新可讓您刪除安裝 AWS IoT Greengrass Core 軟體時下載的 JAR 檔案和其他核心成品。 • 其他小修正和改進。如需詳細資訊，請參閱中的發行版本 GitHub。
Greengrass CLI	Greengrass CLI 的 2.4.0 版本是可用的。 新功能 • 添加對系統資源限制的支援。當您建立本機部署時，您可以設定每個元件的處理序可在核心裝置上使用的 CPU 和 RAM 使用量上限。若要取得更多資訊，請參閱設定元件的系統資源限制和部署建立指令。
AWS IoT依宣告佈建叢集	通過聲明插件的AWS IoT車隊佈建現在可用。如需詳細資訊，請參閱 透過 AWS IoT 叢集佈建安裝 AWS IoT Greengrass 核心軟體。

元件	詳細資訊
	新功能 • 添加支援以安裝具有AWS IoT車隊佈建的 AWS IoT Greengrass Core 軟體。在安裝期間，裝置會連線AWS IoT到以佈建必要的AWS資源並下載裝置憑證以用於一般作業。

發行版本：AWS IoT Greengrass2021 年 6 月 29 日核心 v2.3.0 軟體更新

此版本提供了 Greengrass 核成分的 2.3.0 版本。

發行日期：6 月 29 日

發行亮點

- 大型配置支持-Greengrass 核組件現在支持高達 10 MB 的部署文檔。您現在可以將較大的組態更新部署到 Greengrass 元件。

> **Note**
>
> 若要使用此功能，核心裝置的AWS IoT原則必須允許`greengrass:GetDeploymentConfiguration`權限。如果您使用 AWS IoT GreengrassCore 軟體安裝程式來佈建資源，則核心裝置的AWS IoT政策允許`greengrass:*`，其中包括此權限。如需詳細資訊，請參閱 AWS IoT Greengrass 的裝置身分驗證和授權。

版本詳情

- 公用元件更新

公用元件更新

下表列出AWS提供的元件，其中包括新功能和更新的功能。

> **Important**
>
> 當您部署元件時，AWS IoT Greengrass會安裝該元件所有相依性的最新受支援版本。因此，如果您將新裝置新增至物件群組，或更新以這些裝置為目標的部署，則AWS提供之公用元件的新修補程式版本可能會自動部署到核心裝置。某些自動更新 (例如核心更新) 可能會導致裝置意外重新啟動。
>
> 若要避免對裝置上執行的元件進行意外更新，建議您在建立部署時直接包含該元件的偏好版本。如需有關 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱更新AWS IoT Greengrass核心軟件（OTA）。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.3.0 版本是可用的。 新功能 • 新增對部署組態文件的支援，最高可達 10 MB，最大為 7 KB (針對目標物件的部署) 或 31 KB (針對目標物件群組的部署)。 若要使用此功能，核心裝置的AWS IoT原則必須允許`greengrass:GetDeploymentConfiguration` 權限。如果您使用 AWS IoT GreengrassCore 軟體安裝程式來佈建資源，則核心裝置的AWS IoT政策允許`greengrass:*` ，其中包括此權限。如需詳細資訊，請參閱 AWS IoT Greengrass 的裝置身分驗證和授權。 • 添加`iot:thingName` 配方變量。您可以使用此 recipe 變數來取得方案中核心裝置AWS IoT物件的名稱。如需詳細資訊，請參閱 配方變數。 錯誤修復和改進 • 其他小修正和改進。如需詳細資訊，請參閱中的發行版本 GitHub。

發行版本：AWS IoT Greengrass2021 年 6 月 18 日核心 v2.2.0 軟體更新

此版本提供 Greengrass 核心元件 2.2.0 版、新提供的元件，以及AWS提供元件的更新。AWS

發行日期：二零二一年六月十八日

發行亮點

- 用戶端裝置支援 AWS — 提供的全新用戶端裝置元件可讓您使用雲端探索將用戶端裝置連線到核心裝置。您可以同步用戶端裝置，AWS IoT Core並與 Greengrass 元件中的用戶端裝置互動。如需詳細資訊，請參閱 與本機 IoT 裝置互動。
- 本機陰影服務 — 新的陰影管理員元件會在核心裝置上啟用本機陰影服務。離線時，您可以使用此陰影服務與本機陰影互動，使用中的 Greengrass 間通訊 (IPC) 程式庫。AWS IoT Device SDK您也可以使用陰影管理員元件，將本機陰影狀態與同步化AWS IoT Core。如需詳細資訊，請參閱 與裝置陰影互動。

版本詳情

- 公用元件更新

公用元件更新

下表列出AWS提供的元件，其中包括新功能和更新的功能。

> **Important**
>
> 當您部署元件時，AWS IoT Greengrass會安裝該元件所有相依性的最新受支援版本。因此，如果您將新裝置新增至物件群組，或更新以這些裝置為目標的部署，則AWS提供之公用元件的新修補程式版本可能會自動部署到核心裝置。某些自動更新 (例如核心更新) 可能會導致裝置意外重新啟動。
> 若要避免對裝置上執行的元件進行意外更新，建議您在建立部署時直接包含該元件的偏好版本。如需有關 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱更新AWS IoT Greengrass核心軟件（OTA）。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.2.0 版本是可用的。 新功能 • 添加用於本地陰影管理的 IPC 操作。 錯誤修復和改進 • 減少 JAR 檔案的大小。

元件	詳細資訊
	• 減少記憶體使用量。 • 修復了某些情況下未更新日誌配置的問題。 • 其他小的修正和改進。如需詳細資訊，請參閱（詳見）的版本 GitHub。
陰影管理	新的陰影管理器組件的 2.0.0 版可用。 新功能 • 添加對經典陰影和命名陰影的支持。 • 添加對使用 IPC 本地陰影管理的支持。 • 添加對陰影同步的支援AWS IoT Core。
用戶端裝置驗證	新客戶端設備身份驗證組件的 2.0.0 版可用。 新功能 • 添加對 Greengrass 用戶端裝置的支援，這些裝置是透過 MQTT 連線到核心裝置的本機 IoT 裝置。 • 添加對客戶端設備及其 MQTT 操作的身份驗證和授權的支持。
MQTT 經紀商模式	新的 Moquette MQTT 代理組件的 2.0.0 版本可用。 新功能 • 新增對處理與用戶端裝置通訊的本機 Moquette MQTT 代理程式的支援。
MQTT 大橋	新的 MQTT 橋接器元件的 2.0.0 版可供使用。 新功能 • 添加對本地 MQTT 代理，本地 Greengrass 發布/訂閱代理和 MQTT 代理之間轉送消息的支持。AWS IoT Core
IP 偵測器	新 IP 檢測器組件的 2.0.0 版可用。 新功能 • 添加支持，以將核心設備的本地 MQTT 代理端點報告到AWS IoT Greengrass雲服務以供客戶端設備連接。

元件	詳細資訊
日誌管理器	日誌管理器組件的 2.1.1 版可用。 錯誤修復和改進 • 修正了某些情況下系統日誌配置未更新的問題。
DLR 物體偵測	您可以使用 DLR 物件偵測的 2.1.2 版本。 錯誤修復和改進 • 修正在範例 DLR 物件偵測推論結果中造成不正確邊界方塊的影像縮放問題。
TensorFlow 精簡型物體偵測	TensorFlow 精簡版對象檢測的 2.1.1 版可用。 錯誤修復和改進 • 修正影像縮放問題，此問題會導致範例 TensorFlow Lite 物件偵測推論結果中不正確的邊界方塊。

發行版本：AWS IoT Greengrass2021 年 4 月 26 日核心 v2.1.0 軟體更新

此版本提供了 Greengrass 核組件的 2.1.0 版和更新提供的組件。AWS

發行日期：四月二十六日

發行亮點

- 碼頭集線器和 Amazon Elastic Container Registry (Amazon ECR) 集成-新的 Docker 應用程序管理器組件使您可以從 Amazon ECR 下載公共或私有映像。您也可以使用此元件從 Docker 集線器和 AWS Marketplace. 如需詳細資訊，請參閱 運行碼頭容器。
- 適用於AWS IoT Greengrass核心軟體的碼頭檔案和碼頭影像 — 您可以使用 Greengrass 泊塢視窗映像在使用 Amazon Linux 2 作為基礎作業系統的泊塢視窗容器AWS IoT Greengrass中執行。您還可以使用AWS IoT Greengrass碼頭文件來構建自己的 Greengrass 圖像。如需詳細資訊，請參閱 在 Docker 容器中執行 AWS IoT Greengrass 核心軟體。

- Sup@@ port 其他機器學習架構和平台 — 您可以部署範例機器學習推論元件，這些元件使用預先訓練的模型，使用 TensorFlow Lite 2.5.0 和 DLR 1.6.0 執行範例影像分類和物件偵測。此版本還擴展了對 Armv8 (AArch64) 裝置的範例機器學習支援。如需詳細資訊，請參閱 執行機器學習推論。

版本詳情

- 平台支援更新
- 公用元件更新

平台支援更新

平台	詳細資訊
Docker	的碼頭檔案和碼頭影像現在可供使AWS IoT Greengrass用。 Dockerfile AWS IoT Greengrass提供一個碼頭檔案來建置一個容器映像檔，該映像檔具有安裝在 Amazon Linux 2 (x86_64) 基礎映像上的AWS IoT Greengrass核心軟體和相依性。您可以修改 Docker 文件中的基本映像以在不同的平台架構AWS IoT Greengrass上運行。 Docker 映像檔 AWS IoT Greengrass提供預先建立的 Docker 映像檔，該映像檔在 Amazon Linux 2 (x86_64) 基礎映像上安裝了AWS IoT Greengrass核心軟體和相依性。 如需詳細資訊，請參閱 在 Docker 容器中執行 AWS IoT Greengrass 核心軟體。

公用元件更新

下表列出AWS提供的元件，其中包括新功能和更新的功能。

> **Important**
>
> 當您部署元件時，AWS IoT Greengrass會安裝該元件所有相依性的最新受支援版本。因此，如果您將新裝置新增至物件群組，或更新以這些裝置為目標的部署，則AWS提供之公用元件的新修補程式版本可能會自動部署到核心裝置。某些自動更新 (例如核心更新) 可能會導致裝置意外重新啟動。
>
> 若要避免對裝置上執行的元件進行意外更新，建議您在建立部署時直接包含該元件的偏好版本。如需有關 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱更新AWS IoT Greengrass核心軟件（OTA）。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.1.0 版本是可用的。 新功能 • 支援從 Amazon ECR 中的私有儲存庫下載 Docker 映像檔。 • 新增下列參數以自訂核心裝置上的 MQTT 組態： • `maxInFlightPublishes` — 可同時進行中未確認的 MQTT QoS 1 訊息的最大數量。 • `maxPublishRetry` — 重試失敗發佈之郵件的次數上限。 • 新增`fleetstatusservice` 組態參數以設定核心裝置將裝置狀態發佈至的間隔AWS 雲端。 • 其他小修正和改進。如需詳細資訊，請參閱中的發行版本 GitHub。 錯誤修復和改進 • 修正核心重新啟動時造成陰影部署重複的問題。 • 修正當核心發生服務載入例外狀況時造成當機的問題。 • 改善元件相依性解決方案，使包含循環相依性的部署失敗。 • 修正先前已從核心裝置移除外掛程式元件時，無法重新部署該元件的問題。 • 修正導致 Lambda 元件或以 root 身分執行之元件的`HOME`環境變數設定為*`/greengrass/v2`* `/work`目錄的問題。`HOME`變數現在已正確設定為執行元件之使用者的主目錄。 • 其他小修正和改進。如需詳細資訊，請參閱中的發行版本 GitHub。

元件	詳細資訊
碼頭應用程序管理器	新的 Docker 應用程序管理器組件的 2.0.0 版本可用。 新功能 • 管理登入資料以從 Amazon ECR 中的私有儲存庫下載映像。 • 從 Amazon ECR，碼頭集線器和. AWS Marketplace
发 Lambda 器	Lambda 啟動器元件的 2.0.4 版本可供使用。 錯誤修復和改進 • 修正元件未正確傳遞`AddGroupOwner` 至 Lambda 函數容器的問題。
舊版訂閱路由器	舊版訂閱路由器元件 2.1.0 版可供使用。 錯誤修復和改進 • 添加支援為`source`和`target`指定元件名稱而不是 ARN。如果您為訂閱指定元件名稱，則不需要在每次 Lambda 函數版本變更時重新設定訂閱。
本機除錯主控台	本地調試控制台組件的 2.1.0 版可用。 新功能 • 使用 HTTPS 來保護您與本機除錯主控台的連線安全。HTTPS 預設為啟用狀態。 錯誤修復和改進 • 您可以在設定編輯器中關閉快閃列訊息。

元件	詳細資訊
日誌管理器	日誌管理器組件的 2.1.0 版可用。 錯誤修復和改進 • 對於打印到標準輸出（標準輸出）logFileDirectoryPath 和logFileRegex 標準錯誤（stderr）的 Greengrass 組件使用默認值和工作。 • 將記錄檔上傳至 CloudWatch 記錄檔時，透過設定的網路 Proxy 正確路由傳送流量。 • 正確處理記錄資料流名稱中的冒號字元 (:)。 CloudWatch 記錄檔資料流名稱不支援冒號。 • 從記錄串流中移除物件群組名稱，以簡化記錄資料流名稱。 • 移除在正常行為期間列印的錯誤記錄訊息。
DLR 影像分類	DLR 影像分類元件的 2.1.1 版可供使用。 新功能 • 使用深度學習執行階段 v1.6.0。 • 在 Arch64 (AArch64) 平台上新增對範例影像分類的支援。這擴展了對運行 NVIDIA 傑特森的 Greengrass 核心設備的機器學習支持，例如傑特森納米。 • 啟用相機整合以進行範例推論。使用新的UseCamera 組態參數可啟用範例推論程式碼，以存取 Greengrass 核心裝置上的攝影機，並在擷取的映像上在本機執行推論。 • 新增將推論結果發佈至. AWS 雲端 使用新的PublishResultsOnTopic 組態參數來指定您要發佈結果的主題。 • 新增可讓您為要執行推論之影像指定自訂目錄的ImageDirectory 組態參數。 錯誤修復和改進 • 將推論結果寫入元件記錄檔，而不是個別的推論檔案。 • 使用 AWS IoT Greengrass Core 軟體記錄模組來記錄元件輸出。 • 使用讀AWS IoT Device SDK取零組件模型組態並套用模型組態變更。

元件	詳細資訊
DLR 物體偵測	DLR 物件偵測元件的 2.1.1 版可供使用。 新功能 • 使用深度學習執行階段 v1.6.0。 • 添加對在 Armv8（AArch64）平台上進行樣本對象檢測的支持。這擴展了對運行 NVIDIA 傑特森的 Greengrass 核心設備的機器學習支持，例如傑特森納米。 • 啟用相機整合以進行範例推論。使用新的`UseCamera` 組態參數可啟用範例推論程式碼，以存取 Greengrass 核心裝置上的攝影機，並在擷取的映像上在本機執行推論。 • 新增將推論結果發佈至. AWS 雲端 使用新的`PublishResultsOnTopic` 組態參數來指定您要發佈結果的主題。 • 新增可讓您為要執行推論之影像指定自訂目錄的`ImageDirectory` 組態參數。 錯誤修復和改進 • 將推論結果寫入元件記錄檔，而不是個別的推論檔案。 • 使用 AWS IoT Greengrass Core 軟體記錄模組來記錄元件輸出。 • 使用讀AWS IoT Device SDK取零組件模型組態並套用模型組態變更。
DLR 圖像分類模型商店	DLR 圖像分類模型存儲組件的 2.1.1 版可用。 新功能 • 為 Arch64 平台新增一個範例 ResNet -50 影像分類模型。這擴展了對運行 NVIDIA 傑特森的 Greengrass 核心設備的機器學習支持，例如傑特森納米。
DLR 物件偵測模型商店	DLR 對象檢測模型存儲組件的版本 2.1.1 是可用的。 新功能 • 新增適用於 AArch64 平台的範例 Yolov3 物件偵測模型。這擴展了對運行 NVIDIA 傑特森的 Greengrass 核心設備的機器學習支持，例如傑特森納米。

元件	詳細資訊
DLR 安裝程式	您可以使用 DLR 元件的 1.6.1 版本。 新功能 • 安裝深度學習執行階段 v1.6.0 及其相依性。 • 添加對在 ARV8（AArch64）平台上安裝 DLR 的支持。這擴展了對運行 NVIDIA 傑特森的 Greengrass 核心設備的機器學習支持，例如傑特森納米。 錯誤修復和改進 • 在虛擬環境AWS IoT Device SDK中安裝，以讀取元件組態並套用組態變更。 • 其他小錯誤修復和改進。
TensorFlow 精簡圖像分類	新的 TensorFlow Lite 圖像分類組件的 2.1.0 版可用。 新功能 • 使用 TensorFlow Lite 新增對範例影像分類推論的支援。
TensorFlow 精簡型物體偵測	新的 TensorFlow Lite 對象檢測組件的 2.1.0 版可用。 新功能 • 使用 TensorFlow Lite 添加對樣本對象檢測推論的支持。
TensorFlow 精簡版圖片分類模型店	新的 TensorFlow Lite 圖像分類模型商店組件的版本 2.1.0 可用。 新功能 • 使用 Lite 提供預先訓練的 MobileNet v1 量化模型，以進行樣本影像分類推論。 TensorFlow
TensorFlow 精簡型物件偵測模型商店	新的 TensorFlow Lite 對象檢測模型存儲組件的版本 2.1.0 可用。 新功能 • 提供在 COCO 數據集上訓練的預先訓練的單次射擊檢測（SSD）MobileNet 模型，以使用 TensorFlow Lite 進行樣本對象檢測推論。

元件	詳細資訊
TensorFlow 精簡版	新的TensorFlow 精簡版組件的 2.5.0 版本可用。 新功能 • 在虛擬環境中安裝TensorFlow 精簡版 v1.6.0 及其依賴關係 7，阿爾姆瓦 8（AArch64）和 x86_64 平台。

發行版本：AWS IoT Greengrass2021 年 3 月 9 日核心 v2.0.5 軟體更新

此版本提供了 Greengrass 核組件的 2.0.5 版本和更新提供的組件。AWS它修正了網路 Proxy 支援的問題，以及中國區域中 Greengrass 資料平面端點的AWS問題。

發行日期：二零二一年三月九日

公用元件更新

下表列出AWS提供的元件，其中包括新功能和更新的功能。

> **Important**
>
> 當您部署元件時，AWS IoT Greengrass會安裝該元件所有相依性的最新受支援版本。因此，如果您將新裝置新增至物件群組，或更新以這些裝置為目標的部署，則AWS提供之公用元件的新修補程式版本可能會自動部署到核心裝置。某些自動更新 (例如核心更新) 可能會導致裝置意外重新啟動。
> 若要避免對裝置上執行的元件進行意外更新，建議您在建立部署時直接包含該元件的偏好版本。如需有關 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱更新AWS IoT Greengrass核心軟件（OTA）。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.0.5 版本是可用的。

元件	詳細資訊
	錯誤修復和改進 • 下載AWS提供的元件時，透過設定的網路 Proxy 正確路由流量。 • 在AWS中國地區使用正確的 Greengrass 資料平面端點。

發行版本：AWS IoT Greengrass2021 年 2 月 4 日核心 v2.0.4 軟體更新

此版本提供了 Greengrass 核成分的 2.0.4 版本。它包含用於透過連接埠 443 設定 HTTPS 通訊的新`greengrassDataPlanePort`參數，並修正錯誤。現在，AWS IoT Greengrass核心軟體安裝程式執`iam:GetPolicy`行`sts:GetCallerIdentity`時需要最低 IAM 政策，以及執行時間`--provision true`。

發行日期：二零二一年二月四日

公用元件更新

下表列出AWS提供的元件，其中包括新功能和更新的功能。

> **Important**
>
> 當您部署元件時，AWS IoT Greengrass會安裝該元件所有相依性的最新受支援版本。因此，如果您將新裝置新增至物件群組，或更新以這些裝置為目標的部署，則AWS提供之公用元件的新修補程式版本可能會自動部署到核心裝置。某些自動更新 (例如核心更新) 可能會導致裝置意外重新啟動。
> 若要避免對裝置上執行的元件進行意外更新，建議您在建立部署時直接包含該元件的偏好版本。如需有關 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱更新AWS IoT Greengrass核心軟件（OTA）。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.0.4 版本是可用的。

元件	詳細資訊
	新功能 • 透過連接埠 443 啟用 HTTPS 流量。您可以使用核心元件 2.0.4 版的新`greengrassDataPlanePort` 組態參數，將 HTTPS 通訊設定為透過連接埠 443 傳輸，而非預設連接埠 8443。如需詳細資訊，請參閱 透過連接埠 443 設定 HTTPS。 • 新增工作路徑 recipe 變數。您可以使用此 recipe 變數來取得元件工作資料夾的路徑，您可以使用這些路徑在元件及其相依性之間共用檔案。如需詳細資訊，請參閱工作路徑 recipe 變數。 錯誤修復和改進 • 如果角色政策已存在，則防止建立權杖交換 AWS Identity and Access Management (IAM) 角色政策。 由於此變更，安裝程式現在需要`iam:GetPolicy` 和執行`sts:GetCallerIdentity` 時使用`--provision true` 。如需詳細資訊，請參閱 安裝程式佈建資源的最低 IAM 政策。 • 正確處理尚未成功註冊之部署的取消。 • 更新組態，以在復原部署時移除具有較新時間戳記的舊項目。 • 其他小修正和改進。如需詳細資訊，請參閱（詳見）的版本 GitHub。

從AWS IoT Greengrass版本 1 遷移

AWS IoT Greengrass Version 2是AWS IoT Greengrass核心軟體、API 和主控台的主要版本。 AWS IoT Greengrass V2引入了一些改進功能AWS IoT Greengrass V1，例如模組化應用程式、對大型裝置的部署，以及對其他平台的支援。

> **Note**
>
> 2023 年 6 月 30 日之後，AWS IoT Greengrass Version 1不再收到功能更新、增強功能、錯誤修正或安全性修補程式。如需詳細資訊，請參閱AWS IoT Greengrass V1維護政策。如果您使用AWS IoT Greengrass V1，我們強烈建議您移轉至AWS IoT Greengrass V2.

請遵循本指南中的指示，從移轉AWS IoT Greengrass V1到AWS IoT Greengrass V2。

我可以在 V2 上執行 V1 應用程式嗎？

大多數 V1 應用程式都可以在 V2 核心裝置上執行，而無需變更應用程式程式碼 如果您的 V1 應用程式使用下列功能，您將無法在 V2 上執行它們。

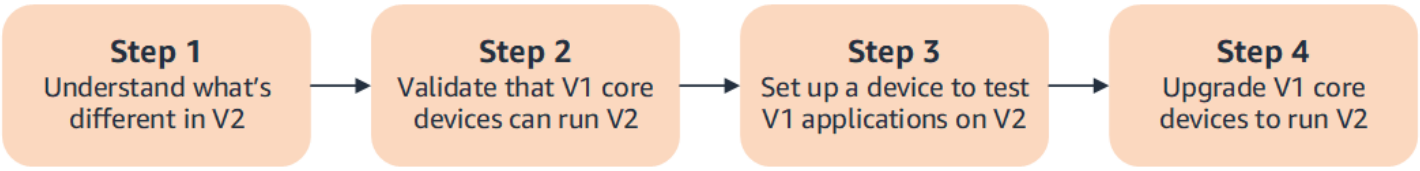
- C 和 C ++ Lambda 函數的運行時

如果您的 V1 應用程式使用下列任一功能，您必須修改應用程式程式碼，以使用 AWS IoT Device SDK V2 在上執行應用程式AWS IoT Greengrass V2。

- 與本機陰影服務互動
- 將訊息發佈到本機連線的裝置 (Greengrass 裝置)

移轉概觀

在高層級上，您可以使用下列程序將核心裝置從升級AWS IoT Greengrass V1到AWS IoT Greengrass V2。您遵循的確切程序取決於您環境的特定需求。



1. 了解 V1 和 V2 之間的差異

   AWS IoT Greengrass V2引入裝置叢集和可部署軟體的新基本概念，而 V2 則簡化了 V1 的數個概念。

   AWS IoT Greengrass V2雲服務和AWS IoT Greengrass核心軟件 v2.x 與AWS IoT Greengrass V1雲服務和AWS IoT Greengrass核心軟件 v1.x 不向後兼容。因此，AWS IoT Greengrass V1 over-the-air (OTA) 更新無法將核心裝置從 V1 升級到 V2。

2. 驗證 V1 核心裝置可以執行 V2

   驗證 V1 核心裝置是否可以執行AWS IoT Greengrass核心軟體 v2.x 和AWS IoT Greengrass V2功能。 AWS IoT Greengrass V2有不同的裝置需求與AWS IoT Greengrass V1.

3. 設定新裝置以在 V2 上測試 V1 應用程式

   若要將生產中裝置的風險降到最低，請建立新裝置以在 V2 上測試 V1 應用程式。安裝 AWS IoT Greengrass Core 軟體 v2.x 之後，您可以建立和部署AWS IoT Greengrass V2元件，以移轉和測試應用程式AWS IoT Greengrass V1。

4. 升級 V1 核心裝置以執行 V2

   升級現有的 V1 核心裝置以執行AWS IoT Greengrass核心軟體 v2.x 和AWS IoT Greengrass V2元件。若要將裝置叢集從 V1 移轉至 V2，請針對叢集中的每個裝置重複此步驟。

和之間 AWS IoT Greengrass V1 的差異 AWS IoT Greengrass V2

AWS IoT Greengrass V2 介紹裝置、叢集和可部署軟體的新基本概念。本節說明 V2 中不同的 V1 概念。

Greengrass 概念和術語

概念	AWS IoT Greengrass V1	AWS IoT Greengrass V2
應用程式碼	在中 AWS IoT Greengrass V1，Lambda 函數定義了在核心裝置上執行的軟體。在每個 Greengrass 群組中，您可以定義函數使用的訂閱和本機資源。對於 AWS IoT Greengrass 核心軟體在容器化	在中 AWS IoT Greengrass V2，元件是在核心裝置上執行的軟體模組。 • 每個元件都有一個方案，可定義元件的中繼資料、參數、相依性和指令碼，以便

概念	AWS IoT Greengrass V1	AWS IoT Greengrass V2
	Lambda 執行階段環境中執行的 Lambda 函數，您可以定義容器參數，例如記憶體限制。	在元件生命週期中的每個步驟執行。 • 方案也會定義元件的成品，也就是二進位檔案，例如指令碼、編譯的程式碼和靜態資源。 • 將元件部署到核心裝置時，核心裝置會下載元件配方和成品以執行元件。 您可以將 V1 Lambda 函數匯入為在中的 Lambda 執行階段環境中執行的元件 AWS IoT Greengrass V2。匯入 Lambda 函數時，您可以指定函數的訂閱、本機資源和容器參數。如需詳細資訊，請參閱步驟 2：建立和部署 AWS IoT Greengrass V2 元件以移轉 AWS IoT Greengrass V1 應用程式。 如需如何建立自訂元件的詳細資訊，請參閱〈〉開發AWS IoT Greengrass元件。

概念	AWS IoT Greengrass V1	AWS IoT Greengrass V2
AWS IoT Greengrass 群組和部署	在中 AWS IoT Greengrass V1，群組會定義核心裝置、該核心裝置的設定和軟體，以及可連線至該核心裝置的項目清單。 AWS IoT 您可以建立部署，將群組的設定傳送至核心裝置。	在中 AWS IoT Greengrass V2，您可以使用部署來定義在核心裝置上執行的軟體元件和組態。 • 每個部署都以單一核心裝置 (即 AWS IoT 物件) 或可包含多個核心裝置的 AWS IoT 物件群組為目標。 • 物件群組的部署是連續的，因此當您將核心裝置新增至物件群組時，它會接收該群組的軟體組態。 如需詳細資訊，請參閱 將AWS IoT Greengrass元件部署到裝置。 在中 AWS IoT Greengrass V2，您也可以使用 Greengrass CLI 建立本機部署，以在您開發這些元件的裝置上測試自訂軟體元件。如需詳細資訊，請參閱 建立 AWS IoT Greengrass 元件。

概念	AWS IoT Greengrass V1	AWS IoT Greengrass V2
AWS IoT Greengrass 核心軟體	在中 AWS IoT Greengrass V1， AWS IoT Greengrass Core 軟體是包含軟體及其所有功能的單一套件。您安裝 AWS IoT Greengrass 核心軟體的邊緣裝置稱為 Greengrass 核心。	在中 AWS IoT Greengrass V2， AWS IoT Greengrass Core 軟體是模組化的，因此您可以選擇要安裝的內容來控制記憶體佔用量。 • Greengrass 核組件是核心軟件的最低要求安裝。 AWS IoT Greengrass 您在其上安裝核心的邊緣裝置稱為 Greengrass 核心裝置。 • 核心處理核心裝置上其他元件的部署、協調和生命週期管理。 • 串流管理員、秘密管理員和記錄管理員等功能是您僅在需要這些功能時才部署的元件。如需詳細資訊，請參閱 AWS-提供的組件。

概念	AWS IoT Greengrass V1	AWS IoT Greengrass V2
連接器	在中 AWS IoT Greengrass V1，連接器是預先建置的模組，您可以部署到 AWS IoT Greengrass V1 核心裝置，以便與本機基礎結構、裝置通訊協定及其他雲端服務互動。AWS	在中 AWS IoT Greengrass V2， AWS 提供 Greengrass 元件，這些元件會實作 V1 中連接器所提供的功能。下列 AWS IoT Greengrass V2 元件提供 Greengrass V1 連接器功能： • CloudWatch 公制元件 • AWS IoT Device Defender 元件 • Firehose 組件 • 通訊協定介面卡元件 • Amazon SNS 組件 如需詳細資訊，請參閱 AWS-提供的組件。

概念	AWS IoT Greengrass V1	AWS IoT Greengrass V2
連接的設備（Greengrass 設備）	在中 AWS IoT Greengrass V1，已連接的裝置是 AWS IoT 您新增至 Greengrass 群組以連線至該群組中的核心裝置並透過 MQTT 進行通訊的項目。每次新增或移除連線的裝置時，都必須部署該群組。您可以使用訂閱在核心裝置上連線的裝置和應用程式之間轉送訊息。 AWS IoT Core	在中 AWS IoT Greengrass V2，連接的裝置稱為 Greengrass 用戶端裝置。 • 您可以將用戶端裝置與核心裝置建立關聯，以連接它們並透過 MQTT 進行通訊。 • 若要授權用戶端裝置進行連線，您需要定義可套用至用戶端裝置群組的授權原則，因此您不需要建立部署即可新增或移除用戶端裝置。 • 若要在用戶端裝置 AWS IoT Core、和 Greengrass 元件之間轉送訊息，您可以設定選用的 MQTT 橋接器元件。 在 AWS IoT Greengrass V1 和中 AWS IoT Greengrass V2，裝置可以執行 FreeRTOS 或使用AWS IoT Device SDK或 Greengrass 探索 API 來取得可連線的核心裝置相關資訊。Greengrass 探索 API 是向下相容的，因此，如果您的用戶端裝置連線到 V1 核心裝置，您可以將它們連接到 V2 核心裝置，而不需要變更其程式碼。 如需用戶端裝置的詳細資訊，請參閱與本機 IoT 裝置互動。

概念	AWS IoT Greengrass V1	AWS IoT Greengrass V2
本地資源	在中 AWS IoT Greengrass V1，在容器中執行的 Lambda 函數可設定為存取核心裝置檔案系統上的磁碟區和裝置。這些檔案系統資源稱為本機資源。	在中 AWS IoT Greengrass V2，您可以執行 Lambda 函數、Docker 容器或原生作業系統程序或自訂執行階段的元件。 • 將容器化 Lambda 函數匯入為元件時，必須指定函數使用的本機資源。 • 非容器化 Lambda 函數和非 Lambda 元件可直接與核心裝置上的本機資源搭配使用，因此您不需要指定元件使用的本機資源。
本機陰影服務	在中 AWS IoT Greengrass V1，本機陰影服務預設為啟用，且僅支援未命名的傳統陰影。您可以在 Lambda 函數中使用 AWS IoT Greengrass 核心 SDK 與裝置上的陰影互動。	在中 AWS IoT Greengrass V2，您可以透過部署陰影管理員元件來啟用本機陰影服務。 • 您可以在 Lambda 函數和自訂元件中使用 AWS IoT Device SDK V2，與裝置上的陰影互動。 • 本機陰影服務支援具名陰影。 • 本機陰影服務可讓您刪除陰影，並將已刪除的陰影與同步化 AWS IoT Core。 如需詳細資訊，請參閱 與裝置陰影互動。

概念	AWS IoT Greengrass V1	AWS IoT Greengrass V2

訂閱	在中 AWS IoT Greengrass V1，您可以定義 Greengrass 群組的訂閱，以指定 Lambda 函數、連接器、連線裝置、AWS IoT Core MQTT 代理程式和本機陰影服務之間的通訊通道。訂閱可指定 Lambda 函數接收事件訊息以作為函數承載使用的位置。	在中 AWS IoT Greengrass V2，您可以指定通訊管道，而不使用訂閱。
		• 元件會管理自己的通訊管道，以便與本機發佈/訂閱訊息、 AWS IoT Core MQTT 訊息和本機陰影服務互動。
		  • 若要開發對來自其他元件或 AWS IoT Core MQTT 代理程式之訊息做出反應的元件，您可以使用處理序間通訊 (IPC) 介面來進行本機發佈/訂閱訊息和 MQTT 訊息。AWS IoT Core
		  • 若要開發與本機陰影服務互動的元件，您可以使用本機陰影服務的 IPC 介面。
		  • 在元件組態中，您可以定義授權原則，以指定元件有權使用的主題和本機陰影。
		• 若要設定用戶端裝置、本機發佈/訂閱代理程式和 MQTT 代理程式之間的通訊通道，您可以設定並部署 AWS IoT Core MQTT 橋接器元件。MQTT 橋接器元件可讓您與元件中的用戶端裝置互動，並在用戶端裝置和. AWS IoT Core

概念	AWS IoT Greengrass V1	AWS IoT Greengrass V2
訪問其他 AWS 服務	在中 AWS IoT Greengrass V1，您可以將稱為群組角色的 AWS Identity and Access Management (IAM) 角色附加到 Greengrass 群組。群組角色定義了該群組核心裝置上 Lambda 函數和 AWS IoT Greengrass 功能用來存取的許可 AWS 服務。	在中 AWS IoT Greengrass V2，您可以將 AWS IoT 角色別名附加到 Greengrass 核心裝置。角色別名指向稱為權杖交換角色的 IAM 角色。權杖交換角色定義了核心裝置上 Greengrass 元件用來存取的權限。 AWS 服務如需更多詳細資訊，請參閱 授權核心設備與AWS服務。

驗證 V1 核心裝置可以執行 V2 軟體

AWS IoT Greengrass核心軟件 v2.x 與AWS IoT Greengrass核心軟件 v1.x 具有不同的要求。將 V1 核心裝置升級至 V2 之前，請先檢閱的裝置需求AWS IoT Greengrass V2。 AWS IoT Greengrass V2目前不支援使用 Yocto 專案的自訂 Linux 系統進行移轉。

您可以使用 AWS IoT Device Tester(IDT)AWS IoT Greengrass V2 來驗證裝置是否符合執行AWS IoT Greengrass核心軟體 v2.x 的需求。IDT 是一個可下載的測試框架，可在您的主機上運行並連接到要驗證的設備。依照指示使用 IDT 執行資AWS IoT Greengrass格套件。設定 IDT 時，您可以選擇驗證裝置是否支援選用功能，例如 Docker、機器學習 (ML)、資料串流管理以及硬體安全性整合。

如果 IDT 報告 V1 核心裝置的 V2 測試失敗或錯誤，您無法將該裝置從 V1 升級至 V2。

設定新的 V2 核心裝置以測試 V1 應用程式

設定新的 AWS IoT Greengrass V2 核心裝置，為您的 AWS IoT Greengrass V1 應用程 AWS式部署和測試提供的元件和 AWS Lambda 功能。您也可以使用此 V2 核心裝置來開發和測試在核心裝置上執行原生處理程序的其他自訂 Greengrass 元件。在 V2 核心裝置上測試應用程式之後，您可以將現有的 V1 核心裝置升級至 V2，並部署提供 V1 功能的 V2 元件。

步驟 1：在新設備 AWS IoT Greengrass V2 上安裝

在新裝置上安裝 AWS IoT Greengrass 核心軟體 v2.x。您可以依照入門教學課程來設定裝置，並學習如何開發和部署元件。本教學課程使用自動佈建來快速設定裝置。當您安裝 AWS IoT Greengrass 核

心軟體 v2.x 時，請指定要部署 Greengrass CLI 的`--deploy-dev-tools`引數，以便您可以直接在裝置上開發、測試和偵錯元件。如需有關其他安裝選項的詳細資訊，包括如何在 Proxy 後方安裝 AWS IoT Greengrass Core 軟體或使用硬體安全性模組 (HSM)，請參閱安裝 AWS IoT Greengrass 核心軟體。

(選擇性) 啟用記錄至 Amazon CloudWatch 日誌

若要讓 V2 核心裝置將日誌上傳到 Amazon CloudWatch Logs，您可以部署 AWS提供的日誌管理員元件。您可以使用 CloudWatch 記錄檔來檢視元件記錄，以便在不存取核心裝置的檔案系統的情況下偵錯和疑難排解。如需詳細資訊，請參閱 監控AWS IoT Greengrass日誌。

步驟 2：建立和部署 AWS IoT Greengrass V2 元件以移轉 AWS IoT Greengrass V1 應用程式

您可以在上執行大多數 AWS IoT Greengrass V1 應用程式 AWS IoT Greengrass V2。您可以將 Lambda 函數匯入為執行於上的元件 AWS IoT Greengrass V2，也可以使用AWS提供與 AWS IoT Greengrass 連接器相同功能的元件。

您還可以開發自定義組件來構建任何功能或運行時在 Greengrass 核心設備上運行。如需有關如何在本機開發和測試元件的資訊，請參閱建立 AWS IoT Greengrass 元件。

主題

- 匯入 V1 Lambda 函數
- 使用 V1 連接器
- 運行碼頭容器
- 執行機器學習推論
- Connect V1 Greengrass 設備
- 啟用本機陰影服務
- 與整合 AWS IoT SiteWise

匯入 V1 Lambda 函數

您可以將 Lambda 函數匯入為 AWS IoT Greengrass V2 元件。從下列方法中選擇：

- 將 V1 Lambda 函數直接匯入為綠色元件。
- 更新您的 Lambda 函數以使用 AWS IoT Device SDK v2 中的 Greengrass 程式庫，然後將 Lambda 函數匯入為 Greengrass 元件。

- 建立使用非 Lambda 程式碼和 AWS IoT Device SDK v2 的自訂元件，以實作與 Lambda 函數相同的功能。

如果您的 Lambda 函數使用諸如串流管理員或本機密等功能，您必須定義封裝這些功能的所 AWS提供元件的相依性。部署 Lambda 函數元件時，部署也會包含您定義為相依性之每個功能的元件。在部署中，您可以設定參數，例如要部署到核心裝置的密碼。並非所有 V1 功能都需要 V2 上的 Lambda 函數具有元件相依性。下列清單說明如何在 V2 Lambda 函數元件中使用 V1 功能。

- 使用其他 AWS 服務

  如果您的 Lambda 函數使用 AWS 認證向其他 AWS 服務發出請求，則核心裝置的權杖交換角色必須允許核心裝置執行 Lambda 函數使用的 AWS 作業。如需詳細資訊，請參閱 授權核心設備與AWS服務。
- 流管理器

  如果您的 Lambda 函數使用串流管理員`aws.greengrass.StreamManager`，請在匯入函數時指定為元件相依性。部署串流管理員元件時，請指定要為目標核心裝置設定的串流管理員參數。核心裝置的權杖交換角色必須允許核心裝置存取您搭配串流管理員使用的 AWS 雲端 目的地。如需詳細資訊，請參閱 串流管理員。
- 當地秘密

  如果您的 Lambda 函數使用本機密，請在匯入函數時指定`aws.greengrass.SecretManager`為元件相依性。部署 Secret Manager 元件時，請指定要部署到目標核心裝置的秘密資源。核心設備的令牌交換角色必須允許核心設備檢索要部署的秘密資源。如需詳細資訊，請參閱 秘密經理。

  部署 Lambda 函數元件時，請將其設定為具有允許在 V2 中使用 GetSecretValue IPC 作業的 IPC 授權政策。 AWS IoT Device SDK
- 局部陰影

  如果您的 Lambda 函數與本機陰影互動，您必須更新 Lambda 函數程式碼才能使用 AWS IoT Device SDK V2。您也必須在匯入函數時指定`aws.greengrass.ShadowManager`為元件相依性。如需詳細資訊，請參閱 與裝置陰影互動。

  部署 Lambda 函數元件時，請將其設定為具有 IPC 授權原則，以授與 V2 中使用陰影 IPC 作業的 AWS IoT Device SDK 權限。
- 訂閱
  - 如果您的 Lambda 函數訂閱來自雲端來源的訊息，請在匯入函數時將這些訂閱指定為事件來源。

- 如果您的 Lambda 函數訂閱來自其他 Lambda 函數的訊息，或者您的 Lambda 函數將訊息發佈到 AWS IoT Core 或其他 Lambda 函數，請在部署 Lambda 函數時設定和部署舊版訂閱路由器元件。部署舊版訂閱路由器元件時，請指定 Lambda 函數使用的訂閱。

  > **Note**
  >
  > 只有當 Lambda 函數使用 AWS IoT Greengrass Core SDK 中的函數時，才需要舊版訂閱路由器元件。`publish()`如果您更新 Lambda 函數程式碼以使用 AWS IoT Device SDK V2 中的處理序間通訊 (IPC) 介面，則不需要部署舊版訂閱路由器元件。如需詳細資訊，請參閱下列處理序間通訊服務：
  >
  > - 發佈/訂閱本地訊息
  > - 發布/訂閱MQTT 訊 AWS IoT Core 息

- 如果您的 Lambda 函數訂閱來自本機連線裝置的訊息，請在匯入函數時將這些訂閱指定為事件來源。您還必須設定和部署 MQTT 橋接器元件，以將連線裝置的郵件轉送到您指定為事件來源的本機發佈/訂閱主題。
- 如果您的 Lambda 函數將訊息發佈到本機連線的裝置，則必須更新 Lambda 函數程式碼，以使用 AWS IoT Device SDK V2 來發佈本機發佈/訂閱訊息。您也必須設定並部署 MQTT 橋接器元件，以將訊息從本機發佈/訂閱訊息代理程式轉送至連線的裝置。

- 本機磁碟區和裝置

  如果您的容器化 Lambda 函數存取本機磁碟區或裝置，請在匯入 Lambda 函數時指定這些磁碟區和裝置。此功能不需要組件依賴關係。

如需詳細資訊，請參閱 執行AWS Lambda函數。

使用 V1 連接器

您可以部署 AWS提供與某些 AWS IoT Greengrass 連接器相同功能的元件。建立部署時，您可以設定連接器的參數。

下列 AWS IoT Greengrass V2 元件提供 Greengrass V1 連接器功能：

- CloudWatch 公制元件
- AWS IoT Device Defender 元件
- Firehose 組件
- 通訊協定介面卡元件

- Amazon SNS 組件

運行碼頭容器

AWS IoT Greengrass V2 不提供直接取代 V1 Docker 應用程式部署連接器的元件。不過，您可以使用 Docker 應用程式管理員元件下載 Docker 映像，然後建立自訂元件，以便從下載的映像執行 Docker 容器。如需詳細資訊，請參閱 運行碼頭容器 及 碼頭應用程序管理器。

執行機器學習推論

AWS IoT Greengrass V2 提供安裝 Amazon SageMaker Edge 管理器代理程式的 Amazon SageMaker 邊緣管理器元件，並可讓您使用 SageMaker 新編譯的模型做為 Greengrass 核心裝置上的模型元件。AWS IoT Greengrass V2 還提供在您的設備上安裝深度學習運行時和 TensorFlow Lite 的組件。您可以使用對應的 DLR 和 TensorFlow Lite 模型和推論組件來執行樣本圖像分類和對象檢測推論。若要使用其他機器學習架構 (例如 MXNet 和) TensorFlow，您可以開發使用這些架構的自訂元件。

Connect V1 Greengrass 設備

中的已連線裝 AWS IoT Greengrass V1 置稱為中的用戶端裝置 AWS IoT Greengrass V2。 AWS IoT Greengrass V2 對用戶端裝置的支援向後相容 AWS IoT Greengrass V1，因此您可以將 V1 用戶端裝置連接到 V2 核心裝置，而無需變更其應用程式程式碼。若要讓用戶端裝置連線到 V2 核心裝置，請部署啟用用戶端裝置支援的 Greengrass 元件，並將用戶端裝置與核心裝置建立關聯。若要在用戶端裝置、 AWS IoT Core 雲端服務和 Greengrass 元件 (包括 Lambda 函數) 之間轉送訊息，請部署並設定 M QTT 橋接器元件。您可以部署 IP 偵測器元件以自動偵測連線資訊，也可以手動管理端點。如需詳細資訊，請參閱 與本機 IoT 裝置互動。

啟用本機陰影服務

在中 AWS IoT Greengrass V2，本機陰影服務是由 AWS提供的陰影管理員元件實作。 AWS IoT Greengrass V2 也包括對已命名陰影的支援。若要讓您的元件與本機陰影互動，並將陰影狀態同步到 AWS IoT Core、設定和部署陰影管理員元件，以及在元件程式碼中使用陰影 IPC 作業。如需詳細資訊，請參閱 與裝置陰影互動。

與整合 AWS IoT SiteWise

如果您使用 V1 核心裝置做為 AWS IoT SiteWise 閘道，請依照指示將新的 V2 核心裝置設定為 AWS IoT SiteWise 閘道。 AWS IoT SiteWise 提供可為您部署 AWS IoT SiteWise 元件的安裝指令碼。

步驟 3：測試您的 AWS IoT Greengrass V2 應用程式

在您建立 V2 元件並將其部署到新的 V2 核心裝置之後，請確認您的應用程式符合您的期望。您可以檢查設備的日誌以查看組件的標準輸出（stdout）和標準錯誤（stderr）消息。如需詳細資訊，請參閱 監控AWS IoT Greengrass日誌。

如果您將 Greengrass CLI 部署到核心裝置，您可以使用它來偵錯元件及其組態。如需詳細資訊，請參閱 Greengrass CLI 命令。

驗證應用程式在 V2 核心裝置上運作之後，您可以將應用程式的 Greengrass 元件部署到其他核心裝置。如果您開發了執行原生處理序或 Docker 容器的自訂元件，則必須先將這些元件發佈到 AWS IoT Greengrass 服務，才能將它們部署到其他核心裝置。

將 Greengrass V1 核心設備升級到 Greengrass V2

確認應用程式和元件可在AWS IoT Greengrass V2核心裝置上運作之後，您就可以在目前執行 v1.x 的裝置 (例如生產裝置) 上安裝 AWS IoT Greengrass Core 軟體 v2.x。然後，部署 Greengrass V2 組件以在設備上運行您的 Greengrass 應用程序。

若要將裝置叢集從 V1 升級至 V2，請針對每個要升級的裝置完成以下步驟。您可以使用物件群組將 V2 元件部署到核心裝置叢集。

> **Tip**
>
> 我們建議您建立指令碼，以自動化一組裝置的升級程序。如果您使用AWS Systems Manager來管理叢集，則可以使用 Systems Manager 在每個裝置上執行該指令碼，將叢集從 V1 升級到 V2。
> 如有關如何將升級程序自動化的問題，請連絡您的AWS企業 Support 代表。

步驟 1：安裝AWS IoT Greengrass核心軟件 v2.x

從下列選項中選擇，在 V1 AWS IoT Greengrass 核心裝置上安裝核心軟體 v2.x：

- 以更少的步驟升級

  若要以更少的步驟進行升級，您可以先解除安裝 v1.x 軟體，然後再安裝 v2.x 軟體。

- 以最短的停機時間降

  若要以最短的停機時間進行升級，您可以同時安裝這兩個版本的 AWS IoT Greengrass Core 軟體。安裝AWS IoT Greengrass核心軟體 v2.x 並確認您的 V2 應用程式是否正確運作之後，請解除安裝 AWS IoT Greengrass核心軟體 v1.x。在選擇此選項之前，請考慮同時執行這兩個版本的 AWS IoT Greengrass Core 軟體所需的額外 RAM。

在安裝 v2.x 之前，請先解除安裝AWS IoT Greengrass核心 v1.x

如果您要依序升級，請先解除安裝 AWS IoT Greengrass Core 軟體 v1.x，然後再將 v2.x 安裝到裝置上。

若要解除安裝AWS IoT Greengrass核心軟體 v1.x

1. 如果AWS IoT Greengrass核心軟體 v1.x 以服務的形式執行，您必須停止、停用及移除該服務。

   a. 停止執行中的AWS IoT Greengrass核心軟體 v1.x 服務。

   ```
   sudo systemctl stop greengrass
   ```

   b. 等到服務停止。您可以使用`list`命令來檢查服務的狀態。

   ```
   sudo systemctl list-units --type=service | grep greengrass
   ```

   c. 停用服務。

   ```
   sudo systemctl disable greengrass
   ```

   d. 移除服務。

   ```
   sudo rm /etc/systemd/system/greengrass.service
   ```

2. 如果AWS IoT Greengrass核心軟體 v1.x 並未以服務形式執行，請使用下列命令停止精靈。將綠根替換為您的 *Greengrass #*文件夾的名稱。預設位置為 /greengrass。

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

3. (選擇性) 將 Greengrass 根資料夾和自訂寫入資料夾 (如果適用) 備份至裝置上的其他資料夾。

   a. 使用以下命令將當前 Greengrass 根文件夾複製到不同的文件夾，然後刪除根文件夾。

```
sudo cp -r /greengrass-root /path/to/greengrass-backup
rm -rf /greengrass-root
```

b. 使用以下命令將寫入文件夾移動到不同的文件夾，然後刪除寫入文件夾。

```
sudo cp -r /write-directory /path/to/write-directory-backup
rm -rf /write-directory
```

然後，您可以使用的安裝說明在裝置上安裝軟體。AWS IoT Greengrass V2

> **Tip**
>
> 若要在將核心裝置從 V1 移轉至 V2 時重複使用核心裝置的身分識別，請遵循指示以手動佈建方式安裝 AWS IoT Greengrass Core 軟體。首先從裝置移除 V1 核心軟體，然後重複使用 V1 核心裝置的AWS IoT物件和憑證，並更新憑證的AWS IoT原則，以授與 v2.x 軟體所需的權限。

在已經執行 v1.x 的裝置上安裝AWS IoT Greengrass核心軟體 v2.x

如果您在已經執行AWS IoT Greengrass核心軟體 v1.x 的裝置上安裝 AWS IoT Greengrass Core v2.x 軟體，請記住下列事項：

- V2 核心裝置的AWS IoT物件名稱必須是唯一的。請勿使用與 V1 核心裝置相同的物件名稱。
- 您用於AWS IoT Greengrass核心軟體 v2.x 的連接埠必須與您用於 v1.x 的連接埠不同。
  - 將 V1 串流管理員設定為使用 8088 以外的連接埠。如需詳細資訊，請參閱設定串流管理員。
  - 將 V1 MQTT 代理程式設定為使用 8883 以外的連接埠。如需詳細資訊，請參閱設定本機訊息的 MQTT 連接埠。
- AWS IoT Greengrass V2不提供重命名 Greengrass 系統服務的選項。如果您以系統服務的形式執行 Greengrass，您必須執行下列其中一項動作，以避免發生衝突的系統服務名稱：
  - 在安裝 v2.x 之前，請重新命名 v1.x 的 Greengrass 服務。
  - 在沒有系統服務的情況下安裝 AWS IoT Greengrass Core 軟體 v2.x，然後手動將軟體設定為具有以外名稱的系統服務。`greengrass`

  若要重新命名 v1.x 的 Greengrass 服務

  1. 停止AWS IoT Greengrass核心軟體 v1.x 服務。

```
sudo systemctl stop greengrass
```

2. 等待服務停止。該服務最多可能需要幾分鐘才能停止。您可以使用命`list-units`令來檢查服務是否已停止。

```
sudo systemctl list-units --type=service | grep greengrass
```

3. 停用服務。

```
sudo systemctl disable greengrass
```

4. 重新命名服務。

```
sudo mv /etc/systemd/system/greengrass.service /etc/systemd/system/greengrass-v1.service
```

5. 重新載入服務並啟動它。

```
sudo systemctl daemon-reload
sudo systemctl reset-failed
sudo systemctl enable greengrass-v1
sudo systemctl start greengrass-v1
```

然後，您可以使用的安裝說明在裝置上安裝軟體。AWS IoT Greengrass V2

> **Tip**
>
> 若要在將核心裝置從 V1 移轉至 V2 時重複使用核心裝置的身分識別，請遵循指示以手動佈建方式安裝 AWS IoT Greengrass Core 軟體。首先從裝置移除 V1 核心軟體，然後重複使用 V1 核心裝置的AWS IoT物件和憑證，並更新憑證的AWS IoT原則，以授與 v2.x 軟體所需的權限。

步驟 2：將AWS IoT Greengrass V2元件部署到核心裝置

在裝置上安裝 AWS IoT Greengrass Core 軟體 v2.x 之後，請建立包含下列資源的部署。若要將元件部署到類似裝置的叢集，請為包含這些裝置的物群組建立部署。

- 您從 V1 Lambda 函數建立的 Lambda 函數元件。如需詳細資訊，請參閱 執行AWS Lambda函數。

- 如果您使用 V1 訂閱，則為舊版訂閱路由器元件。
- 如果您使用串流管理員，則為串流管理員元件。如需詳細資訊，請參閱 管理核心裝 Greengrass 資料串流。
- 如果您使用本地密碼，密碼管理器組件。
- 如果您使用 V1 連接器，則是AWS提供的連接器元件。
- 如果您使用 Docker 容器，Doc ker 應用程序管理器組件。如需詳細資訊，請參閱 運行碼頭容器。
- 如果您使用機器學習推論，也就是機器學習支援的元件。如需詳細資訊，請參閱 執行機器學習推論。
- 如果您使用連線的裝置，用戶端裝置支援的元件。您也必須啟用用戶端裝置支援，並將用戶端裝置與核心裝置建立關聯。如需詳細資訊，請參閱 與本機 IoT 裝置互動。
- 如果您使用裝置陰影，則為陰影管理員元件。如需詳細資訊，請參閱 與裝置陰影互動。
- 如果您將日誌從 Greengrass 核心裝置上傳到 Amazon CloudWatch 日誌，即日誌管理器元件。如需詳細資訊，請參閱 監控AWS IoT Greengrass日誌。
- 如果與整合AWS IoT SiteWise，請遵循指示將 V2 核心裝置設定為AWS IoT SiteWise閘道。 AWS IoT SiteWise提供可為您部署AWS IoT SiteWise元件的安裝指令碼。
- 您為實作自訂功能而開發的使用者定義元件。

若要取得有關建立和修訂部署的資訊，請參閱將AWS IoT Greengrass元件部署到裝置。

教學課程：AWS IoT Greengrass V2 入門

您可以完成此入門教學課程，以了解的基本功能AWS IoT Greengrass V2。在此教學課程中，您將執行下列操作：

1. 在 Linux 設備上安裝和配置AWS IoT Greengrass核心軟件，例如樹莓派或 Windows 設備。這個設備是一個 Greengrass 核心設備。
2. 在您的 Greengrass 核心設備上開發你好世界組件。組件是在 Greengrass 核心設備上運行的軟件模塊。
3. 將該元件上傳至AWS IoT Greengrass V2中的AWS 雲端.
4. 將該元件從部署AWS 雲端到您的 Greengrass 核心裝置。

> **Note**
>
> 本教學課程說明如何設定開發環境並探索的功能AWS IoT Greengrass。如需如何設定和設定生產裝置的詳細資訊，請參閱下列內容：
>
> - 設定AWS IoT Greengrass核心裝置
> - 安裝 AWS IoT Greengrass 核心軟體

您可以期望在本教程中花費 20 到 30 分鐘。

主題

必要條件

若要完成此入門教學課程，您需要以下項目：

- AWS 帳戶。如果您沒有帳戶，請參閱 步驟 1：設定 AWS 帳戶。
- 使用一個AWS 區域支持AWS IoT Greengrass V2. 如需支援區域的清單，請參閱《AWS 一般參考》中的 AWS IoT Greengrass V2 端點和配額。
- 具有管理員權限的 AWS Identity and Access Management (IAM) 使用者。
- 設置為 Greengrass 核心設備的設備, 如樹莓派與樹莓派操作系統 (以前稱為覆盆子), 或視窗 10 設備. 您必須擁有此裝置的管理員權限，或是取得管理員權限的能力，例如透過sudo。此裝置必須具有網際網路連線。

  您也可以選擇使用符合需求的其他裝置來安裝和執行 AWS IoT Greengrass Core 軟體。如需詳細資訊，請參閱 支援平台和需求。

  如果您的開發電腦符合這些需求，您可以在本教學中將其設定為 Greengrass 核心裝置。
- Python 3.5 或更高版本為設備上的所有用戶安裝並添加到PATH環境變量中。在視窗上，您還必須為所有使用者安裝適用於視窗的 Python 啟動器。

> **Important**
>
> 在視窗中，默認情況下不會為所有用戶安裝 Python。當您安裝 Python 時，您必須自定義安裝以配置它，以便AWS IoT Greengrass核心軟件運行 Python 腳本。例如，如果您使用圖形化 Python 安裝程式，請執行下列動作：
>
> 1. 選取 [為所有使用者安裝啟動器 (建議)]。
> 2. 選擇 Customize installation。
> 3. 選擇 Next。
> 4. 選取 Install for all users。
> 5. 選取 Add Python to environment variables。
> 6. 選擇 Install (安裝)。
>
> 如需詳細資訊，請參閱 Python 3 文件中的在視窗上使用 Python。

- AWS Command Line Interface（AWS CLI）在您的開發計算機和設備上安裝和配置憑據。確保您使用相同AWS 區域的AWS CLI在開發計算機和設備上配置。若要AWS IoT Greengrass V2搭配使用AWS CLI，您必須具有下列其中一個版本或更新版本：
  - 最低版本AWS CLI第一版本：
  - 最AWS CLI低版本：

> **Tip**
>
> 您可以運行以下命令來檢查您擁有AWS CLI的版本。
>
> ```
> aws --version
> ```

如需詳細資訊，請參閱《AWS Command Line Interface使用指南》AWS CLI中的〈安裝、更新AWS CLI和解除安裝〉和〈設定〉。

> **Note**
>
> 如果您使用 32 位元 ARM 裝置 (例如具有 32 位元作業系統的樹莓派)，請安裝 AWS CLI V1。 AWS CLIV2 不適用於 32 位元 ARM 裝置。如需詳細資訊，請參閱安裝、更新和解除安裝AWS CLI版本 1。

步驟 1：設定 AWS 帳戶

註冊 AWS 帳戶

如果您還沒有 AWS 帳戶，請完成以下步驟建立新帳戶。

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，並在電話鍵盤輸入驗證碼。

註冊 AWS 帳戶時，會建立 AWS 帳戶根使用者。根使用者有權存取該帳戶中的所有 AWS 服務和資源。作為最佳安全實務，將管理存取權指派給管理使用者，並且僅使用根使用者來執行需要根使用者存取權的任務。

註冊程序完成後，AWS 會傳送一封確認電子郵件給您。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。

建立管理使用者

當您註冊 AWS 帳戶 之後，請保護您的 AWS 帳戶根使用者，啟用 AWS IAM Identity Center，並建立管理使用者，讓您可以不使用根使用者處理日常作業。

保護您的 AWS 帳戶根使用者

1. 選擇 根使用者 並輸入您的 AWS 帳戶電子郵件地址，以帳戶擁有者身分登入 AWS Management Console。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)。

   如需指示，請參閱《IAM 使用者指南》中的為 AWS 帳戶根使用者啟用虛擬 MFA 裝置 (主控台)。

建立管理使用者

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，將管理權限授予管理使用者。

   若要取得有關使用 IAM Identity Center 目錄 做為身分識別來源的教學課程，請參閱《使用 AWS IAM Identity Center 使用者指南中的以預設 IAM Identity Center 目錄 設定使用者存取權限。

以管理員的身分登入

- 若要使用您的 IAM 身分中心使用者登入，請使用建立 IAM 身分中心使用者時傳送至您電子郵件地址的登入 URL。

如需有關如何使用 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。

步驟 2：設定環境

請依照本節中的步驟設定 Linux 或 Windows 裝置作為AWS IoT Greengrass核心裝置使用。

設置一個 Linux 設備 (樹莓派)

這些步驟假設您使用樹莓派與樹莓派作業系統. 如果您使用不同的裝置或作業系統，請參閱您裝置的相關文件。

若要設定樹莓派 AWS IoT Greengrass V2

1. 在樹莓派上啟用 SSH 以遠程連接到它。如需詳細資訊，請參閱樹莓派文件中的 SSH (安全殼層)。
2. 找到您的樹莓派的 IP 地址連接到它與 SSH. 為此，您可以在 Raspberry Pi 上運行以下命令。

   ```
   hostname -I
   ```

3. 使用 SSH Connect 到您的樹莓派。

   在您的開發電腦上，執行下列命令。*####################pi-ip-address############ IP #####*

   ```
   ssh username@pi-ip-address
   ```

   > **Important**
   >
   > 如果您的開發計算機使用舊版 Windows，則可能沒有該ssh命令，或者您可能有ssh但無法連接到 Raspberry Pi。要連接到樹莓派，您可以安裝和配置 PuTTY，這是一個免費的開源 SSH 客戶端。請參閱 PuTTY 文檔以連接到您的樹莓派。

4. 安裝 Java 執行階段，AWS IoT Greengrass核心軟體需要執行。在樹莓派上，使用以下命令安裝 Java 11。

   ```
   sudo apt install default-jdk
   ```

   安裝完成後，請執行下列命令以驗證 Java 是否在您的樹莓派上執行。

```
java -version
```

此指令會列印在裝置上執行的 Java 版本。輸出看起來可能類似於下列範例。

```
openjdk version "11.0.9.1" 2020-11-04
OpenJDK Runtime Environment (build 11.0.9.1+1-post-Debian-1deb10u2)
OpenJDK 64-Bit Server VM (build 11.0.9.1+1-post-Debian-1deb10u2, mixed mode)
```

提示：在樹莓派上設置內核參數

如果您的設備是 Raspberry Pi，則可以完成以下步驟來查看和更新其 Linux 內核參數：

1. 開啟 `/boot/cmdline.txt` 檔案。此檔案指定樹莓派啟動時要套用的 Linux 核心參數。

   例如，在 Linux 系統上，您可以執行下列命令來使用 GNU nano 來開啟檔案。

   ```
   sudo nano /boot/cmdline.txt
   ```

2. 確認`/boot/cmdline.txt`檔案包含下列核心參數。
   此`systemd.unified_cgroup_hierarchy=0`參數會指定使用 cgroup v1 而不是使用 cgroup v2。

   ```
   cgroup_enable=memory cgroup_memory=1 systemd.unified_cgroup_hierarchy=0
   ```

   如果`/boot/cmdline.txt`檔案不包含這些參數，或包含這些具有不同值的參數，請更新檔案以包含這些參數和值。

3. 如果您更新了`/boot/cmdline.txt`文件，請重新啟動 Raspberry Pi 以應用更改。

   ```
   sudo reboot
   ```

設定 Linux 裝置 (其他)

若要設定下列項目的 Linux 設備 AWS IoT Greengrass V2

1. 安裝 Java 執行階段，AWS IoT Greengrass核心軟體需要執行。我們建議您使用 Amazon Corretto 或 OpenJDK 長期支持版本。需要版本 8 或更高版本。以下命令向您展示如何在設備上安裝 OpenJDK。

   - 針對 Debian 為基礎或 Ubuntu 為基礎的發行版本：

     ```
     sudo apt install default-jdk
     ```

   - 針對 Red Hat 為基礎的發行版本：

     ```
     sudo yum install java-11-openjdk-devel
     ```

   - 針對 Amazon Linux 2：

     ```
     sudo amazon-linux-extras install java-openjdk11
     ```

   - 對於 Amazon

     ```
     sudo dnf install java-11-amazon-corretto -y
     ```

   安裝完成時，請執行下列命令以確認 Java 是否在 Linux 裝置上執行。

   ```
   java -version
   ```

   此指令會列印在裝置上執行的 Java 版本。例如，在基於 Debian 的發行版上，輸出可能類似於以下示例。

   ```
   openjdk version "11.0.9.1" 2020-11-04
   OpenJDK Runtime Environment (build 11.0.9.1+1-post-Debian-1deb10u2)
   OpenJDK 64-Bit Server VM (build 11.0.9.1+1-post-Debian-1deb10u2, mixed mode)
   ```

2. (選擇性) 建立在裝置上執行元件的預設系統使用者和群組。您也可以選擇讓 AWS IoT Greengrass Core 軟體安裝程式在安裝期間使用安裝程式引數建立此使用者和群組。`--component-default-user`如需詳細資訊，請參閱 安裝器引數。

```
sudo useradd --system --create-home ggc_user
sudo groupadd --system ggc_group
```

3. 確認執行 AWS IoT Greengrass Core 軟體的使用者 (通常`root`) 具有`sudo`與任何使用者和任何群組一起執行的權限。

   a. 執行下列命令以開啟`/etc/sudoers`檔案。

   ```
   sudo visudo
   ```

   b. 確認使用者的權限看起來像下列範例。

   ```
   root    ALL=(ALL:ALL) ALL
   ```

4. (選擇性) 若要執行容器化 Lambda 函數，您必須啟用 cgroup v1，並且必須啟用並掛接記憶體和裝置 cgroup。如果您不打算執行容器化 Lambda 函數，則可以略過此步驟。

   若要啟用這些 cgroups 選項，請使用下列 Linux 核心參數來啟動裝置。

   ```
   cgroup_enable=memory cgroup_memory=1 systemd.unified_cgroup_hierarchy=0
   ```

   如需有關檢視和設定裝置核心參數的資訊，請參閱作業系統和開機載入程式的說明文件。依照指示永久設定核心參數。

5. 依照中的需求清單所指示，在您的裝置上安裝所有其他必要的相依性裝置要求。

設定視窗裝置

若要設定下列項目的視窗裝置 AWS IoT Greengrass V2

1. 安裝 Java 執行階段，AWS IoT Greengrass核心軟體需要執行。我們建議您使用 Amazon Corretto 或 OpenJDK 長期支持版本。需要版本 8 或更高版本。
2. 檢查系統變數 PATH 上是否可用 Java，如果沒有，則加入它。此 LocalSystem 帳戶會執行AWS IoT Greengrass核心軟體，因此您必須將 Java 新增至系統變數 PATH，而非使用者的 PATH 使用者變數。請執行下列操作：

   a. 按視窗鍵開啟開始功能表。

   b. 鍵入**`environment variables`**以從開始功能表搜尋系統選項。

c. 在開始功能表搜尋結果中，選擇 [編輯系統環境變數] 以開啟 [系統屬性] 視窗。

d. 選擇環境變數... 以開啟 [環境變數] 視窗。

e. 在 [系統變數] 下，選取 [路徑]，然後選擇 [編輯]。在 「編輯」環境變數視窗中，您可以在不同的行上檢視每個路徑。

f. 檢查 Java 安裝bin資料夾的路徑是否存在。路徑看起來可能類似下列範例。

```
C:\\Program Files\\Amazon Corretto\\jdk11.0.13_8\\bin
```

g. 如果路徑中缺少 Java 安裝的bin資料夾，請選擇 [新增] 加入，然後選擇 [確定]。

3. 以系統管理員身分開啟 Windows 命令提示字元 (cmd.exe)。

4. 在 Windows 裝置上的 LocalSystem 帳戶中建立預設使用者。以安全*##*取代密碼。

```
net user /add ggc_user password
```

> **Tip**
>
> 視您的 Windows 組態而定，使用者的密碼可能會設定為在 future 的某個日期到期。為確保您的 Greengrass 應用程式繼續運作，請追蹤密碼何時到期，並在密碼到期之前進行更新。您也可以將使用者的密碼設定為永不過期。
>
> - 若要檢查使用者及其密碼何時到期，請執行下列命令。
>
>   ```
>   net user ggc_user | findstr /C:expires
>   ```
>
> - 若要將使用者的密碼設定為永不過期，請執行下列命令。
>
>   ```
>   wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
>   ```
>
> - 如果您使用的是 Windows 10 或更新版本，而該wmic命令已被取代，請執行下列 PowerShell 命令。
>
>   ```
>   Get-CimInstance -Query "SELECT * from Win32_UserAccount WHERE name =
>    'ggc_user'" | Set-CimInstance -Property @{PasswordExpires="False"}
>   ```

5. 在設備上從 Microsoft 下載並安裝該PsExec實用程序。

6. 使用此 PsExec 公用程式將預設使用者的使用者名稱和密碼儲存在 LocalSystem 帳戶的認證管理員執行個體中。*################*

```
psexec -s cmd /c cmdkey /generic:ggc_user /user:ggc_user /pass:password
```

如果打PsExec License Agreement開，請選Accept擇同意許可證並運行命令。

> **Note**
>
> 在 Windows 裝置上， LocalSystem 帳戶會執行 Greengrass 核心，而且您必須使用公用 PsExec 程式將預設使用者資訊儲存在帳戶中。 LocalSystem 使用認證管理員應用程式會將此資訊儲存在目前登入使用者的 Windows 帳戶中，而非 LocalSystem 帳戶中。

步驟 3：安裝AWS IoT Greengrass核心軟體

請按照本節中的步驟將 Raspberry Pi 設置為可用於本地開發的AWS IoT Greengrass核心設備。在本節中，您將下載並執行安裝程式，該安裝程式會執行下列動作來為您的裝置設定AWS IoT Greengrass Core 軟體：

- 安裝 Greengrass 組件。核心是一個強制性組件，是在設備上運行AWS IoT Greengrass核心軟件的最低要求。如需詳細資訊，請參閱 Greengrass 詳情。
- 將您的裝置註冊為AWS IoT物件，並下載可讓裝置連線的數位憑證AWS。如需詳細資訊，請參閱AWS IoT Greengrass 的裝置身分驗證和授權。
- 將裝置的物AWS IoT件新增至物件群組 (物件群組或AWS IoT物件群組)。物件群組可讓您管理 Greengrass 核心裝置的叢集。將軟體元件部署到裝置時，您可以選擇部署到個別裝置或裝置群組。如需詳細資訊，請參閱AWS IoT Core開發人員指南AWS IoT中的使用管理裝置。
- 建立可讓 Greengrass 核心裝置與AWS服務互動的 IAM 角色。依預設，此角色可讓您的裝置與日誌互動，AWS IoT並將日誌傳送到 AmazonCloudWatch 日誌。如需詳細資訊，請參閱授權核心設備與AWS服務。
- 安裝指AWS IoT Greengrass令行介面 (`greengrass-cli`)，您可以使用它來測試您在核心裝置上開發的自訂元件。如需詳細資訊，請參閱綠色命令行界面。

安裝AWS IoT Greengrass核心軟體 (主控台)

1. 登入 AWS IoT Greengrass 主控台。
2. 在 [開始使用 Greengrass] 下，選擇 [設定一個核心裝置]。

3. 在步驟 1：註冊 Greengrass 核心裝置下，針對核心裝置名稱，輸入 Greengrass 核心裝置AWS IoT的項目名稱。如果該物件不存在，安裝程序將創建它。
4. 在「步驟 2：新增至物件群組」下以套用持續部署，針對「物件」群組，選擇您要新增核心裝置的AWS IoT物件群組。

   - 如果您選取 [輸入新群組名稱]，則在 [物件群組名稱] 中，輸入要建立之新群組的名稱。安裝程式會為您建立新群組。
   - 如果您選取 「選取現有群組」，則在「物件群組名稱」中，選擇您要使用的現有群組。
   - 如果您選取 [無群組]，則安裝程式不會將核心裝置新增至物件群組。
5. 在「步驟 3：安裝核心軟體」下，完成以下步驟。

   a. 選擇您的核心設備的操作系統：Linux 或視窗。

   b. 將您的AWS登入資料提供給裝置，以便安裝程式可以為您的核心裝置佈建AWS IoT和 IAM 資源。為了提高安全性，我們建議您取得 IAM 角色的臨時登入資料，該角色僅允許佈建所需的最低許可。如需詳細資訊，請參閱 安裝程式佈建資源的最低 IAM 政策。

      > **Note**
      >
      > 安裝程式不會儲存或儲存您的認證。

      在您的裝置上，執行下列其中一項動作以擷取認證，並將其提供給 AWS IoT Greengrass Core 軟體安裝程式：

      - (建議) 使用來源的暫時認證 AWS IAM Identity Center

        i. 從 IAM 身分中心提供存取金鑰 ID、秘密存取金鑰和工作階段權杖。如需詳細資訊，請參閱 IAM 身分中心使用指南中的取得和重新整理臨時登入資料中的手動登入資料重新整

        ii. 執行下列命令，將認證提供給 AWS IoT Greengrass Core 軟體。

        Linux or Unix

        ```
        export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
        export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/
        bPxRfiCYEXAMPLEKEY
        export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o5OytwEXAMPLE=
        ```

Windows Command Prompt (CMD)

```
set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
set AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o5OytwEXAMPLE=
```

PowerShell

```
$env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
$env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/
bPxRfiCYEXAMPLEKEY"
$env:AWS_SESSION_TOKEN="AQoDYXdzEJr1K...o5OytwEXAMPLE="
```

- 使用 IAM 角色的臨時安全登入資料：

  i. 從您假設的 IAM 角色提供存取金鑰 ID、秘密存取金鑰和工作階段權杖。如需如何擷取這些登入資料的詳細資訊，請參閱 IAM 使用者指南中的要求臨時安全登入資料。

  ii. 執行下列命令，將認證提供給 AWS IoT Greengrass Core 軟體。

  Linux or Unix

  ```
  export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
  export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/
  bPxRfiCYEXAMPLEKEY
  export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o5OytwEXAMPLE=
  ```

  Windows Command Prompt (CMD)

  ```
  set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
  set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
  set AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o5OytwEXAMPLE=
  ```

  PowerShell

  ```
  $env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
  $env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/
  bPxRfiCYEXAMPLEKEY"
  $env:AWS_SESSION_TOKEN="AQoDYXdzEJr1K...o5OytwEXAMPLE="
  ```

- 使用 IAM 使用者的長期登入資料：

i. 為您的 IAM 使用者提供存取金鑰 ID 和秘密存取金鑰。您可以為稍後刪除的佈建建立 IAM 使用者。如需提供給使用者的 IAM 政策，請參閱安裝程式佈建資源的最低 IAM 政策。如需如何擷取長期登入資料的詳細資訊，請參閱《IAM 使用者指南》中的「管理 IAM 使用者存取金鑰」。

ii. 執行下列命令，將認證提供給 AWS IoT Greengrass Core 軟體。

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

```
$env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
$env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/
bPxRfiCYEXAMPLEKEY"
```

iii. (選擇性) 如果您建立 IAM 使用者來佈建 Greengrass 裝置，請刪除該使用者。

iv. (選用) 如果您使用現有 IAM 使用者的存取金鑰 ID 和秘密存取金鑰，請更新使用者的金鑰，使其不再有效。如需詳細資訊，請參閱AWS Identity and Access Management使用指南中的更新存取金鑰。

c. 在 [執行安裝程式] 下，完成下列步驟。

i. 在 [下載安裝程式] 下，選擇 [複製]，然後在核心裝置上執行複製的命令。此命令會下載最新版本的 AWS IoT Greengrass Core 軟體，並將其解壓縮到您的裝置上。

ii. 在 [執行安裝程式] 下，選擇 [複製]，然後在核心裝置上執行複製的命令。此命令會使用 AWS IoT您先前指定的物件和物件群組名稱來執行 AWS IoT Greengrass Core 軟體安裝程式，並為核心裝置設定AWS資源。

此指令也會執行下列作業：

- 將 AWS IoT Greengrass Core 軟體設定為在開機時執行的系統服務。在 Linux 設備上，這需要系統初始化系統。

  > **Important**
  >
  > 在 Windows 核心裝置上，您必須將AWS IoT Greengrass核心軟體設定為系統服務。

- 部署 AWS IoT GreengrassCLI 元件，這是一個命令列工具，可讓您在核心裝置上開發自訂 Greengrass 元件。
- 指定使用`ggc_user`系統使用者在核心裝置上執行軟體元件。在 Linux 裝置上，此指令也會指定使用`ggc_group`系統群組，而安裝程式會為您建立系統使用者和群組。

當您執行此命令時，您應該會看到下列訊息，指出安裝程式已成功。

```
Successfully configured Nucleus with provisioned resource details!
Configured Nucleus to deploy aws.greengrass.Cli component
Successfully set up Nucleus as a system service
```

> **Note**
>
> 如果您的 Linux 裝置沒有 systemd，安裝程式就不會將軟體設定為系統服務，而且您也不會看到將核心設定為系統服務的成功訊息。

安裝AWS IoT Greengrass核心軟體 (CLI)

若要安裝和設定AWS IoT Greengrass核心軟體

1. 在 Greengrass 核心裝置上，執行下列命令以切換至主目錄。

   Linux or Unix

   ```
   cd ~
   ```

Windows Command Prompt (CMD)

```
cd %USERPROFILE%
```

PowerShell

```
cd ~
```

2. 在您的核心裝置上，將 AWS IoT Greengrass Core 軟體下載至名為`greengrass-nucleus-latest.zip`.

   Linux or Unix

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip > greengrass-nucleus-latest.zip
   ```

   Windows Command Prompt (CMD)

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip > greengrass-nucleus-latest.zip
   ```

   PowerShell

   ```
   iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip -OutFile greengrass-nucleus-latest.zip
   ```

   下載此軟體，即表示您同意 Greengrass 核心軟體授權合約之規定。

3. 將 AWS IoT Greengrass Core 軟件解壓縮到設備上的文件夾。以您要使用的資料夾取*`GreengrassInstaller`*代。

   Linux or Unix

   ```
   unzip greengrass-nucleus-latest.zip -d GreengrassInstaller && rm greengrass-nucleus-latest.zip
   ```

Windows Command Prompt (CMD)

```
mkdir GreengrassInstaller && tar -xf greengrass-nucleus-latest.zip -
C GreengrassInstaller && del greengrass-nucleus-latest.zip
```

PowerShell

```
Expand-Archive -Path greengrass-nucleus-latest.zip -DestinationPath .\
\GreengrassInstaller
rm greengrass-nucleus-latest.zip
```

4. 執行下列命令以啟動 AWS IoT Greengrass Core 軟體安裝程式。此命令會執行下列動作：

   - 建立核心裝置運作所需的AWS資源。
   - 將 AWS IoT Greengrass Core 軟體設定為在開機時執行的系統服務。在 Linux 設備上，這需要系統初始化系統。

   > **Important**
   >
   > 在 Windows 核心裝置上，您必須將AWS IoT Greengrass核心軟體設定為系統服務。

   - 部署 AWS IoT GreengrassCLI 元件，這是一個命令列工具，可讓您在核心裝置上開發自訂 Greengrass 元件。
   - 指定使用`ggc_user`系統使用者在核心裝置上執行軟體元件。在 Linux 裝置上，此指令也會指定使用`ggc_group`系統群組，而安裝程式會為您建立系統使用者和群組。

   如下所示取代指令中的引數值。

   a. */greengrass/v2*或 *C:\greengrass\v2*：用來安裝 AWS IoT Greengrass Core 軟體的根資料夾路徑。

   b. *GreengrassInstaller*。解壓縮 AWS IoT Greengrass Core 軟體安裝程式的資料夾路徑。

   c. *##*。要AWS 區域在其中尋找或建立資源。

   d. *MyGreengrassCore*。您 Greengrass 核心裝置AWS IoT的項目名稱。如果該物件不存在，安裝程序將創建它。安裝程式會下載憑證以進行驗證。AWS IoT如需詳細資訊，請參閱 AWS IoT Greengrass 的裝置身分驗證和授權。

> **Note**
>
> 物件名稱不能包含冒號 (:) 字元。

e. *MyGreengrassCoreGroup*。您 Greengrass 核心裝置的AWS IoT物件群組名稱。如果物件群組不存在，安裝程式會建立該物件並將該物件新增至其中。如果物群組存在且具有使用中部署，則核心裝置會下載並執行部署指定的軟體。

> **Note**
>
> 物件群組名稱不能包含冒號 (:) 字元。

f. *##### V2IoT ThingPolicy*。允許 Greengrass 核心裝置與和通訊的AWS IoT原則名稱。AWS IoT AWS IoT Greengrass如果原AWS IoT則不存在，安裝程式會以此名稱建立寬鬆AWS IoT原則。您可以針對您的使用案例限制此原則的權限。如需詳細資訊，請參閱 AWS IoT Greengrass V2核心裝置的最低AWS IoT原則。

g. *##### V2 TokenExchangeRole*。允許 Greengrass 核心裝置取得臨時登入資料的 IAM 角色名稱。AWS如果角色不存在，安裝程式會建立該角色，並建立並附加名為的策略*GreengrassV2TokenExchangeRole*Access。如需詳細資訊，請參閱 授權核心設備與AWS服務。

h. *GreengrassCoreTokenExchangeRoleAlias*。IAM 角色的別名，可讓 Greengrass 核心裝置稍後取得臨時登入資料。如果角色別名不存在，安裝程式會建立它，並將其指向您指定的 IAM 角色。如需詳細資訊，請參閱 授權核心設備與AWS服務。

Linux or Unix

```
sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \
  -jar ./GreengrassInstaller/lib/Greengrass.jar \
  --aws-region region \
  --thing-name MyGreengrassCore \
  --thing-group-name MyGreengrassCoreGroup \
  --thing-policy-name GreengrassV2IoTThingPolicy \
  --tes-role-name GreengrassV2TokenExchangeRole \
  --tes-role-alias-name GreengrassCoreTokenExchangeRoleAlias \
  --component-default-user ggc_user:ggc_group \
  --provision true \
  --setup-system-service true \
```

```
  --deploy-dev-tools true
```

Windows Command Prompt (CMD)

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" ^
  -jar ./GreengrassInstaller/lib/Greengrass.jar ^
  --aws-region region ^
  --thing-name MyGreengrassCore ^
  --thing-group-name MyGreengrassCoreGroup ^
  --thing-policy-name GreengrassV2IoTThingPolicy ^
  --tes-role-name GreengrassV2TokenExchangeRole ^
  --tes-role-alias-name GreengrassCoreTokenExchangeRoleAlias ^
  --component-default-user ggc_user ^
  --provision true ^
  --setup-system-service true ^
  --deploy-dev-tools true
```

PowerShell

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" `
  -jar ./GreengrassInstaller/lib/Greengrass.jar `
  --aws-region region `
  --thing-name MyGreengrassCore `
  --thing-group-name MyGreengrassCoreGroup `
  --thing-policy-name GreengrassV2IoTThingPolicy `
  --tes-role-name GreengrassV2TokenExchangeRole `
  --tes-role-alias-name GreengrassCoreTokenExchangeRoleAlias `
  --component-default-user ggc_user `
  --provision true `
  --setup-system-service true `
  --deploy-dev-tools true
```

> **Note**
>
> 如果您在記憶體有限的裝置AWS IoT Greengrass上執行，您可以控制 AWS IoT Greengrass Core 軟體使用的記憶體數量。若要控制記憶體分配，您可以在核心元件的`jvmOptions`組態參數中設定 JVM 堆積大小選項。如需詳細資訊，請參閱 使用 JVM 選項控制內存分配。

當您執行此命令時，您應該會看到下列訊息，指出安裝程式已成功。

```
Successfully configured Nucleus with provisioned resource details!
Configured Nucleus to deploy aws.greengrass.Cli component
Successfully set up Nucleus as a system service
```

> **Note**
>
> 如果您的 Linux 裝置沒有 systemd，安裝程式就不會將軟體設定為系統服務，而且您也不會看到將核心設定為系統服務的成功訊息。

(選擇性) 執行 Greengrass 軟體 (Linux)

如果您將軟體安裝為系統服務，安裝程式會為您執行軟體。否則，您必須運行該軟件。若要查看安裝程式是否將軟體設定為系統服務，請在安裝程式輸出中尋找下列行。

```
Successfully set up Nucleus as a system service
```

如果您沒有看到此訊息，請執行下列動作以執行軟體：

1. 運行以下命令以運行該軟件。

   ```
   sudo /greengrass/v2/alts/current/distro/bin/loader
   ```

   如果成功啟動，軟件將打印以下消息。

   ```
   Launched Nucleus successfully.
   ```

2. 您必須保持目前的命令殼層處於開啟狀態，以保持AWS IoT Greengrass核心軟體執行。如果您使用 SSH 連線至核心裝置，請在開發電腦上執行下列命令，以開啟第二個 SSH 工作階段，您可以使用此工作階段在核心裝置上執行其他命令。*####################pi-ip-address#### IP #####*

   ```
   ssh username@pi-ip-address
   ```

如需有關如何與 Greengrass 系統服務互動的詳細資訊，請參閱。[將 Greengrass 核配置為系統服務]

驗證設備上的綠色 CLI 安裝

Greengrass CLI 最多可能需要一分鐘的時間來部署。執行下列命令以檢查部署狀態。替換*MyGreengrassCore*為核心設備的名稱。

```
aws greengrassv2 list-effective-deployments --core-device-thing-name MyGreengrassCore
```

指coreDeviceExecutionStatus示核心裝置的部署狀態。當狀態為時SUCCEEDED，請執行下列命令以確認 Greengrass CLI 已安裝並執行。以根資料夾的路徑取*/greengrass/v2*代。

Linux or Unix

```
/greengrass/v2/bin/greengrass-cli help
```

Windows Command Prompt (CMD)

```
C:\greengrass\v2\bin\greengrass-cli help
```

PowerShell

```
C:\greengrass\v2\bin\greengrass-cli help
```

該命令輸出 Greengrass CLI 的幫助信息。如果找greengrass-cli不到，則部署可能無法安裝 Greengrass CLI。如需詳細資訊，請參閱 [疑難排 AWS IoT Greengrass V2]。

您也可以執行下列命令，以手動方式將 AWS IoT Greengrass CLI 部署到您的裝置。

- 將*##*替換為您 AWS 區域 使用的區域。確保您使用的 AWS 區域 是您用來在設備 AWS CLI 上配置的相同。
- 用您的 ID 替換*## ID* AWS 帳戶 。
- 替換*MyGreengrassCore*為核心設備的名稱。

Linux, macOS, or Unix

```
aws greengrassv2 create-deployment \
```

```
  --target-arn "arn:aws:iot:region:account-id:thing/MyGreengrassCore" \
  --components '{
    "aws.greengrass.Cli": {
      "componentVersion": "2.12.3"
    }
  }'
```

Windows Command Prompt (CMD)

```
aws greengrassv2 create-deployment ^
  --target-arn "arn:aws:iot:region:account-id:thing/MyGreengrassCore" ^
  --components "{\"aws.greengrass.Cli\":{\"componentVersion\":\"2.12.3\"}}"
```

PowerShell

```
aws greengrassv2 create-deployment `
  --target-arn "arn:aws:iot:region:account-id:thing/MyGreengrassCore" `
  --components '{\"aws.greengrass.Cli\":{\"componentVersion\":\"2.12.3\"}}'
```

> **Tip**
>
> 您可以將*/greengrass/v2*/bin（Linux）或*C:\greengrass\v2*\bin（Windows）添加到PATH環境變量中，以便在沒有絕對路徑的greengrass-cli情況下運行。

AWS IoT GreengrassCore 軟體和本機開發工具會在您的裝置上執行。接下來，您可以在設備上開發 Hello WorldAWS IoT Greengrass 組件。

步驟 4：在設備上開發和測試組件

組件是在 AWS IoT Greengrass 核心設備上運行的軟件模塊。元件可讓您建立和管理複雜的應用程式，做為獨立建置區塊，您可以從一個 Greengrass 核心裝置重複使用到另一個核心裝置。每個組件都由配方和文物組成。

- 食譜

  每個組件都包含一個 recipe 文件，該文件定義了其元數據。方案也會指定元件的組態參數、元件相依性、生命週期和平台相容性。元件生命週期會定義安裝、執行和關閉元件的命令。如需詳細資訊，請參閱 AWS IoT Greengrass元件配方參考。

您可以定義 JSON 或 YAML 格式的配方。

- 文物

  組件可以有任意數量的加工品，這是組件二進製文件。成品可以包括指令碼、已編譯的程式碼、靜態資源，以及元件使用的任何其他檔案。元件也可以從元件相依性取用人工因素。

使用時 AWS IoT Greengrass，您可以使用 Greengrass CLI 在 Greengrass 核心裝置上本機開發和測試元件，而無需與雲端互動。 AWS 當您完成本機元件時，您可以使用元件方案和成品在 AWS 雲端的 AWS IoT Greengrass 服務中建立該元件，然後將其部署到所有 Greengrass 核心裝置。如需元件的詳細資訊，請參閱開發AWS IoT Greengrass元件。

在本節中，您將學習如何在核心裝置本機建立和執行基本 Hello World 元件。

若要在您的裝置上開發 Hello World 元件

1. 使用配方和成品的子資料夾，為您的元件建立資料夾。在 Greengrass 核心裝置上執行下列命令，以建立這些資料夾並變更為元件資料夾。將 *~/greengrassv2 #% ######%\ Greengrassv2 ############*料夾路徑。

   Linux or Unix

   ```
   mkdir -p ~/greengrassv2/{recipes,artifacts}
   cd ~/greengrassv2
   ```

   Windows Command Prompt (CMD)

   ```
   mkdir %USERPROFILE%\greengrassv2\\recipes, %USERPROFILE%\greengrassv2\\artifacts
   cd %USERPROFILE%\greengrassv2
   ```

   PowerShell

   ```
   mkdir ~/greengrassv2/recipes, ~/greengrassv2/artifacts
   cd ~/greengrassv2
   ```

2. 使用文字編輯器建立 recipe 檔案，以定義元件的中繼資料、參數、相依性、生命週期和平台功能。在 recipe 檔案名稱中包含元件版本，以便識別哪個方案反映哪個元件版本。您可以為您的方案選擇 YAML 或 JSON 格式。

   例如，在 Linux 系統上，您可以執行下列命令來使用 GNU nano 來建立檔案。

JSON

```
nano recipes/com.example.HelloWorld-1.0.0.json
```

YAML

```
nano recipes/com.example.HelloWorld-1.0.0.yaml
```

> **Note**
>
> AWS IoT Greengrass 使用語義版本的組件。語義版本遵循一個主要的。 未成年人。 修補程式編號系統。例如，版本`1.0.0`代表元件的第一個主要發行版本。如需詳細資訊，請參閱語意版本規格。

3. 將下列配方貼到檔案中。

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.HelloWorld",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "My first AWS IoT Greengrass component.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "Message": "world"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "run": "python3 -u {artifacts:path}/hello_world.py {configuration:/
Message}"
      }
    },
```

```
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "run": "py -3 -u {artifacts:path}/hello_world.py {configuration:/
Message}"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.HelloWorld
ComponentVersion: '1.0.0'
ComponentDescription: My first AWS IoT Greengrass component.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    Message: world
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      run: |
        python3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
  - Platform:
      os: windows
    Lifecycle:
      run: |
        py -3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
```

這個配方的ComponentConfiguration部分定義了一個參數Message，默認為world。本Manifests節定義資訊清單，這是一組平台的生命週期指示和成品。例如，您可以定義多個資訊清單，為各種平台指定不同的安裝指示。在資訊清單中，Lifecycle區段會指示 Greengrass 核心裝置以Message參數值作為引數執行 Hello World 指令碼。

4. 執行下列命令，為元件加工品建立資料夾。

Linux or Unix

```
mkdir -p artifacts/com.example.HelloWorld/1.0.0
```

Windows Command Prompt (CMD)

```
mkdir artifacts\com.example.HelloWorld\1.0.0
```

PowerShell

```
mkdir artifacts\com.example.HelloWorld\1.0.0
```

> **Important**
>
> 您必須針對人工因素資料夾路徑使用下列格式。包括您在方案中指定的元件名稱和版本。
>
> ```
> artifacts/componentName/componentVersion/
> ```

5. 使用文字編輯器為您的 Hello World 元件建立 Python 指令碼人工因素檔案。

   例如，在 Linux 系統上，您可以執行下列命令來使用 GNU nano 來建立檔案。

   ```
   nano artifacts/com.example.HelloWorld/1.0.0/hello_world.py
   ```

   將以下 Python 腳本複製並粘貼到文件中。

   ```
   import sys

   message = "Hello, %s!" % sys.argv[1]

   # Print the message to stdout, which Greengrass saves in a log file.
   print(message)
   ```

6. 使用本機 AWS IoT Greengrass CLI 管理您的核心裝置上的元件。

執行下列命令，將元件部署至 AWS IoT Greengrass 核心。使用您的 AWS IoT Greengrass V2 根資料夾取代*/greengrass/v2*或 *C:\greengrass\v2*，並將 *~/greengrassv2 #% ##### #%\ Greengrassv2* 取代為您的元件開發資料夾。

Linux or Unix

```
sudo /greengrass/v2/bin/greengrass-cli deployment create \
  --recipeDir ~/greengrassv2/recipes \
  --artifactDir ~/greengrassv2/artifacts \
  --merge "com.example.HelloWorld=1.0.0"
```

Windows Command Prompt (CMD)

```
C:\greengrass\v2\bin\greengrass-cli deployment create ^
  --recipeDir %USERPROFILE%\greengrassv2\recipes ^
  --artifactDir %USERPROFILE%\greengrassv2\artifacts ^
  --merge "com.example.HelloWorld=1.0.0"
```

PowerShell

```
C:\greengrass\v2\bin\greengrass-cli deployment create `
  --recipeDir ~/greengrassv2/recipes `
  --artifactDir ~/greengrassv2/artifacts `
  --merge "com.example.HelloWorld=1.0.0"
```

此命令會新增使用中配方的元件，以`recipes`及中的 Python 指令碼`artifacts`。此選`--merge`項會新增或更新您指定的元件和版本。

7. AWS IoT Greengrass 核心軟件將標準輸出從組件進程保存到文件夾中的日誌文件`logs`。執行下列命令以驗證 Hello World 元件是否執行並列印訊息。

   Linux or Unix

   ```
   sudo tail -f /greengrass/v2/logs/com.example.HelloWorld.log
   ```

   Windows Command Prompt (CMD)

   ```
   type C:\greengrass\v2\logs\com.example.HelloWorld.log
   ```

該type命令將文件的內容寫入終端。多次執行此命令以觀察檔案中的變更。

PowerShell

```
gc C:\greengrass\v2\logs\com.example.HelloWorld.log -Tail 10 -Wait
```

您應該會看到類似下列範例的訊息。

```
Hello, world!
```

> **Note**
>
> 如果檔案不存在，表示本機部署可能尚未完成。如果檔案在 15 秒內不存在，則部署可能會失敗。例如，如果您的配方無效，則可能會發生這種情況。執行下列命令以檢視 AWS IoT Greengrass 核心記錄檔。此檔案包含來自 Greengrass 核心裝置部署服務的記錄檔。
>
> Linux or Unix
>
> ```
> sudo tail -f /greengrass/v2/logs/greengrass.log
> ```
>
> Windows Command Prompt (CMD)
>
> ```
> type C:\greengrass\v2\logs\greengrass.log
> ```
>
> 該type命令將文件的內容寫入終端。多次執行此命令以觀察檔案中的變更。
>
> PowerShell
>
> ```
> gc C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
> ```

8. 修改本地組件以迭代和測試您的代碼。hello_world.py在文字編輯器中開啟，並在第 4 行新增下列程式碼，以編輯 AWS IoT Greengrass 核心記錄的訊息。

```
message += " Greetings from your first Greengrass component."
```

該hello_world.py腳本現在應該具有以下內容。

```
import sys

message = "Hello, %s!" % sys.argv[1]
message += " Greetings from your first Greengrass component."

# Print the message to stdout, which Greengrass saves in a log file.
print(message)
```

9. 執行下列命令，以您所做的變更更新元件。

   Linux or Unix

   ```
   sudo /greengrass/v2/bin/greengrass-cli deployment create \
     --recipeDir ~/greengrassv2/recipes \
     --artifactDir ~/greengrassv2/artifacts \
     --merge "com.example.HelloWorld=1.0.0"
   ```

   Windows Command Prompt (CMD)

   ```
   C:\greengrass\v2\bin\greengrass-cli deployment create ^
     --recipeDir %USERPROFILE%\greengrassv2\recipes ^
     --artifactDir %USERPROFILE%\greengrassv2\artifacts ^
     --merge "com.example.HelloWorld=1.0.0"
   ```

   PowerShell

   ```
   C:\greengrass\v2\bin\greengrass-cli deployment create `
     --recipeDir ~/greengrassv2/recipes `
     --artifactDir ~/greengrassv2/artifacts `
     --merge "com.example.HelloWorld=1.0.0"
   ```

   此指令會使用最新的 Hello World 人工因素更新`com.example.HelloWorld`元件。

10. 執行下列命令以重新啟動元件。當您重新啟動元件時，核心裝置會使用最新的變更。

    Linux or Unix

    ```
    sudo /greengrass/v2/bin/greengrass-cli component restart \
      --names "com.example.HelloWorld"
    ```

Windows Command Prompt (CMD)

```
C:\greengrass\v2\bin\greengrass-cli component restart ^
  --names "com.example.HelloWorld"
```

PowerShell

```
C:\greengrass\v2\bin\greengrass-cli component restart `
  --names "com.example.HelloWorld"
```

11. 再次檢查記錄檔，以確認 Hello World 元件是否會列印新訊息。

    Linux or Unix

    ```
    sudo tail -f /greengrass/v2/logs/com.example.HelloWorld.log
    ```

    Windows Command Prompt (CMD)

    ```
    type C:\greengrass\v2\logs\com.example.HelloWorld.log
    ```

    該type命令將文件的內容寫入終端。多次執行此命令以觀察檔案中的變更。

    PowerShell

    ```
    gc C:\greengrass\v2\logs\com.example.HelloWorld.log -Tail 10 -Wait
    ```

    您應該會看到類似下列範例的訊息。

    ```
    Hello, world! Greetings from your first Greengrass component.
    ```

12. 您可以更新元件的組態參數，以測試不同的組態。部署元件時，您可以指定組態更新，以定義如何在核心裝置上修改元件的組態。您可以指定要重設為預設值的組態值，以及要合併至核心裝置的新組態值。如需詳細資訊，請參閱 更新零組件組態。

    請執行下列操作：

    a. 使用文字編輯器建立名為hello-world-config-update.json包含組態更新的檔案

       例如，在 Linux 系統上，您可以執行下列命令來使用 GNU nano 來建立檔案。

```
nano hello-world-config-update.json
```

b. 將下列 JSON 物件複製並貼到檔案中。此 JSON 對象定義了一個配置更新，該更新`friend`將值合併到`Message`參數以更新其值。此組態更新不會指定任何要重設的值。您不需要重設`Message`參數，因為合併更新會取代現有的值。

```
{
  "com.example.HelloWorld": {
    "MERGE": {
      "Message": "friend"
    }
  }
}
```

c. 執行下列命令，將組態更新部署至 Hello World 元件。

Linux or Unix

```
sudo /greengrass/v2/bin/greengrass-cli deployment create \
  --merge "com.example.HelloWorld=1.0.0" \
  --update-config hello-world-config-update.json
```

Windows Command Prompt (CMD)

```
C:\greengrass\v2\bin\greengrass-cli deployment create ^
  --merge "com.example.HelloWorld=1.0.0" ^
  --update-config hello-world-config-update.json
```

PowerShell

```
C:\greengrass\v2\bin\greengrass-cli deployment create `
  --merge "com.example.HelloWorld=1.0.0" `
  --update-config hello-world-config-update.json
```

d. 再次檢查記錄檔以確認 Hello World 元件是否輸出新訊息。

Linux or Unix

```
sudo tail -f /greengrass/v2/logs/com.example.HelloWorld.log
```

Windows Command Prompt (CMD)

```
type C:\greengrass\v2\logs\com.example.HelloWorld.log
```

該type命令將文件的內容寫入終端。多次執行此命令以觀察檔案中的變更。

PowerShell

```
gc C:\greengrass\v2\logs\com.example.HelloWorld.log -Tail 10 -Wait
```

您應該會看到類似下列範例的訊息。

```
Hello, friend! Greetings from your first Greengrass component.
```

13. 完成測試元件後，請將其從核心裝置中移除。執行下列命令。

Linux or Unix

```
sudo /greengrass/v2/bin/greengrass-cli deployment create --
remove="com.example.HelloWorld"
```

Windows Command Prompt (CMD)

```
C:\greengrass\v2\bin\greengrass-cli deployment create --
remove="com.example.HelloWorld"
```

PowerShell

```
C:\greengrass\v2\bin\greengrass-cli deployment create --
remove="com.example.HelloWorld"
```

> **Important**
>
> 您必須執行此步驟，才能在將元件上傳至核心裝置之後將元件部署回核心裝置 AWS IoT Greengrass。否則，部署會失敗並顯示版本相容性錯誤，因為本機部署會指定不同版本的元件。

執行下列命令，並確認com.example.HelloWorld元件未出現在裝置上的元件清單中。

Linux or Unix

```
sudo /greengrass/v2/bin/greengrass-cli component list
```

Windows Command Prompt (CMD)

```
C:\greengrass\v2\bin\greengrass-cli component list
```

PowerShell

```
C:\greengrass\v2\bin\greengrass-cli component list
```

您的 Hello World 元件已完成，您現在可以將其上傳到 AWS IoT Greengrass 雲端服務。然後，您可以將該組件部署到 Greengrass 核心設備。

步驟 5：在AWS IoT Greengrass服務中建立元件

當您在核心裝置上完成開發元件時，您可以將AWS IoT Greengrass它上傳到AWS 雲端. 您也可以直接在AWS IoT Greengrass主控台中建立元件。 AWS IoT Greengrass提供裝載元件的元件管理服務，以便您可以將它們部署到個別裝置或裝置叢集。若要將元件上傳至AWS IoT Greengrass服務，請完成下列步驟：

- 將元件成品上傳到 S3 儲存貯體。
- 將每個成品的亞馬遜簡單存儲服務（Amazon S3）URI 添加到組件方案中。
- 從元件方案AWS IoT Greengrass中建立元件。

在本節中，您會在 Greengrass 核心裝置上完成這些步驟，以便將 Hello World 元件上傳至服務。AWS IoT Greengrass

在AWS IoT Greengrass（控制台）中創建組件

1. 在您的AWS帳戶中使用 S3 儲存貯體來託管AWS IoT Greengrass元件成品。將元件部署到核心裝置時，裝置會從值區下載元件的成品。

您可以使用現有的 S3 儲存貯體，也可以建立新儲存貯體。

a. 在 Amazon S3 主控台的「儲存貯體」下，選擇「建立儲存貯體」。

b. 針對「值區名稱」，輸入唯一的值區名稱。例如，您可以使用 **greengrass-component-artifacts-*region*-*123456789012***. 將 *123456789012* 取代為您在本教學課程中使用的AWS帳戶 ID 和*#AWS ###*。

c. 對於「AWS區域」，請選取您在此自學課程中使用的「AWS區域」。

d. 選擇建立儲存貯體。

e. 在「值區」下，選擇您建立的值區，然後將指`hello_world.py`令碼上傳至值區中的`artifacts/com.example.HelloWorld/1.0.0`資料夾。如需將物件上傳到 S3 儲存貯體的相關資訊，請參閱 Amazon 簡單儲存服務使用者指南中的上傳物件。

f. 複製 S3 儲存貯體中`hello_world.py`物件的 S3 URI。此 URI 看起來應該類似於下列範例。將*########## S3 ####*的名稱。

```
s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.HelloWorld/1.0.0/hello_world.py
```

2. 允許核心裝置存取 S3 儲存貯體中的元件成品。

每個核心裝置都有一個核心裝置 IAM 角色，可讓其與雲端互動AWS IoT並將記錄檔傳送到AWS雲端。依預設，此裝置角色不允許存取 S3 儲存貯體，因此您必須建立並附加政策，以允許核心裝置從 S3 儲存貯體擷取元件成品。

如果您的裝置角色已允許存取 S3 儲存貯體，則可以略過此步驟。否則，請建立允許存取的 IAM 政策，並將其附加到角色，如下所示：

a. 在 IAM 主控台導覽功能表中，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。使用 S3 儲存貯體的名稱取代 *DOC/EXAMPLE* 儲存貯體，該儲存貯體包含要下載的核心裝置的元件成品。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*"
```

```
    }
  ]
}
```

c. 選擇下一步。

d. 在「策略詳細資料」區段中，輸入做為「名稱」**MyGreengrassV2ComponentArtifactPolicy**。

e. 選擇建立政策。

f. 在 IAM 主控台導覽功能表中，選擇 [角色]，然後選擇核心裝置的角色名稱。您在安裝 AWS IoT Greengrass Core 軟體時指定了此角色名稱。如果您未指定名稱，預設值為`GreengrassV2TokenExchangeRole`。

g. 在「權限」下，選擇「新增權限」，然後選擇「附加策略

h. 在 [新增權限] 頁面上，選取您建立之`MyGreengrassV2ComponentArtifactPolicy`原則旁邊的核取方塊，然後選擇 [新增權限]。

3. 使用元件方案在AWS IoT Greengrass主控台中建立元件。

a. 在AWS IoT Greengrass主控台瀏覽功能表中，選擇 [元件]，然後選擇 [建立元件]。

b. 在 [元件資訊] 下，選擇 [將方案輸入為 JSON]。預留位置方案看起來應類似下列範例。

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.HelloWorld",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "My first AWS IoT Greengrass component.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "Message": "world"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "run": "python3 -u {artifacts:path}/hello_world.py \"{configuration:/
Message}\""
      },
      "Artifacts": [
```

```
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.HelloWorld/1.0.0/hello_world.py"
        }
      ]
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "run": "py -3 -u {artifacts:path}/hello_world.py \"{configuration:/
Message}\""
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.HelloWorld/1.0.0/hello_world.py"
        }
      ]
    }
  ]
}
```

c. 將每個`Artifacts`區段中的預留位置 URI 取代為`hello_world.py`物件的 S3 URI。

d. 選擇 [建立元件]。

e. 在 com 上。例如。 HelloWorld「元件」頁面上，確認元件的「狀態」 是否為可建置。

在 () 中建立您的AWS IoT Greengrass元AWS CLI件

若要上傳您的 Hello World 元件

1. 使用 S3 儲存貯體AWS 帳戶來託管AWS IoT Greengrass元件成品。將元件部署到核心裝置時，裝置會從值區下載元件的成品。

   您可以使用現有的 S3 儲存貯體，或執行下列命令來建立儲存貯體。此指令會使用您的 AWS 帳戶 ID 建立值區，並形AWS 區域成唯一的值區名稱。將 *123456789012* 取代為您在本教學課程中使用的AWS 帳戶識別碼和*#AWS ###*。

   ```
   aws s3 mb s3://greengrass-component-artifacts-123456789012-region
   ```

如果要求成功，命令會輸出下列資訊。

```
make_bucket: greengrass-component-artifacts-123456789012-region
```

2. 允許核心裝置存取 S3 儲存貯體中的元件成品。

   每個核心裝置都有一個核心裝置 IAM 角色AWS IoT，可讓它與AWS 雲端. 依預設，此裝置角色不允許存取 S3 儲存貯體，因此您必須建立並附加政策，以允許核心裝置從 S3 儲存貯體擷取元件成品。

   如果核心裝置的角色已允許存取 S3 儲存貯體，您可以略過此步驟。否則，請建立允許存取的 IAM 政策，並將其附加到角色，如下所示：

   a. 創建一個名為的文件，component-artifact-policy.json並將以下 JSON 複製到該文件中。此政策允許存取 S3 儲存貯體中的所有檔案。將########## S3 ####的名稱。

      ```
      {
        "Version": "2012-10-17",
        "Statement": [
          {
            "Effect": "Allow",
            "Action": [
              "s3:GetObject"
            ],
            "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*"
          }
        ]
      }
      ```

   b. 執行下列命令，從中的策略文件建立策略component-artifact-policy.json。

      Linux or Unix

      ```
      aws iam create-policy \\
        --policy-name MyGreengrassV2ComponentArtifactPolicy \\
        --policy-document file://component-artifact-policy.json
      ```

      Windows Command Prompt (CMD)

      ```
      aws iam create-policy ^
        --policy-name MyGreengrassV2ComponentArtifactPolicy ^
      ```

```
  --policy-document file://component-artifact-policy.json
```

PowerShell

```
aws iam create-policy `
  --policy-name MyGreengrassV2ComponentArtifactPolicy `
  --policy-document file://component-artifact-policy.json
```

從輸出中的政策中繼資料複製政策 Amazon 資源名稱 (ARN)。您可以使用此 ARN 將此原則附加至下一個步驟中的核心裝置角色。

c. 執行下列命令，將原則附加至核心裝置角色。TokenExchangeRole以核心裝置的角色名稱取代 *Greenrassv2*。您在安裝 AWS IoT Greengrass Core 軟體時指定了此角色名稱。將原則 ARN 取代為上一個步驟的 ARN。

Linux or Unix

```
aws iam attach-role-policy \\
  --role-name GreengrassV2TokenExchangeRole \\
  --policy-arn
 arn:aws:iam::123456789012:policy/MyGreengrassV2ComponentArtifactPolicy
```

Windows Command Prompt (CMD)

```
aws iam attach-role-policy ^
  --role-name GreengrassV2TokenExchangeRole ^
  --policy-arn
 arn:aws:iam::123456789012:policy/MyGreengrassV2ComponentArtifactPolicy
```

PowerShell

```
aws iam attach-role-policy `
  --role-name GreengrassV2TokenExchangeRole `
  --policy-arn
 arn:aws:iam::123456789012:policy/MyGreengrassV2ComponentArtifactPolicy
```

如果命令沒有輸出，則成功。核心裝置現在可以存取您上傳到此 S3 儲存貯體的成品。

3. 上傳你好世界 Python 腳本工件到 S3 存儲桶。

執行下列命令，將指令碼上傳至AWS IoT Greengrass核心中指令碼所在值區中的相同路徑。將*########## S3 ####*的名稱。

Linux or Unix

```
aws s3 cp \
  artifacts/com.example.HelloWorld/1.0.0/hello_world.py \
  s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.HelloWorld/1.0.0/hello_world.py
```

Windows Command Prompt (CMD)

```
aws s3 cp ^
  artifacts/com.example.HelloWorld/1.0.0/hello_world.py ^
  s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.HelloWorld/1.0.0/hello_world.py
```

PowerShell

```
aws s3 cp `
  artifacts/com.example.HelloWorld/1.0.0/hello_world.py `
  s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.HelloWorld/1.0.0/hello_world.py
```

`upload:`如果要求成功，命令會輸出以開頭的行。

4. 將成品的 Amazon S3 URI 新增至元件方案。

   Amazon S3 URI 由儲存貯體名稱和儲存貯體中成品物件的路徑組成。指令碼人工因素的 Amazon S3 URI 是您在上一個步驟中將成品上傳到的 URI。此 URI 看起來應該類似於下列範例。將*###### ###### S3 ####*的名稱。

   ```
   s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.HelloWorld/1.0.0/hello_world.py
   ```

   若要將成品新增至方案，請新增包`Artifacts`含具有 Amazon S3 URI 之結構的清單。

   JSON

   ```
   "Artifacts": [
     {
       "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.HelloWorld/1.0.0/
   hello_world.py"
   ```

```
  }
]
```

在文字編輯器中開啟 recipe 檔案。

例如，在 Linux 系統上，您可以執行下列命令來使用 GNU nano 來建立檔案。

```
nano recipes/com.example.HelloWorld-1.0.0.json
```

將神器添加到配方中。您的 recipe 檔案看起來應該類似於下列範例。

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.HelloWorld",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "My first AWS IoT Greengrass component.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "Message": "world"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "run": "python3 -u {artifacts:path}/hello_world.py \"{configuration:/
Message}\""
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.HelloWorld/1.0.0/hello_world.py"
        }
      ]
    },
    {
      "Platform": {
        "os": "windows"
      },
```

```
      "Lifecycle": {
        "run": "py -3 -u {artifacts:path}/hello_world.py \"{configuration:/
Message}\""
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.HelloWorld/1.0.0/hello_world.py"
        }
      ]
    }
  ]
}
```

YAML

```
Artifacts:
  - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.HelloWorld/1.0.0/
hello_world.py
```

在文字編輯器中開啟 recipe 檔案。

例如，在 Linux 系統上，您可以執行下列命令來使用 GNU nano 來建立檔案。

```
nano recipes/com.example.HelloWorld-1.0.0.yaml
```

將神器添加到配方中。您的 recipe 檔案看起來應該類似於下列範例。

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.HelloWorld
ComponentVersion: '1.0.0'
ComponentDescription: My first AWS IoT Greengrass component.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    Message: world
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      run: |
```

```
        python3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.HelloWorld/1.0.0/
hello_world.py
  - Platform:
      os: windows
    Lifecycle:
      run: |
        py -3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.HelloWorld/1.0.0/
hello_world.py
```

5. AWS IoT Greengrass從方案中建立元件資源。執行下列命令，從 recipe (您提供為二進位檔案) 建立元件。

   JSON

   ```
   aws greengrassv2 create-component-version --inline-recipe fileb://recipes/
   com.example.HelloWorld-1.0.0.json
   ```

   YAML

   ```
   aws greengrassv2 create-component-version --inline-recipe fileb://recipes/
   com.example.HelloWorld-1.0.0.yaml
   ```

   如果要求成功，回應看起來會類似下列範例。

   ```
   {
     "arn":
    "arn:aws:greengrass:region:123456789012:components:com.example.HelloWorld:versions:1.0.0",
     "componentName": "com.example.HelloWorld",
     "componentVersion": "1.0.0",
     "creationTimestamp": "Mon Nov 30 09:04:05 UTC 2020",
     "status": {
       "componentState": "REQUESTED",
       "message": "NONE",
       "errors": {}
     }
   }
   ```

arn從輸出複製，以在下一個步驟中檢查組件的狀態。

> **Note**
>
> 您也可以在 [元件] 頁面的AWS IoT Greengrass主控台中看到 Hello World 元件。

6. 確認元件已建立並準備好部署。當您建立元件時，其狀態為REQUESTED。然後，AWS IoT Greengrass驗證元件是否可部署。您可以執行下列命令來查詢元件狀態，並確認元件是否可部署。arn使用上一個步驟中的 ARN 取代。

```
aws greengrassv2 describe-component --arn
 "arn:aws:greengrass:region:123456789012:components:com.example.HelloWorld:versions:1.0.0"
```

如果組件驗證，則響應指示組件狀態為DEPLOYABLE。

```
{
  "arn":
 "arn:aws:greengrass:region:123456789012:components:com.example.HelloWorld:versions:1.0.0",
  "componentName": "com.example.HelloWorld",
  "componentVersion": "1.0.0",
  "creationTimestamp": "2020-11-30T18:04:05.823Z",
  "publisher": "Amazon",
  "description": "My first Greengrass component.",
  "status": {
    "componentState": "DEPLOYABLE",
    "message": "NONE",
    "errors": {}
  },
  "platforms": [
    {
      "os": "linux",
      "architecture": "all"
    }
  ]
}
```

你好世界組件現在可在AWS IoT Greengrass. 您可以將其部署回 Greengrass 核心裝置或其他核心裝置。

步驟 6：部署元件

使用AWS IoT Greengrass，您可以將元件部署到個別裝置或裝置群組。部署元件時，請在每個目標AWS IoT Greengrass裝置上安裝並執行該元件的軟體。您可以指定要部署的元件，以及要為每個元件部署的組態更新。您也可以控制部署如何向部署作為目標的設備進行部署。如需詳細資訊，請參閱 將AWS IoT Greengrass元件部署到裝置。

在本節中，您將您的 Hello World 元件部署回 Greengrass 核心裝置。

部署您的元件 (主控台)

1. 在AWS IoT Greengrass主控台瀏覽功能表中，選擇 [元件]。
2. 在 [元件] 頁面的 [我的元件] 索引標籤上，選擇com.example.HelloWorld。
3. 在 com.example.HelloWorld頁面中，選擇部署。
4. 從新增至部署中，選擇建立新部署，然後選擇下一步。
5. 在指定目標頁面上，執行下列作業：

   a. 在 Name (名稱) 方塊中，輸入 **`Deployment for MyGreengrassCore`**。

   b. 針對部署目標，選擇核心裝置，以及核心裝置的項AWS IoT目名稱。此自學課程中的預設值為*`MyGreengrassCore`*。

   c. 選擇下一步。
6. 在 [選取元件] 頁面的 [我的元件] 下，確認已選取com.example.HelloWorld元件，然後選擇 [下一步]。
7. 在 [設定元件] 頁面上 com.example.HelloWorld，選擇並執行下列動作：

   a. 選擇 設定元件。

   b. 在組態更新下的要合併的組態中，輸入下列組態。

   ```
   {
     "Message": "universe"
   }
   ```

   此組態更新會將此部署中裝置`universe`的 Hello World `Message` 參數設定為。

   c. 選擇確認。

   d. 選擇下一步。
8. 在設定進階設定頁面上，保留預設組態設定，然後選擇 下一步。

9. 在 Review (檢閱) 頁面，選擇 Deploy (部署)。
10. 確認部署已成功完成。可能需要幾分鐘才能完成部署。檢查 Hello World 日誌以驗證更改。在您的 Greengrass 核心設備上運行以下命令。

    Linux or Unix

    ```
    sudo tail -f /greengrass/v2/logs/com.example.HelloWorld.log
    ```

    Windows Command Prompt (CMD)

    ```
    type C:\greengrass\v2\\logs\\com.example.HelloWorld.log
    ```

    PowerShell

    ```
    gc C:\greengrass\v2\\logs\\com.example.HelloWorld.log -Tail 10 -Wait
    ```

    您應該會看到類似下列範例的訊息。

    ```
    Hello, universe! Greetings from your first Greengrass component.
    ```

    > **Note**
    >
    > 如果記錄訊息未變更，表示部署失敗或無法到達核心裝置。如果您的核心裝置未連線到網際網路，或者沒有從 S3 儲存貯體擷取成品的權限，就會發生這種情況。在核心裝置上執行下列命令，以檢視 AWS IoT Greengrass Core 軟體記錄檔。此檔案包含來自 Greengrass 核心裝置部署服務的記錄檔。
    >
    > Linux or Unix
    >
    > ```
    > sudo tail -f /greengrass/v2/logs/greengrass.log
    > ```
    >
    > Windows Command Prompt (CMD)
    >
    > ```
    > type C:\greengrass\v2\\logs\\greengrass.log
    > ```
    >
    > 該type命令將文件的內容寫入終端。多次執行此命令以觀察檔案中的變更。

PowerShell

```
gc C:\greengrass\v2\\logs\\greengrass.log -Tail 10 -Wait
```

如需詳細資訊，請參閱 疑難排 AWS IoT Greengrass V2。

部署您的元件 (AWS CLI)

若要部署您的 Hello World 元件

1. 在您的開發電腦上，建立名為的檔案，hello-world-deployment.json並將下列 JSON 複製到檔案中。此檔案定義要部署的元件和組態。

```
{
  "components": {
    "com.example.HelloWorld": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "merge": "{\"Message\":\"universe\"}"
      }
    }
  }
}
```

   此組態檔案會指定部署您在先前程序中開發並發佈之 Hello World 元件1.0.0的版本。configurationUpdate指定要合併 JSON 編碼字串中的組件組態。此組態更新會將此部署中裝置universe的 Hello World Message 參數設定為。

2. 執行下列命令，將元件部署到 Greengrass 核心裝置。您可以部署至個別裝置或物群組 (裝置群組) 的物件。*MyGreengrassCore*替換為核心設備的AWS IoT物件的名稱。

   Linux or Unix

```
aws greengrassv2 create-deployment \
  --target-arn "arn:aws:iot:region:account-id:thing/MyGreengrassCore" \
  --cli-input-json file://hello-world-deployment.json
```

Windows Command Prompt (CMD)

```
aws greengrassv2 create-deployment ^
  --target-arn "arn:aws:iot:region:account-id:thing/MyGreengrassCore" ^
  --cli-input-json file://hello-world-deployment.json
```

PowerShell

```
aws greengrassv2 create-deployment `
  --target-arn "arn:aws:iot:region:account-id:thing/MyGreengrassCore" `
  --cli-input-json file://hello-world-deployment.json
```

指令會輸出類似下列範例的回應。

```
{
  "deploymentId": "deb69c37-314a-4369-a6a1-3dff9fce73a9",
  "iotJobId": "b5d92151-6348-4941-8603-bdbfb3e02b75",
  "iotJobArn": "arn:aws:iot:region:account-id:job/b5d92151-6348-4941-8603-
bdbfb3e02b75"
}
```

3. 確認部署已成功完成。可能需要幾分鐘才能完成部署。檢查 Hello World 日誌以驗證更改。在您的 Greengrass 核心設備上運行以下命令。

Linux or Unix

```
sudo tail -f /greengrass/v2/logs/com.example.HelloWorld.log
```

Windows Command Prompt (CMD)

```
type C:\greengrass\v2\\logs\\com.example.HelloWorld.log
```

PowerShell

```
gc C:\greengrass\v2\\logs\\com.example.HelloWorld.log -Tail 10 -Wait
```

您應該會看到類似下列範例的訊息。

```
Hello, universe! Greetings from your first Greengrass component.
```

> **Note**
>
> 如果記錄訊息未變更，表示部署失敗或無法到達核心裝置。如果您的核心裝置未連線到網際網路，或者沒有從 S3 儲存貯體擷取成品的權限，就會發生這種情況。在核心裝置上執行下列命令，以檢視 AWS IoT Greengrass Core 軟體記錄檔。此檔案包含來自 Greengrass 核心裝置部署服務的記錄檔。
>
> Linux or Unix
>
> ```
> sudo tail -f /greengrass/v2/logs/greengrass.log
> ```
>
> Windows Command Prompt (CMD)
>
> ```
> type C:\greengrass\v2\\logs\\greengrass.log
> ```
>
> 該type命令將文件的內容寫入終端。多次執行此命令以觀察檔案中的變更。
>
> PowerShell
>
> ```
> gc C:\greengrass\v2\\logs\\greengrass.log -Tail 10 -Wait
> ```
>
> 如需更多詳細資訊，請參閱 疑難排 AWS IoT Greengrass V2。

後續步驟

您已完成此教學課程。AWS IoT Greengrass核心軟件和您的 Hello World 組件在您的設備上運行。此外，您的 Hello World 元件也可以在AWS IoT Greengrass雲端服務中使用，以部署到其他裝置。如需本指導主題的詳細資訊，請參閱下列各項：

- 建立 AWS IoT Greengrass 元件
- 發佈元件以部署至核心裝置
- 將AWS IoT Greengrass元件部署到裝置

設定AWS IoT Greengrass核心裝置

完成本節中的工作以安裝、設定和執行 AWS IoT Greengrass Core 軟體。

> **Note**
>
> 本節說明 AWS IoT Greengrass Core 軟體的進階安裝與組態。如果您是第一次使用的使用者 AWS IoT Greengrass V2，建議您先完成入門教學課程，以設定核心裝置並探索的功能AWS IoT Greengrass。

支援平台和需求

在開始之前，請確定您符合下列需求，才能安裝並執行 AWS IoT Greengrass Core 軟體。

> **Tip**
>
> 您可以在合AWS作夥伴裝置目錄AWS IoT Greengrass V2中搜尋符合條件的裝置。

主題

支援平台

AWS IoT Greengrass正式支持運行以下平台的設備。具有未包含在此清單中的平台的裝置可能會運作，但只能在這些指定的平台上AWS IoT Greengrass進行測試。

Linux

架構：

- Armv7l

- Armv8 (AArch64)
- x86_64

Windows

架構：

- x86_64

版本：

- Windows 10
- Windows 11
- Windows Server 2019
- Windows Server 2022

> **Note**
>
> Windows 裝置目前不支援某些AWS IoT Greengrass功能。如需詳細資訊，請參閱 通過操作系統的 Greengrass 功能兼容性 及 視窗裝置的功能注意事項。

Linux 平台也可以AWS IoT Greengrass V2在碼頭容器中運行。如需詳細資訊，請參閱 在 Docker 容器中執行 AWS IoT Greengrass 核心軟體。

若要建置以 Linux 為基礎的自訂作業系統，您可以在專案AWS IoT Greengrass V2中使用的 BitBake `meta-aws`方法。該`meta-aws`項目提供了一些配方，您可以使用它來構建使用 Yocto 項目構建框架的嵌入式 Linux 系統中構建AWS邊緣軟件功能。OpenEmbeddedYocto 項目是一個開源協作項目，可幫助您為嵌入式應用程序構建基於 Linux 的自定義系統，無論硬件架構如何。在您的AWS IoT Greengrass V2裝置上安裝、設定及自動執行 AWS IoT Greengrass Core 軟體的 BitBake 秘訣。

裝置要求

裝置必須符合下列需求，才能安裝並執行AWS IoT Greengrass核心軟體 v2.x。

> **Note**
>
> 您可以使AWS IoT Device Tester用 in AWS IoT Greengrass 來驗證您的裝置是否可以執行 AWS IoT Greengrass Core 軟體並與AWS 雲端. 如需詳細資訊，請參閱 AWS IoT Device Tester對於 AWS IoT Greengrass V2 使用。

Linux

- 使用一個AWS 區域支持AWS IoT Greengrass V2. 如需支援區域的清單，請參閱《AWS 一般參考》中的 AWS IoT Greengrass V2 端點和配額。
- AWS IoT Greengrass核心軟體可用的最低 256 MB 磁碟空間。此需求不包括部署至核心裝置的元件。
- 至少配置給AWS IoT Greengrass核心軟體的 96 MB 記憶體。此需求不包括在核心裝置上執行的元件。如需詳細資訊，請參閱 使用 JVM 選項控制內存分配。
- Java 執行階段環境 (JRE) 版本 8 或更高版本。Java 必須在設備上的 PATH 環境變量上可用。若要使用 Java 開發自訂元件，您必須安裝 Java 開發套件 (JDK)。我們建議您使用 Amazon Corretto 或 OpenJDK 長期支持版本。需要版本 8 或更高版本。
- GNU C 函式庫 (格列) 2.25 版或更高版本。
- 您必須以 root 使用者身分執行 AWS IoT Greengrass Core 軟體。例如`sudo`，使用。
- 執行 AWS IoT Greengrass Core 軟體 (例如`root`) 的 root 使用者必須擁有`sudo`與任何使用者和任何群組一起執行的權限。`/etc/sudoers`檔案必須授與此使用者權限，才能以其他群組的`sudo`身分執行。中的使用者權限`/etc/sudoers`應如下列範例所示。

  ```
  root    ALL=(ALL:ALL) ALL
  ```

- 核心裝置必須能夠對一組端點和連接埠執行輸出要求。如需詳細資訊，請參閱 允許裝置流量透過 Proxy 或防火牆。
- 目`/tmp`錄必須以`exec`權限裝載。
- 下列所有命令介面命令：
  - `ps -ax -o pid,ppid`
  - `sudo`
  - `sh`
  - `kill`
  - `cp`

  - `chmod`
  - `rm`
  - `ln`
  - `echo`
  - `exit`
  - `id`
  - `uname`
  - `grep`
- 您的設備可能還需要以下可選的 shell 命令：
  - (選用) `systemctl`。此命令用於設置AWS IoT Greengrass核心軟件作為系統服務。
  - (選擇性) `useradd` `groupadd`、與`usermod`。這些指令可用來設定`ggc_user`系統使用者和`ggc_group`系統群組。
  - (選用) `mkfifo`。這個命令用於運行 Lambda 函數作為組件。
- 若要設定元件程序的系統資源限制，您的裝置必須執行 Linux 核心 2.6.24 版或更新版本。
- 若要執行 Lambda 函數，您的裝置必須符合其他需求。如需詳細資訊，請參閱 Lambda 函數要求。

Windows

- 使用一個AWS 區域支持AWS IoT Greengrass V2. 如需支援區域的清單，請參閱《AWS 一般參考》中的 AWS IoT Greengrass V2 端點和配額。
- AWS IoT Greengrass核心軟體可用的最低 256 MB 磁碟空間。此需求不包括部署至核心裝置的元件。
- 至少配置給AWS IoT Greengrass核心軟體的 160 MB 記憶體。此需求不包括在核心裝置上執行的元件。如需詳細資訊，請參閱 使用 JVM 選項控制內存分配。
- Java 執行階段環境 (JRE) 版本 8 或更高版本。Java 必須在設備上的 PATH 系統變數上可用。若要使用 Java 開發自訂元件，您必須安裝 Java 開發套件 (JDK)。我們建議您使用 Amazon Corretto 或 OpenJDK 長期支持版本。需要版本 8 或更高版本。

  > **Note**
  >
  > 若要使用 Greengrass 核心的 2.5.0 版本，您必須使用 64 位元版本的 Java 執行階段環境 (JRE)。Greengrass 核 2.5.1 版本支持 32 位和 64 位元的 JRE。

- 安裝 AWS IoT Greengrass Core 軟體的使用者必須是系統管理員。
- 您必須將 AWS IoT Greengrass Core 軟體安裝為系統服務。指`--setup-system-service true`定安裝軟體的時間。
- 每個執行元件處理序的使用者都必須存在於 LocalSystem 帳戶中，而且使用者的名稱和密碼必須位於 LocalSystem 帳戶的認證管理員執行個體中。當您依照指示安裝 AWS IoT Greengrass Core 軟體時，即可設定此使用者。
- 核心裝置必須能夠對一組端點和連接埠執行輸出要求。如需詳細資訊，請參閱 允許裝置流量透過 Proxy 或防火牆。

Lambda 函數要求

您的裝置必須符合下列需求才能執行 Lambda 函數：

- 以 Linux 為基礎的作業系統。
- 您的設備必須具有 `mkfifo` shell 命令。
- 您的裝置必須執行 Lambda 函數所需的程式設計語言程式庫。您必須在裝置上安裝所需的程式庫，並將其新增至PATH環境變數。Greengrass 支持所 Lambda 支持的 Python，Node.js 和 Java 運行時的版本。Greengrass 不會對已淘汰的 Lambda 執行階段版本套用任何其他限制。如需 Lambda 執行階段AWS IoT Greengrass支援的詳細資訊，請參閱執行AWS Lambda函數。
- 若要執行容器化 Lambda 函數，您的裝置必須符合下列需求：
  - Linux 核心版本 4.4 或更新版本。
  - 核心必須支援 cgroups v1，而且您必須啟用並掛載下列 cgroup：
    - 用AWS IoT Greengrass來設定容器化 Lambda 函數的記憶體限制的記憶體 cgroup。
    - 用於容器化 Lambda 函數的裝置 cgroup，以存取系統裝置或磁碟區。

    AWS IoT Greengrass核心軟體不支援 cgroup v2。

    若要符合此需求，請使用下列 Linux 核心參數來啟動裝置。

    ```
    cgroup_enable=memory cgroup_memory=1 systemd.unified_cgroup_hierarchy=0
    ```

    > **Tip**
    >
    > 在 Raspberry Pi 上，編輯`/boot/cmdline.txt`文件以設置設備的內核參數。

- 您必須在設備上啟用以下 Linux 內核配置：
  - 命名空間：
    - CONFIG_IPC_NS
    - CONFIG_UTS_NS
    - CONFIG_USER_NS
    - CONFIG_PID_NS
  - Cgroups：
    - CONFIG_CGROUP_DEVICE
    - CONFIG_CGROUPS
    - CONFIG_MEMCG
  - 其他：
    - CONFIG_POSIX_MQUEUE
    - CONFIG_OVERLAY_FS
    - CONFIG_HAVE_ARCH_SECCOMP_FILTER
    - CONFIG_SECCOMP_FILTER
    - CONFIG_KEYS
    - CONFIG_SECCOMP
    - CONFIG_SHMEM

> **Tip**
>
> 請查看您 Linux 發行版的文件，以瞭解如何驗證和設定 Linux 核心參數。您也可以使用 AWS IoT Greengrass to AWS IoT Device Tester 來驗證您的裝置是否符合這些需求。如需詳細資訊，請參閱 AWS IoT Device Tester對於 AWS IoT Greengrass V2 使用。

視窗裝置的功能注意事項

Windows 裝置目前不支援某些AWS IoT Greengrass功能。檢閱功能差異，以確認 Windows 裝置是否符合您的需求。如需詳細資訊，請參閱 通過操作系統的 Greengrass 功能兼容性。

設置一個 AWS 帳戶

如果您還沒有 AWS 帳戶，請完成以下步驟建立新帳戶。

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，並在電話鍵盤輸入驗證碼。

   註冊 AWS 帳戶時，會建立 AWS 帳戶根使用者。根使用者有權存取該帳戶中的所有 AWS 服務和資源。作為最佳安全實務，將管理存取權指派給管理使用者，並且僅使用根使用者來執行需要根使用者存取權的任務。

若要建立管理員使用者，請選擇下列其中一個選項。

選擇一種管理管理員的方式	到	By	您也可以
在 IAM Identity Center (建議)	使用短期憑證存取 AWS。 這與安全性最佳實務一致。有關最佳實務的資訊，請參閱 IAM 使用者指南中的 IAM 安全最佳實務。	請遵循 AWS IAM Identity Center 使用者指南的入門中的說明。	請參閱 AWS Command Line Interface 使用者指南中的設定 AWS CLI 以使用 AWS IAM Identity Center 設定程式設計存取。
在 IAM 中 (不建議使用)	使用長期憑證存取 AWS。	請遵循 IAM 使用者指南中建立您的第一個 IAM 管理員使用者和使用者群組的說明。	請參閱 IAM 使用者指南中的管理 IAM 使用者的存取金鑰，設定程式設計存取。

安裝 AWS IoT Greengrass 核心軟體

AWS IoT Greengrass延伸AWS至邊緣裝置，以便它們可以對其產生的資料採取行動，同時將其用 AWS 雲端於管理、分析和持久儲存。在邊緣裝置上安裝 AWS IoT Greengrass Core 軟體，以便AWS IoT Greengrass與之整合AWS 雲端。

> **Important**
>
> 在您下載並安裝 AWS IoT Greengrass Core 軟體之前，請檢查您的核心裝置是否符合安裝和執行 AWS IoT Greengrass Core 軟體 2.0 的需求。

AWS IoT Greengrass核心軟體包含一個安裝程式，可將您的裝置設定為 Greengrass 核心裝置。當您執行安裝程式時，您可以設定選項，例如根資料夾和AWS 區域要使用的。您可以選擇讓安裝程式為您建立所需AWS IoT的 IAM 資源。您也可以選擇部署本機開發工具，以設定用於自訂元件開發的裝置。

AWS IoT Greengrass核心軟體需要下列AWS IoT和 IAM 資源才能連線到AWS 雲端並進行操作：

- AWS IoT 物件。當您將裝置註冊為AWS IoT物件時，該裝置可以使用數位憑證來進行驗證AWS。此憑證可讓裝置與AWS IoT和通訊AWS IoT Greengrass。如需詳細資訊，請參閱 AWS IoT Greengrass 的裝置身分驗證和授權。
- (選擇性) 物AWS IoT件群組。您可以使用物件群組來管理 Greengrass 核心裝置的叢集。將軟體元件部署到裝置時，您可以選擇部署到個別裝置或裝置群組。您可以將裝置新增至物件群組，以將該物件群組的軟體元件部署至裝置。如需詳細資訊，請參閱 將AWS IoT Greengrass元件部署到裝置。
- IAM 角色 Greengrass 核心裝置會使用AWS IoT Core登入資料提供者授權對具有 IAM 角色之AWS服務的呼叫。此角色可讓您的裝置與 AWS IoT Amazon 簡單儲存服務 (Amazon S3) 互動、將 CloudWatch 日誌傳送至 Amazon 日誌，以及下載自訂元件成品。如需詳細資訊，請參閱 授權核心設備與AWS服務。
- AWS IoT角色別名。Greengrass 核心裝置會使用角色別名來識別要使用的 IAM 角色。角色別名可讓您變更 IAM 角色，但保持裝置設定不變。如需詳細資訊，請參閱AWS IoT Core開發人員指南中的授權直接呼叫AWS服務。

選擇下列其中一個選項，在AWS IoT Greengrass核心裝置上安裝 Core 軟體。

- 快速安裝

  選擇此選項，以盡可能少的步驟設定 Greengrass 核心裝置。安裝程式會為您建立所需AWS IoT的 IAM 資源。此選項需要您提供AWS認證給安裝程式，以便在AWS 帳戶.

您無法使用此選項安裝在防火牆或網路 Proxy 後方。如果您的裝置位於防火牆或網路 Proxy 後方，請考慮手動安裝。

如需詳細資訊，請參閱 使用自動資源佈建安裝AWS IoT Greengrass核心軟體。

- 手動安裝

  選擇此選項可手動建立必要的AWS資源，或安裝在防火牆或網路 Proxy 後方。通過使用手動安裝，您不需要授予安裝程序權限來在您的中創建資源AWS 帳戶，因為您創建了必要的AWS IoT和 IAM 資源。您也可以將裝置設定為透過連接埠 443 或透過網路 Proxy 進行連線。您也可以將 AWS IoT Greengrass Core 軟體設定為使用儲存在硬體安全性模組 (HSM)、信任平台模組 (TPM) 或其他加密元素中的私密金鑰和憑證。

  如需詳細資訊，請參閱 透過手動佈建資源安裝 AWS IoT Greengrass 核心軟體。

- 以AWS IoT叢集佈建進行安裝

  選擇此選項可從AWS IoT叢集佈建範本建立所需的AWS資源。您可以選擇此選項在車隊中建立類似的裝置，或者如果您製造客戶稍後啟動的裝置，例如車輛或智慧家居裝置。裝置使用宣告憑證來驗證和佈建AWS資源，包括 X.509 用戶端憑證，裝置用來連線以進AWS 雲端行正常作業。您可以在製造期間將宣告憑證內嵌或快閃儲存到裝置的硬體中，並且可以使用相同的宣告憑證和金鑰來佈建多個裝置。您也可以將裝置設定為透過連接埠 443 或透過網路 Proxy 進行連線。

  如需詳細資訊，請參閱 透過 AWS IoT 叢集佈建安裝 AWS IoT Greengrass 核心軟體。

- 以自訂佈建進行安裝

  選擇此選項可開發佈建必要AWS資源的自訂 Java 應用程式。如果您建立自己的 X.509 用戶端憑證，或想要對佈建程序進行更多控制，則可以選擇此選項。 AWS IoT Greengrass提供可實作的介面，以便在自訂佈建應用程式和 AWS IoT Greengrass Core 軟體安裝程式之間交換資訊。

  如需詳細資訊，請參閱 使用自訂資源佈建安裝 AWS IoT Greengrass 核心軟體。

AWS IoT Greengrass 也提供執行 AWS IoT Greengrass 核心軟體的容器化環境。您可以使用碼頭文件在碼頭容器AWS IoT Greengrass中運行。

主題

- 使用自動資源佈建安裝AWS IoT Greengrass核心軟體
- 透過手動佈建資源安裝 AWS IoT Greengrass 核心軟體
- 透過 AWS IoT 叢集佈建安裝 AWS IoT Greengrass 核心軟體

- 使用自訂資源佈建安裝 AWS IoT Greengrass 核心軟體
- 安裝器引數

使用自動資源佈建安裝AWS IoT Greengrass核心軟體

AWS IoT Greengrass核心軟體包含一個安裝程式，可將您的裝置設定為 Greengrass 核心裝置。若要快速設定裝置，安裝程式可以佈建核心裝置操作所需的AWS IoT物件、物件群組、IAM AWS IoT 角色和角色別名。AWS IoT安裝程式也可以將本機開發工具部署到核心裝置，以便您可以使用該裝置來開發和測試自訂軟體元件。安裝程式需要AWS認證，才能佈建這些資源並建立部署。

如果您無法提供AWS認證給設備，則可以佈建核心設備操作所需的AWS資源。您也可以將開發工具部署到核心裝置，做為開發裝置使用。這可讓您在執行安裝程式時提供較少的裝置權限。如需詳細資訊，請參閱 透過手動佈建資源安裝 AWS IoT Greengrass 核心軟體。

> **Important**
>
> 在您下載 AWS IoT Greengrass Core 軟體之前，請檢查您的核心裝置是否符合安裝和執行 AWS IoT Greengrass Core 軟體 2.0 的需求。

主題

- 設定裝置環境
- 提供AWS認證給設備
- 下載AWS IoT Greengrass核心軟體
- 安裝 AWS IoT Greengrass 核心軟體

設定裝置環境

請依照本節中的步驟設定 Linux 或 Windows 裝置作為AWS IoT Greengrass核心裝置使用。

設定裝置

若要設定下列項目的 Linux 設備 AWS IoT Greengrass V2

1. 安裝 Java 執行階段，AWS IoT Greengrass核心軟體需要執行。我們建議您使用 Amazon Corretto 或 OpenJDK 長期支持版本。需要版本 8 或更高版本。以下命令向您展示如何在設備上安裝 OpenJDK。

- 針對 Debian 為基礎或 Ubuntu 為基礎的發行版本：

```
sudo apt install default-jdk
```

- 針對 Red Hat 為基礎的發行版本：

```
sudo yum install java-11-openjdk-devel
```

- 針對 Amazon Linux 2：

```
sudo amazon-linux-extras install java-openjdk11
```

- 對於 Amazon

```
sudo dnf install java-11-amazon-corretto -y
```

安裝完成時，請執行下列命令以確認 Java 是否在 Linux 裝置上執行。

```
java -version
```

此指令會列印在裝置上執行的 Java 版本。例如，在基於 Debian 的發行版上，輸出可能類似於以下示例。

```
openjdk version "11.0.9.1" 2020-11-04
OpenJDK Runtime Environment (build 11.0.9.1+1-post-Debian-1deb10u2)
OpenJDK 64-Bit Server VM (build 11.0.9.1+1-post-Debian-1deb10u2, mixed mode)
```

2. (選擇性) 建立在裝置上執行元件的預設系統使用者和群組。您也可以選擇讓 AWS IoT Greengrass Core 軟體安裝程式在安裝期間使用安裝程式引數建立此使用者和群組。`--component-default-user`如需詳細資訊，請參閱 安裝器引數。

```
sudo useradd --system --create-home ggc_user
sudo groupadd --system ggc_group
```

3. 確認執行 AWS IoT Greengrass Core 軟體的使用者 (通常`root`) 具有`sudo`與任何使用者和任何群組一起執行的權限。

   a. 執行下列命令以開啟`/etc/sudoers`檔案。

```
sudo visudo
```

b. 確認使用者的權限看起來像下列範例。

```
root    ALL=(ALL:ALL) ALL
```

4. (選擇性) 若要執行容器化 Lambda 函數，您必須啟用 cgroup v1，並且必須啟用並掛接記憶體和裝置 cgroup。如果您不打算執行容器化 Lambda 函數，則可以略過此步驟。

   若要啟用這些 cgroups 選項，請使用下列 Linux 核心參數來啟動裝置。

   ```
   cgroup_enable=memory cgroup_memory=1 systemd.unified_cgroup_hierarchy=0
   ```

   如需有關檢視和設定裝置核心參數的資訊，請參閱作業系統和開機載入程式的說明文件。依照指示永久設定核心參數。

5. 依照中的需求清單所指示，在您的裝置上安裝所有其他必要的相依性裝置要求。

設定視窗裝置

> **Note**
>
> 此功能適用於 v2.5.0 及更高版 Greeng rass 核組件。

若要設定下列項目的視窗裝置 AWS IoT Greengrass V2

1. 安裝 Java 執行階段，AWS IoT Greengrass核心軟體需要執行。我們建議您使用 Amazon Corretto 或 OpenJDK 長期支持版本。需要版本 8 或更高版本。
2. 檢查系統變數 PATH 上是否可用 Java，如果沒有，則加入它。此 LocalSystem 帳戶會執行AWS IoT Greengrass核心軟體，因此您必須將 Java 新增至系統變數 PATH，而非使用者的 PATH 使用者變數。請執行下列操作：

   a. 按視窗鍵開啟開始功能表。

   b. 鍵入**environment variables**以從開始功能表搜尋系統選項。

   c. 在開始功能表搜尋結果中，選擇 [編輯系統環境變數] 以開啟 [系統屬性] 視窗。

   d. 選擇環境變數... 以開啟 [環境變數] 視窗。

e. 在 [系統變數] 下，選取 [路徑]，然後選擇 [編輯]。在 「編輯」環境變數視窗中，您可以在不同的行上檢視每個路徑。

f. 檢查 Java 安裝bin資料夾的路徑是否存在。路徑看起來可能類似下列範例。

```
C:\\Program Files\\Amazon Corretto\\jdk11.0.13_8\\bin
```

g. 如果路徑中缺少 Java 安裝的bin資料夾，請選擇 [新增] 加入，然後選擇 [確定]。

3. 以系統管理員身分開啟 Windows 命令提示字元 (cmd.exe)。

4. 在 Windows 裝置上的 LocalSystem 帳戶中建立預設使用者。以安全*##*取代密碼。

```
net user /add ggc_user password
```

> **Tip**
>
> 視您的 Windows 組態而定，使用者的密碼可能會設定為在 future 的某個日期到期。為確保您的 Greengrass 應用程式繼續運作，請追蹤密碼何時到期，並在密碼到期之前進行更新。您也可以將使用者的密碼設定為永不過期。
>
> - 若要檢查使用者及其密碼何時到期，請執行下列命令。
>
>   ```
>   net user ggc_user | findstr /C:expires
>   ```
>
> - 若要將使用者的密碼設定為永不過期，請執行下列命令。
>
>   ```
>   wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
>   ```
>
> - 如果您使用的是 Windows 10 或更新版本，而該wmic命令已被取代，請執行下列 PowerShell 命令。
>
>   ```
>   Get-CimInstance -Query "SELECT * from Win32_UserAccount WHERE name =
>    'ggc_user'" | Set-CimInstance -Property @{PasswordExpires="False"}
>   ```

5. 在設備上從 Microsoft 下載並安裝該PsExec實用程序。

6. 使用此 PsExec 公用程式將預設使用者的使用者名稱和密碼儲存在 LocalSystem 帳戶的認證管理員執行個體中。*################*

```
psexec -s cmd /c cmdkey /generic:ggc_user /user:ggc_user /pass:password
```

如果打PsExec License Agreement開，請選Accept擇同意許可證並運行命令。

> **Note**
>
> 在 Windows 裝置上， LocalSystem 帳戶會執行 Greengrass 核心，而且您必須使用公用 PsExec 程式將預設使用者資訊儲存在帳戶中。 LocalSystem 使用認證管理員應用程式會將此資訊儲存在目前登入使用者的 Windows 帳戶中，而非 LocalSystem 帳戶中。

提供AWS認證給設備

將您的AWS認證提供給您的裝置，以便安裝程式可以佈建所需的AWS資源。如需所需許可的詳細資訊，請參閱安裝程式佈建資源的最低 IAM 政策。

提供AWS認證給設備

- 將您的AWS登入資料提供給裝置，以便安裝程式可以為您的核心裝置佈建AWS IoT和 IAM 資源。為了提高安全性，我們建議您取得 IAM 角色的臨時登入資料，該角色僅允許佈建所需的最低許可。如需詳細資訊，請參閱 安裝程式佈建資源的最低 IAM 政策。

  > **Note**
  >
  > 安裝程式不會儲存或儲存您的認證。

  在您的裝置上，執行下列其中一項動作以擷取認證，並將其提供給 AWS IoT Greengrass Core 軟體安裝程式：

  - (建議) 使用來源的暫時認證 AWS IAM Identity Center

    a. 從 IAM 身分中心提供存取金鑰 ID、秘密存取金鑰和工作階段權杖。如需詳細資訊，請參閱 IAM 身分中心使用指南中的取得和重新整理臨時登入資料中的手動登入資料重新整

    b. 執行下列命令，將認證提供給 AWS IoT Greengrass Core 軟體。

    Linux or Unix

    ```
    export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
    export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
    export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o5OytwEXAMPLE=
    ```

Windows Command Prompt (CMD)

```
set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
set AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o5OytwEXAMPLE=
```

PowerShell

```
$env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
$env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
$env:AWS_SESSION_TOKEN="AQoDYXdzEJr1K...o5OytwEXAMPLE="
```

- 使用 IAM 角色的臨時安全登入資料：

  a. 從您假設的 IAM 角色提供存取金鑰 ID、秘密存取金鑰和工作階段權杖。如需如何擷取這些登入資料的詳細資訊，請參閱 IAM 使用者指南中的要求臨時安全登入資料。

  b. 執行下列命令，將認證提供給 AWS IoT Greengrass Core 軟體。

  Linux or Unix

  ```
  export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
  export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
  export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o5OytwEXAMPLE=
  ```

  Windows Command Prompt (CMD)

  ```
  set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
  set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
  set AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o5OytwEXAMPLE=
  ```

  PowerShell

  ```
  $env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
  $env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
  $env:AWS_SESSION_TOKEN="AQoDYXdzEJr1K...o5OytwEXAMPLE="
  ```

- 使用 IAM 使用者的長期登入資料：

  a. 為您的 IAM 使用者提供存取金鑰 ID 和秘密存取金鑰。您可以為稍後刪除的佈建建立 IAM 使用者。如需提供給使用者的 IAM 政策，請參閱安裝程式佈建資源的最低 IAM 政策。如

需如何擷取長期登入資料的詳細資訊，請參閱《IAM 使用者指南》中的「管理 IAM 使用者存取金鑰」。

b. 執行下列命令，將認證提供給 AWS IoT Greengrass Core 軟體。

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

```
$env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
$env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
```

c. (選擇性) 如果您建立 IAM 使用者來佈建 Greengrass 裝置，請刪除該使用者。

d. (選用) 如果您使用現有 IAM 使用者的存取金鑰 ID 和秘密存取金鑰，請更新使用者的金鑰，使其不再有效。如需詳細資訊，請參閱AWS Identity and Access Management使用指南中的更新存取金鑰。

下載AWS IoT Greengrass核心軟體

您可以從下列位置下載最新版本的 AWS IoT Greengrass Core 軟體：

- https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest. 壓縮

> **Note**
>
> 您可以從下列位置下載特定版本的 AWS IoT Greengrass Core 軟體。將##取代為要下載的版本。
>
> ```
> https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
> ```

若要下載AWS IoT Greengrass核心軟體

1. 在您的核心裝置上，將 AWS IoT Greengrass Core 軟體下載至名為`greengrass-nucleus-latest.zip`.

   Linux or Unix

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip > greengrass-nucleus-latest.zip
   ```

   Windows Command Prompt (CMD)

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip > greengrass-nucleus-latest.zip
   ```

   PowerShell

   ```
   iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip -OutFile greengrass-nucleus-latest.zip
   ```

   下載此軟體，即表示您同意 Greengrass 核心軟體授權合約之規定。

2. (選擇性) 驗證 Greengrass 核軟體簽章

   > **Note**
   >
   > 此功能適用於 Greengrass 核 2.9.5 及更高版本。

   a. 使用以下命令來驗證 Greengrass 核工件的簽名：

      Linux or Unix

      ```
      jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
      ```

      Windows Command Prompt (CMD)

      根據您安裝的 JDK 版本，檔案名稱可能會有所不同。以您安裝的 JDK 版本取代*jdk17.0.6_10*。

```
"C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -
verify -certs -verbose greengrass-nucleus-latest.zip
```

PowerShell

根據您安裝的 JDK 版本，檔案名稱可能會有所不同。以您安裝的 JDK 版本取代*jdk17.0.6_10*。

```
'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -
verify -certs -verbose greengrass-nucleus-latest.zip
```

b. `jarsigner`調用產生指示驗證結果的輸出。

   i. 如果 Greengrass 核 zip 文件簽名，則輸出包含以下語句：

   ```
   jar verified.
   ```

   ii. 如果 Greengrass 核壓縮檔未簽署，則輸出會包含下列陳述式：

   ```
   jar is unsigned.
   ```

c. 如果您提供了 Jarsigner `-certs -verbose` 選項`-verify`和選項，則輸出也會包含詳細的簽署者憑證資訊。

3. 將 AWS IoT Greengrass Core 軟件解壓縮到設備上的文件夾。以您要使用的資料夾取*GreengrassInstaller*代。

   Linux or Unix

   ```
   unzip greengrass-nucleus-latest.zip -d GreengrassInstaller && rm greengrass-
   nucleus-latest.zip
   ```

   Windows Command Prompt (CMD)

   ```
   mkdir GreengrassInstaller && tar -xf greengrass-nucleus-latest.zip -
   C GreengrassInstaller && del greengrass-nucleus-latest.zip
   ```

PowerShell

```
Expand-Archive -Path greengrass-nucleus-latest.zip -DestinationPath .\
\GreengrassInstaller
rm greengrass-nucleus-latest.zip
```

4. (選擇性) 執行下列命令以查看 AWS IoT Greengrass Core 軟體的版本。

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

> **Important**
>
> 如果您安裝的 Greengrass 核心版本早於 v2.4.0，請不要在安裝核心軟體之後移除此資料夾。AWS IoT GreengrassAWS IoT Greengrass核心軟件使用此文件夾中的文件運行。如果您下載了最新版本的軟體，請安裝 v2.4.0 或更新版本，而且您可以在安裝 AWS IoT Greengrass Core 軟體之後移除此資料夾。

安裝 AWS IoT Greengrass 核心軟體

使用指定執行下列動作的引數來執行安裝程式：

- 建立核心裝置運作所需的AWS資源。
- 指定使用`ggc_user`系統使用者在核心裝置上執行軟體元件。在 Linux 裝置上，此指令也會指定使用`ggc_group`系統群組，而安裝程式會為您建立系統使用者和群組。
- 將 AWS IoT Greengrass Core 軟體設定為在開機時執行的系統服務。在 Linux 設備上，這需要系統初始化系統。

> **Important**
>
> 在 Windows 核心裝置上，您必須將AWS IoT Greengrass核心軟體設定為系統服務。

若要使用本機開發工具設定開發裝置，請指定`--deploy-dev-tools true`引數。安裝完成後，本機開發工具最多可能需要一分鐘的時間來部署。

如需可指定之引數的詳細資訊，請參閱安裝器引數。

> **Note**
>
> 如果您在記憶體有限的裝置AWS IoT Greengrass上執行，您可以控制 AWS IoT Greengrass Core 軟體使用的記憶體數量。若要控制記憶體分配，您可以在核心元件的`jvmOptions`組態參數中設定 JVM 堆積大小選項。如需詳細資訊，請參閱 使用 JVM 選項控制內存分配。

安裝 AWS IoT Greengrass 核心軟體

1. 執行AWS IoT Greengrass核心安裝程式。取代指令中的引數值，如下所示。

   a. */greengrass/v2*或 *C:\greengrass\v2*：用來安裝AWS IoT Greengrass核心軟體的根資料夾路徑。

   b. *GreengrassInstaller*。解壓縮 AWS IoT Greengrass Core 軟體安裝程式的資料夾路徑。

   c. *##*。要AWS 區域在其中尋找或建立資源。

   d. *MyGreengrassCore*。您 Greengrass 核心裝置AWS IoT的項目名稱。如果該物件不存在，安裝程序將創建它。安裝程式會下載憑證以進行驗證。AWS IoT如需詳細資訊，請參閱 AWS IoT Greengrass 的裝置身分驗證和授權。

      > **Note**
      >
      > 物件名稱不能包含冒號 (`:`) 字元。

   e. *MyGreengrassCoreGroup*。您 Greengrass 核心裝置的AWS IoT物件群組名稱。如果物件群組不存在，安裝程式會建立該物件並將該物件新增至其中。如果物群組存在且具有使用中部署，則核心裝置會下載並執行部署指定的軟體。

      > **Note**
      >
      > 物件群組名稱不能包含冒號 (`:`) 字元。

   f. *##### V2IoT ThingPolicy*。允許 Greengrass 核心裝置與和通訊的AWS IoT原則名稱。AWS IoT AWS IoT Greengrass如果原AWS IoT則不存在，安裝程式會以此名稱建立寬鬆AWS IoT原則。您可以針對您的使用案例限制此原則的權限。如需詳細資訊，請參閱 AWS IoT Greengrass V2核心裝置的最低AWS IoT原則。

   g. *##### V2 TokenExchangeRole*。允許 Greengrass 核心裝置取得臨時登入資料的 IAM 角色名稱。AWS如果角色不存在，安裝程式會建立該角色，並建立並附加名為的策

略*GreengrassV2TokenExchangeRole*Access。如需詳細資訊，請參閱 授權核心設備與AWS服務。

h. *GreengrassCoreTokenExchangeRoleAlias*。IAM 角色的別名，可讓 Greengrass 核心裝置稍後取得臨時登入資料。如果角色別名不存在，安裝程式會建立它，並將其指向您指定的 IAM 角色。如需詳細資訊，請參閱 授權核心設備與AWS服務。

Linux or Unix

```
sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \
  -jar ./GreengrassInstaller/lib/Greengrass.jar \
  --aws-region region \
  --thing-name MyGreengrassCore \
  --thing-group-name MyGreengrassCoreGroup \
  --thing-policy-name GreengrassV2IoTThingPolicy \
  --tes-role-name GreengrassV2TokenExchangeRole \
  --tes-role-alias-name GreengrassCoreTokenExchangeRoleAlias \
  --component-default-user ggc_user:ggc_group \
  --provision true \
  --setup-system-service true
```

Windows Command Prompt (CMD)

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" ^
  -jar ./GreengrassInstaller/lib/Greengrass.jar ^
  --aws-region region ^
  --thing-name MyGreengrassCore ^
  --thing-group-name MyGreengrassCoreGroup ^
  --thing-policy-name GreengrassV2IoTThingPolicy ^
  --tes-role-name GreengrassV2TokenExchangeRole ^
  --tes-role-alias-name GreengrassCoreTokenExchangeRoleAlias ^
  --component-default-user ggc_user ^
  --provision true ^
  --setup-system-service true
```

PowerShell

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" `
  -jar ./GreengrassInstaller/lib/Greengrass.jar `
  --aws-region region `
  --thing-name MyGreengrassCore `
```

```
  --thing-group-name MyGreengrassCoreGroup `
  --thing-policy-name GreengrassV2IoTThingPolicy `
  --tes-role-name GreengrassV2TokenExchangeRole `
  --tes-role-alias-name GreengrassCoreTokenExchangeRoleAlias `
  --component-default-user ggc_user `
  --provision true `
  --setup-system-service true
```

> **Important**
>
> 在 Windows 核心裝置上，您必須指`--setup-system-service true`定將AWS IoT Greengrass核心軟體設定為系統服務。

如果成功，安裝程式會列印下列訊息：

- 如果您指定`--provision`，安裝程式會列印是`Successfully configured Nucleus with provisioned resource details`否已成功設定資源。
- 如果您指定`--deploy-dev-tools`，安裝程式會列印是`Configured Nucleus to deploy aws.greengrass.Cli component`否成功建立部署。
- 如果您指定`--setup-system-service true`，安裝程式會列印它是`Successfully set up Nucleus as a system service`否設定並以服務形式執行軟體。
- 如果未指定`--setup-system-service true`，安裝程式會列印是`Launched Nucleus successfully`否成功並執行軟體。

2. 如果您已安裝 Greengrass 核 v2.0.4 或更新版本，請略過此步驟。如果您下載了最新版本的軟件，則安裝了 v2.0.4 或更高版本。

   執行下列命令，為您的 AWS IoT Greengrass Core 軟體根資料夾設定所需的檔案權限。取代*/greengrass/v2*為您在安裝指令中指定的根資料夾，並將 */greengrass* 取代為根資料夾的父資料夾。

   ```
   sudo chmod 755 /greengrass/v2 && sudo chmod 755 /greengrass
   ```

如果您將 AWS IoT Greengrass Core 軟體安裝為系統服務，安裝程式會為您執行該軟體。否則，您必須手動運行該軟件。如需詳細資訊，請參閱 執行AWS IoT Greengrass核心軟體。

> **Note**
>
> 根據預設，安裝程式建立的 IAM 角色不允許存取 S3 儲存貯體中的元件成品。若要在 Amazon S3 中部署定義成品的自訂元件，您必須向角色新增許可，以允許核心裝置擷取元件成品。如需詳細資訊，請參閱 允許存取 S3 儲存貯體的存取。
> 如果您還沒有適用於元件成品的 S3 儲存貯體，您可以在稍後建立儲存貯體後新增這些許可。

如需有關如何設定及使用軟體的詳細資訊AWS IoT Greengrass，請參閱下列內容：

- 設定 AWS IoT Greengrass 核心軟體
- 開發AWS IoT Greengrass元件
- 將AWS IoT Greengrass元件部署到裝置
- 綠色命令行界面

透過手動佈建資源安裝 AWS IoT Greengrass 核心軟體

AWS IoT Greengrass 核心軟體包含一個安裝程式，可將您的裝置設定為 Greengrass 核心裝置。若要手動設定裝置，您可以建立要使用的必要 AWS IoT 和 IAM 資源。如果您手動建立這些資源，則不需要提供 AWS 認證給安裝程式。

當您手動安裝 AWS IoT Greengrass Core 軟體時，您也可以將裝置設定為使用網路 Proxy，或在連接埠 443 AWS 上連線。例如，如果您的裝置在防火牆或網路 Proxy 後方執行，您可能需要指定這些組態選項。如需詳細資訊，請參閱 連線至連接埠 443 或透過網路代理。

您也可以透過 PKCS #11 介面將 AWS IoT Greengrass 核心軟體設定為使用硬體安全性模組 (HSM)。此功能可讓您安全地儲存私密金鑰和憑證檔案，以免在軟體中公開或複製這些檔案。您可以將私密金鑰和憑證儲存在硬體模組上，例如 HSM、信任平台模組 (TPM) 或其他加密元素。此功能僅適用於 Linux 裝置。如需有關硬體安全性及其使用需求的詳細資訊，請參閱硬體安全整合。

> **Important**
>
> 在您下載 AWS IoT Greengrass Core 軟體之前，請檢查您的核心裝置是否符合安裝和執行 AWS IoT Greengrass Core 軟體 2.0 的需求。

主題

- 擷取 AWS IoT 端點
- 建立物 AWS IoT 件
- 建立物件憑證
- 配置物憑證
- 建立權杖交換角色
- 將憑證下載到裝置
- 設定裝置環境
- 下載 AWS IoT Greengrass 核心軟體
- 安裝 AWS IoT Greengrass 核心軟體

擷取 AWS IoT 端點

取得您的 AWS IoT 端點 AWS 帳戶，並儲存以供稍後使用。您的裝置會使用這些端點連線至 AWS IoT。請執行下列操作：

1. 取得適 AWS IoT 用於您的 AWS 帳戶.

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

   如果要求成功，回應看起來類似下列範例。

   ```
   {
     "endpointAddress": "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
   }
   ```

2. 取 AWS IoT 得您的 AWS 帳戶.

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

   如果要求成功，回應看起來類似下列範例。

   ```
   {
     "endpointAddress": "device-credentials-prefix.credentials.iot.us-
   west-2.amazonaws.com"
   }
   ```

建立物 AWS IoT 件

AWS IoT 東西代表連接到的設備和邏輯實體 AWS IoT。Greengrass 核心設備是事情。 AWS IoT 當您將裝置註冊為 AWS IoT 物件時，該裝置可以使用數位憑證來進行驗證 AWS。

在本節中，您會建立代表您裝置的 AWS IoT 物件。

若要建立 AWS IoT 物件

1. 為您的裝置建立項 AWS IoT 目。在您的開發電腦上，執行下列命令。

   - 以要*MyGreengrassCore*使用的物件名稱取代。這個名字也是您的 Greengrass 核心裝置的名稱。

   > **Note**
   >
   > 物件名稱不能包含冒號 (:) 字元。

   ```
   aws iot create-thing --thing-name MyGreengrassCore
   ```

   如果要求成功，回應看起來類似下列範例。

   ```
   {
     "thingName": "MyGreengrassCore",
     "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
     "thingId": "8cb4b6cd-268e-495d-b5b9-1713d71dbf42"
   }
   ```

2. (選擇性) 將 AWS IoT 物件新增至新的或現有的物件群組。您可以使用物件群組來管理 Greengrass 核心裝置的叢集。將軟體元件部署到裝置時，您可以鎖定個別裝置或裝置群組。您可以將裝置新增至具有作用中 Greengrass 部署的物群組，以將該物件群組的軟體元件部署至裝置。請執行下列操作：

   a. (選擇性) 建立 AWS IoT 物件群組。

      - 以要建立的物件群組名稱取*MyGreengrassCoreGroup*代。

> **Note**
>
> 物件群組名稱不能包含冒號 (:) 字元。

```
aws iot create-thing-group --thing-group-name MyGreengrassCoreGroup
```

如果要求成功，回應看起來類似下列範例。

```
{
  "thingGroupName": "MyGreengrassCoreGroup",
  "thingGroupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/
MyGreengrassCoreGroup",
  "thingGroupId": "4df721e1-ff9f-4f97-92dd-02db4e3f03aa"
}
```

b. 將 AWS IoT 物件新增至物件群組。

- 替換*MyGreengrassCore*為您的 AWS IoT 事物的名稱。
- *MyGreengrassCoreGroup*以物件群組的名稱取代。

```
aws iot add-thing-to-thing-group --thing-name MyGreengrassCore --thing-group-
name MyGreengrassCoreGroup
```

如果請求成功，該命令沒有任何輸出。

建立物件憑證

當您將裝置註冊為 AWS IoT 物件時，該裝置可以使用數位憑證來進行驗證 AWS。此憑證可讓裝置與 AWS IoT 和通訊 AWS IoT Greengrass。

在本節中，您會建立並下載裝置可用來連線的憑證 AWS。

如果您想要將 AWS IoT Greengrass Core 軟體設定為使用硬體安全性模組 (HSM) 來安全地儲存私密金鑰和憑證，請依照下列步驟從 HSM 中的私密金鑰建立憑證。否則，請按照以下步驟在 AWS IoT 服務中創建證書和私鑰。硬體安全性功能僅適用於 Linux 裝置。如需有關硬體安全性及其使用需求的詳細資訊，請參閱硬體安全整合。

在 AWS IoT 服務中建立憑證和私密金鑰

若要建立物憑證

1. 建立下載 AWS IoT 物件憑證的資料夾。

```
mkdir greengrass-v2-certs
```

2. 建立並下載 AWS IoT 物件的憑證。

```
aws iot create-keys-and-certificate --set-as-active --certificate-pem-outfile
 greengrass-v2-certs/device.pem.crt --public-key-outfile greengrass-v2-certs/
public.pem.key --private-key-outfile greengrass-v2-certs/private.pem.key
```

如果要求成功，回應看起來類似下列範例。

```
{
  "certificateArn": "arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
  "certificateId":
 "aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
  "certificatePem": "-----BEGIN CERTIFICATE-----
MIICiTCCAfICCQD6m7oRw0uXOjANBgkqhkiG9w
 0BAQUFADCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZ
 WF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIw
 EAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5
 jb20wHhcNMTEwNDI1MjA0NTIxWhcNMTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBh
 MCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBb
 WF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMx
 HzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5jb20wgZ8wDQYJKoZIhvcNAQE
 BBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ21uUSfwfEvySWtC2XADZ4nB+BLYgVI
 k60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9TrDHudUZg3qX4waLG5M43q7Wgc/MbQ
 ITxOUSQv7c7ugFFDzQGBzZswY6786m86gpEIbb3OhjZnzcvQAaRHhdlQWIMm2nr
 AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4nUhVVxYUntneD9+h8Mg9q6q+auN
 KyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0FkbFFBjvSfpJIlJ00zbhNYS5f6Guo
 EDmFJl0ZxBHjJnyp378OD8uTs7fLvjx79LjSTbNYiytVbZPQUQ5Yaxu2jXnimvw
 3rrszlaEXAMPLE=
-----END CERTIFICATE-----",
  "keyPair": {
    "PublicKey": "-----BEGIN PUBLIC KEY-----\
MIIBIjANBgkqhkEXAMPLEQEFAAOCAQ8AMIIBCgKCAQEAEXAMPLE1nnyJwKSMHw4h\
MMEXAMPLEuuN/dMAS3fyce8DW/4+EXAMPLEyjmoF/YVF/gHr99VEEXAMPLE5VF13\
```

```
59VK7cEXAMPLE67GK+y+jikqXOgHh/xJTwo
+sGpWEXAMPLEDz18xOd2ka4tCzuWEXAMPLEahJbYkCPUBSU8opVkR7qkEXAMPLE1DR6sx2HocliOOLtu6Fkw91swQWE
\\GB3ZPrNh0PzQYvjUStZeccyNCx2EXAMPLEvp9mQOUXP6plfgxwKRX2fEXAMPLEDa\
hJLXkX3rHU2xbxJSq7D+XEXAMPLEcw+LyFhI5mgFRl88eGdsAEXAMPLElnI9EesG\
FQIDAQAB\
-----END PUBLIC KEY-----\
",
    "PrivateKey": "-----BEGIN RSA PRIVATE KEY-----\
key omitted for security reasons\
-----END RSA PRIVATE KEY-----\
"
  }
}
```

儲存憑證的 Amazon 資源名稱 (ARN)，以便稍後用來設定憑證。

從 HSM 中的私密金鑰建立憑證

> **Note**
>
> 此功能適用於 v2.5.3 及更高版 Greeng rass 核組件。 AWS IoT Greengrass 目前在 Windows 核心裝置上不支援此功能。

若要建立物憑證

1. 在核心裝置上，初始化 HSM 中的 PKCS #11 權杖，然後產生私密金鑰。私密金鑰必須是具有 RSA-2048 金鑰大小 (或更大) 的 RSA 金鑰或 ECC 金鑰。

   > **Note**
   >
   > 若要搭配 ECC 金鑰使用硬體安全性模組，您必須使用 Greengrass 核心 v2.5.6 或更新版本。
   > 要使用硬件安全模塊和密碼管理器，您必須使用帶有 RSA 密鑰的硬件安全模塊。

   請查看 HSM 的文件，以了解如何初始化權杖並產生私密金鑰。如果您的 HSM 支援物件 ID，請在產生私密金鑰時指定物件 ID。儲存初始化權杖並產生私密金鑰時指定的插槽 ID、使用者 PIN

碼、物件標籤、物件 ID (如果您的 HSM 使用的話)。稍後當您將物憑證匯入 HSM 並設定 AWS IoT Greengrass Core 軟體時，您可以使用這些值。

2. 從私密金鑰建立憑證簽署要求 (CSR)。 AWS IoT 使用此 CSR 為您在 HSM 中產生的私密金鑰建立物件憑證。如需如何從私密金鑰建立 CSR 的詳細資訊，請參閱 HSM 的說明文件。CSR 是一個檔案，例如`iotdevicekey.csr`。

3. 將 CSR 從裝置複製到您的開發電腦。如果在開發電腦和裝置上啟用了 SSH 和 SCP，您可以使用開發電腦上的`scp`指令來傳輸 CSR。取代您裝置*`device-ip-address`*的 IP 位址，並將 *`~/iotdevicekey.csr ###### CSR #`*案的路徑。

```
scp device-ip-address:~/iotdevicekey.csr iotdevicekey.csr
```

4. 在開發電腦上，建立下載物件憑證 AWS IoT 的資料夾。

```
mkdir greengrass-v2-certs
```

5. 使用 CSR 檔案建立 AWS IoT 物件的憑證，並將其下載到您的開發電腦。

```
aws iot create-certificate-from-csr --set-as-active --certificate-signing-
request=file://iotdevicekey.csr --certificate-pem-outfile greengrass-v2-certs/
device.pem.crt
```

如果要求成功，回應看起來類似下列範例。

```
{
  "certificateArn": "arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
  "certificateId":
 "aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
  "certificatePem": "-----BEGIN CERTIFICATE-----
MIICiTCCAfICCQD6m7oRw0uXOjANBgkqhkiG9w
 0BAQUFADCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZ
 WF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIw
 EAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5
 jb20wHhcNMTEwNDI1MjA0NTIxWhcNMTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBh
 MCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBb
 WF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMx
 HzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5jb20wgZ8wDQYJKoZIhvcNAQE
 BBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ21uUSfwfEvySWtC2XADZ4nB+BLYgVI
 k60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9TrDHudUZg3qX4waLG5M43q7Wgc/MbQ
 ITxOUSQv7c7ugFFDzQGBzZswY6786m86gpEIbb3OhjZnzcvQAaRHhdlQWIMm2nr
```

```
 AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4nUhVVxYUntneD9+h8Mg9q6q+auN
 KyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0FkbFFBjvSfpJIlJ00zbhNYS5f6Guo
 EDmFJl0ZxBHjJnyp378OD8uTs7fLvjx79LjSTbNYiytVbZPQUQ5Yaxu2jXnimvw
 3rrszlaEXAMPLE=
-----END CERTIFICATE-----"
}
```

儲存憑證的 ARN，以便稍後用來設定憑證。

配置物憑證

將物憑證附加至您先前建立的 AWS IoT 物件，並將 AWS IoT 原則新增至憑證以定義核心裝置的 AWS IoT 權限。

若要設定物件的憑證

1. 將憑證附加至物 AWS IoT 件。

   - 替換*MyGreengrassCore*為您的 AWS IoT 事物的名稱。
   - 將憑證 Amazon 資源名稱 (ARN) 取代為您在上一個步驟中建立的憑證的 ARN。

   ```
   aws iot attach-thing-principal --thing-name MyGreengrassCore
    --principal arn:aws:iot:us-west-2:123456789012:cert/
   aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
   ```

   如果請求成功，該命令沒有任何輸出。

2. 建立並附加 AWS IoT 定義 Greengrass 核心裝置 AWS IoT 權限的原則。下列原則允許存取所有 MQTT 主題和 Greengrass 作業，因此您的裝置可以處理需要新 Greengrass 作業的自訂應用程式和 future 變更。您可以根據您的使用案例來限制此政策。如需詳細資訊，請參閱 AWS IoT Greengrass V2核心裝置的最低AWS IoT原則。

   如果您之前已設定過 Greengrass 核心裝置，則可以附加其 AWS IoT 原則，而不是建立新的原則。

   請執行下列操作：

   a. 建立包含 Greengrass 核心裝置所需之 AWS IoT 原則文件的檔案。

      例如，在 Linux 系統上，您可以執行下列命令來使用 GNU nano 來建立檔案。

```
nano greengrass-v2-iot-policy.json
```

將下列 JSON 複製到檔案中。

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

b. 從 AWS IoT 策略文件建立策略。

- 以要建*#########ThingPolicy####*。

```
aws iot create-policy --policy-name GreengrassV2IoTThingPolicy --policy-
document file://greengrass-v2-iot-policy.json
```

如果要求成功，回應看起來類似下列範例。

```
{
  "policyName": "GreengrassV2IoTThingPolicy",
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy",
  "policyDocument": "{
    \\"Version\\": \\"2012-10-17\\",
    \\"Statement\\": [
      {
```

```
        \\"Effect\\": \\"Allow\\",
        \\"Action\\": [
          \\"iot:Publish\\",
          \\"iot:Subscribe\\",
          \\"iot:Receive\\",
          \\"iot:Connect\\",
          \\"greengrass:*\\"
        ],
        \\"Resource\\": [
          \\"*\\"
        ]
      }
    ]
  }",
  "policyVersionId": "1"
}
```

c. 將 AWS IoT 原則附加至物 AWS IoT 件的憑證。

- 以要附*#########ThingPolicy####*。
- 將目標 ARN 取代為物件憑證的 AWS IoT ARN。

```
aws iot attach-policy --policy-name GreengrassV2IoTThingPolicy
 --target arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
```

如果請求成功，該命令沒有任何輸出。

建立權杖交換角色

Greengrass 核心裝置使用 IAM 服務角色 (稱為權杖交換角色) 來授權對服務的呼叫。 AWS 裝置使用 AWS IoT 登入資料提供者取得此角色的臨時 AWS 登入資料，以便裝置與之互動 AWS IoT、將日誌傳送到 Amazon CloudWatch Logs，以及從 Amazon S3 下載自訂元件成品。如需詳細資訊，請參閱 授權核心設備與AWS服務。

您可以使用 AWS IoT 角色別名來設定 Greengrass 核心裝置的權杖交換角色。角色別名可讓您變更裝置的 Token 交換角色，但保持裝置設定不變。如需詳細資訊，請參閱AWS IoT Core 開發人員指南中的授權直接呼叫 AWS 服務。

在本節中，您會建立權杖交換 IAM 角色和指向該 AWS IoT 角色的角色別名。如果您已經設置了 Greengrass 核心設備，則可以使用其令牌交換角色和角色別名，而不是創建新的。然後，您將設備的 AWS IoT 東西配置為使用該角色和別名。

若要建立權杖交換 IAM 角色

1. 建立 IAM 角色，您的裝置可用作權杖交換角色。請執行下列操作：

   a. 建立包含 Token 交換角色所需之信任原則文件的檔案。

      例如，在 Linux 系統上，您可以執行下列命令來使用 GNU nano 來建立檔案。

      ```
      nano device-role-trust-policy.json
      ```

      將下列 JSON 複製到檔案中。

      ```
      {
        "Version": "2012-10-17",
        "Statement": [
          {
            "Effect": "Allow",
            "Principal": {
              "Service": "credentials.iot.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
          }
        ]
      }
      ```

   b. 使用信任原則文件建立權杖交換角色。

      - TokenExchangeRole以要建立的身分與存取權管理員角色的名稱取代 *GreenGrassv2*。

      ```
      aws iam create-role --role-name GreengrassV2TokenExchangeRole --assume-role-
      policy-document file://device-role-trust-policy.json
      ```

      如果要求成功，回應看起來類似下列範例。

      ```
      {
        "Role": {
      ```

```
    "Path": "/",
    "RoleName": "GreengrassV2TokenExchangeRole",
    "RoleId": "AROAZ2YMUHYHK5OKM77FB",
    "Arn": "arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole",
    "CreateDate": "2021-02-06T00:13:29+00:00",
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": "credentials.iot.amazonaws.com"
          },
          "Action": "sts:AssumeRole"
        }
      ]
    }
  }
```

c. 建立包含權杖交換角色所需之存取原則文件的檔案。

例如，在 Linux 系統上，您可以執行下列命令來使用 GNU nano 來建立檔案。

```
nano device-role-access-policy.json
```

將下列 JSON 複製到檔案中。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    }
  ]
```

```
}
```

> **Note**
>
> 此存取政策不允許存取 S3 儲存貯體中的元件成品。若要在 Amazon S3 中部署定義成品的自訂元件，您必須向角色新增許可，以允許核心裝置擷取元件成品。如需詳細資訊，請參閱 允許存取 S3 儲存貯體的存取。
> 如果您還沒有適用於元件成品的 S3 儲存貯體，您可以在稍後建立儲存貯體後新增這些許可。

d. 從政策文件建立 IAM 政策。

- TokenExchangeRoleAccess以要建立的身分與存取權管理政策的名稱取代 *GreenGrassv2*。

```
aws iam create-policy --policy-name GreengrassV2TokenExchangeRoleAccess --
policy-document file://device-role-access-policy.json
```

如果要求成功，回應看起來類似下列範例。

```
{
  "Policy": {
    "PolicyName": "GreengrassV2TokenExchangeRoleAccess",
    "PolicyId": "ANPAZ2YMUHYHACI7C5Z66",
    "Arn": "arn:aws:iam::123456789012:policy/
GreengrassV2TokenExchangeRoleAccess",
    "Path": "/",
    "DefaultVersionId": "v1",
    "AttachmentCount": 0,
    "PermissionsBoundaryUsageCount": 0,
    "IsAttachable": true,
    "CreateDate": "2021-02-06T00:37:17+00:00",
    "UpdateDate": "2021-02-06T00:37:17+00:00"
  }
}
```

e. 將 IAM 政策附加到權杖交換角色。

- 以 IAM 角*#######TokenExchangeRole# V2*。

- 將政策 ARN 取代為您在上一個步驟中建立的 IAM 政策的 ARN。

```
aws iam attach-role-policy --role-name GreengrassV2TokenExchangeRole --policy-
arn arn:aws:iam::123456789012:policy/GreengrassV2TokenExchangeRoleAccess
```

如果請求成功，該命令沒有任何輸出。

2. 建立指向權杖交換 AWS IoT 角色的角色別名。

- 以要建立之角色別名的名稱取*GreengrassCoreTokenExchangeRoleAlias*代。
- 將角色 ARN 取代為您在上一個步驟中建立的 IAM 角色的 ARN。

```
aws iot create-role-alias --role-alias GreengrassCoreTokenExchangeRoleAlias --role-
arn arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole
```

如果要求成功，回應看起來類似下列範例。

```
{
  "roleAlias": "GreengrassCoreTokenExchangeRoleAlias",
  "roleAliasArn": "arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias"
}
```

> **Note**
>
> 若要建立角色別名，您必須擁有權限才能將權杖交換 IAM 角色傳遞給 AWS IoT。如果您在嘗試建立角色別名時收到錯誤訊息，請檢查您的 AWS 使用者是否具有此權限。如需詳細資訊，請參閱《使用指南》中的授與使用AWS Identity and Access Management 者將角色傳遞給 AWS 服務的權限。

3. 建立並附加 AWS IoT 原則，讓您的 Greengrass 核心裝置使用角色別名來承擔權杖交換角色。如果您之前已經設定過 Greengrass 核心裝置，則可以附加其角色別名 AWS IoT 原則，而不是建立新的別名原則。請執行下列操作：

a. (選擇性) 建立包含角色別名所需之 AWS IoT 原則文件的檔案。

例如，在 Linux 系統上，您可以執行下列命令來使用 GNU nano 來建立檔案。

```
nano greengrass-v2-iot-role-alias-policy.json
```

將下列 JSON 複製到檔案中。

- 將資源 ARN 取代為角色別名的 ARN。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:AssumeRoleWithCertificate",
      "Resource": "arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias"
    }
  ]
}
```

b. 從 AWS IoT 策略文件建立策略。

- 以要建立的 AWS IoT 策略名稱取*GreengrassCoreTokenExchangeRoleAliasPolicy*代。

```
aws iot create-policy --policy-name GreengrassCoreTokenExchangeRoleAliasPolicy
 --policy-document file://greengrass-v2-iot-role-alias-policy.json
```

如果要求成功，回應看起來類似下列範例。

```
{
  "policyName": "GreengrassCoreTokenExchangeRoleAliasPolicy",
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassCoreTokenExchangeRoleAliasPolicy",
  "policyDocument": "{
    \\"Version\\":\\"2012-10-17\\",
    \\"Statement\\": [
      {
        \\"Effect\\": \\"Allow\\",
        \\"Action\\": \\"iot:AssumeRoleWithCertificate\\",
```

```
        \\"Resource\\": \\"arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias\\"
      }
    ]
  }",
  "policyVersionId": "1"
}
```

c. 將 AWS IoT 原則附加至物 AWS IoT 件的憑證。

- 以角*GreengrassCoreTokenExchangeRoleAliasPolicy*色別名 AWS IoT 原則的名稱取代。
- 將目標 ARN 取代為物件憑證的 AWS IoT ARN。

```
aws iot attach-policy --policy-name GreengrassCoreTokenExchangeRoleAliasPolicy
 --target arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
```

如果請求成功，該命令沒有任何輸出。

將憑證下載到裝置

之前，您已將裝置的憑證下載到開發電腦。在本節中，您會將憑證複製到您的核心裝置，以使用用來連線的憑證來設定裝置 AWS IoT。您也會下載 Amazon 根憑證授權單位 (CA) 憑證。如果您使用 HSM，也會將憑證檔案匯入本節中的 HSM。

- 如果您先前在 AWS IoT 服務中建立了物憑證和私密金鑰，請依照下列步驟下載含有私密金鑰和憑證檔案的憑證。
- 如果您先前從硬體安全性模組 (HSM) 中的私密金鑰建立物憑證，請依照下列步驟在 HSM 中下載含有私密金鑰和憑證的憑證。

下載包含私密金鑰和憑證檔案的憑證

將憑證下載到裝置

1. 將 AWS IoT 物件憑證從開發電腦複製到裝置。如果在開發電腦和裝置上啟用了 SSH 和 SCP，您可以使用開發電腦上的`scp`指令來傳輸憑證。*device-ip-address*替換為設備的 IP 地址。

```
scp -r greengrass-v2-certs/ device-ip-address:~
```

2. 在設備上創建綠色根文件夾。您稍後將 AWS IoT Greengrass 核心軟體安裝到此資料夾中。

   Linux or Unix

   - 以要使*/greengrass/v2*用的資料夾取代。

   ```
   sudo mkdir -p /greengrass/v2
   ```

   Windows Command Prompt

   - 將 *C:\greengrass\v2* 取代為要使用的資料夾。

   ```
   mkdir C:\greengrass\v2
   ```

   PowerShell

   - 將 *C:\greengrass\v2* 取代為要使用的資料夾。

   ```
   mkdir C:\greengrass\v2
   ```

3. (僅限 Linux) 設定 Greengrass 根資料夾的父系權限。

   - 將 */greengrass* 替換為根文件夾的父文件夾。

   ```
   sudo chmod 755 /greengrass
   ```

4. 將 AWS IoT 物件憑證複製到 Greengrass 根資料夾。

   Linux or Unix

   - */greengrass/v2*以 Greengrass 色根資料夾取代。

   ```
   sudo cp -R ~/greengrass-v2-certs/* /greengrass/v2
   ```

Windows Command Prompt

- 將 *C:\greengrass\v2* 取代為要使用的資料夾。

```
robocopy %USERPROFILE%\greengrass-v2-certs C:\greengrass\v2 /E
```

PowerShell

- 將 *C:\greengrass\v2* 取代為要使用的資料夾。

```
cp -Path ~\greengrass-v2-certs\* -Destination C:\greengrass\v2
```

5. 下載 Amazon 根憑證授權單位 (CA) 憑證。 AWS IoT 憑證預設會與 Amazon 的根 CA 憑證相關聯。

   Linux or Unix

   ```
   sudo curl -o /greengrass/v2/AmazonRootCA1.pem https://www.amazontrust.com/
   repository/AmazonRootCA1.pem
   ```

   Windows Command Prompt (CMD)

   ```
   curl -o C:\greengrass\v2\\AmazonRootCA1.pem https://www.amazontrust.com/
   repository/AmazonRootCA1.pem
   ```

   PowerShell

   ```
   iwr -Uri https://www.amazontrust.com/repository/AmazonRootCA1.pem -OutFile C:
   \greengrass\v2\\AmazonRootCA1.pem
   ```

在 HSM 中下載具有私密金鑰和憑證的憑證

> **Note**
>
> 此功能適用於 v2.5.3 及更高版 Greeng rass 核組件。 AWS IoT Greengrass 目前在 Windows 核心裝置上不支援此功能。

將憑證下載到裝置

1. 將 AWS IoT 物件憑證從開發電腦複製到裝置。如果在開發電腦和裝置上啟用了 SSH 和 SCP，您可以使用開發電腦上的scp指令來傳輸憑證。*device-ip-address*替換為設備的 IP 地址。

   ```
   scp -r greengrass-v2-certs/ device-ip-address:~
   ```

2. 在設備上創建綠色根文件夾。您稍後將 AWS IoT Greengrass 核心軟體安裝到此資料夾中。

   Linux or Unix

   - 以要使*/greengrass/v2*用的資料夾取代。

   ```
   sudo mkdir -p /greengrass/v2
   ```

   Windows Command Prompt

   - 將 *C:\greengrass\v2* 取代為要使用的資料夾。

   ```
   mkdir C:\greengrass\v2
   ```

   PowerShell

   - 將 *C:\greengrass\v2* 取代為要使用的資料夾。

   ```
   mkdir C:\greengrass\v2
   ```

3. (僅限 Linux) 設定 Greengrass 根資料夾的父系權限。

   - 將 */greengrass* 替換為根文件夾的父文件夾。

   ```
   sudo chmod 755 /greengrass
   ```

4. 將物憑證檔案匯入 HSM。~/greengrass-v2-certs/device.pem.crt請查看 HSM 的說明文件，以了解如何將憑證匯入其中。使用先前在 HSM 中產生私密金鑰的相同權杖、插槽識別碼、使用者 PIN 碼、物件標籤和物件 ID (如果您的 HSM 使用的話) 匯入憑證。

> **Note**
>
> 如果您先前產生的私密金鑰不含物件 ID，且憑證具有物件 ID，請將私密金鑰的物件 ID 設定為與憑證相同的值。請查看 HSM 的文件，以了解如何設定私密金鑰物件的物件 ID。

5. (選擇性) 刪除物憑證檔案，使其僅存在於 HSM 中。

```
rm ~/greengrass-v2-certs/device.pem.crt
```

6. 下載 Amazon 根憑證授權單位 (CA) 憑證。 AWS IoT 憑證預設會與 Amazon 的根 CA 憑證相關聯。

   Linux or Unix

   ```
   sudo curl -o /greengrass/v2/AmazonRootCA1.pem https://www.amazontrust.com/
   repository/AmazonRootCA1.pem
   ```

   Windows Command Prompt (CMD)

   ```
   curl -o C:\greengrass\v2\\AmazonRootCA1.pem https://www.amazontrust.com/
   repository/AmazonRootCA1.pem
   ```

   PowerShell

   ```
   iwr -Uri https://www.amazontrust.com/repository/AmazonRootCA1.pem -OutFile C:
   \greengrass\v2\\AmazonRootCA1.pem
   ```

設定裝置環境

請依照本節中的步驟設定 Linux 或 Windows 裝置作為 AWS IoT Greengrass 核心裝置使用。

設定裝置

若要設定下列項目的 Linux 設備 AWS IoT Greengrass V2

1. 安裝 Java 執行階段， AWS IoT Greengrass 核心軟體需要執行。我們建議您使用 Amazon Corretto 或 OpenJDK 長期支持版本。需要版本 8 或更高版本。以下命令向您展示如何在設備上安裝 OpenJDK。

- 針對 Debian 為基礎或 Ubuntu 為基礎的發行版本：

```
sudo apt install default-jdk
```

- 針對 Red Hat 為基礎的發行版本：

```
sudo yum install java-11-openjdk-devel
```

- 針對 Amazon Linux 2：

```
sudo amazon-linux-extras install java-openjdk11
```

- 對於 Amazon

```
sudo dnf install java-11-amazon-corretto -y
```

安裝完成時，請執行下列命令以確認 Java 是否在 Linux 裝置上執行。

```
java -version
```

此指令會列印在裝置上執行的 Java 版本。例如，在基於 Debian 的發行版上，輸出可能類似於以下示例。

```
openjdk version "11.0.9.1" 2020-11-04
OpenJDK Runtime Environment (build 11.0.9.1+1-post-Debian-1deb10u2)
OpenJDK 64-Bit Server VM (build 11.0.9.1+1-post-Debian-1deb10u2, mixed mode)
```

2. (選擇性) 建立在裝置上執行元件的預設系統使用者和群組。您也可以選擇讓 AWS IoT Greengrass Core 軟體安裝程式在安裝期間使用安裝程式引數建立此使用者和群組。`--component-default-user`如需詳細資訊，請參閱 安裝器引數。

```
sudo useradd --system --create-home ggc_user
sudo groupadd --system ggc_group
```

3. 確認執行 AWS IoT Greengrass Core 軟體的使用者 (通常`root`) 具有`sudo`與任何使用者和任何群組一起執行的權限。

   a. 執行下列命令以開啟`/etc/sudoers`檔案。

```
sudo visudo
```

b. 確認使用者的權限看起來像下列範例。

```
root    ALL=(ALL:ALL) ALL
```

4. (選擇性) 若要執行容器化 Lambda 函數，您必須啟用 cgroup v1，並且必須啟用並掛接記憶體和裝置 cgroup。如果您不打算執行容器化 Lambda 函數，則可以略過此步驟。

   若要啟用這些 cgroups 選項，請使用下列 Linux 核心參數來啟動裝置。

   ```
   cgroup_enable=memory cgroup_memory=1 systemd.unified_cgroup_hierarchy=0
   ```

   如需有關檢視和設定裝置核心參數的資訊，請參閱作業系統和開機載入程式的說明文件。依照指示永久設定核心參數。

5. 依照中的需求清單所指示，在您的裝置上安裝所有其他必要的相依性裝置要求。

設定視窗裝置

> **Note**
>
> 此功能適用於 v2.5.0 及更高版 Greeng rass 核組件。

若要設定視窗裝置 AWS IoT Greengrass V2

1. 安裝 Java 執行階段， AWS IoT Greengrass 核心軟體需要執行。我們建議您使用 Amazon Corretto 或 OpenJDK 長期支持版本。需要版本 8 或更高版本。
2. 檢查系統變數 PATH 上是否可用 Java，如果沒有，則加入它。此 LocalSystem 帳戶會執行 AWS IoT Greengrass 核心軟體，因此您必須將 Java 新增至系統變數 PATH，而非使用者的 PATH 使用者變數。請執行下列操作：

   a. 按視窗鍵開啟開始功能表。

   b. 鍵入**environment variables**以從開始功能表搜尋系統選項。

   c. 在開始功能表搜尋結果中，選擇 [編輯系統環境變數] 以開啟 [系統屬性] 視窗。

   d. 選擇環境變數... 以開啟 [環境變數] 視窗。

e. 在 [系統變數] 下，選取 [路徑]，然後選擇 [編輯]。在 「編輯」環境變數視窗中，您可以在不同的行上檢視每個路徑。

f. 檢查 Java 安裝bin資料夾的路徑是否存在。路徑看起來可能類似下列範例。

```
C:\\Program Files\\Amazon Corretto\\jdk11.0.13_8\\bin
```

g. 如果路徑中缺少 Java 安裝的bin資料夾，請選擇 [新增] 加入，然後選擇 [確定]。

3. 以系統管理員身分開啟 Windows 命令提示字元 (cmd.exe)。

4. 在 Windows 裝置上的 LocalSystem 帳戶中建立預設使用者。以安全*##*取代密碼。

```
net user /add ggc_user password
```

> **Tip**
>
> 視您的 Windows 組態而定，使用者的密碼可能會設定為在 future 的某個日期到期。為確保您的 Greengrass 應用程式繼續運作，請追蹤密碼何時到期，並在密碼到期之前進行更新。您也可以將使用者的密碼設定為永不過期。
>
> - 若要檢查使用者及其密碼何時到期，請執行下列命令。
>
>   ```
>   net user ggc_user | findstr /C:expires
>   ```
>
> - 若要將使用者的密碼設定為永不過期，請執行下列命令。
>
>   ```
>   wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
>   ```
>
> - 如果您使用的是 Windows 10 或更新版本，而該wmic命令已被取代，請執行下列 PowerShell 命令。
>
>   ```
>   Get-CimInstance -Query "SELECT * from Win32_UserAccount WHERE name =
>    'ggc_user'" | Set-CimInstance -Property @{PasswordExpires="False"}
>   ```

5. 在設備上從 Microsoft 下載並安裝該PsExec實用程序。

6. 使用此 PsExec 公用程式將預設使用者的使用者名稱和密碼儲存在 LocalSystem 帳戶的認證管理員執行個體中。*################*

```
psexec -s cmd /c cmdkey /generic:ggc_user /user:ggc_user /pass:password
```

如果打PsExec License Agreement開，請選Accept擇同意許可證並運行命令。

> **Note**
>
> 在 Windows 裝置上， LocalSystem 帳戶會執行 Greengrass 核心，而且您必須使用公用 PsExec 程式將預設使用者資訊儲存在帳戶中。 LocalSystem 使用認證管理員應用程式會將此資訊儲存在目前登入使用者的 Windows 帳戶中，而非 LocalSystem 帳戶中。

下載 AWS IoT Greengrass 核心軟體

您可以從下列位置下載最新版本的 AWS IoT Greengrass Core 軟體：

- https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest. 壓縮

> **Note**
>
> 您可以從下列位置下載特定版本的 AWS IoT Greengrass Core 軟體。將##取代為要下載的版本。
>
> ```
> https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
> ```

若要下載 AWS IoT Greengrass 核心軟體

1. 在您的核心裝置上，將 AWS IoT Greengrass Core 軟體下載至名為`greengrass-nucleus-latest.zip`.

   Linux or Unix

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip > greengrass-nucleus-latest.zip
   ```

   Windows Command Prompt (CMD)

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip > greengrass-nucleus-latest.zip
   ```

PowerShell

```
iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip -OutFile greengrass-nucleus-latest.zip
```

下載此軟體，即表示您同意 Greengrass 核心軟體授權合約之規定。

2. (選擇性) 驗證 Greengrass 核軟體簽章

> **Note**
>
> 此功能適用於 Greengrass 核 2.9.5 及更高版本。

a. 使用以下命令來驗證 Greengrass 核工件的簽名：

Linux or Unix

```
jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

根據您安裝的 JDK 版本，檔案名稱可能會有所不同。以您安裝的 JDK 版本取代*jdk17.0.6_10*。

```
"C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -verify -certs -verbose greengrass-nucleus-latest.zip
```

PowerShell

根據您安裝的 JDK 版本，檔案名稱可能會有所不同。以您安裝的 JDK 版本取代*jdk17.0.6_10*。

```
'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -verify -certs -verbose greengrass-nucleus-latest.zip
```

b. `jarsigner`調用產生指示驗證結果的輸出。

i. 如果 Greengrass 核 zip 文件簽名，則輸出包含以下語句：

```
jar verified.
```

ii. 如果 Greengrass 核壓縮檔未簽署，則輸出會包含下列陳述式：

```
jar is unsigned.
```

c. 如果您提供了 Jarsigner `-certs -verbose` 選項`-verify`和選項，則輸出也會包含詳細的簽署者憑證資訊。

3. 將 AWS IoT Greengrass Core 軟件解壓縮到設備上的文件夾。以您要使用的資料夾取*GreengrassInstaller*代。

   Linux or Unix

   ```
   unzip greengrass-nucleus-latest.zip -d GreengrassInstaller && rm greengrass-
   nucleus-latest.zip
   ```

   Windows Command Prompt (CMD)

   ```
   mkdir GreengrassInstaller && tar -xf greengrass-nucleus-latest.zip -
   C GreengrassInstaller && del greengrass-nucleus-latest.zip
   ```

   PowerShell

   ```
   Expand-Archive -Path greengrass-nucleus-latest.zip -DestinationPath .\
   \GreengrassInstaller
   rm greengrass-nucleus-latest.zip
   ```

4. (選擇性) 執行下列命令以查看 AWS IoT Greengrass Core 軟體的版本。

   ```
   java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
   ```

> **Important**
>
> 如果您安裝的 Greengrass 核心版本早於 v2.4.0，請不要在安裝核心軟體之後移除此資料夾。AWS IoT Greengrass AWS IoT Greengrass 核心軟件使用此文件夾中的文件運行。
> 如果您下載了最新版本的軟體，請安裝 v2.4.0 或更新版本，而且您可以在安裝 AWS IoT Greengrass Core 軟體之後移除此資料夾。

安裝 AWS IoT Greengrass 核心軟體

使用指定下列動作的引數執行安裝程式：

- 從指定使用您先前建立的 AWS 資源和憑證的部分組態檔進行安裝。 AWS IoT Greengrass 核心軟體使用組態檔案來指定裝置上每個 Greengrass 元件的組態。安裝程式會從您提供的部分組態檔案建立完整的組態檔案。
- 指定使用`ggc_user`系統使用者在核心裝置上執行軟體元件。在 Linux 裝置上，此指令也會指定使用`ggc_group`系統群組，而安裝程式會為您建立系統使用者和群組。
- 將 AWS IoT Greengrass Core 軟體設定為在開機時執行的系統服務。在 Linux 設備上，這需要系統初始化系統。

> **Important**
>
> 在 Windows 核心裝置上，您必須將 AWS IoT Greengrass 核心軟體設定為系統服務。

如需可指定之引數的詳細資訊，請參閱安裝器引數。

> **Note**
>
> 如果您在記憶體有限的裝置 AWS IoT Greengrass 上執行，您可以控制 AWS IoT Greengrass Core 軟體使用的記憶體數量。若要控制記憶體分配，您可以在核心元件的`jvmOptions`組態參數中設定 JVM 堆積大小選項。如需詳細資訊，請參閱 使用 JVM 選項控制內存分配。

- 如果您先前在 AWS IoT 服務中建立了物憑證和私密金鑰，請依照下列步驟使用私密金鑰和憑證檔案安裝 AWS IoT Greengrass Core 軟體。
- 如果您先前從硬體安全性模組 (HSM) 中的私密金鑰建立物憑證，請依照下列步驟在 HSM 中使用私密金鑰和憑證安裝 AWS IoT Greengrass Core 軟體。

使用私鑰和證書文件安裝 AWS IoT Greengrass Core 軟件

若要安裝 AWS IoT Greengrass 核心軟體

1. 檢查 AWS IoT Greengrass 核心軟體的版本。

   - 以包含軟體的資料夾路徑取*`GreengrassInstaller`*代。

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

2. 使用文字編輯器建立名為提供config.yaml給安裝程式的組態檔案。

   例如，在 Linux 系統上，您可以執行下列命令來使用 GNU nano 來建立檔案。

   ```
   nano GreengrassInstaller/config.yaml
   ```

   將下列 YAML 內容複製到檔案中。此部分配置文件指定系統參數和 Greengrass 核參數。

   ```
   ---
   system:
     certificateFilePath: "/greengrass/v2/device.pem.crt"
     privateKeyPath: "/greengrass/v2/private.pem.key"
     rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
     rootpath: "/greengrass/v2"
     thingName: "MyGreengrassCore"
   services:
     aws.greengrass.Nucleus:
       componentType: "NUCLEUS"
       version: "2.12.3"
       configuration:
         awsRegion: "us-west-2"
         iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
         iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
         iotCredEndpoint: "device-credentials-prefix.credentials.iot.us-
   west-2.amazonaws.com"
   ```

   然後，執行下列動作：

   - */greengrass/v2*以 Greengrass 根資料夾取代的每個執行個體。
   - *MyGreengrassCore*以 AWS IoT 物件的名稱取代。
   - 以 AWS IoT Greengrass 核心軟體的版本取代 *2.12.3*。
   - 將 *us-west-2* 取代為您建立資源的 AWS 區域 位置。
   - 取代*GreengrassCoreTokenExchangeRoleAlias*為權杖交換角色別名的名稱。
   - iotDataEndpoint以您的 AWS IoT 資料端點取代。
   - iotCredEndpoint以您的 AWS IoT 認證端點取代。

> **Note**
>
> 在此組態檔案中，您可以自訂其他核心組態選項，例如要使用的連接埠和網路 Proxy，如下列範例所示。如需詳細資訊，請參閱 Greengrass 核組態。

```
---
system:
  certificateFilePath: "/greengrass/v2/device.pem.crt"
  privateKeyPath: "/greengrass/v2/private.pem.key"
  rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
  rootpath: "/greengrass/v2"
  thingName: "MyGreengrassCore"
services:
  aws.greengrass.Nucleus:
    componentType: "NUCLEUS"
    version: "2.12.3"
    configuration:
      awsRegion: "us-west-2"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      iotCredEndpoint: "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
      mqtt:
        port: 443
      greengrassDataPlanePort: 443
      networkProxy:
        noProxyAddresses: "http://192.168.0.1,www.example.com"
        proxy:
          url: "https://my-proxy-server:1100"
          username: "Mary_Major"
          password: "pass@word1357"
```

3. 執行安裝程式，並指--init-config定提供組態檔案。

   - 使用 Greengrass 根資料夾取代*/greengrass/v2*或 *C:\greengrass\v2*。
   - 將的每個執行個*GreengrassInstaller*體取代為解壓縮安裝程式的資料夾。

Linux or Unix

```
sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \
  -jar ./GreengrassInstaller/lib/Greengrass.jar \
  --init-config ./GreengrassInstaller/config.yaml \
  --component-default-user ggc_user:ggc_group \
  --setup-system-service true
```

Windows Command Prompt (CMD)

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" ^
  -jar ./GreengrassInstaller/lib/Greengrass.jar ^
  --init-config ./GreengrassInstaller/config.yaml ^
  --component-default-user ggc_user ^
  --setup-system-service true
```

PowerShell

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" `
  -jar ./GreengrassInstaller/lib/Greengrass.jar `
  --init-config ./GreengrassInstaller/config.yaml `
  --component-default-user ggc_user `
  --setup-system-service true
```

Important

在 Windows 核心裝置上，您必須指`--setup-system-service true`定將 AWS IoT Greengrass 核心軟體設定為系統服務。

如果您指定`--setup-system-service true`，安裝程式會列印是`Successfully set up Nucleus as a system service`否將軟體設定為系統服務並執行。否則，如果安裝程式成功安裝軟體，則不會輸出任何訊息。

> **Note**
>
> 當您執行沒有`deploy-dev-tools`引數的安裝程式時，您無法使用`--provision true`引數來部署本機開發工具。如需直接在裝置上部署 Greengrass CLI 的相關資訊，請參閱。綠色命令行界面

4. 檢視根資料夾中的檔案，以確認安裝。

   Linux or Unix

   ```
   ls /greengrass/v2
   ```

   Windows Command Prompt (CMD)

   ```
   dir C:\greengrass\v2
   ```

   PowerShell

   ```
   ls C:\greengrass\v2
   ```

   如果安裝成功，則根資料夾會包含數個資料夾`config`，例如`packages`、和`logs`。

在 HSM 中使用私密金鑰和憑證安裝 AWS IoT Greengrass Core 軟體

> **Note**
>
> 此功能適用於 v2.5.3 及更高版 Greeng rass 核組件。 AWS IoT Greengrass 目前在 Windows 核心裝置上不支援此功能。

若要安裝 AWS IoT Greengrass 核心軟體

1. 檢查 AWS IoT Greengrass 核心軟體的版本。

   - 以包含軟體的資料夾路徑取*`GreengrassInstaller`*代。

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

2. 若要讓 AWS IoT Greengrass 核心軟體使用 HSM 中的私密金鑰和憑證，請在安裝 Core 軟體時安裝 PKCS #11 提供者元件。 AWS IoT Greengrass PKCS #11 提供者元件是您可以在安裝期間設定的外掛程式。您可以從下列位置下載最新版本的 PKCS #11 提供者元件：

   - https://d2s8p88vqu9w66.cloudfront.net/releases/Pkcs11Provider/aws.greengrass.crypto.Pkcs11Provider-latest.jar

   將 PKCS #11 提供者外掛程式下載至名為的檔案。`aws.greengrass.crypto.Pkcs11Provider.jar`以您要使用的資料夾取*`GreengrassInstaller`*代。

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/Pkcs11Provider/
   aws.greengrass.crypto.Pkcs11Provider-latest.jar > GreengrassInstaller/
   aws.greengrass.crypto.Pkcs11Provider.jar
   ```

   下載此軟體，即表示您同意 Greengrass 核心軟體授權合約之規定。

3. 使用文字編輯器建立名為提供`config.yaml`給安裝程式的組態檔案。

   例如，在 Linux 系統上，您可以執行下列命令來使用 GNU nano 來建立檔案。

   ```
   nano GreengrassInstaller/config.yaml
   ```

   將下列 YAML 內容複製到檔案中。此部分組態檔案會指定系統參數、Greengrass 核參數和 PKCS #11 提供者參數。

   ```
   ---
   system:
     certificateFilePath: "pkcs11:object=iotdevicekey;type=cert"
     privateKeyPath: "pkcs11:object=iotdevicekey;type=private"
     rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
     rootpath: "/greengrass/v2"
     thingName: "MyGreengrassCore"
   services:
     aws.greengrass.Nucleus:
       componentType: "NUCLEUS"
       version: "2.12.3"
   ```

```
    configuration:
      awsRegion: "us-west-2"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
      iotCredEndpoint: "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
  aws.greengrass.crypto.Pkcs11Provider:
    configuration:
      name: "softhsm_pkcs11"
      library: "/usr/local/Cellar/softhsm/2.6.1/lib/softhsm/libsofthsm2.so"
      slot: 1
      userPin: "1234"
```

然後，執行下列動作：

- 將 PKCS #11 URI 中的每個 *iotdevicekey* 執行個體取代為您建立私密金鑰並匯入憑證的物件標籤。
- */greengrass/v2*以 Greengrass 根資料夾取代的每個執行個體。
- *MyGreengrassCore*以 AWS IoT 物件的名稱取代。
- 以 AWS IoT Greengrass 核心軟體的版本取代 *2.12.3*。
- 將 *us-west-2* 取代為您建立資源的 AWS 區域 位置。
- 取代*GreengrassCoreTokenExchangeRoleAlias*為權杖交換角色別名的名稱。
- `iotDataEndpoint`以您的 AWS IoT 資料端點取代。
- `iotCredEndpoint`以您的 AWS IoT 認證端點取代。
- 將`aws.greengrass.crypto.Pkcs11Provider`元件的組態參數取代為核心裝置上 HSM 組態的值。

> **Note**
>
> 在此組態檔案中，您可以自訂其他核心組態選項，例如要使用的連接埠和網路 Proxy，如下列範例所示。如需詳細資訊，請參閱 Greengrass 核組態。
>
> ```
> ---
> system:
>   certificateFilePath: "pkcs11:object=iotdevicekey;type=cert"
>   privateKeyPath: "pkcs11:object=iotdevicekey;type=private"
>   rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
>   rootpath: "/greengrass/v2"
> ```

```
  thingName: "MyGreengrassCore"
services:
  aws.greengrass.Nucleus:
    componentType: "NUCLEUS"
    version: "2.12.3"
    configuration:
      awsRegion: "us-west-2"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
      iotCredEndpoint: "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
      mqtt:
        port: 443
      greengrassDataPlanePort: 443
      networkProxy:
        noProxyAddresses: "http://192.168.0.1,www.example.com"
        proxy:
          url: "https://my-proxy-server:1100"
          username: "Mary_Major"
          password: "pass@word1357"
  aws.greengrass.crypto.Pkcs11Provider:
    configuration:
      name: "softhsm_pkcs11"
      library: "/usr/local/Cellar/softhsm/2.6.1/lib/softhsm/libsofthsm2.so"
      slot: 1
      userPin: "1234"
```

4. 執行安裝程式，並指--init-config定提供組態檔案。

   - */greengrass/v2*以 Greengrass 色根資料夾取代。
   - 將的每個執行個*GreengrassInstaller*體取代為解壓縮安裝程式的資料夾。

```
sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \
  -jar ./GreengrassInstaller/lib/Greengrass.jar \
  --trusted-plugin ./GreengrassInstaller/aws.greengrass.crypto.Pkcs11Provider.jar \
  --init-config ./GreengrassInstaller/config.yaml \
  --component-default-user ggc_user:ggc_group \
  --setup-system-service true
```

> **Important**
>
> 在 Windows 核心裝置上，您必須指`--setup-system-service true`定將 AWS IoT Greengrass 核心軟體設定為系統服務。

如果您指定`--setup-system-service true`，安裝程式會列印是`Successfully set up Nucleus as a system service`否將軟體設定為系統服務並執行。否則，如果安裝程式成功安裝軟體，則不會輸出任何訊息。

> **Note**
>
> 當您執行沒有`deploy-dev-tools`引數的安裝程式時，您無法使用`--provision true`引數來部署本機開發工具。如需直接在裝置上部署 Greengrass CLI 的相關資訊，請參閱。綠色命令行界面

5. 檢視根資料夾中的檔案，以確認安裝。

   Linux or Unix

   ```
   ls /greengrass/v2
   ```

   Windows Command Prompt (CMD)

   ```
   dir C:\greengrass\v2
   ```

   PowerShell

   ```
   ls C:\greengrass\v2
   ```

   如果安裝成功，則根資料夾會包含數個資料夾`config`，例如`packages`、和`logs`。

如果您將 AWS IoT Greengrass Core 軟體安裝為系統服務，安裝程式會為您執行該軟體。否則，您必須手動運行該軟件。如需詳細資訊，請參閱 執行AWS IoT Greengrass核心軟體。

如需有關如何設定及使用軟體的詳細資訊 AWS IoT Greengrass，請參閱下列內容：

- 設定 AWS IoT Greengrass 核心軟體
- 開發AWS IoT Greengrass元件
- 將AWS IoT Greengrass元件部署到裝置
- 綠色命令行界面

透過 AWS IoT 叢集佈建安裝 AWS IoT Greengrass 核心軟體

此功能適用於 v2.4.0 和更高版 Greeng rass 核組件。

透過 AWS IoT 叢集佈建，您可以設定 AWS IoT 為在 X.509 裝置憑證和私密金鑰首次連線到裝置時產生並安全地傳遞至 AWS IoT 裝置。 AWS IoT 提供由 Amazon 根憑證授權單位 (CA) 簽署的用戶端憑證。您也可以配 AWS IoT 置為使用叢集佈建佈建的 Greengrass 核心裝置指定物件群組、物件類型和權限。您可以定義佈建範本，以定義 AWS IoT 佈建每個裝置的方式。佈建範本會指定佈建時要為設備建立的物件、原則和憑證資源。如需詳細資訊，請參閱AWS IoT Core 開發人員指南中的佈建範本。

AWS IoT Greengrass 提供 AWS IoT 叢集佈建外掛程式，您可以使用 AWS IoT 叢集佈建建立的 AWS 資源來安裝 AWS IoT Greengrass Core 軟體。車隊配置插件使用通過聲明進行配置。裝置使用佈建宣告憑證和私密金鑰來取得唯一的 X.509 裝置憑證和可用於一般作業的私密金鑰。您可以在製造過程中將索賠憑證和私密金鑰嵌入每個裝置，以便您的客戶稍後在每部裝置上線時啟用裝置。您可以在多個裝置上使用相同的宣告憑證和私密金鑰。如需詳細資訊，請參閱AWS IoT Core 開發人員指南中的透過宣告進行佈建。

> **Note**
>
> 叢集佈建外掛程式目前不支援在硬體安全模組 (HSM) 中儲存私密金鑰和憑證檔案。若要使用 HSM，請透過手動佈建來安裝 AWS IoT Greengrass 核心軟體。

若要透過 AWS IoT 叢集佈建來安裝 AWS IoT Greengrass Core 軟體，您必須在 AWS IoT 用於佈建 Greengrass 核心裝置的 AWS 帳戶 資源中設定資源。這些資源包括佈建範本、宣告憑證和權杖交換 IAM 角色。建立這些資源之後，您可以重複使用它們來佈建叢集中的多個核心裝置。如需詳細資訊，請參閱 為 Greengrass 核心裝置設定AWS IoT叢集佈建。

> **Important**
>
> 在您下載 AWS IoT Greengrass Core 軟體之前，請檢查您的核心裝置是否符合安裝和執行 AWS IoT Greengrass Core 軟體 2.0 的需求。

主題

必要條件

若要透過 AWS IoT 叢集佈建來安裝 AWS IoT Greengrass Core 軟體，您必須先為 Greengrass 核心裝置設定 AWS IoT 叢集佈建。完成這些步驟一次之後，您可以使用叢集佈建，在任意數量的裝置上安裝 AWS IoT Greengrass Core 軟體。

擷取 AWS IoT 端點

取得您的 AWS IoT 端點 AWS 帳戶，並儲存以供稍後使用。您的裝置會使用這些端點連線至 AWS IoT。請執行下列操作：

1. 取得適 AWS IoT 用於您的 AWS 帳戶.

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

   如果要求成功，回應看起來類似下列範例。

   ```
   {
     "endpointAddress": "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
   }
   ```

2. 取 AWS IoT 得您的 AWS 帳戶.

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

如果要求成功，回應看起來類似下列範例。

```
{
  "endpointAddress": "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
}
```

將憑證下載到裝置

裝置使用宣告憑證和私密金鑰來驗證其要求，以佈建 AWS 資源並取得 X.509 裝置憑證。您可以在製造過程中將宣告憑證和私密金鑰嵌入裝置，或在安裝期間將憑證和金鑰複製到裝置。在本節中，您會將宣告憑證和私密金鑰複製到裝置。您也會將 Amazon 根憑證授權單位 (CA) 憑證下載到裝置。

> **Important**
>
> 佈建宣告私密金鑰應始終受到保護，包括在 Greengrass 核心裝置上。我們建議您使用 Amazon CloudWatch 指標和日誌來監控濫用的跡象，例如未經授權使用宣告憑證來佈建裝置。如果您偵測到誤用，請停用佈建宣告憑證，使其無法用於裝置佈建。如需詳細資訊，請參閱AWS IoT Core 開發人員指南 AWS IoT中的監控。
>
> 為了協助您更有效地管理在您的裝置數目以及自行註冊的裝置數目 AWS 帳戶，您可以在建立叢集佈建範本時指定預先佈建勾點。預先佈建勾點是一種 AWS Lambda 函數，可驗證裝置在註冊期間提供的範本參數。例如，您可以建立預先佈建勾點，以根據資料庫檢查裝置 ID，以確認裝置是否具有佈建權限。如需詳細資訊，請參閱AWS IoT Core 開發人員指南中的預先佈建勾點。

將宣告憑證下載至裝置

1. 將宣告憑證和私密金鑰複製到裝置。如果在開發電腦和裝置上啟用了 SSH 和 SCP，您可以使用開發電腦上的`scp`命令來傳輸宣告憑證和私密金鑰。下列範例命令會將開發電腦`claim-certs`上名為的資料夾傳輸這些檔案至裝置。*`device-ip-address`*替換為設備的 IP 地址。

   ```
   scp -r claim-certs/ device-ip-address:~
   ```

2. 在設備上創建綠色根文件夾。您稍後將 AWS IoT Greengrass 核心軟體安裝到此資料夾中。

Linux or Unix

- 以要使*/greengrass/v2*用的資料夾取代。

```
sudo mkdir -p /greengrass/v2
```

Windows Command Prompt

- 將 *C:\greengrass\v2* 取代為要使用的資料夾。

```
mkdir C:\greengrass\v2
```

PowerShell

- 將 *C:\greengrass\v2* 取代為要使用的資料夾。

```
mkdir C:\greengrass\v2
```

3. (僅限 Linux) 設定 Greengrass 根資料夾的父系權限。

   - 將 */greengrass* 替換為根文件夾的父文件夾。

   ```
   sudo chmod 755 /greengrass
   ```

4. 將宣告憑證移至 Greengrass 根資料夾。

   - 使用 Greengrass 根資料夾取代*/greengrass/v2*或 *C:\greengrass\v2*。

   Linux or Unix

   ```
   sudo mv ~/claim-certs /greengrass/v2
   ```

   Windows Command Prompt (CMD)

   ```
   move %USERPROFILE%\claim-certs C:\greengrass\v2
   ```

PowerShell

```
mv -Path ~\claim-certs -Destination C:\greengrass\v2
```

5. 下載 Amazon 根憑證授權單位 (CA) 憑證。 AWS IoT 憑證預設會與 Amazon 的根 CA 憑證相關聯。

   Linux or Unix

   ```
   sudo curl -o /greengrass/v2/AmazonRootCA1.pem https://www.amazontrust.com/repository/AmazonRootCA1.pem
   ```

   Windows Command Prompt (CMD)

   ```
   curl -o C:\greengrass\v2\\AmazonRootCA1.pem https://www.amazontrust.com/repository/AmazonRootCA1.pem
   ```

   PowerShell

   ```
   iwr -Uri https://www.amazontrust.com/repository/AmazonRootCA1.pem -OutFile C:\greengrass\v2\\AmazonRootCA1.pem
   ```

設定裝置環境

請依照本節中的步驟設定 Linux 或 Windows 裝置作為 AWS IoT Greengrass 核心裝置使用。

設定裝置

若要設定下列項目的 Linux 設備 AWS IoT Greengrass V2

1. 安裝 Java 執行階段， AWS IoT Greengrass 核心軟體需要執行。我們建議您使用 Amazon Corretto 或 OpenJDK 長期支持版本。需要版本 8 或更高版本。以下命令向您展示如何在設備上安裝 OpenJDK。

   - 針對 Debian 為基礎或 Ubuntu 為基礎的發行版本：

     ```
     sudo apt install default-jdk
     ```

   - 針對 Red Hat 為基礎的發行版本：

```
sudo yum install java-11-openjdk-devel
```

- 針對 Amazon Linux 2：

```
sudo amazon-linux-extras install java-openjdk11
```

- 對於 Amazon

```
sudo dnf install java-11-amazon-corretto -y
```

安裝完成時，請執行下列命令以確認 Java 是否在 Linux 裝置上執行。

```
java -version
```

此指令會列印在裝置上執行的 Java 版本。例如，在基於 Debian 的發行版上，輸出可能類似於以下示例。

```
openjdk version "11.0.9.1" 2020-11-04
OpenJDK Runtime Environment (build 11.0.9.1+1-post-Debian-1deb10u2)
OpenJDK 64-Bit Server VM (build 11.0.9.1+1-post-Debian-1deb10u2, mixed mode)
```

2. (選擇性) 建立在裝置上執行元件的預設系統使用者和群組。您也可以選擇讓 AWS IoT Greengrass Core 軟體安裝程式在安裝期間使用安裝程式引數建立此使用者和群組。`--component-default-user`如需詳細資訊，請參閱 安裝器引數。

```
sudo useradd --system --create-home ggc_user
sudo groupadd --system ggc_group
```

3. 確認執行 AWS IoT Greengrass Core 軟體的使用者 (通常`root`) 具有`sudo`與任何使用者和任何群組一起執行的權限。

   a. 執行下列命令以開啟`/etc/sudoers`檔案。

   ```
   sudo visudo
   ```

   b. 確認使用者的權限看起來像下列範例。

   ```
   root    ALL=(ALL:ALL) ALL
   ```

4. (選擇性) 若要執行容器化 Lambda 函數，您必須啟用 cgroup v1，並且必須啟用並掛接記憶體和裝置 cgroup。如果您不打算執行容器化 Lambda 函數，則可以略過此步驟。

   若要啟用這些 cgroups 選項，請使用下列 Linux 核心參數來啟動裝置。

   ```
   cgroup_enable=memory cgroup_memory=1 systemd.unified_cgroup_hierarchy=0
   ```

   如需有關檢視和設定裝置核心參數的資訊，請參閱作業系統和開機載入程式的說明文件。依照指示永久設定核心參數。
5. 依照中的需求清單所指示，在您的裝置上安裝所有其他必要的相依性裝置要求。

設定視窗裝置

> **Note**
>
> 此功能適用於 v2.5.0 及更高版 Greeng rass 核組件。

若要設定下列項目的視窗裝置 AWS IoT Greengrass V2

1. 安裝 Java 執行階段， AWS IoT Greengrass 核心軟體需要執行。我們建議您使用 Amazon Corretto 或 OpenJDK 長期支持版本。需要版本 8 或更高版本。
2. 檢查系統變數 PATH 上是否可用 Java，如果沒有，則加入它。此 LocalSystem 帳戶會執行 AWS IoT Greengrass 核心軟體，因此您必須將 Java 新增至系統變數 PATH，而非使用者的 PATH 使用者變數。請執行下列操作：

   a. 按視窗鍵開啟開始功能表。
   b. 鍵入**environment variables**以從開始功能表搜尋系統選項。
   c. 在開始功能表搜尋結果中，選擇 [編輯系統環境變數] 以開啟 [系統屬性] 視窗。
   d. 選擇環境變數... 以開啟 [環境變數] 視窗。
   e. 在 [系統變數] 下，選取 [路徑]，然後選擇 [編輯]。在 「編輯」環境變數視窗中，您可以在不同的行上檢視每個路徑。
   f. 檢查 Java 安裝`bin`資料夾的路徑是否存在。路徑看起來可能類似下列範例。

      ```
      C:\\Program Files\\Amazon Corretto\\jdk11.0.13_8\\bin
      ```

   g. 如果路徑中缺少 Java 安裝的`bin`資料夾，請選擇 [新增] 加入，然後選擇 [確定]。

3. 以系統管理員身分開啟 Windows 命令提示字元 (cmd.exe)。
4. 在 Windows 裝置上的 LocalSystem 帳戶中建立預設使用者。以安全*##*取代密碼。

```
net user /add ggc_user password
```

> **Tip**
>
> 視您的 Windows 組態而定，使用者的密碼可能會設定為在 future 的某個日期到期。為確保您的 Greengrass 應用程式繼續運作，請追蹤密碼何時到期，並在密碼到期之前進行更新。您也可以將使用者的密碼設定為永不過期。
>
> - 若要檢查使用者及其密碼何時到期，請執行下列命令。
>
>   ```
>   net user ggc_user | findstr /C:expires
>   ```
>
> - 若要將使用者的密碼設定為永不過期，請執行下列命令。
>
>   ```
>   wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
>   ```
>
> - 如果您使用的是 Windows 10 或更新版本，而該wmic命令已被取代，請執行下列 PowerShell 命令。
>
>   ```
>   Get-CimInstance -Query "SELECT * from Win32_UserAccount WHERE name =
>    'ggc_user'" | Set-CimInstance -Property @{PasswordExpires="False"}
>   ```

5. 在設備上從 Microsoft 下載並安裝該PsExec實用程序。
6. 使用此 PsExec 公用程式將預設使用者的使用者名稱和密碼儲存在 LocalSystem 帳戶的認證管理員執行個體中。*################*

```
psexec -s cmd /c cmdkey /generic:ggc_user /user:ggc_user /pass:password
```

如果打PsExec License Agreement開，請選Accept擇同意許可證並運行命令。

> **Note**
>
> 在 Windows 裝置上， LocalSystem 帳戶會執行 Greengrass 核心，而且您必須使用公用 PsExec 程式將預設使用者資訊儲存在帳戶中。 LocalSystem 使用認證管理員應用程式會將此資訊儲存在目前登入使用者的 Windows 帳戶中，而非 LocalSystem 帳戶中。

下載 AWS IoT Greengrass 核心軟體

您可以從下列位置下載最新版本的 AWS IoT Greengrass Core 軟體：

- https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest. 壓縮

> **Note**
>
> 您可以從下列位置下載特定版本的 AWS IoT Greengrass Core 軟體。將*##*取代為要下載的版本。
>
> ```
> https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
> ```

若要下載 AWS IoT Greengrass 核心軟體

1. 在您的核心裝置上，將 AWS IoT Greengrass Core 軟體下載至名為`greengrass-nucleus-latest.zip`.

   Linux or Unix

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip > greengrass-nucleus-latest.zip
   ```

   Windows Command Prompt (CMD)

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip > greengrass-nucleus-latest.zip
   ```

PowerShell

```
iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip -OutFile greengrass-nucleus-latest.zip
```

下載此軟體，即表示您同意 Greengrass 核心軟體授權合約之規定。

2. (選擇性) 驗證 Greengrass 核軟體簽章

> **Note**
>
> 此功能適用於 Greengrass 核 2.9.5 及更高版本。

a. 使用以下命令來驗證 Greengrass 核工件的簽名：

Linux or Unix

```
jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

根據您安裝的 JDK 版本，檔案名稱可能會有所不同。以您安裝的 JDK 版本取代*jdk17.0.6_10*。

```
"C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -verify -certs -verbose greengrass-nucleus-latest.zip
```

PowerShell

根據您安裝的 JDK 版本，檔案名稱可能會有所不同。以您安裝的 JDK 版本取代*jdk17.0.6_10*。

```
'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -verify -certs -verbose greengrass-nucleus-latest.zip
```

b. `jarsigner`調用產生指示驗證結果的輸出。

i. 如果 Greengrass 核 zip 文件簽名，則輸出包含以下語句：

```
jar verified.
```

ii. 如果 Greengrass 核壓縮檔未簽署，則輸出會包含下列陳述式：

```
jar is unsigned.
```

c. 如果您提供了 Jarsigner `-certs -verbose` 選項`-verify`和選項，則輸出也會包含詳細的簽署者憑證資訊。

3. 將 AWS IoT Greengrass Core 軟件解壓縮到設備上的文件夾。以您要使用的資料夾取*GreengrassInstaller*代。

Linux or Unix

```
unzip greengrass-nucleus-latest.zip -d GreengrassInstaller && rm greengrass-
nucleus-latest.zip
```

Windows Command Prompt (CMD)

```
mkdir GreengrassInstaller && tar -xf greengrass-nucleus-latest.zip -
C GreengrassInstaller && del greengrass-nucleus-latest.zip
```

PowerShell

```
Expand-Archive -Path greengrass-nucleus-latest.zip -DestinationPath .\
\GreengrassInstaller
rm greengrass-nucleus-latest.zip
```

4. (選擇性) 執行下列命令以查看 AWS IoT Greengrass Core 軟體的版本。

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

Important

如果您安裝的 Greengrass 核心版本早於 v2.4.0，請不要在安裝核心軟體之後移除此資料夾。AWS IoT Greengrass AWS IoT Greengrass 核心軟件使用此文件夾中的文件運行。
如果您下載了最新版本的軟體，請安裝 v2.4.0 或更新版本，而且您可以在安裝 AWS IoT Greengrass Core 軟體之後移除此資料夾。

下載 AWS IoT 車隊佈建外掛程式

您可以從以下位置下載最新版本的 AWS IoT 叢集佈建外掛程式：

- https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-FleetProvisioningByClaim / fleetprovisioningbyclaim-latest.jar

> **Note**
>
> 您可以從以下位置下載特定版本的 AWS IoT 叢集佈建外掛程式。將*##*取代為要下載的版本。如需叢集佈建外掛程式每個版本的詳細資訊，請參閱AWS IoT 車隊配置插件更新日誌。
>
> ```
> https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-
> FleetProvisioningByClaim/fleetprovisioningbyclaim-version.jar
> ```

車隊配置插件是開源的。若要檢視其原始程式碼，請參閱上的AWS IoT 叢集佈建外掛程式 GitHub。

若要下載 AWS IoT 叢集佈建外掛程式

- 在您的裝置上，將 AWS IoT 叢集佈建外掛程式下載到名為的檔案`aws.greengrass.FleetProvisioningByClaim.jar`。以您要使用的資料夾取*`GreengrassInstaller`*代。

  Linux or Unix

  ```
  curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-
  FleetProvisioningByClaim/fleetprovisioningbyclaim-latest.jar
   > GreengrassInstaller/aws.greengrass.FleetProvisioningByClaim.jar
  ```

  Windows Command Prompt (CMD)

  ```
  curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-
  FleetProvisioningByClaim/fleetprovisioningbyclaim-latest.jar
   > GreengrassInstaller/aws.greengrass.FleetProvisioningByClaim.jar
  ```

PowerShell

```
iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-
FleetProvisioningByClaim/fleetprovisioningbyclaim-latest.jar -
OutFile GreengrassInstaller/aws.greengrass.FleetProvisioningByClaim.jar
```

下載此軟體，即表示您同意 Greengrass 核心軟體授權合約之規定。

安裝 AWS IoT Greengrass 核心軟體

使用指定下列動作的引數執行安裝程式：

- 從指定使用叢集佈建外掛程式佈建 AWS 資源的部分組態檔進行安裝。 AWS IoT Greengrass 核心軟體使用組態檔案來指定裝置上每個 Greengrass 元件的組態。安裝程式會從您提供的部分組態檔案以及叢集佈建外掛程式所建立的 AWS 資源，建立完整的組態檔案。
- 指定使用`ggc_user`系統使用者在核心裝置上執行軟體元件。在 Linux 裝置上，此指令也會指定使用`ggc_group`系統群組，而安裝程式會為您建立系統使用者和群組。
- 將 AWS IoT Greengrass Core 軟體設定為在開機時執行的系統服務。在 Linux 設備上，這需要系統初始化系統。

> **Important**
>
> 在 Windows 核心裝置上，您必須將 AWS IoT Greengrass 核心軟體設定為系統服務。

如需可指定之引數的詳細資訊，請參閱安裝器引數。

> **Note**
>
> 如果您在記憶體有限的裝置 AWS IoT Greengrass 上執行，您可以控制 AWS IoT Greengrass Core 軟體使用的記憶體數量。若要控制記憶體分配，您可以在核心元件的`jvmOptions`組態參數中設定 JVM 堆積大小選項。如需詳細資訊，請參閱 使用 JVM 選項控制內存分配。

若要安裝 AWS IoT Greengrass 核心軟體

1. 檢查 AWS IoT Greengrass 核心軟體的版本。

- 以包含軟體的資料夾路徑取*GreengrassInstaller*代。

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

2. 使用文字編輯器建立名為提供config.yaml給安裝程式的組態檔案。

   例如，在 Linux 系統上，您可以執行下列命令來使用 GNU nano 建立檔案。

   ```
   nano GreengrassInstaller/config.yaml
   ```

   將下列 YAML 內容複製到檔案中。此部分組態檔案會指定叢集佈建外掛程式的參數。若要取得有關可指定之選項的更多資訊，請參閱〈〉設定AWS IoT叢集佈建外掛程式。

   Linux or Unix

   ```
   ---
   services:
     aws.greengrass.Nucleus:
       version: "2.12.3"
     aws.greengrass.FleetProvisioningByClaim:
       configuration:
         rootPath: "/greengrass/v2"
         awsRegion: "us-west-2"
         iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
         iotCredentialEndpoint: "device-credentials-prefix.credentials.iot.us-
   west-2.amazonaws.com"
         iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
         provisioningTemplate: "GreengrassFleetProvisioningTemplate"
         claimCertificatePath: "/greengrass/v2/claim-certs/claim.pem.crt"
         claimCertificatePrivateKeyPath: "/greengrass/v2/claim-certs/
   claim.private.pem.key"
         rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
         templateParameters:
           ThingName: "MyGreengrassCore"
           ThingGroupName: "MyGreengrassCoreGroup"
   ```

   Windows

   ```
   ---
   services:
   ```

```
  aws.greengrass.Nucleus:
    version: "2.12.3"
  aws.greengrass.FleetProvisioningByClaim:
    configuration:
      rootPath: "C:\\greengrass\\v2"
      awsRegion: "us-west-2"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
      iotCredentialEndpoint: "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      provisioningTemplate: "GreengrassFleetProvisioningTemplate"
      claimCertificatePath: "C:\\greengrass\\v2\\claim-certs\\claim.pem.crt"
      claimCertificatePrivateKeyPath: "C:\\greengrass\\v2\\claim-certs\
\claim.private.pem.key"
      rootCaPath: "C:\\greengrass\\v2\\AmazonRootCA1.pem"
      templateParameters:
        ThingName: "MyGreengrassCore"
        ThingGroupName: "MyGreengrassCoreGroup"
```

然後，執行下列動作：

- 將 *2.12.3* 取代為 AWS IoT Greengrass 核心軟體的版本。
- 將*/greengrass/v2*或 *C:\greengrass\v2* 的每個執行個體取代為 Greengrass 根資料夾。

  > **Note**
  >
  > 在 Windows 裝置上，您必須將路徑分隔符號指定為雙反斜線 (\\)，例如。C:\\greengrass\\v2

- 將 *us-west-2* 取代為您建立佈建範本和其他資源的 AWS 區域。
- `iotDataEndpoint`以您的 AWS IoT 資料端點取代。
- `iotCredentialEndpoint`以您的 AWS IoT 認證端點取代。
- 取代*GreengrassCoreTokenExchangeRoleAlias*為權杖交換角色別名的名稱。
- *GreengrassFleetProvisioningTemplate*以叢集佈建範本的名稱取代。
- 以裝置上宣告憑證的路徑取代。`claimCertificatePath`
- 取代為`claimCertificatePrivateKeyPath`裝置上宣告憑證私密金鑰的路徑。
- 將範本參數 (`templateParameters`) 取代為用來佈建裝置的值。這個例子是指定義`ThingName`和`ThingGroupName`參數的示例模板。

Note

在此組態檔案中，您可以自訂其他組態選項，例如要使用的連接埠和網路 Proxy，如下列範例所示。如需詳細資訊，請參閱 Greengrass 核組態。

Linux or Unix

```
---
services:
  aws.greengrass.Nucleus:
    version: "2.12.3"
    configuration:
      mqtt:
        port: 443
      greengrassDataPlanePort: 443
      networkProxy:
        noProxyAddresses: "http://192.168.0.1,www.example.com"
        proxy:
          url: "http://my-proxy-server:1100"
          username: "Mary_Major"
          password: "pass@word1357"
  aws.greengrass.FleetProvisioningByClaim:
    configuration:
      rootPath: "/greengrass/v2"
      awsRegion: "us-west-2"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-
west-2.amazonaws.com"
      iotCredentialEndpoint: "device-credentials-
prefix.credentials.iot.us-west-2.amazonaws.com"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      provisioningTemplate: "GreengrassFleetProvisioningTemplate"
      claimCertificatePath: "/greengrass/v2/claim-certs/claim.pem.crt"
      claimCertificatePrivateKeyPath: "/greengrass/v2/claim-certs/
claim.private.pem.key"
      rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
      templateParameters:
        ThingName: "MyGreengrassCore"
        ThingGroupName: "MyGreengrassCoreGroup"
      mqttPort: 443
      proxyUrl: "http://my-proxy-server:1100"
      proxyUserName: "Mary_Major"
```

```
      proxyPassword: "pass@word1357"
```

Windows

```
---
services:
  aws.greengrass.Nucleus:
    version: "2.12.3"
    configuration:
      mqtt:
        port: 443
      greengrassDataPlanePort: 443
      networkProxy:
        noProxyAddresses: "http://192.168.0.1,www.example.com"
        proxy:
          url: "http://my-proxy-server:1100"
          username: "Mary_Major"
          password: "pass@word1357"
  aws.greengrass.FleetProvisioningByClaim:
    configuration:
      rootPath: "C:\\greengrass\\v2"
      awsRegion: "us-west-2"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-
west-2.amazonaws.com"
      iotCredentialEndpoint: "device-credentials-
prefix.credentials.iot.us-west-2.amazonaws.com"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      provisioningTemplate: "GreengrassFleetProvisioningTemplate"
      claimCertificatePath: "C:\\greengrass\\v2\\claim-certs\
\claim.pem.crt"
      claimCertificatePrivateKeyPath: "C:\\greengrass\\v2\\claim-certs\
\claim.private.pem.key"
      rootCaPath: "C:\\greengrass\\v2\\AmazonRootCA1.pem"
      templateParameters:
        ThingName: "MyGreengrassCore"
        ThingGroupName: "MyGreengrassCoreGroup"
      mqttPort: 443
      proxyUrl: "http://my-proxy-server:1100"
      proxyUserName: "Mary_Major"
      proxyPassword: "pass@word1357"
```

若要使用 HTTPS 代理伺服器，您必須使用叢集佈建外掛程式的 1.1.0 版或更新版本。您必須另外指定`rootCaPath`下的`system`，如下列範例所示。

Linux or Unix

```
---
system:
  rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
services:
  ...
```

Windows

```
---
system:
  rootCaPath: "C:\\greengrass\\v2\\AmazonRootCA1.pem"
services:
  ...
```

3. 執行安裝程式。指定`--trusted-plugin`提供叢集佈建外掛程式，並指`--init-config`定提供組態檔案。

   - *`/greengrass/v2`*以 Greengrass 色根資料夾取代。
   - 將的每個執行個*`GreengrassInstaller`*體取代為解壓縮安裝程式的資料夾。

   Linux or Unix

   ```
   sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \
     -jar ./GreengrassInstaller/lib/Greengrass.jar \
     --trusted-plugin ./GreengrassInstaller/
   aws.greengrass.FleetProvisioningByClaim.jar \
     --init-config ./GreengrassInstaller/config.yaml \
     --component-default-user ggc_user:ggc_group \
     --setup-system-service true
   ```

   Windows Command Prompt (CMD)

   ```
   java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" ^
   ```

```
  -jar ./GreengrassInstaller/lib/Greengrass.jar ^
  --trusted-plugin ./GreengrassInstaller/
aws.greengrass.FleetProvisioningByClaim.jar ^
  --init-config ./GreengrassInstaller/config.yaml ^
  --component-default-user ggc_user ^
  --setup-system-service true
```

PowerShell

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" `
  -jar ./GreengrassInstaller/lib/Greengrass.jar `
  --trusted-plugin ./GreengrassInstaller/
aws.greengrass.FleetProvisioningByClaim.jar `
  --init-config ./GreengrassInstaller/config.yaml `
  --component-default-user ggc_user `
  --setup-system-service true
```

Important

在 Windows 核心裝置上，您必須指`--setup-system-service true`定將 AWS IoT Greengrass 核心軟體設定為系統服務。

如果您指定`--setup-system-service true`，安裝程式會列印是`Successfully set up Nucleus as a system service`否將軟體設定為系統服務並執行。否則，如果安裝程式成功安裝軟體，則不會輸出任何訊息。

Note

當您執行沒有`deploy-dev-tools`引數的安裝程式時，您無法使用`--provision true`引數來部署本機開發工具。如需直接在裝置上部署 Greengrass CLI 的相關資訊，請參閱。綠色命令行界面

4. 檢視根資料夾中的檔案來確認安裝。

Linux or Unix

```
ls /greengrass/v2
```

Windows Command Prompt (CMD)

```
dir C:\greengrass\v2
```

PowerShell

```
ls C:\greengrass\v2
```

如果安裝成功，則根資料夾會包含數個資料夾`config`，例如`packages`、和`logs`。

如果您將 AWS IoT Greengrass Core 軟體安裝為系統服務，安裝程式會為您執行該軟體。否則，您必須手動運行該軟件。如需詳細資訊，請參閱 執行AWS IoT Greengrass核心軟體。

如需有關如何設定及使用軟體的詳細資訊 AWS IoT Greengrass，請參閱下列內容：

- 設定 AWS IoT Greengrass 核心軟體
- 開發AWS IoT Greengrass元件
- 將AWS IoT Greengrass元件部署到裝置
- 綠色命令行界面

為 Greengrass 核心裝置設定AWS IoT叢集佈建

若要透過叢集佈建來安裝 AWS IoT Greengrass Core 軟體，您必須先在AWS 帳戶. 這些資源可讓裝置自行註冊，AWS IoT並以 Greengrass 核心裝置的形式運作。按照本節中的步驟進行一次，在AWS 帳戶.

- 權杖交換 IAM 角色，核心裝置用來授權對AWS服務的呼叫。
- 指向權杖交換AWS IoT角色的角色別名。
- (選擇性) AWS IoT 原則，核心裝置使用此原則來授權AWS IoT和AWS IoT Greengrass服務的呼叫。此AWS IoT原則必須允`iot:AssumeRoleWithCertificate`許指向 Token 交換AWS IoT角色之角色別名的權限。

  您可以AWS IoT針對叢集中的所有核心裝置使用單一原則，也可以設定叢集佈建範本，為每個核心裝置建立AWS IoT原則。
- AWS IoT叢集佈建範本。此範本必須指定下列項目：

  - 物AWS IoT件資源。您可以指定現有物件群組的清單，以便在每個裝置上線時將元件部署到其中。
  - AWS IoT策略資源。此資源可以定義下列其中一個屬性：
    - 現有AWS IoT策略的名稱。如果您選擇此選項，您從此範本建立的核心裝置會使用相同的AWS IoT原則，而且您可以管理其作為叢集的權限。
    - 一份AWS IoT政策文件。如果您選擇此選項，您從此範本建立的每個核心裝置都會使用唯一的AWS IoT原則，而且您可以管理每個個別核心裝置的權限。
  - 憑AWS IoT證資源。此憑證資源必須使用`AWS::IoT::Certificate::Id`參數將憑證附加至核心裝置。如需詳細資訊，請參閱AWS IoT開發人員指南中的 J ust-in-time 佈建。
- 叢集AWS IoT佈建範本的佈建宣告憑證和私密金鑰。您可以在製造過程中將此憑證和私密金鑰嵌入裝置中，以便裝置在上線時可以自行註冊和佈建。

  > **Important**
  >
  > 佈建宣告私密金鑰應始終受到保護，包括在 Greengrass 核心裝置上。我們建議您使用 Amazon CloudWatch 指標和日誌來監控濫用的跡象，例如未經授權使用宣告憑證來佈建裝置。如果您偵測到誤用，請停用佈建宣告憑證，使其無法用於裝置佈建。如需詳細資訊，請參閱AWS IoT Core開發人員指南AWS IoT中的監控。
  >
  > 為了協助您更有效地管理在您的裝置數目以及自行註冊的裝置數目AWS 帳戶，您可以在建立叢集佈建範本時指定預先佈建勾點。預先佈建勾點是一種AWS Lambda函數，可驗證裝置在註冊期間提供的範本參數。例如，您可以建立預先佈建勾點，以根據資料庫檢查裝置 ID，以確認裝置是否具有佈建權限。如需詳細資訊，請參閱AWS IoT Core開發人員指南中的預先佈建勾點。

- 您附加至佈建宣告憑證的AWS IoT原則，以允許裝置註冊並使用叢集佈建範本。

主題

- 建立權杖交換角色
- 建立 AWS IoT 政策
- 建立叢集佈建範本
- 建立佈建宣告憑證和私密金鑰

建立權杖交換角色

Greengrass 核心裝置使用 IAM 服務角色 (稱為權杖交換角色) 來授權對服務的呼叫。AWS裝置使用AWS IoT登入資料提供者取得此角色的臨時AWS登入資料，以便裝置與之互動AWS IoT、將日誌傳送

到 Amazon CloudWatch Logs，以及從 Amazon S3 下載自訂元件成品。如需詳細資訊，請參閱 授權核心設備與AWS服務。

您可以使用AWS IoT角色別名來設定 Greengrass 核心裝置的權杖交換角色。角色別名可讓您變更裝置的 Token 交換角色，但保持裝置設定不變。如需詳細資訊，請參閱AWS IoT Core開發人員指南中的授權直接呼叫AWS服務。

在本節中，您會建立權杖交換 IAM 角色和指向該AWS IoT角色的角色別名。如果您已經設置了 Greengrass 核心設備，則可以使用其令牌交換角色和角色別名，而不是創建新的。

若要建立權杖交換 IAM 角色

1. 建立 IAM 角色，您的裝置可用作權杖交換角色。請執行下列操作：

   a. 建立包含 Token 交換角色所需之信任原則文件的檔案。

      例如，在 Linux 系統上，您可以執行下列命令來使用 GNU nano 建立檔案。

      ```
      nano device-role-trust-policy.json
      ```

      將下列 JSON 複製到檔案中。

      ```
      {
        "Version": "2012-10-17",
        "Statement": [
          {
            "Effect": "Allow",
            "Principal": {
              "Service": "credentials.iot.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
          }
        ]
      }
      ```

   b. 使用信任原則文件建立權杖交換角色。

      - TokenExchangeRole以要建立的身分與存取權管理員角色的名稱取代 *GreenGrassv2*。

```
aws iam create-role --role-name GreengrassV2TokenExchangeRole --assume-role-
policy-document file://device-role-trust-policy.json
```

如果要求成功，回應看起來類似下列範例。

```
{
  "Role": {
    "Path": "/",
    "RoleName": "GreengrassV2TokenExchangeRole",
    "RoleId": "AROAZ2YMUHYHK5OKM77FB",
    "Arn": "arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole",
    "CreateDate": "2021-02-06T00:13:29+00:00",
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": "credentials.iot.amazonaws.com"
          },
          "Action": "sts:AssumeRole"
        }
      ]
    }
  }
```

c. 建立包含權杖交換角色所需之存取原則文件的檔案。

例如，在 Linux 系統上，您可以執行下列命令來使用 GNU nano 建立檔案。

```
nano device-role-access-policy.json
```

將下列 JSON 複製到檔案中。

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

```
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    }
  ]
}
```

> **Note**
>
> 此存取政策不允許存取 S3 儲存貯體中的元件成品。若要在 Amazon S3 中部署定義成品的自訂元件，您必須向角色新增許可，以允許核心裝置擷取元件成品。如需詳細資訊，請參閱 允許存取 S3 儲存貯體的存取。
> 如果您還沒有適用於元件成品的 S3 儲存貯體，您可以在稍後建立儲存貯體後新增這些許可。

d. 從政策文件建立 IAM 政策。

- TokenExchangeRoleAccess以要建立的身分與存取權管理政策的名稱取代 *GreenGrassv2*。

```
aws iam create-policy --policy-name GreengrassV2TokenExchangeRoleAccess --
policy-document file://device-role-access-policy.json
```

如果要求成功，回應看起來類似下列範例。

```
{
  "Policy": {
    "PolicyName": "GreengrassV2TokenExchangeRoleAccess",
    "PolicyId": "ANPAZ2YMUHYHACI7C5Z66",
    "Arn": "arn:aws:iam::123456789012:policy/
GreengrassV2TokenExchangeRoleAccess",
    "Path": "/",
    "DefaultVersionId": "v1",
    "AttachmentCount": 0,
    "PermissionsBoundaryUsageCount": 0,
    "IsAttachable": true,
```

```
    "CreateDate": "2021-02-06T00:37:17+00:00",
    "UpdateDate": "2021-02-06T00:37:17+00:00"
  }
}
```

e. 將 IAM 政策附加到權杖交換角色。

- 以 IAM 角*#######TokenExchangeRole# V2*。
- 將政策 ARN 取代為您在上一個步驟中建立的 IAM 政策的 ARN。

```
aws iam attach-role-policy --role-name GreengrassV2TokenExchangeRole --policy-
arn arn:aws:iam::123456789012:policy/GreengrassV2TokenExchangeRoleAccess
```

如果請求成功，命令沒有任何輸出。

2. 建立指向權杖交換AWS IoT角色的角色別名。

- 以要建立之角色別名的名稱取*GreengrassCoreTokenExchangeRoleAlias*代。
- 將角色 ARN 取代為您在上一個步驟中建立的 IAM 角色的 ARN。

```
aws iot create-role-alias --role-alias GreengrassCoreTokenExchangeRoleAlias --role-
arn arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole
```

如果要求成功，回應看起來類似下列範例。

```
{
  "roleAlias": "GreengrassCoreTokenExchangeRoleAlias",
  "roleAliasArn": "arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias"
}
```

> **Note**
>
> 若要建立角色別名，您必須擁有權限才能將權杖交換 IAM 角色傳遞給AWS IoT。如果您在嘗試建立角色別名時收到錯誤訊息，請檢查您的AWS使用者是否具有此權限。如需詳細資訊，請參閱《使用指南》中的授與使用AWS Identity and Access Management者將角色傳遞給AWS服務的權限。

建立 AWS IoT 政策

將裝置註冊為AWS IoT物件後，該裝置即可使用數位憑證進行驗證AWS。此憑證包含一或多個AWS IoT原則，用來定義裝置可與憑證搭配使用的權限。這些原則可讓裝置與AWS IoT和通訊AWS IoT Greengrass。

透過AWS IoT叢集佈建，裝置會連線AWS IoT到以建立和下載裝置憑證。在您在下一節中建立的叢集佈建範本中，您可以指定是否AWS IoT將相同的AWS IoT原則附加至所有裝置的憑證，還是為每個裝置建立新原則。

在本節中，您會建立AWS IoT附加至所有裝置憑證的AWS IoT原則。透過這種方法，您可以將所有裝置的權限視為叢集來管理。如果您想要為每個裝置建立新AWS IoT規則，可以略過本節，並在定義叢集範本時參考其中的政策。

如要建立 AWS IoT 政策

- 建立AWS IoT定義 Greengrass 核心裝置叢集AWS IoT權限的原則。下列原則允許存取所有 MQTT 主題和 Greengrass 作業，因此您的裝置可以處理需要新 Greengrass 作業的自訂應用程式和 future 變更。此政策還允許`iot:AssumeRoleWithCertificate`權限，允許您的設備使用您在上一節中創建的令牌交換角色。您可以根據您的使用案例來限制此政策。如需詳細資訊，請參閱AWS IoT Greengrass V2核心裝置的最低AWS IoT原則。

  請執行下列操作：

  a. 建立包含 Greengrass 核心裝置所需之AWS IoT原則文件的檔案。

     例如，在 Linux 系統上，您可以執行下列命令來使用 GNU nano 建立檔案。

     ```
     nano greengrass-v2-iot-policy.json
     ```

     將下列 JSON 複製到檔案中。

     - 將`iot:AssumeRoleWithCertificate`資源取代為您在上一節中建立的AWS IoT角色別名的 ARN。

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

```
      "Action": [
        "iot:Publish",
        "iot:Subscribe",
        "iot:Receive",
        "iot:Connect",
        "greengrass:*"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "iot:AssumeRoleWithCertificate",
      "Resource": "arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias"
    }
  ]
}
```

b. 從AWS IoT策略文件建立策略。

- 以要建*#########ThingPolicy####*。

```
aws iot create-policy --policy-name GreengrassV2IoTThingPolicy --policy-
document file://greengrass-v2-iot-policy.json
```

如果要求成功，回應看起來類似下列範例。

```
{
  "policyName": "GreengrassV2IoTThingPolicy",
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy",
  "policyDocument": "{
    \"Version\": \"2012-10-17\",
    \"Statement\": [
      {
        \"Effect\": \"Allow\",
        \"Action\": [
          \"iot:Publish\",
          \"iot:Subscribe\",
          \"iot:Receive\",
```

```
          \"iot:Connect\",
          \"greengrass:*\"
        ],
        \"Resource\": [
          \"*\"
        ]
      },
      {
        \"Effect\": \"Allow\",
        \"Action\": \"iot:AssumeRoleWithCertificate\",
        \"Resource\": \"arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias\"
      }
    ]
  }",
  "policyVersionId": "1"
}
```

建立叢集佈建範本

AWS IoT叢集佈建範本定義如何佈建AWS IoT項目、原則和憑證。若要使用叢集佈建外掛程式佈建 Greengrass 核心裝置，您必須建立指定下列內容的範本：

- 物AWS IoT件資源。您可以指定現有物件群組的清單，以便在每個裝置上線時將元件部署到其中。
- AWS IoT策略資源。此資源可以定義下列其中一個屬性：
  - 現有AWS IoT策略的名稱。如果您選擇此選項，您從此範本建立的核心裝置會使用相同的AWS IoT原則，而且您可以管理其作為叢集的權限。
  - 一份AWS IoT政策文件。如果您選擇此選項，您從此範本建立的每個核心裝置都會使用唯一的AWS IoT原則，而且您可以管理每個個別核心裝置的權限。
- 憑AWS IoT證資源。此憑證資源必須使用`AWS::IoT::Certificate::Id`參數將憑證附加至核心裝置。如需詳細資訊，請參閱AWS IoT開發人員指南中的 J ust-in-time 佈建。

在範本中，您可以指定將AWS IoT物件新增至現有物件群組的清單。當核心裝置第一次連線到AWS IoT Greengrass時，它會接收其所屬成員之每個物件群組的 Greengrass 部署。您可以使用物件群組，在每個裝置上線後立即將最新軟體部署到其中。如需詳細資訊，請參閱 將AWS IoT Greengrass元件部署到裝置。

此AWS IoT服務需要在佈建裝置AWS 帳戶時建立和更新AWS IoT資源的權限。若要授予AWS IoT服務存取權，您可以建立 IAM 角色，並在建立範本時提供該角色。 AWS IoT提供受管理的策略

AWSIoTThingsRegistration，允許存取佈建設備時可AWS IoT能使用的所有權限。您可以使用此受管理的原則，或建立自訂原則，針對您的使用案例在受管理原則中設定權限的範圍。

在本節中，您會建立可AWS IoT為裝置佈建資源的 IAM 角色，並建立使用該 IAM 角色的叢集佈建範本。

若要建立叢集佈建範本

1. 建立AWS IoT可假設在AWS 帳戶. 中佈建資源的 IAM 角色 請執行下列操作：

   a. 建立包含允許AWS IoT擔任角色之信任原則文件的檔案。

      例如，在 Linux 系統上，您可以執行下列命令來使用 GNU nano 建立檔案。

      ```
      nano aws-iot-trust-policy.json
      ```

      將下列 JSON 複製到檔案中。

      ```
      {
        "Version": "2012-10-17",
        "Statement": [
          {
            "Effect": "Allow",
            "Principal": {
              "Service": "iot.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
          }
        ]
      }
      ```

   b. 使用信任政策文件建立 IAM 角色。

      - 以要建立的 IAM 角色名稱取*GreengrassFleetProvisioningRole*代。

      ```
      aws iam create-role --role-name GreengrassFleetProvisioningRole --assume-role-
      policy-document file://aws-iot-trust-policy.json
      ```

      如果要求成功，回應看起來類似下列範例。

      ```
      {
      ```

```
  "Role": {
    "Path": "/",
    "RoleName": "GreengrassFleetProvisioningRole",
    "RoleId": "AROAZ2YMUHYHK5OKM77FB",
    "Arn": "arn:aws:iam::123456789012:role/GreengrassFleetProvisioningRole",
    "CreateDate": "2021-07-26T00:15:12+00:00",
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": "iot.amazonaws.com"
          },
          "Action": "sts:AssumeRole"
        }
      ]
    }
  }
}
```

c. 檢閱AWSIoTThingsRegistration原則，該原則允許存取佈建裝置時可AWS IoT能使用的所有權限。您可以使用這個受管理的原則，或建立自訂原則來定義使用案例的範圍下限權限。如果您選擇建立自訂原則，請立即執行。

d. 將 IAM 政策附加到叢集佈建角色。

- 將 *GreengrassFleetProvisioningRole* 取代為 IAM 角色的名稱。
- 如果您在上一個步驟中建立了自訂政策，請將政策 ARN 取代為要使用的 IAM 政策的 ARN。

```
aws iam attach-role-policy --role-name GreengrassFleetProvisioningRole --
policy-arn arn:aws:iam::aws:policy/service-role/AWSIoTThingsRegistration
```

如果請求成功，命令沒有任何輸出。

2. (選擇性) 建立預先佈建勾點，此AWS Lambda函數可驗證裝置在註冊期間提供的範本參數。您可以使用預先佈建掛鉤來更好地控制. AWS 帳戶 如需詳細資訊，請參閱AWS IoT Core開發人員指南中的預先佈建勾點。

3. 建立叢集佈建範本。請執行下列操作：

a. 建立包含佈建範本文件的檔案。

例如，在 Linux 系統上，您可以執行下列命令來使用 GNU nano 建立檔案。

```
nano greengrass-fleet-provisioning-template.json
```

編寫佈建範本文件。您可以從下列範例佈建範本開始，此範本會指定建立具AWS IoT有下列內容的物件：

- 物件的名稱是您在ThingName範本參數中指定的值。
- 物件是您在ThingGroupName範本參數中指定之物件群組的成員。物件群組必須存在於您的AWS 帳戶.
- 物件的憑證已GreengrassV2IoTThingPolicy附加名為的AWS IoT原則。

如需詳細資訊，請參閱AWS IoT Core開發人員指南中的佈建範本。

```
{
  "Parameters": {
    "ThingName": {
      "Type": "String"
    },
    "ThingGroupName": {
      "Type": "String"
    },
    "AWS::IoT::Certificate::Id": {
      "Type": "String"
    }
  },
  "Resources": {
    "MyThing": {
      "OverrideSettings": {
        "AttributePayload": "REPLACE",
        "ThingGroups": "REPLACE",
        "ThingTypeName": "REPLACE"
      },
      "Properties": {
        "AttributePayload": {},
        "ThingGroups": [
          {
            "Ref": "ThingGroupName"
```

```
            }
          ],
          "ThingName": {
            "Ref": "ThingName"
          }
        },
        "Type": "AWS::IoT::Thing"
      },
      "MyPolicy": {
        "Properties": {
          "PolicyName": "GreengrassV2IoTThingPolicy"
        },
        "Type": "AWS::IoT::Policy"
      },
      "MyCertificate": {
        "Properties": {
          "CertificateId": {
            "Ref": "AWS::IoT::Certificate::Id"
          },
          "Status": "Active"
        },
        "Type": "AWS::IoT::Certificate"
      }
    }
}
```

> **Note**
>
> *MyThingMyPolicy*、和*MyCertificate*是識別叢集佈建範本中每個資源規格的任意名稱。 AWS IoT不會在從範本建立的資源中使用這些名稱。您可以使用這些名稱，或將其取代為可協助您識別範本中每個資源的值。

b. 從佈建範本文件建立叢集佈建範本。

- 以要建立的範本名稱取*GreengrassFleetProvisioningTemplate*代。
- 將範本描述取代為範本的描述。
- 將佈建角色 ARN 取代為您先前建立之角色的 ARN。

Linux or Unix

```
aws iot create-provisioning-template \
  --template-name GreengrassFleetProvisioningTemplate \
  --description "A provisioning template for Greengrass core devices." \
  --provisioning-role-arn "arn:aws:iam::123456789012:role/
GreengrassFleetProvisioningRole" \
  --template-body file://greengrass-fleet-provisioning-template.json \
  --enabled
```

Windows Command Prompt (CMD)

```
aws iot create-provisioning-template ^
  --template-name GreengrassFleetProvisioningTemplate ^
  --description "A provisioning template for Greengrass core devices." ^
  --provisioning-role-arn "arn:aws:iam::123456789012:role/
GreengrassFleetProvisioningRole" ^
  --template-body file://greengrass-fleet-provisioning-template.json ^
  --enabled
```

PowerShell

```
aws iot create-provisioning-template `
  --template-name GreengrassFleetProvisioningTemplate `
  --description "A provisioning template for Greengrass core devices." `
  --provisioning-role-arn "arn:aws:iam::123456789012:role/
GreengrassFleetProvisioningRole" `
  --template-body file://greengrass-fleet-provisioning-template.json `
  --enabled
```

Note

如果您已建立預先佈建勾點，請使用引數指定預先佈建勾點的 Lambda 函數的 ARN。`--pre-provisioning-hook`

```
--pre-provisioning-hook targetArn=arn:aws:lambda:us-
west-2:123456789012:function:GreengrassPreProvisioningHook
```

如果要求成功，回應看起來類似下列範例。

```
{
    "templateArn": "arn:aws:iot:us-west-2:123456789012:provisioningtemplate/
GreengrassFleetProvisioningTemplate",
    "templateName": "GreengrassFleetProvisioningTemplate",
    "defaultVersionId": 1
}
```

建立佈建宣告憑證和私密金鑰

宣告憑證是 X.509 憑證，可讓裝置登錄為AWS IoT物件，並擷取用於一般作業的唯一 X.509 裝置憑證。建立宣告憑證之後，您會附加AWS IoT原則，讓裝置可以使用它來建立唯一的裝置憑證，並使用叢集佈建範本進行佈建。具有宣告憑證的裝置只能使用您在AWS IoT原則中允許的佈建範本進行佈建。

在本節中，您會建立宣告憑證，並將其設定為與您在上一節中建立的叢集佈建範本搭配使用的裝置。

> **Important**
>
> 佈建宣告私密金鑰應始終受到保護，包括在 Greengrass 核心裝置上。我們建議您使用 Amazon CloudWatch 指標和日誌來監控濫用的跡象，例如未經授權使用宣告憑證來佈建裝置。如果您偵測到誤用，請停用佈建宣告憑證，使其無法用於裝置佈建。如需詳細資訊，請參閱AWS IoT Core開發人員指南AWS IoT中的監控。
>
> 為了協助您更有效地管理在您的裝置數目以及自行註冊的裝置數目AWS 帳戶，您可以在建立叢集佈建範本時指定預先佈建勾點。預先佈建勾點是一種AWS Lambda函數，可驗證裝置在註冊期間提供的範本參數。例如，您可以建立預先佈建勾點，以根據資料庫檢查裝置 ID，以確認裝置是否具有佈建權限。如需詳細資訊，請參閱AWS IoT Core開發人員指南中的預先佈建勾點。

若要建立佈建宣告憑證和私密金鑰

1. 建立下載宣告憑證和私密金鑰的資料夾。

   ```
   mkdir claim-certs
   ```

2. 建立並儲存用於佈建的憑證和私密金鑰。 AWS IoT提供由 Amazon 根憑證授權單位 (CA) 簽署的用戶端憑證。

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

如果要求成功，回應會包含有關憑證的資訊。儲存憑證的 ARN 以供稍後使用。

3. 建立並附加AWS IoT原則，以允許裝置使用憑證建立唯一的裝置憑證，並使用叢集佈建範本進行佈建。下列原則允許存取裝置佈建 MQTT API。如需詳細資訊，請參閱AWS IoT Core開發人員指南中的裝置佈建 MQTT API。

   請執行下列操作：

   a. 建立包含 Greengrass 核心裝置所需之AWS IoT原則文件的檔案。

      例如，在 Linux 系統上，您可以執行下列命令來使用 GNU nano 建立檔案。

      ```
      nano greengrass-provisioning-claim-iot-policy.json
      ```

      將下列 JSON 複製到檔案中。

- 將*##*的每個執行個體取代為設定叢集佈建的AWS 區域位置。
- 將*## ID #########AWS ### ID*。
- 將的每個執行*GreengrassFleetProvisioningTemplate*個體取代為您在上一節中建立的叢集佈建範本的名稱。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:Connect",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish",
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:region:account-id:topic/$aws/certificates/create/*",
        "arn:aws:iot:region:account-id:topic/$aws/provisioning-
templates/GreengrassFleetProvisioningTemplate/provision/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "iot:Subscribe",
      "Resource": [
        "arn:aws:iot:region:account-id:topicfilter/$aws/certificates/create/*",
        "arn:aws:iot:region:account-id:topicfilter/$aws/provisioning-
templates/GreengrassFleetProvisioningTemplate/provision/*"
      ]
    }
  ]
}
```

b. 從AWS IoT策略文件建立策略。

- 以要建立的策略名稱取*GreengrassProvisioningClaimPolicy*代。

```
aws iot create-policy --policy-name GreengrassProvisioningClaimPolicy --policy-
document file://greengrass-provisioning-claim-iot-policy.json
```

如果要求成功，回應看起來類似下列範例。

```
{
  "policyName": "GreengrassProvisioningClaimPolicy",
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassProvisioningClaimPolicy",
  "policyDocument": "{
    \"Version\": \"2012-10-17\",
    \"Statement\": [
      {
        \"Effect\": \"Allow\",
        \"Action\": \"iot:Connect\",
        \"Resource\": \"*\"
      },
      {
        \"Effect\": \"Allow\",
        \"Action\": [
          \"iot:Publish\",
          \"iot:Receive\"
        ],
        \"Resource\": [
          \"arn:aws:iot:region:account-id:topic/$aws/certificates/create/*\",
          \"arn:aws:iot:region:account-id:topic/$aws/provisioning-
templates/GreengrassFleetProvisioningTemplate/provision/*\"
        ]
      },
      {
        \"Effect\": \"Allow\",
        \"Action\": \"iot:Subscribe\",
        \"Resource\": [
          \"arn:aws:iot:region:account-id:topicfilter/$aws/certificates/create/
*\",
          \"arn:aws:iot:region:account-id:topicfilter/$aws/provisioning-
templates/GreengrassFleetProvisioningTemplate/provision/*\"
        ]
      }
    ]
  }",
```

```
  "policyVersionId": "1"
}
```

4. 將AWS IoT原則附加至佈建宣告憑證。

   - 以要附加的策略名稱取*GreengrassProvisioningClaimPolicy*代。
   - 將目標 ARN 取代為佈建宣告憑證的 ARN。

```
aws iot attach-policy --policy-name GreengrassProvisioningClaimPolicy --
target arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
```

   如果請求成功，命令沒有任何輸出。

您現在擁有佈建宣告憑證和私密金鑰，裝置可用來註冊AWS IoT並佈建為 Greengrass 核心裝置。您可以在製造過程中將宣告憑證和私密金鑰嵌入裝置，或在安裝 AWS IoT Greengrass Core 軟體之前將憑證和金鑰複製到裝置。如需更多詳細資訊，請參閱 透過 AWS IoT 叢集佈建安裝 AWS IoT Greengrass 核心軟體。

設定AWS IoT叢集佈建外掛程式

AWS IoT叢集佈建外掛程式提供下列組態參數，您可以在安裝含有叢集佈建的 AWS IoT Greengrass Core 軟體時自訂這些參數。

`rootPath`

用作AWS IoT Greengrass核心軟體根目錄的資料夾路徑。

`awsRegion`

叢集佈建外掛程式用來佈建AWS資源。AWS 區域

`iotDataEndpoint`

AWS IoT您的AWS 帳戶.

`iotCredentialEndpoint`

AWS IoT您的AWS 帳戶.

`iotRoleAlias`

指向權杖交換 IAM AWS IoT 角色的角色別名。AWS IoT認證提供者會擔任此角色，以允許 Greengrass 核心裝置與服務互動。AWS如需詳細資訊，請參閱 授權核心設備與AWS服務。

`provisioningTemplate`

用於佈建AWS資源的AWS IoT叢集佈建範本。此範本必須指定下列項目：

- 物AWS IoT件資源。您可以指定現有物件群組的清單，以便在每個裝置上線時將元件部署到其中。
- AWS IoT策略資源。此資源可以定義下列其中一個屬性：
  - 現有AWS IoT策略的名稱。如果您選擇此選項，您從此範本建立的核心裝置會使用相同的AWS IoT原則，而且您可以管理其作為叢集的權限。
  - AWS IoT政策文件。如果您選擇此選項，您從此範本建立的每個核心裝置都會使用唯一的AWS IoT原則，而且您可以管理每個個別核心裝置的權限。
- 憑AWS IoT證資源。此憑證資源必須使用`AWS::IoT::Certificate::Id`參數將憑證附加至核心裝置。如需詳細資訊，請參閱AWS IoT開發人員指南中的 J ust-in-time 佈建。

如需詳細資訊，請參閱AWS IoT Core開發人員指南中的佈建範本。

`claimCertificatePath`

您在中指定之佈建範本的佈建宣告憑證路徑`provisioningTemplate`。如需詳細資訊，請參閱 AWS IoT Core API 參考中的 CreateProvisioningClaim。

`claimCertificatePrivateKeyPath`

您在中指定之佈建範本的佈建宣告憑證私密金鑰路徑`provisioningTemplate`。如需詳細資訊，請參閱 AWS IoT Core API 參考中的 CreateProvisioningClaim。

> **Important**
>
> 佈建宣告私密金鑰應始終受到保護，包括在 Greengrass 核心裝置上。我們建議您使用 Amazon CloudWatch 指標和日誌來監控濫用的跡象，例如未經授權使用宣告憑證來佈建裝置。如果您偵測到誤用，請停用佈建宣告憑證，使其無法用於裝置佈建。如需詳細資訊，請參閱AWS IoT Core開發人員指南AWS IoT中的監控。
>
> 為了協助您更有效地管理在您的裝置數目以及自行註冊的裝置數目AWS 帳戶，您可以在建立叢集佈建範本時指定預先佈建勾點。預先佈建勾點是一種AWS Lambda函數，可驗證裝置在註冊期間提供的範本參數。例如，您可以建立預先佈建勾點，以根據資料庫檢查裝置

ID，以確認裝置是否具有佈建權限。如需詳細資訊，請參閱AWS IoT Core開發人員指南中的預先佈建勾點。

`rootCaPath`

Amazon 根憑證授權單位 (CA) 憑證的路徑。

`templateParameters`

(選擇性) 要提供給叢集佈建範本的參數對應。如需詳細資訊，請參閱AWS IoT Core開發人員指南中的佈建範本參數一節。

`deviceId`

(選擇性) 當叢集佈建外掛程式建立 MQTT 連線時，用作用戶端 ID 的裝置識別碼。AWS IoT

預設值：一個隨機的 UUID。

`mqttPort`

(選擇性) 用於 MQTT 連線的連接埠。

預設：8883

`proxyUrl`

(選擇性) 代理伺服器的格式 URL `scheme://userinfo@host:port`。若要使用 HTTPS 代理伺服器，您必須使用叢集佈建外掛程式的 1.1.0 版或更新版本。

- `scheme`— 該計劃，必須是`http`或`https`。

  Important

  核心裝置必須執行 Greengrass 核心 v2.5.0 或更新版本才能使用 HTTPS 代理伺服器。如果您設定 HTTPS 代理，則必須將代理伺服器 CA 憑證新增至核心裝置的 Amazon 根 CA 憑證。如需詳細資訊，請參閱 讓核心裝置信任 HTTPS 代理。

- `userinfo`— (選用) 使用者名稱和密碼資訊。如果您在中指定此資訊`url`，Greengrass 核心裝置會忽略和欄位`username`。`password`
- `host`— 代理伺服器的主機名稱或 IP 位址。
- `port`— (選用) 連接埠號碼。如果您未指定連接埠，則 Greengrass 核心裝置會使用下列預設值：

- `http`— 80
- `https`— 443

`proxyUserName`

(選擇性) 驗證 Proxy 伺服器的使用者名稱。

`proxyPassword`

(選擇性) 驗證 Proxy 伺服器的使用者名稱。

社会责任路径

(選擇性) 憑證簽署要求 (CSR) 檔案的路徑，用於從 CSR 建立裝置憑證。如需詳細資訊，請參閱AWS IoT Core開發人員指南中的依宣告進行佈建。

csrPrivateKey路徑

(選擇性，如果宣告`csrPath`為必要) 用於產生 CSR 之私密金鑰的路徑。私密金鑰必須已用來產生 CSR。如需詳細資訊，請參閱AWS IoT Core開發人員指南中的依宣告進行佈建。

AWS IoT 車隊配置插件更新日誌

下表描述了通過聲明插件（`aws.greengrass.FleetProvisioningByClaim`）佈建的 AWS IoT 車隊的每個版本的更改。

版本	變更
1.2.1	錯誤修復和改進 • 修正了在 Greengrass 核心啟動期間，叢集佈建外掛程式處於離線狀態的問題。車隊佈建外掛程式現在可以無限期地重試 MQTT 連線呼叫。
1.2.0	錯誤修復和改進 • 通過帶有可配置私鑰路徑的證書簽名請求添加對設備佈建的支持。 • 小修正和改進。
1.1.0	錯誤修復和改進 • 添加在 Windows 設備上配置插件時對其他文件路徑格式的支持。 • 添加對 HTTPS 網絡代理配置的支持。如需詳細資訊，請參閱 連線至連接埠 443 或透過網路代理 及 讓核心裝置信任 HTTPS 代理。

版本	變更
1.0.0	初始版本。

使用自訂資源佈建安裝 AWS IoT Greengrass 核心軟體

此功能適用於 v2.4.0 和更高版 Greeng rass 核組件。

AWS IoT Greengrass Core 軟件安裝程序提供了一個 Java 接口，您可以在佈建所需 AWS 資源的自定義插件中實現該接口。您可以開發佈建外掛程式以使用自訂 X.509 用戶端憑證，或執行其他安裝程序不支援的複雜佈建步驟。如需詳細資訊，請參閱AWS IoT Core 開發人員指南中的建立您自己的用戶端憑證。

若要在安裝 AWS IoT Greengrass Core 軟體時執行自訂佈建外掛程式，請建立提供給安裝程式的 JAR 檔案。安裝程序運行插件，並且插件返回一個配置，該配置定義 Greengrass 核心設備的 AWS 資源。安裝程式會使用此資訊來設定裝置上的 AWS IoT Greengrass Core 軟體。如需詳細資訊，請參閱 開發自定義配置插件。

> **Important**
>
> 在您下載 AWS IoT Greengrass Core 軟體之前，請檢查您的核心裝置是否符合安裝和執行 AWS IoT Greengrass Core 軟體 2.0 的需求。

主題

必要條件

若要使用自訂佈建來安裝 AWS IoT Greengrass Core 軟體，您必須具備下列項目：

- 實作的自訂佈建外掛程式的 JAR 檔案`DeviceIdentityInterface`。自訂佈建外掛程式必須傳回每個系統和核心組態參數的值。否則，您必須在安裝期間在組態檔案中提供這些值。如需詳細資訊，請參閱 開發自定義配置插件。

設定裝置環境

請依照本節中的步驟設定 Linux 或 Windows 裝置作為 AWS IoT Greengrass 核心裝置使用。

設定裝置

若要設定下列項目的 Linux 設備 AWS IoT Greengrass V2

1. 安裝 Java 執行階段， AWS IoT Greengrass 核心軟體需要執行。我們建議您使用 Amazon Corretto 或 OpenJDK 長期支持版本。需要版本 8 或更高版本。以下命令向您展示如何在設備上安裝 OpenJDK。

   - 針對 Debian 為基礎或 Ubuntu 為基礎的發行版本：

     ```
     sudo apt install default-jdk
     ```

   - 針對 Red Hat 為基礎的發行版本：

     ```
     sudo yum install java-11-openjdk-devel
     ```

   - 針對 Amazon Linux 2：

     ```
     sudo amazon-linux-extras install java-openjdk11
     ```

   - 對於 Amazon

     ```
     sudo dnf install java-11-amazon-corretto -y
     ```

   安裝完成時，請執行下列命令以確認 Java 是否在 Linux 裝置上執行。

   ```
   java -version
   ```

   此指令會列印在裝置上執行的 Java 版本。例如，在基於 Debian 的發行版上，輸出可能類似於以下示例。

```
openjdk version "11.0.9.1" 2020-11-04
OpenJDK Runtime Environment (build 11.0.9.1+1-post-Debian-1deb10u2)
OpenJDK 64-Bit Server VM (build 11.0.9.1+1-post-Debian-1deb10u2, mixed mode)
```

2. (選擇性) 建立在裝置上執行元件的預設系統使用者和群組。您也可以選擇讓 AWS IoT Greengrass Core 軟體安裝程式在安裝期間使用安裝程式引數建立此使用者和群組。`--component-default-user`如需詳細資訊，請參閱 安裝器引數。

```
sudo useradd --system --create-home ggc_user
sudo groupadd --system ggc_group
```

3. 確認執行 AWS IoT Greengrass Core 軟體的使用者 (通常`root`) 具有`sudo`與任何使用者和任何群組一起執行的權限。

   a. 執行下列命令以開啟`/etc/sudoers`檔案。

   ```
   sudo visudo
   ```

   b. 確認使用者的權限看起來像下列範例。

   ```
   root    ALL=(ALL:ALL) ALL
   ```

4. (選擇性) 若要執行容器化 Lambda 函數，您必須啟用 cgroup v1，並且必須啟用並掛接記憶體和裝置 cgroup。如果您不打算執行容器化 Lambda 函數，則可以略過此步驟。

   若要啟用這些 cgroups 選項，請使用下列 Linux 核心參數來啟動裝置。

   ```
   cgroup_enable=memory cgroup_memory=1 systemd.unified_cgroup_hierarchy=0
   ```

   如需有關檢視和設定裝置核心參數的資訊，請參閱作業系統和開機載入程式的說明文件。依照指示永久設定核心參數。

5. 依照中的需求清單所指示，在您的裝置上安裝所有其他必要的相依性裝置要求。

設定視窗裝置

> **Note**
>
> 此功能適用於 v2.5.0 及更高版 Greeng rass 核組件。

若要設定下列項目的視窗裝置 AWS IoT Greengrass V2

1. 安裝 Java 執行階段， AWS IoT Greengrass 核心軟體需要執行。我們建議您使用 Amazon Corretto 或 OpenJDK 長期支持版本。需要版本 8 或更高版本。
2. 檢查系統變數 PATH 上是否可用 Java，如果沒有，則加入它。此 LocalSystem 帳戶會執行 AWS IoT Greengrass 核心軟體，因此您必須將 Java 新增至系統變數 PATH，而非使用者的 PATH 使用者變數。請執行下列操作：

   a. 按視窗鍵開啟開始功能表。

   b. 鍵入**environment variables**以從開始功能表搜尋系統選項。

   c. 在開始功能表搜尋結果中，選擇 [編輯系統環境變數] 以開啟 [系統屬性] 視窗。

   d. 選擇環境變數... 以開啟 [環境變數] 視窗。

   e. 在 [系統變數] 下，選取 [路徑]，然後選擇 [編輯]。在 「編輯」環境變數視窗中，您可以在不同的行上檢視每個路徑。

   f. 檢查 Java 安裝`bin`資料夾的路徑是否存在。路徑看起來可能類似下列範例。

   ```
   C:\\Program Files\\Amazon Corretto\\jdk11.0.13_8\\bin
   ```

   g. 如果路徑中缺少 Java 安裝的`bin`資料夾，請選擇 [新增] 加入，然後選擇 [確定]。
3. 以系統管理員身分開啟 Windows 命令提示字元 (`cmd.exe`)。
4. 在 Windows 裝置上的 LocalSystem 帳戶中建立預設使用者。以安全*##*取代密碼。

```
net user /add ggc_user password
```

> **Tip**
>
> 視您的 Windows 組態而定，使用者的密碼可能會設定為在 future 的某個日期到期。為確保您的 Greengrass 應用程式繼續運作，請追蹤密碼何時到期，並在密碼到期之前進行更新。您也可以將使用者的密碼設定為永不過期。
>
> - 若要檢查使用者及其密碼何時到期，請執行下列命令。
>
>   ```
>   net user ggc_user | findstr /C:expires
>   ```
>
> - 若要將使用者的密碼設定為永不過期，請執行下列命令。
>
>   ```
>   wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
>   ```

- 如果您使用的是 Windows 10 或更新版本，而該wmic命令已被取代，請執行下列 PowerShell 命令。

```
Get-CimInstance -Query "SELECT * from Win32_UserAccount WHERE name =
 'ggc_user'" | Set-CimInstance -Property @{PasswordExpires="False"}
```

5. 在設備上從 Microsoft 下載並安裝該PsExec實用程序。
6. 使用此 PsExec 公用程式將預設使用者的使用者名稱和密碼儲存在 LocalSystem 帳戶的認證管理員執行個體中。*################*

```
psexec -s cmd /c cmdkey /generic:ggc_user /user:ggc_user /pass:password
```

如果打PsExec License Agreement開，請選Accept擇同意許可證並運行命令。

> **Note**
>
> 在 Windows 裝置上， LocalSystem 帳戶會執行 Greengrass 核心，而且您必須使用公用 PsExec 程式將預設使用者資訊儲存在帳戶中。 LocalSystem 使用認證管理員應用程式會將此資訊儲存在目前登入使用者的 Windows 帳戶中，而非 LocalSystem 帳戶。

下載 AWS IoT Greengrass 核心軟體

您可以從下列位置下載最新版本的 AWS IoT Greengrass Core 軟體：

- https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest. 壓縮

> **Note**
>
> 您可以從下列位置下載特定版本的 AWS IoT Greengrass Core 軟體。將*##*取代為要下載的版本。
>
> ```
> https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
> ```

若要下載 AWS IoT Greengrass 核心軟體

1. 在您的核心裝置上，將 AWS IoT Greengrass Core 軟體下載至名為`greengrass-nucleus-latest.zip`.

   Linux or Unix

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip > greengrass-nucleus-latest.zip
   ```

   Windows Command Prompt (CMD)

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip > greengrass-nucleus-latest.zip
   ```

   PowerShell

   ```
   iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip -OutFile greengrass-nucleus-latest.zip
   ```

   下載此軟體，即表示您同意 Greengrass 核心軟體授權合約之規定。

2. (選擇性) 驗證 Greengrass 核軟體簽章

   > **Note**
   >
   > 此功能適用於 Greengrass 核 2.9.5 及更高版本。

   a. 使用以下命令來驗證 Greengrass 核工件的簽名：

      Linux or Unix

      ```
      jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
      ```

      Windows Command Prompt (CMD)

      根據您安裝的 JDK 版本，檔案名稱可能會有所不同。以您安裝的 JDK 版本取代*jdk17.0.6_10*。

```
"C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -
verify -certs -verbose greengrass-nucleus-latest.zip
```

PowerShell

根據您安裝的 JDK 版本，檔案名稱可能會有所不同。以您安裝的 JDK 版本取代*jdk17.0.6_10*。

```
'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -
verify -certs -verbose greengrass-nucleus-latest.zip
```

b. `jarsigner`調用產生指示驗證結果的輸出。

   i. 如果 Greengrass 核 zip 文件簽名，則輸出包含以下語句：

   ```
   jar verified.
   ```

   ii. 如果 Greengrass 核壓縮檔未簽署，則輸出會包含下列陳述式：

   ```
   jar is unsigned.
   ```

c. 如果您提供了 Jarsigner `-certs -verbose` 選項`-verify`和選項，則輸出也會包含詳細的簽署者憑證資訊。

3. 將 AWS IoT Greengrass Core 軟件解壓縮到設備上的文件夾。以您要使用的資料夾取*GreengrassInstaller*代。

   Linux or Unix

   ```
   unzip greengrass-nucleus-latest.zip -d GreengrassInstaller && rm greengrass-
   nucleus-latest.zip
   ```

   Windows Command Prompt (CMD)

   ```
   mkdir GreengrassInstaller && tar -xf greengrass-nucleus-latest.zip -
   C GreengrassInstaller && del greengrass-nucleus-latest.zip
   ```

PowerShell

```
Expand-Archive -Path greengrass-nucleus-latest.zip -DestinationPath .\
\GreengrassInstaller
rm greengrass-nucleus-latest.zip
```

4. (選擇性) 執行下列命令以查看 AWS IoT Greengrass Core 軟體的版本。

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

> **Important**
>
> 如果您安裝的 Greengrass 核心版本早於 v2.4.0，請不要在安裝核心軟體之後移除此資料夾。AWS IoT Greengrass AWS IoT Greengrass 核心軟件使用此文件夾中的文件運行。
> 如果您下載了最新版本的軟體，請安裝 v2.4.0 或更新版本，而且您可以在安裝 AWS IoT Greengrass Core 軟體之後移除此資料夾。

安裝 AWS IoT Greengrass 核心軟體

使用指定下列動作的引數執行安裝程式：

- 從指定使用自訂佈建外掛程式佈建 AWS 資源的部分組態檔案進行安裝。 AWS IoT Greengrass 核心軟體使用組態檔案來指定裝置上每個 Greengrass 元件的組態。安裝程式會從您提供的部分組態檔案以及自訂佈建外掛程式所建立的 AWS 資源建立完整的組態檔案。
- 指定使用`ggc_user`系統使用者在核心裝置上執行軟體元件。在 Linux 裝置上，此指令也會指定使用`ggc_group`系統群組，而安裝程式會為您建立系統使用者和群組。
- 將 AWS IoT Greengrass Core 軟體設定為在開機時執行的系統服務。在 Linux 設備上，這需要系統初始化系統。

> **Important**
>
> 在 Windows 核心裝置上，您必須將 AWS IoT Greengrass 核心軟體設定為系統服務。

如需可指定之引數的詳細資訊，請參閱安裝器引數。

> **Note**
>
> 如果您在記憶體有限的裝置 AWS IoT Greengrass 上執行，您可以控制 AWS IoT Greengrass Core 軟體使用的記憶體數量。若要控制記憶體分配，您可以在核心元件的`jvmOptions`組態參數中設定 JVM 堆積大小選項。如需詳細資訊，請參閱 使用 JVM 選項控制內存分配。

若要安裝 AWS IoT Greengrass 核心軟體 (Linux)

1. 檢查 AWS IoT Greengrass 核心軟體的版本。

   - 以包含軟體的資料夾路徑取*GreengrassInstaller*代。

   ```
   java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
   ```

2. 使用文字編輯器建立名為提供`config.yaml`給安裝程式的組態檔案。

   例如，在 Linux 系統上，您可以執行下列命令來使用 GNU nano 建立檔案。

   ```
   nano GreengrassInstaller/config.yaml
   ```

   將下列 YAML 內容複製到檔案中。

   ```
   ---
   system:
     rootpath: "/greengrass/v2"
     # The following values are optional. Return them from the provisioning plugin or
    set them here.
     # certificateFilePath: ""
     # privateKeyPath: ""
     # rootCaPath: ""
     # thingName: ""
   services:
     aws.greengrass.Nucleus:
       version: "2.12.3"
       configuration:
         # The following values are optional. Return them from the provisioning plugin
    or set them here.
         # awsRegion: ""
         # iotRoleAlias: ""
   ```

```
    # iotDataEndpoint: ""
    # iotCredEndpoint: ""
  com.example.CustomProvisioning:
    configuration:
      # You can specify configuration parameters to provide to your plugin.
      # pluginParameter: ""
```

然後，執行下列動作：

- 以 AWS IoT Greengrass 核心軟體的版本取代 *2.12.3*。
- */greengrass/v2*以 Greengrass 根資料夾取代的每個執行個體。
- (選擇性) 指定系統和核心組態值。如果您的佈建外掛程式未提供這些值，則必須設定這些值。
- (選擇性) 指定要提供給佈建外掛程式的組態參數。

> **Note**
>
> 在此組態檔中，您可以自訂其他組態選項，例如要使用的連接埠和網路 Proxy，如下列範例所示。如需詳細資訊，請參閱 Greengrass 核組態。
>
> ```
> ---
> system:
>   rootpath: "/greengrass/v2"
>   # The following values are optional. Return them from the provisioning
>  plugin or set them here.
>   # certificateFilePath: ""
>   # privateKeyPath: ""
>   # rootCaPath: ""
>   # thingName: ""
> services:
>   aws.greengrass.Nucleus:
>     version: "2.12.3"
>     configuration:
>       mqtt:
>         port: 443
>       greengrassDataPlanePort: 443
>       networkProxy:
>         noProxyAddresses: "http://192.168.0.1,www.example.com"
>         proxy:
>           url: "http://my-proxy-server:1100"
>           username: "Mary_Major"
> ```

```
            password: "pass@word1357"
        # The following values are optional. Return them from the provisioning
    plugin or set them here.
        # awsRegion: ""
        # iotRoleAlias: ""
        # iotDataEndpoint: ""
        # iotCredEndpoint: ""
     com.example.CustomProvisioning:
       configuration:
         # You can specify configuration parameters to provide to your plugin.
         # pluginParameter: ""
```

3. 執行安裝程式。指定`--trusted-plugin`提供您的自訂佈建外掛程式，並指`--init-config`定提供組態檔案。

   - 使用 Greengrass 根資料夾取代*/greengrass/v2*或 *C:\greengrass\v2*。
   - 將的每個執行個*GreengrassInstaller*體取代為解壓縮安裝程式的資料夾。
   - 將自定義配置插件 JAR 文件的路徑替換為插件的 JAR 文件的路徑。

   Linux or Unix

   ```
   sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \
     -jar ./GreengrassInstaller/lib/Greengrass.jar \
     --trusted-plugin /path/to/com.example.CustomProvisioning.jar \
     --init-config ./GreengrassInstaller/config.yaml \
     --component-default-user ggc_user:ggc_group \
     --setup-system-service true
   ```

   Windows Command Prompt (CMD)

   ```
   java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" ^
     -jar ./GreengrassInstaller/lib/Greengrass.jar ^
     --trusted-plugin /path/to/com.example.CustomProvisioning.jar ^
     --init-config ./GreengrassInstaller/config.yaml ^
     --component-default-user ggc_user ^
     --setup-system-service true
   ```

   PowerShell

   ```
   java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" `
   ```

```
  -jar ./GreengrassInstaller/lib/Greengrass.jar `
  --trusted-plugin /path/to/com.example.CustomProvisioning.jar `
  --init-config ./GreengrassInstaller/config.yaml `
  --component-default-user ggc_user `
  --setup-system-service true
```

> **Important**
>
> 在 Windows 核心裝置上，您必須指`--setup-system-service true`定將 AWS IoT Greengrass 核心軟體設定為系統服務。

如果您指定`--setup-system-service true`，安裝程式會列印是`Successfully set up Nucleus as a system service`否將軟體設定為系統服務並執行。否則，如果安裝程式成功安裝軟體，則不會輸出任何訊息。

> **Note**
>
> 當您執行沒有`deploy-dev-tools`引數的安裝程式時，您無法使用`--provision true`引數來部署本機開發工具。如需直接在裝置上部署 Greengrass CLI 的相關資訊，請參閱。綠色命令行界面

4. 檢視根資料夾中的檔案來確認安裝。

   Linux or Unix

   ```
   ls /greengrass/v2
   ```

   Windows Command Prompt (CMD)

   ```
   dir C:\greengrass\v2
   ```

   PowerShell

   ```
   ls C:\greengrass\v2
   ```

   如果安裝成功，則根資料夾會包含數個資料夾`config`，例如`packages`、和`logs`。

如果您將 AWS IoT Greengrass Core 軟體安裝為系統服務，安裝程式會為您執行該軟體。否則，您必須手動運行該軟件。如需詳細資訊，請參閱 執行AWS IoT Greengrass核心軟體。

如需有關如何設定及使用軟體的詳細資訊 AWS IoT Greengrass，請參閱下列內容：

- 設定 AWS IoT Greengrass 核心軟體
- 開發AWS IoT Greengrass元件
- 將AWS IoT Greengrass元件部署到裝置
- 綠色命令行界面

開發自定義配置插件

要開發自定義配置插件，請創建一個實現`com.aws.greengrass.provisioning.DeviceIdentityInterface`界面。您可以在項目中包含 Greengrass 核 JAR 文件，以訪問此接口及其類。此接口定義了輸入插件配置並輸出配置配置的方法。配置配置定義了系統的配置和Greengrass 核成分。所以此AWS IoT Greengrass核心軟件安裝程序使用此置備配置來配置AWS IoT Greengrass設備上的核心軟件。

開發自定義置備插件後，將其構建為 JAR 文件，您可以提供給AWS IoT Greengrass在安裝過程中運行插件的核心軟件安裝程序。安裝程序在安裝程序使用的同一 JVM 中運行自定義置備插件，因此您可以創建僅包含插件代碼的 JAR。

> **Note**
>
> 所以此AWS IoT機羣佈建插件實作`DeviceIdentityInterface`，以在安裝過程中使用機羣佈建。隊列配置插件是開源的，因此您可以瀏覽其源代碼，以查看如何使用置備插件接口的示例。如需詳細資訊，請參閱 。AWS IoT機羣佈建插件(在 GitHub 上)。

主題

- 要求
- 實作 DeviceIdentityInterface 介面

要求

要開發自定義配置插件，您必須創建滿足以下要求的 Java 類：

- 使用`com.aws.greengrass`軟件包中的軟件包或`com.aws.greengrass`軟體。

- 有一個沒有任何參數的構造函數。
- 實作`DeviceIdentityInterface`界面。如需詳細資訊，請參閱 實作 DeviceIdentityInterface 介面。

實作 DeviceIdentityInterface 介面

若要使用 AWS for WordPress`com.aws.greengrass.provisioning.DeviceIdentityInterface`接口，將 Greengrass 核心作為依賴項添加到您的項目中。

若要使用 AWS for WordPress DeviceIdentityInterface 在自定義配置插件項目中

- 您可以將 Greengrass 核 JAR 文件添加為庫，或添加 Greengrass 核作為 Maven 依賴項。執行下列任意一項：
  - 要將 Greengrass 核 JAR 文件添加為庫，請下載AWS IoT Greengrass核心軟件，其中包含 Greengrass 核 JAR。您可以下載最新版本的AWS IoT Greengrass來自以下位置的核心軟體：
    - https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip

    您可以找到 Greengrass 核 JAR 文件（`Greengrass.jar`）在`lib`文件夾。將此 JAR 文件添加到您的項目中。
  - 要在 Maven 項目中使用 Greengrass 核，請在`nucleus`中的工件`com.aws.greengrass`組。您還必須添加`greengrass-common`存儲庫，因為 Greengrass 核在 Maven 中央存儲庫中不可用。

```
<project ...>
    ...
    <repositories>
        <repository>
            <id>greengrass-common</id>
            <name>greengrass common</name>
            <url>https://d2jrmugq4soldf.cloudfront.net/snapshots</url>
        </repository>
    </repositories>
    ...
    <dependencies>
        <dependency>
            <groupId>com.aws.greengrass</groupId>
```

```
            <artifactId>nucleus</artifactId>
            <version>2.5.0-SNAPSHOT</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>
</project>
```

設備標識介面

所以此com.aws.greengrass.provisioning.DeviceIdentityInterface界面的形態如下。

> **Note**
>
> 您也可以在綠色草案配置軟件包的Greengrass 原始碼(在 GitHub 上)。

```
public interface com.aws.greengrass.provisioning.DeviceIdentityInterface {
    ProvisionConfiguration updateIdentityConfiguration(ProvisionContext context)
            throws RetryableProvisioningException, InterruptedException;

    // Return the name of the plugin.
    String name();
}

com.aws.greengrass.provisioning.ProvisionConfiguration {
    SystemConfiguration systemConfiguration;
    NucleusConfiguration nucleusConfiguration
}

com.aws.greengrass.provisioning.ProvisionConfiguration.SystemConfiguration {
    String certificateFilePath;
    String privateKeyPath;
    String rootCAPath;
    String thingName;
}

com.aws.greengrass.provisioning.ProvisionConfiguration.NucleusConfiguration {
    String awsRegion;
    String iotCredentialsEndpoint;
    String iotDataEndpoint;
    String iotRoleAlias;
}
```

```
com.aws.greengrass.provisioning.ProvisioningContext {
    Map<String, Object> parameterMap;
    String provisioningPolicy;  // The policy is always "PROVISION_IF_NOT_PROVISIONED".
}

com.aws.greengrass.provisioning.exceptions.RetryableProvisioningException {}
```

每個配置值SystemConfiguration和NucleusConfiguration才能安裝AWS IoT Greengrass核心軟件，但您可以返回null。如果您的自定義配置插件返回null對於任何配置值，則必須在系統或 Nucleus 配置中提供該值，當您創建config.yaml文件以提供給AWS IoT Greengrass核心軟體安裝程序。如果您的自定義置備插件返回一個非空值，您也可以在config.yaml，則安裝程序將替換config.yaml使用插件返回的值。

安裝器引數

AWS IoT Greengrass核心軟體包含一個安裝程式，可設定軟體並佈建 Greengrass 核心裝置執行所需的AWS資源。安裝程式包含下列引數，您可以指定用來設定安裝：

`-h, --help`

(選擇性) 顯示安裝程式的說明資訊。

`--version`

(選擇性) 顯示AWS IoT Greengrass核心軟體的版本。

`-Droot`

(選擇性) 用作AWS IoT Greengrass核心軟體根目錄的資料夾路徑。

> **Note**
>
> 此引數`-jar`會設定 JVM 屬性，因此您必須在執行安裝程式之前指定它。例如，指定 `java -Droot="/greengrass/v2" -jar /path/to/Greengrass.jar`。

預設：

- Linux：`~/.greengrass`
- Windows：`%USERPROFILE%/.greengrass`

`-ar, --aws-region`

AWS 區域該AWS IoT Greengrass核心軟件用來檢索或創建其所需的AWS資源。

`-p, --provision`

(選擇性) 您可以將此裝置註冊為AWS IoT物件，並佈建核心裝置所需的AWS資源。如果您指定`true`，AWS IoT Greengrass核心軟體會佈建AWS IoT物件、(選用) AWS IoT 物件群組、IAM 角色和AWS IoT角色別名。

預設：`false`

`-tn, --thing-name`

(選擇性) 您註冊為此核心裝置的AWS IoT物件名稱。如果名稱的東西不存在於您的內容AWS 帳戶，則 AWS IoT Greengrass Core 軟件將創建它。

> **Note**
>
> 物件名稱不能包含冒號 (`:`) 字元。

您必須指`--provision true`定套用此引數。

預設值：`GreengrassV2IotThing_`加上一個隨機的 UUID。

`-tgn, --thing-group-name`

(選擇性) 您新增此核心裝置AWS IoT物件的AWS IoT物件群組名稱。如果部署以此物件群組為目標，則此核心裝置會在連線至時接收該部署AWS IoT Greengrass。如果具有此名稱的物件群組不存在於您的中AWS 帳戶，則 AWS IoT Greengrass Core 軟體會建立該物件群組。

> **Note**
>
> 物件群組名稱不能包含冒號 (`:`) 字元。

您必須指`--provision true`定套用此引數。

`-tpn, --thing-policy-name`

此功能適用於 v2.4.0 和更高版 Greeng rass 核組件。

(選擇性) 要附加至此核心裝置AWS IoT物件憑證的AWS IoT原則名稱。如果您的中不存在具有此名稱的AWS IoT原則AWS 帳戶，則 AWS IoT Greengrass Core 軟體會建立它。

根據預設，AWS IoT GreengrassCore 軟體會建立寬容AWS IoT原則。您可以縮小此原則的範圍，或建立自訂原則，以限制使用案例的權限。如需詳細資訊，請參閱 AWS IoT Greengrass V2核心裝置的最低AWS IoT原則。

您必須指`--provision true`定套用此引數。

預設：`GreengrassV2IoTThingPolicy`

`-trn`, `--tes-role-name`

(選用) 要用來取得AWS認證的 IAM 角色名稱，讓核心裝置與AWS服務互動。如果您的中不存在具有此名稱的角色AWS 帳戶，則 AWS IoT Greengrass Core 軟體會使用`GreengrassV2TokenExchangeRoleAccess`原則建立該角色。此角色無法存取託管元件成品的 S3 儲存貯體。因此，您必須在建立元件時，將權限新增至人工因素的 S3 儲存貯體和物件。如需詳細資訊，請參閱 授權核心設備與AWS服務。

您必須指`--provision true`定套用此引數。

預設：`GreengrassV2TokenExchangeRole`

`-tra`, `--tes-role-alias-name`

(選擇性) 指向為此核心裝置提供AWS登入資料的 IAM 角色的角色別名名稱。AWS IoT如果您的中不存在具有此名稱的角色別名AWS 帳戶，則 AWS IoT Greengrass Core 軟體會建立該別名，並將其指向您指定的 IAM 角色。

您必須指`--provision true`定套用此引數。

預設：`GreengrassV2TokenExchangeRoleAlias`

`-ss`, `--setup-system-service`

(選擇性) 您可以將 AWS IoT Greengrass Core 軟體設定為在此裝置開機時執行的系統服務。系統服務名稱為`greengrass`。如需詳細資訊，請參閱 將 Greengrass 核配置為系統服務。

在 Linux 作業系統上，此引數要求您必須在裝置上使用 systemd 初始化系統。

> **Important**
>
> 在 Windows 核心裝置上，您必須將AWS IoT Greengrass核心軟體設定為系統服務。

預設：`false`

`-u, --component-default-user`

AWS IoT GreengrassCore 軟體用來執行元件的使用者名稱或識別碼。例如，您可以指定 **`ggc_user`**。當您在 Windows 作業系統上執行安裝程式時，需要此值。

在 Linux 作業系統上，您也可以選擇性地指定群組。指定以冒號分隔的使用者和群組。例如 **`ggc_user:ggc_group`**。

下列其他考量適用於 Linux 作業系統：

- 如果您以 root 身份執行，預設元件使用者就是組態檔案中定義的使用者。如果配置文件沒有定義用戶，則默認為`ggc_user:ggc_group`。如果`ggc_user`或`ggc_group`不存在，軟件將創建它們。
- 如果您以非 root 使用者身分執行，AWS IoT GreengrassCore 軟體會使用該使用者來執行元件。
- 如果您未指定群組，AWS IoT GreengrassCore 軟體會使用系統使用者的主要群組。

如需詳細資訊，請參閱 設定執行元件的使用者。

`-d, --deploy-dev-tools`

(選擇性) 您可以下載 Greengrass CLI 元件並將其部署到這個核心裝置。您可以使用此工具來開發和偵錯此核心裝置上的元件。

> **Important**
>
> 我們建議您僅在開發環境中使用此元件，而不是在生產環境中使用。此元件可讓您存取通常在生產環境中不需要的資訊和作業。只將此元件部署到您需要的核心裝置，以遵循最低權限原則。

您必須指`--provision true`定套用此引數。

預設：`false`

`-init, --init-config`

(選擇性) 用來安裝 AWS IoT Greengrass Core 軟體的組態檔案路徑。例如，您可以使用此選項來設定具有特定核心組態的新核心裝置。

> **Important**
>
> 您指定的組態檔案會與核心裝置上的現有組態檔案合併。這包括核心裝置上的元件和元件組態。我們建議組態檔案僅列出您嘗試變更的組態。

`-tp, --trusted-plugin`

(選擇性) 要載入為受信任外掛程式之 JAR 檔案的路徑。使用此選項可提供佈建外掛程式 JAR 檔案，例如透過叢集佈建或自訂佈建進行安裝，或使用私密金鑰和憑證安裝在硬體安全性模組中。

`-s, --start`

(選擇性) 您可以在安裝 AWS IoT Greengrass Core 軟體後啟動該軟體，並選擇性地佈建資源。

預設：`true`

執行AWS IoT Greengrass核心軟體

安裝 AWS IoT Greengrass Core 軟件後，請運行它以將設備連接到AWS IoT Greengrass。

當您安裝 AWS IoT Greengrass Core 軟體時，您可以指定是否使用 systemd 將其安裝為系統服務。如果您選擇此選項，安裝程式會為您執行軟體，並將其設定為在裝置開機時執行。

> **Important**
>
> 在 Windows 核心裝置上，您必須將AWS IoT Greengrass核心軟體設定為系統服務。

主題

檢查 AWS IoT Greengrass Core 軟件是否作為系統服務運行

當您安裝 AWS IoT Greengrass Core 軟體時，您可以指定將 AWS IoT Greengrass Core 軟體安裝為系統服務的`--setup-system-service true`引數。Linux 設備需要系統初始化系統將AWS IoT Greengrass核心軟件設置為系統服務。如果您使用此選項，安裝程式會為您執行軟體，並將其設定為

在裝置開機時執行。如果安裝程式成功將 AWS IoT Greengrass Core 軟體安裝為系統服務，則會輸出下列訊息。

```
Successfully set up Nucleus as a system service
```

如果您之前安裝了 AWS IoT Greengrass Core 軟件，但沒有安裝程序輸出，則可以檢查軟件是否安裝為系統服務。

若要檢查AWS IoT Greengrass核心軟體是否已安裝為系統服務

- 運行以下命令來檢查 Greengrass 系統服務的狀態。

  Linux or Unix (systemd)

  ```
  sudo systemctl status greengrass.service
  ```

  如果AWS IoT Greengrass核心軟體安裝為系統服務且作用中，則回應類似於下列範例。

  ```
  # greengrass.service - Greengrass Core
     Loaded: loaded (/etc/systemd/system/greengrass.service; enabled; vendor
   preset: disabled)
     Active: active (running) since Thu 2021-02-11 01:33:44 UTC; 4 days ago
   Main PID: 16107 (sh)
     CGroup: /system.slice/greengrass.service
             ##16107 /bin/sh /greengrass/v2/alts/current/distro/bin/loader
             ##16111 java -Dlog.store=FILE -Droot=/greengrass/v2 -jar /greengrass/
  v2/alts/current/distro/lib/Greengrass...
  ```

  如果找`greengrass.service`不到`systemctl`或找不到，則不會將 AWS IoT Greengrass Core 軟體安裝為系統服務。若要執行軟體，請參閱[在沒有系統服務的情況下執行 AWS IoT Greengrass Core 軟體](#)。

  Windows Command Prompt (CMD)

  ```
  sc query greengrass
  ```

  響應看起來類似於下面的例子，如果AWS IoT Greengrass核心軟件安裝為 Windows 服務和活動。

  ```
  SERVICE_NAME: greengrass
  ```

```
        TYPE               : 10  WIN32_OWN_PROCESS
        STATE              : 4  RUNNING
                                (STOPPABLE, NOT_PAUSABLE, ACCEPTS_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x0
```

PowerShell

```
Get-Service greengrass
```

響應看起來類似於下面的例子，如果AWS IoT Greengrass核心軟件安裝為 Windows 服務和活動。

```
Status   Name               DisplayName
------   ----               -----------
Running  greengrass         greengrass
```

運行AWS IoT Greengrass核心軟件作為系統服務

如果AWS IoT Greengrass核心軟體安裝為系統服務，您可以使用系統服務管理員來啟動、停止和管理軟體。如需詳細資訊，請參閱 [將 Greengrass 核配置為系統服務](#)。

若要執行AWS IoT Greengrass核心軟體

- 執行下列命令來啟動 AWS IoT Greengrass Core 軟體。

  Linux or Unix (systemd)

  ```
  sudo systemctl start greengrass.service
  ```

  Windows Command Prompt (CMD)

  ```
  sc start greengrass
  ```

PowerShell

```
Start-Service greengrass
```

在沒有系統服務的情況下執行 AWS IoT Greengrass Core 軟體

在 Linux 核心裝置上，如果AWS IoT Greengrass核心軟體未安裝為系統服務，您可以執行該軟體的載入程式指令碼來執行軟體。

若要在沒有系統服務的情況下執行 AWS IoT Greengrass Core 軟體

- 執行下列命令來啟動 AWS IoT Greengrass Core 軟體。如果您在終端機中執行此命令，則必須保持終端機工作階段開啟，以保持AWS IoT Greengrass核心軟體的執行狀態。
  - 將*/greengrass/v2*或 *C:\greengrass\v2* 取代為您使用的 Greengrass 根資料夾。

```
sudo /greengrass/v2/alts/current/distro/bin/loader
```

如果成功啟動，軟件將打印以下消息。

```
Launched Nucleus successfully.
```

在 Docker 容器中執行 AWS IoT Greengrass 核心軟體

AWS IoT Greengrass 可以配置為在 Docker 容器中運行。Docker 是一個平台，可為您提供各種工具來構建，運行，測試和部署基於 Linux 容器的應用程序。當您執行 AWS IoT Greengrass Docker 映像檔時，您可以選擇是否提供您的 AWS 認證給 Docker 容器，並允許 AWS IoT Greengrass 核心軟體安裝程式自動佈建 Greengrass 核心裝置運作所需的 AWS 資源。如果您不想提供 AWS 認證，則可以在 Docker 容器中手動佈建 AWS 資源並執行 AWS IoT Greengrass Core 軟體。

主題

- AWS IoT Greengrass在具有自動資源佈建的 Docker 容器中執行
- AWS IoT Greengrass在具有手動資源佈建的 Docker 容器中執行
- Docker 容器中 AWS IoT Greengrass 的疑難排解

支援平台和需求

主機電腦必須符合下列最低需求，才能在 Docker 容器中安裝和執行 AWS IoT Greengrass Core 軟體：

- 具有網際網路連線的 Linux 作業系統。
- 碼頭引擎版本 18.09 或更高版本。
- （可選）碼頭編寫版本 1.22 或更高版本。只有當你想使用碼頭構成 CLI 來運行 Docker 映像時，才需要碼頭構成。

若要在 Docker 容器內執行 Lambda 函數元件，您必須設定容器以符合其他需求。如需詳細資訊，請參閱 Lambda 函數要求。

在處理模式下執行元件

AWS IoT Greengrass 不支援在 AWS IoT Greengrass Docker 容器內的隔離執行階段環境中執行 Lambda 函數或 AWS提供的元件。您必須在處理模式中執行這些元件，而不需要任何隔離。

設定 Lambda 函數元件時，請將隔離模式設定為無容器。如需詳細資訊，請參閱 執行AWS Lambda函數。

當您部署下列任何 AWS提供的元件時，請更新每個元件的組態，以將`containerMode`參數設定為`NoContainer`。如需有關組態更新的詳細資訊，請參閱更新零組件組態。

- CloudWatch 度量
- 設備後衛
- Firehose
- 通訊協定介面卡
- Amazon SNS

AWS IoT Greengrass 碼頭軟體下載

AWS IoT Greengrass 提供一個碼頭檔案來建置一個容器映像檔，該映像檔具有安裝在 Amazon Linux 2 (x86_64) 基礎映像上的 AWS IoT Greengrass 核心軟體和相依性。您可以修改 Docker 文件中的基本映像以在不同的平台架構 AWS IoT Greengrass 上運行。

從下載碼頭文件包。GitHub

碼頭文件使用舊版本的 Greengrass。您應該更新該文件以使用所需的 Greengrass 版本。如需有關從 Docker 檔案建置 AWS IoT Greengrass 容器影像的資訊，請參閱。從碼頭文件構建AWS IoT Greengrass容器映像

選擇如何佈建 AWS 資源

當您在 Docker 容器中安裝 AWS IoT Greengrass Core 軟體時，您可以選擇是否要自動佈建 Greengrass 核心裝置操作所需的 AWS 資源，還是使用手動佈建的資源。

- 自動資源佈建 — 當您第 AWS IoT 一次執行 AWS IoT Greengrass 容器映像時，安裝程式會佈建物件、 AWS IoT 物群組、IAM AWS IoT 角色和角色別名。安裝程式也可以將本機開發工具部署到核心裝置，以便您可以使用該裝置來開發和測試自訂軟體元件。若要自動佈建這些資源，您必須提供 AWS 認證作為 Docker 映像的環境變數。

  若要使用自動佈建，您必須設定 Docker 環境變數`PROVISION=true`並掛載認證檔案，以提供您的認 AWS 證給容器。
- 手動資源佈建 — 如果您不想提供 AWS 認證給容器，則可以在執行 AWS IoT Greengrass 容器映像之前手動佈建 AWS 資源。您必須建立組態檔，將這些資源的相關資訊提供給 Docker 容器內的 AWS IoT Greengrass Core 軟體安裝程式。

  若要使用手動佈建，您必須設定 Docker 環境變數`PROVISION=false`。手動佈建是預設選項。

如需更多詳細資訊，請參閱 從碼頭文件構建AWS IoT Greengrass容器映像。

從碼頭文件構建AWS IoT Greengrass容器映像

AWS提供了一個 Docker 文件，您可以下載並使用它在 Docker 容器中運行AWS IoT Greengrass核心軟件。碼頭文件包含用於構建AWS IoT Greengrass容器映像的源代碼。

在建置AWS IoT Greengrass容器映像檔之前，您必須設定 Docker 檔案，以選取您要安裝的 AWS IoT Greengrass Core 軟體版本。您也可以設定環境變數，以選擇在安裝期間佈建資源的方式，以及自訂其他安裝選項。本節介紹如何從 Docker 文件配置和構建 AWS IoT Greengrass Docker 映像。

下載碼頭文件包

您可以從以下位置下載AWS IoT Greengrass碼頭文件包： GitHub

AWS Greengrass 泊塢工人存儲庫

下載套件之後，請將內容解壓縮至電腦上的*download-directory/*aws-greengrass-docker-*nucleus-version*資料夾。碼頭文件使用舊版本的 Greengrass。您應該更新該文件以使用所需的 Greengrass 版本。

指定AWS IoT Greengrass核心軟體版本

使用 Docker 檔案中的下列建置引數來指定您要在 Doc AWS IoT Greengrass ker 映像檔中使用的AWS IoT Greengrass核心軟體版本。默認情況下，碼頭文件使用的AWS IoT Greengrass核心軟件的最新版本。

`GREENGRASS_RELEASE_VERSION`

AWS IoT Greengrass核心軟體的版本。默認情況下，碼頭文件下載 Greengrass 核的最新可用版本。將值設定為您要下載的原子核的版本。

設定環境變數

環境變數可讓您自訂AWS IoT Greengrass核心軟體在 Docker 容器中的安裝方式。您可以通過各種方式為 D AWS IoT Greengrass ocker 映像設置環境變量。

- 若要使用相同的環境變數建立多個影像，請直接在 Docker 檔案中設定環境變數。
- 如果您使`docker run`用啟動容器，請在命令中傳遞環境變數做為引數，或在環境變數檔案中設定環境變數，然後將檔案當作引數傳遞。如需有關在 Docker 中設定環境變數的詳細資訊，請參閱 Docker 文件中的環境變數。
- 如果您使`docker-compose up`用啟動容器，請在環境變量文件中設置環境變量，然後將該文件作為參數傳遞。如需有關在撰寫中設定環境變數的詳細資訊，請參閱 Docker 文件。

您可以為 AWS IoT Greengrass Docker 映像檔設定下列環境變數。

> **Note**
>
> 請勿修改 Docker 檔案中的`TINI_KILL_PROCESS_GROUP`變數。此變數允許轉送`SIGTERM`至 PID 群組中的所有 PID，以便在 Docker 容器停止時，AWS IoT Greengrass核心軟體可以正確關閉。

`GGC_ROOT_PATH`

(選擇性) 容器內用作AWS IoT Greengrass核心軟體根目錄的資料夾路徑。

預設：`/greengrass/v2`

`PROVISION`

(選擇性) 決定AWS IoT Greengrass核心是否佈建AWS資源。

- 如果您指定`true`，AWS IoT Greengrass核心軟體會將容器映像註冊為AWS IoT物件，並佈建 Greengrass 核心裝置所需的AWS資源。AWS IoT Greengrass核心軟體會佈建AWS IoT物件 (選用) AWS IoT 物件群組、IAM 角色和AWS IoT角色別名。如需詳細資訊，請參閱 AWS IoT Greengrass在具有自動資源佈建的 Docker 容器中執行。
- 如果您指定`false`，則必須建立組態檔，以提供給 AWS IoT Greengrass Core 安裝程式，該檔案指定要使用您手動建立的AWS資源和憑證。如需詳細資訊，請參閱 AWS IoT Greengrass在具有手動資源佈建的 Docker 容器中執行。

預設：`false`

`AWS_REGION`

(選擇性) AWS IoT Greengrass Core 軟體用來擷取或建立必要AWS資源的。AWS 區域

預設：`us-east-1`。

`THING_NAME`

(選擇性) 您註冊為此核心裝置的AWS IoT物件名稱。如果您的內容不存在具有此名稱的內容AWS 帳戶，則 AWS IoT Greengrass Core 軟件將創建它。

您必須指`PROVISION=true`定套用此引數。

預設值：`GreengrassV2IotThing_`加上一個隨機的 UUID。

`THING_GROUP_NAME`

(選擇性) 您新增此核心裝置的AWS IoT物群組名稱AWS IoT如果部署以此物件群組為目標，則該群組中的這個核心裝置和其他核心裝置會在連線時收到該部署AWS IoT Greengrass。如果具有此名稱的物件群組不存在於您的中AWS 帳戶，則 AWS IoT Greengrass Core 軟體會建立該物件群組。

您必須指`PROVISION=true`定套用此引數。

`TES_ROLE_NAME`

(選擇性) 要用來取得AWS認證的 IAM 角色名稱，讓 Greengrass 核心裝置與服務互動。AWS 如果您的中不存在具有此名稱的角色AWS 帳戶，則 AWS IoT Greengrass Core 軟體會使用`GreengrassV2TokenExchangeRoleAccess`原則建立該角色。此角色無法存取託管元件成品的 S3 儲存貯體。因此，您必須在建立元件時，將權限新增至人工因素的 S3 儲存貯體和物件。如需詳細資訊，請參閱 授權核心設備與AWS服務。

預設：`GreengrassV2TokenExchangeRole`

`TES_ROLE_ALIAS_NAME`

(選擇性) 指向提供 Greengrass 核心裝置AWS登入資料的 IAM 角色的角色別名名稱。AWS IoT如果您的中不存在具有此名稱的角色別名AWS 帳戶，則 AWS IoT Greengrass Core 軟體會建立該別名，並將其指向您指定的 IAM 角色。

預設：`GreengrassV2TokenExchangeRoleAlias`

`COMPONENT_DEFAULT_USER`

(選擇性) AWS IoT Greengrass Core 軟體用來執行元件之系統使用者和群組的名稱或識別碼。指定使用者和群組，以冒號分隔。群組為選用項目。例如，您可以指定 **`ggc_user:ggc_group`** 或 **`ggc_user`**。

- 如果您以 root 身份執行，則預設為組態檔定義的使用者和群組。如果設定檔未定義使用者和群組，則預設值為`ggc_user:ggc_group`。如果`ggc_user`或`ggc_group`不存在，軟件將創建它們。
- 如果您以非 root 使用者身分執行，AWS IoT GreengrassCore 軟體會使用該使用者來執行元件。
- 如果您未指定群組，AWS IoT GreengrassCore 軟體會使用系統使用者的主要群組。

如需詳細資訊，請參閱 設定執行元件的使用者。

`DEPLOY_DEV_TOOLS`

定義是否要在容器映像檔中下載和部署 Greengrass CLI 元件。您可以使用 Greengrass CLI 在本地開發和調試組件。

> **Important**
>
> 我們建議您僅在開發環境中使用此元件，而不是在生產環境中使用。此元件可讓您存取通常在生產環境中不需要的資訊和作業。只將此元件部署到您需要的核心裝置，以遵循最低權限原則。

預設：`false`

`INIT_CONFIG`

(選擇性) 用來安裝 AWS IoT Greengrass Core 軟體的組態檔案路徑。您可以使用此選項來設定具有特定核心組態的新 Greengrass 核心裝置，或指定手動佈建的資源，例如。您必須將組態檔掛載到您在此引數中指定的路徑。

`TRUSTED_PLUGIN`

此功能適用於 v2.4.0 和更高版 Greeng rass 核組件。

(選擇性) 要載入為受信任外掛程式之 JAR 檔案的路徑。使用此選項可提供佈建外掛程式 JAR 檔案，例如透過叢集佈建或自訂佈建進行安裝。

`THING_POLICY_NAME`

此功能適用於 v2.4.0 和更高版 Greeng rass 核組件。

(選擇性) 要附加至此核心裝置AWS IoT物件憑證的AWS IoT原則名稱。如果具有此名稱的AWS IoT策略不存在於您的 AWS IoT Greengrass Core 軟件AWS 帳戶中，則會創建它。

您必須指`PROVISION=true`定套用此引數。

> **Note**
>
> 根據預設，AWS IoT GreengrassCore 軟體會建立寬容AWS IoT原則。您可以縮小此原則的範圍，或建立自訂原則，以限制使用案例的權限。如需詳細資訊，請參閱 AWS IoT Greengrass V2核心裝置的最低AWS IoT原則。

指定要安裝的相依性

AWS IoT GreengrassDockerfile 中的 RUN 指令會準備容器環境以執行AWS IoT Greengrass核心軟體安裝程式。您可以自訂AWS IoT Greengrass核心軟體安裝程式在 Docker 容器中執行之前所安裝的相依性。

建立映AWS IoT Greengrass像

使用AWS IoT Greengrass碼頭文件來構建AWS IoT Greengrass容器映像。您可以使用碼頭 CLI 或碼頭構成 CLI 來構建映像並啟動容器。您也可以使用 Docker CLI 來構建映像，然後使用 Docker 撰寫從該映像啟動容器。

Docker

1. 在主機上，執行下列命令以切換至包含已設定 Docker 檔案的目錄。

   ```
   cd download-directory/aws-greengrass-docker-nucleus-version
   ```

2. 運行以下命令從碼頭文件構建AWS IoT Greengrass容器映像。

   ```
   sudo docker build -t "platform/aws-iot-greengrass:nucleus-version" ./
   ```

Docker Compose

1. 在主機上，執行下列命令以切換至包含 Docker 檔案和撰寫檔案的目錄。

   ```
   cd download-directory/aws-greengrass-docker-nucleus-version
   ```

2. 執行下列命令，以使用撰寫檔案來建置AWS IoT Greengrass容器映像。

   ```
   docker-compose -f docker-compose.yml build
   ```

您已成功建立AWS IoT Greengrass容器映像檔。碼頭映像已安裝AWS IoT Greengrass核心軟件。您現在可以在 Docker 容器中執行AWS IoT Greengrass核心軟體。

AWS IoT Greengrass在具有自動資源佈建的 Docker 容器中執行

本教學課程說明如何使用自動佈建的AWS資源和本機開發工具，在 Docker 容器中安裝和執行 AWS IoT Greengrass Core 軟體。您可以使用此開發環境來探索 Docker 容器中的AWS IoT Greengrass功能。軟體需要AWS憑證才能佈建這些資源並部署本機開發工具。

如果您無法提供AWS認證給容器，則可以佈建核心裝置運作所需的AWS資源。您也可以將開發工具部署到核心裝置，做為開發裝置使用。這可讓您在執行容器時提供較少的裝置權限。如需詳細資訊，請參閱 AWS IoT Greengrass在具有手動資源佈建的 Docker 容器中執行。

必要條件

若要完成此自學課程，您需要下列內容。

- AWS 帳戶。如果您沒有帳戶，請參閱 設置一個 AWS 帳戶。
- 具有為 Greengrass 核心裝置佈建AWS IoT和 IAM 資源許可的 IAM 使用者。AWSAWS IoT GreengrassCore 軟體安裝程式會使用您的AWS認證來自動佈建這些資源。如需自動佈建資源的最低 IAM 政策的相關資訊，請參閱安裝程式佈建資源的最低 IAM 政策。
- AWS IoT Greengrass碼頭圖像。您可以從AWS IoT Greengrass碼頭文件構建圖像。
- 執行 Docker 容器的主機電腦必須符合下列需求：
  - 具有網際網路連線的 Linux 作業系統。
  - 碼頭引擎版本 18.09 或更高版本。
  - （可選）碼頭編寫版本 1.22 或更高版本。只有當您想要使用 Docker 撰寫 CLI 來執行 Docker 映像時，才需要碼頭構成。

設定 AWS 憑證

在此步驟中，您會在包含AWS安全性認證的主機電腦上建立認證檔案。當您執行 AWS IoT Greengrass Docker 映像時，您必須將包含此認證檔案的資料夾掛載到 Docker 容器/root/.aws/中。安AWS IoT Greengrass裝程式會使用這些認證在您的AWS 帳戶. 如需安裝程式自動佈建資源所需的最低 IAM 政策的相關資訊，請參閱安裝程式佈建資源的最低 IAM 政策。

1. 擷取下列其中一項。
   - IAM 使用者的長期登入資料。如需如何擷取長期登入資料的詳細資訊，請參閱《IAM 使用者指南》中的「管理 IAM 使用者存取金鑰」。
   - (建議使用) IAM 角色的臨時登入資料。如需如何擷取臨時登入資料的詳細資訊，請參閱《IAM 使用者指南》AWS CLI中的〈使用臨時安全登入資料〉。
2. 建立放置認證檔案的資料夾。

   ```
   mkdir ./greengrass-v2-credentials
   ```

3. 使用文字編輯器建立在./greengrass-v2-credentials資料夾credentials中命名的規劃檔。

   例如，您可以運行以下命令來使用 GNU nano 創建credentials文件。

```
nano ./greengrass-v2-credentials/credentials
```

4. 以下列格式將您的AWS認證新增至credentials檔案。

```
[default]
aws_access_key_id = AKIAIOSFODNN7EXAMPLE
aws_secret_access_key = wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
aws_session_token
 = AQoEXAMPLEH4aoAH0gNCAPy...truncated...zrkuWJOgQs8IZZaIv2BXIa2R4Olgk
```

僅包含aws_session_token臨時登入資料。

> **Important**
>
> 啟動AWS IoT Greengrass容器之後，從主機電腦移除認證檔案。如果您不刪除憑據文件，則您的AWS憑據將保持掛載在容器中。如需詳細資訊，請參閱 在容器中執行AWS IoT Greengrass核心軟體。

建立環境檔案

本教程使用環境文件來設置將傳遞給 Docker 容器內AWS IoT Greengrass核心軟件安裝程序的環境變量。您也可以在docker run命令中使用-e或--env引數來設定 Docker 容器中的環境變數，或者您也可以在docker-compose.yml檔案中的environment區塊中設定變數。

1. 使用文字編輯器建立名為的環境檔案.env。

   例如，在基於 Linux 的系統上，您可以運行以下命令來使用 GNU nano 在當前目錄.env中創建。

```
nano .env
```

2. 將下列內容複製到檔案中。

```
GGC_ROOT_PATH=/greengrass/v2
AWS_REGION=region
PROVISION=true
THING_NAME=MyGreengrassCore
THING_GROUP_NAME=MyGreengrassCoreGroup
TES_ROLE_NAME=GreengrassV2TokenExchangeRole
```

```
TES_ROLE_ALIAS_NAME=GreengrassCoreTokenExchangeRoleAlias
COMPONENT_DEFAULT_USER=ggc_user:ggc_group
```

然後，取代下列值。

- */greengrass/v2*。 您要用於安裝的 Greengrass 根資料夾。您可以使用GGC_ROOT環境變數來設定此值。
- *##*。您建立資源的AWS 區域位置。
- *MyGreengrassCore*。AWS IoT 物件的名稱。如果東西不存在，安裝程序將創建它。安裝程式會下載憑證以進行驗證。AWS IoT
- *MyGreengrassCoreGroup*。AWS IoT物件群組的名稱。如果物件群組不存在，安裝程式會建立該物件並將該物件新增至其中。如果物群組存在且具有使用中部署，則核心裝置會下載並執行部署指定的軟體。
- *##### V2 TokenExchangeRole*。取代為允許 Greengrass 核心裝置取得臨時登入資料的 IAM 權杖交換角色名稱。AWS如果角色不存在，安裝程式會建立該角色，並建立並附加名為 *Greengr TokenExchangeRole* assv2 存取的原則。如需詳細資訊，請參閱 [授權核心設備與AWS服務](#)。
- *GreengrassCoreTokenExchangeRoleAlias*。權杖交換角色別名。如果角色別名不存在，安裝程式會建立它，並將其指向您指定的 IAM Token 交換角色。如需詳細資訊，請參閱

> **Note**
>
> 您可以將DEPLOY_DEV_TOOLS環境變數設定`true`為部署 Greengrass CLI 元件，這可讓您在 Docker 容器內部開發自訂元件。我們建議您僅在開發環境中使用此元件，而非生產環境。此元件可讓您存取通常在生產環境中不需要的資訊和作業。只將此元件部署到您需要的核心裝置，以遵循最低權限原則。

在容器中執行AWS IoT Greengrass核心軟體

本教學課程說明如何啟動您在 Docker 容器中建置的 Docker 映像檔。您可以使用碼頭 CLI 或碼頭構成 CLI 在 Docker 容器中執行AWS IoT Greengrass核心軟體映像。

Docker

1. 執行下列命令以啟動 Docker 容器。

```
docker run --rm --init -it --name docker-image \
 -v path/to/greengrass-v2-credentials:/root/.aws/:ro \
 --env-file .env \
 -p 8883 \
 your-container-image:version
```

此示例命令使用以下參數進 docker 運行：

- `--rm`。 當容器離開時將其清理。
- `--init`。 在容器中使用初始化過程。

> **Note**
>
> 當您停止 Docker 容器時，需要`--init`參數才能關閉AWS IoT Greengrass核心軟體。

- `-it`。 (選擇性) 在前景中以互動程序的形式執行 Docker 容器。您可以將其替換為`-d`參數，以分離模式運行 Docker 容器。有關更多信息，請參閱 Docker 文檔中的分離與前景。
- `--name`。 運行一個名為的容器 `aws-iot-greengrass`
- `-v`。 將磁碟區掛載到 Docker 容器中，以便在容器內AWS IoT Greengrass執行組態檔案和憑證檔案。
- `--env-file`。 (選擇性) 指定環境檔案，以設定要傳遞至 Docker 容器內AWS IoT Greengrass核心軟體安裝程式的環境變數。只有當您建立環境檔案來設定環境變數時，才需要此引數。如果您沒有創建環境文件，則可以使用`--env`參數直接在 Docker run 命令中設置環境變量。
- `-p`。 (選擇性) 將 8883 容器連接埠發佈至主機。如果您要透過 MQTT 進行連線和通訊，則需要此引數，因為 MQTT 流量AWS IoT Greengrass使用連接埠 8883。若要開啟其他連接埠，請使用其他`-p`引數。

> **Note**
>
> 若要以更高的安全性執行 Docker 容器，您可以使用`--cap-drop`和`--cap-add`引數來選擇性地啟用容器的 Linux 功能。如需詳細資訊，請參閱 Docker 文件中的執行階段權限和 Linux 功能。

2. 從主機裝置`./greengrass-v2-credentials`上移除認證。

```
rm -rf ./greengrass-v2-credentials
```

> **Important**
>
> 您正在移除這些認證，因為它們提供了核心裝置僅在安裝期間所需的廣泛權限。如果您不移除這些認證，Greengrass 元件和容器中執行的其他處理序可以存取它們。如果您需要為 Greengrass 元件提供AWS認證，請使用權杖交換服務。如需詳細資訊，請參閱 與AWS服務互動。

Docker Compose

1. 使用文字編輯器建立名為`docker-compose.yml`的 Docker 撰寫檔案。

   例如，在基於 Linux 的系統上，您可以運行以下命令來使用 GNU nano 在當前目錄`docker-compose.yml`中創建。

   ```
   nano docker-compose.yml
   ```

   > **Note**
   >
   > 您也可以從GitHub中下載並使用最新版本的AWS提供的撰寫檔案。

2. 將下列內容新增至撰寫檔案。您的檔案看起來應該如下列範例：將`####`替換為 docker 映像的名稱。

   ```
   version: '3.7'

   services:
     greengrass:
       init: true
       container_name: aws-iot-greengrass
       image: docker-image
       volumes:
         - ./greengrass-v2-credentials:/root/.aws/:ro
       env_file: .env
       ports:
         - "8883:8883"
   ```

此範例中的下列參數是選用的：

- `ports`將 8883 容器連接埠發佈至主機。如果您要透過 MQTT 進行連線和通訊，則需要此參數，因為 MQTT 流量AWS IoT Greengrass使用連接埠 8883。
- `env_file`指定環境檔案，以設定要傳遞給 Docker 容器內AWS IoT Greengrass核心軟體安裝程式的環境變數。只有當您建立環境檔案來設定環境變數時，才需要此參數。如果您未建立環境檔案，則可以使用環境參數直接在 Compose 檔案中設定變數。

> **Note**
>
> 若要以更高的安全性執行 Docker 容器，您可以在撰寫檔案`cap_add`中使用`cap_drop`和來選擇性地啟用容器的 Linux 功能。如需詳細資訊，請參閱 Docker 文件中的執行階段權限和 Linux 功能。

3. 執行下列命令以啟動 Docker 容器。

```
docker-compose -f docker-compose.yml up
```

4. 從主機裝置`./greengrass-v2-credentials`上移除認證。

```
rm -rf ./greengrass-v2-credentials
```

> **Important**
>
> 您正在移除這些認證，因為它們提供了核心裝置僅在安裝期間所需的廣泛權限。如果您不移除這些認證，Greengrass 元件和容器中執行的其他處理序可以存取它們。如果您需要為 Greengrass 元件提供AWS認證，請使用權杖交換服務。如需詳細資訊，請參閱 與AWS服務互動。

後續步驟

AWS IoT Greengrass核心軟件現在正在 Docker 容器中運行。執行下列命令以擷取目前執行中容器的容器 ID。

```
docker ps
```

然後，您可以執行下列命令來存取容器，並探索在容器內執行的 AWS IoT Greengrass Core 軟體。

```
docker exec -it container-id /bin/bash
```

如需有關建立簡單元件的資訊，請參閱步驟 4：在設備上開發和測試組件中的 教學課程：AWS IoT Greengrass V2 入門

> **Note**
>
> 當您使用 `docker exec` Docker 容器內執行命令時，這些命令不會記錄在 Docker 記錄中。要在 Docker 日誌中記錄命令，請將交互式外殼附加到 Docker 容器。如需詳細資訊，請參閱 將交互式外殼附加到 Docker 容器。

AWS IoT Greengrass核心記錄檔會呼叫`greengrass.log`並位於中*/greengrass/v2*/`logs`。元件記錄檔也位於相同的目錄中。若要將 Greengrass 記錄檔複製到主機上的暫存目錄，請執行下列命令：

```
docker cp container-id:/greengrass/v2/logs /tmp/logs
```

如果您想要在容器結束或移除之後保留記錄檔，建議您只將*/greengrass/v2*/`logs`目錄繫結掛載到主機上的暫存記錄目錄，而不是掛載整個 Greengrass 目錄。如需詳細資訊，請參閱 在碼頭容器之外保存 Greengrass 日誌。

若要停止執行中的 AWS IoT Greengrass Docker 容器，請執行`docker stop`或`docker-compose -f docker-compose.yml stop`。此動作會傳送`SIGTERM`至 Greengrass 處理程序，並關閉在容器中啟動的所有相關聯處理序。Docker 容器使用`docker-init`可執行文件初始化為進程 PID 1，這有助於刪除任何剩餘的殭屍進程。有關更多信息，請參閱 Docker 文檔中的指定初始化進程。

如需疑難排解 Docker 容器AWS IoT Greengrass中執行問題的相關資訊，請參閱Docker 容器中 AWS IoT Greengrass 的疑難排解。

AWS IoT Greengrass在具有手動資源佈建的 Docker 容器中執行

本教學課程說明如何使用手動佈建的AWS資源在 Docker 容器中安裝和執行 AWS IoT Greengrass Core 軟體。

主題

- 必要條件

- 擷取AWS IoT端點
- 建立物AWS IoT件
- 建立物件憑證
- 配置物憑證
- 建立權杖交換角色
- 將憑證下載至裝置
- 建立組態檔案
- 建立環境檔案
- 在容器中執行AWS IoT Greengrass核心軟體
- 後續步驟

必要條件

為了完成本教學，您需要以下項目：

- AWS 帳戶。如果您沒有帳戶，請參閱 設置一個 AWS 帳戶。
- AWS IoT Greengrass碼頭圖像。您可以從AWS IoT Greengrass碼頭文件構建圖像。
- 您執行 Docker 容器的主機電腦必須符合下列需求：
  - 具有網際網路連線的 Linux 作業系統。
  - 碼頭引擎版本 18.09 或更高版本。
  - （可選）碼頭編寫版本 1.22 或更高版本。只有當您想要使用 Docker 撰寫 CLI 來執行 Docker 映像時，才需要碼頭構成。

擷取AWS IoT端點

取得您的AWS IoT端點AWS 帳戶，並儲存以供稍後使用。您的裝置會使用這些端點連線至AWS IoT。請執行下列操作：

1. 取得適AWS IoT用於您的AWS 帳戶.

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

   如果要求成功，回應看起來類似下列範例。

```
{
  "endpointAddress": "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
}
```

2. 取AWS IoT得您的AWS 帳戶.

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

如果要求成功，回應看起來類似下列範例。

```
{
  "endpointAddress": "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
}
```

建立物AWS IoT件

AWS IoT東西代表連接到的設備和邏輯實體AWS IoT。Greengrass 核心設備是事情。AWS IoT當您將裝置註冊為AWS IoT物件時，該裝置可以使用數位憑證來進行驗證AWS。

在本節中，您會建立代表您裝置的AWS IoT物件。

若要建立AWS IoT物件

1. 為您的裝置建立項AWS IoT目。在您的開發電腦上，執行下列命令。

   - 以要*MyGreengrassCore*使用的物件名稱取代。這個名字也是 Greengrass 核心裝置的名稱。

   > **Note**
   >
   > 物件名稱不能包含冒號 (:) 字元。

```
aws iot create-thing --thing-name MyGreengrassCore
```

如果要求成功，回應看起來類似下列範例。

```
{
```

```
  "thingName": "MyGreengrassCore",
  "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
  "thingId": "8cb4b6cd-268e-495d-b5b9-1713d71dbf42"
}
```

2. (選擇性) 將AWS IoT物件新增至新的或現有的物件群組。您可以使用物件群組來管理 Greengrass 核心裝置的叢集。將軟體元件部署到裝置時，您可以鎖定個別裝置或裝置群組。您可以將裝置新增至具有作用中 Greengrass 部署的物群組，以將該物件群組的軟體元件部署到裝置。請執行下列操作：

   a. (選擇性) 建立AWS IoT物件群組。

      - 以要建立的物件群組名稱取*MyGreengrassCoreGroup*代。

      > **Note**
      >
      > 物件群組名稱不能包含冒號 (:) 字元。

      ```
      aws iot create-thing-group --thing-group-name MyGreengrassCoreGroup
      ```

      如果要求成功，回應看起來類似下列範例。

      ```
      {
        "thingGroupName": "MyGreengrassCoreGroup",
        "thingGroupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/
      MyGreengrassCoreGroup",
        "thingGroupId": "4df721e1-ff9f-4f97-92dd-02db4e3f03aa"
      }
      ```

   b. 將AWS IoT物件新增至物件群組。

      - 替換*MyGreengrassCore*為您的AWS IoT事物的名稱。
      - *MyGreengrassCoreGroup*以物件群組的名稱取代。

      ```
      aws iot add-thing-to-thing-group --thing-name MyGreengrassCore --thing-group-
      name MyGreengrassCoreGroup
      ```

      如果請求成功，命令沒有任何輸出。

建立物件憑證

當您將裝置註冊為AWS IoT物件時，該裝置可以使用數位憑證來進行驗證AWS。此憑證可讓裝置與AWS IoT和通訊AWS IoT Greengrass。

在本節中，您會建立並下載裝置可用來連線的憑證AWS。

若要建立物憑證

1. 建立下載AWS IoT物件憑證的資料夾。

```
mkdir greengrass-v2-certs
```

2. 建立並下載AWS IoT物件的憑證。

```
aws iot create-keys-and-certificate --set-as-active --certificate-pem-outfile
 greengrass-v2-certs/device.pem.crt --public-key-outfile greengrass-v2-certs/
public.pem.key --private-key-outfile greengrass-v2-certs/private.pem.key
```

如果要求成功，回應看起來類似下列範例。

```
{
  "certificateArn": "arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
  "certificateId":
 "aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
  "certificatePem": "-----BEGIN CERTIFICATE-----
MIICiTCCAfICCQD6m7oRw0uXOjANBgkqhkiG9w
 0BAQUFADCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZ
 WF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIw
 EAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5
 jb20wHhcNMTEwNDI1MjA0NTIxWhcNMTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBh
 MCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBb
 WF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMx
 HzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5jb20wgZ8wDQYJKoZIhvcNAQE
 BBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ21uUSfwfEvySWtC2XADZ4nB+BLYgVI
 k60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9TrDHudUZg3qX4waLG5M43q7Wgc/MbQ
 ITxOUSQv7c7ugFFDzQGBzZswY6786m86gpEIbb3OhjZnzcvQAaRHhdlQWIMm2nr
 AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4nUhVVxYUntneD9+h8Mg9q6q+auN
 KyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0FkbFFBjvSfpJIlJ00zbhNYS5f6Guo
 EDmFJl0ZxBHjJnyp378OD8uTs7fLvjx79LjSTbNYiytVbZPQUQ5Yaxu2jXnimvw
 3rrszlaEXAMPLE=
```

```
-----END CERTIFICATE-----",
  "keyPair": {
    "PublicKey": "-----BEGIN PUBLIC KEY-----\
MIIBIjANBgkqhkEXAMPLEQEFAAOCAQ8AMIIBCgKCAQEAEXAMPLE1nnyJwKSMHw4h\
MMEXAMPLEuuN/dMAS3fyce8DW/4+EXAMPLEyjmoF/YVF/gHr99VEEXAMPLE5VF13\
59VK7cEXAMPLE67GK+y+jikqXOgHh/xJTwo
+sGpWEXAMPLEDz18xOd2ka4tCzuWEXAMPLEahJbYkCPUBSU8opVkR7qkEXAMPLE1DR6sx2HocliOOLtu6Fkw91swQWE
\\GB3ZPrNh0PzQYvjUStZeccyNCx2EXAMPLEvp9mQOUXP6plfgxwKRX2fEXAMPLEDa\
hJLXkX3rHU2xbxJSq7D+XEXAMPLEcw+LyFhI5mgFRl88eGdsAEXAMPLElnI9EesG\
FQIDAQAB\
-----END PUBLIC KEY-----\
",
    "PrivateKey": "-----BEGIN RSA PRIVATE KEY-----\
key omitted for security reasons\
-----END RSA PRIVATE KEY-----\
"
  }
}
```

儲存憑證的 Amazon 資源名稱 (ARN)，以便稍後用來設定憑證。

配置物憑證

將物憑證附加至您先前建立的AWS IoT物件，並將AWS IoT原則新增至憑證以定義核心裝置的AWS IoT權限。

若要設定物件的憑證

1. 將憑證附加至物AWS IoT件。

   - 替換*MyGreengrassCore*為您的AWS IoT事物的名稱。
   - 將憑證 Amazon 資源名稱 (ARN) 取代為您在上一個步驟中建立的憑證的 ARN。

   ```
   aws iot attach-thing-principal --thing-name MyGreengrassCore
    --principal arn:aws:iot:us-west-2:123456789012:cert/
   aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
   ```

   如果請求成功，命令沒有任何輸出。

2. 建立並附加AWS IoT定義 Greengrass 核心裝置AWS IoT權限的原則。下列原則允許存取所有 MQTT 主題和 Greengrass 作業，因此您的裝置可以處理需要新 Greengrass 作業的自訂應用程式和 future 變更。您可以根據您的使用案例限制此政策。如需詳細資訊，請參閱 AWS IoT Greengrass V2核心裝置的最低AWS IoT原則。

   如果您之前已設定過 Greengrass 核心裝置，則可以附加其AWS IoT原則，而不是建立新的原則。

   請執行下列操作：

   a. 建立包含 Greengrass 核心裝置所需之AWS IoT原則文件的檔案。

      例如，在 Linux 系統上，您可以執行下列命令來使用 GNU nano 建立檔案。

      ```
      nano greengrass-v2-iot-policy.json
      ```

      將下列 JSON 複製到檔案中。

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

   b. 從AWS IoT策略文件建立策略。

      - 以要建*#########ThingPolicy####*。

```
aws iot create-policy --policy-name GreengrassV2IoTThingPolicy --policy-
document file://greengrass-v2-iot-policy.json
```

如果要求成功，回應看起來類似下列範例。

```
{
  "policyName": "GreengrassV2IoTThingPolicy",
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy",
  "policyDocument": "{
    \\"Version\\": \\"2012-10-17\\",
    \\"Statement\\": [
      {
        \\"Effect\\": \\"Allow\\",
        \\"Action\\": [
          \\"iot:Publish\\",
          \\"iot:Subscribe\\",
          \\"iot:Receive\\",
          \\"iot:Connect\\",
          \\"greengrass:*\\"
        ],
        \\"Resource\\": [
          \\"*\\"
        ]
      }
    ]
  }",
  "policyVersionId": "1"
}
```

c. 將AWS IoT原則附加至物AWS IoT件的憑證。

- 以要附*#########ThingPolicy####*。
- 將目標 ARN 取代為物件憑證的 AWS IoT ARN。

```
aws iot attach-policy --policy-name GreengrassV2IoTThingPolicy
 --target arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
```

如果請求成功，命令沒有任何輸出。

建立權杖交換角色

Greengrass 核心裝置使用 IAM 服務角色 (稱為權杖交換角色) 來授權對服務的呼叫。AWS裝置使用AWS IoT登入資料提供者取得此角色的臨時AWS登入資料，以便裝置與之互動AWS IoT、將日誌傳送到 Amazon CloudWatch Logs，以及從 Amazon S3 下載自訂元件成品。如需詳細資訊，請參閱 授權核心設備與AWS服務。

您可以使用AWS IoT角色別名來設定 Greengrass 核心裝置的權杖交換角色。角色別名可讓您變更裝置的 Token 交換角色，但保持裝置設定不變。如需詳細資訊，請參閱AWS IoT Core開發人員指南中的授權直接呼叫AWS服務。

在本節中，您會建立權杖交換 IAM 角色和指向該AWS IoT角色的角色別名。如果您已經設置了 Greengrass 核心設備，則可以使用其令牌交換角色和角色別名，而不是創建新的。然後，您將設備的AWS IoT東西配置為使用該角色和別名。

若要建立權杖交換 IAM 角色

1. 建立 IAM 角色，您的裝置可用作權杖交換角色。請執行下列操作：

   a. 建立包含 Token 交換角色所需之信任原則文件的檔案。

      例如，在 Linux 系統上，您可以執行下列命令來使用 GNU nano 建立檔案。

      ```
      nano device-role-trust-policy.json
      ```

      將下列 JSON 複製到檔案中。

      ```
      {
        "Version": "2012-10-17",
        "Statement": [
          {
            "Effect": "Allow",
            "Principal": {
              "Service": "credentials.iot.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
          }
        ]
      ```

```
}
```

b. 使用信任原則文件建立權杖交換角色。

- TokenExchangeRole以要建立的身分與存取權管理員角色的名稱取代 *GreenGrassv2*。

```
aws iam create-role --role-name GreengrassV2TokenExchangeRole --assume-role-
policy-document file://device-role-trust-policy.json
```

如果要求成功，回應看起來類似下列範例。

```
{
  "Role": {
    "Path": "/",
    "RoleName": "GreengrassV2TokenExchangeRole",
    "RoleId": "AROAZ2YMUHYHK5OKM77FB",
    "Arn": "arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole",
    "CreateDate": "2021-02-06T00:13:29+00:00",
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": "credentials.iot.amazonaws.com"
          },
          "Action": "sts:AssumeRole"
        }
      ]
    }
  }
```

c. 建立包含權杖交換角色所需之存取原則文件的檔案。

例如，在 Linux 系統上，您可以執行下列命令來使用 GNU nano 建立檔案。

```
nano device-role-access-policy.json
```

將下列 JSON 複製到檔案中。

```
{
```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    }
  ]
}
```

> **Note**
>
> 此存取政策不允許存取 S3 儲存貯體中的元件成品。若要在 Amazon S3 中部署定義成品的自訂元件，您必須向角色新增許可，以允許核心裝置擷取元件成品。如需詳細資訊，請參閱 允許存取 S3 儲存貯體的存取。
> 如果您還沒有適用於元件成品的 S3 儲存貯體，您可以在稍後建立儲存貯體後新增這些許可。

d. 從政策文件建立 IAM 政策。

- TokenExchangeRoleAccess以要建立的身分與存取權管理政策的名稱取代 *GreenGrassv2*。

```
aws iam create-policy --policy-name GreengrassV2TokenExchangeRoleAccess --
policy-document file://device-role-access-policy.json
```

如果要求成功，回應看起來類似下列範例。

```
{
  "Policy": {
    "PolicyName": "GreengrassV2TokenExchangeRoleAccess",
    "PolicyId": "ANPAZ2YMUHYHACI7C5Z66",
    "Arn": "arn:aws:iam::123456789012:policy/
GreengrassV2TokenExchangeRoleAccess",
```

```
    "Path": "/",
    "DefaultVersionId": "v1",
    "AttachmentCount": 0,
    "PermissionsBoundaryUsageCount": 0,
    "IsAttachable": true,
    "CreateDate": "2021-02-06T00:37:17+00:00",
    "UpdateDate": "2021-02-06T00:37:17+00:00"
  }
}
```

e. 將 IAM 政策附加到權杖交換角色。

- 以 IAM 角*#######TokenExchangeRole# V2*。
- 將政策 ARN 取代為您在上一個步驟中建立的 IAM 政策的 ARN。

```
aws iam attach-role-policy --role-name GreengrassV2TokenExchangeRole --policy-
arn arn:aws:iam::123456789012:policy/GreengrassV2TokenExchangeRoleAccess
```

如果請求成功，命令沒有任何輸出。

2. 建立指向權杖交換AWS IoT角色的角色別名。

- 以要建立之角色別名的名稱取*GreengrassCoreTokenExchangeRoleAlias*代。
- 將角色 ARN 取代為您在上一個步驟中建立的 IAM 角色的 ARN。

```
aws iot create-role-alias --role-alias GreengrassCoreTokenExchangeRoleAlias --role-
arn arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole
```

如果要求成功，回應看起來類似下列範例。

```
{
  "roleAlias": "GreengrassCoreTokenExchangeRoleAlias",
  "roleAliasArn": "arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias"
}
```

> **Note**
>
> 若要建立角色別名，您必須擁有權限才能將權杖交換 IAM 角色傳遞給AWS IoT。如果您在嘗試建立角色別名時收到錯誤訊息，請檢查您的AWS使用者是否具有此權限。如需詳細資訊，請參閱《使用指南》中的授與使用AWS Identity and Access Management者將角色傳遞給AWS服務的權限。

3. 建立並附加AWS IoT原則，讓您的 Greengrass 核心裝置使用角色別名來承擔權杖交換角色。如果您之前已經設定過 Greengrass 核心裝置，則可以附加其角色別名AWS IoT原則，而不是建立新的別名原則。請執行下列操作：

   a. (選擇性) 建立包含角色別名所需之AWS IoT原則文件的檔案。

      例如，在 Linux 系統上，您可以執行下列命令來使用 GNU nano 建立檔案。

      ```
      nano greengrass-v2-iot-role-alias-policy.json
      ```

      將下列 JSON 複製到檔案中。

      - 將資源 ARN 取代為角色別名的 ARN。

      ```
      {
        "Version":"2012-10-17",
        "Statement": [
          {
            "Effect": "Allow",
            "Action": "iot:AssumeRoleWithCertificate",
            "Resource": "arn:aws:iot:us-west-2:123456789012:rolealias/
      GreengrassCoreTokenExchangeRoleAlias"
          }
        ]
      }
      ```

   b. 從AWS IoT策略文件建立策略。

      - 以要建立的AWS IoT策略名稱取*GreengrassCoreTokenExchangeRoleAliasPolicy*代。

```
aws iot create-policy --policy-name GreengrassCoreTokenExchangeRoleAliasPolicy
 --policy-document file://greengrass-v2-iot-role-alias-policy.json
```

如果要求成功，回應看起來類似下列範例。

```
{
  "policyName": "GreengrassCoreTokenExchangeRoleAliasPolicy",
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassCoreTokenExchangeRoleAliasPolicy",
  "policyDocument": "{
    \\"Version\\":\\"2012-10-17\\",
    \\"Statement\\": [
      {
        \\"Effect\\": \\"Allow\\",
        \\"Action\\": \\"iot:AssumeRoleWithCertificate\\",
        \\"Resource\\": \\"arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias\\"
      }
    ]
  }",
  "policyVersionId": "1"
}
```

c. 將AWS IoT原則附加至物AWS IoT件的憑證。

- 以角*GreengrassCoreTokenExchangeRoleAliasPolicy*色別名AWS IoT原則的名稱取代。
- 將目標 ARN 取代為物件憑證的 AWS IoT ARN。

```
aws iot attach-policy --policy-name GreengrassCoreTokenExchangeRoleAliasPolicy
 --target arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
```

如果請求成功，命令沒有任何輸出。

將憑證下載至裝置

之前，您已將裝置的憑證下載到開發電腦。在本節中，您將下載 Amazon 根憑證授權單位 (CA) 憑證。然後，如果您打算在與開發電腦不同的電腦上執行 Docker 中的 AWS IoT Greengrass Core 軟體，您可以將憑證複製到該主機電腦。AWS IoT Greengrass核心軟體會使用這些憑證來連線至AWS IoT雲端服務。

將憑證下載到裝置

1. 在開發電腦上，下載 Amazon 根憑證授權單位 (CA) 憑證。 AWS IoT憑證預設會與 Amazon 的根 CA 憑證相關聯。

   Linux or Unix

   ```
   sudo curl -o ./greengrass-v2-certs/AmazonRootCA1.pem https://
   www.amazontrust.com/repository/AmazonRootCA1.pem
   ```

   Windows Command Prompt (CMD)

   ```
   curl -o .\greengrass-v2-certs\AmazonRootCA1.pem https://www.amazontrust.com/
   repository/AmazonRootCA1.pem
   ```

   PowerShell

   ```
   iwr -Uri https://www.amazontrust.com/repository/AmazonRootCA1.pem -OutFile .
   \greengrass-v2-certs\AmazonRootCA1.pem
   ```

2. 如果您打算在與開發電腦不同的裝置上執行 Docker 中的 AWS IoT Greengrass Core 軟體，請將憑證複製到主機電腦。如果在開發電腦和主機電腦上啟用了 SSH 和 SCP，您可以使用開發電腦上的`scp`指令來傳輸憑證。*`device-ip-address`*以主機的 IP 位址取代。

   ```
   scp -r greengrass-v2-certs/ device-ip-address:~
   ```

建立組態檔案

1. 在主機電腦上，建立放置組態檔的資料夾。

   ```
   mkdir ./greengrass-v2-config
   ```

2. 使用文字編輯器建立在./greengrass-v2-config資料夾config.yaml中命名的規劃檔。

   例如，您可以執行下列命令來使用 GNU nano 來建立. config.yaml

   ```
   nano ./greengrass-v2-config/config.yaml
   ```

3. 將下列 YAML 內容複製到檔案中。此部分配置文件指定系統參數和 Greengrass 核參數。

   ```
   ---
   system:
     certificateFilePath: "/tmp/certs/device.pem.crt"
     privateKeyPath: "/tmp/certs/private.pem.key"
     rootCaPath: "/tmp/certs/AmazonRootCA1.pem"
     rootpath: "/greengrass/v2"
     thingName: "MyGreengrassCore"
   services:
     aws.greengrass.Nucleus:
       componentType: "NUCLEUS"
       version: "nucleus-version"
       configuration:
         awsRegion: "region"
         iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
         iotDataEndpoint: "device-data-prefix-ats.iot.region.amazonaws.com"
         iotCredEndpoint: "device-credentials-prefix.credentials.region.amazonaws.com"
   ```

   然後，取代下列值：

   - */tmp/ ##*。Docker 容器中的目錄，當您啟動容器時，您可以掛載下載的憑證。
   - */greengrass/v2*。 您要用於安裝的 Greengrass 根資料夾。您可以使用GGC_ROOT環境變數來設定此值。
   - *MyGreengrassCore*。AWS IoT 物件的名稱。
   - *###*。要安裝的AWS IoT Greengrass核心軟體版本。此值必須與您下載的 Docker 映像檔或 Docker 檔案的版本相符。如果您下載了帶有latest標籤的 Greengrass 碼頭圖像，請使用**docker inspect *image-id***查看映像版本。
   - *##*。您建立AWS IoT資源的AWS 區域位置。您也必須為環境檔案中的AWS_REGION環境變數指定相同的值。
   - *GreengrassCoreTokenExchangeRoleAlias*。權杖交換角色別名。
   - *device-data-prefix*。AWS IoT資料端點的前置詞。
   - *device-credentials-prefix*。AWS IoT憑證端點的前置詞。

建立環境檔案

本教程使用環境文件來設置將傳遞給 Docker 容器內AWS IoT Greengrass核心軟件安裝程序的環境變量。您也可以在`docker run`命令中使用`-e`或`--env`引數來設定 Docker 容器中的環境變數，或者您也可以在`docker-compose.yml`檔案中的`environment`區塊中設定變數。

1. 使用文字編輯器建立名為的環境檔案`.env`。

   例如，在基於 Linux 的系統上，您可以運行以下命令來使用 GNU nano 在當前目錄`.env`中創建。

   ```
   nano .env
   ```

2. 將下列內容複製到檔案中。

   ```
   GGC_ROOT_PATH=/greengrass/v2
   AWS_REGION=region
   PROVISION=false
   COMPONENT_DEFAULT_USER=ggc_user:ggc_group
   INIT_CONFIG=/tmp/config/config.yaml
   ```

   然後，取代下列值。

   - */greengrass/v2*。 用來安裝 AWS IoT Greengrass Core 軟體的根資料夾路徑。
   - *##*。您建立AWS IoT資源的AWS 區域位置。您必須為組態檔案中的`awsRegion`組態參數指定相同的值。
   - */tmp /##/*。啟動 Docker 容器時掛載配置檔案的資料夾。

   > **Note**
   >
   > 您可以將`DEPLOY_DEV_TOOLS`環境變數設定`true`為部署 Greengrass CLI 元件，這可讓您在 Docker 容器內部開發自訂元件。我們建議您僅在開發環境中使用此元件，而不是在生產環境中使用。此元件可讓您存取通常在生產環境中不需要的資訊和作業。只將此元件部署到您需要的核心裝置，以遵循最低權限原則。

在容器中執行AWS IoT Greengrass核心軟體

本教學課程說明如何啟動您在 Docker 容器中建置的 Docker 映像檔。您可以使用碼頭 CLI 或碼頭構成 CLI 在 Docker 容器中執行AWS IoT Greengrass核心軟體映像。

Docker

- 本教學課程說明如何啟動您在 Docker 容器中建置的 Docker 映像檔。

```
docker run --rm --init -it --name docker-image \
 -v path/to/greengrass-v2-config:/tmp/config/:ro \
 -v path/to/greengrass-v2-certs:/tmp/certs:ro \
 --env-file .env \
 -p 8883 \
 your-container-image:version
```

此示例命令使用以下參數進 docker 運行：

- `--rm`。 當容器離開時將其清理。
- `--init`。 在容器中使用初始化過程。

> **Note**
>
> 當您停止 Docker 容器時，需要`--init`參數才能關閉AWS IoT Greengrass核心軟體。

- `-it`。 (選擇性) 在前景中以互動程序的形式執行 Docker 容器。您可以將其替換為`-d`參數，以分離模式運行 Docker 容器。有關更多信息，請參閱 Docker 文檔中的分離與前景。
- `--name`。 運行一個名為的容器 `aws-iot-greengrass`
- `-v`。 將磁碟區掛載到 Docker 容器中，以便在容器內AWS IoT Greengrass執行組態檔案和憑證檔案。
- `--env-file`。 (選擇性) 指定環境檔案，以設定要傳遞至 Docker 容器內AWS IoT Greengrass核心軟體安裝程式的環境變數。只有當您建立環境檔案來設定環境變數時，才需要此引數。如果您沒有創建環境文件，則可以使用`--env`參數直接在 Docker run 命令中設置環境變量。
- `-p`。 (選擇性) 將 8883 容器連接埠發佈至主機。如果您要透過 MQTT 進行連線和通訊，則需要此引數，因為 MQTT 流量AWS IoT Greengrass使用連接埠 8883。若要開啟其他連接埠，請使用其他`-p`引數。

> **Note**
>
> 若要以更高的安全性執行 Docker 容器，您可以使用`--cap-drop`和`--cap-add`引數來選擇性地啟用容器的 Linux 功能。如需詳細資訊，請參閱 Docker 文件中的執行階段權限和 Linux 功能。

Docker Compose

1. 使用文字編輯器建立名為`docker-compose.yml`的 Docker 撰寫檔案。

   例如，在基於 Linux 的系統上，您可以運行以下命令來使用 GNU nano 在當前目錄`docker-compose.yml`中創建。

   ```
   nano docker-compose.yml
   ```

   > **Note**
   >
   > 您也可以從GitHub中下載並使用最新版本的AWS提供的撰寫檔案。

2. 將下列內容新增至撰寫檔案。您的檔案看起來應該如下列範例：替換：版本*your-container-name### Doc* ker 映像的名稱。

   ```
   version: '3.7'

   services:
     greengrass:
       init: true
       build:
         context: .
       container_name: aws-iot-greengrass
       image: your-container-name:version
       volumes:
         - /path/to/greengrass-v2-config:/tmp/config/:ro
         - /path/to/greengrass-v2-certs:/tmp/certs:ro
       env_file: .env
       ports:
         - "8883:8883"
   ```

此範例中的下列參數是選用的：

- `ports`將 8883 容器連接埠發佈至主機。如果您要透過 MQTT 進行連線和通訊，則需要此參數，因為 MQTT 流量AWS IoT Greengrass使用連接埠 8883。
- `env_file`指定環境檔案，以設定要傳遞給 Docker 容器內AWS IoT Greengrass核心軟體安裝程式的環境變數。只有當您建立環境檔案來設定環境變數時，才需要此參數。如果您未建立環境檔案，則可以使用環境參數直接在 Compose 檔案中設定變數。

> **Note**
>
> 若要以更高的安全性執行 Docker 容器，您可以在撰寫檔案`cap_add`中使用`cap_drop`和來選擇性地啟用容器的 Linux 功能。如需詳細資訊，請參閱 Docker 文件中的執行階段權限和 Linux 功能。

3. 執行下列命令以啟動容器。

```
docker-compose -f docker-compose.yml up
```

後續步驟

AWS IoT Greengrass核心軟件現在正在 Docker 容器中運行。執行下列命令以擷取目前執行中容器的容器 ID。

```
docker ps
```

然後，您可以執行下列命令來存取容器，並探索在容器內執行的AWS IoT Greengrass核心軟體。

```
docker exec -it container-id /bin/bash
```

如需有關建立簡單元件的資訊，請參閱步驟 4：在設備上開發和測試組件中的 教學課程：AWS IoT Greengrass V2 入門

> **Note**
>
> 當您使用 `docker exec` Docker 容器內執行命令時，這些命令不會記錄在 Docker 記錄中。要在 Docker 日誌中記錄命令，請將交互式外殼附加到 Docker 容器。如需詳細資訊，請參閱 將交互式外殼附加到 Docker 容器。

AWS IoT Greengrass核心記錄檔會呼叫`greengrass.log`並位於中*/greengrass/v2*/`logs`。元件記錄檔也位於相同的目錄中。若要將 Greengrass 記錄檔複製到主機上的暫存目錄，請執行下列命令：

```
docker cp container-id:/greengrass/v2/logs /tmp/logs
```

如果您想要在容器結束或移除之後保留記錄檔，建議您只將*/greengrass/v2*/`logs`目錄繫結掛載到主機上的暫存記錄目錄，而不是掛載整個 Greengrass 目錄。如需詳細資訊，請參閱 在碼頭容器之外保存 Greengrass 日誌。

若要停止執行中的 AWS IoT Greengrass Docker 容器，請執行`docker stop`或`docker-compose -f docker-compose.yml stop`。此動作會傳送`SIGTERM`至 Greengrass 處理程序，並關閉在容器中啟動的所有相關聯處理序。Docker 容器使用`docker-init`可執行文件初始化為進程 PID 1，這有助於刪除任何剩餘的殭屍進程。如需詳細資訊，請參閱 Docker 文件中的指定初始化程序。

如需疑難排解 Docker 容器AWS IoT Greengrass中執行問題的相關資訊，請參閱Docker 容器中 AWS IoT Greengrass 的疑難排解。

Docker 容器中 AWS IoT Greengrass 的疑難排解

使用下列資訊可協助您疑難排解在 Docker 容器AWS IoT Greengrass中執行的問題，以及在 Docker 容器AWS IoT Greengrass中偵錯問題。

主題

- 疑難排解執行 Docker 容器的問題
- 在 Docker 容器中對 AWS IoT Greengrass 進行偵錯

疑難排解執行 Docker 容器的問題

請使用以下資訊針對 Docker 容器中執行 AWS IoT Greengrass 的問題進行疑難排解。

主題

錯誤：無法從非 TTY 裝置執行互動式登入

當您執行`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AWS CLI版本 2 或第 1 版。我們建議您使用AWS CLI版本 2。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：未知的選項：-no-include-email

當您執行`aws ecr get-login`命令時，可能會發生此錯誤。請確定您已安裝最新的 AWS CLI 版本 (例如，執行：`pip install awscli --upgrade --user`)。如需詳細資訊，請參閱AWS Command Line Interface使用者指南中的AWS Command Line Interface在 Microsoft 視窗上安裝。

錯誤：防火牆封鎖了 Windows 和容器之間的檔案共用。

在 Windows 電腦上執行 Docker 時，您可能會收到此錯誤或`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 作業時發生錯誤 (AccessDeniedException)：使用者:arn: aw:iam:: ***## ID: ###/#########:***<user-name>ecr: * GetAuthorizationToken

如果您沒有足夠的權限存取 Amazon ECR 儲存庫，則在執行`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 儲存庫政策範例和存取 Amazon ECR 使用者指南中的一個 Amazon ECR 儲存庫。

錯誤：您已達到提取率上限

Docker Hub 限制了匿名和免費 Docker Hub 用戶可以提出的提取請求的數量。如果您超過匿名或免費使用者提取要求的速率限制，則會收到下列其中一個錯誤：

```
ERROR: toomanyrequests: Too Many Requests.
```

```
You have reached your pull rate limit.
```

若要解決這些錯誤，您可以等待幾個小時，再嘗試其他提取要求。如果您打算持續提交大量提取請求，請訪問 Docker Hub 網站以獲取有關速率限制的信息，以及驗證和升級 Docker 帳戶的選項。

在 Docker 容器中對 AWS IoT Greengrass 進行偵錯

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。

在碼頭容器之外保存 Greengrass 日誌

停止AWS IoT Greengrass容器之後，您可以使用下列`docker cp`命令將 Greengrass 記錄從 Docker 容器複製到暫存記錄目錄。

```
docker cp container-id:/greengrass/v2/logs /tmp/logs
```

若要在容器結束或移除後仍保留記錄檔，您必須在繫結掛載目錄之後執行 AWS IoT Greengrass Docker 容器。*/greengrass/v2*/logs

若要繫結掛載*/greengrass/v2*/logs目錄，請在執行新的 AWS IoT Greengrass Docker 容器時執行下列其中一項動作。

- 包含-v */tmp/logs*:*/greengrass/v2*/logs:ro在您的`docker run`命令中。

  在執行命令之前，請修改撰寫檔案中的`volumes`區塊，以包含下列`docker-compose up`行。

  ```
  volumes:
   - /tmp/logs:/greengrass/v2/logs:ro
  ```

然後，您可以在主機*/tmp/logs*上檢查日誌，以在 Docker 容器內運行時AWS IoT Greengrass查看 Greengrass 日誌。

如需有關執行 Greengrass 碼頭容器的資訊，請參閱和 透過手動佈建AWS IoT Greengrass在 Docker 中執行 使用自動佈建AWS IoT Greengrass在 Docker 中執行

將交互式外殼附加到 Docker 容器

當您使`docker exec`用 Docker 容器內執行命令時，Docker 記錄中不會擷取這些命令。在 Docker 日誌中記錄命令可以幫助您調查 Greengrass 碼頭容器的狀態。執行以下任意一項：

- 在單獨的終端機中運行以下命令，以將終端機的標準輸入，輸出和錯誤附加到正在運行的容器。這使您可以從當前終端查看和控制 Docker 容器。

```
docker attach container-id
```

- 在單獨的終端中運行以下命令。這使您可以在互動模式下運行命令，即使未附加容器也是如此。

```
docker exec -it container-id sh -c "command > /proc/1/fd/1"
```

如需一般AWS IoT Greengrass疑難排解，請參閱疑難排解。

設定 AWS IoT Greengrass 核心軟體

AWS IoT Greengrass 核心軟體提供可用來設定軟體的選項。您可以建立部署，在每個 AWS IoT Greengrass 核心裝置上設定 Core 軟體。

主題

- 部署 Greengrass 核組件
- 將 Greengrass 核配置為系統服務
- 使用 JVM 選項控制內存分配
- 設定執行元件的使用者
- 設定元件的系統資源限制
- 連線至連接埠 443 或透過網路代理
- 使用私有 CA 簽署的裝置憑證
- 設定 MQTT 逾時和快取設定

部署 Greengrass 核組件

AWS IoT Greengrass 提供 AWS IoT Greengrass 核心軟體做為可部署到 Greengrass 核心裝置的元件。您可以建立部署，將相同的設定套用至多個 Greengrass 核心裝置。如需詳細資訊，請參閱 Greengrass 核 及 更新AWS IoT Greengrass核心軟件（OTA）。

將 Greengrass 核配置為系統服務

您必須在設備的初始化系統中將 AWS IoT Greengrass Core 軟件配置為系統服務，才能執行以下操作：

- 設備啟動時啟動 AWS IoT Greengrass Core 軟件。如果您管理大量設備，這是一個很好的做法。
- 安裝並執行外掛程式元件。幾個 AWS提供的元件是外掛程式元件，可讓它們直接與 Greengrass 核介面。如需元件類型的詳細資訊，請參閱元件類型。
- 將 over-the-air （OTA）更新應用於核心設備的 AWS IoT Greengrass 核心軟件。如需詳細資訊，請參閱 更新AWS IoT Greengrass核心軟件（OTA）。
- 當部署將元件更新為新版本或更新特定組態參數時，啟用元件以重新啟動 Core 軟體或核心裝置。AWS IoT Greengrass 如需詳細資訊，請參閱啟動程序生命週期步驟。

> **Important**
>
> 在 Windows 核心裝置上，您必須將 AWS IoT Greengrass 核心軟體設定為系統服務。

主題

將核子核配置為系統服務（Linux）

Linux 設備支持不同的初始化系統，例如初始化，系統和 SystemV。您可以在安裝 AWS IoT Greengrass Core 軟體時使用`--setup-system-service true`引數來啟動核心作為系統服務，並將其設定為在裝置開機時啟動。安裝程式會使用 systemd 將 AWS IoT Greengrass 核心軟體設定為系統服務。

您也可以手動配置核心作為系統服務運行。以下範例為 systemd 的服務檔案：

```
[Unit]
Description=Greengrass Core

[Service]
Type=simple
PIDFile=/greengrass/v2/alts/loader.pid
```

```
RemainAfterExit=no
Restart=on-failure
RestartSec=10
ExecStart=/bin/sh /greengrass/v2/alts/current/distro/bin/loader

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

設定系統服務之後，您可以執行下列命令來設定在開機時啟動裝置，以及啟動或停止 AWS IoT Greengrass Core 軟體。

- 若要檢查服務的狀態 (系統)

  ```
  sudo systemctl status greengrass.service
  ```

- 為了使原子核在設備啟動時啟動。

  ```
  sudo systemctl enable greengrass.service
  ```

- 在設備啟動時阻止原子核啟動。

  ```
  sudo systemctl disable greengrass.service
  ```

- 若要啟動 AWS IoT Greengrass 核心軟體。

  ```
  sudo systemctl start greengrass.service
  ```

- 要停止 AWS IoT Greengrass 核心軟件。

  ```
  sudo systemctl stop greengrass.service
  ```

將核心設定為系統服務 (Windows)

當您安裝核 AWS IoT Greengrass 心軟體時，您可以使用`--setup-system-service true`引數來啟動核心作為 Windows 服務，並將其設定為在裝置開機時啟動。

設定服務之後，您可以執行下列命令來設定在開機時啟動裝置，以及啟動或停止 AWS IoT Greengrass Core 軟體。您必須以系統管理員身分執行命令提示字元，才能執行這些命令。 PowerShell

Windows Command Prompt (CMD)

- 若要檢查服務的狀態

```
sc query "greengrass"
```

- 為了使原子核在設備啟動時啟動。

```
sc config "greengrass" start=auto
```

- 在設備啟動時阻止原子核啟動。

```
sc config "greengrass" start=disabled
```

- 若要啟動 AWS IoT Greengrass 核心軟體。

```
sc start "greengrass"
```

- 要停止 AWS IoT Greengrass 核心軟件。

```
sc stop "greengrass"
```

> **Note**
>
> 在 Windows 設備上， AWS IoT Greengrass 核心軟件在關閉 Greengrass 組件進程時忽略此關閉信號。如果 AWS IoT Greengrass 核心軟體在執行此命令時忽略關機訊號，請等待幾秒鐘，然後再試一次。

PowerShell

- 若要檢查服務的狀態

```
Get-Service -Name "greengrass"
```

- 為了使原子核在設備啟動時啟動。

```
Set-Service -Name "greengrass" -Status stopped -StartupType automatic
```

- 在設備啟動時阻止原子核啟動。

```
Set-Service -Name "greengrass" -Status stopped -StartupType disabled
```

- 若要啟動 AWS IoT Greengrass 核心軟體。

```
Start-Service -Name "greengrass"
```

- 要停止 AWS IoT Greengrass 核心軟件。

```
Stop-Service -Name "greengrass"
```

> **Note**
>
> 在 Windows 設備上， AWS IoT Greengrass 核心軟件在關閉 Greengrass 組件進程時忽略此關閉信號。如果 AWS IoT Greengrass 核心軟體在執行此命令時忽略關機訊號，請等待幾秒鐘，然後再試一次。

使用 JVM 選項控制內存分配

如果您在內存有限的設備 AWS IoT Greengrass 上運行，則可以使用 Java 虛擬機器（JVM）選項來控制最大堆積大小，垃圾收集模式和編譯器選項，這些選項控制 AWS IoT Greengrass Core 軟件使用的內存量。JVM 中的堆積大小會決定應用程式在記憶體回收發生之前或應用程式記憶體不足之前可以使用多少記憶體。堆積大小的上限，指定了在繁重的作業進行期間擴充堆疊時，JVM 可以分配的最大記憶體容量。

若要控制記憶體配置，請建立新部署或修訂包含核心元件的現有部署，並在核心元件`jvmOptions`組態的組態參數中指定 JVM 選項。

根據您的需求，您可以使用較少的記憶體配置或最少的記憶體配置來執行 AWS IoT Greengrass Core 軟體。

減少記憶體分配

若要在減少記憶體配置的情況下執行 AWS IoT Greengrass Core 軟體，建議您使用下列範例組態合併更新來設定核心組態中的 JVM 選項：

```
{
  "jvmOptions": "-Xmx64m -XX:+UseSerialGC -XX:TieredStopAtLevel=1"
```

```
}
```

最小記憶體分配

若要以最少的記憶體配置執行 AWS IoT Greengrass Core 軟體，建議您使用下列範例組態合併更新來設定核心組態中的 JVM 選項：

```
{
  "jvmOptions": "-Xmx32m -XX:+UseSerialGC -Xint"
}
```

這些範例組態合併更新使用下列 JVM 選項：

`-Xmx`*NN*`m`

設定 JVM 堆積大小上限。

若要減少記憶體配置，請使用`-Xmx64m`作為起始值，將堆積大小限制為 64 MB。對於最小記憶體配置，請使用`-Xmx32m`作為起始值，將堆積大小限制為 32 MB。

您可以根據實際需求增加或減少此`-Xmx`值；不過，我們強烈建議您不要將堆集大小上限設定為 16 MB 以下。如果最大堆積大小對您的環境來說太低，則 AWS IoT Greengrass 核心軟體可能會因為記憶體不足而遇到非預期的錯誤。

`-XX:+UseSerialGC`

指定使用 JVM 堆積空間的序列記憶體回收。串行垃圾回收器速度較慢，但使用的記憶體比其他 JVM 記憶體回收實作少。

`-XX:TieredStopAtLevel=1`

指示 JVM 一次使用 Java just-in-time （ JIT ）編譯器。由於 JIT 編譯的程式碼會在裝置記憶體中使用空間，因此使用 JIT 編譯器會比單一編譯耗用更多記憶體。

`-Xint`

指示 JVM 不要使用 just-in-time (JIT) 編譯器。相反地，JVM 會以僅解譯模式執行。此模式比執行 JIT 編譯的程式碼慢；但是，編譯後的程式碼不會使用記憶體中的任何空間。

如需有關建立組態合併更新的資訊，請參閱更新零組件組態。

設定執行元件的使用者

AWS IoT Greengrass 核心軟體可以以不同於執行軟體的系統使用者和群組的身分執行元件程序。這會增加安全性，因為您可以以 root 或系統管理員使用者身分執行 AWS IoT Greengrass Core 軟體，而不會將這些權限授與核心裝置上執行的元件。

下表指出 AWS IoT Greengrass 核心軟體可以以您指定的使用者身分執行的元件類型。如需詳細資訊，請參閱 元件類型。

元件類型	設定元件使用者
原子核	 否
外掛程式	 否
一般	 是
Lambda（非容器化）	 是
容器化	 是

您必須先建立元件使用者，才能在部署規劃中指定元件使用者。在 Windows 裝置上，您也必須將使用者的使用者名稱和密碼儲存在 LocalSystem 帳戶的認證管理員執行個體中。如需詳細資訊，請參閱 在 Windows 裝置上設定元件使用者。

在 Linux 裝置上設定元件使用者時，您也可以選擇性地指定群組。您可以指定以冒號 (:) 分隔的使用者和群組，格式如下：*user*:*group*。如果您未指定群組， AWS IoT Greengrass Core 軟體會預設為使用者的主要群組。您可以使用名稱或 ID 來識別使用者和群組。

在 Linux 裝置上，您也可以以不存在的系統使用者身分執行元件 (也稱為未知使用者)，以提高安全性。Linux 進程可以發出信號由同一用戶運行的任何其他進程。不明的使用者不會執行其他處理程序，因此您可以以未知使用者身分執行元件，以防止元件在核心裝置上發出信號傳送其他元件。若要以未知使用者身分執行元件，請指定核心裝置上不存在的使用者 ID。您也可以指定不存在的群組 ID，以未知群組的形式執行。

您可以為每個元件和每個核心裝置設定使用者。

- 設定元件

  您可以將每個元件設定為與該元件特有的使用者一起執行。建立部署時，您可以為該元件的`runWith`規劃中的每個元件指定使用者。如果您設定元件， AWS IoT Greengrass Core 軟體會以指定的使用者身分執行元件。否則，它預設會以您為核心裝置設定的預設使用者身分執行元件。如需有關在部署組態中指定元件使用者的詳細資訊，請參閱中的`runWith`組態參數建立部署。

- 設定核心裝置的預設使用者

  您可以設定 AWS IoT Greengrass Core 軟體用來執行元件的預設使用者。當 AWS IoT Greengrass 核心軟體執行元件時，它會檢查您是否為該元件指定了使用者，並使用它來執行元件。如果元件未指定使用者，則 AWS IoT Greengrass Core 軟體會以您為核心裝置設定的預設使用者身分執行該元件。如需詳細資訊，請參閱 設定預設元件使用者。

> **Note**
>
> 在以 Windows 為基礎的裝置上，您必須至少指定一個預設使用者來執行元件。
>
> 在 Linux 裝置上，如果您未設定使用者執行元件，則需要考量下列事項：
>
> - 如果您以 root 身份運行 AWS IoT Greengrass Core 軟件，則該軟件將無法運行組件。如果您以 root 身份執行，則必須指定預設使用者來執行元件。

- 如果您以非 root 使用者身分執行 AWS IoT Greengrass Core 軟體，則軟體會以該使用者的身分執行元件。

主題

在 Windows 裝置上設定元件使用者

在以 Windows 為基礎的裝置上設定元件使用者

1. 在裝置上的 LocalSystem 帳戶中建立元件使用者。

```
net user /add component-user password
```

2. 使用 Microsoft 的 PsExec 公用程式，將元件使用者的使用者名稱和密碼儲存在 LocalSystem 帳戶的認證管理員執行個體中。

```
psexec -s cmd /c cmdkey /generic:component-user /user:component-user /pass:password
```

> **Note**
>
> 在 Windows 裝置上， LocalSystem 帳戶會執行 Greengrass 核心，而且您必須使用公用 PsExec 程式將元件使用者資訊儲存在帳戶中。 LocalSystem使用認證管理員應用程式會將此資訊儲存在目前登入使用者的 Windows 帳戶中，而非 LocalSystem帳戶中。

設定預設元件使用者

您可以使用部署來規劃核心裝置上的預設使用者。在此部署中，您會更新核心元件組態。

> **Note**
>
> 當您使用`--component-default-user`選項安裝 AWS IoT Greengrass Core 軟體時，您也可以設定預設使用者。如需詳細資訊，請參閱 安裝 AWS IoT Greengrass 核心軟體。

建立為元件指定下列規劃更新的aws.greengrass.Nucleus部署。

Linux

```
{
  "runWithDefault": {
    "posixUser": "ggc_user:ggc_group"
  }
}
```

Windows

```
{
  "runWithDefault": {
    "windowsUser": "ggc_user"
  }
}
```

> **Note**
>
> 您指定的使用者必須存在，且此使用者的使用者名稱和密碼必須儲存在 Windows 裝置上 LocalSystem 帳戶的認證管理員執行個體中。如需詳細資訊，請參閱 在 Windows 裝置上設定元件使用者。

下列範例會定義 Linux 裝置的部署，該裝置設定為預設使用者並ggc_group設定ggc_user為預設群組。組merge態更新需要序列化的 JSON 物件。

```
{
  "components": {
    "aws.greengrass.Nucleus": {
      "version": "2.12.3",
      "configurationUpdate": {
        "merge": "{\"runWithDefault\":{\"posixUser\":\"ggc_user:ggc_group\"}}"
      }
    }
  }
}
```

設定元件的系統資源限制

> **Note**
>
> 此功能適用於 v2.4.0 和更高版 Greeng rass 核組件。 AWS IoT Greengrass 目前在 Windows 核心裝置上不支援此功能。

您可以設定每個元件的處理序可在核心裝置上使用的 CPU 和 RAM 使用量上限。

下表顯示支援系統資源限制的元件類型。如需詳細資訊，請參閱 元件類型。

元件類型	設定系統資源限制
原子核	 否
外掛程式	 否
一般	 是
Lambda（非容器化）	 是

元件類型	設定系統資源限制
容器化	 否

> **Important**
>
> 當您在 Docker 容器中執行 AWS IoT Greengrass 核心軟體時，系統資源限制不受支援。

您可以為每個元件和每個核心裝置設定系統資源限制。

- 設定元件

  您可以使用特定於該元件的系統資源限制來設定每個元件。建立部署時，您可以為部署中的每個元件指定系統資源限制。如果元件支援系統資源限制， AWS IoT Greengrass 核心軟體就會將限制套用至元件的程序。如果您未指定元件的系統資源限制，則 AWS IoT Greengrass Core 軟體會使用您為核心裝置設定的任何預設值。如需詳細資訊，請參閱 建立部署。

- 設定核心裝置的預設值

  您可以設定 AWS IoT Greengrass Core 軟體套用至支援這些限制之元件的預設系統資源限制。AWS IoT Greengrass 核心軟體執行元件時，會套用您為該元件指定的系統資源限制。如果該元件未指定系統資源限制，則 AWS IoT Greengrass Core 軟體會套用您為核心裝置設定的預設系統資源限制。如果您未指定預設的系統資源限制， AWS IoT Greengrass 核心軟體預設不會套用任何系統資源限制。如需詳細資訊，請參閱 設定預設系統資源限制。

設定預設系統資源限制

您可以部署 Greengrass 核心元件，以設定核心裝置的預設系統資源限制。若要設定預設系統資源限制，請建立指定下列`aws.greengrass.Nucleus`元件組態更新的部署。

```
{
  "runWithDefault": {
    "systemResourceLimits": {
      "cpu": cpuTimeLimit,
```

```
      "memory": memoryLimitInKb
    }
  }
}
```

下列範例會定義將 CPU 時間限制設定為的部署2，相當於具有 4 個 CPU 核心的裝置使用率 50%。此範例也會將記憶體使用量設定為 100 MB。

```
{
  "components": {
    "aws.greengrass.Nucleus": {
      "version": "2.12.3",
      "configurationUpdate": {
        "merge": "{\"runWithDefault\":{\"systemResourceLimits\":\"cpus\":2,\"memory
\":102400}}}"
      }
    }
  }
}
```

連線至連接埠 443 或透過網路代理

AWS IoT Greengrass 核心裝置與 AWS IoT Core 使用具有 TLS 用戶端驗證的 MQTT 訊息通訊協定進行通訊。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。其他限制性環境可能需要所有流量在連接到互聯網之前通過代理。

> **Note**
>
> 執行 Greengrass 核心元件 v2.0.3 及更早版本的 Greengrass 核心裝置使用連接埠 8443 連線至資料平面端點。 AWS IoT Greengrass 這些裝置必須能夠在連接埠 8443 上連線到此端點。如需詳細資訊，請參閱 允許裝置流量透過 Proxy 或防火牆。

若要在這些情況下啟用通訊，請 AWS IoT Greengrass 提供下列組態選項：

- 透過連接埠 443 進行 MQTT 通訊。如果您的網路允許連線至連接埠 443，您可以設定 Greengrass 核心裝置使用連接埠 443 作為 MQTT 流量，而非預設連接埠 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。與使用憑證型用戶端驗證的預設組態不同，連接埠 443 上的 MQTT 會使用裝置服務角色進行驗證。

如需詳細資訊，請參閱 透過連接埠 443 設定 MQTT。

- 透過連接埠 443 進行 HTTPS 通訊。 AWS IoT Greengrass 核心軟體預設會透過連接埠 8443 傳送 HTTPS 流量，但您可以將其設定為使用連接埠 443。 AWS IoT Greengrass 使用應用程式層通訊協定網路 (ALPN) TLS 延伸功能來啟用此連線。與預設組態一樣，連接埠 443 上的 HTTPS 會使用憑證型用戶端驗證。

  > **Important**
  >
  > 若要使用 ALPN 並透過連接埠 443 啟用 HTTPS 通訊，您的核心裝置必須執行 Java 8 更新 252 或更新版本。Java 版本 9 及更新版本的所有更新也支援 ALPN。

  如需詳細資訊，請參閱 透過連接埠 443 設定 HTTPS。

- 透過網路代理的連線。您可以配置網絡代理服務器作為連接到 Greengrass 核心設備的中介。 AWS IoT Greengrass 支援 HTTP 和 HTTPS 代理伺服器的基本驗證。

  核心裝置必須執行 Greengrass 核心 v2.5.0 或更新版本才能使用 HTTPS 代理伺服器。

  AWS IoT Greengrass 核心軟體會透過、、和NO_PROXY環境變數 ALL_PROXY HTTP_PROXYHTTPS_PROXY，將 Proxy 組態傳遞給元件。元件必須使用這些設定，才能透過 Proxy 連線。組件使用通用庫（例如 boto3，cURL 和 python `requests` 包），這些庫通常默認使用這些環境變量進行連接。如果組件也指定了這些環境變量，則 AWS IoT Greengrass 不會覆蓋它們。

  如需詳細資訊，請參閱 設定網路代理。

透過連接埠 443 設定 MQTT

您可以在現有核心裝置上透過連接埠 443 設定 MQTT，或在新的核心裝置上安裝 AWS IoT Greengrass Core 軟體時。

主題

- 在現有核心裝置上透過連接埠 443 設定 MQTT
- 在安裝期間透過連接埠 443 設定 MQTT

在現有核心裝置上透過連接埠 443 設定 MQTT

您可以使用部署，透過連接埠 443 在單一核心裝置或核心裝置群組上設定 MQTT。在此部署中，您會更新核心元件組態。當您更新核心的mqtt組態時，核心會重新啟動。

若要透過連接埠 443 設定 MQTT，請建立為元件指定下列組態更新的部署。aws.greengrass.Nucleus

```
{
  "mqtt": {
    "port": 443
  }
}
```

下列範例會定義透過連接埠 443 設定 MQTT 的部署。組merge態更新需要序列化的 JSON 物件。

```
{
  "components": {
    "aws.greengrass.Nucleus": {
      "version": "2.12.3",
      "configurationUpdate": {
        "merge": "{\"mqtt\":{\"port\":443}}"
      }
    }
  }
}
```

在安裝期間透過連接埠 443 設定 MQTT

當您在核心裝置上安裝 AWS IoT Greengrass Core 軟體時，您可以透過連接埠 443 設定 MQTT。使用--init-config安裝程式引數透過連接埠 443 設定 MQTT。當您使用手動佈建、叢集佈建或自訂佈建進行安裝時，您可以指定此引數。

透過連接埠 443 設定 HTTPS

此功能需要 Greengrass 核 v2.0.4 或更新版本。

您可以在現有的核心裝置上透過連接埠 443 設定 HTTPS，或在新的 AWS IoT Greengrass 核心裝置上安裝 Core 軟體時。

主題

- 在現有核心裝置上透過連接埠 443 設定 HTTPS
- 在安裝期間透過連接埠 443 設定 HTTPS

在現有核心裝置上透過連接埠 443 設定 HTTPS

您可以使用部署，透過連接埠 443 在單一核心裝置或核心裝置群組上設定 HTTPS。在此部署中，您會更新核心元件組態。

若要透過連接埠 443 設定 HTTPS，請建立為`aws.greengrass.Nucleus`元件指定下列組態更新的部署。

```
{
  "greengrassDataPlanePort": 443
}
```

下列範例會定義透過連接埠 443 設定 HTTPS 的部署。組`merge`態更新需要序列化的 JSON 物件。

```
{
  "components": {
    "aws.greengrass.Nucleus": {
      "version": "2.12.3",
      "configurationUpdate": {
        "merge": "{\"greengrassDataPlanePort\":443}"
      }
    }
  }
}
```

在安裝期間透過連接埠 443 設定 HTTPS

當您在核心裝置上安裝 AWS IoT Greengrass 核心軟體時，您可以透過連接埠 443 設定 HTTPS。使用`--init-config`安裝程式引數，透過連接埠 443 設定 HTTPS。當您使用手動佈建、叢集佈建或自訂佈建進行安裝時，您可以指定此引數。

設定網路代理

請遵循本節中的程序，將 Greengrass 核心裝置設定為透過 HTTP 或 HTTPS 網路代理伺服器連線到網際網路。如需核心裝置使用的端點和通訊埠的詳細資訊，請參閱允許裝置流量透過 Proxy 或防火牆。

Important

如果您的核心裝置執行的 Greengrass 核心版本早於 v2.4.0，則您的裝置角色必須允許下列權限才能使用網路 Proxy：

- `iot:Connect`
- `iot:Publish`
- `iot:Receive`
- `iot:Subscribe`

這是必要的，因為裝置會使用 Token 交換服務的 AWS 認證來驗證與的 MQTT 連線。 AWS IoT裝置使用 MQTT 接收和安裝來自的部署 AWS 雲端，因此除非您對裝置的角色定義這些權限，否則您的裝置將無法運作。裝置通常會使用 X.509 憑證來驗證 MQTT 連線，但裝置在使用 Proxy 時無法進行驗證。
如需如何設定裝置角色的詳細資訊，請參閱授權核心設備與AWS服務。

主題

在現有核心裝置上設定網路 Proxy

您可以使用部署在單一核心裝置或核心裝置群組上設定網路 Proxy。在此部署中，您會更新核心元件組態。當您更新核心的`networkProxy`組態時，核心會重新啟動。

若要設定網路 Proxy，請為合併下列組態更新的`aws.greengrass.Nucleus`元件建立部署。此組態更新包含 networkProxy 物件。

```
{
  "networkProxy": {
    "noProxyAddresses": "http://192.168.0.1,www.example.com",
    "proxy": {
      "url": "https://my-proxy-server:1100"
    }
```

```
  }
}
```

下列範例會定義設定網路 Proxy 的部署。組merge態更新需要序列化的 JSON 物件。

```
{
  "components": {
    "aws.greengrass.Nucleus": {
      "version": "2.12.3",
      "configurationUpdate": {
        "merge": "{\"networkProxy\":{\"noProxyAddresses\":
\"http://192.168.0.1,www.example.com\",\"proxy\":{\"url\":\"https://my-proxy-
server:1100\",\"username\":\"Mary_Major\",\"password\":\"pass@word1357\"}}}"
      }
    }
  }
}
```

在安裝期間設定網路 Proxy

當您在核心裝置上安裝 AWS IoT Greengrass Core 軟體時，您可以設定網路 Proxy。使用--init-config安裝程式引數來設定網路 Proxy。當您使用手動佈建、叢集佈建或自訂佈建進行安裝時，您可以指定此引數。

讓核心裝置信任 HTTPS 代理

當您將核心裝置設定為使用 HTTPS Proxy 時，必須將 Proxy 伺服器憑證鏈新增至核心裝置，才能讓它信任 HTTPS 代理。否則，當核心裝置嘗試透過 Proxy 路由傳送流量時，可能會遇到錯誤。將代理伺服器 CA 憑證新增至核心裝置的 Amazon 根 CA 憑證檔案。

啟用核心裝置信任 HTTPS 代理

1. 在核心裝置上尋找 Amazon 根 CA 憑證檔案。

   - 如果您使用自動佈建安裝 AWS IoT Greengrass 核心軟體，則 Amazon 根 CA 憑證檔案會存在於*/greengrass/v2*/rootCA.pem。
   - 如果您使用手動佈建或叢集佈建安裝 AWS IoT Greengrass Core 軟體，則 Amazon 根 CA 憑證檔案可能存在於*/greengrass/v2*/AmazonRootCA1.pem。

   如果這些位置不存在 Amazon 根 CA 憑證，請檢查中的system.rootCaPath屬性*/greengrass/v2*/config/effectiveConfig.yaml以尋找其位置。

2. 將代理伺服器 CA 憑證檔案的內容新增至 Amazon 根 CA 憑證檔案。

   下列範例顯示新增至 Amazon 根 CA 憑證檔案的代理伺服器 CA 憑證。

```
-----BEGIN CERTIFICATE-----
MIIEFTCCAv2gAwIQWgIVAMHSAzWG/5YVRYtRQOxXUTEpHuEmApzGCSqGSIb3DQEK
\nCwUAhuL9MQswCQwJVUzEPMAVUzEYMBYGA1UECgwP1hem9uLmNvbSBJbmMuMRww
... content of proxy CA certificate ...
+vHIRlt0e5JAm5\noTIZGoFbK82A0/nO7f/t5PSIDAim9V3Gc3pSXxCCAQoFYnui
GaPUlGk1gCE84a0X\n7Rp/lND/PuMZ/s8YjlkY2NmYmNjMCAXDTE5MTEyN2cM216
gJMIADggEPADf2/m45hzEXAMPLE=
-----END CERTIFICATE-----

-----BEGIN CERTIFICATE-----
MIIDQTCCAimgF6AwIBAgITBmyfz/5mjAo54vB4ikPmljZKyjANJmApzyMZFo6qBg
ADA5MQswCQYDVQQGEwJVUzEPMA0tMVT8QtPHRh8jrdkGA1UEChMGDV3QQDExBBKW
... content of root CA certificate ...
o/ufQJQWUCyziar1hem9uMRkwFwYVPSHCb2XV4cdFyQzR1KldZwgJcIQ6XUDgHaa
5MsI+yMRQ+hDaXJiobldXgjUka642M4UwtBV8oK2xJNDd2ZhwLnoQdeXeGADKkpy
rqXRfKoQnoZsG4q5WTP46EXAMPLE
-----END CERTIFICATE-----
```

networkProxy 物件

使用 networkProxy 物件，指定網路代理的相關資訊。此物件包含下列資訊：

noProxyAddresses

(選擇性) 以逗號分隔的 IP 位址或主機名稱清單 (不包括代理伺服器)。

proxy

要連線的代理伺服器。此物件包含下列資訊：

url

代理伺服器的格式 URL scheme://userinfo@host:port。

- scheme— 該計劃，必須是http或https。

> **Important**
>
> 核心裝置必須執行 Greengrass 核心 v2.5.0 或更新版本才能使用 HTTPS 代理伺服器。

如果您設定 HTTPS 代理，則必須將代理伺服器 CA 憑證新增至核心裝置的 Amazon 根 CA 憑證。如需詳細資訊，請參閱 讓核心裝置信任 HTTPS 代理。

- `userinfo`— (選用) 使用者名稱和密碼資訊。如果您在中指定此資訊`url`，Greengrass 核心裝置會忽略和欄位`username`。`password`
- `host`— 代理伺服器的主機名稱或 IP 位址。
- `port`— (選擇性) 連接埠號碼。如果您未指定連接埠，則 Greengrass 核心裝置會使用下列預設值：
  - `http`— 80
  - `https`— 443

`username`

(選擇性) 驗證 Proxy 伺服器的使用者名稱。

`password`

(選擇性) 驗證 Proxy 伺服器的密碼。

使用私有 CA 簽署的裝置憑證

如果您使用的是自訂私有憑證授權單位 (CA)，則必須將 Greengrass 核心設定為。**`greengrassDataPlaneEndpoint`** **`iotdata`**您可以在部署或安裝期間使用 **`--init-config`** installer 引數來設定此選項。

您可以自訂裝置所連接的 Greengrass 資料平面端點。您可以將此組態選項設定為，**`iotdata`**將 Greengrass 資料平面端點設定為與 IoT 資料端點相同的端點，您可以使用. **`iotDataEndpoint`**

設定 MQTT 逾時和快取設定

在 AWS IoT Greengrass 環境中，元件可以使用 MQTT 進行通訊。 AWS IoT Core AWS IoT Greengrass 核心軟體會管理元件的 MQTT 訊息。當核心裝置失去與的連線時 AWS 雲端，軟體會快取 MQTT 訊息，以便稍後在連線恢復時重試。您可以設定訊息逾時和快取大小等設定。如需詳細資訊，請參閱 Greengrass 核心元件的`mqtt`和`mqtt.spooler`組態參數。

AWS IoT Core 對其 MQTT 訊息代理程式強加服務配額。這些配額可能會套用至您在核心裝置和之間傳送的郵件 AWS IoT Core。如需詳細資訊，請參閱AWS IoT Core AWS 一般參考.

更新AWS IoT Greengrass核心軟件（OTA）

AWS IoT Greengrass核心軟件包括 Greengrass 核組件和其他可選組件，您可以將這些組件部署到設備以執行軟件 over-the-air （OTA）更新。此功能內置於AWS IoT Greengrass核心軟件中。

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。

主題

要求

下列需求適用於部署AWS IoT Greengrass核心軟體的 OTA 更新：

- Greengrass 核心裝置必須與連線才能AWS 雲端接收部署。
- Greengrass 核心裝置必須正確設定並佈建憑證和金鑰，以便使用和進行驗證。AWS IoT Core AWS IoT Greengrass
- AWS IoT Greengrass核心軟體必須設定並以系統服務的身分執行。如果您從 JAR 文件運行核，`Greengrass.jar`則 OTA 更新不起作用。如需詳細資訊，請參閱 將 Greengrass 核配置為系統服務。

核心裝置的注意事項

在執行 OTA 更新之前，請注意對您更新的核心設備及其連接的客戶端設備的影響：

- Greengrass 核關閉。

- 核心裝置上執行的所有元件也會關閉。如果這些元件寫入本機資源，除非正確關閉，否則可能會讓這些資源處於不正確的狀態。元件可以使用處理序間通訊，告知核心元件延遲更新，直到清除其使用的資源為止。
- 核心元件關閉時，核心裝置會失去與AWS 雲端和本機裝置的連線。核心裝置在關機時不會從用戶端裝置路由傳送訊息。
- 以元件執行的長壽命 Lambda 函數會遺失其動態狀態資訊，並捨棄所有擱置中的工作。

Greengrass 核更新行為

當您部署元件時，AWS IoT Greengrass會安裝該元件所有相依性的最新受支援版本。因此，如果您將新裝置新增至物件群組，或更新以這些裝置為目標的部署，則AWS提供之公用元件的新修補程式版本可能會自動部署到核心裝置。某些自動更新 (例如核心更新) 可能會導致裝置意外重新啟動。

當 Greengrass 核心元件的版本變更時，核AWS IoT Greengrass心軟體 (包括核心和裝置上的所有其他元件) 會重新啟動以套用變更。由於核心元件更新時會對核心裝置造成影響，因此您可能想要控制新的核心修補程式版本何時部署到您的裝置。若要這麼做，您必須在部署中直接包含 Greengrass 核心元件。直接包含元件表示您在部署組態中包含該元件的特定版本，而且不需要依賴元件相依性將該元件部署到您的裝置。如需有關在元件配方中定義相依性的詳細資訊，請參閱食譜格式。

檢閱下表，瞭解 Greengrass 核心元件的更新行為，根據您的動作和部署設定。

動作	部署組態	核更新行為
在不修訂部署的情況下，將新設備新增至現有部署目標的物群組。	該部署不直接包括 Greengrass 核。 部署直接包含至少一個AWS提供的元件，或包含依賴於AWS提供的元件或 Greengrass 核心的自訂元件。	在新裝置上，安裝符合所有元件相依性需求的最新修補程式版本的核心。 在現有設備上，不會更新核的已安裝版本。
在不修訂部署的情況下，將新設備新增至現有部署目標的物群組。	部署直接包括 Greengrass 核的特定版本。	在新裝置上，安裝指定的核心版本。 在現有設備上，不會更新核的已安裝版本。

動作	部署組態	核更新行為
建立新部署或修訂既有部署。	該部署不直接包括 Greengrass 核。 部署直接包含至少一個AWS提供的元件，或包含依賴於AWS提供的元件或 Greengrass 核心的自訂元件。	在所有目標裝置上，安裝最新的核心修補程式版本，以符合所有元件相依性需求，包括在您新增至目標物件群組的任何新裝置上。
建立新部署或修訂既有部署。	部署直接包括 Greengrass 核的特定版本。	在所有目標裝置上，安裝指定的核心版本，包括您新增至目標物件群組的任何新裝置。

執行 OTA 更新

若要執行 OTA 更新，請建立包含核心元件和要安裝的版本的部署。

解除安裝AWS IoT Greengrass核心軟體

您可以卸載 AWS IoT Greengrass Core 軟件，以將其從不想用作 Greengrass 核心設備的設備中刪除。您也可以使用這些步驟來清除失敗的安裝。

若要解除安裝AWS IoT Greengrass核心軟體

1. 如果您以系統服務的身分執行軟體，則必須停止、停用及移除該服務。請根據您的作業系統執行下列指令。

   Linux

   1. 停止 服務。

      ```
      sudo systemctl stop greengrass.service
      ```

   2. 停用服務。

      ```
      sudo systemctl disable greengrass.service
      ```

3. 移除服務。

```
sudo rm /etc/systemd/system/greengrass.service
```

4. 確認服務已刪除。

```
sudo systemctl daemon-reload && sudo systemctl reset-failed
```

Windows (Command Prompt)

> **Note**
> 您必須以管理員身份運行命令提示符才能運行這些命令。

1. 停止 服務。

```
sc stop "greengrass"
```

2. 停用服務。

```
sc config "greengrass" start=disabled
```

3. 移除服務。

```
sc delete "greengrass"
```

4. 重新啟動裝置。

Windows (PowerShell)

> **Note**
> 您必須以系統管理員身分執 PowerShell 行，才能執行這些命令。

1. 停止 服務。

```
Stop-Service -Name "greengrass"
```

2. 停用服務。

```
Set-Service -Name "greengrass" -Status stopped -StartupType disabled
```

3. 移除服務。

   - 對於 PowerShell 6.0 及更高版本：

```
Remove-Service -Name "greengrass" -Confirm:$false -Verbose
```

   - 對於 6.0 以前的 PowerShell 版本：

```
Get-Item HKLM:\SYSTEM\CurrentControlSet\Services\greengrass | Remove-Item
 -Force -Verbose
```

4. 重新啟動裝置。

2. 從設備中刪除根文件夾。以根資料夾的路徑取代*/greengrass/v2*或 *C:\greengrass\v2*。

   Linux

```
sudo rm -rf /greengrass/v2
```

   Windows (Command Prompt)

```
rmdir /s /q C:\greengrass\v2
```

   Windows (PowerShell)

```
cmd.exe /c "rmdir /s /q C:\greengrass\v2"
```

3. 從AWS IoT Greengrass服務中刪除核心裝置。此步驟會從中移除核心裝置的狀態資訊AWS 雲端。如果您計劃將 AWS IoT Greengrass Core 軟體重新安裝到具有相同名稱的核心裝置，請務必完成此步驟。

   - 若要從AWS IoT Greengrass主控台刪除核心裝置，請執行下列動作：

     a. 導覽至 AWS IoT Greengrass主控台。

     b. 選擇核心裝置。

c. 選擇要刪除的核心裝置。

d. 選擇刪除。

e. 在確認模式中，選擇「刪除」。

- 若要使用刪除核心裝置AWS Command Line Interface，請使用此DeleteCoreDevice作業。執行下列命令，並*MyGreengrassCore*以核心裝置的名稱取代。

```
aws greengrassv2 delete-core-device --core-device-thing-name MyGreengrassCore
```

AWS IoT Greengrass V2 教學課程

您可以完成以下自學課程以瞭解AWS IoT Greengrass V2及其功能。

主題

教學課程：開發延遲元件更新的 Greengrass 元件

您可以完成此自學課程以開發延遲 over-the-air 部署更新的元件。當您將更新部署到裝置時，您可能會想要根據條件延遲更新，例如下所示：

- 該設備具有低電池電量。
- 裝置正在執行無法中斷的程序或工作。
- 該設備的互聯網連接有限或昂貴。

> **Note**
>
> 組件是在AWS IoT Greengrass核心設備上運行的軟件模塊。元件可讓您建立和管理複雜的應用程式，做為獨立建置區塊，您可以從一個 Greengrass 核心裝置重複使用到另一個核心裝置。

在此教學課程中，您將執行下列操作：

1. 在您的開發計算機上安裝綠色開發套件 CLI（GDK CLI）。GDK CLI 提供的功能可協助您開發自訂的 Greengrass 元件。
2. 開發 Hello World 元件，在核心裝置的電池電量低於閾值時延遲元件更新。此元件會使用 SubscribeToComponentUpdatesIPC 作業訂閱更新通知。當它收到通知時，它會檢查電池電量是否低於可定制的閾值。如果電池電量低於閾值，則會使用 DeferComponentUpdateIPC 操作將更新延遲 30 秒。您可以使用 GDK CLI 在開發電腦上開發此元件。

> **Note**
>
> 此元件會從您在核心裝置上建立的檔案讀取電池電量，以模擬真實電池，因此您可以在沒有電池的情況下在核心裝置上完成本教學課程。

3. 將該元件發佈至AWS IoT Greengrass服務。
4. 將該元件從部署AWS 雲端到 Greengrass 核心裝置以對其進行測試。然後，您可以修改核心裝置上的虛擬電池電量，並建立其他部署，以查看核心裝置在電池電量不足時如何延遲更新。

您可以預期在本教程上花費 20-30 分鐘。

必要條件

為了完成本教學，您需要以下項目：

- AWS 帳戶。如果您沒有帳戶，請參閱 設置一個 AWS 帳戶。
- 具有管理員權限的 AWS Identity and Access Management (IAM) 使用者。
- 具有互聯網連接的 Greengrass 核心設備。如需如何設定核心裝置的詳細資訊，請參閱設定AWS IoT Greengrass核心裝置。
  - Python 3.6 或更高版本為核心設備上的所有用戶安裝並添加到PATH環境變量中。在視窗上，您還必須為所有使用者安裝適用於視窗的 Python 啟動器。

> **Important**
>
> 在視窗中，默認情況下不會為所有用戶安裝 Python。當您安裝 Python 時，您必須自定義安裝以配置它，以便AWS IoT Greengrass核心軟件運行 Python 腳本。例如，如果您使用圖形化 Python 安裝程式，請執行下列動作：
>
> 1. 選取 [為所有使用者安裝啟動器 (建議)]。
> 2. 選擇 Customize installation。
> 3. 選擇 Next。
> 4. 選取 Install for all users。
> 5. 選取 Add Python to environment variables。
> 6. 選擇 Install (安裝)。

如需詳細資訊，請參閱 Python 3 文件中的在視窗上使用 Python。

- 具有網際網路連線的視窗、macOS 或類似 UNIX 的開發電腦。
  - 在您的開發計算機上安裝了 Python 3.6 或更高版本。
  - Git 安裝在您的開發計算機上。
  - AWS Command Line Interface（AWS CLI）在您的開發計算機上安裝和配置憑據。如需詳細資訊，請參閱《AWS Command Line Interface使用指南》AWS CLI中的〈安裝、更新AWS CLI和解除安裝〉和〈設定〉。

> **Note**
>
> 如果您使用樹莓派或其他 32 位 ARM 設備，請安裝 AWS CLI V1。 AWS CLIV2 不適用於 32 位元 ARM 裝置。如需詳細資訊，請參閱安裝、更新和解除安裝AWS CLI版本 1。

第 1 步：安裝格 Greengrass 開發工具包 CLI

Greengrass 開發套件 CLI (GDK CLI) 提供的功能可協助您開發自訂的 Greengrass 元件。您可以使用 GDK CLI 來建立、建置和發佈自訂元件。

如果您尚未在開發電腦上安裝 GDK CLI，請完成以下步驟進行安裝。

若要安裝最新版本的 GDK CLI

1. 在您的開發電腦上，執行下列命令，從其GitHub存放庫安裝最新版本的 GDK CLI。

```
python3 -m pip install -U git+https://github.com/aws-greengrass/aws-greengrass-gdk-cli.git@v1.6.2
```

2. 執行下列命令以驗證 GDK CLI 是否已成功安裝。

```
gdk --help
```

如果找不到該gdk命令，請將其文件夾添加到 PATH。

- 在 Linux 設備上，添加/home/*MyUser*/.local/bin到 PATH，並替換為您*MyUser*的用戶的名稱。

- 在 Windows 設備上，添加*PythonPath*\\Scripts到路徑，並*PythonPath*替換為設備上 Python 文件夾的路徑。

步驟 2：開發延遲更新的元件

在本節中，您會在 Python 中開發 Hello World 元件，當核心裝置的電池電量低於部署元件時所設定的臨界值時，會延遲元件更新。在這個元件中，您可以使用 Python AWS IoT Device SDK v2 中的處理序間通訊 (IPC) 介面。當核心裝置收到部署時，您可以使用 SubscribeToComponentUpdatesIPC 作業接收通知。然後，您可以使用 DeferComponentUpdateIPC 操作根據設備的電池電量延遲或確認更新。

若要開發延遲更新的 Hello World 元件

1. 在開發電腦上，建立元件原始程式碼的資料夾。

   ```
   mkdir com.example.BatteryAwareHelloWorld
   cd com.example.BatteryAwareHelloWorld
   ```

2. 使用文字編輯器建立名為的檔案gdk-config.json。GDK CLI 會從名為的 GDK CLI 組態檔讀取gdk-config.json，以建置和發佈元件。此組態檔案存在於元件資料夾的根目錄中。

   例如，在 Linux 系統上，您可以執行下列命令來使用 GNU nano 建立檔案。

   ```
   nano gdk-config.json
   ```

   將下列 JSON 複製到檔案中。

   - 用您的名字替換 *Amazon*。
   - 將 *us-west-2* 替換為核心設備的運行AWS 區域位置。GDK CLI 會發佈此AWS 區域中的元件。
   - 以要使*greengrass-component-artifacts*用的 S3 儲存貯體前綴取代。當您使用 GDK CLI 發佈元件時，GDK CLI 會將元件的成品上傳到名稱由此值組成的 S3 儲存貯體，以及您的 AWS 區域 AWS 帳戶 ID 使用下列格式：。*bucketPrefix*-*region*-*accountId*

     例如，如果您指定**greengrass-component-artifacts**和且**us-west-2**您的AWS 帳戶識別碼為**123456789012**，GDK CLI 會使用名greengrass-component-artifacts-us-west-2-123456789012為的 S3 儲存貯體。

```
{
  "component": {
    "com.example.BatteryAwareHelloWorld": {
      "author": "Amazon",
      "version": "NEXT_PATCH",
      "build": {
        "build_system" : "zip"
      },
      "publish": {
        "region": "us-west-2",
        "bucket": "greengrass-component-artifacts"
      }
    }
  },
  "gdk_version": "1.0.0"
}
```

組態檔案會指定下列項目：

- GDK CLI 將 Greengrass 元件發佈到雲端服務時所使用的版本。AWS IoT Greengrass `NEXT_PATCH`指定選擇AWS IoT Greengrass雲端服務中可用的最新版本之後的下一個修補程式版本。如果元件還沒有AWS IoT Greengrass雲端服務中的版本，GDK CLI 會使用`1.0.0`.
- 元件的建置系統。當您使用`zip`建置系統時，GDK CLI 會將元件的來源封裝成 ZIP 檔案，成為元件的單一成品。
- AWS 區域其中 GDK CLI 發布的 Greengrass 件。
- 在其中 GDK CLI 上傳元件成品的 S3 儲存貯體的前置詞。

3. 使用文字編輯器在名為的檔案中建立元件原始碼`main.py`。

   例如，在 Linux 系統上，您可以執行下列命令來使用 GNU nano 建立檔案。

   ```
   nano main.py
   ```

   將下面的 Python 代碼複製到文件中。

   ```
   import json
   import os
   import sys
   import time
   ```

```
import traceback

from pathlib import Path

from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2

HELLO_WORLD_PRINT_INTERVAL = 15  # Seconds
DEFER_COMPONENT_UPDATE_INTERVAL = 30 * 1000  # Milliseconds


class BatteryAwareHelloWorldPrinter():
    def __init__(self, ipc_client: GreengrassCoreIPCClientV2, battery_file_path:
 Path, battery_threshold: float):
        self.battery_file_path = battery_file_path
        self.battery_threshold = battery_threshold
        self.ipc_client = ipc_client
        self.subscription_operation = None

    def on_component_update_event(self, event):
        try:
            if event.pre_update_event is not None:
                if self.is_battery_below_threshold():
                    self.defer_update(event.pre_update_event.deployment_id)
                    print('Deferred update for deployment %s' %
                          event.pre_update_event.deployment_id)
                else:
                    self.acknowledge_update(
                        event.pre_update_event.deployment_id)
                    print('Acknowledged update for deployment %s' %
                          event.pre_update_event.deployment_id)
            elif event.post_update_event is not None:
                print('Applied update for deployment')
        except:
            traceback.print_exc()

    def subscribe_to_component_updates(self):
        if self.subscription_operation == None:
            # SubscribeToComponentUpdates returns a tuple with the response and the
 operation.
            _, self.subscription_operation =
 self.ipc_client.subscribe_to_component_updates(
                on_stream_event=self.on_component_update_event)

    def close_subscription(self):
```

```
        if self.subscription_operation is not None:
            self.subscription_operation.close()
            self.subscription_operation = None

    def defer_update(self, deployment_id):
        self.ipc_client.defer_component_update(
            deployment_id=deployment_id,
 recheck_after_ms=DEFER_COMPONENT_UPDATE_INTERVAL)

    def acknowledge_update(self, deployment_id):
        # Specify recheck_after_ms=0 to acknowledge a component update.
        self.ipc_client.defer_component_update(
            deployment_id=deployment_id, recheck_after_ms=0)

    def is_battery_below_threshold(self):
        return self.get_battery_level() < self.battery_threshold

    def get_battery_level(self):
        # Read the battery level from the virtual battery level file.
        with self.battery_file_path.open('r') as f:
            data = json.load(f)
            return float(data['battery_level'])

    def print_message(self):
        message = 'Hello, World!'
        if self.is_battery_below_threshold():
            message += ' Battery level (%d) is below threshold (%d), so the
 component will defer updates' % (
                self.get_battery_level(), self.battery_threshold)
        else:
            message += ' Battery level (%d) is above threshold (%d), so the
 component will acknowledge updates' % (
                self.get_battery_level(), self.battery_threshold)
        print(message)


def main():
    # Read the battery threshold and virtual battery file path from command-line
 args.
    args = sys.argv[1:]
    battery_threshold = float(args[0])
    battery_file_path = Path(args[1])
    print('Reading battery level from %s and deferring updates when below %d' % (
        str(battery_file_path), battery_threshold))
```

```
    try:
        # Create an IPC client and a Hello World printer that defers component
 updates.
        ipc_client = GreengrassCoreIPCClientV2()
        hello_world_printer = BatteryAwareHelloWorldPrinter(
            ipc_client, battery_file_path, battery_threshold)
        hello_world_printer.subscribe_to_component_updates()
        try:
            # Keep the main thread alive, or the process will exit.
            while True:
                hello_world_printer.print_message()
                time.sleep(HELLO_WORLD_PRINT_INTERVAL)
        except InterruptedError:
            print('Subscription interrupted')
        hello_world_printer.close_subscription()
    except Exception:
        print('Exception occurred', file=sys.stderr)
        traceback.print_exc()
        exit(1)


if __name__ == '__main__':
    main()
```

這個 Python 應用程序執行以下操作：

- 從您稍後將在核心裝置上建立的虛擬電池電量檔案讀取核心裝置的電池電量。此虛擬電池電量文件模仿真正的電池，因此您可以在沒有電池的核心設備上完成本教程。
- 讀取電池臨界值的命令列引數，以及虛擬電池電量檔案的路徑。元件方案會根據組態參數設定這些命令列引數，因此您可以在部署元件時自訂這些值。
- 使用適用於 Python 的 V2 中的 IPC 客戶端 AWS IoT Device SDK V2 與AWS IoT Greengrass核心軟件進行通信。與原始 IPC 用戶端相比，IPC 用戶端 V2 可減少在自訂元件中使用 IPC 所需撰寫的程式碼量。
- 訂閱使用 SubscribeToComponentUpdatesIPC 作業更新通知。AWS IoT GreengrassCore 軟體會在每次部署之前和之後傳送通知。每次收到通知時，該組件都會調用以下函數。如果通知是針對即將進行的部署，則元件會檢查電池電量是否低於閾值。如果電池電量低於閾值，則元件會使用 DeferComponentUpdateIPC 操作將更新延遲 30 秒。否則，如果電池電量不低於閾值，則組件會確認更新，因此可以繼續進行更新。

```
def on_component_update_event(self, event):
    try:
        if event.pre_update_event is not None:
            if self.is_battery_below_threshold():
                self.defer_update(event.pre_update_event.deployment_id)
                print('Deferred update for deployment %s' %
                      event.pre_update_event.deployment_id)
            else:
                self.acknowledge_update(
                    event.pre_update_event.deployment_id)
                print('Acknowledged update for deployment %s' %
                      event.pre_update_event.deployment_id)
        elif event.post_update_event is not None:
            print('Applied update for deployment')
    except:
        traceback.print_exc()
```

> **Note**
>
> AWS IoT Greengrass核心軟體不會傳送本機部署的更新通知，因此您可以使用AWS IoT Greengrass雲端服務部署此元件進行測試。

4. 使用文字編輯器在名為`recipe.json`或的檔案中建立元件配方`recipe.yaml`。元件方案會定義元件的中繼資料、預設組態參數，以及平台特定的生命週期指令碼。

   JSON

   例如，在 Linux 系統上，您可以執行下列命令來使用 GNU nano 建立檔案。

   ```
   nano recipe.json
   ```

   將下列 JSON 複製到檔案中。

   ```
   {
     "RecipeFormatVersion": "2020-01-25",
     "ComponentName": "COMPONENT_NAME",
     "ComponentVersion": "COMPONENT_VERSION",
     "ComponentDescription": "This Hello World component defers updates when the
    battery level is below a threshold.",
     "ComponentPublisher": "COMPONENT_AUTHOR",
   ```

```
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "BatteryThreshold": 50,
      "LinuxBatteryFilePath": "/home/ggc_user/virtual_battery.json",
      "WindowsBatteryFilePath": "C:\\Users\\ggc_user\\virtual_battery.json"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "python3 -m pip install --user awsiotsdk --upgrade",
        "run": "python3 -u {artifacts:decompressedPath}/
com.example.BatteryAwareHelloWorld/main.py \"{configuration:/BatteryThreshold}\"
 \"{configuration:/LinuxBatteryFilePath}\""
      },
      "Artifacts": [
        {
          "Uri": "s3://BUCKET_NAME/COMPONENT_NAME/COMPONENT_VERSION/
com.example.BatteryAwareHelloWorld.zip",
          "Unarchive": "ZIP"
        }
      ]
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "py -3 -m pip install --user awsiotsdk --upgrade",
        "run": "py -3 -u {artifacts:decompressedPath}/
com.example.BatteryAwareHelloWorld/main.py \"{configuration:/BatteryThreshold}\"
 \"{configuration:/WindowsBatteryFilePath}\""
      },
      "Artifacts": [
        {
          "Uri": "s3://BUCKET_NAME/COMPONENT_NAME/COMPONENT_VERSION/
com.example.BatteryAwareHelloWorld.zip",
          "Unarchive": "ZIP"
        }
      ]
    }
```

```
  ]
}
```

YAML

例如，在 Linux 系統上，您可以執行下列命令來使用 GNU nano 建立檔案。

```
nano recipe.yaml
```

將下列 YAML 複製到檔案中。

```
---
RecipeFormatVersion: "2020-01-25"
ComponentName: "COMPONENT_NAME"
ComponentVersion: "COMPONENT_VERSION"
ComponentDescription: "This Hello World component defers updates when the
 battery level is below a threshold."
ComponentPublisher: "COMPONENT_AUTHOR"
ComponentConfiguration:
  DefaultConfiguration:
    BatteryThreshold: 50
    LinuxBatteryFilePath: "/home/ggc_user/virtual_battery.json"
    WindowsBatteryFilePath: "C:\\Users\\ggc_user\\virtual_battery.json"
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: python3 -m pip install --user awsiotsdk --upgrade
      run: python3 -u {artifacts:decompressedPath}/
com.example.BatteryAwareHelloWorld/main.py "{configuration:/BatteryThreshold}"
 "{configuration:/LinuxBatteryFilePath}"
    Artifacts:
      - Uri: "s3://BUCKET_NAME/COMPONENT_NAME/COMPONENT_VERSION/
com.example.BatteryAwareHelloWorld.zip"
        Unarchive: ZIP
  - Platform:
      os: windows
    Lifecycle:
      install: py -3 -m pip install --user awsiotsdk --upgrade
      run: py -3 -u {artifacts:decompressedPath}/
com.example.BatteryAwareHelloWorld/main.py "{configuration:/BatteryThreshold}"
 "{configuration:/WindowsBatteryFilePath}"
```

```
    Artifacts:
      - Uri: "s3://BUCKET_NAME/COMPONENT_NAME/COMPONENT_VERSION/
com.example.BatteryAwareHelloWorld.zip"
        Unarchive: ZIP
```

此配方指定以下內容：

- 電池臨界值的預設組態參數、Linux 核心裝置上的虛擬電池檔案路徑，以及 Windows 核心裝置上的虛擬電池檔案路徑。
- 安裝適用於 Python 之 AWS IoT Device SDK v2 的最新版本的`install`生命週期。
- 在中執行 Python 應用程式的`run`生命週期`main.py`。
- 預留位置，例如`COMPONENT_NAME`和`COMPONENT_VERSION`，其中 GDK CLI 會在建置元件方案時取代資訊。

如需有關元件配方的更多資訊，請參閱AWS IoT Greengrass元件配方參考。

步驟 3：將元件發佈至AWS IoT Greengrass服務

在本節中，您將 Hello World 元件發佈到AWS IoT Greengrass雲端服務。在AWS IoT Greengrass雲端服務中可用元件之後，您可以將其部署到核心裝置。您可以使用 GDK CLI 將元件從開發電腦發佈到AWS IoT Greengrass雲端服務。GDK CLI 會為您上傳元件的配方和成品。

若要將 Hello World 元件發佈至AWS IoT Greengrass服務

1. 執行下列命令，以使用 GDK CLI 建置元件。組件構建命令會根據 GDK CLI 配置文件創建方案和成品。在此程序中，GDK CLI 會建立包含元件原始程式碼的 ZIP 檔案。

   ```
   gdk component build
   ```

   您應該會看到類似下列範例的訊息。

   ```
   [2022-04-28 11:20:16] INFO - Getting project configuration from gdk-config.json
   [2022-04-28 11:20:16] INFO - Found component recipe file 'recipe.yaml' in the
    project directory.
   [2022-04-28 11:20:16] INFO - Building the component
    'com.example.BatteryAwareHelloWorld' with the given project configuration.
   [2022-04-28 11:20:16] INFO - Using 'zip' build system to build the component.
   ```

```
[2022-04-28 11:20:16] WARNING - This component is identified as using 'zip' build
 system. If this is incorrect, please exit and specify custom build command in the
 'gdk-config.json'.
[2022-04-28 11:20:16] INFO - Zipping source code files of the component.
[2022-04-28 11:20:16] INFO - Copying over the build artifacts to the greengrass
 component artifacts build folder.
[2022-04-28 11:20:16] INFO - Updating artifact URIs in the recipe.
[2022-04-28 11:20:16] INFO - Creating component recipe in 'C:\Users\finthomp
\greengrassv2\com.example.BatteryAwareHelloWorld\greengrass-build\recipes'.
```

2. 執行下列命令，將元件發佈至AWS IoT Greengrass雲端服務。元件發佈命令會將元件的 ZIP 檔案成品上傳至 S3 儲存貯體。然後，它會更新元件方案中的 ZIP 檔案的 S3 URI，並將配方上傳至AWS IoT Greengrass服務。在此過程中，GDK CLI 會檢查AWS IoT Greengrass雲端服務中已經可用的 Hello World 元件版本，因此它可以選擇該版本之後的下一個修補程式版本。如果元件尚不存在，GDK CLI 會使用版本`1.0.0`。

   ```
   gdk component publish
   ```

   您應該會看到類似下列範例的訊息。輸出會告訴您 GDK CLI 建立的元件版本。

   ```
   [2022-04-28 11:20:29] INFO - Getting project configuration from gdk-config.json
   [2022-04-28 11:20:29] INFO - Found component recipe file 'recipe.yaml' in the
    project directory.
   [2022-04-28 11:20:29] INFO - Found credentials in shared credentials file: ~/.aws/
   credentials
   [2022-04-28 11:20:30] INFO - No private version of the component
    'com.example.BatteryAwareHelloWorld' exist in the account. Using '1.0.0' as the
    next version to create.
   [2022-04-28 11:20:30] INFO - Publishing the component
    'com.example.BatteryAwareHelloWorld' with the given project configuration.
   [2022-04-28 11:20:30] INFO - Uploading the component built artifacts to s3 bucket.
   [2022-04-28 11:20:30] INFO - Uploading component artifacts to S3
    bucket: greengrass-component-artifacts-us-west-2-123456789012. If this is your
    first time using this bucket, add the 's3:GetObject' permission to each core
    device's token exchange role to allow it to download the component artifacts. For
    more information, see https://docs.aws.amazon.com/greengrass/v2/developerguide/
   device-service-role.html.
   [2022-04-28 11:20:30] INFO - Not creating an artifacts bucket as it already exists.
   [2022-04-28 11:20:30] INFO - Updating the component recipe
    com.example.BatteryAwareHelloWorld-1.0.0.
   [2022-04-28 11:20:31] INFO - Creating a new greengrass component
    com.example.BatteryAwareHelloWorld-1.0.0
   ```

```
[2022-04-28 11:20:31] INFO - Created private version '1.0.0' of the component in
 the account.'com.example.BatteryAwareHelloWorld'.
```

3. 從輸出複製 S3 儲存貯體名稱。稍後您可以使用值區名稱，以允許核心裝置從此值區下載元件成品。
4. (選擇性) 在AWS IoT Greengrass主控台中檢視元件，以確認該元件已成功上傳。請執行下列操作：

   a. 在AWS IoT Greengrass主控台瀏覽功能表中，選擇 [元件]。

   b. 在 [元件] 頁面上，選擇 [我的元件] 索引標籤，然後選擇com.example.BatteryAwareHelloWorld。

      在此頁面上，您可以查看元件的配方以及元件的其他相關資訊。

5. 允許核心裝置存取 S3 儲存貯體中的元件成品。

   每個核心裝置都有一個核心裝置 IAM 角色，可讓其與雲端互動AWS IoT並將記錄檔傳送到AWS雲端。依預設，此裝置角色不允許存取 S3 儲存貯體，因此您必須建立並附加政策，以允許核心裝置從 S3 儲存貯體擷取元件成品。

   如果您的裝置角色已允許存取 S3 儲存貯體，則可以略過此步驟。否則，請建立允許存取的 IAM 政策，並將其附加到角色，如下所示：

   a. 在 IAM 主控台導覽功能表中，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。將 *greengrass-component-artifacts-us-WEST 2-123456789012* 取代為 GDK CLI 上傳元件成品的 S3 儲存貯體的名稱。

      例如，如果您在 GDK CLI 組態檔**us-west-2**中指定了 **greengrass-component-artifacts** AND，而您的AWS 帳戶識別碼是**123456789012**，GDK CLI 會使用名為的 S3 儲存貯體。greengrass-component-artifacts-us-west-2-123456789012

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
```

```
      "Resource": "arn:aws:s3:::greengrass-component-artifacts-us-
west-2-123456789012/*"
    }
  ]
}
```

c. 選擇下一步。

d. 在「策略詳細資料」區段中，輸入做為「名稱」**MyGreengrassV2ComponentArtifactPolicy**。

e. 選擇建立政策。

f. 在 IAM 主控台導覽功能表中，選擇 [角色]，然後選擇核心裝置的角色名稱。您在安裝 AWS IoT Greengrass Core 軟體時指定了這個角色名稱。如果您未指定名稱，則預設值為`GreengrassV2TokenExchangeRole`。

g. 在「權限」下，選擇「新增權限」，然後選擇「附加策略

h. 在 [新增權限] 頁面上，選取您建立之`MyGreengrassV2ComponentArtifactPolicy`原則旁邊的核取方塊，然後選擇 [新增權限]。

步驟 4：在核心裝置上部署和測試元件

在本節中，您要將元件部署到核心裝置以測試其功能。在核心裝置上，您可以建立虛擬電池電量檔案以模仿真實電池。然後，您可以建立其他部署並觀察核心裝置上的元件記錄檔，以查看元件延期並確認更新。

若要部署和測試延遲更新的 Hello World 元件

1. 使用文字編輯器建立虛擬電池電量檔案。這個文件模仿了一個真正的電池。

   - 在 Linux 核心裝置上，建立名為`/home/ggc_user/virtual_battery.json`. 使用`sudo`權限執行文字編輯器。
   - 在 Windows 核心裝置上，建立名為`C:\Users\ggc_user\virtual_battery.json`. 以系統管理員身分執行文字編輯器。

   例如，在 Linux 系統上，您可以執行下列命令來使用 GNU nano 建立檔案。

   ```
   sudo nano /home/ggc_user/virtual_battery.json
   ```

   將下列 JSON 複製到檔案中。

```
{
  "battery_level": 50
}
```

2. 將 Hello World 元件部署到核心裝置。請執行下列操作：

   a. 在AWS IoT Greengrass主控台瀏覽功能表中，選擇 [元件]。

   b. 在 [元件] 頁面上，選擇 [我的元件] 索引標籤，然後選擇com.example.BatteryAwareHelloWorld。

   c. 在 com.example.BatteryAwareHelloWorld頁面中，選擇部署。

   d. 從 [新增至部署] 中，選擇要修訂的現有部署，或選擇建立新部署，然後選擇 [下一步]。

   e. 如果您選擇建立新部署，請為部署選擇目標核心裝置或物件群組。在 [指定目標] 頁面的 [部署目標] 下，選擇核心裝置或物件群組，然後選擇 [下一步]。

   f. 在 [選取元件] 頁面上，確認已選取com.example.BatteryAwareHelloWorld元件，然後選擇 [下一步]。

   g. 在 [設定元件] 頁面上，選取 com.example.BatteryAwareHelloWorld，然後執行下列動作：

      i. 選擇 設定元件。

      ii. 在設定com.example.BatteryAwareHelloWorld強制回應的組態更新下，在要合併的組態中，輸入下列組態更新。

      ```
      {
        "BatteryThreshold": 70
      }
      ```

      iii. 選擇 [確認] 關閉強制回應，然後選擇 [下一步]。

   h. 在 [確認進階設定] 頁面的 [部署原則] 區段的 [元件更新原則] 下，確認已選取 [通知元件]。建立新部署時，依預設會選取「通知元件」。

   i. 在 Review (檢閱) 頁面，選擇 Deploy (部署)。

   部署最多可能需要一分鐘的時間才能完成。

3. AWS IoT Greengrass核心軟件將標準輸出從組件進程保存到文件夾中的日誌文件`logs`。執行下列命令以確認 Hello World 元件是否執行並列印狀態訊息。

Linux or Unix

```
sudo tail -f /greengrass/v2/logs/com.example.BatteryAwareHelloWorld.log
```

Windows Command Prompt (CMD)

```
type C:\greengrass\v2\logs\com.example.BatteryAwareHelloWorld.log
```

PowerShell

```
gc C:\greengrass\v2\logs\com.example.BatteryAwareHelloWorld.log -Tail 10 -Wait
```

您應該會看到類似下列範例的訊息。

```
Hello, World! Battery level (50) is below threshold (70), so the component will
 defer updates.
```

> **Note**
>
> 如果檔案不存在，則部署可能尚未完成。如果檔案在 30 秒內不存在，則部署可能會失敗。例如，如果核心裝置沒有從 S3 儲存貯體下載元件成品的權限，就會發生這種情況。執行下列命令以檢視AWS IoT Greengrass核心軟體記錄檔。此檔案包含來自 Greengrass 核心裝置部署服務的記錄檔。
>
> Linux or Unix
>
> ```
> sudo tail -f /greengrass/v2/logs/greengrass.log
> ```
>
> Windows Command Prompt (CMD)
>
> ```
> type C:\greengrass\v2\logs\greengrass.log
> ```
>
> 該type命令將文件的內容寫入終端。多次執行此命令以觀察檔案中的變更。

PowerShell

```
gc C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
```

4. 建立核心裝置的新部署，以確認元件是否延期更新。請執行下列操作：

   a. 在AWS IoT Greengrass主控台導覽功能表中，選擇「部署」。

   b. 選擇您先前建立或修訂的部署。

   c. 在部署頁面上，選擇「修訂」。

   d. 在「修訂部署」模式中，選擇「修訂部署」。

   e. 在每個步驟中選擇 [下一步]，然後選擇 [部署]。

5. 執行下列命令以再次檢視元件的記錄，並確認其延遲更新。

   Linux or Unix

   ```
   sudo tail -f /greengrass/v2/logs/com.example.BatteryAwareHelloWorld.log
   ```

   Windows Command Prompt (CMD)

   ```
   type C:\greengrass\v2\logs\com.example.BatteryAwareHelloWorld.log
   ```

   PowerShell

   ```
   gc C:\greengrass\v2\logs\com.example.BatteryAwareHelloWorld.log -Tail 10 -Wait
   ```

   您應該會看到類似下列範例的訊息。元件會延遲更新 30 秒，因此元件會重複列印此訊息。

   ```
   Deferred update for deployment 50722a95-a05f-4e2a-9414-da80103269aa.
   ```

6. 使用文字編輯器編輯虛擬電池電量檔案，並將電池電量變更為高於閾值的值，以便繼續進行部署。

   - 在 Linux 核心裝置上，編輯名為/home/ggc_user/virtual_battery.json. 使用sudo權限執行文字編輯器。
   - 在 Windows 核心裝置上，編輯名為的檔案C:\Users\ggc_user\virtual_battery.json。以系統管理員身分執行文字編輯器。

例如，在 Linux 系統上，您可以執行下列命令來使用 GNU nano 建立檔案。

```
sudo nano /home/ggc_user/virtual_battery.json
```

將電池電量變更為80。

```
{
  "battery_level": 80
}
```

7. 執行下列命令以再次檢視元件的記錄檔，並確認其確認更新。

   Linux or Unix

   ```
   sudo tail -f /greengrass/v2/logs/com.example.BatteryAwareHelloWorld.log
   ```

   Windows Command Prompt (CMD)

   ```
   type C:\greengrass\v2\logs\com.example.BatteryAwareHelloWorld.log
   ```

   PowerShell

   ```
   gc C:\greengrass\v2\logs\com.example.BatteryAwareHelloWorld.log -Tail 10 -Wait
   ```

   您應該會看到類似下列範例的訊息。

   ```
   Hello, World! Battery level (80) is above threshold (70), so the component will
    acknowledge updates.
   Acknowledged update for deployment f9499eb2-4a40-40a7-86c1-c89887d859f1.
   ```

您已完成此教學課程。Hello World 元件會根據核心裝置的電池電量延遲或確認更新。如需本自學課程探討之主題的詳細資訊，請參閱下列內容：

- 開發AWS IoT Greengrass元件
- 將AWS IoT Greengrass元件部署到裝置
- 使AWS IoT Device SDK用與 Greengrass 核、其他元件和通訊 AWS IoT Core

- AWS IoT Greengrass開發套件命令列介面

教學課程：透過 MQTT 與本地 IoT 裝置互動

您可以完成此教學課程，將核心裝置設定為與透過 MQTT 連線至核心裝置的本機 IoT 裝置 (稱為用戶端裝置) 互動。在本教學課程中，您會設定AWS IoT要使用雲端探索連線至核心裝置做為用戶端裝置的項目。當您設定雲端探索時，用戶端裝置可以傳送要求至AWS IoT Greengrass雲端服務以探索核心裝置。來源的回應AWS IoT Greengrass包括您設定要探查的用戶端裝置之核心裝置的連線資訊和憑證。然後，用戶端裝置可以使用此資訊連接到可用的核心裝置，在此裝置可透過 MQTT 進行通訊。

在此教學課程中，您將執行下列操作：

1. 如有需要，請檢閱並更新核心裝置的權限。
2. 將用戶端裝置與核心裝置建立關聯，以便他們可以使用雲端探索來探索核心裝置。
3. 將 Greengrass 元件部署到核心裝置，以啟用用戶端裝置支援。
4. 將用戶端裝置 Connect 至核心裝置，並測試與AWS IoT Core雲端服務的通訊。
5. 開發與用戶端裝置通訊的自訂 Greengrass 元件。
6. 開發與用戶端AWS IoT裝置裝置陰影互動的自訂元件。

本教學課程使用單一核心裝置和單一用戶端裝置。您也可以按照教學課程連接和測試多個客戶端設備。

您可以預期在本教程中花費 30-60 分鐘。

必要條件

為了完成本教學，您需要以下項目：

- AWS 帳戶。如果您沒有帳戶，請參閱 設置一個 AWS 帳戶。
- 具有管理員權限的 AWS Identity and Access Management (IAM) 使用者。
- 一個 Greengrass 核心設備。如需如何設定核心裝置的詳細資訊，請參閱設定AWS IoT Greengrass核心裝置。
  - 核心裝置必須執行 Greengrass 核 v2.6.0 或更新版本。此版本包括本地發布/訂閱通信中的通配符支持以及客戶端設備陰影的支持。

> **Note**
>
> 用戶端裝置支援需要 Greengrass 核 v2.2.0 或更新版本。不過，本教學課程將探討較新的功能，例如支援本機發佈/訂閱中的 MQTT 萬用字元，以及用戶端裝置陰影的支援。這些功能需 Greengrass v2.6.0 或更高版本。

  - 核心裝置必須與用戶端裝置位於相同的網路上才能連線。
  - (選擇性) 若要完成開發自訂 Greengrass 元件的模組，核心裝置必須執行 Greengrass CLI。如需詳細資訊，請參閱 安裝 Greengrass CLI。
- 在本教程中作為客戶端設備連接的AWS IoT東西。如需詳細資訊，請參閱AWS IoT Core開發人員指南中的建立AWS IoT資源。
  - 用戶端裝置的AWS IoT策略必須允許`greengrass:Discover`權限。如需詳細資訊，請參閱 用戶端裝置的最低AWS IoT原則。
  - 用戶端裝置必須與核心裝置位於相同的網路上。
  - 用戶端裝置必須執行 Python 3。
  - 用戶端裝置必須執行 Git。

步驟 1：檢閱並更新核心裝置AWS IoT原則

若要支援用戶端裝置，核心裝置的AWS IoT策略必須允許下列權限：

- `greengrass:PutCertificateAuthorities`
- `greengrass:VerifyClientDeviceIdentity`
- `greengrass:VerifyClientDeviceIoTCertificateAssociation`
- `greengrass:GetConnectivityInfo`
- `greengrass:UpdateConnectivityInfo`— (選用) IP 偵測器元件需要此權限，IP 偵測器元件會向AWS IoT Greengrass雲端服務報告核心裝置的網路連線資訊。

如需有關核心裝置的這些權限和AWS IoT原則的詳細資訊，請參閱資料平面操作的AWS IoT 政策和支援用戶端裝置的最低AWS IoT原則。

在本節中，您可以檢閱核心裝置的AWS IoT原則，並新增任何遺失的必要權限。如果您使用 AWS IoT GreengrassCore 軟體安裝程式佈建資源，您的核心裝置會有允許存取所有AWS IoT Greengrass動作

的AWS IoT原則 (`greengrass:*`)。在此情況下，只有當您計劃將陰影管理員元件設定為同步處理裝置陰影時，才必須更新AWS IoT原則AWS IoT Core。否則，您可以跳過此部分。

檢閱和更新核心裝置的AWS IoT政策

1. 在AWS IoT Greengrass主控台瀏覽功能表中，選擇 [核心裝置]。
2. 在 [核心裝置] 頁面上，選擇要更新的核心裝置。
3. 在核心裝置詳細資料頁面上，選擇核心裝置物件的連結。此連結會在AWS IoT主控台中開啟物件詳細資訊頁面。
4. 在物件詳細資訊頁面上，選擇 [憑證]。
5. 在「憑證」索引標籤中，選擇物件的使用中憑證。
6. 在憑證詳細資料頁面上，選擇 [原則]。
7. 在「策略」索引標籤中，選擇要檢閱和更新的AWS IoT策略。您可以將必要的權限新增至任何附加至核心裝置作用中憑證的原則。

   > **Note**
   >
   > 如果您使用 AWS IoT GreengrassCore 軟體安裝程式佈建資源，則有兩個AWS IoT原則。建議您選擇名為GreengrassV2IoTThingPolicy的策略 (如果存在的話)。依預設，您使用快速安裝程式建立的核心裝置會使用此原則名稱。如果您將權限新增至此原則，也會將這些權限授與使用此原則的其他核心裝置。

8. 在原則概觀中，選擇 [編輯作用中版本]。
9. 檢閱所需權限的原則，並新增任何遺失的必要權限。
10. 若要將新的原則版本設定為作用中版本，請在 [原則版本狀態] 下選取 [將編輯的版本設定為此原則的作用中版本]。
11. 選擇「另存為新版本」。

步驟 2：啟用用戶端裝置支援

若要讓用戶端裝置使用雲端探索連線至核心裝置，您必須關聯這些裝置。當您將用戶端裝置與核心裝置建立關聯時，您可以讓該用戶端裝置擷取核心裝置的 IP 位址和憑證以用於連線。

若要讓用戶端裝置安全地連線至核心裝置並與 Greengrass 元件通訊AWS IoT Core，請將下列 Greengrass 元件部署到核心裝置：

- 用戶端裝置驗證 (aws.greengrass.clientdevices.Auth)

  部署用戶端裝置驗證元件以驗證用戶端裝置並授權用戶端裝置動作。這個組件允許你的AWS IoT東西連接到一個核心設備。

  此組件需要一些配置才能使用它。您必須指定用戶端裝置群組，以及每個群組獲授權執行的作業，例如透過 MQTT 進行連線和通訊。如需詳細資訊，請參閱用戶端裝置驗證元件組態。

- MQTT 3.1.1 經紀商 (平均) (aws.greengrass.clientdevices.mqtt.Moquette)

  部署 Moquette MQTT 代理程式元件以執行輕量型 MQTT 代理程式。Moquette MQTT 代理程式符合 MQTT 3.1.1 規範，並包含對 QoS 0、QoS 1、QoS 2、保留訊息、最後將訊息和持續訂閱的本機支援。

  您不需要配置此組件即可使用它。不過，您可以設定此元件操作 MQTT 代理程式的連接埠。依預設，它會使用連接埠 8883。

- MQTT 大橋 (aws.greengrass.clientdevices.mqtt.Bridge)

  (選擇性) 部署 MQTT 橋接器元件，以在用戶端裝置 (本機 MQTT)、本機發佈/訂閱和 MQTT 之間轉送訊息。AWS IoT Core設定此元件，以便與 Greengrass 元件的用戶端裝置同步處理用戶端裝置，AWS IoT Core並與其互動。

  此元件需要組態才能使用。您必須指定此元件轉送訊息的主題對映。如需詳細資訊，請參閱 MQTT 橋接器元件組態。

- IP 偵測器 (aws.greengrass.clientdevices.IPDetector)

  (選擇性) 部署 IP 偵測器元件，以自動向AWS IoT Greengrass雲端服務報告核心裝置的 MQTT 代理程式端點。如果您有複雜的網路設定，例如路由器將 MQTT 代理程式連接埠轉送至核心裝置的網路設定，則無法使用此元件。

  您不需要配置此組件即可使用它。

在本節中，您可以使用AWS IoT Greengrass主控台來關聯用戶端裝置，並將用戶端裝置元件部署到核心裝置。

啟用用戶端裝置支援

1. 導覽至 AWS IoT Greengrass主控台。
2. 在左側導覽選單中，選擇 [核心裝置]。

3. 在 [核心裝置] 頁面上，選擇要啟用用戶端裝置支援的核心裝置。
4. 在核心裝置詳細資料頁面上，選擇 [用戶端裝置] 索引標籤。
5. 在用戶端裝置索引標籤上，選擇 [設定雲端探索]。

   [設定核心裝置探索] 頁面隨即開啟。您可以在此頁面上建立用戶端裝置與核心裝置的關聯，並部署用戶端裝置元件。此頁面會在 「步驟 1：選取目標核心裝置」中為您選取核心裝置。

   > **Note**
   >
   > 您也可以使用此頁面來設定物件群組的核心裝置探索。如果您選擇此選項，您可以將用戶端裝置元件部署到物件群組中的所有核心裝置。但是，如果您選擇此選項，則必須在稍後建立部署後，手動將用戶端裝置與每個核心裝置相關聯。在本教學課程中，您會設定單一核心裝置。

6. 在步驟 2：關聯用戶端裝置中，將用戶端裝置的AWS IoT物件與核心裝置建立關聯。這可讓用戶端裝置使用雲端探索擷取核心裝置的連線資訊和憑證。請執行下列操作：

   a. 選擇關聯用戶端裝置。
   b. 在「關聯用戶端裝置與核心裝置」強制回應中，輸入要關聯之AWS IoT物件的名稱。
   c. 選擇新增。
   d. 選擇 Associate (關聯)。

7. 在步驟 3：設定和部署 Greengrass 元件中，部署元件以啟用用戶端裝置支援。如果目標核心裝置具有先前的部署，則此頁面會修訂該部署。否則，此頁面會為核心裝置建立新部署。執行下列動作來設定和部署用戶端裝置元件：

   a. 核心裝置必須執行 Greengrass 核 v2.6.0 或更新版本才能完成本教學課程。如果核心裝置執行舊版，請執行下列動作：

      i. 選取方塊以部署aws.greengrass.Nucleus元件。
      ii. 對於aws.greengrass.Nucleus零組件，請選擇編輯模型組態。
      iii. 對於「元件」版本，請選擇版本 2.6.0 或更新版本。
      iv. 選擇確認。

> **Note**
>
> 如果您從較早的次要版本升級 Greengrass 核心核心，且核心裝置執行依賴於核心的元件，您也必須將AWS提供的元件更新為較新版本。AWS稍後在本自學課程中檢閱部署時，您可以規劃這些元件的版本。如需詳細資訊，請參閱 更新AWS IoT Greengrass核心軟件（OTA）。

b. 對於aws.greengrass.clientdevices.Auth零組件，請選擇編輯模型組態。

c. 在用戶端裝置驗證元件的編輯組態模式中，設定允許用戶端裝置在核心裝置上發佈和訂閱 MQTT 代理程式的授權原則。請執行下列操作：

   i. 在 [設定] 下的 [合併程式碼區塊的組態] 中，輸入下列組態，其中包含用戶端裝置授權原則。每個裝置群組授權原則都會指定一組處理行動以及用戶端裝置可以在其上執行這些處理行動的資源。

   - 此原則允許名稱開頭MyClientDevice為的用戶端裝置在所有 MQTT 主題上連線並進行通訊。將 *MyClientDevice** 取代為要連接為用戶端裝置的AWS IoT物件名稱。您也可以使用與用戶端裝置名稱相符的*萬用字元來指定名稱。*萬用字元必須位於名稱的末尾。

     如果您有第二個用戶端裝置要連線，請將 *MyOtherClientDevice** 取代為該用戶端裝置的名稱，或符合該用戶端裝置名稱的萬用字元模式。否則，您可以移除或保留選取規則的這個區段，以允許名稱相符的用戶端裝置MyOtherClientDevice*進行連線和通訊。

   - 此原則使用OR操作員也允許名稱開頭為的用戶端裝置在所有 MQTT 主題上進行連線和通訊。MyOtherClientDevice您可以在選取規則中移除此子句，或修改它以符合要連線的用戶端裝置。

   - 此原則允許用戶端裝置針對所有 MQTT 主題發佈和訂閱。若要遵循最佳安全性做法，請將mqtt:publish和mqtt:subscribe作業限制為用戶端裝置用於通訊的最小主題集。

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
```

```
        "selectionRule": "thingName: MyClientDevice* OR
 thingName: MyOtherClientDevice*",
        "policyName": "MyClientDevicePolicy"
      }
    },
    "policies": {
      "MyClientDevicePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish to all
 topics.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to all
 topics.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}
```

如需詳細資訊，請參閱用戶端裝置驗證元件組態。

ii. 選擇確認。

d. 對於aws.greengrass.clientdevices.mqtt.Bridge零組件，請選擇編輯模型組態。

e. 在 MQTT 橋接器元件的編輯組態模式中，設定將 MQTT 訊息從用戶端裝置轉送至的主題對應。AWS IoT Core請執行下列操作：

i. 在「組態」下，在「要合併程式碼的組態」區塊中，輸入下列設定。此組態指定將`clients/+/hello/world`主題篩選器的 MQTT 訊息從用戶端裝置轉送至AWS IoT Core雲端服務。例如，此主題篩選符合`clients/MyClientDevice1/hello/world`主題。

```
{
  "mqttTopicMapping": {
    "HelloWorldIotCoreMapping": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

如需詳細資訊，請參閱 MQTT 橋接器元件組態。

ii. 選擇確認。

8. 選擇複查並建置，以複查此頁面為您建立的部署。

9. 如果您先前尚未在此區域中設定 Greengrass 服務角色，主控台會開啟強制回應來為您設定服務角色。用戶端裝置驗證元件使用此服務角色來驗證用戶端裝置的身分識別，而 IP 偵測器元件則使用此服務角色來管理核心裝置連線資訊。選擇 授予許可。

10. 在「複查」頁面上，選擇「部署」以開始部署至核心裝置。

11. 若要驗證部署是否成功，請檢查部署狀態，然後檢查核心裝置上的記錄檔。若要檢查核心裝置上的部署狀態，您可以在部署概觀中選擇目標。如需詳細資訊，請參閱下列內容：

- 檢查部署狀態
- 監控AWS IoT Greengrass日誌

步驟 3：Connect 用戶端裝置

用戶端裝置可以使用AWS IoT Device SDK來探索、連線和與核心裝置進行通訊。客戶端設備必須是一AWS IoT件事。如需詳細資訊，請參閱AWS IoT Core開發人員指南中的建立物件物件。

在本節中，您將安裝適用於 Python Greengrass AWS IoT Device SDK v2，並從. AWS IoT Device SDK

> **Note**
>
> 也提供其他程式設計語言版本AWS IoT Device SDK。本教學課程將 AWS IoT Device SDK v2 用於 Python，但您可以針對您的使用案例探索其他軟體開發套件。如需詳細資訊，請參閱AWS IoT Core開發人員指南中的AWS IoT裝置 SDK。

將用戶端裝置連線到核心裝置

1. 下載並安裝適用於 Python 的 AWS IoT Device SDK v2 到作為客戶端設備連接的AWS IoT東西。

   在用戶端裝置上，執行下列動作：

   a. 克隆 AWS IoT Device SDK v2 的 Python 存儲庫下載它。

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

   b. 安裝適用於 Python 的 AWS IoT Device SDK V2。

   ```
   python3 -m pip install --user ./aws-iot-device-sdk-python-v2
   ```

2. 切換到示例文件夾在 AWS IoT Device SDK v2 中的 Python。

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

3. 執行範例 Greengrass 探索應用程式。此應用程式需要引數，這些引數會指定用戶端裝置物件名稱、要使用的 MQTT 主題和訊息，以及驗證和保護連線的憑證。下列範例會將 Hello World 訊息傳送至`clients/`*`MyClientDevice1`*`/hello/world`主題。

   > **Note**
   >
   > 本主題與您將 MQTT 橋接器設定為將郵件轉送至較早版本的主題相符。AWS IoT Core

   - 將 *`MyClientDevice1`* 取代為用戶端裝置的物件名稱。
   - 將 *`~/##/ AmazonRoot CA1.pem`* 取代為用戶端裝置上 Amazon 根 CA 憑證的路徑。

- 將*#/##/## .pem.crt* 替換為客戶端設備上的設備證書的路徑。
- 將*#/##/## .pem.key ###########*件的路徑。
- 將 *us-east-1* 取代為用戶端裝置和核心裝置運作所在的AWS區域。

```
python3 basic_discovery.py \\
  --thing_name MyClientDevice1 \\
  --topic 'clients/MyClientDevice1/hello/world' \\
  --message 'Hello World!' \\
  --ca_file ~/certs/AmazonRootCA1.pem \\
  --cert ~/certs/device.pem.crt \\
  --key ~/certs/private.pem.key \\
  --region us-east-1 \\
  --verbosity Warn
```

探索範例應用程式會傳送訊息 10 次並中斷連線。它也會訂閱發佈訊息的相同主題。如果輸出指出應用程式收到有關該主題的 MQTT 訊息，則用戶端裝置可以成功與核心裝置通訊。

```
Performing greengrass discovery...
awsiot.greengrass_discovery.DiscoverResponse(gg_groups=[awsiot.greengrass_discovery.GGGroup
coreDevice-MyGreengrassCore',
 cores=[awsiot.greengrass_discovery.GGCore(thing_arn='arn:aws:iot:us-
east-1:123456789012:thing/MyGreengrassCore',
 connectivity=[awsiot.greengrass_discovery.ConnectivityInfo(id='203.0.113.0',
 host_address='203.0.113.0', metadata='', port=8883)])],
 certificate_authorities=['-----BEGIN CERTIFICATE-----\
MIICiT...EXAMPLE=\
-----END CERTIFICATE-----\
'])])
Trying core arn:aws:iot:us-east-1:123456789012:thing/MyGreengrassCore at host
 203.0.113.0 port 8883
Connected!
Published topic clients/MyClientDevice1/hello/world: {"message": "Hello World!",
 "sequence": 0}

Publish received on topic clients/MyClientDevice1/hello/world
b'{"message": "Hello World!", "sequence": 0}'
Published topic clients/MyClientDevice1/hello/world: {"message": "Hello World!",
 "sequence": 1}

Publish received on topic clients/MyClientDevice1/hello/world
```

```
b'{"message": "Hello World!", "sequence": 1}'

...

Published topic clients/MyClientDevice1/hello/world: {"message": "Hello World!",
 "sequence": 9}

Publish received on topic clients/MyClientDevice1/hello/world
b'{"message": "Hello World!", "sequence": 9}'
```

如果應用程式輸出錯誤，請參閱 Greengrass 探索問題疑難排解。

您也可以在核心裝置上檢視 Greengrass 記錄檔，以驗證用戶端裝置是否成功連線並傳送訊息。如需詳細資訊，請參閱 監控AWS IoT Greengrass日誌。

4. 確認 MQTT 橋接器是否將訊息從用戶端裝置轉送到。AWS IoT Core您可以使用主控台中的 MQTT 測試用戶端來訂閱 MQTT AWS IoT Core 主題篩選器。請執行下列操作：

   a. 導覽至 AWS IoT主控台。
   b. 在左側導覽功能表的 [測試] 下，選擇 [MQTT 測試用戶端]。
   c. 在 [訂閱主題] 索引標籤上，對於 [主題] 篩選器，輸入`clients/+/hello/world`以訂閱來自核心裝置的用戶端裝置訊息。
   d. 選擇 Subscribe (訂閱)。
   e. 再次在用戶端裝置上執行發佈/訂閱應用程式。

      MQTT 測試用戶端會針對符合此主題篩選器的主題，顯示您從用戶端裝置傳送的訊息。

步驟 4：開發與用戶端裝置通訊的元件

您可以開發與用戶端裝置通訊的 Greengrass 元件。元件使用處理序間通訊 (IPC) 和本機發佈/訂閱介面，在核心裝置上進行通訊。若要與用戶端裝置互動，請將 MQTT 橋接器元件設定為在用戶端裝置和本機發佈/訂閱介面之間轉送訊息。

在本節中，您將更新 MQTT 橋接器元件，以將訊息從用戶端裝置轉送至本機發佈/訂閱介面。然後，您開發一個組件，該組件可以訂閱這些消息，並在收到消息時打印消息。

開發與用戶端裝置通訊的元件

1. 修訂核心裝置的部署，並將 MQTT 橋接器元件設定為將訊息從用戶端裝置轉送至本機發佈/訂閱。請執行下列操作：

a. 導覽至 AWS IoT Greengrass主控台。

b. 在左側導覽選單中，選擇 [核心裝置]。

c. 在 [核心裝置] 頁面上，選擇您在本教學課程中使用的核心裝置。

d. 在核心裝置詳細資料頁面上，選擇 [用戶端裝置] 索引標籤。

e. 在用戶端裝置索引標籤上，選擇 [設定雲端探索]。

   [設定核心裝置探索] 頁面隨即開啟。在此頁面上，您可以變更或設定要部署到核心裝置的用戶端裝置元件。

f. 在步驟 3 中，為零組aws.greengrass.clientdevices.mqtt.Bridge件選擇編輯模型組態。

g. 在 MQTT 橋接器元件的編輯組態模式中，設定將 MQTT 訊息從用戶端裝置轉送至本機發佈/訂閱介面的主題對應。請執行下列操作：

   i. 在「組態」下，在「要合併程式碼的組態」區塊中，輸入下列設定。此組態指定將符合主題篩選器的主題從用戶端裝置轉送至AWS IoT Core雲端服務和本機 Greengrass 發佈/訂閱代理程式的 MQTT 訊息。`clients/+/hello/world`

   ```
   {
     "mqttTopicMapping": {
       "HelloWorldIotCoreMapping": {
         "topic": "clients/+/hello/world",
         "source": "LocalMqtt",
         "target": "IotCore"
       },
       "HelloWorldPubsubMapping": {
         "topic": "clients/+/hello/world",
         "source": "LocalMqtt",
         "target": "Pubsub"
       }
     }
   }
   ```

   如需詳細資訊，請參閱 MQTT 橋接器元件組態。

   ii. 選擇確認。

h. 選擇複查並建置，以複查此頁面為您建立的部署。

i. 在「複查」頁面上，選擇「部署」以開始部署至核心裝置。

j. 若要驗證部署是否成功，請檢查部署狀態，然後檢查核心裝置上的記錄檔。若要檢查核心裝置上的部署狀態，您可以在部署概觀中選擇目標。如需詳細資訊，請參閱下列內容：

- 檢查部署狀態
- 監控AWS IoT Greengrass日誌

2. 開發並部署 Greengrass 元件，該元件會從用戶端裝置訂閱 Hello World 訊息。請執行下列操作：

   a. 在核心裝置上建立配方和成品的資料夾。

   Linux or Unix

   ```
   mkdir recipes
   mkdir -p artifacts/com.example.clientdevices.MyHelloWorldSubscriber/1.0.0
   ```

   Windows Command Prompt (CMD)

   ```
   mkdir recipes
   mkdir artifacts\com.example.clientdevices.MyHelloWorldSubscriber\1.0.0
   ```

   PowerShell

   ```
   mkdir recipes
   mkdir artifacts\com.example.clientdevices.MyHelloWorldSubscriber\1.0.0
   ```

   > **Important**
   >
   > 您必須針對人工因素資料夾路徑使用下列格式。包括您在方案中指定的元件名稱和版本。
   >
   > ```
   > artifacts/componentName/componentVersion/
   > ```

   b. 使用文字編輯器建立包含下列內容的元件配方。此配方指定為 Python 安裝 AWS IoT Device SDK v2，並執行訂閱主題並列印訊息的指令碼。

   例如，在 Linux 系統上，您可以執行下列命令來使用 GNU nano 來建立檔案。

   ```
   nano recipes/com.example.clientdevices.MyHelloWorldSubscriber-1.0.0.json
   ```

   將下列配方複製到檔案中。

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.clientdevices.MyHelloWorldSubscriber",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that subscribes to Hello World messages
 from client devices.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.clientdevices.MyHelloWorldSubscriber:pubsub:1": {
            "policyDescription": "Allows access to subscribe to all topics.",
            "operations": [
              "aws.greengrass#SubscribeToTopic"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "python3 -m pip install --user awsiotsdk",
        "run": "python3 -u {artifacts:path}/hello_world_subscriber.py"
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "py -3 -m pip install --user awsiotsdk",
        "run": "py -3 -u {artifacts:path}/hello_world_subscriber.py"
      }
    }
```

```
  ]
}
```

c. 使用文字編輯器建立以下列內容命名hello_world_subscriber.py的 Python 指令碼加工品。此應用程序使用發布/訂閱 IPC 服務來訂閱clients/+/hello/world主題並打印收到的消息。

例如，在 Linux 系統上，您可以執行下列命令來使用 GNU nano 來建立檔案。

```
nano artifacts/com.example.clientdevices.MyHelloWorldSubscriber/1.0.0/
hello_world_subscriber.py
```

將下面的 Python 代碼複製到文件中。

```
import sys
import time
import traceback

from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2

CLIENT_DEVICE_HELLO_WORLD_TOPIC = 'clients/+/hello/world'
TIMEOUT = 10


def on_hello_world_message(event):
    try:
        message = str(event.binary_message.message, 'utf-8')
        print('Received new message: %s' % message)
    except:
        traceback.print_exc()


try:
    ipc_client = GreengrassCoreIPCClientV2()

    # SubscribeToTopic returns a tuple with the response and the operation.
    _, operation = ipc_client.subscribe_to_topic(
        topic=CLIENT_DEVICE_HELLO_WORLD_TOPIC,
 on_stream_event=on_hello_world_message)
    print('Successfully subscribed to topic: %s' %
          CLIENT_DEVICE_HELLO_WORLD_TOPIC)
```

```
    # Keep the main thread alive, or the process will exit.
    try:
        while True:
            time.sleep(10)
    except InterruptedError:
        print('Subscribe interrupted.')

    operation.close()
except Exception:
    print('Exception occurred when using IPC.', file=sys.stderr)
    traceback.print_exc()
    exit(1)
```

> **Note**
>
> 這個組件使用的 IPC 客戶端 V2 中的 Python 與AWS IoT Greengrass核心軟件進行通信 AWS IoT Device SDK V 2。與原始 IPC 用戶端相比，IPC 用戶端 V2 可減少在自訂元件中使用 IPC 所需撰寫的程式碼量。

d. 您可 Greengrass 使用 CLI 來部署元件。

Linux or Unix

```
sudo /greengrass/v2/bin/greengrass-cli deployment create \
  --recipeDir recipes \
  --artifactDir artifacts \
  --merge "com.example.clientdevices.MyHelloWorldSubscriber=1.0.0"
```

Windows Command Prompt (CMD)

```
C:\greengrass\v2/bin/greengrass-cli deployment create ^
  --recipeDir recipes ^
  --artifactDir artifacts ^
  --merge "com.example.clientdevices.MyHelloWorldSubscriber=1.0.0"
```

PowerShell

```
C:\greengrass\v2/bin/greengrass-cli deployment create `
  --recipeDir recipes `
  --artifactDir artifacts `
```

```
  --merge "com.example.clientdevices.MyHelloWorldSubscriber=1.0.0"
```

3. 檢視元件記錄檔，以確認元件安裝成功並訂閱主題。

   Linux or Unix

   ```
   sudo tail -f /greengrass/v2/logs/
   com.example.clientdevices.MyHelloWorldSubscriber.log
   ```

   PowerShell

   ```
   gc C:\greengrass\v2/logs/com.example.clientdevices.MyHelloWorldSubscriber.log -
   Tail 10 -Wait
   ```

   您可以保持記錄摘要開啟，以確認核心裝置是否接收訊息。

4. 在用戶端裝置上，再次執行範例 Greengrass 探索應用程式，以將訊息傳送至核心裝置。

   ```
   python3 basic_discovery.py \\
     --thing_name MyClientDevice1 \\
     --topic 'clients/MyClientDevice1/hello/world' \\
     --message 'Hello World!' \\
     --ca_file ~/certs/AmazonRootCA1.pem \\
     --cert ~/certs/device.pem.crt \\
     --key ~/certs/private.pem.key \\
     --region us-east-1 \\
     --verbosity Warn
   ```

5. 再次檢視元件記錄檔，以確認元件是否從用戶端裝置接收和列印訊息。

   Linux or Unix

   ```
   sudo tail -f /greengrass/v2/logs/
   com.example.clientdevices.MyHelloWorldSubscriber.log
   ```

   PowerShell

   ```
   gc C:\greengrass\v2/logs/com.example.clientdevices.MyHelloWorldSubscriber.log -
   Tail 10 -Wait
   ```

步驟 5：開發與用戶端裝置陰影互動的元件

您可以開發與客戶端設備的AWS IoT設備陰影進行交互的 Greengrass 組件。陰影是儲存AWS IoT物件 (例如用戶端裝置) 目前或所需狀態資訊的 JSON 文件。自定義組件可以訪問客戶端設備的陰影來管理其狀態，即使客戶端設備未連接到AWS IoT也是如此。每個AWS IoT物件都有一個未命名的陰影，您也可以為每個物件創建多個命名陰影。

在本節中，您將部署陰影管理員元件以管理核心裝置上的陰影。您也會更新 MQTT 橋接器元件，以便在用戶端裝置和陰影管理員元件之間轉送陰影訊息。然後，您會開發更新用戶端裝置陰影的元件，並在用戶端裝置上執行範例應用程式，以回應來自元件的陰影更新。此元件代表智慧型照明管理應用程式，其中核心裝置可管理智慧型燈光的色彩狀態，這些智慧型燈具做為用戶端裝置連接。

開發與用戶端裝置陰影互動的元件

1. 修訂核心裝置的部署，以部署陰影管理員元件，並設定 MQTT 橋接器元件，在用戶端裝置與本機發佈/訂閱之間轉送陰影訊息 (陰影管理員進行通訊)。請執行下列操作：

   a. 導覽至 AWS IoT Greengrass主控台。

   b. 在左側導覽選單中，選擇 [核心裝置]。

   c. 在 [核心裝置] 頁面上，選擇您在本教學課程中使用的核心裝置。

   d. 在核心裝置詳細資料頁面上，選擇 [用戶端裝置] 索引標籤。

   e. 在用戶端裝置索引標籤上，選擇 [設定雲端探索]。

      [設定核心裝置探索] 頁面隨即開啟。在此頁面上，您可以變更或設定要部署到核心裝置的用戶端裝置元件。

   f. 在步驟 3 中，為零組aws.greengrass.clientdevices.mqtt.Bridge件選擇編輯模型組態。

   g. 在 MQTT 橋接器元件的編輯組態模式中，設定主題對應，以便在用戶端裝置與本機發佈/訂閱介面之間轉送裝置陰影主題的 MQTT 訊息。您也會確認部署指定了相容的 MQTT 橋接器版本。用戶端裝置陰影支援需要 MQTT 橋接器 v2.2.0 或更新版本。請執行下列操作：

      i. 對於「元件」版本，請選擇 2.2.0 或更新版本。

      ii. 在「組態」下，在「要合併程式碼的組態」區塊中，輸入下列設定。此配置指定轉送有關陰影主題的 MQTT 郵件。

```
{
  "mqttTopicMapping": {
    "HelloWorldIotCoreMapping": {
      "topic": "clients/+/hello/world",
```

```
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "HelloWorldPubsubMapping": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ShadowsLocalMqttToPubsub": {
      "topic": "$aws/things/+/shadow/#",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ShadowsPubsubToLocalMqtt": {
      "topic": "$aws/things/+/shadow/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    }
  }
}
```

如需詳細資訊，請參閱 MQTT 橋接器元件組態。

iii. 選擇確認。

h. 在步驟 3 中，選取要部署的aws.greengrass.ShadowManager元件。

i. 選擇複查並建置，以複查此頁面為您建立的部署。

j. 在「複查」頁面上，選擇「部署」以開始部署至核心裝置。

k. 若要驗證部署是否成功，請檢查部署狀態，然後檢查核心裝置上的記錄檔。若要檢查核心裝置上的部署狀態，您可以在部署概觀中選擇目標。如需詳細資訊，請參閱下列內容：

- 檢查部署狀態
- 監控AWS IoT Greengrass日誌

2. 開發和部署管理智慧型照明用戶端裝置的 Greengrass 元件。請執行下列操作：

a. 在核心裝置上建立元件成品的資料夾。

Linux or Unix

```
mkdir -p artifacts/com.example.clientdevices.MySmartLightManager/1.0.0
```

Windows Command Prompt (CMD)

```
mkdir artifacts\com.example.clientdevices.MySmartLightManager\1.0.0
```

PowerShell

```
mkdir artifacts\com.example.clientdevices.MySmartLightManager\1.0.0
```

> **Important**
>
> 您必須針對人工因素資料夾路徑使用下列格式。包括您在方案中指定的元件名稱和版本。
>
> ```
> artifacts/componentName/componentVersion/
> ```

b. 使用文字編輯器建立包含下列內容的元件配方。此配方指定為 Python 安裝 AWS IoT Device SDK v2，並執行與智慧燈用戶端裝置的陰影互動的指令碼，以管理其顏色。

例如，在 Linux 系統上，您可以執行下列命令來使用 GNU nano 來建立檔案。

```
nano recipes/com.example.clientdevices.MySmartLightManager-1.0.0.json
```

將下列配方複製到檔案中。

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.clientdevices.MySmartLightManager",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that interacts with smart light client
 devices.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.Nucleus": {
      "VersionRequirement": "^2.6.0"
    },
    "aws.greengrass.ShadowManager": {
      "VersionRequirement": "^2.2.0"
    },
```

```
    "aws.greengrass.clientdevices.mqtt.Bridge": {
      "VersionRequirement": "^2.2.0"
    }
  },
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "smartLightDeviceNames": [],
      "accessControl": {
        "aws.greengrass.ShadowManager": {
          "com.example.clientdevices.MySmartLightManager:shadow:1": {
            "policyDescription": "Allows access to client devices' unnamed
shadows",
            "operations": [
              "aws.greengrass#GetThingShadow",
              "aws.greengrass#UpdateThingShadow"
            ],
            "resources": [
              "$aws/things/MyClientDevice*/shadow"
            ]
          }
        },
        "aws.greengrass.ipc.pubsub": {
          "com.example.clientdevices.MySmartLightManager:pubsub:1": {
            "policyDescription": "Allows access to client devices' unnamed
shadow updates",
            "operations": [
              "aws.greengrass#SubscribeToTopic"
            ],
            "resources": [
              "$aws/things/+/shadow/update/accepted"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "python3 -m pip install --user awsiotsdk",
        "run": "python3 -u {artifacts:path}/smart_light_manager.py"
```

```
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "py -3 -m pip install --user awsiotsdk",
        "run": "py -3 -u {artifacts:path}/smart_light_manager.py"
      }
    }
  ]
}
```

c. 使用文字編輯器建立以下列內容命名`smart_light_manager.py`的 Python 指令碼加工品。此應用程式使用 shadow IPC 服務來取得和更新用戶端裝置陰影，以及本機發佈/訂閱 IPC 服務，以接收報告的陰影更新。

例如，在 Linux 系統上，您可以執行下列命令來使用 GNU nano 來建立檔案。

```
nano artifacts/com.example.clientdevices.MySmartLightManager/1.0.0/
smart_light_manager.py
```

將下面的 Python 代碼複製到文件中。

```
import json
import random
import sys
import time
import traceback
from uuid import uuid4

from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2
from awsiot.greengrasscoreipc.model import ResourceNotFoundError

SHADOW_COLOR_PROPERTY = 'color'
CONFIGURATION_CLIENT_DEVICE_NAMES = 'smartLightDeviceNames'
COLORS = ['red', 'orange', 'yellow', 'green', 'blue', 'purple']
SHADOW_UPDATE_TOPIC = '$aws/things/+/shadow/update/accepted'
SET_COLOR_INTERVAL = 15
```

```
class SmartLightDevice():
    def __init__(self, client_device_name: str, reported_color: str = None):
        self.name = client_device_name
        self.reported_color = reported_color
        self.desired_color = None


class SmartLightDeviceManager():
    def __init__(self, ipc_client: GreengrassCoreIPCClientV2):
        self.ipc_client = ipc_client
        self.devices = {}
        self.client_tokens = set()
        self.shadow_update_accepted_subscription_operation = None
        self.client_device_names_configuration_subscription_operation = None
        self.update_smart_light_device_list()

    def update_smart_light_device_list(self):
        # Update the device list from the component configuration.
        response = self.ipc_client.get_configuration(
            key_path=[CONFIGURATION_CLIENT_DEVICE_NAMES])
        # Identify the difference between the configuration and the currently
 tracked devices.
        current_device_names = self.devices.keys()
        updated_device_names =
 response.value[CONFIGURATION_CLIENT_DEVICE_NAMES]
        added_device_names = set(updated_device_names) -
 set(current_device_names)
        removed_device_names = set(current_device_names) -
 set(updated_device_names)
        # Stop tracking any smart light devices that are no longer in the
 configuration.
        for name in removed_device_names:
            print('Removing %s from smart light device manager' % name)
            self.devices.pop(name)
        # Start tracking any new smart light devices that are in the
 configuration.
        for name in added_device_names:
            print('Adding %s to smart light device manager' % name)
            device = SmartLightDevice(name)
            device.reported_color = self.get_device_reported_color(device)
            self.devices[name] = device
            print('Current color for %s is %s' % (name,
 device.reported_color))
```

```
    def get_device_reported_color(self, smart_light_device):
        try:
            response = self.ipc_client.get_thing_shadow(
                thing_name=smart_light_device.name, shadow_name='')
            shadow = json.loads(str(response.payload, 'utf-8'))
            if 'reported' in shadow['state']:
                return shadow['state']['reported'].get(SHADOW_COLOR_PROPERTY)
            return None
        except ResourceNotFoundError:
            return None

    def request_device_color_change(self, smart_light_device, color):
        # Generate and track a client token for the request.
        client_token = str(uuid4())
        self.client_tokens.add(client_token)
        # Create a shadow payload, which must be a blob.
        payload_json = {
            'state': {
                'desired': {
                    SHADOW_COLOR_PROPERTY: color
                }
            },
            'clientToken': client_token
        }
        payload = bytes(json.dumps(payload_json), 'utf-8')
        self.ipc_client.update_thing_shadow(
            thing_name=smart_light_device.name, shadow_name='',
payload=payload)
        smart_light_device.desired_color = color

    def subscribe_to_shadow_update_accepted_events(self):
        if self.shadow_update_accepted_subscription_operation == None:
            # SubscribeToTopic returns a tuple with the response and the
operation.
            _, self.shadow_update_accepted_subscription_operation =
self.ipc_client.subscribe_to_topic(
                topic=SHADOW_UPDATE_TOPIC,
on_stream_event=self.on_shadow_update_accepted_event)
            print('Successfully subscribed to shadow update accepted topic')

    def close_shadow_update_accepted_subscription(self):
        if self.shadow_update_accepted_subscription_operation is not None:
            self.shadow_update_accepted_subscription_operation.close()
```

```
    def on_shadow_update_accepted_event(self, event):
        try:
            message = str(event.binary_message.message, 'utf-8')
            accepted_payload = json.loads(message)
            # Check for reported states from smart light devices and ignore
desired states from components.
            if 'reported' in accepted_payload['state']:
                # Process this update only if it uses a client token created by
this component.
                client_token = accepted_payload.get('clientToken')
                if client_token is not None and client_token in
self.client_tokens:
                    self.client_tokens.remove(client_token)
                    shadow_state = accepted_payload['state']['reported']
                    if SHADOW_COLOR_PROPERTY in shadow_state:
                        reported_color = shadow_state[SHADOW_COLOR_PROPERTY]
                        topic = event.binary_message.context.topic
                        client_device_name = topic.split('/')[2]
                        if client_device_name in self.devices:
                            # Set the reported color for the smart light
device.
                            self.devices[client_device_name].reported_color =
reported_color
                            print(
                                'Received shadow update confirmation from
client device: %s' % client_device_name)
                    else:
                        print("Shadow update doesn't specify color")
        except:
            traceback.print_exc()

    def subscribe_to_client_device_name_configuration_updates(self):
        if self.client_device_names_configuration_subscription_operation ==
None:
            # SubscribeToConfigurationUpdate returns a tuple with the response
and the operation.
            _, self.client_device_names_configuration_subscription_operation =
self.ipc_client.subscribe_to_configuration_update(
                key_path=[CONFIGURATION_CLIENT_DEVICE_NAMES],
on_stream_event=self.on_client_device_names_configuration_update_event)
            print(
                'Successfully subscribed to configuration updates for smart
light device names')
```

```
    def close_client_device_names_configuration_subscription(self):
        if self.client_device_names_configuration_subscription_operation is not
 None:

 self.client_device_names_configuration_subscription_operation.close()

    def on_client_device_names_configuration_update_event(self, event):
        try:
            if CONFIGURATION_CLIENT_DEVICE_NAMES in
 event.configuration_update_event.key_path:
                print('Received configuration update for list of client
 devices')
                self.update_smart_light_device_list()
        except:
            traceback.print_exc()


def choose_random_color():
    return random.choice(COLORS)

def main():
    try:
        # Create an IPC client and a smart light device manager.
        ipc_client = GreengrassCoreIPCClientV2()
        smart_light_manager = SmartLightDeviceManager(ipc_client)
        smart_light_manager.subscribe_to_shadow_update_accepted_events()

 smart_light_manager.subscribe_to_client_device_name_configuration_updates()
        try:
            # Keep the main thread alive, or the process will exit.
            while True:
                # Set each smart light device to a random color at a regular
 interval.
                for device_name in smart_light_manager.devices:
                    device = smart_light_manager.devices[device_name]
                    desired_color = choose_random_color()
                    print('Chose random color (%s) for %s' %
                          (desired_color, device_name))
                    if desired_color == device.desired_color:
                        print('Desired color for %s is already %s' %
                              (device_name, desired_color))
                    elif desired_color == device.reported_color:
                        print('Reported color for %s is already %s' %
                              (device_name, desired_color))
```

```
                    else:
                        smart_light_manager.request_device_color_change(
                            device, desired_color)
                        print('Requested color change for %s to %s' %
                              (device_name, desired_color))
                time.sleep(SET_COLOR_INTERVAL)
        except InterruptedError:
            print('Application interrupted')
        smart_light_manager.close_shadow_update_accepted_subscription()

 smart_light_manager.close_client_device_names_configuration_subscription()
    except Exception:
        print('Exception occurred', file=sys.stderr)
        traceback.print_exc()
        exit(1)


if __name__ == '__main__':
    main()
```

這個 Python 應用程序執行以下操作：

- 讀取元件的組態，以取得要管理的智慧型照明用戶端裝置清單。
- 使用 SubscribeToConfigurationUpdate IPC 作業訂閱組態更新通知。每次元件的組態變更時，AWS IoT GreengrassCore 軟體都會傳送通知。當元件收到組態設定更新通知時，會更新其管理的智慧型照明用戶端裝置清單。
- 獲取每個智能燈客戶端設備的陰影以獲取其初始顏色狀態。
- 每 15 秒將每個智慧燈用戶端裝置的顏色設定為隨機顏色。元件會更新用戶端裝置的物件陰影，以變更其顏色。此作業會透過 MQTT 傳送陰影增量事件至用戶端裝置。
- 使用 IPC 作業，在本機發佈/訂閱介面上訂閱陰影更新接受的SubscribeToTopic訊息。此元件會接收這些訊息，以追蹤每個智慧型照明用戶端裝置的色彩。當智慧型照明用戶端裝置收到陰影更新時，會傳送 MQTT 訊息以確認其已收到更新。MQTT 橋接器會將此訊息轉送至本機發佈/訂閱介面。

d. 您可 Greengrass 使用 CLI 來部署元件。部署此元件時，您可以指定用戶端裝置的清單`smartLightDeviceNames`，以及其管理陰影的用戶端裝置。將 *`MyClientDevice1`* 取代為用戶端裝置的物件名稱。

Linux or Unix

```
sudo /greengrass/v2/bin/greengrass-cli deployment create \
  --recipeDir recipes \
  --artifactDir artifacts \
  --merge "com.example.clientdevices.MySmartLightManager=1.0.0" \
  --update-config '{
    "com.example.clientdevices.MySmartLightManager": {
      "MERGE": {
        "smartLightDeviceNames": [
          "MyClientDevice1"
        ]
      }
    }
  }'
```

Windows Command Prompt (CMD)

```
C:\greengrass\v2/bin/greengrass-cli deployment create ^
  --recipeDir recipes ^
  --artifactDir artifacts ^
  --merge "com.example.clientdevices.MySmartLightManager=1.0.0" ^
  --update-config '{"com.example.clientdevices.MySmartLightManager":
{"MERGE":{"smartLightDeviceNames":["MyClientDevice1"]}}}'
```

PowerShell

```
C:\greengrass\v2/bin/greengrass-cli deployment create `
  --recipeDir recipes `
  --artifactDir artifacts `
  --merge "com.example.clientdevices.MySmartLightManager=1.0.0" `
  --update-config '{
    "com.example.clientdevices.MySmartLightManager": {
      "MERGE": {
        "smartLightDeviceNames": [
          "MyClientDevice1"
        ]
      }
    }
  }'
```

3. 檢視元件記錄檔，以確認元件是否已順利安裝並執行。

   Linux or Unix

   ```
   sudo tail -f /greengrass/v2/logs/
   com.example.clientdevices.MySmartLightManager.log
   ```

   PowerShell

   ```
   gc C:\greengrass\v2/logs/com.example.clientdevices.MySmartLightManager.log -Tail
    10 -Wait
   ```

   元件會傳送變更智慧型照明用戶端裝置顏色的請求。陰影管理員會接收要求並設定陰影的`desired`狀態。但是，智能燈客戶端設備尚未運行，因此陰影的`reported`狀態不會改變。元件的記錄檔包含下列訊息。

   ```
   2022-07-07T03:49:24.908Z [INFO] (Copier)
    com.example.clientdevices.MySmartLightManager: stdout. Chose random color (blue)
    for MyClientDevice1.
    {scriptName=services.com.example.clientdevices.MySmartLightManager.lifecycle.Run,
    serviceName=com.example.clientdevices.MySmartLightManager, currentState=RUNNING}
   2022-07-07T03:49:24.912Z [INFO] (Copier)
    com.example.clientdevices.MySmartLightManager: stdout.
    Requested color change for MyClientDevice1 to blue.
    {scriptName=services.com.example.clientdevices.MySmartLightManager.lifecycle.Run,
    serviceName=com.example.clientdevices.MySmartLightManager, currentState=RUNNING}
   ```

   您可以保持記錄摘要開啟，以查看元件何時列印訊息。

4. 下載並執行使用 Greengrass 探索並訂閱裝置陰影更新的範例應用程式。在用戶端裝置上，執行下列動作：

   a. 切換到示例文件夾在 AWS IoT Device SDK v2 中的 Python。此範例應用程式使用 sample 資料夾中的命令列剖析模組。

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

   b. 使用文字編輯器建立以下列內容命名`basic_discovery_shadow.py`的 Python 指令碼。此應用程序使用 Greengrass 發現和陰影來保持客戶端設備和核心設備之間的屬性同步。

例如，在 Linux 系統上，您可以執行下列命令來使用 GNU nano 來建立檔案。

```
nano basic_discovery_shadow.py
```

將下面的 Python 代碼複製到文件中。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0.

from awscrt import io
from awscrt import mqtt
from awsiot import iotshadow
from awsiot.greengrass_discovery import DiscoveryClient
from awsiot import mqtt_connection_builder
from concurrent.futures import Future
import sys
import threading
import traceback
from uuid import uuid4

# Parse arguments
import utils.command_line_utils;
cmdUtils = utils.command_line_utils.CommandLineUtils("Basic Discovery -
 Greengrass discovery example with device shadows.")
cmdUtils.add_common_mqtt_commands()
cmdUtils.add_common_topic_message_commands()
cmdUtils.add_common_logging_commands()
cmdUtils.register_command("key", "<path>", "Path to your key in PEM format.",
 True, str)
cmdUtils.register_command("cert", "<path>", "Path to your client certificate in
 PEM format.", True, str)
cmdUtils.remove_command("endpoint")
cmdUtils.register_command("thing_name", "<str>", "The name assigned to your IoT
 Thing", required=True)
cmdUtils.register_command("region", "<str>", "The region to connect through.",
 required=True)
cmdUtils.register_command("shadow_property", "<str>", "The name of the shadow
 property you want to change (optional, default='color'", default="color")
# Needs to be called so the command utils parse the commands
cmdUtils.get_args()

# Using globals to simplify sample code
```

```
is_sample_done = threading.Event()
mqtt_connection = None
shadow_thing_name = cmdUtils.get_command_required("thing_name")
shadow_property = cmdUtils.get_command("shadow_property")

SHADOW_VALUE_DEFAULT = "off"

class LockedData:
    def __init__(self):
        self.lock = threading.Lock()
        self.shadow_value = None
        self.disconnect_called = False
        self.request_tokens = set()

locked_data = LockedData()


def on_connection_interupted(connection, error, **kwargs):
    print('connection interrupted with error {}'.format(error))


def on_connection_resumed(connection, return_code, session_present, **kwargs):
    print('connection resumed with return code {}, session present
 {}'.format(return_code, session_present))

# Try IoT endpoints until we find one that works
def try_iot_endpoints():
    for gg_group in discover_response.gg_groups:
        for gg_core in gg_group.cores:
            for connectivity_info in gg_core.connectivity:
                try:
                    print('Trying core {} at host {} port
 {}'.format(gg_core.thing_arn, connectivity_info.host_address,
 connectivity_info.port))
                    mqtt_connection = mqtt_connection_builder.mtls_from_path(
                        endpoint=connectivity_info.host_address,
                        port=connectivity_info.port,
                        cert_filepath=cmdUtils.get_command_required("cert"),
                        pri_key_filepath=cmdUtils.get_command_required("key"),

 ca_bytes=gg_group.certificate_authorities[0].encode('utf-8'),
                        on_connection_interrupted=on_connection_interupted,
                        on_connection_resumed=on_connection_resumed,
                        client_id=cmdUtils.get_command_required("thing_name"),
```

```
                        clean_session=False,
                        keep_alive_secs=30)

                    connect_future = mqtt_connection.connect()
                    connect_future.result()
                    print('Connected!')
                    return mqtt_connection

                except Exception as e:
                    print('Connection failed with exception {}'.format(e))
                    continue

    exit('All connection attempts failed')

# Function for gracefully quitting this sample
def exit(msg_or_exception):
    if isinstance(msg_or_exception, Exception):
        print("Exiting sample due to exception.")
        traceback.print_exception(msg_or_exception.__class__, msg_or_exception,
 sys.exc_info()[2])
    else:
        print("Exiting sample:", msg_or_exception)

    with locked_data.lock:
        if not locked_data.disconnect_called:
            print("Disconnecting...")
            locked_data.disconnect_called = True
            future = mqtt_connection.disconnect()
            future.add_done_callback(on_disconnected)

def on_disconnected(disconnect_future):
    # type: (Future) -> None
    print("Disconnected.")

    # Signal that sample is finished
    is_sample_done.set()

def on_get_shadow_accepted(response):
    # type: (iotshadow.GetShadowResponse) -> None
    try:
        with locked_data.lock:
            # check that this is a response to a request from this session
            try:
                locked_data.request_tokens.remove(response.client_token)
```

```
            except KeyError:
                return

            print("Finished getting initial shadow state.")
            if locked_data.shadow_value is not None:
                print("  Ignoring initial query because a delta event has
 already been received.")
                return

        if response.state:
            if response.state.delta:
                value = response.state.delta.get(shadow_property)
                if value:
                    print("  Shadow contains delta value '{}'.".format(value))
                    change_shadow_value(value)
                    return

            if response.state.reported:
                value = response.state.reported.get(shadow_property)
                if value:
                    print("  Shadow contains reported value
 '{}'.".format(value))

 set_local_value_due_to_initial_query(response.state.reported[shadow_property])
                    return

        print("  Shadow document lacks '{}' property. Setting
 defaults...".format(shadow_property))
        change_shadow_value(SHADOW_VALUE_DEFAULT)
        return

    except Exception as e:
        exit(e)

def on_get_shadow_rejected(error):
    # type: (iotshadow.ErrorResponse) -> None
    try:
        # check that this is a response to a request from this session
        with locked_data.lock:
            try:
                locked_data.request_tokens.remove(error.client_token)
            except KeyError:
                return
```

```
        if error.code == 404:
            print("Thing has no shadow document. Creating with defaults...")
            change_shadow_value(SHADOW_VALUE_DEFAULT)
        else:
            exit("Get request was rejected. code:{} message:'{}'".format(
                error.code, error.message))

    except Exception as e:
        exit(e)

def on_shadow_delta_updated(delta):
    # type: (iotshadow.ShadowDeltaUpdatedEvent) -> None
    try:
        print("Received shadow delta event.")
        if delta.state and (shadow_property in delta.state):
            value = delta.state[shadow_property]
            if value is None:
                print("  Delta reports that '{}' was deleted. Resetting
 defaults...".format(shadow_property))
                change_shadow_value(SHADOW_VALUE_DEFAULT)
                return
            else:
                print("  Delta reports that desired value is '{}'. Changing
 local value...".format(value))
                if (delta.client_token is not None):
                    print ("  ClientToken is: " + delta.client_token)
                change_shadow_value(value, delta.client_token)
        else:
            print("  Delta did not report a change in
 '{}'".format(shadow_property))

    except Exception as e:
        exit(e)

def on_publish_update_shadow(future):
    #type: (Future) -> None
    try:
        future.result()
        print("Update request published.")
    except Exception as e:
        print("Failed to publish update request.")
        exit(e)

def on_update_shadow_accepted(response):
```

```
    # type: (iotshadow.UpdateShadowResponse) -> None
    try:
        # check that this is a response to a request from this session
        with locked_data.lock:
            try:
                locked_data.request_tokens.remove(response.client_token)
            except KeyError:
                return

        try:
            if response.state.reported != None:
                if shadow_property in response.state.reported:
                    print("Finished updating reported shadow value to
 '{}'.".format(response.state.reported[shadow_property])) # type: ignore
                else:
                    print ("Could not find shadow property with name:
 '{}'.".format(shadow_property)) # type: ignore
            else:
                print("Shadow states cleared.") # when the shadow states are
 cleared, reported and desired are set to None
        except:
            exit("Updated shadow is missing the target property")

    except Exception as e:
        exit(e)

def on_update_shadow_rejected(error):
    # type: (iotshadow.ErrorResponse) -> None
    try:
        # check that this is a response to a request from this session
        with locked_data.lock:
            try:
                locked_data.request_tokens.remove(error.client_token)
            except KeyError:
                return

        exit("Update request was rejected. code:{} message:'{}'".format(
            error.code, error.message))

    except Exception as e:
        exit(e)

def set_local_value_due_to_initial_query(reported_value):
    with locked_data.lock:
```

```
        locked_data.shadow_value = reported_value

def change_shadow_value(value, token=None):
    with locked_data.lock:
        if locked_data.shadow_value == value:
            print("Local value is already '{}'.".format(value))
            return

        print("Changed local shadow value to '{}'.".format(value))
        locked_data.shadow_value = value

        print("Updating reported shadow value to '{}'...".format(value))

        reuse_token = token is not None
        # use a unique token so we can correlate this "request" message to
        # any "response" messages received on the /accepted and /rejected
 topics
        if not reuse_token:
            token = str(uuid4())

        # if the value is "clear shadow" then send a UpdateShadowRequest with
 None
        # for both reported and desired to clear the shadow document
 completely.
        if value == "clear_shadow":
            tmp_state = iotshadow.ShadowState(reported=None, desired=None,
 reported_is_nullable=True, desired_is_nullable=True)
            request = iotshadow.UpdateShadowRequest(
                thing_name=shadow_thing_name,
                state=tmp_state,
                client_token=token,
            )
        # Otherwise, send a normal update request
        else:
            # if the value is "none" then set it to a Python none object to
            # clear the individual shadow property
            if value == "none":
                value = None

            request = iotshadow.UpdateShadowRequest(
            thing_name=shadow_thing_name,
            state=iotshadow.ShadowState(
                reported={ shadow_property: value }
                ),
```

```
                client_token=token,
            )

        future = shadow_client.publish_update_shadow(request,
 mqtt.QoS.AT_LEAST_ONCE)

        if not reuse_token:
            locked_data.request_tokens.add(token)

        future.add_done_callback(on_publish_update_shadow)


if __name__ == '__main__':
    tls_options =
 io.TlsContextOptions.create_client_with_mtls_from_path(cmdUtils.get_command_required("
 cmdUtils.get_command_required("key"))
    if cmdUtils.get_command(cmdUtils.m_cmd_ca_file):
        tls_options.override_default_trust_store_from_path(None,
 cmdUtils.get_command(cmdUtils.m_cmd_ca_file))
    tls_context = io.ClientTlsContext(tls_options)

    socket_options = io.SocketOptions()

    print('Performing greengrass discovery...')
    discovery_client =
 DiscoveryClient(io.ClientBootstrap.get_or_create_static_default(),
 socket_options, tls_context, cmdUtils.get_command_required("region"))
    resp_future =
 discovery_client.discover(cmdUtils.get_command_required("thing_name"))
    discover_response = resp_future.result()

    print(discover_response)
    if cmdUtils.get_command("print_discover_resp_only"):
        exit(0)

    mqtt_connection = try_iot_endpoints()
    shadow_client = iotshadow.IotShadowClient(mqtt_connection)

    try:
        # Subscribe to necessary topics.
        # Note that is **is** important to wait for "accepted/rejected"
 subscriptions
        # to succeed before publishing the corresponding "request".
        print("Subscribing to Update responses...")
```

```
        update_accepted_subscribed_future, _ =
shadow_client.subscribe_to_update_shadow_accepted(

request=iotshadow.UpdateShadowSubscriptionRequest(thing_name=shadow_thing_name),
            qos=mqtt.QoS.AT_LEAST_ONCE,
            callback=on_update_shadow_accepted)

        update_rejected_subscribed_future, _ =
shadow_client.subscribe_to_update_shadow_rejected(

request=iotshadow.UpdateShadowSubscriptionRequest(thing_name=shadow_thing_name),
            qos=mqtt.QoS.AT_LEAST_ONCE,
            callback=on_update_shadow_rejected)

        # Wait for subscriptions to succeed
        update_accepted_subscribed_future.result()
        update_rejected_subscribed_future.result()

        print("Subscribing to Get responses...")
        get_accepted_subscribed_future, _ =
shadow_client.subscribe_to_get_shadow_accepted(

request=iotshadow.GetShadowSubscriptionRequest(thing_name=shadow_thing_name),
            qos=mqtt.QoS.AT_LEAST_ONCE,
            callback=on_get_shadow_accepted)

        get_rejected_subscribed_future, _ =
shadow_client.subscribe_to_get_shadow_rejected(

request=iotshadow.GetShadowSubscriptionRequest(thing_name=shadow_thing_name),
            qos=mqtt.QoS.AT_LEAST_ONCE,
            callback=on_get_shadow_rejected)

        # Wait for subscriptions to succeed
        get_accepted_subscribed_future.result()
        get_rejected_subscribed_future.result()

        print("Subscribing to Delta events...")
        delta_subscribed_future, _ =
shadow_client.subscribe_to_shadow_delta_updated_events(

request=iotshadow.ShadowDeltaUpdatedSubscriptionRequest(thing_name=shadow_thing_name),
            qos=mqtt.QoS.AT_LEAST_ONCE,
            callback=on_shadow_delta_updated)
```

```
        # Wait for subscription to succeed
        delta_subscribed_future.result()

        # The rest of the sample runs asynchronously.

        # Issue request for shadow's current state.
        # The response will be received by the on_get_accepted() callback
        print("Requesting current shadow state...")

        with locked_data.lock:
            # use a unique token so we can correlate this "request" message to
            # any "response" messages received on the /accepted and /rejected
topics
            token = str(uuid4())

            publish_get_future = shadow_client.publish_get_shadow(

request=iotshadow.GetShadowRequest(thing_name=shadow_thing_name,
client_token=token),
                qos=mqtt.QoS.AT_LEAST_ONCE)

            locked_data.request_tokens.add(token)

        # Ensure that publish succeeds
        publish_get_future.result()

    except Exception as e:
        exit(e)

    # Wait for the sample to finish (user types 'quit', or an error occurs)
    is_sample_done.wait()
```

這個 Python 應用程序執行以下操作：

- 使用 Greengrass 發現來發現並連接到核心設備。
- 從核心裝置要求陰影文件以取得屬性的初始狀態。
- 訂閱陰影增量事件，當內容的值與其`desired``reported`值不同時，核心裝置會傳送此事件。當應用程式收到 shadow delta 事件時，會變更內容的值，並將更新傳送至核心裝置，以將新值設定為其`reported`值。

此應用程式結合了來自 v2 的 Greengrass 探索和陰影樣本。AWS IoT Device SDK

c. 執行範例應用程式。此應用程式需要指定用戶端裝置物件名稱、要使用的 shadow 屬性，以及驗證和保護連線的憑證的引數。

- 將 *MyClientDevice1* 取代為用戶端裝置的物件名稱。
- 將 *~/##/ AmazonRoot CA1.pem* 取代為用戶端裝置上 Amazon 根 CA 憑證的路徑。
- 將*#/##/## .pem.crt* 替換為客戶端設備上的設備證書的路徑。
- 將*#/##/## .pem.key ###########*件的路徑。
- 將 *us-east-1* 取代為用戶端裝置和核心裝置運作所在的AWS區域。

```
python3 basic_discovery_shadow.py \
  --thing_name MyClientDevice1 \
  --shadow_property color \
  --ca_file ~/certs/AmazonRootCA1.pem \
  --cert ~/certs/device.pem.crt \
  --key ~/certs/private.pem.key \
  --region us-east-1 \
  --verbosity Warn
```

範例應用程式會訂閱陰影主題，並等待從核心裝置接收陰影增量事件。如果輸出指示應用程式接收並回應陰影增量事件，則用戶端裝置可以成功地與核心裝置上的陰影互動。

```
Performing greengrass discovery...
awsiot.greengrass_discovery.DiscoverResponse(gg_groups=[awsiot.greengrass_discovery.GGG
coreDevice-MyGreengrassCore',
 cores=[awsiot.greengrass_discovery.GGCore(thing_arn='arn:aws:iot:us-
east-1:123456789012:thing/MyGreengrassCore',
 connectivity=[awsiot.greengrass_discovery.ConnectivityInfo(id='203.0.113.0',
 host_address='203.0.113.0', metadata='', port=8883)])],
 certificate_authorities=['-----BEGIN CERTIFICATE-----
\nMIICiT...EXAMPLE=\n-----END CERTIFICATE-----\n'])])
Trying core arn:aws:iot:us-east-1:123456789012:thing/MyGreengrassCore at host
 203.0.113.0 port 8883
Connected!
Subscribing to Update responses...
Subscribing to Get responses...
Subscribing to Delta events...
```

```
Requesting current shadow state...
Received shadow delta event.
  Delta reports that desired value is 'purple'. Changing local value...
  ClientToken is: 3dce4d3f-e336-41ac-aa4f-7882725f0033
Changed local shadow value to 'purple'.
Updating reported shadow value to 'purple'...
Update request published.
```

如果應用程式輸出錯誤，請參閱 Greengrass 探索問題疑難排解。

您也可以在核心裝置上檢視 Greengrass 記錄檔，以驗證用戶端裝置是否成功連線並傳送訊息。如需詳細資訊，請參閱 監控AWS IoT Greengrass日誌。

5. 再次檢視元件記錄檔，以確認元件是否接收來自 Smart Light 用戶端裝置的陰影更新確認。

Linux or Unix

```
sudo tail -f /greengrass/v2/logs/
com.example.clientdevices.MySmartLightManager.log
```

PowerShell

```
gc C:\greengrass\v2/logs/com.example.clientdevices.MySmartLightManager.log -Tail
 10 -Wait
```

元件會記錄訊息，以確認 Smart Light 用戶端裝置已變更其顏色。

```
2022-07-07T03:49:24.908Z [INFO] (Copier)
 com.example.clientdevices.MySmartLightManager: stdout. Chose random color (blue)
 for MyClientDevice1.
 {scriptName=services.com.example.clientdevices.MySmartLightManager.lifecycle.Run,
 serviceName=com.example.clientdevices.MySmartLightManager, currentState=RUNNING}
2022-07-07T03:49:24.912Z [INFO] (Copier)
 com.example.clientdevices.MySmartLightManager: stdout.
 Requested color change for MyClientDevice1 to blue.
 {scriptName=services.com.example.clientdevices.MySmartLightManager.lifecycle.Run,
 serviceName=com.example.clientdevices.MySmartLightManager, currentState=RUNNING}
2022-07-07T03:49:24.959Z [INFO] (Copier)
 com.example.clientdevices.MySmartLightManager: stdout. Received
 shadow update confirmation from client device: MyClientDevice1.
```

```
{scriptName=services.com.example.clientdevices.MySmartLightManager.lifecycle.Run,
serviceName=com.example.clientdevices.MySmartLightManager, currentState=RUNNING}
```

> **Note**
>
> 用戶端裝置的陰影會在核心裝置與用戶端裝置之間同步。不過，核心裝置不會與用戶端裝置的陰影同步處理AWS IoT Core。例如，您可以將陰影同步處理，AWS IoT Core以檢視或修改叢集中所有裝置的狀態。如需如何設定陰影管理員元件與陰影同步的詳細資訊AWS IoT Core，請參閱同步本地設備陰影 AWS IoT Core。

您已完成此教學課程。用戶端裝置會連線至核心裝置、將 MQTT 訊息傳送至 Greengrass 元件，AWS IoT Core並接收來自核心裝置的陰影更新。如需本自學課程所涵蓋主題的詳細資訊，請參閱下列內容：

- 關聯用戶端裝置
- 管理核心裝置端點
- 測試用戶端裝置通訊
- Greengrass 發現寧靜 API
- 在用戶端裝置之間轉送 MQTT 訊息，AWS IoT Core
- 與元件中的用戶端裝置互動
- 與裝置陰影互動
- 與用戶端裝置陰影互動並同步

教學課程：開始使用 SageMaker 邊緣管理員

> **Important**
>
> SageMaker 邊緣管理員將於 2024 年 4 月 26 日停止使用。如需有關繼續將模型部署到邊緣裝置的詳細資訊，請參閱SageMaker 邊緣管理員生命週期結束。

Amazon SageMaker 邊緣管理器是在邊緣裝置上執行的軟體代理程式。 SageMaker Edge Manager 提供邊緣裝置的模型管理功能，讓您可以直接在 Greengrass 核心裝置上封裝和使用 Amazon SageMaker 新編譯模型。透過使用 SageMaker Edge Manager，您也可以從核心裝置取樣模型輸入和輸出資料，並將該資料傳送至以進AWS 雲端行監視和分析。如需 SageMaker 邊緣管理員如何在

Greengrass 核心裝置上運作的詳細資訊，請參閱。在核心設備上使用 Amazon SageMaker 邊緣管理器

本教學課程說明如何開始使用 SageMaker Edge Manager，搭配現有核心裝置上AWS提供的範例元件。這些範例元件會使用 SageMaker Edge Manager 元件做為相依性來部署 Edge Manager 代理程式，並使用使用 Neo 編譯的預先訓練模型執行推論。 SageMaker 如需 SageMaker 邊緣管理員代理程式的詳細資訊，請參閱 Amazon SageMaker 開發人員指南中的SageMaker 邊緣管理員。

若要在現有 Greengrass 核心裝置上設定和使用 SageMaker Edge Manager 代理程式，請AWS提供範例程式碼，讓您用來建立下列範例推論和模型元件。

- 影像分類
  - `com.greengrass.SageMakerEdgeManager.ImageClassification`
  - `com.greengrass.SageMakerEdgeManager.ImageClassification.Model`
- 物體偵測
  - `com.greengrass.SageMakerEdgeManager.ObjectDetection`
  - `com.greengrass.SageMakerEdgeManager.ObjectDetection.Model`

本教學課程說明如何部署範例元件和 SageMaker Edge 管理員代理程式。

主題

必要條件

若要完成此自學課程，您必須符合下列先決條件：

- 在 Amazon Linux 2，基於 Debian 的 Linux 平台（x86_64 或 Armv8）或視窗（x86_64）上運行的 Greengrass 核心設備。如果您沒有帳戶，請參閱 教學課程：AWS IoT Greengrass V2 入門。
- Python 3.6 或更高版本，包括您`pip`的 Python 版本，安裝在您的核心設備上。
- OpenGL API GLX 運行時（`libgl1-mesa-glx`）安裝在您的核心設備上。
- 具有管理員權限的 AWS Identity and Access Management (IAM) 使用者。

- 具備網際網路功能的 Windows、Mac 或類 Unix 開發電腦，符合下列需求：
  - 安 Python 了 3.6 或更高版本。
  - AWS CLI使用 IAM 管理員使用者登入資料進行安裝和設定。如需詳細資訊，請參閱安裝AWS CLI和配置 AWS CLI.
- 以下 S3 存儲桶在同一個AWS 帳戶和AWS 區域您的 Greengrass 核心設備中創建：
  - 用於存放範例推論和模型元件中包含的成品的 S3 儲存貯體。本教學課程使用 *DOC-EXAMPLE-BUCKET1* 來參考此值區。
  - 與 SageMaker 邊緣裝置叢集關聯的 S3 儲存貯體。 SageMaker Edge Manager 需要 S3 儲存貯體來建立邊緣裝置叢集，並存放在裝置上執行推論時所產生的範例資料。本教學課程使用 *DOC-EXAMPLE-BUCKET2* 來參考此值區。

  如需建立 S3 儲存貯體的相關資訊，請參閱開始使用 Amazon S3。
- 設定了以下內容的 Greengrass 裝置角色：
  - 允許`credentials.iot.amazonaws.com`和擔任角色`sagemaker.amazonaws.com`的信任關係，如以下 IAM 政策範例所示。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
       },
      "Action": "sts:AssumeRole"
    },
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "sagemaker.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

  - AmazonSageMakerEdgeDeviceFleetPolicyIAM 受管政策。
  - AmazonSageMakerFullAccessIAM 受管政策。

- 包含元件成品的 S3 儲存貯體的`s3:GetObject`動作，如以下 IAM 政策範例所示。

```
{
  "Version": "2012-10-17",
  "Statement": [
  {
    "Action": [
      "s3:GetObject"
    ],
    "Resource": [
      "arn:aws:s3:::DOC-EXAMPLE-BUCKET1/*"
    ],
    "Effect": "Allow"
  }
  ]
}
```

在邊緣管理器中設置您的 Greengrass 核心設備 SageMaker

邊緣管理員中的 SageMaker 邊緣裝置叢集是邏輯分組裝置的集合。若要搭配使用 SageMaker Edge ManagerAWS IoT Greengrass，您必須建立邊緣裝置叢集，該叢集使用與您部署 SageMaker Edge Manager 代理程式之 Greengrass 核心裝置相同的AWS IoT角色別名。然後，您必須將核心裝置註冊為該叢集的一部分。

主題

- 建立邊緣裝置叢集
- 註冊 Greengrass 設備

建立邊緣裝置叢集

建立邊緣裝置叢集 (主控台)

1. 在 Amazon 主 SageMaker 控台中，選擇邊緣管理員，然後選擇邊緣裝置叢集。
2. 在 [裝置叢集] 頁面上，選擇 [建立裝置叢集]。
3. 在 [裝置叢集內容] 底下，執行下列動作：
   - 針對裝置叢集名稱，輸入裝置叢集的名稱。

- 對於 IAM 角色，請輸入您在設定 Greengrass 核心裝置時指定的AWS IoT角色別名的 Amazon 資源名稱 (ARN)。
- 停用「建立 IAM 角色別名」切換。

4. 選擇下一步。
5. 在「輸出組態」下，針對 S3 儲存貯體 URI，輸入您要與裝置叢集建立關聯的 S3 儲存貯體 URI。
6. 選擇提交。

註冊 Greengrass 設備

將您的 Greengrass 核心裝置註冊為邊緣裝置 (主控台)

1. 在 Amazon 主 SageMaker 控台中，選擇邊緣管理員，然後選擇邊緣裝置。
2. 在 [裝置] 頁面上，選擇 [註冊裝置]。
3. 在 [裝置內容] 底下，對於 [裝置群名稱]，輸入您所建立之裝置叢集的名稱，然後選擇 [下一步]。
4. 選擇下一步。
5. 在 [裝置來源] 下，對於 [裝置名稱]，輸入 Greengrass 核心裝置的AWS IoT物件名稱。
6. 選擇提交。

建立範例元件

AWS 雲端為了協助您開始使用 SageMaker Edge Manager 元件，請在上AWS提供 Python 指令碼 GitHub ，以建立範例推論和模型元件，並將它們上傳到中。在開發電腦上完成下列步驟。

建立範例元件的步驟

1. 將AWS IoT Greengrass元件範例儲存庫下載 GitHub 至您的開發電腦。
2. 導航到下載的/machine-learning/sagemaker-edge-manager文件夾。

```
cd download-directory/machine-learning/sagemaker-edge-manager
```

3. 執行下列命令，以建立範例元件並將其上傳至AWS 雲端。

```
python3 create_components.py -r region -b DOC-EXAMPLE-BUCKET
```

將*##*取代為AWS 區域您建立 Greengrass 核心裝置的位置，並將 *DOC-EXAMPLE-BUCKET1* 取代為 S3 儲存貯體的名稱，以存放元件成品。

> **Note**
>
> 根據預設，指令碼會為影像分類和物件偵測推論建立範例元件。若要僅針對特定類型的推論建立元件，請指定`-i` *`ImageClassification | ObjectDetection`*引數。

範例推論和模型元件，以配合 SageMaker 邊緣管理員使用，現在會在您的AWS 帳戶. 若要在AWS IoT Greengrass主控台中查看範例元件，請選擇 [元件]，然後在 [我的元件] 底下搜尋下列元件：

- `com.greengrass.SageMakerEdgeManager.ImageClassification`
- `com.greengrass.SageMakerEdgeManager.ImageClassification.Model`
- `com.greengrass.SageMakerEdgeManager.ObjectDetection`
- `com.greengrass.SageMakerEdgeManager.ObjectDetection.Model`

執行範例影像分類推論

若要使用AWS提供的範例元件和 SageMaker Edge Manager 代理程式執行映像分類推論，您必須將這些元件部署到核心裝置。部署這些元件會下載 SageMaker 新編譯的預先訓練 Resnet-50 模型，並在您的裝置上安裝 SageMaker 邊緣管理員代理程式。 SageMaker Edge 管理員代理程式會載入模型，並發佈有關該`gg/sageMakerEdgeManager/image-classification`主題的推論結果。若要檢視這些推論結果，請使用主控台中的 AWS IoT MQTT 用戶端來訂閱此AWS IoT主題。

主題

- 訂閱通知主題
- 部署範例元件
- 檢視推論結果

訂閱通知主題

在此步驟中，您可以在AWS IoT主控台中設定 AWS IoT MQTT 用戶端，以監視範例推論元件所發佈的 MQTT 訊息。依預設，元件會針對`gg/sageMakerEdgeManager/image-classification`主題發

佈推論結果。在將元件部署到 Greengrass 核心裝置之前，請先訂閱本主題，以便在元件第一次執行時查看推論結果。

訂閱預設通知主題

1. 在AWS IoT主控台導覽功能表中，選擇 [測試]、[MQTT 測試用戶端]。
2. 在 [訂閱主題] 下的 [主題名稱] 方塊中，輸入**gg/sageMakerEdgeManager/image-classification**。
3. 選擇 Subscribe (訂閱)。

部署範例元件

在此步驟中，您可以設定下列元件並將其部署到核心裝置：

- `aws.greengrass.SageMakerEdgeManager`
- `com.greengrass.SageMakerEdgeManager.ImageClassification`
- `com.greengrass.SageMakerEdgeManager.ImageClassification.Model`

若要部署您的元件 (主控台)

1. 在AWS IoT Greengrass主控台導覽功能表中，選擇「部署」，然後為您要修訂的目標裝置選擇部署。
2. 在部署頁面上，選擇 [修訂]，然後選擇 [修訂部署]。
3. 在 [指定目標] 頁面上，選擇 [下一步]。
4. 在 [選取元件] 頁面上，執行下列動作：
   a. 在我的零組件之下，選擇下列零組件：
      - `com.greengrass.SageMakerEdgeManager.ImageClassification`
      - `com.greengrass.SageMakerEdgeManager.ImageClassification.Model`
   b. 在 [公用元件] 下，關閉 [僅顯示選取的元件] 切換，然後選取`aws.greengrass.SageMakerEdgeManager`元件。
   c. 選擇下一步。
5. 在 [設定元件] 頁面上，選取`aws.greengrass.SageMakerEdgeManager`元件並執行下列動作。

a. 選擇 設定元件。

b. 在組態更新下的要合併的組態中，輸入下列組態。

```
{
    "DeviceFleetName": "device-fleet-name",
    "BucketName": "DOC-EXAMPLE-BUCKET"
}
```

以您建立的邊緣裝置叢集名稱取代，並以*device-fleet-name*與您的裝置叢集相關聯的 S3 儲存貯體的名稱取代 *DOC/EXAMPLE-* BUCKET 儲存貯體。

c. 選擇確認，然後選擇下一步。

6. 在設定進階設定頁面上，保留預設組態設定，然後選擇 下一步。

7. 在「複查」頁面上，選擇「建置」

若要部署您的元件 (AWS CLI)

1. 在您的開發電腦上，建立一個`deployment.json`檔案來定義 SageMaker Edge Manager 元件的部署組態。該檔案應如以下範例所示。

```
{
  "targetArn":"targetArn",
  "components": {
    "aws.greengrass.SageMakerEdgeManager": {
      "componentVersion": "1.0.x",
      "configurationUpdate": {
        "merge": "{\"DeviceFleetName\":\"device-fleet-name\",\"BucketName\":\"DOC-
EXAMPLE-BUCKET2\"}"
      }
    },
    "com.greengrass.SageMakerEdgeManager.ImageClassification": {
      "componentVersion": "1.0.x",
      "configurationUpdate": {
      }
    },
    "com.greengrass.SageMakerEdgeManager.ImageClassification.Model": {
      "componentVersion": "1.0.x",
      "configurationUpdate": {
      }
    },
```

```
  }
}
```

- 在targetArn欄位中，*targetArn*以下列格式將物件或物群組的 Amazon 資源名稱 (ARN) 取代為目標部署：
  - 物件：arn:aws:iot:*region*:*account-id*:thing/*thingName*
  - 物件群組：arn:aws:iot:*region*:*account-id*:thinggroup/*thingGroupName*
- 在merge欄位中，以*device-fleet-name*您建立的 Edge 裝置叢集的名稱取代。然後，將 *DOC-EXAMPLE-BUCKET2* 取代為與裝置叢集相關聯的 S3 儲存貯體的名稱。
- 以最新的可用版本取代每個元件的元件版本。

2. 執行下列命令以在裝置上部署元件：

```
aws greengrassv2 create-deployment \
    --cli-input-json file://path/to/deployment.json
```

可能需要幾分鐘才能完成部署。在下一個步驟中，檢查元件記錄檔以確認部署是否已順利完成，並檢視推論結果。

檢視推論結果

部署元件之後，您可以在 Greengrass 核心裝置的元件記錄檔和主控台的 AWS IoT MQTT 用戶端中檢視推論結果。AWS IoT若要訂閱元件發佈推論結果的主題，請參閱訂閱通知主題。

- AWS IoTMQTT 用戶端 — 若要檢視推論元件在預設通知主題上發佈的結果，請完成下列步驟：

  1. 在AWS IoT主控台導覽功能表中，選擇 [測試]、[MQTT 測試用戶端]。
  2. 在「訂閱」下，選擇**gg/sageMakerEdgeManager/image-classification**。

- 元件記錄 — 若要在元件記錄檔中檢視推論結果，請在 Greengrass 核心裝置上執行下列命令。

```
sudo tail -f /greengrass/v2/logs/
com.greengrass.SageMakerEdgeManager.ImageClassification.log
```

如果您在元件記錄檔或 MQTT 用戶端中看不到推論結果，表示部署失敗或無法連線至核心裝置。如果您的核心裝置未連線至網際網路，或是沒有執行元件的正確權限，就會發生這種情況。在核心裝置上執行下列命令，以檢視 AWS IoT Greengrass Core 軟體記錄檔。此檔案包含來自 Greengrass 核心裝置部署服務的記錄檔。

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

如需更多詳細資訊，請參閱 機器學習推論疑難排解。

教學課程：使 TensorFlow 用 Lite 執行範例影像分類推論

本教學課程說明如何使用 TensorFlow Lite 影像分類推論元件，在 Greengrass 核心裝置上執行範例影像分類推論。此元件包括下列元件相依性：

- TensorFlow 精簡版圖像分類模型存儲組件
- TensorFlow 精簡版運行組件

部署此元件時，它會下載預先訓練的 MobileNet v1 模型，並安裝 TensorFlow Lite 執行階段及其相依性。此元件會發佈有關該`ml/tflite/image-classification`主題的推論結果。若要檢視這些推論結果，請使用主控台中的 AWS IoT MQTT 用戶端來訂閱此AWS IoT主題。

在本教學課程中，您將部署範例推論元件，以對提供的範例映像執行映像分類。AWS IoT Greengrass 完成此教學課程之後，您可以完成教學課程：使 TensorFlow 用 Lite 對相機中的影像執行範例影像分類推論，其中會示範如何修改範例推論元件，以便在 Greengrass 核心裝置上本機相機的影像上執行影像分類。

如需 Greengrass 裝置上機器學習的詳細資訊，請參閱。執行機器學習推論

主題

- 必要條件
- 步驟 1：訂閱預設通知主題
- 步驟 2：部署 TensorFlow 精簡版影像分類元件
- 步驟 3：檢視推論結果
- 後續步驟

必要條件

為了完成本教學，您需要以下項目：

- 一個 Linux Greengrass 核心設備。如果您沒有帳戶，請參閱 教學課程：AWS IoT Greengrass V2 入門。核心裝置必須符合下列要求：
  - 在運行 Amazon Linux 2 或 Ubuntu 18.04 的 Greengrass 核心設備上，GNU C 庫（glibc）2.27 版或更高版本安裝在設備上。
  - 在 ARMV7L 設備上，如樹莓派，對於設備上安裝了 OpenCV-Python 的依賴關係。執行下列命令以安裝相依性。

    ```
    sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
     libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
    ```

  - 樹莓派運行樹莓派 OS 靶心設備必須滿足以下要求:
    - NumPy 1.22.4 或更新版本已安裝在裝置上。樹莓派 OS 靶心包括早期版本的 NumPy，因此您可以運行以下命令在設備 NumPy 上升級。

      ```
      pip3 install --upgrade numpy
      ```

    - 裝置上啟用的舊式攝影機堆疊。樹莓派 OS Bullseye 包括一個新的相機堆疊，預設情況下啟用且不相容，因此您必須啟用傳統的相機堆疊。

      啟用舊式相機堆疊的步驟

      1. 運行以下命令以打開樹莓派配置工具。

         ```
         sudo raspi-config
         ```

      2. 選取介面選項。
      3. 選取「舊式相機」以啟用舊式相機堆疊。
      4. 重新啟動 Raspberry Pi。

步驟 1：訂閱預設通知主題

在此步驟中，您可以在AWS IoT主控台中設定 AWS IoT MQTT 用戶端，以觀看由 TensorFlow Lite 映像分類元件發佈的 MQTT 訊息。依預設，元件會針對`ml/tflite/image-classification`主題發

佈推論結果。在將元件部署到 Greengrass 核心裝置之前，請先訂閱本主題，以便在元件第一次執行時查看推論結果。

訂閱預設通知主題

1. 在AWS IoT主控台導覽功能表中，選擇 [測試]、[MQTT 測試用戶端]。
2. 在 [訂閱主題] 下的 [主題名稱] 方塊中，輸入**ml/tflite/image-classification**。
3. 選擇 Subscribe (訂閱)。

步驟 2：部署 TensorFlow 精簡版影像分類元件

在此步驟中，您將 TensorFlow Lite 映像分類元件部署到核心裝置：

若要部署 L TensorFlow ite 映像分類元件 (主控台)

1. 在AWS IoT Greengrass主控台瀏覽功能表中，選擇 [元件]。
2. 在元面頁面上的公用元件索引標籤上，選擇 `aws.greengrass.TensorFlowLiteImageClassification`。
3. 在 aws.greengrass.TensorFlowLiteImageClassification頁面中，選擇部署。
4. 從新增至部署中，選擇下列其中一項：
   a. 若要將此元件合併至目標裝置上的現有部署，請選擇 [新增至現有部署]，然後選取要修訂的部署。
   b. 若要在目標裝置上建立新部署，請選擇 [建立新部署]。如果您的設備上有現有的部署，則選擇此步驟將取代現有部署。
5. 在指定目標頁面上，執行下列作業：
   a. 在部署資訊下，輸入或修改部署的易記名稱。
   b. 在部署目標下，選取部署的目標，然後選擇下一步。如果您要修訂既有部署，則無法變更部署目標。
6. 在 [選取元件] 頁面的 [公用元件] 下，確認已選取`aws.greengrass.TensorFlowLiteImageClassification`元件，然後選擇 [下一步]。
7. 在 [設定元件] 頁面上，保留預設組態設定，然後選擇 [下一步]。
8. 在設定進階設定頁面上，保留預設組態設定，然後選擇 下一步。
9. 在「複查」頁面上，選擇「建置」

若要部署 L TensorFlow ite 映像分類元件 (AWS CLI)

1. 建立deployment.json檔案以定義 TensorFlow Lite 影像分類元件的部署規劃。此檔案應如下所示：

```
{
  "targetArn":"targetArn",
  "components": {
    "aws.greengrass.TensorFlowLiteImageClassification": {
      "componentVersion": 2.1.0,
      "configurationUpdate": {
      }
    }
  }
}
```

   - 在targetArn欄位中，*targetArn*以下列格式將物件或物群組的 Amazon 資源名稱 (ARN) 取代為目標部署：
     - 物件：arn:aws:iot:*region*:*account-id*:thing/*thingName*
     - 物件群組：arn:aws:iot:*region*:*account-id*:thinggroup/*thingGroupName*
   - 本自學課程使用元件版本 2.1.0。
     在aws.greengrass.TensorFlowLiteObjectDetection組件物件中，取代 *2.1.0* 以使用不同版本的 TensorFlow Lite 物件偵測元件。
2. 執行下列命令，在裝置上部署 TensorFlow Lite 映像分類元件：

```
aws greengrassv2 create-deployment \
    --cli-input-json file://path/to/deployment.json
```

可能需要幾分鐘才能完成部署。在下一個步驟中，檢查元件記錄檔以確認部署是否已順利完成，並檢視推論結果。

步驟 3：檢視推論結果

部署元件之後，您可以在 Greengrass 核心裝置的元件記錄檔和主控台的 AWS IoT MQTT 用戶端中檢視推論結果。AWS IoT若要訂閱元件發佈推論結果的主題，請參閱步驟 1：訂閱預設通知主題。

- AWS IoTMQTT 用戶端 — 若要檢視推論元件在預設通知主題上發佈的結果，請完成下列步驟：

1. 在AWS IoT主控台導覽功能表中，選擇 [測試]、[MQTT 測試用戶端]。
2. 在「訂閱」下，選擇**ml/tflite/image-classification**。

   您應該會看到類似下列範例的訊息。

```
{
  "timestamp": "2021-01-01 00:00:00.000000",
  "inference-type": "image-classification",
  "inference-description": "Top 5 predictions with score 0.3 or above ",
  "inference-results": [
    {
      "Label": "cougar, puma, catamount, mountain lion, painter, panther, Felis
 concolor",
      "Score": "0.5882352941176471"
    },
    {
      "Label": "Persian cat",
      "Score": "0.5882352941176471"
    },
    {
      "Label": "tiger cat",
      "Score": "0.5882352941176471"
    },
    {
      "Label": "dalmatian, coach dog, carriage dog",
      "Score": "0.5607843137254902"
    },
    {
      "Label": "malamute, malemute, Alaskan malamute",
      "Score": "0.5450980392156862"
    }
  ]
}
```

- 元件記錄 — 若要在元件記錄檔中檢視推論結果，請在 Greengrass 核心裝置上執行下列命令。

```
sudo tail -f /greengrass/v2/logs/aws.greengrass.TensorFlowLiteImageClassification.log
```

您應該會看到類似下列範例的結果。

```
2021-01-01 00:00:00.000000 [INFO] (Copier)
 aws.greengrass.TensorFlowLiteImageClassification: stdout. Publishing results to the
 IoT core....
 {scriptName=services.aws.greengrass.TensorFlowLiteImageClassification.lifecycle.Run.script,
 serviceName=aws.greengrass.TensorFlowLiteImageClassification, currentState=RUNNING}

2021-01-01 00:00:00.000000 [INFO] (Copier)
 aws.greengrass.TensorFlowLiteImageClassification: stdout. {"timestamp":
 "2021-01-01 00:00:00.000000", "inference-type": "image-classification", "inference-
description": "Top 5 predictions with score 0.3 or above ", "inference-results":
 [{"Label": "cougar, puma, catamount, mountain lion, painter, panther, Felis
 concolor", "Score": "0.5882352941176471"}, {"Label": "Persian cat", "Score":
 "0.5882352941176471"}, {"Label": "tiger cat", "Score": "0.5882352941176471"},
 {"Label": "dalmatian, coach dog, carriage dog", "Score": "0.5607843137254902"},
 {"Label": "malamute, malemute, Alaskan malamute", "Score": "0.5450980392156862"}]}.
 {scriptName=services.aws.greengrass.TensorFlowLiteImageClassification.lifecycle.Run.script,
 serviceName=aws.greengrass.TensorFlowLiteImageClassification, currentState=RUNNING}
```

如果您在元件記錄檔或 MQTT 用戶端中看不到推論結果，表示部署失敗或無法連線至核心裝置。如果您的核心裝置未連線至網際網路，或是沒有執行元件的正確權限，就會發生這種情況。在核心裝置上執行下列命令，以檢視 AWS IoT Greengrass Core 軟體記錄檔。此檔案包含來自 Greengrass 核心裝置部署服務的記錄檔。

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

如需詳細資訊，請參閱 機器學習推論疑難排解。

後續步驟

如果您的 Greengrass 核心裝置具有支援的攝影機介面，您可以完成此操作教學課程：使 TensorFlow 用 Lite 對相機中的影像執行範例影像分類推論，其中顯示如何修改範例推論元件，以便對來自攝影機的影像執行影像分類。

若要進一步探索範例 TensorFlow Lite 影像分類推論元件的組態，請嘗試下列動作：

- 修改組`InferenceInterval`態參數以變更推論程式碼執行的頻率。
- 修改推論元件`ImageDirectory`組態中的`ImageName`和組態參數，以指定用於推論的自訂影像。

如需自訂公用元件組態或建立自訂機器學習元件的相關資訊，請參閱自訂您的機器學習元件。

教學課程：使 TensorFlow 用 Lite 對相機中的影像執行範例影像分類推論

本教學課程說明如何使用 TensorFlow Lite 影像分類推論元件，對 Greengrass 核心裝置上本機攝影機的影像執行範例影像分類推論。此元件包括下列元件相依性：

- TensorFlow 精簡版圖像分類模型存儲組件
- TensorFlow 精簡版運行組件

> **Note**
>
> 本教程訪問樹莓派或 NVIDIA 傑特森納米設備攝像頭模塊, 但AWS IoT Greengrass支持 ARMv7L 上的其他設備, Armv8, 或 x86_64 平台. 若要為其他裝置設定相機，請參閱您裝置的相關文件。

如需 Greengrass 裝置上機器學習的詳細資訊，請參閱。執行機器學習推論

主題

必要條件

若要完成此自學課程，您必須先完成教學課程：使 TensorFlow 用 Lite 執行範例影像分類推論。

您也需要下列項目：

- 一種具 Greengrass 攝像機接口的 Linux 核心設備。本教學課程會存取下列其中一個受支援裝置上的相機模組：
  - 樹莓派運行樹莓派操作系統 (以前稱為覆盆子)

- 奈米傑森

如需有關設定 Greengrass 核心裝置的資訊，請參閱。教學課程：AWS IoT Greengrass V2 入門

核心裝置必須符合下列要求：

- 在運行 Amazon Linux 2 或 Ubuntu 18.04 的 Greengrass 核心設備上，GNU C 庫（glibc）2.27 版或更高版本安裝在設備上。
- 在 ARMV7L 設備上，如樹莓派，對於設備上安裝了 OpenCV-Python 的依賴關係。執行下列命令以安裝相依性。

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
 libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- 樹莓派運行樹莓派操作系統靶心設備必須滿足以下要求:
  - NumPy 1.22.4 或更新版本已安裝在裝置上。樹莓派 OS 靶心包括早期版本的 NumPy，因此您可以運行以下命令在設備 NumPy 上升級。

    ```
    pip3 install --upgrade numpy
    ```

  - 裝置上啟用的舊式攝影機堆疊。樹莓派 OS Bullseye 包括一個新的相機堆疊，預設情況下啟用且不相容，因此您必須啟用傳統的相機堆疊。

    啟用舊式相機堆疊

    1. 運行以下命令以打開樹莓派配置工具。

       ```
       sudo raspi-config
       ```

    2. 選取介面選項。
    3. 選取「舊式相機」以啟用舊式相機堆疊。
    4. 重新啟動 Raspberry Pi。
- 對於樹莓派或 NVIDIA 傑特森納米設備，樹莓派相機模塊 V2-8 百萬像素，1080p 的。若要了解如何設定相機，請參閱 Raspberry Pi 文件中的連接相機。

步驟 1：在設備上配置相機模塊

在此步驟中，您將安裝並啟用裝置的相機模組。在設備上運行以下命令。

Raspberry Pi (Armv7l)

1. 安裝攝影機模組的picamera介面。執行下列命令以安裝攝影機模組和本教學課程所需的其他 Python 程式庫。

   ```
   sudo apt-get install -y python3-picamera
   ```

2. 驗證皮卡馬拉已成功安裝。

   ```
   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**
   >
   > 如果設備上安裝的 python3 Python 可執行文件是python3.7，請使用python3.7而不是本教程中的命令。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. 重新啟動裝置。

   ```
   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. 執行下列指令以測試相機設定。

   ```
   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。

7. 執行下列命令以建立符號連結，讓推論元件能夠從執行階段元件所建立的虛擬環境存取您的攝影機。

```
sudo ln -s /usr/lib/python3/dist-packages/picamera "MLRootPath/
greengrass_ml_tflite_venv/lib/python3.7/site-packages"
```

本教學課程的 *ML RootPath* 預設值為*/greengrass/v2*/work/variant.TensorFlowLite/greengrass_ml。在中第一次部署推論元件時，會建立此位置中教學課程：使 TensorFlow 用 Lite 執行範例影像分類推論的greengrass_ml_tflite_venv資料夾。

Jetson Nano (Armv8)

1. 執行下列指令以測試相機設定。

   ```
   gst-launch-1.0 nvarguscamerasrc num-buffers=1 ! "video/x-raw(memory:NVMM),
    width=1920, height=1080, format=NV12, framerate=30/1" ! nvjpegenc ! filesink
    location=test.jpg
   ```

   這會擷取並儲存名test.jpg為目前目錄的影像。

2. (選擇性) 重新啟動裝置。如果在上一個步驟中執行命gst-launch令時遇到問題，重新啟動裝置可能會解決這些問題。

   ```
   sudo reboot
   ```

> **Note**
>
> 對於 Armv8 (AArch64) 裝置 (例如 Jetson Nano)，您不需要建立符號連結，即可讓推論元件從執行階段元件建立的虛擬環境存取攝影機。

步驟 2：驗證您對預設通知主題的訂閱

在中教學課程：使 TensorFlow 用 Lite 執行範例影像分類推論，您已在AWS IoT主控台中設定 AWS IoT MQTT 用戶端，以監視由 TensorFlow Lite 映像分類元件針對主題發佈的 MQTT 訊息。ml/tflite/image-classification在主AWS IoT控台中，確認此訂閱存在。如果沒有，請在步驟 1：訂閱預設通知主題將元件部署到 Greengrass 核心裝置之前，遵循中的步驟訂閱本主題。

步驟 3：修改 TensorFlow Lite 映像分類元件組態並加以部署

在此步驟中，您可以設定 TensorFlow Lite 映像分類元件並將其部署至核心裝置：

若要設定和部署 TensorFlow Lite 映像分類元件 (主控台)

1. 在AWS IoT Greengrass主控台瀏覽功能表中，選擇 [元件]。
2. 在元面頁面上的公用元件索引標籤上，選擇 `aws.greengrass.TensorFlowLiteImageClassification`。
3. 在 aws.greengrass.TensorFlowLiteImageClassification頁面中，選擇部署。
4. 從新增至部署中，選擇下列其中一項：
   a. 若要將此元件合併至目標裝置上的現有部署，請選擇 [新增至現有部署]，然後選取要修訂的部署。
   b. 若要在目標裝置上建立新部署，請選擇 [建立新部署]。如果您的設備上有現有的部署，則選擇此步驟將取代現有部署。
5. 在指定目標頁面上，執行下列作業：
   a. 在部署資訊下，輸入或修改部署的易記名稱。
   b. 在部署目標下，選取部署的目標，然後選擇下一步。如果您要修訂既有部署，則無法變更部署目標。
6. 在 [選取元件] 頁面的 [公用元件] 下，確認已選取`aws.greengrass.TensorFlowLiteImageClassification`元件，然後選擇 [下一步]。
7. 在 [設定元件] 頁面上，執行下列動作：
   a. 選取推論元件，然後選擇設定元件。
   b. 在「組態更新」下，在「要合併的組態」方塊中輸入下列組態更新。

      ```
      {
        "InferenceInterval": "60",
        "UseCamera": "true"
      }
      ```

      透過此組態更新，元件會存取裝置上的攝影機模組，並對相機拍攝的影像執行推論。推論程式碼每 60 秒執行一次。
   c. 選擇確認，然後選擇下一步。
8. 在設定進階設定頁面上，保留預設組態設定，然後選擇 下一步。

9. 在「複查」頁面上，選擇「建置」

若要設定和部署 L TensorFlow ite 映像分類元件 (AWS CLI)

1. 建立`deployment.json`檔案以定義 TensorFlow Lite 影像分類元件的部署規劃。此檔案看起來應如下所示：

```
{
  "targetArn":"targetArn",
  "components": {
    "aws.greengrass.TensorFlowLiteImageClassification": {
      "componentVersion": 2.1.0,
      "configurationUpdate": {
        "InferenceInterval": "60",
        "UseCamera": "true"
      }
    }
  }
}
```

   - 在`targetArn`欄位中，*targetArn*以下列格式將物件或物群組的 Amazon 資源名稱 (ARN) 取代為目標部署：
     - 物件：`arn:aws:iot:`*region*`:`*account-id*`:thing/`*thingName*
     - 物件群組：`arn:aws:iot:`*region*`:`*account-id*`:thinggroup/`*thingGroupName*
   - 本自學課程使用元件版本 2.1.0。
     在`aws.greengrass.TensorFlowLiteImageClassification`元件物件中，取代 *2.1.0* 以使用不同版本的 TensorFlow Lite 影像分類元件。

   透過此組態更新，元件會存取裝置上的攝影機模組，並對相機拍攝的影像執行推論。推論程式碼每 60 秒執行一次。取代下列值

2. 執行下列命令，在裝置上部署 TensorFlow Lite 映像分類元件：

```
aws greengrassv2 create-deployment \
    --cli-input-json file://path/to/deployment.json
```

可能需要幾分鐘才能完成部署。在下一個步驟中，檢查元件記錄檔以確認部署是否已順利完成，並檢視推論結果。

步驟 4：檢視推論結果

部署元件之後，您可以在 Greengrass 核心裝置的元件記錄檔和主控台的 AWS IoT MQTT 用戶端中檢視推論結果。AWS IoT若要訂閱元件發佈推論結果的主題，請參閱步驟 2：驗證您對預設通知主題的訂閱。

- AWS IoTMQTT 用戶端 — 若要檢視推論元件在預設通知主題上發佈的結果，請完成下列步驟：
  1. 在AWS IoT主控台導覽功能表中，選擇 [測試]、[MQTT 測試用戶端]。
  2. 在「訂閱」下，選擇**ml/tflite/image-classification**。
- 元件記錄 — 若要在元件記錄檔中檢視推論結果，請在 Greengrass 核心裝置上執行下列命令。

```
sudo tail -f /greengrass/v2/logs/aws.greengrass.TensorFlowLiteImageClassification.log
```

如果您在元件記錄檔或 MQTT 用戶端中看不到推論結果，表示部署失敗或無法連線至核心裝置。如果您的核心裝置未連線至網際網路，或者沒有執行元件所需的權限，就會發生這種情況。在核心裝置上執行下列命令，以檢視 AWS IoT Greengrass Core 軟體記錄檔。此檔案包含來自 Greengrass 核心裝置部署服務的記錄檔。

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

如需詳細資訊，請參閱 機器學習推論疑難排解。

後續步驟

本教學課程說明如何使用 TensorFlow Lite 影像分類元件，以及自訂組態選項，對相機拍攝的影像執行範例影像分類。

如需自訂公用元件組態或建立自訂機器學習元件的詳細資訊，請參閱自訂您的機器學習元件。

元件

AWS IoT Greengrass元件是您部署到 Greengrass 核心裝置的軟體模組。組件可以代表應用程序，運行時安裝程序，庫或您將在設備上運行的任何代碼。您可以定義相依於其他元件的元件。例如，您可以定義安裝 Python 的組件，然後將該組件定義為運行 Python 應用程序的組件的依賴項。當您將元件部署到裝置群組時，Greengrass 只會部署裝置所需的軟體模組。

主題

AWS-提供的組件

AWS IoT Greengrass 提供並維護預先建置的元件，您可以將這些元件部署到您的裝置。這些元件包括功能 (例如串流管理員)、 AWS IoT Greengrass V1 連接器 (例如 CloudWatch 指標) 和本機開發工具 (例如 AWS IoT Greengrass CLI)。您可以將這些元件部署到裝置以取得獨立功能，也可以將它們用作自訂 Greengr ass 元件中的相依性。

> **Note**
>
> 幾個 AWS提供的組件取決於 Greengrass 核的特定次要版本。由於這種依賴關係，您需要在將 Greengrass 核更新為新的次要版本時更新這些組件。如需每個元件所依賴之特定原子核版本的詳細資訊，請參閱對應的元件主題。如需更新核心的詳細資訊，請參閱更新AWS IoT Greengrass核心軟件（OTA）。

當組件具有泛型和 Lambda 的組件類型時，該組件的當前版本是泛型類型，而先前版本的組件是 Lambda 類型。

元件	描述	元件類型	支援的作業系統	開放原始碼
Greengrass 核	核 AWS IoT Greengrass 心軟件的核心。使用此元件可設定和更新核心裝置上的軟體。	原子核	Linux、Windows	是
用戶端裝置驗證	讓本機 IoT 裝置 (稱為用戶端裝置) 連線至核心裝置。	外掛程式	Linux、Windows	是
CloudWatch 度量	將自訂指標發佈到 Amazon CloudWatch。	泛型，Lambda	Linux、Windows	是
AWS IoT Device Defender	通知管理員 Greengrass 核心裝置狀態的變更，以識別不尋常的行為。	泛型，Lambda	Linux、Windows	是
磁碟後臺解決程式	為從 Greengrass 核心裝置多工緩衝處理的訊息啟用持續性儲存選項。 AWS IoT Core此元件會將這些輸出郵件儲存在磁碟上。	外掛程式	Linux、Windows	是

元件	描述	元件類型	支援的作業系統	開放原始碼
碼頭應用程序管理器	能 AWS IoT Greengrass 夠從碼頭集線器和 Amazon Elastic Container Registry (Amazon ECR) 下載碼頭映像。	一般	Linux、Windows	否
Kinesis Video Streams 的邊緣連接器	從本機攝影機讀取視訊饋送、將串流發佈至 Kinesis Video Streams，並使用 Grafana 儀表板顯示串流。 AWS IoT TwinMaker	一般	Linux	否
Greengrass	提供指令列介面，您可以使用此介面建立本機部署，並與 Greengrass 核心裝置及其元件互動。	外掛程式	Linux、Windows	是

元件	描述	元件類型	支援的作業系統	開放原始碼
IP 偵測器	向其報告 MQTT 代理程式連線資訊 AWS IoT Greengrass，以便用戶端裝置可以探索如何連線。	外掛程式	Linux、Windows	是
Firehose	透過 Amazon 資料 Firehose 交付串流將資料發佈到中的目的 AWS 雲端地。	Lambda	Linux	否
发 Lambda 器	處理 Lambda 函數的程序和環境組態。	一般	Linux	否
Lambda 經理	處理 Lambda 函數的處理序間通訊和調整。	外掛程式	Linux	否
Lambda 執行期	為每個 Lambda 執行階段提供成品。	一般	Linux	否
舊版訂閱路由器	管理在 AWS IoT Greengrass V1 上執行之 Lambda 函數的訂閱。	一般	Linux	否

元件	描述	元件類型	支援的作業系統	開放原始碼
本機除錯主控台	提供本機主控台，您可以用來偵錯和管理 Greengrass 核心裝置及其元件。	外掛程式	Linux、Windows	是
日誌管理器	在 Greengrass 核心設備上收集和上傳日誌。	外掛程式	Linux、Windows	是
機器學習元件	提供機器學習模型和範例推論程式碼，您可以使用這些程式碼在 Greengrass 核心裝置上執行機器學習推論。	請參閱 機器學習元件。		
通訊協定介面卡	調查來自本地 Modbus RTU 設備的信息。	Lambda	Linux	否
核遙測發射器	將從核心收集的系統健康情況遙測資料發佈到本機主題或 AWS IoT Core MQTT 主題。	外掛程式	Linux、Windows	是

元件	描述	元件類型	支援的作業系統	開放原始碼
MQTT 大橋	在用戶端裝置、本機 AWS IoT Greengrass 發佈/訂閱和之間轉送 MQTT 訊息。AWS IoT Core	外掛程式	Linux、Windows	是
MQTT 3.1.1 經紀商 (平均)	執行 MQTT 3.1.1 代理程式，可處理用戶端裝置與核心裝置之間的訊息。	外掛程式	Linux、Windows	是
MQTT 5 經紀商	執行 MQTT 5 代理程式，以處理用戶端裝置與核心裝置之間的訊息。	一般	Linux、Windows	否
PKCS #11 供應商	讓 Greengrass 元件能夠存取您安全地儲存在硬體安全性模組 (HSM) 中的私密金鑰和憑證。	外掛程式	Linux	是

元件	描述	元件類型	支援的作業系統	開放原始碼
秘密經理	從 AWS Secrets Manager 密碼部署密碼，以便您可以在 Greengrass 核心裝置的自訂元件中安全地使用認證，例如密碼。	外掛程式	Linux、Windows	是
安全隧道	啟用 AWS IoT 安全通道連線，您可以使用這些連線與受限防火牆後方的 Greengrass 核心裝置建立投標通訊。	一般	Linux	否
陰影管理	啟用與核心裝置上的陰影互動。它會管理陰影文件儲存，以及本機陰影狀態與 AWS IoT Device Shadow 服務的同步處理。	外掛程式	Linux、Windows	是

元件	描述	元件類型	支援的作業系統	開放原始碼
Amazon SNS	將訊息發佈到 Amazon SNS 主題。	Lambda	Linux	否
串流管理員	將大量資料從本機來源串流至. AWS 雲端	一般	Linux、Windows	否
Systems Manager 代理	使用管理核心裝置 AWS Systems Manager，讓您修補裝置、執行命令等。	一般	Linux	否
代幣交換服務	提供您可用來與 AWS 服務互動的 AWS 認證。	一般	Linux、Windows	否
IoT 集 SiteWise 電極	從 OPC-UA 伺服器收集資料。	一般	Linux、Windows	否
IoT SiteWise OPC-UA 數據源模擬器	執行產生範例資料的本機 OPC-UA 伺服器。	一般	Linux、Windows	否
IoT SiteWise 出版	將資料發佈至 AWS 雲端。	一般	Linux、Windows	否

元件	描述	元件類型	支援的作業系統	開放原始碼
IoT SiteWise 處理	處理 Greengrass 核心設備上的數據。	一般	Linux、Windows	否

Greengrass 核

Greengrass 核組件（`aws.greengrass.Nucleus`）是一個強制性組件，也是在設備上運行 AWS IoT Greengrass 核心軟件的最低要求。您可以將此元件設定為遠端自訂和更新您的 AWS IoT Greengrass Core 軟體。部署此元件以設定核心裝置上的 Proxy、裝置角色和 AWS IoT 物件組態等設定。

> **Important**
>
> 當核心元件的版本變更時，或當您變更某些組態參數時， AWS IoT Greengrass Core 軟體 (包括核心和裝置上的所有其他元件) 會重新啟動以套用變更。
> 當您部署元件時， AWS IoT Greengrass 會安裝該元件所有相依性的最新受支援版本。因此，如果您將新裝置新增至物件群組，或更新以這些裝置為目標的部署，則 AWS提供之公用元件的新修補程式版本可能會自動部署到核心裝置。某些自動更新 (例如核心更新) 可能會導致裝置意外重新啟動。
> 若要避免對裝置上執行的元件進行意外更新，建議您在建立部署時直接包含該元件的偏好版本。如需有關 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱更新AWS IoT Greengrass核心軟件（OTA）。

主題

- 版本
- 作業系統
- 需求
- 相依性
- 下載與安裝
- 組態
- 本機記錄檔

- 變更記錄

版本

此元件具有下列版本：

- 2.12.x
- 2.11.x
- 2.10.x
- 2.9.x
- 2.8.x
- 2.7.x
- 2.6.x
- 2.5.x
- 2.4.x 版本
- 2.3.x 版本
- 2.2.x 版本
- 2.1.x
- 2.0.x

作業系統

此元件可安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

如需詳細資訊，請參閱 支援平台。

需求

設備必須滿足某些要求才能安裝和運行 Greengrass 核心和核心軟件。 AWS IoT Greengrass 如需詳細資訊，請參閱 裝置要求。

支援在 VPC 中執行的 Greengrass 核元件。若要在 VPC 中部署此元件，需要下列項目。

- Greengrass 核元件必須具有與 AWS IoT data、 AWS IoT 登入資料和 Amazon S3 的連線能力。

相依性

Greengrass 核不包括任何組件依賴關係。但是，幾個 AWS提供的組件包括細胞核作為依賴關係。如需詳細資訊，請參閱 AWS-提供的組件。

如需有關元件相依性的詳細資訊，請參閱元件方案參考。

下載與安裝

您可以下載在設備上設置 Greengrass 核組件的安裝程序。此安裝程序將您的設備設置為 Greengrass 核心設備。您可以執行兩種類型的安裝：為您建立必要 AWS 資源的快速安裝，或是您自行建立 AWS 資源的手動安裝。如需詳細資訊，請參閱 安裝 AWS IoT Greengrass 核心軟體。

您還可以按照教程安裝 Greengrass 核並探索 Greengrass 組件的開發。如需詳細資訊，請參閱 教學課程：AWS IoT Greengrass V2 入門。

組態

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。某些參數需要重新啟動 AWS IoT Greengrass Core 軟體才能生效。如需設定此元件的原因和方式的詳細資訊，請參閱設定 AWS IoT Greengrass 核心軟體。

`iotRoleAlias`

指向權杖交換 IAM AWS IoT 角色的角色別名。 AWS IoT 認證提供者會擔任此角色，以允許 Greengrass 核心裝置與服務互動。 AWS 如需詳細資訊，請參閱 授權核心設備與AWS服務。

當您使用`--provision true`選項執行 AWS IoT Greengrass Core 軟體時，軟體會提供角色別名，並在核心元件中設定其值。

`interpolateComponentConfiguration`

(選擇性) 您可以啟用 Greengrass 核來插入零組件模型組態中的零組件方案變數，並合併模型組態更新。我們建議您將此選項設定為，以`true`便核心裝置可以執行在其組態中使用 recipe 變數的 Greengrass 元件。

此功能適用於此元件的 v2.6.0 及更新版本。

預設：`false`

`networkProxy`

(選擇性) 用於所有連線的網路 Proxy。如需詳細資訊，請參閱 連線至連接埠 443 或透過網路代理。

> **Important**
>
> 當您將變更部署到此組態參數時， AWS IoT Greengrass 核心軟體會重新啟動，變更才會生效。

此物件包含下列資訊：

`noProxyAddresses`

(選擇性) 以逗號分隔的 IP 位址或主機名稱清單 (不包括 Proxy)。

`proxy`

要連線的代理伺服器。此物件包含下列資訊：

`url`

代理伺服器的格式 URL `scheme://userinfo@host:port`。

- `scheme`— 該計劃，必須是`http`或`https`。

  > **Important**
  >
  > 核心裝置必須執行 Greengrass 核心 v2.5.0 或更新版本才能使用 HTTPS 代理伺服器。
  > 如果您設定 HTTPS 代理，則必須將代理伺服器 CA 憑證新增至核心裝置的 Amazon 根 CA 憑證。如需詳細資訊，請參閱 讓核心裝置信任 HTTPS 代理。

- `userinfo`— (選用) 使用者名稱和密碼資訊。如果您在中指定此資訊`url`，Greengrass 核心裝置會忽略和欄位`username`。`password`
- `host`— 代理伺服器的主機名稱或 IP 位址。
- `port`— (選擇性) 連接埠號碼。如果您未指定連接埠，則 Greengrass 核心裝置會使用下列預設值：
  - `http`— 80
  - `https`— 443

`username`

(選擇性) 驗證 Proxy 伺服器的使用者名稱。

`password`

(選擇性) 驗證 Proxy 伺服器的密碼。

`mqtt`

（可選）Greengrass 核心設備的 MQTT 配置。如需詳細資訊，請參閱 連線至連接埠 443 或透過網路代理。

> **Important**
>
> 當您將變更部署到此組態參數時， AWS IoT Greengrass 核心軟體會重新啟動，變更才會生效。

此物件包含下列資訊：

`port`

(選擇性) 用於 MQTT 連線的連接埠。

預設：`8883`

`keepAliveTimeoutMs`

(選擇性) 用戶端傳送以維持 MQTT 連線作用中的每`PING`則訊息之間的時間 (毫秒)。此值必須大於`pingTimeoutMs`。

預設值：`60000`（60 秒）

`pingTimeoutMs`

(選擇性) 用戶端等待從伺服器接收`PINGACK`訊息的時間 (毫秒)。如果等待超過逾時，核心裝置會關閉並重新開啟 MQTT 連線。此值必須小於`keepAliveTimeoutMs`。

預設值：`30000`（30 秒）

`operationTimeoutMs`

(選擇性) 用戶端等待 MQTT 作業 (例如`CONNECT`或`PUBLISH`) 完成的時間 (毫秒)。此選項不適用於 MQTT `PING` 或保持有效訊息。

預設值：30000（30 秒）

`maxInFlightPublishes`

(選擇性) 可同時傳送中未確認的 MQTT QoS 1 訊息數目上限。

此功能適用於此元件的 v2.1.0 及更新版本。

預設：5

有效範圍：最大值為 100

`maxMessageSizeInBytes`

(選擇性) MQTT 郵件的大小上限。如果消息超過此大小，Greengrass 核會拒絕帶有錯誤的消息。

此功能適用於此元件的 v2.1.0 及更新版本。

預設值：131072

有效範圍：最大值2621440（2.5 MB）

`maxPublishRetry`

(選擇性) 重試失敗發佈之郵件的次數上限。您可以指-1定無限次重試。

此功能適用於此元件的 v2.1.0 及更新版本。

預設：100

`spooler`

(選擇性) Greengrass 核心裝置的 MQTT 多工緩衝處理程式組態。此物件包含下列資訊：

`storageType`

用於存儲消息的存儲類型。如果storageType設定為Disk，則pluginName可以設定。您可指定為 Memory 或 Disk。

此功能適用於 v2.11.0 及更高版本的綠核組件。

> **Important**
>
> 如果 MQTT 多工緩衝處理程式設定storageType為，Disk且您想要將 Greengrass 核心從 2.11.x 版降級為較早的版本，您必須將組態變更回。MemoryGreengrass 核 2.10.x 版本及更早版本支持的唯一配置是。storageType Memory不遵循此指南可

能會導致多工緩衝處理程式中斷。這會導致您的 Greengrass 核心裝置無法將 MQTT 訊息傳送到. AWS 雲端

預設：Memory

`pluginName`

(選擇性) 外掛程式元件名稱。只有當設定為時，才會使用此元件`Disk`。`storageType`此選項預設為，`aws.greengrass.DiskSpooler`並將使用 Greeng 磁碟後臺解決程式 rass 提供的。

此功能適用於 v2.11.0 及更高版本的綠核組件。

預設：`"aws.greengrass.DiskSpooler"`

`maxSizeInBytes`

(選擇性) 核心裝置將未處理的 MQTT 訊息儲存在記憶體中的快取大小上限。如果快取已滿，則會拒絕新郵件。

預設值：`2621440`(2.5 MB)

`keepQos0WhenOffline`

(選擇性) 您可以多工緩衝處理核心裝置離線時收到的 MQTT QoS 0 訊息。如果您將此選項設定為`true`，核心裝置會多工緩衝處理在離線時無法傳送的 QoS 0 訊息。如果您將此選項設定為`false`，核心裝置會捨棄這些訊息。核心裝置一律會多工緩衝處理 QoS 1 訊息，除非多工緩衝處理器已滿。

預設：`false`

`version`

(選擇性) MQTT 的版本。您可指定為 `mqtt3` 或 `mqtt5`。

此功能適用於 v2.10.0 及更高版本的綠核組件。

預設：`mqtt5`

`receiveMaximum`

(選擇性) 代理程式可傳送的未確認 QoS1 封包數目上限。

此功能適用於 v2.10.0 及更高版本的綠核組件。

預設：100

`sessionExpirySeconds`

(選擇性) 您可以要求從 IoT Core 持續工作階段的時間 (以秒為單位)。預設值為支援的最長時間 AWS IoT Core。

此功能適用於 v2.10.0 及更高版本的綠核組件。

預設：604800 (7 days)

`minimumReconnectDelaySeconds`

(選擇性) 重新連線行為的選項。MQTT 重新連線的最短時間 (以秒為單位)。

此功能適用於 v2.10.0 及更高版本的綠核組件。

預設：1

`maximumReconnectDelaySeconds`

(選擇性) 重新連線行為的選項。MQTT 重新連線的時間上限 (以秒為單位)。

此功能適用於 v2.10.0 及更高版本的綠核組件。

預設：120

`minimumConnectedTimeBeforeRetryResetSeconds`

(選擇性) 重新連線行為的選項。在重試延遲重設回最小值之前，連線必須處於作用中狀態的時間 (秒)。

此功能適用於 v2.10.0 及更高版本的綠核組件。

預設：30

`jvmOptions`

(選擇性) 用來執行 AWS IoT Greengrass 核心軟體的 JVM 選項。如需執行 AWS IoT Greengrass Core 軟體之建議 JVM 選項的詳細資訊，請參閱使用 JVM 選項控制內存分配。

> **Important**
>
> 當您將變更部署到此組態參數時， AWS IoT Greengrass 核心軟體會重新啟動，變更才會生效。

`iotDataEndpoint`

AWS IoT 您的 AWS 帳戶.

當您使用`--provision true`選項執行 AWS IoT Greengrass Core 軟體時，軟體會從中取得資料和認證端點， AWS IoT 並將它們設定在核心元件中。

`iotCredEndpoint`

AWS IoT 您的 AWS 帳戶.

當您使用`--provision true`選項執行 AWS IoT Greengrass Core 軟體時，軟體會從中取得資料和認證端點， AWS IoT 並將它們設定在核心元件中。

`greengrassDataPlaneEndpoint`

此功能在 2.7.0 版及更高版本中提供此功能。

如需詳細資訊，請參閱 使用私有 CA 簽署的裝置憑證。

`greengrassDataPlanePort`

此功能可在此元件的 v2.0.4 及更新版本中使用。

(選擇性) 用於資料平面連線的連接埠。如需詳細資訊，請參閱 連線至連接埠 443 或透過網路代理。

> **Important**
>
> 您必須指定裝置可以進行輸出連線的連接埠。如果您指定封鎖的連接埠，裝置將無法連線 AWS IoT Greengrass 至接收部署。

您可以從以下選項中選擇：

- `443`
- `8443`

預設：`8443`

`awsRegion`

要 AWS 區域 使用的。

`runWithDefault`

用來執行元件的系統使用者。

> **Important**
>
> 當您將變更部署到此組態參數時， AWS IoT Greengrass 核心軟體會重新啟動，變更才會生效。

此物件包含下列資訊：

`posixUser`

系統使用者的名稱或 ID，以及核心裝置用來執行一般和 Lambda 元件的系統群組 (選擇性)。以下列格式指定使用者和群組，並以冒號 (`:`) 分隔：`user:group`。群組為選用項目。如果您未指定群組， AWS IoT Greengrass Core 軟體會使用使用者的主要群組。例如，您可以指定 `ggc_user` 或 `ggc_user:ggc_group`。如需詳細資訊，請參閱 設定執行元件的使用者。

當您使用`--component-default-user` *`ggc_user`*`:`*`ggc_group`*選項執行 AWS IoT Greengrass Core 軟體安裝程式時，軟體會在核心元件中設定此參數。

`windowsUser`

此功能在此元件的 v2.5.0 及更新版本中提供。

用來在 Windows 核心裝置上執行此元件的 Windows 使用者名稱。使用者必須存在於每個 Windows 核心裝置上，且其名稱和密碼必須儲存在 LocalSystem 帳戶的認證管理員執行個體中。如需詳細資訊，請參閱 設定執行元件的使用者。

當您使用`--component-default-user` *`ggc_user`*選項執行 AWS IoT Greengrass Core 軟體安裝程式時，軟體會在核心元件中設定此參數。

`systemResourceLimits`

此功能可在此元件的 v2.4.0 及更新版本中使用。 AWS IoT Greengrass 目前在 Windows 核心裝置上不支援此功能。

依預設，系統資源限制會套用至一般和非容器化 Lambda 元件程序。您可以在建立部署時覆寫個別元件的系統資源限制。如需詳細資訊，請參閱 設定元件的系統資源限制。

此物件包含下列資訊：

`cpus`

每個元件的處理序可以在核心裝置上使用的 CPU 時間上限。核心裝置的 CPU 總時間等於裝置的 CPU 核心數。例如，在具有 4 個 CPU 核心的核心裝置上，您可以將此值設定2為將每

個元件的處理序限制為每個 CPU 核心的 50% 使用率。在具有 1 個 CPU 核心的裝置上，您可以將此值設定`0.25`為將每個元件的處理序限制為 CPU 使用率 25%。如果您將此值設定為大於 CPU 核心數目的數字， AWS IoT Greengrass 核心軟體就不會限制元件的 CPU 使用率。

`memory`

每個元件的處理序可在核心裝置上使用的最大 RAM 容量 (以 KB 為單位)。

`s3EndpointType`

(選擇性) S3 端點類型。此參數只會在美國東部 (維吉尼亞北部) (`us-east-1`) 區域生效。從任何其他區域設定此參數將會被忽略。您可以從以下選項中選擇：

- `REGIONAL`— S3 用戶端和預先簽署的 URL 使用區域端點。
- `GLOBAL`— S3 用戶端和預先簽署的 URL 使用舊版端點。

預設：`GLOBAL`

`logging`

(選擇性) 核心裝置的記錄組態。如需有關如何設定和使用 Greengrass 記錄檔的詳細資訊，請參閱。監控AWS IoT Greengrass日誌

此物件包含下列資訊：

`level`

(選擇性) 要輸出的最低記錄訊息層級。

從下列記錄層級中進行選擇，依照層級順序列出：

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

預設：`INFO`

`format`

(選擇性) 記錄檔的資料格式。您可以從以下選項中選擇：

- `TEXT`— 如果要以文字形式檢視記錄，請選擇此選項。

- JSON— 如果您想要使用 Greengrass CLI 記錄檔命令檢視記錄檔，或以程式設計方式與記錄檔互動，請選擇此選項。

預設：TEXT

`outputType`

(選擇性) 記錄檔的輸出類型。您可以從以下選項中選擇：

- FILE— AWS IoT Greengrass 核心軟件將日誌輸出到您在中指定的目錄中的文件outputDirectory。
- CONSOLE— AWS IoT Greengrass 核心軟件將日誌打印到stdout. 選擇此選項可在核心裝置列印記錄時檢視記錄檔。

預設：FILE

`fileSizeKB`

(選擇性) 每個記錄檔的大小上限 (以 KB 為單位)。記錄檔超過這個檔案大小上限之後， AWS IoT Greengrass Core 軟體就會建立新的記錄檔。

只有當您指定FILE為時，才會套用此參數outputType。

預設：1024

`totalLogsSizeKB`

(選擇性) 每個元件 (包括 Greengrass 核心) 的記錄檔總大小上限 (以 KB 為單位)。Greengrass 核的日誌文件還包括來自插件組件的日誌。當元件的記錄檔總大小超過此大小上限之後， AWS IoT Greengrass 核心軟體就會刪除該元件最舊的記錄檔。

此參數等於記錄管理員元件的磁碟空間限制參數 (diskSpaceLimit)，您可以為 Greengrass 核心 (系統) 和每個元件指定這個參數。 AWS IoT Greengrass 核心軟體使用兩個值中的最小值做為 Greengrass 核心和每個元件的最大總記錄大小。

只有當您指定FILE為時，才會套用此參數outputType。

預設：10240

`outputDirectory`

(選擇性) 記錄檔的輸出目錄。

只有當您指定FILE為時，才會套用此參數outputType。

預設值：*/greengrass/v2*/logs，其中*/greengrass/v2*是 AWS IoT Greengrass 根資料夾。

`fleetstatus`

此參數可在此元件的 v2.1.0 及更新版本中使用。

(選擇性) 核心裝置的叢集狀態設定。

此物件包含下列資訊：

`periodicStatusPublishIntervalSeconds`

(選擇性) 核心裝置將裝置狀態發佈至之間的時間長度 (秒) AWS 雲端。

最低限度:86400(24 小時)

預設值：86400(24 小時)

`telemetry`

(選擇性) 核心裝置的系統健康情況遙測組態。如需遙測指標以及如何處理遙測資料的詳細資訊，請參閱從AWS IoT Greengrass核心裝置收集系統健康狀態遙測資料。

此物件包含下列資訊：

`enabled`

(選擇性) 您可以啟用或停用遙測。

預設：`true`

`periodicAggregateMetricsIntervalSeconds`

(選擇性) 核心裝置彙總指標的間隔 (以秒為單位)。

如果您將此值設定為低於支援的最小值，則核心會改用預設值。

下限：`3600`

預設：`3600`

`periodicPublishMetricsIntervalSeconds`

(選擇性) 核心裝置將遙測度量發佈到 AWS 雲端.

如果您將此值設定為低於支援的最小值，則核心會改用預設值。

下限：86400

預設：86400

`deploymentPollingFrequencySeconds`

(選擇性) 輪詢部署通知的期間 (以秒為單位)。

預設：15

`componentStoreMaxSizeBytes`

(選擇性) 元件儲存區磁碟上的大小上限，其中包含元件配方和成品。

預設值：10000000000(10 GB)

`platformOverride`

(選擇性) 識別核心裝置平台的屬性字典。使用此選項可定義自訂平台屬性，元件配方可用來識別元件的正確生命週期和成品。例如，您可以定義硬體功能屬性，以僅部署最小的一組成品供元件執行。如需詳細資訊，請參閱元件方案中的資訊清單平台參數。

您也可以使用此參數來覆寫核心裝置的`os`和`architecture`平台屬性。

`httpClient`

此參數可在此元件的 v2.5.0 及更新版本中使用。

(選擇性) 核心裝置的 HTTP 用戶端組態。這些組態選項適用於此元件所發出的所有 HTTP 要求。如果核心裝置在較慢的網路上執行，您可以增加這些逾時持續時間，以防止 HTTP 要求逾時。

此物件包含下列資訊：

`connectionTimeoutMs`

(選擇性) 在連線要求逾時之前等待連線開啟的時間量 (毫秒)。

預設值：2000(2 秒)

`socketTimeoutMs`

(選擇性) 在連線逾時之前，透過開啟的連線等待資料傳輸的時間量 (毫秒)。

預設值：30000（30 秒）

Example 範例：組態合併更新

```
{
  "iotRoleAlias": "GreengrassCoreTokenExchangeRoleAlias",
  "networkProxy": {
    "noProxyAddresses": "http://192.168.0.1,www.example.com",
    "proxy": {
      "url": "http://my-proxy-server:1100",
      "username": "Mary_Major",
      "password": "pass@word1357"
    }
  },
  "mqtt": {
    "port": 443
  },
  "greengrassDataPlanePort": 443,
  "jvmOptions": "-Xmx64m",
  "runWithDefault": {
    "posixUser": "ggc_user:ggc_group"
  }
}
```

本機記錄檔

此元件會使用下列記錄檔。

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

若要檢視此元件的記錄

- 在核心裝置上執行下列命令，即時檢視此元件的記錄檔。以 AWS IoT Greengrass 根資料夾的路徑取代*/greengrass/v2*或 *C:\greengrass\v2*。

Linux

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
```

變更記錄

下表說明元件每個版本的變更。

版本	變更
2.12.4	錯誤修復和改進 • 修正在某些 Linux 裝置上啟動時，核心會進入鎖死狀態的問題。
2.12.3	**Warning** 此版本已不再提供。此版本中的改進功能在此元件的更新版本中提供。 錯誤修復和改進 • 修復了核重新啟動和元件恢復期間，原子核無法報告正確組件狀態的問題。 • 一般錯誤修正與改進。
2.12.2	錯誤修復和改進 • 修正舊記錄未正確清理的問題。 • 一般錯誤修正與改進。
2.12.1	錯誤修復和改進 • 修正核心可能會將 MQTT 訂閱複製到部署主題，導致其他記錄和 MQTT 發佈的問題。

版本	變更
2.12.0	新功能 • 可讓您執行啟動程序生命週期步驟，做為復原部署的一部分。
2.11.3	錯誤修復和改進 • 修正核心中的問題，即當元件的相依性失敗時，可能無法正確啟動元件。 新功能 • 添加可配置的 s3 端點類型。
2.11.2	錯誤修復和改進 • 修正核心 MQTT 5 用戶端中的問題，在使用大量 (> 50) 訂閱時，該用戶端可能會顯示為離線狀態。 • 添加 docker 撥號 TCP 失敗的重試。
2.11.1	錯誤修復和改進 • 修正啟動程序工作失敗且部署中繼資料檔案損毀時，核無法啟動的問題。 • 修正部署狀態更新中未報告隨選 Lambda 元件的問題。 • 添加對重複授權策略 ID 的支援。
2.11.0	新功能 • 可讓您取消本機部署。 • 可讓您設定本機部署的失敗處理原則。 • 添加對磁盤後台處理程序插件的支持。
2.10.3	錯誤修復和改進 • 修正使用 PKCS #11 提供者時 Greengrass 未訂閱部署通知的問題。
2.10.2	錯誤修復和改進 • 允許不區分大小寫的元件生命週期剖析。 • 修正無法正確重新建立環境 PATH 變數的問題。 • 修復了組件的代理 URI 編碼，包括具有特殊字符的用戶名的流管理器。

版本	變更
2.10.1	錯誤修復和改進 • 修正了在某些 ARMv8 處理器（包括 Jetson Nano）上啟動時可能導致當機的問題。 • Greengrass 不再關閉中的組件的標準，這會將行為恢復為 2.10.0 之前的行為
2.10.0	新功能 • 添加`interpolateComponentConfiguration` 對空的正則表達式的支持。Greengrass 現在從根配置對象進行插值。 • 添加對 MQTT5 的支持。 • 添加了一種機制，無需掃描即可快速加載插件組件。 • 啟用 Greengrass 通過刪除未使用的 Docker 映像來節省磁盤空間。 錯誤修復和改進 • 修正復原會保留部署中某些組態值的問題。 • 修正 Greengrass 核心驗證自訂非認證和資料端點中 AWS 網域序列的問題。AWS • 更新多群組相依性解析，以透過 AWS 雲端 交涉重新解析所有群組相依性，而不是鎖定至使用中版本。此更新也會移除部署錯誤程式碼`INSTALLED_COMPONENT_NOT_FOUND` 。 • 更新 Greengrass 核心，以便在本機已經存在 Docker 映像檔時略過下載。 • 更新 Greengrass 核心，以在逾時到期之前重新啟動元件安裝步驟。 • 其他小修正和改進。
2.9.6	錯誤修復和改進 • 修正 Greengrass 部署失敗，並顯示錯誤發生錯誤，且後續裝置重新開機無法啟動 Greengrass 的問題。當您在需要 Greengrass 重新啟動的部署的多個物件群組之間移動 Greengrass 裝置時，可能會發生此錯誤。

版本	變更
2.9.5	新功能 • 添加對 Greengrass 核軟體簽名驗證的支援。 錯誤修復和改進 • 修正當本機方案中繼資料區域與 Greengrass 核心啟動區域不相符時，部署失敗的問題。發生這種情況時，Greengrass 核現在會與雲端重新協商。 • 修正 MQTT 訊息多工緩衝處理程式填滿且永不移除訊息的問題。 • 其他小修正和改進。
2.9.4	錯誤修復和改進 • 在丟棄 QOS 0 訊息之前檢查是否有空訊息。 • 如果工作狀態詳細資料值超過 1024 個字元的限制，則截斷它們。 • 如果該路徑包含空格，則更新 Windows 的啟動程序檔，以正確讀取 Greengrass 根路徑。 • 訂閱的更新，以 AWS IoT Core 便在未傳送訂閱回應時捨棄用戶端訊息。 • 確保當主配置文件損壞或丟失時，核從備份文件加載其配置。
2.9.3	錯誤修復和改進 • 確保 MQTT 用戶端 ID 不會重複。 • 添加更強大的文件讀取和寫入，以避免損壞和恢復。 • 在特定的網絡相關錯誤上重試 docker 映像提取。 • 添加 MQTT 連接的`noProxyAddresses` 選項。
2.9.2	錯誤修復和改進 • 修正設定`interpolateComponentConfiguration` 不適用於進行中部署的問題。 • 使用 OSHI 列出所有子進程。
2.9.1	錯誤修復和改進 • 添加修復，如果部署刪除插件組件，Greengrass 重新啟動。

版本	變更
2.9.0	新功能 • 新增建立子部署的功能，以便使用較小的裝置子集重試部署。此功能可提供更有效率的方式來測試和解決失敗的部署。 錯誤修復和改進 • 改善了對沒有useradd、groupadd和的系統的支援usermod。 • 其他小修正和改進。
2.8.1	錯誤修復和改進 • 修正無法從 Greengrass API 錯誤正確產生部署錯誤碼的問題。 • 修正當元件在部署期間達到某個狀態時，叢集狀態更新會傳送不正確資訊的問題。ERRORED • 修正 Greengrass 擁有超過 50 個現有訂閱時，部署無法完成的問題。
2.8.0	新功能 • 更新 Greengrass 核心以報告部署健康狀態回應，其中包含詳細的錯誤碼，當將元件部署到核心裝置時發生問題。如需詳細資訊，請參閱 詳細的部署錯誤代碼。 • 更新 Greengrass 核心以報告元件健康狀態回應，其中包含元件進入或狀態時的詳細錯誤代碼。BROKEN ERRORED如需詳細資訊，請參閱 詳細的元件狀態代碼。 • 擴展狀態訊息欄位，以改善裝置的雲端可用性資訊。 • 提高車隊狀態服務的穩健性。 錯誤修復和改進 • 允許損毀的元件在組態變更時重新安裝。 • 修正啟動程序部署期間核心重新啟動會導致部署失敗的問題。 • 修正 Windows 中當根路徑包含空格時安裝失敗的問題。 • 修正部署期間關閉元件會使用新版本的關機指令碼的問題。 • 各種關機改進。 • 其他小修正和改進。

版本	變更
2.7.0	新功能 • 更新 Greengrass 核心核心，以便在核心裝置套用本機部 AWS IoT Greengrass 署時，將狀態更新傳送至雲端。 • 添加對由未註冊 CA 的自定義證書頒發機構（CA）簽名的客戶端證書的支持 AWS IoT。若要使用此功能，您可以將新的組`greengrassDataPlaneEndpoint` 態選項設定為`iotdata`。如需詳細資訊，請參閱 使用私有 CA 簽署的裝置憑證。 錯誤修復和改進 • 修正當核停止或重新啟動時，Greengrass 核會在特定情況下復原部署的問題。核心現在會在核心重新啟動後繼續部署。 • 當您指定將軟體設定為系統服務時，請更新 Greengrass 安裝程式以遵守`--start`引數。 • 更新的行為，SubscribeToComponentUpdates以在核心更新元件的事件中設定部署 ID。 • 其他小修正和改進。

版本	變更
2.6.0	新功能 • 當您訂閱本機發佈/訂閱主題時，新增對 MQTT 萬用字元的支援。如需詳細資訊，請參閱 發佈/訂閱本地訊息 及 SubscribeToTopic。 • 添加對組件配置中的配方變量的支持，而不是 *component_dependency_name* :configuration: *json_pointer* recipe 變量。當您在方案中定義元件或在部署DefaultConfiguration 中設定元件時，您可以使用這些方法變數。若要啟用此功能，請將interpolateComponentConfiguration組態選項設定為true。如需詳細資訊，請參閱 配方變數 及 在合併更新中使用配方變數。 • 在進程間通信（IPC）授權策略中添加對通*配符的完全支持。您現在可以指定*資源字串中的字元，以符合任何字元組合。如需詳細資訊，請參閱 授權原則中的萬用字元。 • 添加對自定義組件的支持，以調用 Greengrass CLI 使用的 IPC 操作。您可以使用這些 IPC 作業來管理本機部署、檢視元件詳細資料，以及產生可用來登入本機除錯主控台的密碼。如需詳細資訊，請參閱 IPC：管理本機部署和元件。 錯誤修復和改進 • 修正了相依元件在某些情況下重新啟動其硬相依性或變更狀態時不會反應的問題。 • 改善部署失敗時核心裝置向 AWS IoT Greengrass 雲端服務報告的錯誤訊息。 • 修正當核心重新啟動時，Greengrass 核在特定案例中套用物件部署兩次的問題。 • 其他小修正和改進。如需詳細資訊，請參閱中的發行版本 GitHub。

版本	變更
2.5.6	新功能 • 添加對使用 ECC 密鑰的硬件安全模塊的支持。您可以使用硬體安全模組 (HSM) 來安全地儲存裝置的私密金鑰和憑證。如需詳細資訊，請參閱 硬體安全整合。 錯誤修復和改進 • 修正當您在特定案例中部署含有損毀安裝指令碼的元件時，部署永遠不會完成的問題。 • 改善啟動期間的效能。 • 其他小修正和改進。
2.5.5	新功能 • 新增元件的GG_ROOT_CA_PATH 環境變數，以便您可以存取自訂元件中的根憑證授權單位 (CA) 憑證。 錯誤修復和改進 • 添加對使用英語以外顯示語言的 Windows 設備的支持。 • 更新 Greengrass 核心剖析布林安裝程式引數的方式，以便您可以指定不含布林值的布林引數來指定值。true例如，您現在可以指定--provision 而不是使用自動資源佈建--provision true 進行安裝。 • 修正核心裝置在特定情況下佈建後未向 AWS IoT Greengrass 雲端服務回報狀態的問題。 • 其他小修正和改進。
2.5.4	錯誤修復和改進 • 一般錯誤修正與改進。
2.5.3	新功能 • 添加對硬件安全集成的支持。您可以使用硬體安全模組 (HSM) 來安全地儲存裝置的私密金鑰和憑證。如需詳細資訊，請參閱 硬體安全整合。 錯誤修復和改進 • 修正核心建立 MQTT 連線時執行階段例外狀況的問題。 AWS IoT Core

版本	變更
2.5.2	錯誤修復和改進 • 修復了 Greengrass 核心更新後，Windows 服務在您停止或重新啟動設備後無法再次啟動的問題。
2.5.1	**Warning** 此版本已不再提供。此版本中的改進功能在此元件的更新版本中提供。 錯誤修復和改進 • 添加對 Windows 上 Java 運行時環境（JRE）的 32 位版本的支持。 • 變更 AWS IoT 原則未授與`greengrass:ListThingGroupsForCoreDevice` 權限之核心裝置的物件群組移除行為。使用此版本時，部署會繼續進行、記錄警告，而且當您從物件群組移除核心裝置時，不會移除元件。如需詳細資訊，請參閱 將AWS IoT Greengrass元件部署到裝置。 • 修正 Greengrass 核可供 Greengrass 元件處理程序使用的系統環境變數問題。您現在可以重新啟動元件，使其使用最新的系統環境變數。

版本	變更
2.5.0	新功能 • 添加對運行 Windows 的核心設備的支持。 • 變更物件群組移除的行為。使用此版本，您可以從物件群組中移除核心裝置，以便在下次部署中解除安裝該物群組的元件。 由於此更改，核心設備的 AWS IoT 策略必須具有`greengrass:ListThingGroupsForCoreDevice` 權限。如果您使用 AWS IoT Greengrass Core 軟體安裝程式來佈建資源， AWS IoT 則允許`greengrass:*` 預設原則 (包括此權限)。如需詳細資訊，請參閱 AWS IoT Greengrass 的裝置身分驗證和授權。 • 添加對 HTTPS 代理配置的支持。如需詳細資訊，請參閱 連線至連接埠 443 或透過網路代理。 • 加入新的`windowsUser` 組態參數。您可以使用此參數來指定用來在 Windows 核心裝置上執行元件的預設使用者。如需詳細資訊，請參閱 設定執行元件的使用者。 • 新增可用來自訂 HTTP 要求逾時的新`httpClient` 組態選項，以改善慢速網路上的效能。如需詳細資訊，請參閱 httpClient 設定參數。 錯誤修復和改進 • 修復了引導生命週期選項，以從組件重新啟動核心設備。 • 添加對配方變量中連字符的支持。 • 修正隨選 Lambda 函數元件的 IPC 授權。 • 改進了日誌消息並將非關鍵日誌從級別更改`INFO`為`DEBUG`級別，因此日誌更有用。 • 從使用自動佈建安裝 AWS IoT Greengrass Core 軟體時，Greengrass 核心建立的預設`iot:DescribeCertificate` 權杖交換角色移除權限。Greengrass 核不使用此權限。 • 修正問題，如`iam:CreatePolicy` 果相同原則可使用，則自動佈建指令碼不需要`iam:GetPolicy` 權限。 • 其他小修正和改進。

版本	變更
2.4.0	新功能 • 添加對系統資源限制的支援。您可以設定每個元件的處理序可在核心裝置上使用的 CPU 和 RAM 使用量上限。如需詳細資訊，請參閱 設定元件的系統資源限制。 • 添加 IPC 操作以暫停和恢復組件。如需詳細資訊，請參閱 PauseComponent 及 ResumeComponent。 • 添加對佈建外掛程式的支援。您可以指定要在安裝期間執行的 JAR 檔案，以佈建 Greengrass 核心裝置所需的 AWS 資源。Greengrass 核包括一個接口，您可以實現該接口以開發自定義配置插件。如需詳細資訊，請參閱 使用自訂資源佈建安裝 AWS IoT Greengrass 核心軟體。 • 將選用`thing-name-policy` 引數新增至 AWS IoT Greengrass Core 軟體安裝程式。當您使用自動資源佈建來安裝 AWS IoT Greengrass Core 軟體時，您可以使用此選項來指定現有或自訂 AWS IoT 原則。 錯誤修復和改進 • 在啟動時更新記錄配置。這可修正啟動時未套用記錄設定的問題。 • 更新核子加載器符號鏈接，以在安裝期間指向 Greengrass 根文件夾中的組件存儲區。此更新可讓您刪除安裝 AWS IoT Greengrass Core 軟體時下載的 JAR 檔案和其他核心成品。 • 其他小修正和改進。如需詳細資訊，請參閱中的發行版本 GitHub。

版本	變更
2.3.0	新功能 • 新增對部署組態文件的支援，最高可達 10 MB，最大為 7 KB (針對目標物件的部署) 或 31 KB (針對目標物件群組的部署)。 若要使用此功能，核心裝置的 AWS IoT 原則必須允許`greengrass:GetDeploymentConfiguration` 權限。如果您使用 AWS IoT Greengrass Core 軟體安裝程式來佈建資源，則核心裝置的 AWS IoT 政策允許`greengrass:*` ，其中包括此權限。如需詳細資訊，請參閱 AWS IoT Greengrass 的裝置身分驗證和授權。 • 添加`iot:thingName` 配方變量。您可以使用此 recipe 變數來取得方案中核心裝置 AWS IoT 物件的名稱。如需詳細資訊，請參閱 配方變數。 錯誤修復和改進 • 其他小修正和改進。如需詳細資訊，請參閱中的發行版本 GitHub。
2.2.0	新功能 • 添加用於本地陰影管理的 IPC 操作。 錯誤修復和改進 • 減少 JAR 檔案的大小。 • 減少記憶體使用量。 • 修復了某些情況下未更新日誌配置的問題。 • 其他小修正和改進。如需詳細資訊，請參閱中的發行版本 GitHub。

版本	變更
2.1.0	新功能 • 支援從 Amazon ECR 中的私有儲存庫下載 Docker 映像檔。 • 新增下列參數以自訂核心裝置上的 MQTT 組態： • `maxInFlightPublishes` — 可同時進行中未確認的 MQTT QoS 1 訊息的最大數量。 • `maxPublishRetry` — 重試失敗發佈之郵件的次數上限。 • 新增`fleetstatusservice` 組態參數以設定核心裝置將裝置狀態發佈至的間隔 AWS 雲端。 • 其他小修正和改進。如需詳細資訊，請參閱中的發行版本 GitHub。 錯誤修復和改進 • 修正核心重新啟動時造成陰影部署重複的問題。 • 修正當核心發生服務載入例外狀況時造成當機的問題。 • 改善元件相依性解決方案，使包含循環相依性的部署失敗。 • 修正先前已從核心裝置移除外掛程式元件時，無法重新部署該元件的問題。 • 修正導致 Lambda 元件或以 root 身分執行之元件的`HOME`環境變數設定為*`/greengrass/v2`* `/work`目錄的問題。`HOME`變數現在已正確設定為執行元件之使用者的主目錄。 • 其他小修正和改進。如需詳細資訊，請參閱中的發行版本 GitHub。
2.0.5	錯誤修復和改進 • 下載 AWS提供的元件時，透過設定的網路 Proxy 正確路由傳送流量。 • 在 AWS 中國區域中使用正確的 Greengrass 資料平面端點。

版本	變更
2.0.4	新功能 • 透過連接埠 443 啟用 HTTPS 流量。您可以使用核心元件 2.0.4 版的新`greengrassDataPlanePort` 組態參數，將 HTTPS 通訊設定為透過連接埠 443 傳輸，而非預設連接埠 8443。如需詳細資訊，請參閱 透過連接埠 443 設定 HTTPS。 • 新增工作路徑 recipe 變數。您可以使用此 recipe 變數來取得元件工作資料夾的路徑，您可以使用這些路徑在元件及其相依性之間共用檔案。如需詳細資訊，請參閱工作路徑 recipe 變數。 錯誤修復和改進 • 如果角色政策已存在，則防止建立權杖交換 AWS Identity and Access Management (IAM) 角色政策。 由於此變更，安裝程式現在需要`iam:GetPolicy` 和執行`sts:GetCallerIdentity` 時使用`--provision true` 。如需詳細資訊，請參閱 安裝程式佈建資源的最低 IAM 政策。 • 正確處理尚未成功註冊之部署的取消。 • 更新組態，以在復原部署時移除具有較新時間戳記的舊項目。 • 其他小修正和改進。如需詳細資訊，請參閱中的發行版本 GitHub。
2.0.3	初始版本。

用戶端裝置驗證

用戶端裝置 auth 元件 (`aws.greengrass.clientdevices.Auth`) 會驗證用戶端裝置並授權用戶端裝置處理行動。

> **Note**
>
> 用戶端裝置是連線到 Greengrass 核心裝置以傳送 MQTT 訊息和資料進行處理的本機 IoT 裝置。如需詳細資訊，請參閱 與本機 IoT 裝置互動。

主題

- 版本
- Type
- 作業系統
- 要求
- 相依性
- 組態
- 本機記錄檔
- 變更記錄

版本

> **Note**
>
> 客戶端設備身份驗證版本 2.3.0 已經停產。強烈建議您升級至用戶端裝置驗證 2.3.1 或更新版本。

此元件具有下列版本：

- 2.4.x 版本
- 2.3.x 版本
- 2.2.x 版本
- 2.1.x
- 2.0.x

Type

這個組件是一個插件組件（`aws.greengrass.plugin`）。Greengrass 核在與核相同的 Java 虛擬機（JVM）中運行此組件。當您在核心裝置上變更此元件的版本時，核心會重新啟動。

此組件使用與 Greengrass 核相同的日誌文件。如需詳細資訊，請參閱 監控AWS IoT Greengrass日誌。

如需詳細資訊，請參閱 元件類型。

作業系統

此元件可安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件具有下列需求：

- Greengrass 服務角色必須與您的關聯 AWS 帳戶 並允許權限。`iot:DescribeCertificate`
- 核心裝置的 AWS IoT 策略必須允許以下權限：
  - `greengrass:GetConnectivityInfo`，其中資源包括執行此元件的核心裝置的 ARN
  - `greengrass:VerifyClientDeviceIoTCertificateAssociation`，其中資源包括連線至核心裝置的每個用戶端裝置的 Amazon 資源名稱 (ARN)
  - `greengrass:VerifyClientDeviceIdentity`
  - `greengrass:PutCertificateAuthorities`
  - `iot:Publish`，其中資源包括以下 MQTT 主題的 ARN：
    - `$aws/things/`*`coreDeviceThingName`*`*-gci/shadow/get`
  - `iot:Subscribe`，其中資源包括下列 MQTT 主題篩選器的 ARN：
    - `$aws/things/`*`coreDeviceThingName`*`*-gci/shadow/update/delta`
    - `$aws/things/`*`coreDeviceThingName`*`*-gci/shadow/get/accepted`
  - `iot:Receive`，其中資源包括下列 MQTT 主題的 ARN：
    - `$aws/things/`*`coreDeviceThingName`*`*-gci/shadow/update/delta`
    - `$aws/things/`*`coreDeviceThingName`*`*-gci/shadow/get/accepted`

  如需詳細資訊，請參閱 資料平面操作的AWS IoT 政策 及 支援用戶端裝置的最低AWS IoT原則。
- (選用) 若要使用離線驗證， AWS IoT Greengrass 服務使用的 AWS Identity and Access Management (IAM) 角色必須包含下列權限：
  - `greengrass:ListClientDevicesAssociatedWithCoreDevice`，以讓核心裝置列出用戶端以進行離線驗證。
- 支援用戶端裝置驗證元件在 VPC 中執行。若要在 VPC 中部署此元件，需要下列項目。
  - 用戶端裝置驗證元件必須具有與 AWS IoT data、 AWS IoT 登入資料和 Amazon S3 的連線能力。

端點和連接埠

除了基本作業所需的端點和連接埠之外，此元件還必須能夠對下列端點和連接埠執行輸出要求。如需詳細資訊，請參閱 允許裝置流量透過 Proxy 或防火牆。

端點	連線埠	必要	描述
`iot.region.amazonaws.com`	443	是	用來取得有關 AWS IoT 物件憑證的資訊。

相依性

部署元件時， AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的需求，才能成功部署元件。本節列出此元件之已發行版本的相依性，以及定義每個相依性之元件版本的語意版本條件約束。您也可以在AWS IoT Greengrass 主控台中檢視元件每個版本的相依性。在元件詳細資料頁面上，尋找 [相依性] 清單。

2.4.4

下表列出此元件 2.4.4 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.6.0	軟式

2.4.3

下表列出此元件 2.4.3 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.6.0	軟式

2.4.1 and 2.4.2

下表列出此元件 2.4.1 和 2.4.2 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.6.0	軟式

2.3.0 – 2.4.0

下表列出此元件 2.3.0 至 2.4.0 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.6.0	軟式

2.3.0

下表列出此元件 2.3.0 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.6.0	軟式

2.2.3

下表列出此元件 2.2.3 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.6.0	軟式

2.2.2

下表列出此元件 2.2.2 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.6.0	軟式

2.2.1

下表列出此元件 2.2.1 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.6.0	軟式

2.2.0

下表列出此元件 2.2.0 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.6.0	軟式

2.1.0

下表列出此元件 2.1.0 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.2.0	軟式

2.0.4

下表列出此元件 2.0.4 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.2.0	軟式

2.0.2 and 2.0.3

下表列出此元件 2.0.2 和 2.0.3 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.2.0	軟式

2.0.1

下表列出此元件 2.0.1 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.2.0	軟式

2.0.0

下表列出此元件 2.0.0 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.2.0	軟式

如需有關元件相依性的詳細資訊，請參閱元件方案參考。

組態

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

> **Note**
>
> 在用戶端向本機 MQTT 代理人發出訂閱要求時，會評估訂閱權限。如果用戶端現有的訂閱權限遭到撤銷，用戶端將無法再訂閱主題。但是，它將繼續接收來自任何先前訂閱的主題的消息。為了防止這種行為，應在撤銷訂閱權限後重新啟動本機 MQTT 代理程式，以強制重新授權用戶端。
>
> 對於 MQTT 5 代理程式 (EMQX) 元件，請更新`restartIdentifier`組態以重新啟動 MQTT 5 代理程式。如需詳細資訊，請參閱 MQTT 5 代理程式元件組態。

對於 MQTT 3.1.1 代理程式 (Moquette) 元件，當伺服器憑證變更強制用戶端重新授權時，它會每週重新啟動一次。您可以變更核心裝置的連線資訊 (IP 位址)，或進行部署以移除 Broker 元件，然後稍後再次部署，以強制重新啟動。

v2.4.5

`deviceGroups`

裝置群組是具有與核心裝置連線和通訊之權限的用戶端裝置群組。使用選取規則識別用戶端裝置群組，並定義用戶端裝置授權原則，以指定每個裝置群組的權限。

此物件包含下列資訊：

`formatVersion`

此配置物件的格式版本。

您可以從以下選項中選擇：

- `2021-03-05`

`definitions`

此核心裝置的裝置群組。每個定義都會指定一個選取規則，以評估用戶端裝置是否為群組的成員。每個定義也會指定要套用至符合選取規則的用戶端裝置的權限原則。如果用戶端裝置是多個裝置群組的成員，則該裝置的權限將由每個群組的權限原則組成。

此物件包含下列資訊：

`groupNameKey`

此裝置群組的名稱。以可協助您識別此裝置群組的名稱取*`groupNameKey`*代。

此物件包含下列資訊：

`selectionRule`

指定哪些用戶端裝置為此裝置群組成員的查詢。當用戶端裝置連線時，核心裝置會評估此選擇規則，以判斷用戶端裝置是否為此裝置群組的成員。如果用戶端裝置是成員，則核心裝置會使用此裝置群組的策略來授權用戶端裝置的處理行動。

每個選取規則至少包含一個選取規則子句，這是可以與用戶端裝置相符的單一運算式查詢。選取規則使用與 AWS IoT 叢集索引相同的查詢語法。如需有關選取規則語法的詳細資訊，請參閱AWS IoT Core 開發人員指南中的AWS IoT 叢集索引查詢語法。

使用*萬用字元來比對多個用戶端裝置與一個選取規則子句。您可以在物件名稱的開頭和結尾使用此萬用字元，以比對名稱開頭或結尾以您指定字串的用戶端裝置。您也可以使用此萬用字元來比對所有用戶端裝置。

> **Note**
>
> 若要選取包含冒號字元 (:) 的值，請使用反斜線字元 (\) 逸出冒號。在 JSON 等格式中，您必須逸出反斜線字元，因此您必須在冒號字元前輸入兩個反斜線字元。例如，指`thingName: MyTeam\\:ClientDevice1`定選取名稱為的物件`MyTeam:ClientDevice1`。

您可以指定下列選取器：

- `thingName`— 用戶端裝置 AWS IoT 物件的名稱。

Example 範例選取規則

下列選取規則符合名稱為`MyClientDevice1`或的用戶端裝置`MyClientDevice2`。

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example 範例選取規則 (使用萬用字元)

下列選取規則符合名稱開頭為的用戶端裝置`MyClientDevice`。

```
thingName: MyClientDevice*
```

Example 範例選取規則 (使用萬用字元)

下列選取規則符合名稱結尾為的用戶端裝置`MyClientDevice`。

```
thingName: *MyClientDevice
```

Example 範例選取規則 (符合所有裝置)

下列選取規則符合所有用戶端裝置。

```
thingName: *
```

policyName

適用於此裝置群組中用戶端裝置的權限原則。指定您在物件中定義的策略名policies稱。

policies

連線至核心裝置的用戶端裝置的用戶端裝置授權原則。每個授權原則都會指定一組處理行動以及用戶端裝置可以執行這些處理行動的資源。

此物件包含下列資訊：

policyNameKey

此授權原則的名稱。以可協助您識別此授權原則的名稱取*policyNameKey*代。您可以使用此原則名稱來定義要套用至裝置群組的原則。

此物件包含下列資訊：

statementNameKey

本政策聲明的名稱。請*statementNameKey*以可協助您識別此政策聲明的名稱取代。

此物件包含下列資訊：

operations

此原則中允許資源的作業清單。

您可以包括以下任何操作：

- mqtt:connect— 授予連接到核心設備的權限。用戶端裝置必須具有此權限才能連線至核心裝置。

  此作業支援下列資源：

  - mqtt:clientId:*deviceClientId*— 根據用戶端裝置用來連線至核心裝置的 MQTT 代理程式的用戶端 ID 來限制存取。取代*deviceClientId*為要使用的用戶端 ID。

- mqtt:publish— 授予將 MQTT 訊息發佈至主題的權限。

  此作業支援下列資源：

  - mqtt:topic:*mqttTopic*— 根據用戶端裝置發佈訊息的 MQTT 主題限制存取。將 *MQTT ##*取代為要使用的主題。

此資源不支援 MQTT 主題萬用字元。

- `mqtt:subscribe`— 授予訂閱 MQTT 主題篩選器以接收訊息的權限。

  此作業支援下列資源：

  - `mqtt:topicfilter:`*`mqttTopicFilter`*— 根據用戶端裝置可以訂閱訊息的 MQTT 主題限制存取。以要使*`mqttTopicFilter`*用的主題篩選器取代。

    此資源支援+和 # MQTT 主題萬用字元。如需詳細資訊，請參閱AWS IoT Core 開發人員指南中的 MQTT 主題。

    用戶端裝置可以訂閱您允許的確切主題篩選器。例如，如果您允許用戶端裝置訂閱`mqtt:topicfilter:client/+/status`資源，則用戶端裝置可以訂閱，`client/+/status`但無法訂閱`client/client1/status`。

  您可以指定*萬用字元以允許存取所有動作。

`resources`

此原則中允許作業的資源清單。指定與此策略中作業對應的資源。例如，您可以在指定作業的策略中指定 MQTT 主題資源 (`mqtt:topic:`*`mqttTopic`*) 清單。`mqtt:publish`

您可以指定*萬用字元以允許存取所有資源。您無法使用*萬用字元來比對部分資源識別碼。例如，您可以指定**`"resources": "*"`**，但無法指定**`"resources": "mqtt:clientId:*"`**。

`statementDescription`

(選擇性) 此政策聲明的說明。

`certificates`

(選擇性) 此核心裝置的憑證組態選項。此物件包含下列資訊：

`serverCertificateValiditySeconds`

(選擇性) 本機 MQTT 伺服器憑證到期後的時間 (秒)。您可以設定此選項來自訂用戶端裝置中斷連線並重新連線至核心裝置的頻率。

此元件會在到期前 24 小時輪換本機 MQTT 伺服器憑證。MQTT 代理程式 (例如 Moquette MQTT 代理程式元件) 會產生新憑證並重新啟動。發生這種情況時，連接到此核心裝置的所有用戶端裝置都會中斷連線。用戶端裝置可在短時間後重新連線至核心裝置。

預設值：604800(7 天)

最小值：172800（2 天）

最大值：864000（10 天）

`performance`

(選擇性) 此核心裝置的效能組態選項。此物件包含下列資訊：

`maxActiveAuthTokens`

(選擇性) 作用中用戶端裝置授權權杖的數目上限。您可以增加此數目，讓更多數目的用戶端裝置連線到單一核心裝置，而無需重新驗證。

預設：2500

`cloudRequestQueueSize`

(選擇性) 此元件拒絕 AWS 雲端 要求之前，要排入佇列的要求數目上限。

預設：100

`maxConcurrentCloudRequests`

(選擇性) 要傳送至的並行要求數目上限 AWS 雲端。您可以增加此數目，以改善連線大量用戶端裝置的核心裝置上的驗證效能。

預設：1

`certificateAuthority`

(選擇性) 憑證授權單位組態選項，以您自己的中繼憑證授權單位取代核心裝置中繼授權單位。

> **Note**
>
> 如果您使用自訂憑證授權單位 (CA) 設定 Greengrass 核心裝置，並使用相同的 CA 發行用戶端裝置憑證，則 Greengrass 會略過用戶端裝置 MQTT 作業的授權原則檢查。用戶端裝置驗證元件完全信任使用其設定為使用之 CA 簽署的憑證的用戶端。
> 若要在使用自訂 CA 時限制此行為，請使用不同的 CA 或中繼 CA 建立並簽署用戶端裝置，然後調整`certificateUri`和`certificateChainUri`欄位以指向正確的中繼 CA。

此物件包含下列資訊。

证书

憑證的位置。它可以是文件系統 URI 或指向存儲在硬件安全模塊中的證書的 URI。

certificateChainUri

核心裝置 CA 的憑證鏈結位置。這應該是回到根 CA 的完整憑證鏈結。它可以是一個文件系統 URI 或指向存儲在硬件安全模塊中的證書鏈的 URI。

privateKeyUri

核心裝置私密金鑰的位置。這可以是檔案系統 URI 或指向儲存在硬體安全性模組中的憑證私密金鑰的 URI。

security

(選擇性) 此核心裝置的安全性設定選項。此物件包含下列資訊。

clientDeviceTrustDurationMinutes

在需要與核心裝置重新驗證之前，可以信任用戶端裝置的驗證資訊的持續時間 (以分鐘為單位)。預設值為 1。

metrics

(選擇性) 此核心裝置的指標選項。只有在用戶端裝置驗證發生錯誤時，才會顯示錯誤指標。此物件包含下列資訊：

disableMetrics

如果該disableMetrics字段設置為true，則客戶端設備身份驗證將不會收集指標。

預設：false

aggregatePeriodSeconds

以秒為單位的彙總期間，決定用戶端裝置驗證彙總度量並將其傳送至遙測代理程式的頻率。這不會變更發佈指標的頻率，因為遙測代理程式仍會每天發佈一次。

預設：3600

startupTimeoutSeconds

(選擇性) 元件啟動的時間上限 (以秒為單位)。BROKEN如果超過此逾時，組件的狀態會變更為。

預設：120

Example 範例：組態合併更新 (使用限制性原則)

下列範例組態指定允許名稱開頭為的用戶端裝置MyClientDevice連線和發佈/訂閱所有主題。

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
    },
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
```

```
          ]
        }
      }
    }
  }
}
```

Example 範例：組態合併更新 (使用寬容原則)

下列範例設定指定允許所有用戶端裝置連線和發佈/訂閱所有主題。

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}
```

v2.4.2 - v2.4.4

`deviceGroups`

裝置群組是具有與核心裝置連線和通訊之權限的用戶端裝置群組。使用選取規則識別用戶端裝置群組，並定義用戶端裝置授權原則，以指定每個裝置群組的權限。

此物件包含下列資訊：

`formatVersion`

此配置物件的格式版本。

您可以從以下選項中選擇：

- `2021-03-05`

`definitions`

此核心裝置的裝置群組。每個定義都會指定一個選取規則，以評估用戶端裝置是否為群組的成員。每個定義也會指定要套用至符合選取規則的用戶端裝置的權限原則。如果用戶端裝置是多個裝置群組的成員，則該裝置的權限將由每個群組的權限原則組成。

此物件包含下列資訊：

`groupNameKey`

此裝置群組的名稱。以可協助您識別此裝置群組的名稱取*`groupNameKey`*代。

此物件包含下列資訊：

`selectionRule`

指定哪些用戶端裝置為此裝置群組成員的查詢。當用戶端裝置連線時，核心裝置會評估此選擇規則，以判斷用戶端裝置是否為此裝置群組的成員。如果用戶端裝置是成員，則核心裝置會使用此裝置群組的策略來授權用戶端裝置的處理行動。

每個選取規則至少包含一個選取規則子句，這是可以與用戶端裝置相符的單一運算式查詢。選取規則使用與 AWS IoT 叢集索引相同的查詢語法。如需有關選取規則語法的詳細資訊，請參閱AWS IoT Core 開發人員指南中的AWS IoT 叢集索引查詢語法。

使用*萬用字元來比對多個用戶端裝置與一個選取規則子句。您可以在物件名稱結尾使用此萬用字元，以符合名稱以您指定之字串開頭的用戶端裝置。您也可以使用此萬用字元來比對所有用戶端裝置。

> **Note**
>
> 若要選取包含冒號字元 (`:`) 的值，請使用反斜線字元 (`\\`) 逸出冒號。在 JSON 等格式中，您必須逸出反斜線字元，因此您必須在冒號字元前輸入兩個反斜線字元。例如，指`thingName: MyTeam\\\\:ClientDevice1`定選取名稱為的物件`MyTeam:ClientDevice1`。

您可以指定下列選取器：

- `thingName`— 用戶端裝置 AWS IoT 物件的名稱。

Example 範例選取規則

下列選取規則符合名稱為`MyClientDevice1`或的用戶端裝置`MyClientDevice2`。

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example 範例選取規則 (使用萬用字元)

下列選取規則符合名稱開頭為的用戶端裝置`MyClientDevice`。

```
thingName: MyClientDevice*
```

Example 範例選取規則 (符合所有裝置)

下列選取規則符合所有用戶端裝置。

```
thingName: *
```

`policyName`

適用於此裝置群組中用戶端裝置的權限原則。指定您在物件中定義的策略名`policies`稱。

`policies`

連線至核心裝置的用戶端裝置的用戶端裝置授權原則。每個授權原則都會指定一組處理行動以及用戶端裝置可以執行這些處理行動的資源。

此物件包含下列資訊：

`policyNameKey`

此授權原則的名稱。以可協助您識別此授權原則的名稱取*`policyNameKey`*代。您可以使用此原則名稱來定義要套用至裝置群組的原則。

此物件包含下列資訊：

statementNameKey

本政策聲明的名稱。請*statementNameKey*以可協助您識別此政策聲明的名稱取代。

此物件包含下列資訊：

operations

此原則中允許資源的作業清單。

您可以包括以下任何操作：

- mqtt:connect— 授予連接到核心設備的權限。用戶端裝置必須具有此權限才能連線至核心裝置。

  此作業支援下列資源：

  - mqtt:clientId:*deviceClientId*— 根據用戶端裝置用來連線至核心裝置的 MQTT 代理程式的用戶端 ID 來限制存取。取代*deviceClientId*為要使用的用戶端 ID。

- mqtt:publish— 授予將 MQTT 訊息發佈至主題的權限。

  此作業支援下列資源：

  - mqtt:topic:*mqttTopic*— 根據用戶端裝置發佈訊息的 MQTT 主題限制存取。將 *MQTT ##*取代為要使用的主題。

    此資源不支援 MQTT 主題萬用字元。

- mqtt:subscribe— 授予訂閱 MQTT 主題篩選器以接收訊息的權限。

  此作業支援下列資源：

  - mqtt:topicfilter:*mqttTopicFilter*— 根據用戶端裝置可以訂閱訊息的 MQTT 主題限制存取。以要使*mqttTopicFilter*用的主題篩選器取代。

    此資源支援+和 # MQTT 主題萬用字元。如需詳細資訊，請參閱AWS IoT Core 開發人員指南中的 MQTT 主題。

    用戶端裝置可以訂閱您允許的確切主題篩選器。例如，如果您允許用戶端裝置訂閱mqtt:topicfilter:client/+/status資源，則用戶端裝置可以訂閱，client/+/status但無法訂閱client/client1/status。

您可以指定*萬用字元以允許存取所有動作。

`resources`

此原則中允許作業的資源清單。指定與此策略中作業對應的資源。例如，您可以在指定作業的策略中指定 MQTT 主題資源 (`mqtt:topic:`*`mqttTopic`*) 清單。`mqtt:publish`

您可以指定`*`萬用字元以允許存取所有資源。您無法使用`*`萬用字元來比對部分資源識別碼。例如，您可以指定**`"resources": "*"`**，但無法指定**`"resources": "mqtt:clientId:*"`**。

`statementDescription`

(選擇性) 此政策聲明的說明。

`certificates`

(選擇性) 此核心裝置的憑證組態選項。此物件包含下列資訊：

`serverCertificateValiditySeconds`

(選擇性) 本機 MQTT 伺服器憑證到期後的時間 (秒)。您可以設定此選項來自訂用戶端裝置中斷連線並重新連線至核心裝置的頻率。

此元件會在到期前 24 小時輪換本機 MQTT 伺服器憑證。MQTT 代理程式 (例如 Moquette MQTT 代理程式元件) 會產生新憑證並重新啟動。發生這種情況時，連接到此核心裝置的所有用戶端裝置都會中斷連線。用戶端裝置可在短時間後重新連線至核心裝置。

預設值：`604800`(7 天)

最小值：`172800`（2 天）

最大值：`864000`（10 天）

`performance`

(選擇性) 此核心裝置的效能組態選項。此物件包含下列資訊：

`maxActiveAuthTokens`

(選擇性) 作用中用戶端裝置授權權杖的數目上限。您可以增加此數目，讓更多數目的用戶端裝置連線到單一核心裝置，而無需重新驗證。

預設：`2500`

`cloudRequestQueueSize`

(選擇性) 此元件拒絕 AWS 雲端 要求之前，要排入佇列的要求數目上限。

預設：100

`maxConcurrentCloudRequests`

(選擇性) 要傳送至的並行要求數目上限 AWS 雲端。您可以增加此數目，以改善連線大量用戶端裝置的核心裝置上的驗證效能。

預設：1

`certificateAuthority`

(選擇性) 憑證授權單位組態選項，以您自己的中繼憑證授權單位取代核心裝置中繼授權單位。

> **Note**
>
> 如果您使用自訂憑證授權單位 (CA) 設定 Greengrass 核心裝置，並使用相同的 CA 發行用戶端裝置憑證，則 Greengrass 會略過用戶端裝置 MQTT 作業的授權原則檢查。用戶端裝置驗證元件完全信任使用其設定為使用之 CA 簽署的憑證的用戶端。
> 若要在使用自訂 CA 時限制此行為，請使用不同的 CA 或中繼 CA 建立並簽署用戶端裝置，然後調整`certificateUri`和`certificateChainUri`欄位以指向正確的中繼 CA。

此物件包含下列資訊。

证书

憑證的位置。它可以是文件系統 URI 或指向存儲在硬件安全模塊中的證書的 URI。

`certificateChainUri`

核心裝置 CA 的憑證鏈結位置。這應該是回到根 CA 的完整憑證鏈結。它可以是一個文件系統 URI 或指向存儲在硬件安全模塊中的證書鏈的 URI。

`privateKeyUri`

核心裝置私密金鑰的位置。這可以是檔案系統 URI 或指向儲存在硬體安全性模組中的憑證私密金鑰的 URI。

`security`

(選擇性) 此核心裝置的安全性設定選項。此物件包含下列資訊。

clientDeviceTrustDurationMinutes

在需要與核心裝置重新驗證之前，可以信任用戶端裝置的驗證資訊的持續時間 (以分鐘為單位)。預設值為 1。

metrics

(選擇性) 此核心裝置的指標選項。只有在用戶端裝置驗證發生錯誤時，才會顯示錯誤指標。此物件包含下列資訊：

disableMetrics

如果該disableMetrics字段設置為true，則客戶端設備身份驗證將不會收集指標。

預設：false

aggregatePeriodSeconds

以秒為單位的彙總期間，決定用戶端裝置驗證彙總度量並將其傳送至遙測代理程式的頻率。這不會變更發佈指標的頻率，因為遙測代理程式仍會每天發佈一次。

預設：3600

startupTimeoutSeconds

(選擇性) 元件啟動的時間上限 (以秒為單位)。BROKEN如果超過此逾時，組件的狀態會變更為。

預設：120

Example 範例：組態合併更新 (使用限制性原則)

下列範例組態指定允許名稱開頭為的用戶端裝置MyClientDevice連線和發佈/訂閱所有主題。

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
    },
```

```
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
          ]
        }
      }
    }
  }
}
```

Example 範例：組態合併更新 (使用寬容原則)

下列範例設定指定允許所有用戶端裝置連線和發佈/訂閱所有主題。

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
```

```
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}
```

v2.4.0 - v2.4.1

`deviceGroups`

裝置群組是具有與核心裝置連線和通訊之權限的用戶端裝置群組。使用選取規則識別用戶端裝置群組，並定義用戶端裝置授權原則，以指定每個裝置群組的權限。

此物件包含下列資訊：

`formatVersion`

此配置物件的格式版本。

您可以從以下選項中選擇：

- `2021-03-05`

`definitions`

此核心裝置的裝置群組。每個定義都會指定一個選取規則，以評估用戶端裝置是否為群組的成員。每個定義也會指定要套用至符合選取規則的用戶端裝置的權限原則。如果用戶端裝置是多個裝置群組的成員，則該裝置的權限將由每個群組的權限原則組成。

此物件包含下列資訊：

groupNameKey

此裝置群組的名稱。以可協助您識別此裝置群組的名稱取*groupNameKey*代。

此物件包含下列資訊：

`selectionRule`

指定哪些用戶端裝置為此裝置群組成員的查詢。當用戶端裝置連線時，核心裝置會評估此選擇規則，以判斷用戶端裝置是否為此裝置群組的成員。如果用戶端裝置是成員，則核心裝置會使用此裝置群組的策略來授權用戶端裝置的處理行動。

每個選取規則至少包含一個選取規則子句，這是可以與用戶端裝置相符的單一運算式查詢。選取規則使用與 AWS IoT 叢集索引相同的查詢語法。如需有關選取規則語法的詳細資訊，請參閱AWS IoT Core 開發人員指南中的AWS IoT 叢集索引查詢語法。

使用*萬用字元來比對多個用戶端裝置與一個選取規則子句。您可以在物件名稱結尾使用此萬用字元，以符合名稱以您指定之字串開頭的用戶端裝置。您也可以使用此萬用字元來比對所有用戶端裝置。

> **Note**
>
> 若要選取包含冒號字元 (:) 的值，請使用反斜線字元 (\\) 逸出冒號。在 JSON 等格式中，您必須逸出反斜線字元，因此您必須在冒號字元前輸入兩個反斜線字元。例如，指`thingName: MyTeam\\\\:ClientDevice1`定選取名稱為的物件`MyTeam:ClientDevice1`。

您可以指定下列選取器：

- `thingName`— 用戶端裝置 AWS IoT 物件的名稱。

Example 範例選取規則

下列選取規則符合名稱為`MyClientDevice1`或的用戶端裝置`MyClientDevice2`。

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example 範例選取規則 (使用萬用字元)

下列選取規則符合名稱開頭為的用戶端裝置`MyClientDevice`。

```
thingName: MyClientDevice*
```

Example 範例選取規則 (符合所有裝置)

下列選取規則符合所有用戶端裝置。

```
thingName: *
```

`policyName`

適用於此裝置群組中用戶端裝置的權限原則。指定您在物件中定義的策略名`policies`稱。

`policies`

連線至核心裝置的用戶端裝置的用戶端裝置授權原則。每個授權原則都會指定一組處理行動以及用戶端裝置可以執行這些處理行動的資源。

此物件包含下列資訊：

`policyNameKey`

此授權原則的名稱。以可協助您識別此授權原則的名稱取*`policyNameKey`*代。您可以使用此原則名稱來定義要套用至裝置群組的原則。

此物件包含下列資訊：

`statementNameKey`

本政策聲明的名稱。請*`statementNameKey`*以可協助您識別此政策聲明的名稱取代。

此物件包含下列資訊：

`operations`

此原則中允許資源的作業清單。

您可以包括以下任何操作：

- `mqtt:connect`— 授予連接到核心設備的權限。用戶端裝置必須具有此權限才能連線至核心裝置。

  此作業支援下列資源：

- `mqtt:clientId:`*`deviceClientId`*— 根據用戶端裝置用來連線至核心裝置的 MQTT 代理程式的用戶端 ID 來限制存取。取代*`deviceClientId`*為要使用的用戶端 ID。
- `mqtt:publish`— 授予將 MQTT 訊息發佈至主題的權限。

  此作業支援下列資源：

  - `mqtt:topic:`*`mqttTopic`*— 根據用戶端裝置發佈訊息的 MQTT 主題限制存取。將 *`MQTT ##`*取代為要使用的主題。

    此資源不支援 MQTT 主題萬用字元。
- `mqtt:subscribe`— 授予訂閱 MQTT 主題篩選器以接收訊息的權限。

  此作業支援下列資源：

  - `mqtt:topicfilter:`*`mqttTopicFilter`*— 根據用戶端裝置可以訂閱訊息的 MQTT 主題限制存取。以要使*`mqttTopicFilter`*用的主題篩選器取代。

    此資源支援+和 # MQTT 主題萬用字元。如需詳細資訊，請參閱AWS IoT Core 開發人員指南中的 MQTT 主題。

    用戶端裝置可以訂閱您允許的確切主題篩選器。例如，如果您允許用戶端裝置訂閱`mqtt:topicfilter:client/+/status`資源，則用戶端裝置可以訂閱，`client/+/status`但無法訂閱`client/client1/status`。

您可以指定*萬用字元以允許存取所有動作。

`resources`

此原則中允許作業的資源清單。指定與此策略中作業對應的資源。例如，您可以在指定作業的策略中指定 MQTT 主題資源 (`mqtt:topic:`*`mqttTopic`*) 清單。`mqtt:publish`

您可以指定*萬用字元以允許存取所有資源。您無法使用*萬用字元來比對部分資源識別碼。例如，您可以指定**`"resources": "*"`**，但無法指定**`"resources": "mqtt:clientId:*"`**。

`statementDescription`

(選擇性) 此政策聲明的說明。

`certificates`

(選擇性) 此核心裝置的憑證組態選項。此物件包含下列資訊：

`serverCertificateValiditySeconds`

(選擇性) 本機 MQTT 伺服器憑證到期後的時間 (秒)。您可以設定此選項來自訂用戶端裝置中斷連線並重新連線至核心裝置的頻率。

此元件會在到期前 24 小時輪換本機 MQTT 伺服器憑證。MQTT 代理程式 (例如 Moquette MQTT 代理程式元件) 會產生新憑證並重新啟動。發生這種情況時，連接到此核心裝置的所有用戶端裝置都會中斷連線。用戶端裝置可在短時間後重新連線至核心裝置。

預設值：`604800`(7 天)

最小值：`172800`（2 天）

最大值：`864000`（10 天）

`performance`

(選擇性) 此核心裝置的效能組態選項。此物件包含下列資訊：

`maxActiveAuthTokens`

(選擇性) 作用中用戶端裝置授權權杖的數目上限。您可以增加此數目，讓更多數目的用戶端裝置連線到單一核心裝置，而無需重新驗證。

預設：`2500`

`cloudRequestQueueSize`

(選擇性) 此元件拒絕 AWS 雲端 要求之前，要排入佇列的要求數目上限。

預設：`100`

`maxConcurrentCloudRequests`

(選擇性) 要傳送至的並行要求數目上限 AWS 雲端。您可以增加此數目，以改善連線大量用戶端裝置的核心裝置上的驗證效能。

預設：`1`

`certificateAuthority`

(選擇性) 憑證授權單位組態選項，以您自己的中繼憑證授權單位取代核心裝置中繼授權單位。此物件包含下列資訊。

此物件包含下列資訊：

证书

憑證的位置。它可以是文件系統 URI 或指向存儲在硬件安全模塊中的證書的 URI。

certificateChainUri

核心裝置 CA 的憑證鏈結位置。這應該是回到根 CA 的完整憑證鏈結。它可以是一個文件系統 URI 或指向存儲在硬件安全模塊中的證書鏈的 URI。

privateKeyUri

核心裝置私密金鑰的位置。這可以是檔案系統 URI 或指向儲存在硬體安全性模組中的憑證私密金鑰的 URI。

security

(選擇性) 此核心裝置的安全性設定選項。此物件包含下列資訊。

clientDeviceTrustDurationMinutes

在需要與核心裝置重新驗證之前，可以信任用戶端裝置的驗證資訊的持續時間 (以分鐘為單位)。預設值為 1。

metrics

(選擇性) 此核心裝置的指標選項。只有在用戶端裝置驗證發生錯誤時，才會顯示錯誤指標。此物件包含下列資訊：

disableMetrics

如果該disableMetrics字段設置為true，則客戶端設備身份驗證將不會收集指標。

預設：false

aggregatePeriodSeconds

以秒為單位的彙總期間，決定用戶端裝置驗證彙總度量並將其傳送至遙測代理程式的頻率。這不會變更發佈指標的頻率，因為遙測代理程式仍會每天發佈一次。

預設：3600

Example 範例：組態合併更新 (使用限制性原則)

下列範例組態指定允許名稱開頭為的用戶端裝置MyClientDevice連線和發佈/訂閱所有主題。

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
    },
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
          ]
        }
      }
    }
  }
}
```

Example 範例：組態合併更新 (使用寬容原則)

下列範例設定指定允許所有用戶端裝置連線和發佈/訂閱所有主題。

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}
```

v2.3.x

`deviceGroups`

裝置群組是具有與核心裝置連線和通訊之權限的用戶端裝置群組。使用選取規則識別用戶端裝置群組，並定義用戶端裝置授權原則，以指定每個裝置群組的權限。

此物件包含下列資訊：

`formatVersion`

此配置物件的格式版本。

您可以從以下選項中選擇：

- `2021-03-05`

`definitions`

此核心裝置的裝置群組。每個定義都會指定一個選取規則，以評估用戶端裝置是否為群組的成員。每個定義也會指定要套用至符合選取規則的用戶端裝置的權限原則。如果用戶端裝置是多個裝置群組的成員，則該裝置的權限將由每個群組的權限原則組成。

此物件包含下列資訊：

`groupNameKey`

此裝置群組的名稱。以可協助您識別此裝置群組的名稱取*`groupNameKey`*代。

此物件包含下列資訊：

`selectionRule`

指定哪些用戶端裝置為此裝置群組成員的查詢。當用戶端裝置連線時，核心裝置會評估此選擇規則，以判斷用戶端裝置是否為此裝置群組的成員。如果用戶端裝置是成員，則核心裝置會使用此裝置群組的策略來授權用戶端裝置的處理行動。

每個選取規則至少包含一個選取規則子句，這是可以與用戶端裝置相符的單一運算式查詢。選取規則使用與 AWS IoT 叢集索引相同的查詢語法。如需有關選取規則語法的詳細資訊，請參閱AWS IoT Core 開發人員指南中的AWS IoT 叢集索引查詢語法。

使用*萬用字元來比對多個用戶端裝置與一個選取規則子句。您可以在物件名稱結尾使用此萬用字元，以符合名稱以您指定之字串開頭的用戶端裝置。您也可以使用此萬用字元來比對所有用戶端裝置。

> **Note**
>
> 若要選取包含冒號字元 (`:`) 的值，請使用反斜線字元 (`\\`) 逸出冒號。在 JSON 等格式中，您必須逸出反斜線字元，因此您必須在冒號字元前輸入兩個反斜線字元。例如，指`thingName: MyTeam\\\\:ClientDevice1`定選取名稱為的物件`MyTeam:ClientDevice1`。

您可以指定下列選取器：

- `thingName`— 用戶端裝置 AWS IoT 物件的名稱。

Example 範例選取規則

下列選取規則符合名稱為`MyClientDevice1`或的用戶端裝置`MyClientDevice2`。

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example 範例選取規則 (使用萬用字元)

下列選取規則符合名稱開頭為的用戶端裝置MyClientDevice。

```
thingName: MyClientDevice*
```

Example 範例選取規則 (符合所有裝置)

下列選取規則符合所有用戶端裝置。

```
thingName: *
```

`policyName`

適用於此裝置群組中用戶端裝置的權限原則。指定您在物件中定義的策略名policies稱。

`policies`

連線至核心裝置的用戶端裝置的用戶端裝置授權原則。每個授權原則都會指定一組處理行動以及用戶端裝置可以執行這些處理行動的資源。

此物件包含下列資訊：

`policyNameKey`

此授權原則的名稱。以可協助您識別此授權原則的名稱取*`policyNameKey`*代。您可以使用此原則名稱來定義要套用至裝置群組的原則。

此物件包含下列資訊：

`statementNameKey`

本政策聲明的名稱。請*`statementNameKey`*以可協助您識別此政策聲明的名稱取代。

此物件包含下列資訊：

`operations`

此原則中允許資源的作業清單。

您可以包括以下任何操作：

- `mqtt:connect`— 授予連接到核心設備的權限。用戶端裝置必須具有此權限才能連線至核心裝置。

  此作業支援下列資源：

  - `mqtt:clientId:`*`deviceClientId`*— 根據用戶端裝置用來連線至核心裝置的 MQTT 代理程式的用戶端 ID 來限制存取。取代*`deviceClientId`*為要使用的用戶端 ID。

- `mqtt:publish`— 授予將 MQTT 訊息發佈至主題的權限。

  此作業支援下列資源：

  - `mqtt:topic:`*`mqttTopic`*— 根據用戶端裝置發佈訊息的 MQTT 主題限制存取。將 *`MQTT ##`*取代為要使用的主題。

    此資源不支援 MQTT 主題萬用字元。

- `mqtt:subscribe`— 授予訂閱 MQTT 主題篩選器以接收訊息的權限。

  此作業支援下列資源：

  - `mqtt:topicfilter:`*`mqttTopicFilter`*— 根據用戶端裝置可以訂閱訊息的 MQTT 主題限制存取。以要使*`mqttTopicFilter`*用的主題篩選器取代。

    此資源支援+和 # MQTT 主題萬用字元。如需詳細資訊，請參閱AWS IoT Core 開發人員指南中的 MQTT 主題。

    用戶端裝置可以訂閱您允許的確切主題篩選器。例如，如果您允許用戶端裝置訂閱`mqtt:topicfilter:client/+/status`資源，則用戶端裝置可以訂閱，`client/+/status`但無法訂閱`client/client1/status`。

您可以指定*萬用字元以允許存取所有動作。

`resources`

此原則中允許作業的資源清單。指定與此策略中作業對應的資源。例如，您可以在指定作業的策略中指定 MQTT 主題資源 (`mqtt:topic:`*`mqttTopic`*) 清單。`mqtt:publish`

您可以指定*萬用字元以允許存取所有資源。您無法使用*萬用字元來比對部分資源識別碼。例如，您可以指定**`"resources": "*"`**，但無法指定**`"resources": "mqtt:clientId:*"`**。

`statementDescription`

(選擇性) 此政策聲明的說明。

`certificates`

(選擇性) 此核心裝置的憑證組態選項。此物件包含下列資訊：

`serverCertificateValiditySeconds`

(選擇性) 本機 MQTT 伺服器憑證到期後的時間 (秒)。您可以設定此選項來自訂用戶端裝置中斷連線並重新連線至核心裝置的頻率。

此元件會在到期前 24 小時輪換本機 MQTT 伺服器憑證。MQTT 代理程式 (例如 Moquette MQTT 代理程式元件) 會產生新憑證並重新啟動。發生這種情況時，連接到此核心裝置的所有用戶端裝置都會中斷連線。用戶端裝置可在短時間後重新連線至核心裝置。

預設值：`604800`(7 天)

最小值：`172800`（2 天）

最大值：`864000`（10 天）

`performance`

(選擇性) 此核心裝置的效能組態選項。此物件包含下列資訊：

`maxActiveAuthTokens`

(選擇性) 作用中用戶端裝置授權權杖的數目上限。您可以增加此數目，讓更多數目的用戶端裝置連線到單一核心裝置，而無需重新驗證。

預設：`2500`

`cloudRequestQueueSize`

(選擇性) 此元件拒絕 AWS 雲端 要求之前，要排入佇列的要求數目上限。

預設：`100`

`maxConcurrentCloudRequests`

(選擇性) 要傳送至的並行要求數目上限 AWS 雲端。您可以增加此數目，以改善連線大量用戶端裝置的核心裝置上的驗證效能。

預設：`1`

`certificateAuthority`

(選擇性) 憑證授權單位組態選項，以您自己的中繼憑證授權單位取代核心裝置中繼授權單位。此物件包含下列資訊。

证书

憑證的位置。它可以是文件系統 URI 或指向存儲在硬件安全模塊中的證書的 URI。

`certificateChainUri`

核心裝置 CA 的憑證鏈結位置。這應該是回到根 CA 的完整憑證鏈結。它可以是一個文件系統 URI 或指向存儲在硬件安全模塊中的證書鏈的 URI。

`privateKeyUri`

核心裝置私密金鑰的位置。這可以是檔案系統 URI 或指向儲存在硬體安全性模組中的憑證私密金鑰的 URI。

`security`

(選擇性) 此核心裝置的安全性設定選項。此物件包含下列資訊。

`clientDeviceTrustDurationMinutes`

在需要與核心裝置重新驗證之前，可以信任用戶端裝置的驗證資訊的持續時間 (以分鐘為單位)。預設值為 1。

Example 範例：組態合併更新 (使用限制性原則)

下列範例組態指定允許名稱開頭為的用戶端裝置`MyClientDevice`連線和發佈/訂閱所有主題。

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
    },
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
```

```
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
          ]
        }
      }
    }
  }
}
```

Example 範例：組態合併更新 (使用寬容原則)

下列範例設定指定允許所有用戶端裝置連線和發佈/訂閱所有主題。

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
```

```
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}
```

v2.2.x

`deviceGroups`

裝置群組是具有與核心裝置連線和通訊之權限的用戶端裝置群組。使用選取規則識別用戶端裝置群組，並定義用戶端裝置授權原則，以指定每個裝置群組的權限。

此物件包含下列資訊：

`formatVersion`

此配置物件的格式版本。

您可以從以下選項中選擇：

- `2021-03-05`

`definitions`

此核心裝置的裝置群組。每個定義都會指定一個選取規則，以評估用戶端裝置是否為群組的成員。每個定義也會指定要套用至符合選取規則的用戶端裝置的權限原則。如果用戶端裝置是多個裝置群組的成員，則該裝置的權限將由每個群組的權限原則組成。

此物件包含下列資訊：

`groupNameKey`

此裝置群組的名稱。以可協助您識別此裝置群組的名稱取*`groupNameKey`*代。

此物件包含下列資訊：

`selectionRule`

指定哪些用戶端裝置為此裝置群組成員的查詢。當用戶端裝置連線時，核心裝置會評估此選擇規則，以判斷用戶端裝置是否為此裝置群組的成員。如果用戶端裝置是成員，則核心裝置會使用此裝置群組的策略來授權用戶端裝置的處理行動。

每個選取規則至少包含一個選取規則子句，這是可以與用戶端裝置相符的單一運算式查詢。選取規則使用與 AWS IoT 叢集索引相同的查詢語法。如需有關選取規則語法的詳細資訊，請參閱AWS IoT Core 開發人員指南中的AWS IoT 叢集索引查詢語法。

使用*萬用字元來比對多個用戶端裝置與一個選取規則子句。您可以在物件名稱結尾使用此萬用字元，以符合名稱以您指定之字串開頭的用戶端裝置。您也可以使用此萬用字元來比對所有用戶端裝置。

> **Note**
>
> 若要選取包含冒號字元 (:) 的值，請使用反斜線字元 (\\) 逸出冒號。在 JSON 等格式中，您必須逸出反斜線字元，因此您必須在冒號字元前輸入兩個反斜線字元。例如，指`thingName: MyTeam\\\\:ClientDevice1`定選取名稱為的物件`MyTeam:ClientDevice1`。

您可以指定下列選取器：

- `thingName`— 用戶端裝置 AWS IoT 物件的名稱。

Example 範例選取規則

下列選取規則符合名稱為`MyClientDevice1`或的用戶端裝置`MyClientDevice2`。

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example 範例選取規則 (使用萬用字元)

下列選取規則符合名稱開頭為的用戶端裝置`MyClientDevice`。

```
thingName: MyClientDevice*
```

Example 範例選取規則 (符合所有裝置)

下列選取規則符合所有用戶端裝置。

```
thingName: *
```

policyName

適用於此裝置群組中用戶端裝置的權限原則。指定您在物件中定義的策略名policies稱。

policies

連線至核心裝置的用戶端裝置的用戶端裝置授權原則。每個授權原則都會指定一組處理行動以及用戶端裝置可以執行這些處理行動的資源。

此物件包含下列資訊：

policyNameKey

此授權原則的名稱。以可協助您識別此授權原則的名稱取*policyNameKey*代。您可以使用此原則名稱來定義要套用至裝置群組的原則。

此物件包含下列資訊：

statementNameKey

本政策聲明的名稱。請*statementNameKey*以可協助您識別此政策聲明的名稱取代。

此物件包含下列資訊：

operations

此原則中允許資源的作業清單。

您可以包括以下任何操作：

- mqtt:connect— 授予連接到核心設備的權限。用戶端裝置必須具有此權限才能連線至核心裝置。

  此作業支援下列資源：

  - mqtt:clientId:*deviceClientId*— 根據用戶端裝置用來連線至核心裝置的 MQTT 代理程式的用戶端 ID 來限制存取。取代*deviceClientId*為要使用的用戶端 ID。
- mqtt:publish— 授予將 MQTT 訊息發佈至主題的權限。

此作業支援下列資源：

- mqtt:topic:*mqttTopic*— 根據用戶端裝置發佈訊息的 MQTT 主題限制存取。將 *MQTT ##*取代為要使用的主題。

  此資源不支援 MQTT 主題萬用字元。

- mqtt:subscribe— 授予訂閱 MQTT 主題篩選器以接收訊息的權限。

  此作業支援下列資源：

  - mqtt:topicfilter:*mqttTopicFilter*— 根據用戶端裝置可以訂閱訊息的 MQTT 主題限制存取。以要使*mqttTopicFilter*用的主題篩選器取代。

    此資源支援+和 # MQTT 主題萬用字元。如需詳細資訊，請參閱AWS IoT Core 開發人員指南中的 MQTT 主題。

    用戶端裝置可以訂閱您允許的確切主題篩選器。例如，如果您允許用戶端裝置訂閱mqtt:topicfilter:client/+/status資源，則用戶端裝置可以訂閱，client/+/status但無法訂閱client/client1/status。

  您可以指定*萬用字元以允許存取所有動作。

resources

此原則中允許作業的資源清單。指定與此策略中作業對應的資源。例如，您可以在指定作業的策略中指定 MQTT 主題資源 (mqtt:topic:*mqttTopic*) 清單。mqtt:publish

您可以指定*萬用字元以允許存取所有資源。您無法使用*萬用字元來比對部分資源識別碼。例如，您可以指定**"resources": "*"**，但無法指定**"resources": "mqtt:clientId:*"**。

statementDescription

(選擇性) 此政策聲明的說明。

certificates

(選擇性) 此核心裝置的憑證組態選項。此物件包含下列資訊：

serverCertificateValiditySeconds

(選擇性) 本機 MQTT 伺服器憑證到期後的時間 (秒)。您可以設定此選項來自訂用戶端裝置中斷連線並重新連線至核心裝置的頻率。

此元件會在到期前 24 小時輪換本機 MQTT 伺服器憑證。MQTT 代理程式 (例如 Moquette MQTT 代理程式元件) 會產生新憑證並重新啟動。發生這種情況時，連接到此核心裝置的所有用戶端裝置都會中斷連線。用戶端裝置可在短時間後重新連線至核心裝置。

預設值：604800(7 天)

最小值：172800（2 天）

最大值：864000（10 天）

`performance`

(選擇性) 此核心裝置的效能組態選項。此物件包含下列資訊：

`maxActiveAuthTokens`

(選擇性) 作用中用戶端裝置授權權杖的數目上限。您可以增加此數目，讓更多數目的用戶端裝置連線到單一核心裝置，而無需重新驗證。

預設：2500

`cloudRequestQueueSize`

(選擇性) 此元件拒絕 AWS 雲端 要求之前，要排入佇列的要求數目上限。

預設：100

`maxConcurrentCloudRequests`

(選擇性) 要傳送至的並行要求數目上限 AWS 雲端。您可以增加此數目，以改善連線大量用戶端裝置的核心裝置上的驗證效能。

預設：1

Example 範例：組態合併更新 (使用限制性原則)

下列範例組態指定允許名稱開頭為的用戶端裝置`MyClientDevice`連線和發佈/訂閱所有主題。

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
```

```
        "policyName": "MyRestrictivePolicy"
      }
    },
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
          ]
        }
      }
    }
  }
}
```

Example 範例：組態合併更新 (使用寬容原則)

下列範例設定指定允許所有用戶端裝置連線和發佈/訂閱所有主題。

```
{
  "deviceGroups": {
```

```
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}
```

v2.1.x

`deviceGroups`

裝置群組是具有與核心裝置連線和通訊之權限的用戶端裝置群組。使用選取規則識別用戶端裝置群組，並定義用戶端裝置授權原則，以指定每個裝置群組的權限。

此物件包含下列資訊：

`formatVersion`

此配置物件的格式版本。

您可以從以下選項中選擇：

- `2021-03-05`

`definitions`

此核心裝置的裝置群組。每個定義都會指定一個選取規則，以評估用戶端裝置是否為群組的成員。每個定義也會指定要套用至符合選取規則的用戶端裝置的權限原則。如果用戶端裝置是多個裝置群組的成員，則該裝置的權限將由每個群組的權限原則組成。

此物件包含下列資訊：

groupNameKey

此裝置群組的名稱。以可協助您識別此裝置群組的名稱取*groupNameKey*代。

此物件包含下列資訊：

selectionRule

指定哪些用戶端裝置為此裝置群組成員的查詢。當用戶端裝置連線時，核心裝置會評估此選擇規則，以判斷用戶端裝置是否為此裝置群組的成員。如果用戶端裝置是成員，則核心裝置會使用此裝置群組的策略來授權用戶端裝置的處理行動。

每個選取規則至少包含一個選取規則子句，這是可以與用戶端裝置相符的單一運算式查詢。選取規則使用與 AWS IoT 叢集索引相同的查詢語法。如需有關選取規則語法的詳細資訊，請參閱AWS IoT Core 開發人員指南中的AWS IoT 叢集索引查詢語法。

使用*萬用字元來比對多個用戶端裝置與一個選取規則子句。您可以在物件名稱結尾使用此萬用字元，以符合名稱以您指定之字串開頭的用戶端裝置。您也可以使用此萬用字元來比對所有用戶端裝置。

> **Note**
>
> 若要選取包含冒號字元 (:) 的值，請使用反斜線字元 (\\) 逸出冒號。在 JSON 等格式中，您必須逸出反斜線字元，因此您必須在冒號字元前輸入兩個反斜線字元。例如，指thingName: MyTeam\\\\:ClientDevice1定選取名稱為的物件MyTeam:ClientDevice1。

您可以指定下列選取器：

- thingName— 用戶端裝置 AWS IoT 物件的名稱。

Example 範例選取規則

下列選取規則符合名稱為MyClientDevice1或的用戶端裝置MyClientDevice2。

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example 範例選取規則 (使用萬用字元)

下列選取規則符合名稱開頭為的用戶端裝置MyClientDevice。

```
thingName: MyClientDevice*
```

Example 範例選取規則 (符合所有裝置)

下列選取規則符合所有用戶端裝置。

```
thingName: *
```

`policyName`

適用於此裝置群組中用戶端裝置的權限原則。指定您在物件中定義的策略名`policies`稱。

`policies`

連線至核心裝置的用戶端裝置的用戶端裝置授權原則。每個授權原則都會指定一組處理行動以及用戶端裝置可以執行這些處理行動的資源。

此物件包含下列資訊：

`policyNameKey`

此授權原則的名稱。以可協助您識別此授權原則的名稱取*`policyNameKey`*代。您可以使用此原則名稱來定義要套用至裝置群組的原則。

此物件包含下列資訊：

`statementNameKey`

本政策聲明的名稱。請*`statementNameKey`*以可協助您識別此政策聲明的名稱取代。

此物件包含下列資訊：

`operations`

此原則中允許資源的作業清單。

您可以包括以下任何操作：

- `mqtt:connect`— 授予連接到核心設備的權限。用戶端裝置必須具有此權限才能連線至核心裝置。

  此作業支援下列資源：

- `mqtt:clientId:`*`deviceClientId`*— 根據用戶端裝置用來連線至核心裝置的 MQTT 代理程式的用戶端 ID 來限制存取。取代*`deviceClientId`*為要使用的用戶端 ID。
- `mqtt:publish`— 授予將 MQTT 訊息發佈至主題的權限。

  此作業支援下列資源：

  - `mqtt:topic:`*`mqttTopic`*— 根據用戶端裝置發佈訊息的 MQTT 主題限制存取。將 *`MQTT ##`*取代為要使用的主題。

    此資源不支援 MQTT 主題萬用字元。
- `mqtt:subscribe`— 授予訂閱 MQTT 主題篩選器以接收訊息的權限。

  此作業支援下列資源：

  - `mqtt:topicfilter:`*`mqttTopicFilter`*— 根據用戶端裝置可以訂閱訊息的 MQTT 主題限制存取。以要使*`mqttTopicFilter`*用的主題篩選器取代。

    此資源支援+和 # MQTT 主題萬用字元。如需詳細資訊，請參閱AWS IoT Core 開發人員指南中的 MQTT 主題。

    用戶端裝置可以訂閱您允許的確切主題篩選器。例如，如果您允許用戶端裝置訂閱`mqtt:topicfilter:client/+/status`資源，則用戶端裝置可以訂閱，`client/+/status`但無法訂閱`client/client1/status`。

您可以指定*萬用字元以允許存取所有動作。

`resources`

此原則中允許作業的資源清單。指定與此策略中作業對應的資源。例如，您可以在指定作業的策略中指定 MQTT 主題資源 (`mqtt:topic:`*`mqttTopic`*) 清單。`mqtt:publish`

您可以指定*萬用字元以允許存取所有資源。您無法使用*萬用字元來比對部分資源識別碼。例如，您可以指定**`"resources": "*"`**，但無法指定**`"resources": "mqtt:clientId:*"`**。

`statementDescription`

(選擇性) 此政策聲明的說明。

`certificates`

(選擇性) 此核心裝置的憑證組態選項。此物件包含下列資訊：

`serverCertificateValiditySeconds`

(選擇性) 本機 MQTT 伺服器憑證到期後的時間 (秒)。您可以設定此選項來自訂用戶端裝置中斷連線並重新連線至核心裝置的頻率。

此元件會在到期前 24 小時輪換本機 MQTT 伺服器憑證。MQTT 代理程式 (例如 Moquette MQTT 代理程式元件) 會產生新憑證並重新啟動。發生這種情況時，連接到此核心裝置的所有用戶端裝置都會中斷連線。用戶端裝置可在短時間後重新連線至核心裝置。

預設值：604800(7 天)

最小值：172800（2 天）

最大值：864000（10 天）

Example 範例：組態合併更新 (使用限制性原則)

下列範例組態指定允許名稱開頭為的用戶端裝置`MyClientDevice`連線和發佈/訂閱所有主題。

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
    },
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
```

```
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
          ]
        }
      }
    }
  }
}
```

Example 範例：組態合併更新 (使用寬容原則)

下列範例設定指定允許所有用戶端裝置連線和發佈/訂閱所有主題。

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
          ]
        }
```

```
      }
    }
  }
}
```

v2.0.x

`deviceGroups`

裝置群組是具有與核心裝置連線和通訊之權限的用戶端裝置群組。使用選取規則識別用戶端裝置群組，並定義用戶端裝置授權原則，以指定每個裝置群組的權限。

此物件包含下列資訊：

`formatVersion`

此配置物件的格式版本。

您可以從以下選項中選擇：

- `2021-03-05`

`definitions`

此核心裝置的裝置群組。每個定義都會指定一個選取規則，以評估用戶端裝置是否為群組的成員。每個定義也會指定要套用至符合選取規則的用戶端裝置的權限原則。如果用戶端裝置是多個裝置群組的成員，則該裝置的權限將由每個群組的權限原則組成。

此物件包含下列資訊：

`groupNameKey`

此裝置群組的名稱。以可協助您識別此裝置群組的名稱取*`groupNameKey`*代。

此物件包含下列資訊：

`selectionRule`

指定哪些用戶端裝置為此裝置群組成員的查詢。當用戶端裝置連線時，核心裝置會評估此選擇規則，以判斷用戶端裝置是否為此裝置群組的成員。如果用戶端裝置是成員，則核心裝置會使用此裝置群組的策略來授權用戶端裝置的處理行動。

每個選取規則至少包含一個選取規則子句，這是可以與用戶端裝置相符的單一運算式查詢。選取規則使用與 AWS IoT 叢集索引相同的查詢語法。如需有關選取規則語法的詳細資訊，請參閱AWS IoT Core 開發人員指南中的AWS IoT 叢集索引查詢語法。

使用*萬用字元來比對多個用戶端裝置與一個選取規則子句。您可以在物件名稱結尾使用此萬用字元，以符合名稱以您指定之字串開頭的用戶端裝置。您也可以使用此萬用字元來比對所有用戶端裝置。

> **Note**
>
> 若要選取包含冒號字元 (:) 的值，請使用反斜線字元 (\\) 逸出冒號。在 JSON 等格式中，您必須逸出反斜線字元，因此您必須在冒號字元前輸入兩個反斜線字元。例如，指`thingName: MyTeam\\\\:ClientDevice1`定選取名稱為的物件`MyTeam:ClientDevice1`。

您可以指定下列選取器：

- `thingName`— 用戶端裝置 AWS IoT 物件的名稱。

Example 範例選取規則

下列選取規則符合名稱為`MyClientDevice1`或的用戶端裝置`MyClientDevice2`。

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example 範例選取規則 (使用萬用字元)

下列選取規則符合名稱開頭為的用戶端裝置`MyClientDevice`。

```
thingName: MyClientDevice*
```

Example 範例選取規則 (符合所有裝置)

下列選取規則符合所有用戶端裝置。

```
thingName: *
```

`policyName`

適用於此裝置群組中用戶端裝置的權限原則。指定您在物件中定義的策略名`policies`稱。

`policies`

連線至核心裝置的用戶端裝置的用戶端裝置授權原則。每個授權原則都會指定一組處理行動以及用戶端裝置可以執行這些處理行動的資源。

此物件包含下列資訊：

`policyNameKey`

此授權原則的名稱。以可協助您識別此授權原則的名稱取*`policyNameKey`*代。您可以使用此原則名稱來定義要套用至裝置群組的原則。

此物件包含下列資訊：

`statementNameKey`

本政策聲明的名稱。請*`statementNameKey`*以可協助您識別此政策聲明的名稱取代。

此物件包含下列資訊：

`operations`

此原則中允許資源的作業清單。

您可以包括以下任何操作：

- `mqtt:connect`— 授予連接到核心設備的權限。用戶端裝置必須具有此權限才能連線至核心裝置。

  此作業支援下列資源：

  - `mqtt:clientId:`*`deviceClientId`*— 根據用戶端裝置用來連線至核心裝置的 MQTT 代理程式的用戶端 ID 來限制存取。取代*`deviceClientId`*為要使用的用戶端 ID。
- `mqtt:publish`— 授予將 MQTT 訊息發佈至主題的權限。

  此作業支援下列資源：

  - `mqtt:topic:`*`mqttTopic`*— 根據用戶端裝置發佈訊息的 MQTT 主題限制存取。將 *`MQTT ##`*取代為要使用的主題。

    此資源不支援 MQTT 主題萬用字元。
- `mqtt:subscribe`— 授予訂閱 MQTT 主題篩選器以接收訊息的權限。

此作業支援下列資源：

- mqtt:topicfilter:*mqttTopicFilter*— 根據用戶端裝置可以訂閱訊息的 MQTT 主題限制存取。以要使*mqttTopicFilter*用的主題篩選器取代。

  此資源支援+和 # MQTT 主題萬用字元。如需詳細資訊，請參閱AWS IoT Core 開發人員指南中的 MQTT 主題。

  用戶端裝置可以訂閱您允許的確切主題篩選器。例如，如果您允許用戶端裝置訂閱mqtt:topicfilter:client/+/status資源，則用戶端裝置可以訂閱，client/+/status但無法訂閱client/client1/status。

您可以指定*萬用字元以允許存取所有動作。

`resources`

此原則中允許作業的資源清單。指定與此策略中作業對應的資源。例如，您可以在指定作業的策略中指定 MQTT 主題資源 (mqtt:topic:*mqttTopic*) 清單。mqtt:publish

您可以指定*萬用字元以允許存取所有資源。您無法使用*萬用字元來比對部分資源識別碼。例如，您可以指定**"resources": "*"**，但無法指定**"resources": "mqtt:clientId:*"**。

`statementDescription`

(選擇性) 此政策聲明的說明。

Example 範例：組態合併更新 (使用限制性原則)

下列範例組態指定允許名稱開頭為的用戶端裝置MyClientDevice連線和發佈/訂閱所有主題。

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
    },
    "policies": {
```

```
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
          ]
        }
      }
    }
  }
}
```

Example 範例：組態合併更新 (使用寬容原則)

下列範例設定指定允許所有用戶端裝置連線和發佈/訂閱所有主題。

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
```

```
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}
```

本機記錄檔

此元件使用與 Greengrass 核元件相同的記錄檔。

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

若要檢視此元件的記錄

- 在核心裝置上執行下列命令，即時檢視此元件的記錄檔。以 AWS IoT Greengrass 根資料夾的路徑取代*/greengrass/v2*或 *C:\greengrass\v2*。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
```

變更記錄

下表說明元件每個版本中的變更。

版本	變更
2.4.5	新功能 新增萬用字元前置字元以使用`selectionRule` 參數選取物件名稱的支援。 錯誤修復和改進 修正在某些情況下，憑證不會以新連線資訊更新的問題。
2.4.4	版本更新 Greengrass 2.12.0 版本釋放。
2.4.3	版本更新 Greengrass 2.11.0 版本釋放。
2.4.2	新功能 添加一個新的`startupTimeoutSeconds` 配置選項。
2.4.1	版本更新了 Greengrass 2.10.0 版本。
2.4.0	新功能 • 新增對用戶端裝置驗證的支援，以發出遙測代理程式將發佈的作業指標。 錯誤修復和改進 • 修正用戶端裝置驗證需要 10 秒以上才能驗證用戶端裝置身分的問題。 • 其他小的修正和改進。
2.3.2	錯誤修復和改進 • 添加對緩存主機名信息的支持，以便組件在離線時重新啟動時正確生成證書主題。

版本	變更
2.3.1	錯誤修復和改進 • 修復了內存洩漏。
2.3.0	⚠ Warning 此版本不再可用。此版本中的改進功能在此元件的更新版本中提供。 新功能 • 添加了對客戶端設備的離線身份驗證的支持，以便當核心設備未連接到 Internet 時，它們可以繼續連接到核心設備。 • 新增對客戶提供的憑證授權單位的支援，核心裝置用作根憑證來產生 MQTT 代理人憑證。
2.2.3	版本更新 Greengrass 2.8.0 版本的版本。
2.2.2	錯誤修復和改進 • 修正本機 MQTT 伺服器憑證在特定情況下旋轉頻率高於預期的問題。
2.2.1	版本更新了 Greengrass 核 2.7.0 版本釋放。
2.2.0	新功能 • 添加對自定義組件的支持，以調用進程間通信（IPC）操作以驗證和授權客戶端設備。例如，您可以在自訂 MQTT 代理程式元件中使用這些作業。如需詳細資訊，請參閱 IPC：驗證和授權用戶端裝置。 • 新增`maxActiveAuthTokens` 、和`threadPoolSize` 選項`cloudQueueSize` ，您可以設定這些選項來調整此元件的執行方式。

版本	變更
2.1.0	新功能 • 新增您可以設定的`serverCertificateValiditySeconds` 選項，以便在 MQTT 代理程式伺服器憑證到期時自訂。您可以設定伺服器憑證在 2 到 10 天後到期。 錯誤修復和改進 • 修正此元件如何處理組態重設更新的問題。 • 修正本機 MQTT 伺服器憑證在特定情況下旋轉頻率高於預期的問題。 若要套用此修正程式，您也必須使用 v2.1.0 或更新版本的 Moquette MQTT 代理程式元件。 • 改善此元件在輪換憑證時記錄的訊息。 • 版本更新 Greengrass 2.6.0 版本發布。
2.0.4	版本更新了 Greengrass 核 2.5.0 版本。
2.0.3	錯誤修復和改進 • 如果您輪換核心裝置的私密金鑰，認證現在會重新整理。 • 更新以使日誌消息更清晰。
2.0.2	版本更新 Greengrass 2.4.0 版本的版本。
2.0.1	版本更新了 Greengrass 核 2.3.0 版本。
2.0.0	初始版本。

CloudWatch 度量

Amazon 指 CloudWatch 標元件 (`aws.greengrass.Cloudwatch`) 將 Greengrass 核心裝置的自訂指標發佈到 Amazon。 CloudWatch元件可讓元件發佈 CloudWatch 指標，您可以使用這些指標來監視和分析 Greengrass 核心裝置的環境。如需詳細資訊，請參閱 Amazon 使用者指南中的使 CloudWatch 用 Amazon 指 CloudWatch標。

若要使用此元件發佈 CloudWatch 量度，請將訊息發佈至此元件訂閱的主題。依預設，此元件會訂閱`cloudwatch/metric/put`本機發佈/訂閱主題。您可以在部署此元件時指定其他主題，包括 AWS IoT Core MQTT 主題。

此元件會批次 CloudWatch 處理位於相同命名空間的測量結果，並定期將其發佈至。

> **Note**
>
> 此元件提供與 AWS IoT Greengrass V1 中 CloudWatch 度量連接器類似的功能。如需詳細資訊，請參閱 AWS IoT Greengrass V1 開發人員指南中的CloudWatch 度量連接器。

主題

版本

此元件具有下列版本：

- 3.1.x 版本
- 3.0.x
- 2.1.x
- 2.0.x

如需元件每個版本中變更的相關資訊，請參閱變更記錄檔。

Type

v3.x

此元件是一般元件 (`aws.greengrass.generic`)。Greengrass 核會執行元件的生命週期指令碼。

v2.x

這個組件是一個 Lambda 組件（`aws.greengrass.lambda`）。Greengrass 核使用 Lambda 啟動器組件運行此組件的 Lambda 函數。

如需詳細資訊，請參閱 元件類型。

作業系統

v3.x

此元件可安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

v2.x

此元件只能安裝在 Linux 核心裝置上。

要求

此元件具有下列需求：

3.x

- Python 版本 3.7 安裝在核心設備上，並添加到 PATH 環境變量中。
- Greengrass 裝置角色必須允許`cloudwatch:PutMetricData`動作，如下列 IAM 政策範例所示。

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

```
  "Statement": [
    {
      "Action": [
        "cloudwatch:PutMetricData"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

如需詳細資訊，請參閱 Amazon CloudWatch 使用者指南中的 Amazon CloudWatch 許可參考資料。

2.x

- 您的核心裝置必須符合執行 Lambda 函數的需求。如果您希望核心裝置執行容器化 Lambda 函數，則該裝置必須符合要求才能執行此操作。如需詳細資訊，請參閱 Lambda 函數要求。
- Python 版本 3.7 安裝在核心設備上，並添加到 PATH 環境變量中。
- Greengrass 裝置角色必須允許`cloudwatch:PutMetricData`動作，如下列 IAM 政策範例所示。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "cloudwatch:PutMetricData"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

如需詳細資訊，請參閱 Amazon CloudWatch 使用者指南中的 Amazon CloudWatch 許可參考資料。

- 若要從此元件接收輸出資料，您必須在部署此元件時，合併舊版訂閱路由器元件的下列組態更新 (`aws.greengrass.LegacySubscriptionRouter`)。此配置指定此組件發布響應的主題。

Legacy subscription router v2.1.x

```
{
  "subscriptions": {
    "aws-greengrass-cloudwatch": {
      "id": "aws-greengrass-cloudwatch",
      "source": "component:aws.greengrass.Cloudwatch",
      "subject": "cloudwatch/metric/put/status",
      "target": "cloud"
    }
  }
}
```

Legacy subscription router v2.0.x

```
{
  "subscriptions": {
    "aws-greengrass-cloudwatch": {
      "id": "aws-greengrass-cloudwatch",
      "source": "arn:aws:lambda:region:aws:function:aws-greengrass-
cloudwatch:version",
      "subject": "cloudwatch/metric/put/status",
      "target": "cloud"
    }
  }
}
```

- 將##替換為您 AWS 區域 使用的區域。
- 將##取代為此元件執行的 Lambda 函數版本。若要尋找 Lambda 函數版本，您必須檢視要部署之此元件版本的方案。在AWS IoT Greengrass 主控台中開啟此元件的詳細資料頁面，然後尋找 Lambda 函數索引鍵值組。此鍵值對包含 Lambda 函數的名稱和版本。

> **Important**
>
> 每次部署此元件時，您都必須更新舊版訂閱路由器上的 Lambda 函數版本。這可確保您針對部署的元件版本使用正確的 Lambda 函數版本。

如需詳細資訊，請參閱 建立部署。

端點和連接埠

除了基本作業所需的端點和連接埠之外，此元件還必須能夠對下列端點和連接埠執行輸出要求。如需詳細資訊，請參閱 允許裝置流量透過 Proxy 或防火牆。

端點	連線埠	必要	描述
monitoring. *region*.amazonaws.com	443	是	上傳 CloudWatch 指標。

相依性

部署元件時， AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的需求，才能成功部署元件。本節列出此元件之已發行版本的相依性，以及定義每個相依性之元件版本的語意版本條件約束。您也可以在AWS IoT Greengrass 主控台中檢視元件每個版本的相依性。在元件詳細資料頁面上，尋找 [相依性] 清單。

3.0.0 - 3.1.0

下表列出此元件 3.0.0 至 3.1.0 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
代幣交換服務	>=0.0.0	硬式

2.1.2 and 2.1.3

下表列出此元件 2.1.2 和 2.1.3 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	硬式
发 Lambda 器	^2.0.0	硬式

相依性	兼容版本	相依性類型
Lambda 執行階段	^2.0.0	軟式
代幣交換服務	^2.0.0	硬式

2.1.1

下表列出此元件 2.1.1 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	硬式
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
代幣交換服務	^2.0.0	硬式

2.0.8 - 2.1.0

下表列出此元件 2.0.8 至 2.1.0 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.0.0	硬式
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
代幣交換服務	^2.0.0	硬式

2.0.7

下表列出此元件 2.0.7 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	硬式
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
代幣交換服務	^2.0.0	硬式

2.0.6

下表列出此元件 2.0.6 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	硬式
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
代幣交換服務	^2.0.0	硬式

2.0.5

下表列出此元件 2.0.5 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	硬式
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
代幣交換服務	^2.0.0	硬式

2.0.4

下表列出此元件 2.0.4 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	硬式
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
代幣交換服務	^2.0.0	硬式

2.0.3

下表列出此元件 2.0.3 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.3	硬式
发 Lambda 器	>=1.0.0	硬式
Lambda 執行階段	>=1.0.0	軟式
代幣交換服務	>=1.0.0	硬式

如需有關元件相依性的詳細資訊，請參閱元件方案參考。

組態

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

v3.x

`PublishInterval`

(選擇性) 元件發佈指定命名空間的批次測量結果之前，等待的秒數上限。若要將元件設定為在接收測量結果時發佈測量結果 (也就是說不進行批次處理)，請指定`0`。

元件會在接收到相同命名空間中的 20 個度量之後，或在您指定的間隔之後發佈到。CloudWatch

> **Note**
>
> 組件不會指定事件發佈的順序。

這個值最多可以是 900 秒。

預設值：10 秒

`MaxMetricsToRetain`

(選擇性) 元件以較新的測量結果取代之前，所有命名空間要儲存在記憶體中的測量結果數目上限。

當核心裝置沒有網際網路連線時，會套用此限制，因此元件會緩衝稍後要發佈的指標。當緩衝區已滿時，元件會以較新的測量結果取代最舊的測量結果。指定命名空間中的度量只會取代相同命名空間中的度量。

> **Note**
>
> 如果元件的主機程序中斷，則元件不會儲存度量。例如，在部署期間或核心裝置重新啟動時，可能會發生這種情況。

此值必須至少為 2,000 個量度。

預設值：5,000 個量度

`InputTopic`

(選擇性) 元件訂閱接收訊息的主題。如果您指定`true`為`PubSubToIoTCore`，您可以在本主題中使用 MQTT 萬用字元 (+ 和 #)。

預設：`cloudwatch/metric/put`

`OutputTopic`

（可選）組件發布狀態響應的主題。

預設：`cloudwatch/metric/put/status`

`PubSubToIoTCore`

(選用) 定義是否要發佈和訂閱 AWS IoT Core MQTT 主題的字串值。支援的值為 `true` 和 `false`。

預設：`false`

`UseInstaller`

(選擇性) Boolean 值，定義是否使用此元件中的安裝程式指令碼來安裝此元件的 SDK 相依性。

`false`如果您想要使用自訂指令碼安裝相依性，或者想要在預先建置的 Linux 映像檔中包含執行階段相依性，請將此值設定為。若要使用此元件，您必須安裝下列程式庫 (包括任何相依性)，並讓預設的 Greengrass 系統使用者可以使用這些程式庫。

- AWS IoT Device SDK 對於 Python 的 V2
- AWS SDK for Python (Boto3)

預設：`true`

`PublishRegion`

(選擇性) AWS 區域 要發佈 CloudWatch 量度的目標。此值會覆寫核心裝置的預設區域。只有跨區域量度才需要此參數。

`accessControl`

(選擇性) 包含允許元件發行和訂閱指定主題的授權原則的物件。如果您為`InputTopic`和指定自訂值`OutputTopic`，則必須更新此物件中的資源值。

預設：

```
{
  "aws.greengrass.ipc.pubsub": {
    "aws.greengrass.Cloudwatch:pubsub:1": {
      "policyDescription": "Allows access to subscribe to input topics.",
      "operations": [
        "aws.greengrass#SubscribeToTopic"
      ],
      "resources": [
        "cloudwatch/metric/put"
```

```
      ]
    },
    "aws.greengrass.Cloudwatch:pubsub:2": {
      "policyDescription": "Allows access to publish to output topics.",
      "operations": [
        "aws.greengrass#PublishToTopic"
      ],
      "resources": [
        "cloudwatch/metric/put/status"
      ]
    }
  },
  "aws.greengrass.ipc.mqttproxy": {
    "aws.greengrass.Cloudwatch:mqttproxy:1": {
      "policyDescription": "Allows access to subscribe to input topics.",
      "operations": [
        "aws.greengrass#SubscribeToIoTCore"
      ],
      "resources": [
        "cloudwatch/metric/put"
      ]
    },
    "aws.greengrass.Cloudwatch:mqttproxy:2": {
      "policyDescription": "Allows access to publish to output topics.",
      "operations": [
        "aws.greengrass#PublishToIoTCore"
      ],
      "resources": [
        "cloudwatch/metric/put/status"
      ]
    }
  }
}
```

Example 範例：組態合併更新

```
{
  "PublishInterval": 0,
  "PubSubToIoTCore": true
}
```

v2.x

> **Note**
>
> 此元件的預設組態包括 Lambda 函數參數。建議您只編輯下列參數，以便在裝置上設定此元件。

`lambdaParams`

包含此元件 Lambda 函數之參數的物件。此物件包含下列資訊：

`EnvironmentVariables`

包含 Lambda 函數參數的物件。此物件包含下列資訊：

`PUBLISH_INTERVAL`

(選擇性) 元件發佈指定命名空間的批次測量結果之前，等待的秒數上限。若要將元件設定為在接收測量結果時發佈測量結果 (也就是說不進行批次處理)，請指定`0`。

元件會在接收到相同命名空間中的 20 個度量之後，或在您指定的間隔之後發佈到。CloudWatch

> **Note**
>
> 該組件不保證事件發布的順序。

這個值最多可以是 900 秒。

預設值：10 秒

`MAX_METRICS_TO_RETAIN`

(選擇性) 元件以較新的測量結果取代之前，所有命名空間要儲存在記憶體中的測量結果數目上限。

當核心裝置沒有網際網路連線時，會套用此限制，因此元件會緩衝稍後要發佈的指標。當緩衝區已滿時，元件會以較新的測量結果取代最舊的測量結果。指定命名空間中的度量只會取代相同命名空間中的度量。

> **Note**
>
> 如果元件的主機程序中斷，則元件不會儲存度量。例如，在部署期間或核心裝置重新啟動時，可能會發生這種情況。

此值必須至少為 2,000 個量度。

預設值：5,000 個量度

`PUBLISH_REGION`

(選擇性) AWS 區域 要發佈 CloudWatch 量度的目標。此值會覆寫核心裝置的預設區域。只有跨區域量度才需要此參數。

`containerMode`

(選擇性) 此元件的容器化模式。您可以從以下選項中選擇：

- `NoContainer`— 元件不會在隔離的執行階段環境中執行。
- `GreengrassContainer`— 元件在 AWS IoT Greengrass 容器內的隔離執行階段環境中執行。

預設：`GreengrassContainer`

`containerParams`

(選擇性) 包含此元件之容器參數的物件。如果您指定`GreengrassContainer`為，則元件會使用這些參數`containerMode`。

此物件包含下列資訊：

`memorySize`

(選擇性) 配置給元件的記憶體容量 (以 KB 為單位)。

預設值為 64 MB。

`pubsubTopics`

(選擇性) 包含元件訂閱以接收訊息之主題的物件。您可以指定每個主題，以及元件是否訂閱 MQTT 主題 AWS IoT Core 或本機發佈/訂閱主題。

此物件包含下列資訊：

`0`-這是一個數組索引作為一個字符串。

包含下列資訊的物件：

`type`

(選擇性) 此元件用來訂閱訊息的發佈/訂閱訊息類型。您可以從以下選項中選擇：

- PUB_SUB - 訂閱本機發佈/訂閱訊息。如果您選擇此選項，則主題不能包含 MQTT 萬用字元。如需指定此選項時如何從自訂元件傳送郵件的詳細資訊，請參閱發佈/訂閱本地訊息。
- `IOT_CORE`— 訂閱 AWS IoT Core MQTT 訊息。如果您選擇此選項，主題可以包含 MQTT 萬用字元。如需指定此選項時如何從自訂元件傳送郵件的詳細資訊，請參閱發布/訂閱MQTT 訊 AWS IoT Core 息。

預設：PUB_SUB

`topic`

(選擇性) 元件訂閱接收訊息的主題。如果您指定`IotCore`為`type`，您可以在本主題中使用 MQTT 萬用字元 (+和#)。

Example 範例：組態合併更新 (容器模式)

```
{
  "containerMode": "GreengrassContainer"
}
```

Example 範例：組態合併更新 (無容器模式)

```
{
  "containerMode": "NoContainer"
}
```

輸入資料

此元件接受下列主題的測量結果，並將量度發佈至 CloudWatch。根據預設，此元件會訂閱本機發佈/訂閱訊息。如需如何從自訂元件將訊息發佈至此元件的詳細資訊，請參閱發佈/訂閱本地訊息。

從元件版本 v3.0.0 開始，您可以選擇性地將此元件設定為訂閱 MQTT 主題，方法是將組態參數設定PubSubToIoTCore為。true如需有關在自訂元件中將訊息發佈至 MQTT 主題的詳細資訊，請參閱。發布/訂閱MQTT 訊 AWS IoT Core 息

預設主題：cloudwatch/metric/put

該消息接受以下屬性。輸入訊息必須為 JSON 格式。

request

此訊息中的量度。

請求物件包含要發佈到 CloudWatch 的指標資料。測量結果值必須符合PutMetricData作業規格。

類型：object包含下列資訊：

namespace

此要求中測量結果資料的使用者定義命名空間。 CloudWatch 使用命名空間做為量度資料點的容器。

> **Note**
>
> 您無法指定以保留字串開頭的命名空間AWS/。

類型：string

有效模式：[^:].*

metricData

指標的資料。

類型：object包含下列資訊：

metricName

指標的名稱

類型：string

value

指標的值。

> **Note**
>
> CloudWatch 拒絕太小或太大的值。值必須介於8.515920e-109和 1.174271e+108 (底數 10) 或2e-360和 2e360 (底數 2) 之間。 CloudWatch 不支援特殊值NaN，例如+Infinity、和-Infinity。

類型：double

dimensions

(選擇性) 量度的維度。維度提供指標及其資料的其他資訊。一個指標可以定義最多 10 個維度。

此元件會自動包含名為的維度coreName，其中值是核心裝置的名稱。

物件類array型：每個物件都包含下列資訊：

name

(選擇性) 維度名稱。

類型：string

value

(選擇性) 尺寸值。

類型：string

timestamp

(選擇性) 接收測量結果資料的時間，單位為 Unix 紀元時間的秒數。

預設為元件接收訊息的時間。

類型：double

> **Note**
>
> 如果您在此元件的 2.0.3 版和 2.0.7 版之間使用，建議您在從單一來源傳送多個量度時，分別擷取每個量度的時間戳記。請勿使用變數來儲存時間戳記。

`unit`

(選擇性) 量度的單位。

類型：`string`

有效值：`Seconds`,`Microseconds`,`Milliseconds`,`Bytes`,`Kilobytes`,`Megabytes`,`Gigabytes`,`Ter`
`Second`,`Kilobytes/Second`,`Megabytes/Second`,`Gigabytes/Second`,`Terabytes/Second`,`Bits/Second`,`Kilobits/Second`,,`Megabits/Second`,`Gigabits/Second`,`Terabits/Second`,`Count/Second`, None

預設為 None。

> **Note**
>
> 套用至 CloudWatch `PutMetricData` API 的所有配額都會套用至您使用此元件發佈的量度。下列配額特別重要：
>
> - API 承載資料的限制為 40 KB
> - 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CloudWatch 使用指南中的CloudWatch 服務配額。

Example 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

根據預設，此元件會在下列本機發佈/訂閱主題上將回應作為輸出資料發佈。如需如何在自訂元件中訂閱有關此主題之訊息的詳細資訊，請參閱發佈/訂閱本地訊息。

您可以選擇將PubSubToIoTCore組態參數設定為，將此元件配置為發佈至 MQTT 主題。`true`如需有關在自訂元件中訂閱 MQTT 主題訊息的詳細資訊，請參閱。發布/訂閱MQTT 訊 AWS IoT Core 息

> **Note**
>
> 依預設，元件版本 2.0.x 會將回應發佈為 MQTT 主題的輸出資料。您必須在舊版訂閱路由器元件的組態`subject`中將主題指定為。

預設主題：`cloudwatch/metric/put/status`

Example 範例輸出：成功

回應包括測量結果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"
  }
}
```

Example 範例輸出：失敗

```
{
  "response" : {
    "namespace": "Greengrass",
```

```
    "error": "InvalidInputException",
    "error_message": "cw metric is invalid",
    "status": "fail"
  }
}
```

> **Note**
>
> 如果元件偵測到可以重試的錯誤 (例如連線錯誤)，它會在下一個批次中重試發佈。

授權

此元件包括下列協力廠商軟體/授權：

- AWS SDK for Python (Boto3)/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

此元件是根據 Greengrass 核心軟體授權合約發行的。

本機記錄檔

此元件使用下列記錄檔。

Linux

```
/greengrass/v2/logs/aws.greengrass.Cloudwatch.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.Cloudwatch.log
```

若要檢視此元件的記錄

- 在核心裝置上執行下列命令，即時檢視此元件的記錄檔。以 AWS IoT Greengrass 根資料夾的路徑取代*/greengrass/v2*或 *C:\greengrass\v2*。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.Cloudwatch.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.greengrass.Cloudwatch.log -Tail 10 -Wait
  ```

變更記錄

下表說明元件每個版本的變更。

v3.x

版本	變更
3.1.0	錯誤修復和改進 • 添加對 HTTPS 網絡代理配置的支持。如需詳細資訊，請參閱 連線至連接埠 443 或透過網路代理 及 讓核心裝置信任 HTTPS 代理。
3.0.0	此版本的 CloudWatch 指標元件預期的組態參數與版本 2.x 不同。如果您對 2.x 版使用非預設組態，並且想要從 v2.x 升級至 v3.x，則必須更新元件的組態。如需詳細資訊，請參閱CloudWatch測量結果元件組態。 新功能 • 添加對運行 Windows 的核心設備的支持。 • 將元件類型從 Lambda 元件變更為泛型元件。此元件現在不再依賴舊版訂閱路由器元件來建立訂閱。 • 新增`InputTopic` 組態參數，以指定元件訂閱接收訊息的主題。 • 添加新的`OutputTopic` 配置參數以指定組件發布狀態響應的主題。 • 新增`PubSubToIoTCore` 組態參數，以指定是否要發佈和訂閱 AWS IoT Core MQTT 主題。

版本	變更
	• 新增可讓您選擇性地停用安裝元件相依性的安裝指令碼的新`UseInstaller` 組態參數。 錯誤修復和改進 添加對輸入數據中重複時間戳的支持。

v2.x

版本	變更
2.1.3	版本更新 Greengrass 2.11.0 版本釋放。
2.1.2	版本更新了 Greengrass 核 2.7.0 版本釋放。
2.1.1	版本更新 Greengrass 2.6.0 版本發布。
2.1.0	新功能 • 添加對 HTTPS 網絡代理配置的支持。如需詳細資訊，請參閱 連線至連接埠 443 或透過網路代理 及 讓核心裝置信任 HTTPS 代理。
2.0.8	錯誤修復和改進 • 添加對輸入數據中重複時間戳的支持。 • 版本更新了 Greengrass 核 2.5.0 版本。
2.0.7	版本更新 Greengrass 2.4.0 版本的版本。
2.0.6	版本更新了 Greengrass 核 2.3.0 版本。
2.0.5	版本更新了 Greengrass 核 2.2.0 版本。
2.0.4	版本更新 Greengrass 2.1.0 版本發布。
2.0.3	初始版本。

另請參閱

- 在 Amazon 用戶指南中使 CloudWatch 用 Amazon 指 CloudWatch標
- PutMetricData在 Amazon CloudWatch API 參考

AWS IoT Device Defender

AWS IoT Device Defender 元件 (`aws.greengrass.DeviceDefender`) 會通知系統管理員 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。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的 AWS IoT Device Defender。

此元件會讀取核心裝置上的系統指標。然後，它會將指標發佈到 AWS IoT Device Defender。如需有關如何讀取和解譯此元件報告之量度的詳細資訊，請參閱AWS IoT Core 開發人員指南中的裝置量度文件規格。

> **Note**
>
> 此元件提供與中的裝置 Defender 連接器類似的功能 AWS IoT Greengrass V1。如需詳細資訊，請參閱AWS IoT Greengrass V1 開發人員指南中的裝置防禦者連接器。

主題

- 版本
- Type
- 作業系統
- 要求
- 相依性
- 組態
- 輸入資料
- 輸出資料
- 本機記錄檔
- 授權
- 變更記錄

版本

此元件具有下列版本：

- 3.1.x 版本
- 3.0.x
- 2.0.x

如需元件每個版本變更的相關資訊，請參閱變更記錄檔。

Type

v3.x

此元件是一般元件 (`aws.greengrass.generic`)。Greengrass 核會執行元件的生命週期指令碼。

v2.x

這個組件是一個 Lambda 組件（`aws.greengrass.lambda`）。Greengrass 核使用 Lambda 啟動器組件運行此組件的 Lambda 函數。

如需詳細資訊，請參閱 元件類型。

作業系統

v3.x

此元件可安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

v2.x

此元件只能安裝在 Linux 核心裝置上。

要求

此元件具有下列需求：

v3.x

- Python 版本 3.7 安裝在核心設備上，並添加到 PATH 環境變量中。
- AWS IoT Device Defender 設定為使用「偵測」功能來監視違規。如需詳細資訊，請參閱AWS IoT Core 開發人員指南中的偵測。

v2.x

- 您的核心裝置必須符合執行 Lambda 函數的需求。如果您希望核心裝置執行容器化 Lambda 函數，則該裝置必須符合要求才能執行。如需詳細資訊，請參閱 Lambda 函數要求。
- Python 版本 3.7 安裝在核心設備上，並添加到 PATH 環境變量中。
- AWS IoT Device Defender 設定為使用「偵測」功能來監視違規。如需詳細資訊，請參閱AWS IoT Core 開發人員指南中的偵測。
- 安裝在核心裝置上的 psutil 程式庫。版本 5.7.0 是經過驗證可與元件搭配使用的最新版本。
- 核心裝置上安裝的 cbor 程式庫。版本 1.0.0 是驗證與組件一起使用的最新版本。
- 若要從此元件接收輸出資料，您必須在部署此元件時，合併舊版訂閱路由器元件的下列組態更新 (`aws.greengrass.LegacySubscriptionRouter`)。此配置指定此組件發布響應的主題。

  Legacy subscription router v2.1.x

  ```
  {
    "subscriptions": {
      "aws-greengrass-device-defender": {
        "id": "aws-greengrass-device-defender",
        "source": "component:aws.greengrass.DeviceDefender",
        "subject": "$aws/things/+/defender/metrics/json",
        "target": "cloud"
      }
    }
  }
  ```

  Legacy subscription router v2.0.x

  ```
  {
    "subscriptions": {
      "aws-greengrass-device-defender": {
        "id": "aws-greengrass-device-defender",
        "source": "arn:aws:lambda:region:aws:function:aws-greengrass-device-
  defender:version",
  ```

```
      "subject": "$aws/things/+/defender/metrics/json",
      "target": "cloud"
    }
  }
}
```

- 將##替換為您 AWS 區域 使用的區域。
- 將##取代為此元件執行的 Lambda 函數版本。若要尋找 Lambda 函數版本，您必須檢視要部署之此元件版本的方案。在AWS IoT Greengrass 主控台中開啟此元件的詳細資料頁面，然後尋找 Lambda 函數索引鍵值組。此鍵值對包含 Lambda 函數的名稱和版本。

> **Important**
>
> 每次部署此元件時，您都必須更新舊版訂閱路由器上的 Lambda 函數版本。這可確保您針對部署的元件版本使用正確的 Lambda 函數版本。

如需詳細資訊，請參閱 建立部署。

相依性

部署元件時， AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的需求，才能成功部署元件。本節列出此元件之已發行版本的相依性，以及定義每個相依性之元件版本的語意版本條件約束。您也可以在AWS IoT Greengrass 主控台中檢視元件每個版本的相依性。在元件詳細資料頁面上，尋找 [相依性] 清單。

3.1.1

下表列出此元件 3.1.1 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
代幣交換服務	>=0.0.0	硬式

3.0.0 - 3.0.2

下表列出此元件 3.0.0 至 3.0.2 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
代幣交換服務	>=0.0.0	硬式

2.0.10 and 2.0.11

下表列出此元件 2.0.10 和 2.0.11 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	硬式
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
代幣交換服務	^2.0.0	硬式

2.0.9

下表列出此元件 2.0.9 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	硬式
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
代幣交換服務	^2.0.0	硬式

2.0.8

下表列出此元件 2.0.8 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.0.0	硬式
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
代幣交換服務	^2.0.0	硬式

2.0.7

下表列出此元件 2.0.7 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	硬式
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
代幣交換服務	^2.0.0	硬式

2.0.6

下表列出此元件 2.0.6 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	硬式
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式

相依性	兼容版本	相依性類型
代幣交換服務	^2.0.0	硬式

2.0.5

下表列出此元件 2.0.5 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	硬式
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
代幣交換服務	^2.0.0	硬式

2.0.4

下表列出此元件 2.0.4 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	硬式
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
代幣交換服務	^2.0.0	硬式

2.0.3

下表列出此元件 2.0.3 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.3	硬式
发 Lambda 器	>=1.0.0	硬式
Lambda 執行階段	>=1.0.0	軟式
代幣交換服務	>=1.0.0	硬式

如需有關元件相依性的詳細資訊，請參閱元件方案參考。

組態

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

v3.x

`PublishRetryCount`

重試發佈的次數。此功能在 3.1.1 版本中提供。

最小值為 0。

最大值為 72。

預設：5

`SampleIntervalSeconds`

(選擇性) 元件收集並報告量度的每個週期之間的時間 (以秒為單位)。

最低值為 300 秒 (5 分鐘)。

預設值：300 秒

`UseInstaller`

(選擇性) Boolean 值，定義是否使用此元件中的安裝程式指令碼來安裝此元件的相依性。

`false`如果您想要使用自訂指令碼安裝相依性，或者想要在預先建置的 Linux 映像檔中包含執行階段相依性，請將此值設定為。若要使用此元件，您必須安裝下列程式庫 (包括任何相依性)，並讓預設的 Greengrass 系統使用者可以使用這些程式庫。

- AWS IoT Device SDK 對於 Python 的 V2
- CBOR 圖書館。版本 1.0.0 是驗證與組件一起使用的最新版本。
- 普蘇提爾程式庫。版本 5.7.0 是經過驗證可與元件搭配使用的最新版本。

> **Note**
>
> 如果您在設定為使用 HTTPS Proxy 的核心裝置上使用此元件 3.0.0 或 3.0.1 版，則必須將此值設定為。`false`在這些版本的此元件中，安裝程式指令碼不支援 HTTPS Proxy 後方的作業。

預設：`true`

v2.x

> **Note**
>
> 此元件的預設組態包括 Lambda 函數參數。建議您只編輯下列參數，以便在裝置上設定此元件。

`lambdaParams`

包含此元件 Lambda 函數之參數的物件。此物件包含下列資訊：

`EnvironmentVariables`

包含 Lambda 函數參數的物件。此物件包含下列資訊：

`PROCFS_PATH`

(選擇性) 資/proc料夾的路徑。

- 若要在容器中執行此元件，請使用預設值/host-proc。依預設，元件會在容器中執行。
- 若要以無容器模式執行此元件，請/proc為此參數指定。

預設：/host-proc。這是此元件在容器中裝載/proc資料夾的預設路徑。

> **Note**
>
> 此元件對此資料夾具有唯讀存取權。

SAMPLE_INTERVAL_SECONDS

(選擇性) 元件收集並報告量度的每個週期之間的時間 (以秒為單位)。

最低值為 300 秒 (5 分鐘)。

預設值：300 秒

containerMode

(選擇性) 此元件的容器化模式。您可以從以下選項中選擇：

- GreengrassContainer— 元件在 AWS IoT Greengrass 容器內的隔離執行階段環境中執行。
- NoContainer— 元件不會在隔離的執行階段環境中執行。

如果指定此選項，則必須/proc為PROCFS_PATH環境變數參數指定。

預設：GreengrassContainer

containerParams

(選擇性) 包含此元件之容器參數的物件。如果您指定GreengrassContainer為，則元件會使用這些參數containerMode。

此物件包含下列資訊：

memorySize

(選擇性) 配置給元件的記憶體容量 (以 KB 為單位)。

預設值為 5 萬 KB。

pubsubTopics

(選擇性) 包含元件訂閱以接收訊息之主題的物件。您可以指定每個主題，以及元件是否訂閱 MQTT 主題 AWS IoT Core 或本機發佈/訂閱主題。

此物件包含下列資訊：

`0`-這是一個數組索引作為一個字符串。

包含下列資訊的物件：

`type`

(選擇性) 此元件用來訂閱訊息的發佈/訂閱訊息類型。您可以從以下選項中選擇：

- PUB_SUB - 訂閱本機發佈/訂閱訊息。如果您選擇此選項，則主題不能包含 MQTT 萬用字元。如需指定此選項時如何從自訂元件傳送郵件的詳細資訊，請參閱發佈/訂閱本地訊息。
- `IOT_CORE`— 訂閱 AWS IoT Core MQTT 訊息。如果您選擇此選項，主題可以包含 MQTT 萬用字元。如需指定此選項時如何從自訂元件傳送郵件的詳細資訊，請參閱發布/訂閱MQTT 訊 AWS IoT Core 息。

預設：PUB_SUB

`topic`

(選擇性) 元件訂閱接收訊息的主題。如果您指定`IotCore`為`type`，您可以在本主題中使用 MQTT 萬用字元 (+和#)。

Example 範例：組態合併更新 (容器模式)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "PROCFS_PATH": "/host_proc"
    }
  },
  "containerMode": "GreengrassContainer"
}
```

Example 範例：組態合併更新 (無容器模式)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "PROCFS_PATH": "/proc"
    }
  },
  "containerMode": "NoContainer"
```

```
}
```

輸入資料

此組件不接受消息作為輸入數據。

輸出資料

此元件會將安全度量結果發佈至的下列保留主題 AWS IoT Device Defender。發佈指標時，此元件會取代*coreDeviceName*為核心裝置的名稱。

主題 (AWS IoT Core MQTT): `$aws/things/`*coreDeviceName*`/defender/metrics/json`

Example 範例輸出

```
{
  "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
      }
    }
  }
}
```

如需有關此元件報告之量度的詳細資訊，請參閱AWS IoT Core 開發人員指南中的裝置量度文件規格。

本機記錄檔

此元件會使用下列記錄檔。

Linux

```
/greengrass/v2/logs/aws.greengrass.DeviceDefender.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.DeviceDefender.log
```

若要檢視此元件的記錄

- 在核心裝置上執行下列命令，即時檢視此元件的記錄檔。以 AWS IoT Greengrass 根資料夾的路徑取代*/greengrass/v2*或 *C:\greengrass\v2*。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.DeviceDefender.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.greengrass.DeviceDefender.log -Tail 10 -
  Wait
  ```

授權

此元件是根據 Greengrass 核心軟體授權合約發行的。

變更記錄

下表說明元件每個版本中的變更。

v3.x

版本	變更
3.1.1	錯誤修復和改進 • 當網路中斷後連線無法復原時，新增用戶端連線的重試次數。 • 為發佈指標新增可設定的重試。
3.1.0	錯誤修復和改進 • 添加對 HTTPS 網絡代理配置的支持。如需詳細資訊，請參閱 連線至連接埠 443 或透過網路代理 及 讓核心裝置信任 HTTPS 代理。

版本	變更
3.0.1	修正元件如何計算量度差異值的問題。
3.0.0	**Warning** 此版本已不再提供。此版本中的改進功能在此元件的更新版本中提供。 初始版本。

v2.x

版本	變更
2.0.11	版本更新了 Greengrass 核 2.11.0 版本釋放。
2.0.10	版本更新了 Greengrass 核 2.7.0 版本釋放。
2.0.9	版本更新 Greengrass 2.6.0 版本的版本。
2.0.8	版本更新了 Greengrass 核 2.5.0 版本。
2.0.7	版本更新 Greengrass 2.4.0 版本的版本。
2.0.6	版本更新了 Greengrass 核 2.3.0 版本。
2.0.5	版本更新了 Greengrass 核 2.2.0 版本。
2.0.4	版本更新 Greengrass 2.1.0 版本發布。
2.0.3	初始版本。

磁碟後臺解決程式

磁碟多工緩衝處理程式元件 (`aws.greengrass.DiskSpooler`) 為從 Greengrass 核心裝置多工緩衝處理的訊息提供永久儲存選項。 AWS IoT Core此元件會將這些輸出郵件儲存在磁碟上。

主題

版本

此元件具有下列版本：

- 1.0.x

Type

這個組件是一個插件組件（`aws.greengrass.plugin`）。Greengrass 核在與核相同的 Java 虛擬機（JVM）中運行此組件。當您在核心裝置上變更此元件的版本時，核心會重新啟動。

此組件使用與 Greengrass 核相同的日誌文件。如需詳細資訊，請參閱 監控AWS IoT Greengrass日誌。

如需詳細資訊，請參閱 元件類型。

作業系統

此元件可安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件具有下列需求：

- `storageType`應該設定為`Disk`使用此元件。您可以在 Greengrass 核配置中進行設置。
- `maxSizeInBytes`不得設定為大於裝置上的可用空間。您可以在 Greengrass 核配置中進行設置。
- 支援磁碟多工緩衝處理程式元件在 VPC 中執行。

相依性

部署元件時， AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的需求，才能成功部署元件。本節列出此元件之已發行版本的相依性，以及定義每個相依性之元件版本的語意版本條件約束。您也可以在AWS IoT Greengrass 主控台中檢視元件每個版本的相依性。在元件詳細資料頁面上，尋找 [相依性] 清單。

1.0.1 – 1.0.3

下表列出此元件 1.0.1 到 1.0.3 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.11.0	硬式

1.0.0

下表列出此元件 1.0.0 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.11.0	硬式

如需有關元件相依性的詳細資訊，請參閱元件方案參考。

用量

若要使用磁碟多工緩衝處理程式元件，`aws.greengrass.DiskSpooler`必須部署。

若要配置和使用此元件，您必須將設定`pluginName`為`aws.greengrass.DiskSpooler`。

本機記錄檔

此元件使用與 Greengrass 核元件相同的記錄檔。

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

若要檢視此元件的記錄

- 在核心裝置上執行下列命令，即時檢視此元件的記錄檔。以 AWS IoT Greengrass 根資料夾的路徑取代*/greengrass/v2*或 *C:\greengrass\v2*。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

變更記錄

下表說明元件每個版本中的變更。

版本	變更
1.0.3	錯誤修復和改進 透過重複使用資料庫連線來改善效能。
1.0.2	錯誤修復和改進 修正 MQTT 郵件格式欄位在特定情況下不會持續存在的問題。
1.0.1	版本更新了 Greengrass 核 2.12.0 版本釋放。
1.0.0	初始版本。

碼頭應用程序管理器

Docker 應用程序管理器組件（`aws.greengrass.DockerApplicationManager`）允許AWS IoT Greengrass從亞馬遜彈性容器註冊表（Amazon ECR）上託管的公共映像註冊表和私有註冊表下載 Docker 映像。它還可AWS IoT Greengrass以自動管理登入資料，以安全地從 Amazon ECR 中的私有儲存庫下載映像。

當您開發執行 Docker 容器的自訂元件時，請將 Docker 應用程式管理員納入為相依性，以下載指定為元件中的成品的 Docker 映像檔。如需詳細資訊，請參閱 運行碼頭容器。

主題

版本

此元件具有下列版本：

- 2.0.x

Type

此元件是一般元件 (`aws.greengrass.generic`)。Greengrass 核會執行元件的生命週期指令碼。

如需詳細資訊，請參閱 元件類型。

作業系統

此元件可安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件具有下列需求：

- 碼頭引擎 1.9.1 或更高版本安裝在 Greengrass 核心設備上。版本 20.10 是經過驗證可與AWS IoT Greengrass核心軟件配合使用的最新版本。在部署執行 Docker 容器的元件之前，您必須直接在核心裝置上安裝 Docker。
- 在您部署此元件之前，Docker 精靈會在核心裝置上啟動並執行。
- Docker 圖像存儲在以下支持的圖像源之一中：
  - Amazon Elastic Container Registry (Amazon ECR) 中的公共和私有映像存儲庫
  - 公共碼頭集線器存儲庫
  - 公共碼頭信任的註冊表
- Docker 映像檔包含在您的自訂 Docker 容器元件中作為成品。使用下列 URI 格式來指定您的泊塢視窗映像檔：
  - 私人 Amazon ECR 圖片：`docker:account-id.dkr.ecr.region.amazonaws.com/repository/image[:tag|@digest]`
  - 公共 Amazon ECR 圖片：`docker:public.ecr.aws/repository/image[:tag|@digest]`
  - 公共碼頭集線器映像：`docker:name[:tag|@digest]`

  如需詳細資訊，請參閱 運行碼頭容器。

  > **Note**
  >
  > 如果您未在映像的成品 URI 中指定映像標籤或映像摘要，則當您部署自訂 Docker 容器元件時，Docker 應用程式管理員會提取該映像的最新可用版本。若要確保所有核心裝置都執行相同版本的映像，建議您在成品 URI 中加入映像標記或影像摘要。

- 執行 Docker 容器元件的系統使用者必須具有根或管理員權限，或者您必須將 Docker 設定為以非 root 使用者或非管理員使用者身分執行。
  - 在 Linux 裝置上，您可以將使用者新增至`docker`群組，以便在不使用的情況下呼叫`docker`指令`sudo`。

- 在 Windows 裝置上，您可以將使用者新增至`docker-users`群組，以便在沒有系統管理員權限的情況下呼叫`docker`命令。

  Linux or Unix

  若要將`ggc_user`您用來執行 Docker 容器元件的非 root 使用者新增至`docker`群組，請執行下列命令。

  ```
  sudo usermod -aG docker ggc_user
  ```

  如需詳細資訊，請參閱以非 root 使用者身分管理 Docker。

  Windows Command Prompt (CMD)

  若要將`ggc_user`或您用來執行 Docker 容器元件的使用者新增至`docker-users`群組，請以系統管理員身分執行下列命令。

  ```
  net localgroup docker-users ggc_user /add
  ```

  Windows PowerShell

  若要將`ggc_user`或您用來執行 Docker 容器元件的使用者新增至`docker-users`群組，請以系統管理員身分執行下列命令。

  ```
  Add-LocalGroupMember -Group docker-users -Member ggc_user
  ```

- 如果您將 AWS IoT GreengrassCore 軟體設定為使用網路 Proxy，則必須將 Docker 設定為使用相同的 Proxy 伺服器。
- 如果您的 Docker 映像檔儲存在 Amazon ECR 私有登錄中，則必須將權杖交換服務元件作為 Docker 容器元件的相依性包含在 Docker 容器元件中。此外，Greengrass 裝置角色必須允許`ecr:GetAuthorizationToken`、和`ecr:GetDownloadUrlForLayer`動作`ecr:BatchGetImage`，如下列範例 IAM 政策所示。

  ```
  {
    "Version": "2012-10-17",
    "Statement": [
      {
        "Action": [
          "ecr:GetAuthorizationToken",
          "ecr:BatchGetImage",
          "ecr:GetDownloadUrlForLayer"
  ```

```
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

- docker 應用程序管理器組件支持在 VPC 中運行。若要在 VPC 中部署此元件，需要下列項目。
  - docker 應用程序管理器組件必須具有下載圖像的連接性。例如，如果您使用 ECR，則必須具有與下列端點的連線能力。
    - `*.dkr.ecr.region.amazonaws.com`(VPC 端端點`com.amazonaws.region.ecr.dkr`)
    - `api.ecr.region.amazonaws.com`(VPC 端端點`com.amazonaws.region.ecr.api`)

端點和連接埠

除了基本作業所需的端點和連接埠之外，此元件還必須能夠對下列端點和連接埠執行輸出要求。如需詳細資訊，請參閱 允許裝置流量透過 Proxy 或防火牆。

端點	連線埠	必要	描述
`ecr.region.amazonaws.com`	443	否	如果您從 Amazon ECR 下載碼頭圖像，則需要此選項。

`hub.docker.com`
 	
`registry.hub.docker.com/v1`
	443	否	如果您從 Docker 集線器下載 Docker 映像，則需要此選項。

相依性

部署元件時，AWS IoT Greengrass也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的需求，才能成功部署元件。本節列出此元件之已發行版本的相依性，以及定義每個相依性之元件版本的語意版本條件約束。您也可以在AWS IoT Greengrass主控台中檢視元件每個版本的相依性。在元件詳細資料頁面上，尋找 [相依性] 清單。

2.0.11

下表列出此元件 2.0.11 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.1.0	軟式

2.0.10

下表列出此元件 2.0.10 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.1.0	軟式

2.0.9

下表列出此元件 2.0.9 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.1.0	軟式

2.0.8

下表列出此元件 2.0.8 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.1.0	軟式

2.0.7

下表列出此元件 2.0.7 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.1.0	軟式

2.0.6

下表列出此元件 2.0.6 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.1.0	軟式

2.0.5

下表列出此元件 2.0.5 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.1.0	軟式

2.0.4

下表列出此元件 2.0.4 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.1.0	軟式

2.0.3

下表列出此元件 2.0.3 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.1.0	軟式

2.0.2

下表列出此元件 2.0.2 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.1.0	軟式

2.0.1

下表列出此元件 2.0.1 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.1.0	軟式

2.0.0

下表列出此元件 2.0.0 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.1.0	軟式

如需有關元件相依性的詳細資訊，請參閱元件方案參考。

組態

此元件沒有任何組態參數。

本機記錄檔

此元件使用與 Greengrass 核元件相同的記錄檔。

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

若要檢視此元件的記錄

- 在核心裝置上執行下列命令，即時檢視此元件的記錄檔。以AWS IoT Greengrass根資料夾的路徑取代*/greengrass/v2*或 *C:\greengrass\v2*。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

變更記錄

下表說明元件每個版本中的變更。

版本	變更
2.0.11	版本更新了 Greengrass 核 2.12.0 版本釋放。
2.0.10	版本更新了 Greengrass 核 2.11.0 版本釋放。
2.0.9	版本更新了 Greengrass 2.10.0 版本。
2.0.8	版本更新 Greengrass 2.9.0 版本釋放。

版本	變更
2.0.7	版本更新 Greengrass 2.8.0 版本的版本。
2.0.6	版本更新了 Greengrass 核 2.7.0 版本釋放。
2.0.5	版本更新 Greengrass 2.6.0 版本的版本。
2.0.4	版本更新了 Greengrass 核 2.5.0 版本。
2.0.3	版本更新 Greengrass 2.4.0 版本的版本。
2.0.2	版本更新了 Greengrass 核 2.3.0 版本。
2.0.1	版本更新了 Greengrass 核 2.2.0 版本。
2.0.0	初始版本。

另請參閱

- 運行碼頭容器

Kinesis Video Streams 的邊緣連接器

Kinesis Video Streams 元件 (`aws.iot.EdgeConnectorForKVS`) 的邊緣連接器會從本機攝影機讀取視訊饋送，並將串流發佈到 Kinesis Video Streams。您可以將此元件設定為使用即時串流通訊協定 (RTSP) 從網際網路通訊協定 (IP) 攝影機讀取視訊饋送。然後，您可以在 Amazon 受管的 Grafana 或本機 Grafana 伺服器中設定儀表板，以監控影片串流並與之互動。

您可以將此元件與整合，以AWS IoT TwinMaker 便在 Grafana 儀表板中顯示和控制視訊串流。AWS IoT TwinMaker 是一項AWS服務，可讓您建立實體系統的運作數位雙胞胎。您可以使用視覺化AWS IoT TwinMaker 來自感測器、攝影機和企業應用程式的資料，以便追蹤實體工廠、建築物或工業廠房。您也可以使用此資料來監視作業、診斷錯誤及修復錯誤。如需詳細資訊，請參閱《AWS IoT TwinMaker 使用者指南》中的什麼是 AWS IoT TwinMaker ？。

此元件會將其組態儲存在中AWS IoT SiteWise，這是一項用於建立工業資料模型和儲存的AWS服務。在中AWS IoT SiteWise，資產代表物件，例如裝置、設備或其他物件群組。若要設定和使用此元件，您可以為每個 Greengrass 核心裝置以及連接到每個核心裝置的每個 IP 攝影機建立AWS IoT SiteWise 資產。每個資產都有您設定用來控制功能的屬性，例如即時串流、隨選上傳和本機快取。若要指定每個

攝影機的 URL，請在中建立包AWS Secrets Manager含攝影機 URL 的密碼。如果攝影機需要驗證，您也可以在 URL 中指定使用者名稱和密碼。然後，您可以在 IP 攝影機的資產屬性中指定該密碼。

此元件會將每個攝影機的視訊串流上傳至 Kinesis 視訊串流。您可以在每個攝影機的AWS IoT SiteWise資產組態中指定目的地 Kinesis 視訊串流的名稱。如果 Kinesis 視訊串流不存在，此元件會為您建立它。

AWS IoT TwinMaker 提供您可以執行的指令碼來建立這些AWS IoT SiteWise資產和 Secrets Manager密碼。如需有關如何建立這些資源以及如何安裝、設定和使用此元件的詳細資訊，請參閱《使AWS IoT TwinMaker 用指南》中的AWS IoT TwinMaker 視訊整合。

> **Note**
>
> Kinesis Video Streams 元件的邊緣連接器僅適用於下列項目：AWS 區域
>
> - 美國東部 (維吉尼亞北部)
> - 美國西部 (奧勒岡)
> - 歐洲 (法蘭克福)
> - 歐洲 (愛爾蘭)
> - 亞太區域 (新加坡)

主題

版本

此元件具有下列版本：

- 1.0.x

Type

此元件是一般元件 (`aws.greengrass.generic`)。Greengrass 核會執行元件的生命週期指令碼。

如需詳細資訊，請參閱 元件類型。

作業系統

此元件只能安裝在 Linux 核心裝置上。

要求

此元件具有下列需求：

- 您只能將此元件部署到單一核心裝置，因為每個核心裝置的元件組態必須是唯一的。您無法將此元件部署到核心裝置群組。
- 安裝在核心設備上的 GStreamer 1.18.4 或更高版本。如需詳細資訊，請參閱安裝 GStreamer。

  在使用的設備上`apt`，您可以運行以下命令來安裝 GStreamer。

  ```
  sudo apt install -y libgstreamer1.0-dev libgstreamer-plugins-base1.0-dev
   gstreamer1.0-plugins-base-apps
  sudo apt install -y gstreamer1.0-libav
  sudo apt install -y gstreamer1.0-plugins-bad gstreamer1.0-plugins-good gstreamer1.0-
  plugins-ugly gstreamer1.0-tools
  ```

- 每個核心裝置的AWS IoT SiteWise資產。此AWS IoT SiteWise資產代表核心裝置。如需有關如何建立此資產的詳細資訊，請參閱AWS IoT TwinMaker 使用指南中的AWS IoT TwinMaker 視訊整合。
- 您連接到每個核心設備的每個 IP 攝像機的AWS IoT SiteWise資產。這些AWS IoT SiteWise資產代表將視訊串流至每個核心裝置的攝影機。每個攝影機的資產都必須與連線至相機的核心裝置相關聯。攝影機資產具有可讓您設定為指定 Kinesis 視訊串流、驗證密碼和視訊串流參數的內容。如需有關如何建立和設定攝影機資產的詳細資訊，請參閱AWS IoT TwinMaker 使用指南中的AWS IoT TwinMaker 視訊整合。

- 每個 IP 攝像機的AWS Secrets Manager秘密。這個密碼必須定義一個鍵值對，其中鍵是RTSPStreamUrl，並且該值是攝像機的 URL。如果相機需要驗證，請在此 URL 中包含使用者名稱和密碼。當您建立此元件所需的資源時，您可以使用指令碼來建立密碼。如需詳細資訊，請參閱AWS IoT TwinMaker 使用指南中的AWS IoT TwinMaker 視訊整合。

  您也可以使用 Secrets Manager 主控台和 API 來建立其他密碼。若要取得更多資訊，請參閱《AWS Secrets Manager使用指南》中的〈建立密碼〉。

- Greengrass 權杖交換角色必須允許下列AWS Secrets ManagerAWS IoT SiteWise、和 Kinesis Video Streams 動作，如下列範例 IAM 政策所示。

> **Note**
>
> 此範例原則允許裝置取得名為**IPCamera1Url**和的密碼值**IPCamera2Url**。設定每個 IP 攝影機時，您要指定包含該攝影機 URL 的密碼。如果攝影機需要驗證，您也可以在 URL 中指定使用者名稱和密碼。核心設備的令牌交換角色必須允許訪問每個 IP 攝像機連接的密碼。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "secretsmanager:GetSecretValue"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:secretsmanager:region:account-id:secret:IPCamera1Url",
        "arn:aws:secretsmanager:region:account-id:secret:IPCamera2Url"
      ]
    },
    {
      "Action": [
        "iotsitewise:BatchPutAssetPropertyValue",
        "iotsitewise:DescribeAsset",
        "iotsitewise:DescribeAssetModel",
        "iotsitewise:DescribeAssetProperty",
        "iotsitewise:GetAssetPropertyValue",
        "iotsitewise:ListAssetRelationships",
        "iotsitewise:ListAssets",
        "iotsitewise:ListAssociatedAssets",
```

```
        "kinesisvideo:CreateStream",
        "kinesisvideo:DescribeStream",
        "kinesisvideo:GetDataEndpoint",
        "kinesisvideo:PutMedia",
        "kinesisvideo:TagStream"
      ],
      "Effect": "Allow",
      "Resource": [
        "*"
      ]
    }
  ]
}
```

> **Note**
>
> 如果您使用客戶管理的AWS Key Management Service金鑰來加密密碼，則裝置角色也必須允許該`kms:Decrypt`處理行動。

端點和連接埠

除了基本作業所需的端點和連接埠之外，此元件還必須能夠對下列端點和連接埠執行輸出要求。如需詳細資訊，請參閱 允許裝置流量透過 Proxy 或防火牆。

端點	連線埠	必要	描述
`kinesisvideo.` *`region`*`.amazonaws.com`	443	是	將資料上傳至 Kinesis Video Streams。
`data.iotsitewise.` *`region`*`.amazonaws.com`	443	是	將視訊串流中繼資料發佈至 AWS IoT SiteWise.
`secretsmanager.` *`region`*`.amazonaws.com`	443	是	將相機 URL 秘密

端點	連線埠	必要	描述
			下載到核心設備。

相依性

部署元件時，AWS IoT Greengrass也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的需求，才能成功部署元件。本節列出此元件之已發行版本的相依性，以及定義每個相依性之元件版本的語意版本條件約束。您也可以在AWS IoT Greengrass主控台中檢視元件每個版本的相依性。在元件詳細資料頁面上，尋找 [相依性] 清單。

下表列出此元件 1.0.0 至 1.0.4 版的相依性。

相依性	兼容版本	相依性類型
代幣交換服務	>=2.0.3	硬式
流管理器	>=2.0.9	硬式

如需有關元件相依性的詳細資訊，請參閱元件方案參考。

組態

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

`SiteWiseAssetIdForHub`

代表此核心裝置的AWS IoT SiteWise資產 ID。如需有關如何建立此資產並使用該資產與此元件互動的詳細資訊，請參閱使AWS IoT TwinMaker 用指南中的AWS IoT TwinMaker 視訊整合。

Example 範例：組態合併更新

```
{
  "SiteWiseAssetIdForHub": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
}
```

授權

此元件包括下列協力廠商軟體/授權：

- 石英Job 計程序 /Apache 許可證 2.0
- Java 綁定為 GStreamer 1.x /GNU 較寬鬆的通用公共許可證

用量

若要設定此元件並與之互動，您可以在代表核心裝置的AWS IoT SiteWise資產以及其連線的 IP 攝影機上設定屬性。您也可以透過 Grafana 儀表板視覺化視訊串流並與之互動。AWS IoT TwinMaker如需詳細資訊，請參閱AWS IoT TwinMaker 使用指南中的AWS IoT TwinMaker 視訊整合。

本機記錄檔

此元件使用下列記錄檔。

```
/greengrass/v2/logs/aws.iot.EdgeConnectorForKVS.log
```

若要檢視此元件的記錄

- 在核心裝置上執行下列命令，即時檢視此元件的記錄檔。以AWS IoT Greengrass根資料夾的路徑取*/greengrass/v2*代。

  ```
  sudo tail -f /greengrass/v2/logs/aws.iot.EdgeConnectorForKVS.log
  ```

變更記錄

下表說明元件每個版本中的變更。

版本	變更
1.0.4	錯誤修復和改進 • 修正造成即時上傳停止的問題。
1.0.3	一般錯誤修正與改進。
1.0.1	一般錯誤修正與改進。

版本	變更
1.0.0	初始版本。

另請參閱

- 《AWS IoT TwinMaker 使用者指南》中的什麼是 AWS IoT TwinMaker？
- AWS IoT TwinMaker AWS IoT TwinMaker 使用者指南中的視訊整合
- 《AWS IoT SiteWise 使用者指南》中的什麼是 AWS IoT SiteWise？
- 更新《AWS IoT SiteWise使用指南》中的屬性值
- 《AWS Secrets Manager 使用者指南》中的什麼是 AWS Secrets Manager？
- 在AWS Secrets Manager使用者指南中建立和管理密碼

Greengrass

Greengrass CLI 元件 (`aws.greengrass.Cli`) 提供本機命令列介面，您可以在核心裝置上使用，在本機開發和偵錯元件。例如，Greengrass CLI 可讓您建立本機部署，並重新啟動核心裝置上的元件。

您可以在安裝 AWS IoT Greengrass Core 軟體時安裝此元件。如需詳細資訊，請參閱 教學課程：AWS IoT Greengrass V2 入門。

> **Important**
>
> 我們建議您僅在開發環境中使用此元件，而不是在生產環境中使用。此元件可讓您存取通常在生產環境中不需要的資訊和作業。只將此元件部署到您需要的核心裝置，以遵循最低權限原則。

安裝此元件之後，請執行下列命令以檢視其說明文件。安裝此組件時，它會在文件*/greengrass/v2*/bin夾`greengrass-cli`中添加一個符號鏈接。您可以從此路徑運行 Greengrass CLI，或將其添加到`PATH`環境變量中以在沒有絕對路徑的`greengrass-cli`情況下運行。

Linux or Unix

```
/greengrass/v2/bin/greengrass-cli help
```

Windows

```
C:\greengrass\v2\bin\greengrass-cli help
```

例如com.example.HelloWorld，下列指令會重新啟動名為的元件。

Linux or Unix

```
sudo /greengrass/v2/bin/greengrass-cli component restart --names
 "com.example.HelloWorld"
```

Windows

```
C:\greengrass\v2\bin\greengrass-cli component restart --names
 "com.example.HelloWorld"
```

如需詳細資訊，請參閱 綠色命令行界面。

主題

- 版本
- Type
- 作業系統
- 要求
- 相依性
- 組態
- 本機記錄檔
- 變更記錄

版本

此元件具有下列版本：

- 2.12.x
- 2.11.x
- 2.10.x

- 2.9.x
- 2.8.x
- 2.7.x
- 2.6.x
- 2.5.x
- 2.4.x 版本
- 2.3.x 版本
- 2.2.x 版本
- 2.1.x
- 2.0.x

Type

這個組件是一個插件組件（`aws.greengrass.plugin`）。Greengrass 核在與核相同的 Java 虛擬機（JVM）中運行此組件。當您在核心裝置上變更此元件的版本時，核心會重新啟動。

此組件使用與 Greengrass 核相同的日誌文件。如需詳細資訊，請參閱 監控AWS IoT Greengrass日誌。

如需詳細資訊，請參閱 元件類型。

作業系統

此元件可安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件具有下列需求：

- 您必須獲得授權才能使用 Greengrass CLI 與核心軟體進行互動 AWS IoT Greengrass 。執行下列其中一項動作以使用 Greengrass CLI：
  - 使用執行 AWS IoT Greengrass Core 軟體的系統使用者。

- 使用具有 root 權限或管理權限的使用者。在 Linux 核心裝置上，您可以使用sudo來取得根權限。
- 部署元件時，請使用位於AuthorizedPosixGroups或組AuthorizedWindowsGroups態參數中指定之群組中的系統使用者。如需詳細資訊，請參閱 Greengrass CLI 元件組態。

- 支援在虛擬私人雲端中執行的 CLI Greengrass 件。

相依性

部署元件時， AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的需求，才能成功部署元件。本節列出此元件之已發行版本的相依性，以及定義每個相依性之元件版本的語意版本條件約束。您也可以在AWS IoT Greengrass 主控台中檢視元件每個版本的相依性。在元件詳細資料頁面上，尋找 [相依性] 清單。

2.12.0 – 2.12.4

下表列出此元件 2.12.0 到 2.12.4 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.12.0	軟式

2.11.0 – 2.11.3

下表列出此元件 2.11.0 到 2.11.3 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.11.0	軟式

2.10.0 – 2.10.3

下表列出此元件 2.10.0 到 2.10.3 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.5.0	軟式

2.9.0 – 2.9.6

下表列出此元件 2.9.0 到 2.9.6 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.5.0	軟式

2.8.0 – 2.8.1

下表列出此元件 2.8.0 和 2.8.1 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.5.0	軟式

2.7.0

下表列出此元件 2.7.0 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.5.0	軟式

2.6.0

下表列出此元件 2.6.0 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.5.0	軟式

2.5.0 – 2.5.6

下表列出此元件 2.5.0 到 2.5.6 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.5.0	軟式

2.4.0

下表列出此元件 2.4.0 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.1.0	軟式

2.3.0

下表列出此元件 2.3.0 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.1.0	軟式

2.2.0

下表列出此元件 2.2.0 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.1.0	軟式

2.1.0

下表列出此元件 2.1.0 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.1.0	軟式

2.0.x

下表列出此元件 2.0.x 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

> **Note**
>
> Greengrass 核的最低相容版本對應於 Greengrass CLI 元件的修補程式版本。

如需有關元件相依性的詳細資訊，請參閱元件方案參考。

組態

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

2.5.x

`AuthorizedPosixGroups`

(選擇性) 包含以逗號分隔的系統群組清單的字串。您授權這些系統群組使用 Greengrass CLI 與核心軟體互動。 AWS IoT Greengrass 您可以指定群組名稱或群組 ID。例如，`group1,1002,group3`授權三個系統群組 (`group1`、`1002`和`group3`) 使用 Greengrass CLI。

如果您沒有指定任何要授權的群組，您可以使用 Greengrass CLI 做為根使用者 (`sudo`) 或執行 Core 軟體的系統使用者。 AWS IoT Greengrass

`AuthorizedWindowsGroups`

(選擇性) 包含以逗號分隔的系統群組清單的字串。您授權這些系統群組使用 Greengrass CLI 與核心軟體互動。 AWS IoT Greengrass 您可以指定群組名稱或群組 ID。例如，`group1,1002,group3`授權三個系統群組 (`group1`、`1002`和`group3`) 使用 Greengrass CLI。

如果您未指定任何要授權的群組，您可以使用 Greengrass CLI 做為系統管理員或執行 Core 軟體的系統使用者。 AWS IoT Greengrass

Example 範例：組態合併更新

下列範例組態指定要授權三個 POSIX 系統群組 (group11002、和group3) 和兩個 Windows 使用者群組 (Device Operators和QA Engineers) 來使用 Greengrass CLI。

```
{
  "AuthorizedPosixGroups": "group1,1002,group3",
  "AuthorizedWindowsGroups": "Device Operators,QA Engineers"
}
```

2.4.x - 2.0.x

`AuthorizedPosixGroups`

(選擇性) 包含以逗號分隔的系統群組清單的字串。您授權這些系統群組使用 Greengrass CLI 與核心軟體互動。 AWS IoT Greengrass 您可以指定群組名稱或群組 ID。例如，`group1,1002,group3`授權三個系統群組 (group11002、和group3) 使用 Greengrass CLI。

如果您沒有指定任何要授權的群組，您可以使用 Greengrass CLI 做為根使用者 (sudo) 或執行 Core 軟體的系統使用者。 AWS IoT Greengrass

Example 範例：組態合併更新

下列範例組態指定要授權三個系統群組 (group11002、和group3) 使用 Greengrass CLI。

```
{
  "AuthorizedPosixGroups": "group1,1002,group3"
}
```

本機記錄檔

此元件使用與 Greengrass 核元件相同的記錄檔。

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

若要檢視此元件的記錄

- 在核心裝置上執行下列命令，即時檢視此元件的記錄檔。以 AWS IoT Greengrass 根資料夾的路徑取代*/greengrass/v2*或 *C:\greengrass\v2*。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

變更記錄

下表說明元件每個版本中的變更。

版本	變更
2.12.4	版本更新 Greengrass 2.12.4 版本的版本。
2.12.3	**Warning** 此版本已不再提供。此版本中的改進功能在此元件的更新版本中提供。 版本更新 Greengrass 2.12.3 版本的版本。
2.12.2	版本更新 Greengrass 2.12.2 版本的版本。
2.12.1	版本更新 Greengrass 2.12.1 版本的版本。
2.12.0	版本更新了 Greengrass 核 2.12.0 版本釋放。

版本	變更
2.11.3	版本更新 Greengrass 2.11.3 版本釋放。
2.11.2	版本更新 Greengrass 2.11.2 版本發布。
2.11.1	版本更新 Greengrass 2.11.1 版本的版本。
2.11.0	新功能 • 可讓您取消本機部署。 • 可讓您設定本機部署的失敗處理原則。 • 改進了詳細的部署狀態報告。
2.10.3	版本更新了 Greengrass 核 2.10.3 版本發布。
2.10.2	版本更新 Greengrass 2.10.2 版本的版本。
2.10.1	版本更新 Greengrass 2.10.1 版本的版本。
2.10.0	版本更新了 Greengrass 核 2.10.0 版本。
2.9.6	版本更新 Greengrass 2.9.6 版本的版本。
2.9.5	版本更新了 Greengrass 核 2.9.5 版本發布。
2.9.4	版本更新 Greengrass 2.9.4 版本的版本。
2.9.3	版本更新 Greengrass 2.9.3 版本的版本。
2.9.2	版本更新 Greengrass 2.9.2 版本的版本。
2.9.1	版本更新 Greengrass 2.9.1 版本的版本。
2.9.0	版本更新 Greengrass 2.9.0 版本釋放。
2.8.1	版本更新 Greengrass 2.8.1 版本的版本。
2.8.0	版本更新 Greengrass 2.8.0 版本的版本。
2.7.0	版本更新了 Greengrass 核 2.7.0 版本釋放。

版本	變更
2.6.0	新功能 • 添加對自定義組件的支持，以調用 Greengrass CLI 使用的進程間通信（IPC）操作。您可以使用這些 IPC 作業來管理本機部署、檢視元件詳細資料，以及產生可用來登入本機除錯主控台的密碼。如需詳細資訊，請參閱 IPC：管理本機部署和元件。 錯誤修復和改進 • 其他小修正和改進。
2.5.6	版本更新了 Greengrass 核 2.5.6 版本。
2.5.5	版本更新 Greengrass 2.5.5 版本的版本。
2.5.4	版本更新 Greengrass 2.5.4 版本發布。
2.5.3	版本更新 Greengrass 2.5.3 版本的版本。
2.5.2	版本更新 Greengrass 2.5.2 版本的版本。
2.5.1	版本更新了 Greengrass 核 2.5.1 版本。
2.5.0	新功能 • 添加對運行 Windows 的核心設備的支持。 • 新增可指定的新組`AuthorizedWindowsGroups` 態參數，以授權系統群組在 Windows 裝置上使用 Greengrass CLI。 • 為本機部署新增`windowsUser` 參數。您可以使用此參數指定用來在 Windows 核心裝置上執行元件的使用者。
2.4.0	新功能 • 添加對系統資源限制的支援。建立本機部署時，您可以設定每個元件的處理序可在核心裝置上使用的 CPU 和 RAM 使用量上限。若要取得更多資訊，請參閱設定元件的系統資源限制和部署建立指令。
2.3.0	版本更新了 Greengrass 核 2.3.0 版本。
2.2.0	版本更新了 Greengrass 核 2.2.0 版本。

版本	變更
2.1.0	版本更新 Greengrass 2.1.0 版本發布。
2.0.5	版本更新了 Greengrass 核 2.0.5 版本。
2.0.4	版本更新了 Greengrass 核 2.0.4 版本發布。
2.0.3	初始版本。

IP 偵測器

IP 偵測器元件 (`aws.greengrass.clientdevices.IPDetector`) 會執行下列作業：

- 監控 Greengrass 核心裝置的網路連線資訊。此資訊包括核心裝置的網路端點，以及 MQTT 代理程式運作的連接埠。
- 在 AWS IoT Greengrass 雲服務中更新核心設備的連接信息。

用戶端裝置可以使用 Greengrass 雲端探索來擷取關聯核心裝置的連線資訊。然後，用戶端裝置可以嘗試連線到每個核心裝置，直到它們成功連線為止。

> **Note**
>
> 用戶端裝置是連線到 Greengrass 核心裝置以傳送 MQTT 訊息和資料進行處理的本機 IoT 裝置。如需詳細資訊，請參閱 與本機 IoT 裝置互動。

IP 偵測器元件會以偵測到的資訊取代核心裝置的現有連線資訊。由於此元件會移除現有資訊，因此您可以使用 IP 偵測器元件，或手動管理連線資訊。

> **Note**
>
> IP 偵測器元件僅偵測 IPv4 位址。

主題

- 版本

版本

此元件具有下列版本：

- 2.1.x
- 2.0.x

Type

這個組件是一個插件組件（`aws.greengrass.plugin`）。Greengrass 核在與核相同的 Java 虛擬機（JVM）中運行此組件。當您在核心裝置上變更此元件的版本時，核心會重新啟動。

此組件使用與 Greengrass 核相同的日誌文件。如需詳細資訊，請參閱 監控AWS IoT Greengrass日誌。

如需詳細資訊，請參閱 元件類型。

作業系統

此元件可安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件具有下列需求：

- Greengrass 服務角色必須與您的相關聯， AWS 帳戶 並允許和權限。`iot:GetThingShadow` `iot:UpdateThingShadow`
- 核心裝置的 AWS IoT 政策必須允許`greengrass:UpdateConnectivityInfo`權限。如需詳細資訊，請參閱 資料平面操作的AWS IoT 政策 及 支援用戶端裝置的最低AWS IoT原則。
- 如果您將核心裝置的 MQTT 代理程式元件設定為使用預設連接埠 8883 以外的連接埠，則必須使用 IP 偵測器 v2.1.0 或更新版本。將其設定為報告代理程式運作所在的連接埠。
- 如果您有複雜的網路設定，IP 偵測器元件可能無法識別用戶端裝置可連線至核心裝置的端點。如果 IP 偵測器元件無法管理端點，您必須改為手動管理核心裝置端點。例如，如果核心裝置位於將 MQTT 代理程式連接埠轉送至該路由器的後方，則必須將路由器的 IP 位址指定為核心裝置的端點。如需詳細資訊，請參閱 管理核心裝置端點。
- 支援 IP 偵測器元件在 VPC 中執行。

相依性

部署元件時， AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的需求，才能成功部署元件。本節列出此元件之已發行版本的相依性，以及定義每個相依性之元件版本的語意版本條件約束。您也可以在AWS IoT Greengrass 主控台中檢視元件每個版本的相依性。在元件詳細資料頁面上，尋找 [相依性] 清單。

2.1.8 – 2.1.9

下表列出此元件 2.1.8 和 2.1.9 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.2.0	軟式

2.1.7

下表列出此元件 2.1.7 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.2.0	軟式

2.1.6

下表列出此元件 2.1.6 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.2.0	軟式

2.1.5

下表列出此元件 2.1.5 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.2.0	軟式

2.1.4

下表列出此元件 2.1.4 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.2.0	軟式

2.1.3

下表列出此元件 2.1.3 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.2.0	軟式

2.1.2

下表列出此元件 2.1.2 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.2.0	軟式

2.1.1

下表列出此元件 2.1.1 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.2.0	軟式

2.1.0 and 2.0.2

下表列出此元件 2.1.0 和 2.0.2 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.2.0	軟式

2.0.1

下表列出此元件 2.0.1 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.2.0	軟式

2.0.0

下表列出此元件 2.0.0 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.2.0	軟式

如需有關元件相依性的詳細資訊，請參閱元件方案參考。

組態

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

2.1.x

`defaultPort`

(選擇性) 此元件偵測到 IP 位址時要報告的 MQTT 代理程式連接埠。如果您將 MQTT 代理程式設定為使用與預設連接埠 8883 不同的連接埠，則必須指定此參數。

預設：8883

`includeIPv4LoopbackAddrs`

(選擇性) 您可以啟用此選項來偵測和報告 IPv4 回送位址。這些是 IP 位址，例如`localhost`裝置可以與自身通訊的位置。在核心裝置和用戶端裝置在相同系統上執行的測試環境中使用此選項。

預設：`false`

`includeIPv4LinkLocalAddrs`

(選擇性) 您可以啟用此選項來偵測和報告 IPv4 連結本機位址。如果核心裝置的網路沒有動態主機設定通訊協定 (DHCP) 或靜態指派的 IP 位址，請使用此選項。

預設：`false`

2.0.x

`includeIPv4LoopbackAddrs`

(選擇性) 您可以啟用此選項來偵測和報告 IPv4 回送位址。這些是 IP 位址，例如`localhost`裝置可以與自身通訊的位置。在核心裝置和用戶端裝置在相同系統上執行的測試環境中使用此選項。

預設：`false`

`includeIPv4LinkLocalAddrs`

(選擇性) 您可以啟用此選項來偵測和報告 IPv4 連結本機位址。如果核心裝置的網路沒有動態主機設定通訊協定 (DHCP) 或靜態指派的 IP 位址，請使用此選項。

預設：`false`

本機記錄檔

此元件使用與 Greengrass 核元件相同的記錄檔。

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

若要檢視此元件的記錄

- 在核心裝置上執行下列命令，即時檢視此元件的記錄檔。以 AWS IoT Greengrass 根資料夾的路徑取代*/greengrass/v2*或 *C:\greengrass\v2*。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

變更記錄

下表說明元件每個版本的變更。

版本	變更
2.1.9	錯誤修復和改進 • 將取得的 IP 步驟調整為僅在偵錯記錄檔層級傳送記錄檔。
2.1.8	版本更新了 Greengrass 2.12.0 版本。

版本	變更
2.1.7	版本更新 Greengrass 2.11.0 版本釋放。
2.1.6	版本更新了 Greengrass 2.10.0 版本。
2.1.5	版本更新 Greengrass 2.9.0 版本釋放。
2.1.4	版本更新 Greengrass 2.8.0 版本的版本。
2.1.3	版本更新了 Greengrass 核 2.7.0 版本釋放。
2.1.2	錯誤修復和改進 • 改善此元件在特定情況下記錄的錯誤訊息。 • 版本更新 Greengrass 2.6.0 版本發布。
2.1.1	版本更新了 Greengrass 核 2.5.0 版本。
2.1.0	改善項目 • 新增`defaultPort` 參數，可讓您使用非預設 MQTT 代理程式連接埠。 • 更新以使日誌消息更清晰。
2.0.2	版本更新 Greengrass 2.4.0 版本的版本。
2.0.1	版本更新了 Greengrass 核 2.3.0 版本。
2.0.0	初始版本。

Firehose

Firehose 元件 (`aws.greengrass.KinesisFirehose`) 會透過 Amazon 資料 Firehose 交付串流將資料發佈到 Amazon S3、亞馬 Amazon Redshift 和 Amazon 服務等目的地。 OpenSearch 如需詳細資訊，請參閱什麼是 Amazon 資料 Firehose？ 在 Amazon 數據 Firehose 開發人員指南中。

若要使用此元件發佈到 Kinesis 傳遞串流，請將訊息發佈到此元件訂閱的主題。依預設，此元件會訂閱`kinesisfirehose/message`和`kinesisfirehose/message/binary/#`本機發佈/訂閱主題。您可以在部署此元件時指定其他主題，包括 AWS IoT Core MQTT 主題。

> **Note**
>
> 此元件提供與 AWS IoT Greengrass V1 中 Firehose 連接器類似的功能。如需詳細資訊，請參閱 AWS IoT Greengrass V1 開發人員指南中的 Firehose 連接器。

主題

版本

此元件具有下列版本：

- 2.1.x
- 2.0.x

Type

這個組件是一個 Lambda 組件（`aws.greengrass.lambda`）。Greengrass 核使用 Lambda 啟動器組件運行此組件的 Lambda 函數。

如需詳細資訊，請參閱 元件類型。

作業系統

此元件只能安裝在 Linux 核心裝置上。

要求

此元件具有下列需求：

- 您的核心裝置必須符合執行 Lambda 函數的需求。如果您希望核心裝置執行容器化 Lambda 函數，則該裝置必須符合要求才能執行此操作。如需詳細資訊，請參閱 Lambda 函數要求。
- Python 版本 3.7 安裝在核心設備上，並添加到 PATH 環境變量中。
- Greengrass 裝置角色必須允許`firehose:PutRecord`和動`firehose:PutRecordBatch`作，如下列 IAM 政策範例所示。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "firehose:PutRecord",
        "firehose:PutRecordBatch"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:firehose:region:account-id:deliverystream/stream-name"
      ]
    }
  ]
}
```

  您可以動態覆寫此元件的輸入訊息承載中的預設傳遞串流。如果您的應用程式使用此功能，IAM 政策必須包含所有目標串流作為資源。您可以為資源授予精細或條件式存取 (例如，使用萬用字元 * 命名機制)。
- 若要從此元件接收輸出資料，您必須在部署此元件時，合併舊版訂閱路由器元件的下列組態更新 (`aws.greengrass.LegacySubscriptionRouter`)。此配置指定此組件發布響應的主題。

  Legacy subscription router v2.1.x

```
{
  "subscriptions": {
    "aws-greengrass-kinesisfirehose": {
```

```
      "id": "aws-greengrass-kinesisfirehose",
      "source": "component:aws.greengrass.KinesisFirehose",
      "subject": "kinesisfirehose/message/status",
      "target": "cloud"
    }
  }
}
```

Legacy subscription router v2.0.x

```
{
  "subscriptions": {
    "aws-greengrass-kinesisfirehose": {
      "id": "aws-greengrass-kinesisfirehose",
      "source": "arn:aws:lambda:region:aws:function:aws-greengrass-
kinesisfirehose:version",
      "subject": "kinesisfirehose/message/status",
      "target": "cloud"
    }
  }
}
```

- 將##替換為您 AWS 區域 使用的區域。
- 將##取代為此元件執行的 Lambda 函數版本。若要尋找 Lambda 函數版本，您必須檢視要部署之此元件版本的方案。在AWS IoT Greengrass 主控台中開啟此元件的詳細資料頁面，然後尋找 Lambda 函數索引鍵值組。此鍵值對包含 Lambda 函數的名稱和版本。

> **Important**
>
> 每次部署此元件時，您都必須更新舊版訂閱路由器上的 Lambda 函數版本。這可確保您針對部署的元件版本使用正確的 Lambda 函數版本。

如需詳細資訊，請參閱 建立部署。

- 支援 Firehose 元件在 VPC 中執行。若要在 VPC 中部署此元件，需要下列項目。
  - Firehose 元件必須具有的 VPC 端點的連線能力。`firehose.region.amazonaws.com` `com.amazonaws.region.kinesis-firehose`

端點和連接埠

除了基本作業所需的端點和連接埠之外，此元件還必須能夠對下列端點和連接埠執行輸出要求。如需詳細資訊，請參閱 允許裝置流量透過 Proxy 或防火牆。

端點	連線埠	必要	描述
`firehose. region.amazonaws.com`	443	是	上載資料至 Firehose

相依性

部署元件時， AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的需求，才能成功部署元件。本節列出此元件之已發行版本的相依性，以及定義每個相依性之元件版本的語意版本條件約束。您也可以在AWS IoT Greengrass 主控台中檢視元件每個版本的相依性。在元件詳細資料頁面上，尋找 [相依性] 清單。

2.1.7

下表列出此元件 2.1.7 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	硬式
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
代幣交換服務	^2.0.0	硬式

2.1.6

下表列出此元件 2.1.6 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	硬式

相依性	兼容版本	相依性類型
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
代幣交換服務	^2.0.0	硬式

2.1.5

下表列出此元件 2.1.5 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	硬式
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
代幣交換服務	^2.0.0	硬式

2.1.4

下表列出此元件 2.1.4 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	硬式
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
代幣交換服務	^2.0.0	硬式

2.1.3

下表列出此元件 2.1.3 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	硬式
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
代幣交換服務	^2.0.0	硬式

2.1.2

下表列出此元件 2.1.2 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	硬式
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
代幣交換服務	^2.0.0	硬式

2.1.1

下表列出此元件 2.1.1 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	硬式
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
代幣交換服務	^2.0.0	硬式

2.0.8 - 2.1.0

下表列出此元件 2.0.8 和 2.1.0 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	硬式
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
代幣交換服務	^2.0.0	硬式

2.0.7

下表列出此元件 2.0.7 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	硬式
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
代幣交換服務	^2.0.0	硬式

2.0.6

下表列出此元件 2.0.6 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	硬式
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式

相依性	兼容版本	相依性類型
代幣交換服務	^2.0.0	硬式

2.0.5

下表列出此元件 2.0.5 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	硬式
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
代幣交換服務	^2.0.0	硬式

2.0.4

下表列出此元件 2.0.4 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	硬式
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
代幣交換服務	^2.0.0	硬式

2.0.3

下表列出此元件 2.0.3 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.3	硬式
发 Lambda 器	>	硬式
Lambda 執行階段	>	軟式
代幣交換服務	>	硬式

如需有關元件相依性的詳細資訊，請參閱元件方案參考。

組態

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

> **Note**
>
> 此元件的預設組態包括 Lambda 函數參數。建議您只編輯下列參數，以便在裝置上設定此元件。

`lambdaParams`

包含此元件 Lambda 函數之參數的物件。此物件包含下列資訊：

`EnvironmentVariables`

包含 Lambda 函數參數的物件。此物件包含下列資訊：

`DEFAULT_DELIVERY_STREAM_ARN`

元件在其中傳送資料的預設 Firehose 傳送串流的 ARN。您可以使用輸入消息有效負載中的`delivery_stream_arn`屬性覆蓋目標流。

> **Note**
>
> 核心裝置角色必須允許對所有目標傳遞串流執行必要的動作。如需詳細資訊，請參閱要求。

PUBLISH_INTERVAL

(選擇性) 元件將批次資料發佈至 Firehose 之前等待的秒數上限。若要設定元件在接收測量結果時發佈測量結果 (也就是說不進行批次處理)，請指定0。

這個值最多可以是 900 秒。

預設值：10 秒

DELIVERY_STREAM_QUEUE_SIZE

(選擇性) 元件拒絕相同傳遞串流的新記錄之前，要保留在記憶體中的記錄數目上限。

此值必須至少為 2,000 筆記錄。

預設值：5,000 筆記錄

containerMode

(選擇性) 此元件的容器化模式。您可以從以下選項中選擇：

- NoContainer— 元件不會在隔離的執行階段環境中執行。
- GreengrassContainer— 元件在 AWS IoT Greengrass 容器內的隔離執行階段環境中執行。

預設：GreengrassContainer

containerParams

(選擇性) 包含此元件之容器參數的物件。如果您指定GreengrassContainer為，則元件會使用這些參數containerMode。

此物件包含下列資訊：

memorySize

(選擇性) 配置給元件的記憶體容量 (以 KB 為單位)。

預設值為 64 MB

pubsubTopics

(選擇性) 包含元件訂閱以接收訊息之主題的物件。您可以指定每個主題，以及元件是否訂閱 MQTT 主題 AWS IoT Core 或本機發佈/訂閱主題。

此物件包含下列資訊：

0-這是一個數組索引作為一個字符串。

包含下列資訊的物件：

`type`

(選擇性) 此元件用來訂閱訊息的發佈/訂閱訊息類型。您可以從以下選項中選擇：

- PUB_SUB - 訂閱本機發佈/訂閱訊息。如果您選擇此選項，則主題不能包含 MQTT 萬用字元。如需指定此選項時如何從自訂元件傳送郵件的詳細資訊，請參閱發佈/訂閱本地訊息。
- IOT_CORE— 訂閱 AWS IoT Core MQTT 訊息。如果您選擇此選項，主題可以包含 MQTT 萬用字元。如需指定此選項時如何從自訂元件傳送郵件的詳細資訊，請參閱發布/訂閱 MQTT 訊 AWS IoT Core 息。

預設：PUB_SUB

`topic`

(選擇性) 元件訂閱接收訊息的主題。如果您指定`IotCore`為`type`，您可以在本主題中使用 MQTT 萬用字元 (+和#)。

Example 範例：組態合併更新 (容器模式)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "DEFAULT_DELIVERY_STREAM_ARN": "arn:aws:firehose:us-
west-2:123456789012:deliverystream/mystream"
    }
  },
  "containerMode": "GreengrassContainer"
}
```

Example 範例：組態合併更新 (無容器模式)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "DEFAULT_DELIVERY_STREAM_ARN": "arn:aws:firehose:us-
west-2:123456789012:deliverystream/mystream"
    }
  },
```

```
  "containerMode": "NoContainer"
}
```

輸入資料

此元件會接受下列主題的串流內容，並將內容傳送至目標傳送串流。該組件接受兩種類型的輸入數據：

- kinesisfirehose/message 主題上的 JSON 資料。
- kinesisfirehose/message/binary/# 主題上的二進位資料。

JSON 資料的預設主題 (本機發佈/訂閱)：kinesisfirehose/message

該消息接受以下屬性。輸入訊息必須為 JSON 格式。

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。

類型：object包含下列資訊：

data

要傳送到交付串流的資料。

類型：string

delivery_stream_arn

(選擇性) 目標「Firehose」傳送串流的 ARN。指定此屬性可覆寫預設交付串流。

類型：string

id

請求的任意 ID。使用此屬性可將輸入要求對應至輸出回應。當您指定此屬性時，組件會將回應物件中的id屬性設定為此值。

類型：string

Example 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"
}
```

二進位資料的預設主題 (本機發佈/訂閱)：kinesisfirehose/message/binary/#

使用此主題傳送包含二進位資料的訊息。組件不會剖析二進位資料。元件會依原樣串流資料。

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

依預設，此元件會將回應發佈為下列 MQTT 主題的輸出資料。您必須將此主題指定為舊版訂閱路由器元件的組態subject中的。如需如何在自訂元件中訂閱有關此主題之訊息的詳細資訊，請參閱發布/訂閱MQTT 訊 AWS IoT Core 息。

預設主題 (AWS IoT Core MQTT)：kinesisfirehose/message/status

Example 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**
>
> 如果元件偵測到可以重試的錯誤 (例如連線錯誤)，它會在下一個批次中重試發佈。

本機記錄檔

此元件會使用下列記錄檔。

```
/greengrass/v2/logs/aws.greengrass.KinesisFirehose.log
```

若要檢視此元件的記錄

- 在核心裝置上執行下列命令，即時檢視此元件的記錄檔。以 AWS IoT Greengrass 根資料夾的路徑取*/greengrass/v2*代。

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.KinesisFirehose.log
  ```

授權

此元件包括下列協力廠商軟體/授權：

- AWS SDK for Python (Boto3)/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

此元件是根據 Greengrass 核心軟體授權合約發行的。

變更記錄

下表說明元件每個版本中的變更。

版本	變更
2.1.7	版本更新了 Greengrass 核 2.12.0 版本釋放。
2.1.6	版本更新了 Greengrass 核 2.11.0 版本釋放。
2.1.5	版本更新了 Greengrass 核 2.10.0 版本釋放。
2.1.4	版本更新 Greengrass 2.9.0 版本釋放。
2.1.3	版本更新 Greengrass 2.8.0 版本的版本。
2.1.2	版本更新了 Greengrass 核 2.7.0 版本釋放。
2.1.1	版本更新 Greengrass 2.6.0 版本的版本。
2.1.0	新功能 • 添加對 HTTPS 網絡代理配置的支持。如需詳細資訊，請參閱 連線至連接埠 443 或透過網路代理 及 讓核心裝置信任 HTTPS 代理。
2.0.8	版本更新了 Greengrass 核 2.5.0 版本。
2.0.7	版本更新 Greengrass 2.4.0 版本的版本。
2.0.6	版本更新了 Greengrass 核 2.3.0 版本。
2.0.5	版本更新了 Greengrass 核 2.2.0 版本。
2.0.4	版本更新 Greengrass 2.1.0 版本發布。
2.0.3	初始版本。

另請參閱

- 什麼是 Amazon 數據 Firehose？ 在 Amazon 數據 Firehose 開發人員指南

发 Lambda 器

Lambda 啟動器元件 (`aws.greengrass.LambdaLauncher`) 會啟動和停止 AWS IoT Greengrass 核心裝置上的 AWS Lambda 功能。此元件也會設定任何容器化，並以您指定的使用者身分執行程序。

> **Note**
>
> 將 Lambda 函數元件部署到核心裝置時，部署也會包含此元件。如需詳細資訊，請參閱 執行 AWS Lambda函數。

主題

版本

此元件具有下列版本：

- 2.0.x

Type

此元件是一般元件 (`aws.greengrass.generic`)。Greengrass 核會執行元件的生命週期指令碼。

如需詳細資訊，請參閱 元件類型。

作業系統

此元件只能安裝在 Linux 核心裝置上。

要求

此元件具有下列需求：

- 您的核心裝置必須符合執行 Lambda 函數的需求。如果您希望核心裝置執行容器化 Lambda 函數，則該裝置必須符合要求才能執行此操作。如需詳細資訊，請參閱 Lambda 函數要求。
- 支援 Lambda 啟動器元件在 VPC 中執行。

相依性

部署元件時， AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的需求，才能成功部署元件。本節列出此元件之已發行版本的相依性，以及定義每個相依性之元件版本的語意版本條件約束。您也可以在AWS IoT Greengrass 主控台中檢視元件每個版本的相依性。在元件詳細資料頁面上，尋找 [相依性] 清單。

2.0.11 – 2.0.13

下表列出此元件 2.0.11 至 2.0.13 版的相依性。

相依性	兼容版本	相依性類型
Lambda 經理	> = 2.0.0	硬式

2.0.9 – 2.0.10

下表列出此元件 2.0.9 至 2.0.10 版的相依性。

相依性	兼容版本	相依性類型
Lambda 經理	> = 2.0.0	硬式

2.0.4 - 2.0.8

下表列出此元件 2.0.4 至 2.0.8 版的相依性。

相依性	兼容版本	相依性類型
Lambda 經理	> = 2.0.0	硬式

2.0.3

下表列出此元件 2.0.3 版的相依性。

相依性	兼容版本	相依性類型
Lambda 經理	> = 2.0.3	硬式

如需有關元件相依性的詳細資訊，請參閱元件方案參考。

組態

此元件沒有任何組態參數。

本機記錄檔

此元件使用下列記錄檔。

```
/greengrass/v2/logs/lambdaFunctionComponentName.log
```

若要檢視此元件的記錄

- 在核心裝置上執行下列命令，即時檢視此元件的記錄檔。*# AWS IoT Greengrass ####/greengrass/v2#####lambdaFunctionComponent######### Lambda ######## Name#*

  ```
  sudo tail -f /greengrass/v2/logs/lambdaFunctionComponentName.log
  ```

變更記錄

下表說明元件每個版本的變更。

版本	變更
2.0.13	錯誤修復和改進 一般錯誤修正與改進。
2.0.12	錯誤修復和改進 修正如果前一個程序未正確停止，Lambda 啟動器可能會擲回錯誤的問題。
2.0.11	Support Lambda 管理員 2.3.0。
2.0.10	錯誤修復和改進 • 一般錯誤修正與改進。
2.0.9	版本更新了 Greengrass 核 2.5.0 版本。
2.0.8	版本更新 Greengrass 2.4.0 版本的版本。
2.0.7	版本更新了 Greengrass 核 2.3.0 版本。
2.0.6	一般效能改進與錯誤修正。
2.0.4	錯誤修復和改進 • 修正元件未正確傳遞`AddGroupOwner` 至 Lambda 函數容器的問題。
2.0.3	初始版本。

Lambda 經理

Lambda 管理員元件 (`aws.greengrass.LambdaManager`) 會管理在 Greengrass 核心裝置上執行之 AWS Lambda 函數的工作項目和處理序間通訊。

> **Note**
>
> 將 Lambda 函數元件部署到核心裝置時，部署也會包含此元件。如需詳細資訊，請參閱 執行 AWS Lambda函數。

主題

版本

此元件具有下列版本：

- 2.3.x 版本
- 2.2.x 版本
- 2.1.x
- 2.0.x

作業系統

此元件只能安裝在 Linux 核心裝置上。

Type

這個組件是一個插件組件（`aws.greengrass.plugin`）。Greengrass 核在與核相同的 Java 虛擬機（JVM）中運行此組件。當您在核心裝置上變更此元件的版本時，核心會重新啟動。

此組件使用與 Greengrass 核相同的日誌文件。如需詳細資訊，請參閱 監控AWS IoT Greengrass日誌。

如需詳細資訊，請參閱 元件類型。

需求

此元件具有下列需求：

- 您的核心裝置必須符合執行 Lambda 函數的需求。如果您希望核心裝置執行容器化 Lambda 函數，則該裝置必須符合要求才能執行。如需詳細資訊，請參閱 Lambda 函數要求。
- 支援 Lambda 管理員元件在 VPC 中執行。

相依性

部署元件時， AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的需求，才能成功部署元件。本節列出此元件之已發行版本的相依性，以及定義每個相依性之元件版本的語意版本條件約束。您也可以在AWS IoT Greengrass 主控台中檢視元件每個版本的相依性。在元件詳細資料頁面上，尋找 [相依性] 清單。

2.3.2 and 2.3.3

下表列出此元件 2.3.2 和 2.3.3 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.2.10 and 2.3.1

下表列出此元件 2.2.10 和 2.3.1 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.2.8 and 2.2.9

下表列出此元件 2.2.8 和 2.2.9 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0 <2.11.0	軟式

2.2.7

下表列出此元件 2.2.7 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.2.6

下表列出此元件 2.2.6 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.2.5

下表列出此元件 2.2.5 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.2.4

下表列出此元件 2.2.4 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.2.1 - 2.2.3

下表列出此元件 2.2.1 至 2.2.3 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.0.0	軟式

2.2.0

下表列出此元件 2.2.0 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.5.0	軟式

2.1.3 and 2.1.4

下表列出此元件 2.1.3 和 2.1.4 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.2

下表列出此元件 2.1.2 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.1

下表列出此元件 2.1.1 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.0

下表列出此元件 2.1.0 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.0.x

下表列出此元件 2.0.x 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.3	軟式

如需有關元件相依性的詳細資訊，請參閱元件方案參考。

組態

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

`logHandlerMode`

> **Note**
> 僅適用於 lambda 管理器版本 2.3.0 以上

用來選擇要使用的 Lambda 記錄管理員實作。將值設定為以使`optimized`用較少的執行緒來讀取 lambda 記錄。

`getResultTimeoutInSecond`

(選擇性) Lambda 函數在逾時前可執行的時間上限 (以秒為單位)。

預設：`60`

本機記錄檔

此元件使用與 Greengrass 核元件相同的記錄檔。

```
/greengrass/v2/logs/greengrass.log
```

若要檢視此元件的記錄

- 在核心裝置上執行下列命令，即時檢視此元件的記錄檔。以 AWS IoT Greengrass 根資料夾的路徑取*/greengrass/v2*代。

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

變更記錄

下表說明元件每個版本的變更。

版本	變更
2.3.3	錯誤修復和改進 • 一般錯誤修正與改進。
2.3.2	版本更新 Greengrass 2.12.0 版本釋放。
2.3.1	錯誤修復和改進 • 調整某些錯誤的日誌級別。
2.3.0	新功能 • 日誌處理程序已優化以減少 CPU 負載。透過將組態選項設定為來使用此功`logHandlerMode` 能`optimized` 。 錯誤修復和改進 • 不再記錄完整堆疊追蹤`WorkQueueFullException` ，從而改善記錄檔和效能。 • 將 lambda 關機逾時設定為 15 秒至 300 秒，以防止關機逾時。 • 修正變更組態後，隨選 lambda 表示式可能無法重新啟動的問題。

版本	變更
2.2.11	錯誤修復和改進 • 修正 Lambda LegacySubscriptionRouter 組態變更時組態不會更新的問題。
2.2.10	版本更新 Greengrass 2.11.0 版本釋放。
2.2.9	錯誤修復和改進 修復了端口號由於時鐘偏斜而損壞的問題。
2.2.8	版本更新了 Greengrass 2.10.0 版本。
2.2.7	版本更新 Greengrass 2.9.0 版本釋放。
2.2.6	版本更新 Greengrass 2.8.0 版本的版本。
2.2.5	新功能 • 在您訂閱本機發佈/訂閱訊息的事件來源中新增 MQTT 主題萬用字元的支援。 此功能需要 v2.6.0 或更高版 Greeng rass 核成分。 • 版本更新了 Greengrass 核 2.7.0 版本釋放。
2.2.4	版本更新 Greengrass 2.6.0 版本發布。
2.2.3	錯誤修復和改進 • 修正 Lambda 函數的多個執行個體共用一個 cgroup 的問題。此元件使用 cgroup 來管理 Lambda 函數的資源使用量。
2.2.2	錯誤修復和改進 • 修正固定 Lambda 函數元件在特定情況下意外重新啟動的問題。
2.2.1	錯誤修復和改進 • 更改此組件的 Greengrass 核依賴版本約束以解決依賴關係解決問題。

版本	變更
2.2.0	錯誤修復和改進 • 修正 Lambda 函數在重新啟動後無法寫入記錄的問題。 • 修正當主題中有萬用字元時，舊版訂閱路由器會傳送重複訊息的問題。 • 修正非釘選 Lambda 函數無法在. AWS IoT Device SDK
2.1.4	錯誤修復和改進 • 修正導致 Lambda 函數使用 NodeJS 執行階段僅處理一則訊息的問題。 • 版本更新了 Greengrass 核 2.5.0 版本。
2.1.3	版本更新 Greengrass 2.4.0 版本的版本。
2.1.2	版本更新了 Greengrass 核 2.3.0 版本。
2.1.1	版本更新了 Greengrass 核 2.2.0 版本。
2.1.0	版本更新 Greengrass 2.1.0 版本發布。
2.0.3	初始版本。

Lambda 執行期

Lambda 執行階段元件 (`aws.greengrass.LambdaRuntimes`) 提供 Greengrass 核心裝置用來執行函數的執行階段。AWS Lambda

> **Note**
>
> 將 Lambda 函數元件部署到核心裝置時，部署也會包含此元件。如需詳細資訊，請參閱 執行 AWS Lambda函數。

主題

- 相依性
- 組態
- 本機記錄檔
- 變更記錄

版本

此元件具有下列版本：

- 2.0.x

Type

此元件是一般元件 (`aws.greengrass.generic`)。Greengrass 核會執行元件的生命週期指令碼。

如需詳細資訊，請參閱 元件類型。

作業系統

此元件只能安裝在 Linux 核心裝置上。

要求

此元件具有下列需求：

- 您的核心裝置必須符合執行 Lambda 函數的需求。如果您希望核心裝置執行容器化 Lambda 函數，則該裝置必須符合要求才能執行。如需詳細資訊，請參閱 Lambda 函數要求。
- 支援 Lambda 執行階段元件在 VPC 中執行。

相依性

這個組件沒有任何依賴關係。

組態

此元件沒有任何組態參數。

本機記錄檔

此元件不會輸出記錄檔。

變更記錄

下表說明元件每個版本的變更。

版本	變更
2.0.8	版本更新了 Greengrass 核 2.5.0 版本。
2.0.7	版本更新 Greengrass 2.4.0 版本的版本。
2.0.6	版本更新了 Greengrass 核 2.3.0 版本。
2.0.5	版本更新了 Greengrass 核 2.2.0 版本。
2.0.4	版本更新 Greengrass 2.1.0 版本發布。
2.0.3	初始版本。

舊版訂閱路由器

舊版訂閱路由器 (`aws.greengrass.LegacySubscriptionRouter`) 會管理 Greengrass 核心裝置上的訂閱。訂閱是 AWS IoT Greengrass V1 的一項功能，可定義 Lambda 函數可用於核心裝置上 MQTT 訊息的主題。如需詳細資訊，請參閱 AWS IoT GreengrassV1 開發人員指南中的 MQTT 訊息工作流程中的受管理訂閱。

您可以使用此元件為使用AWS IoT Greengrass核心 SDK 的連接器元件和 Lambda 函數元件啟用訂閱。

> **Note**
>
> 只有當 Lambda 函數使用 AWS IoT Greengrass Core SDK 中的函數時，才需要舊版訂閱路由器元件。`publish()`如果您更新 Lambda 函數程式碼以使用 AWS IoT Device SDK V2 中的處理序間通訊 (IPC) 介面，則不需要部署舊版訂閱路由器元件。如需詳細資訊，請參閱下列處理序間通訊服務：
>
> - 發佈/訂閱本地訊息
> - 發布/訂閱MQTT 訊 AWS IoT Core 息

主題

版本

此元件具有下列版本：

- 2.1.x
- 2.0.x

Type

此元件是一般元件 (`aws.greengrass.generic`)。Greengrass 核會執行元件的生命週期指令碼。

如需詳細資訊，請參閱 元件類型。

作業系統

此元件只能安裝在 Linux 核心裝置上。

要求

此元件具有下列需求：

- 支援舊版訂閱路由器在 VPC 中執行。

相依性

部署元件時，AWS IoT Greengrass也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的需求，才能成功部署元件。本節列出此元件之已發行版本的相依性，以及定義每個相依性之元件

版本的語意版本條件約束。您也可以在AWS IoT Greengrass主控台中檢視元件每個版本的相依性。在元件詳細資料頁面上，尋找 [相依性] 清單。

2.1.11

下表列出此元件 2.1.11 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.10

下表列出此元件 2.1.10 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.9

下表列出此元件 2.1.9 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.8

下表列出此元件 2.1.8 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.7

下表列出此元件 2.1.7 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.6

下表列出此元件 2.1.6 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.5

下表列出此元件 2.1.5 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.4

下表列出此元件 2.1.4 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.0.0	軟式

2.1.3

下表列出此元件 2.1.3 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.2

下表列出此元件 2.1.2 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.1

下表列出此元件 2.1.1 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.0

下表列出此元件 2.1.0 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.0.3

下表列出此元件 2.0.3 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.3	軟式

如需有關元件相依性的詳細資訊，請參閱元件方案參考。

組態

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

v2.1.x

`subscriptions`

(選擇性) 要在核心裝置上啟用的訂閱。這是一個物件，其中每個金鑰都是唯一識別碼，而且每個值都是定義該連接器訂閱的物件。當您部署 V1 連接器元件或使用AWS IoT Greengrass核心 SDK 的 Lambda 函數時，您必須設定訂閱。

每個訂閱物件都包含下列資訊：

`id`

此訂閱的唯一識別碼。此 ID 必須與此訂閱物件的金鑰相符。

`source`

Lambda 函數，可使用AWS IoT Greengrass核心開發套件，針對您在中指定的主題發佈 MQTT 訊息。`subject`請指定下列其中一項：

- 核心裝置上 Lambda 函數元件的名稱。使用`component:`前置詞指定元件名稱，例如**`component:com.example.HelloWorldLambda`**。
- 核心裝置上的 Lambda 函數的 Amazon 資源名稱 (ARN)。

> **Important**
>
> 如果 Lambda 函數的版本發生變更，您必須使用新版本的函數來設定訂閱。否則，在版本與訂閱匹配之前，此組件將不會路由消息。
> 您必須指定包含要匯入之函數版本的 Amazon 資源名稱 (ARN)。您不能使用 $LATEST 之類的版本別名。

若要部署 V1 連接器元件的訂閱，請指定元件的名稱或連接器元件的 Lambda 函數的 ARN。

`subject`

MQTT 主題或主題篩選器，來源和目標可以在其上發佈和接收訊息。此值支援+和主#題萬用字元。

`target`

針對您在中指定的主題接收 MQTT 訊息的目標。`subject`訂閱會指定該`source`函數將 MQTT 訊息發佈至核心裝置上的 Lambda 函數，AWS IoT Core或將 MQTT 訊息發佈至核心裝置上的 請指定下列其中一項：

- `cloud`。 `source`函數會將 MQTT 訊息發佈至。AWS IoT Core
- 核心裝置上 Lambda 函數元件的名稱。使用`component:`前置詞指定元件名稱，例如**`component:com.example.HelloWorldLambda`**。
- 核心裝置上的 Lambda 函數的 Amazon 資源名稱 (ARN)。

> **Important**
>
> 如果 Lambda 函數的版本發生變更，您必須使用新版本的函數來設定訂閱。否則，在版本與訂閱匹配之前，此組件將不會路由消息。
> 您必須指定包含要匯入之函數版本的 Amazon 資源名稱 (ARN)。您不能使用 `$LATEST` 之類的版本別名。

預設值：無訂閱

Example 範例組態更新 (定義訂閱AWS IoT Core)

下列範例會指定 `com.example.HelloWorldLambda` Lambda 函數元件將 MQTT 訊息發佈至AWS IoT Core主題。`hello/world`

```
{
  "subscriptions": {
    "Greengrass_HelloWorld_to_cloud": {
      "id": "Greengrass_HelloWorld_to_cloud",
      "source": "component:com.example.HelloWorldLambda",
      "subject": "hello/world",
      "target": "cloud"
    }
  }
}
```

Example 範例組態更新 (定義另一個 Lambda 函數的訂閱)

下列範例指定 `com.example.HelloWorldLambda` Lambda 函數元件將 MQTT 訊息發佈至 `com.example.MessageRelay` Lambda 函數元件的`hello/world`主題。

```
{
  "subscriptions": {
    "Greengrass_HelloWorld_to_MessageRelay": {
      "id": "Greengrass_HelloWorld_to_MessageRelay",
      "source": "component:com.example.HelloWorldLambda",
      "subject": "hello/world",
      "target": "component:com.example.MessageRelay"
    }
  }
}
```

v2.0.x

`subscriptions`

(選擇性) 要在核心裝置上啟用的訂閱。這是一個物件，其中每個金鑰都是唯一識別碼，而且每個值都是定義該連接器訂閱的物件。當您部署 V1 連接器元件或使用AWS IoT Greengrass核心 SDK 的 Lambda 函數時，您必須設定訂閱。

每個訂閱物件都包含下列資訊：

`id`

此訂閱的唯一識別碼。此 ID 必須與此訂閱物件的金鑰相符。

`source`

Lambda 函數，可使用AWS IoT Greengrass核心開發套件，針對您在中指定的主題發佈 MQTT 訊息。`subject`指定下列內容：

- 核心裝置上的 Lambda 函數的 Amazon 資源名稱 (ARN)。

> **Important**
>
> 如果 Lambda 函數的版本發生變更，您必須使用新版本的函數來設定訂閱。否則，在版本與訂閱匹配之前，此組件將不會路由消息。
> 您必須指定包含要匯入之函數版本的 Amazon 資源名稱 (ARN)。您不能使用 `$LATEST` 之類的版本別名。

若要部署 V1 連接器元件的訂閱，請指定連接器元件的 Lambda 函數的 ARN。

`subject`

MQTT 主題或主題篩選器，來源和目標可以在其上發佈和接收訊息。此值支援+和主#題萬用字元。

`target`

針對您在中指定的主題接收 MQTT 訊息的目標。`subject`訂閱會指定該`source`函數將 MQTT 訊息發佈至核心裝置上的 Lambda 函數，AWS IoT Core或將 MQTT 訊息發佈至核心裝置上的 請指定下列其中一項：

- `cloud`。 `source`函數會將 MQTT 訊息發佈至。AWS IoT Core
- 核心裝置上的 Lambda 函數的 Amazon 資源名稱 (ARN)。

> **Important**
>
> 如果 Lambda 函數的版本發生變更，您必須使用新版本的函數來設定訂閱。否則，在版本與訂閱匹配之前，此組件將不會路由消息。
> 您必須指定包含要匯入之函數版本的 Amazon 資源名稱 (ARN)。您不能使用 $LATEST 之類的版本別名。

預設值：無訂閱

Example 範例組態更新 (定義訂閱AWS IoT Core)

下列範例會指定`Greengrass_HelloWorld`函式將 MQTT 訊息發佈至AWS IoT Core主題。`hello/world`

```
"subscriptions": {
  "Greengrass_HelloWorld_to_cloud": {
    "id": "Greengrass_HelloWorld_to_cloud",
    "source": "arn:aws:lambda:us-
west-2:123456789012:function:Greengrass_HelloWorld:5",
    "subject": "hello/world",
    "target": "cloud"
  }
}
```

Example 範例組態更新 (定義另一個 Lambda 函數的訂閱)

下列範例會指定Greengrass_HelloWorld函數將 MQTT 訊息發佈至有Greengrass_MessageRelay關主題的hello/world。

```
"subscriptions": {
  "Greengrass_HelloWorld_to_MessageRelay": {
    "id": "Greengrass_HelloWorld_to_MessageRelay",
    "source": "arn:aws:lambda:us-
west-2:123456789012:function:Greengrass_HelloWorld:5",
    "subject": "hello/world",
    "target": "arn:aws:lambda:us-
west-2:123456789012:function:Greengrass_MessageRelay:5"
  }
}
```

本機記錄檔

此元件不會輸出記錄檔。

變更記錄

下表說明元件每個版本中的變更。

版本	變更
2.1.11	版本更新了 Greengrass 核 2.12.0 版本釋放。
2.1.10	版本更新了 Greengrass 核 2.11.0 版本釋放。
2.1.9	版本更新了 Greengrass 核 2.10.0 版本釋放。
2.1.8	版本更新 Greengrass 2.9.0 版本釋放。
2.1.7	版本更新 Greengrass 2.8.0 版本的版本。
2.1.6	版本更新了 Greengrass 核 2.7.0 版本釋放。
2.1.5	版本更新 Greengrass 2.6.0 版本的版本。
2.1.4	版本更新了 Greengrass 核 2.5.0 版本。

版本	變更
2.1.3	版本更新 Greengrass 2.4.0 版本的版本。
2.1.2	版本更新了 Greengrass 核 2.3.0 版本。
2.1.1	版本更新了 Greengrass 核 2.2.0 版本。
2.1.0	錯誤修復和改進 • 添加支援為`source`和`target`指定元件名稱而不是 ARN。如果您為訂閱指定元件名稱，則不需要在每次 Lambda 函數版本變更時重新設定訂閱。
2.0.3	初始版本。

本機除錯主控台

本機偵錯主控台元件 (`aws.greengrass.LocalDebugConsole`) 提供本機儀表板，可顯示 AWS IoT Greengrass 核心裝置及其元件的相關資訊。您可以使用此儀表板來偵錯核心裝置並管理本機元件。

> **Important**
>
> 我們建議您僅在開發環境中使用此元件，而不是在生產環境中使用。此元件可讓您存取通常在生產環境中不需要的資訊和作業。只將此元件部署到您需要的核心裝置，以遵循最低權限原則。

主題

- 變更記錄

版本

此元件具有下列版本：

- 2.3.x 版本
- 2.2.x 版本
- 2.1.x
- 2.0.x

Type

這個組件是一個插件組件（`aws.greengrass.plugin`）。Greengrass 核在與核相同的 Java 虛擬機（JVM）中運行此組件。當您在核心裝置上變更此元件的版本時，核心會重新啟動。

此組件使用與 Greengrass 核相同的日誌文件。如需詳細資訊，請參閱 監控AWS IoT Greengrass日誌。

如需詳細資訊，請參閱 元件類型。

作業系統

此元件可安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件具有下列需求：

- 您可以使用使用者名稱和密碼登入儀表板。使用者名稱`debug`，也就是為您提供的。您必須使用 AWS IoT Greengrass CLI 建立臨時密碼，以便透過核心裝置上的儀表板驗證您。您必須能夠使用 AWS IoT Greengrass CLI 來使用本機偵錯主控台。如需詳細資訊，請參閱 Greengrass CLI 需求。如需如何產生密碼和登入的詳細資訊，請參閱本機偵錯主控台元件使用情況。
- 支援在 VPC 中執行本機偵錯主控台元件。

相依性

部署元件時， AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的需求，才能成功部署元件。本節列出此元件之已發行版本的相依性，以及定義每個相依性之元件版本的語意版本條件約束。您也可以在AWS IoT Greengrass 主控台中檢視元件每個版本的相依性。在元件詳細資料頁面上，尋找 [相依性] 清單。

2.4.1 – 2.4.2

下表列出此元件 2.4.1 至 2.4.2 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.10.0	硬式
Greengrass CLI	> = 2.10.0	硬式

2.4.0

下表列出此元件 2.4.0 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.10.0	硬式
Greengrass CLI	> = 2.10.0	硬式

2.3.0 and 2.3.1

下表列出此元件 2.3.0 和 2.3.1 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.10.0	硬式
Greengrass CLI	> = 2.10.0	硬式

2.2.9

下表列出此元件 2.2.9 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.1.0	硬式
Greengrass CLI	> = 2.1.0	硬式

2.2.8

下表列出此元件 2.2.8 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.1.0	硬式
Greengrass CLI	> = 2.1.0	硬式

2.2.7

下表列出此元件 2.2.7 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.1.0	硬式
Greengrass CLI	> = 2.1.0	硬式

2.2.6

下表列出此元件 2.2.6 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.1.0	硬式

相依性	兼容版本	相依性類型
Greengrass CLI	> = 2.1.0	硬式

2.2.5

下表列出此元件 2.2.5 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.1.0	硬式
Greengrass CLI	> = 2.1.0	硬式

2.2.4

下表列出此元件 2.2.4 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.1.0	硬式
Greengrass CLI	> = 2.1.0	硬式

2.2.3

下表列出此元件 2.2.3 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.1.0	硬式
Greengrass CLI	>=2.1.0	硬式

2.2.2

下表列出此元件 2.2.2 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.1.0	硬式
Greengrass CLI	> = 2.1.0	硬式

2.2.1

下表列出此元件 2.2.1 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.1.0	硬式
Greengrass CLI	> = 2.1.0	硬式

2.2.0

下表列出此元件 2.2.0 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.1.0	硬式
Greengrass CLI	> = 2.1.0	硬式

2.1.0

下表列出此元件 2.1.0 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.1.0	硬式
Greengrass CLI	> = 2.1.0	硬式

2.0.x

下表列出此元件 2.0.x 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.3	軟式
Greengrass CLI	> = 2.0.3	軟式

如需有關元件相依性的詳細資訊，請參閱元件方案參考。

組態

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

v2.1.x - v2.4.x

`httpsEnabled`

(選擇性) 您可以為本機除錯主控台啟用 HTTPS 通訊。如果您啟用 HTTPS 通訊，本機偵錯主控台會建立自我簽署憑證。Web 瀏覽器會針對使用自我簽署憑證的網站顯示安全性警告，因此您必須手動驗證憑證。然後，您可以繞過警告。如需詳細資訊，請參閱 用量。

預設：`true`

`port`

(選擇性) 用來提供本機除錯主控台的連接埠。

預設：`1441`

`websocketPort`

(選擇性) 用於本機除錯主控台的 websocket 連接埠。

預設：`1442`

`bindHostname`

(選擇性) 用於本機除錯主控台的主機名稱。

如果您在 Docker 容器中執行 AWS IoT Greengrass Core 軟體，請將此參數設定為`0.0.0.0`，以便您可以在 Docker 容器外開啟本機偵錯主控台。

預設：`localhost`

Example 範例：組態合併更新

下列範例組態指定在非預設連接埠上開啟本機偵錯主控台，並停用 HTTPS。

```
{
  "httpsEnabled": false,
  "port": "10441",
  "websocketPort": "10442"
}
```

v2.0.x

`port`

(選擇性) 用來提供本機除錯主控台的連接埠。

預設：`1441`

`websocketPort`

(選擇性) 用於本機除錯主控台的 websocket 連接埠。

預設：`1442`

`bindHostname`

(選擇性) 用於本機除錯主控台的主機名稱。

如果您在 Docker 容器中執行 AWS IoT Greengrass Core 軟體，請將此參數設定為`0.0.0.0`，以便您可以在 Docker 容器外開啟本機偵錯主控台。

預設：`localhost`

Example 範例：組態合併更新

下列範例組態指定在非預設連接埠上開啟本機偵錯主控台。

```
{
  "port": "10441",
  "websocketPort": "10442"
```

```
}
```

用量

若要使用本機偵錯主控台，請從 Greengrass CLI 建立工作階段。當您建立工作階段時，Greengrass CLI 會提供使用者名稱和暫時密碼，讓您用來登入本機偵錯主控台。

請依照下列指示，在核心裝置或開發電腦上開啟本機偵錯主控台。

v2.1.x - v2.4.x

在 2.1.0 及更新版本中，本機偵錯主控台預設會使用 HTTPS。啟用 HTTPS 時，本機偵錯主控台會建立自我簽署憑證以保護連線安全。當您因為此自我簽署憑證而開啟本機偵錯主控台時，Web 瀏覽器會顯示安全性警告。當您使用 Greengrass CLI 建立工作階段時，輸出會包含憑證的指紋，因此您可以驗證憑證是否合法且連線是安全的。

您可以停用 HTTPS。如需詳細資訊，請參閱本機偵錯主控台組態。

開啟本機除錯主控台

1. (選擇性) 若要在開發電腦上檢視本機偵錯主控台，您可以透過 SSH 轉寄主控台的連接埠。不過，您必須先在核心裝置的 SSH 設定檔中啟用該`AllowTcpForwarding`選項。此選項預設為啟用。在開發電腦上執行下列命令，以檢視開發電腦`localhost:1441`上的儀表板。

   ```
   ssh -L 1441:localhost:1441 -L 1442:localhost:1442 username@core-device-ip-address
   ```

   > **Note**
   >
   > 您可以從`1441`和變更預設連接埠`1442`。如需詳細資訊，請參閱本機偵錯主控台組態。

2. 建立工作階段以使用本機偵錯主控台。當您建立工作階段時，您會產生用來驗證的密碼。本機偵錯主控台需要密碼來提高安全性，因為您可以使用此元件來檢視重要資訊並在核心裝置上執行作業。如果您在元件組態中啟用 HTTPS，本機偵錯主控台也會建立憑證來保護連線的安全。HTTPS 預設為啟用狀態。

   使用 AWS IoT Greengrass CLI 建立工作階段。此命令會產生隨機 43 個字元的密碼，該密碼會在 8 小時後過期。以 AWS IoT Greengrass V2 根資料夾的路徑取代*/greengrass/v2*或*C:\greengrass\v2*。

Linux or Unix

```
sudo /greengrass/v2/bin/greengrass-cli get-debug-password
```

Windows

```
C:\greengrass\v2\bin\greengrass-cli get-debug-password
```

如果您已將本機偵錯主控台設定為使用 HTTPS，命令輸出看起來像下列範例。當您開啟本機偵錯主控台時，您可以使用憑證指紋來確認連線是否安全。

```
Username: debug
Password: bEDp3MOHdj8ou2w5de_sCBI2XAaguy3a8XxREXAMPLE
Password expires at: 2021-04-01T17:01:43.921999931-07:00
The local debug console is configured to use TLS security. The certificate is
 self-signed so you will need to bypass your web browser's security warnings to
 open the console.
Before you bypass the security warning, verify that the certificate fingerprint
 matches the following fingerprints.
SHA-256: 15 0B 2C E2 54 8B 22 DE 08 46 54 8A B1 2B 25 DE FB 02 7D 01 4E 4A 56 67
 96 DA A6 CC B1 D2 C4 1B
SHA-1: BC 3E 16 04 D3 80 70 DA E0 47 25 F9 90 FA D6 02 80 3E B5 C1
```

調試視圖組件創建持續 8 小時的會話。之後，您必須產生新密碼才能再次檢視本機偵錯主控台。

3. 開啟並登入儀表板。在 Greengrass 核心裝置上檢視儀表板，或者如果您透過 SSH 轉送連接埠，請在開發電腦上檢視儀表板。執行以下任意一項：

   - 如果您在本機除錯主控台中啟用 HTTPS (此為預設設定)，請執行下列動作：

     a. 在核心設備`https://localhost:1441`上打開，或者在開發計算機上打開（如果您通過 SSH 轉發端口）。

        您的瀏覽器可能會顯示有關無效安全性憑證的安全性警告。

     b. 如果您的瀏覽器顯示安全性警告，請確認憑證是否合法並略過安全性警告。請執行下列操作：

i. 找出憑證的 SHA-256 或 SHA-1 指紋，並確認它是否符合先前列印的指`get-debug-password`令的 SHA-256 或 SHA-1 指紋。您的瀏覽器可能會提供一個或兩個指紋。請參閱瀏覽器的文件，以檢視憑證及其指紋。在某些瀏覽器中，憑證指紋稱為指紋。

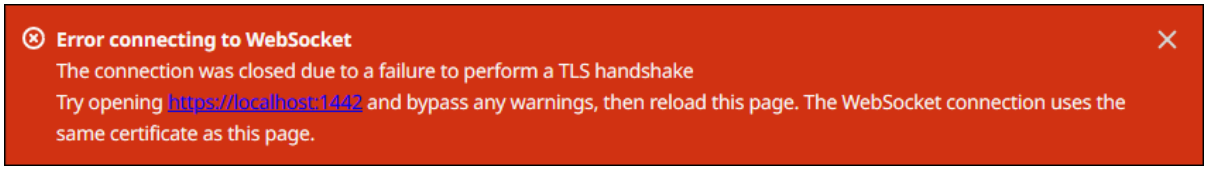
> **Note**
>
> 如果憑證指紋不相符，請移Step 2至建立新的工作階段。如果憑證指紋仍不相符，表示您的連線可能不安全。

ii. 如果憑證指紋相符，請略過瀏覽器的安全性警告，以開啟本機偵錯主控台。請參閱瀏覽器的文件，以略過瀏覽器安全性警告。

c. 使用先前列印的`get-debug-password`命令的使用者名稱和密碼登入網站。

本機除錯主控台隨即開啟。

d. 如果本機偵錯主控台顯示錯誤，指出 WebSocket 由於 TLS 交涉失敗而無法連線到，您必須略過 WebSocket URL 的自我簽署安全性警告。



請執行下列操作：

i. `https://localhost:1442`在您開啟本機偵錯主控台的相同瀏覽器中開啟。

ii. 驗證憑證並略過安全性警告。

略過警告後，瀏覽器可能會顯示 HTTP 404 頁面。

iii. `https://localhost:1441`再次打開。

本機偵錯主控台會顯示核心裝置的相關資訊。

- 如果您在本機偵錯主控台中停用 HTTPS，請執行下列動作：

a. 在核心設備`http://localhost:1441`上打開，或者如果您通過 SSH 轉發端口，請在開發計算機上打開它。

b. 使用先前列印的`get-debug-password`命令的使用者名稱和密碼登入網站。

本機除錯主控台隨即開啟。

v2.0.x

開啟本機除錯主控台

1. (選擇性) 若要在開發電腦上檢視本機偵錯主控台，您可以透過 SSH 轉寄主控台的連接埠。不過，您必須先在核心裝置的 SSH 設定檔中啟用該`AllowTcpForwarding`選項。此選項預設為啟用。在開發電腦上執行下列命令，以檢視開發電腦`localhost:1441`上的儀表板。

   ```
   ssh -L 1441:localhost:1441 -L 1442:localhost:1442 username@core-device-ip-address
   ```

   > **Note**
   >
   > 您可以從`1441`和變更預設連接埠`1442`。如需詳細資訊，請參閱本機偵錯主控台組態。

2. 建立工作階段以使用本機偵錯主控台。當您建立工作階段時，您會產生用來驗證的密碼。本機偵錯主控台需要密碼來提高安全性，因為您可以使用此元件來檢視重要資訊並在核心裝置上執行作業。

   使用 AWS IoT Greengrass CLI 建立工作階段。此命令會產生隨機 43 個字元的密碼，該密碼會在 8 小時後過期。以 AWS IoT Greengrass V2 根資料夾的路徑取代*/greengrass/v2*或*C:\greengrass\v2*。

   Linux or Unix

   ```
   sudo /greengrass/v2/bin/greengrass-cli get-debug-password
   ```

   Windows

   ```
   C:\greengrass\v2\bin\greengrass-cli get-debug-password
   ```

   命令輸出如下列範例所示。

   ```
   Username: debug
   Password: bEDp3MOHdj8ou2w5de_sCBI2XAaguy3a8XxREXAMPLE
   ```

```
Password will expire at: 2021-04-01T17:01:43.921999931-07:00
```

調試視圖組件創建會話持續 4 小時，然後你必須生成一個新的密碼再次查看本地調試控制台。

3. 在核心設備`http://localhost:1441`上打開，或者如果您通過 SSH 轉發端口，請在開發計算機上打開它。
4. 使用先前列印的`get-debug-password`命令的使用者名稱和密碼登入網站。

本機除錯主控台隨即開啟。

本機記錄檔

此元件使用與 Greengrass 核元件相同的記錄檔。

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

若要檢視此元件的記錄

- 在核心裝置上執行下列命令，即時檢視此元件的記錄檔。以 AWS IoT Greengrass 根資料夾的路徑取代*/greengrass/v2*或 *C:\greengrass\v2*。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

變更記錄

下表說明元件每個版本中的變更。

版本	變更
2.4.2	錯誤修復和改進 • 一般錯誤修正與改進。
2.4.1	版本更新 Greengrass 2.12.0 版本釋放。
2.4.0	新功能 • 添加流管理器調試控制台。
2.3.1	版本更新了 Greengrass 核 2.11.0 版本釋放。
2.3.0	版本更新了 Greengrass 2.10.0 版本。 新功能 • 包括 PubSub 和 AWS IoT Core MQTT 調試客戶端。
2.2.7	版本更新 Greengrass 2.9.0 版本釋放。
2.2.6	版本更新 Greengrass 2.8.0 版本的版本。
2.2.5	版本更新了 Greengrass 核 2.7.0 版本釋放。
2.2.4	版本更新 Greengrass 2.6.0 版本發布。
2.2.3	錯誤修復和改進 • 修正當元件無法解密包含 SSL 私密金鑰的金鑰儲存庫時，無法啟動的問題。 • 版本更新了 Greengrass 核 2.5.0 版本。
2.2.2	版本更新 Greengrass 2.4.0 版本的版本。
2.2.1	版本更新了 Greengrass 核 2.3.0 版本。
2.2.0	版本更新了 Greengrass 核 2.2.0 版本。
2.1.0	新功能 • 使用 HTTPS 來保護您與本機偵錯主控台的連線安全。HTTPS 預設為啟用狀態。

版本	變更
	錯誤修復和改進 • 您可以在設定編輯器中關閉快閃列訊息。
2.0.3	初始版本。

日誌管理器

日誌管理器元件 (`aws.greengrass.LogManager`) 會將日誌從AWS IoT Greengrass核心裝置上傳到 Amazon CloudWatch 日誌。您可以從 Greengrass 核心、其他 Greengrass 元件，以及其他不是 Greengrass 元件的應用程式和服務上傳記錄檔。如需如何在記錄檔和本機檔案系統中監視 CloudWatch 記錄的詳細資訊，請參閱監控AWS IoT Greengrass日誌。

當您使用記錄檔管理員元件寫入 CloudWatch 記錄檔時，需要考量下列事項：

- 記錄延遲

  > **Note**
  >
  > 我們建議您升級至記錄檔管理員 2.3.0 版，以減少輪換和使用中記錄檔的記錄延遲。當您升級至記錄檔管理員 2.3.0 時，我們建議您也升級至 Greengrass 核心 2.9.1。

  記錄管理員元件 2.2.8 版 (及更早版本) 只會從輪換的記錄檔處理和上傳記錄檔。根據預設，AWS IoT Greengrass核心軟體會每小時或在 1,024 KB 之後輪換記錄檔。因此，記錄管理員元件只會在AWS IoT Greengrass核心軟體或 Greengrass 元件寫入超過 1,024 KB 的記錄檔之後上傳記錄檔。您可以設定較低的記錄檔大小限制，使記錄檔更頻繁地旋轉。這會導致記錄檔管理員元件更頻繁地將記錄檔上傳至 CloudWatch 記錄檔。

  記錄管理員元件 2.3.0 版 (及更新版本) 會處理並上傳所有記錄檔。當您寫入新的記錄檔時，記錄檔管理員 2.3.0 版 (及更新版本) 會處理並直接上傳該使用中的記錄檔，而不是等待輪換。這意味著您可以在 5 分鐘或更短的時間內查看新日誌。

  記錄管理員元件會定期上傳新的記錄檔。根據預設，記錄管理員元件會每 5 分鐘上傳一次新的記錄檔。您可以設定較低的上傳間隔，因此記錄管理員元件會將記錄檔上傳至 CloudWatch 記錄檔的頻率，方法是設定`periodicUploadIntervalSec`. 如需如何設定此週期間隔的相關資訊，請參閱組態。

日誌可以從相同的 Greengrass 文件系統以近乎即時的方式上傳。如果您需要即時觀察記錄，請考慮使用檔案系統記錄檔。

> **Note**
>
> 如果您使用不同的檔案系統來寫入記錄檔，記錄檔管理員會還原為記錄檔管理員元件 2.2.8 及更早版本中的行為。如需存取檔案系統記錄檔的相關資訊，請參閱存取檔案系統記錄。

- 時鐘偏斜

  記錄管理員元件會使用標準簽章第 4 版簽署程序來建立對 CloudWatch 記錄的 API 要求。如果核心裝置上的系統時間不同步超過 15 分鐘，則 CloudWatch Logs 會拒絕要求。如需詳細資訊，請參閱 AWS 一般參考 中的 Signature 第 4 版簽署程序。

如需有關此元件上傳記錄檔的記錄群組和記錄資料流的資訊，請參閱用量。

主題

版本

此元件具有下列版本：

- 2.3.x 版本
- 2.2.x 版本
- 2.1.x

- 2.0.x

Type

這個組件是一個插件組件（`aws.greengrass.plugin`）。Greengrass 核在與核相同的 Java 虛擬機（JVM）中運行此組件。當您在核心裝置上變更此元件的版本時，核心會重新啟動。

此組件使用與 Greengrass 核相同的日誌文件。如需詳細資訊，請參閱 監控AWS IoT Greengrass日誌。

如需詳細資訊，請參閱 元件類型。

作業系統

此元件可安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件具有下列需求：

- Greengrass 裝置角色必須允許`logs:CreateLogGroup`、`logs:CreateLogStream`、和`logs:DescribeLogStreams`動作`logs:PutLogEvents`，如以下範例 IAM 政策所示。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:logs:*:*:*"
    }
  ]
```

```
}
```

> **Note**
>
> 您在安裝 AWS IoT Greengrass Core 軟體時建立的 Greengrass 裝置角色預設包含此範例原則中的權限。

如需詳細資訊，請參閱 Amazon CloudWatch 日誌使用者指南中的對 CloudWatch 日誌使用身分型政策 (IAM 政策)。

- 支援在 VPC 中執行記錄管理員元件。若要在 VPC 中部署此元件，需要下列項目。
  - 記錄管理員元件必須具有`logs.region.amazonaws.com`的 VPC 端點的`com.amazonaws.us-east-1.logs`連線。

端點和連接埠

除了基本作業所需的端點和連接埠之外，此元件還必須能夠對下列端點和連接埠執行輸出要求。如需詳細資訊，請參閱 允許裝置流量透過 Proxy 或防火牆。

端點	連線埠	必要	描述
`logs.`*`region`*`.amazonaws.com`	443	否	如果您將記錄寫入記錄檔，則為 CloudWatch 必要項

相依性

部署元件時，AWS IoT Greengrass也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的需求，才能成功部署元件。本節列出此元件之已發行版本的相依性，以及定義每個相依性之元件版本的語意版本條件約束。您也可以在AWS IoT Greengrass主控台中檢視元件每個版本的相依性。在元件詳細資料頁面上，尋找 [相依性] 清單。

2.3.7

下表列出此元件 2.3.7 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.1.0	軟式

2.3.5 and 2.3.6

下表列出此元件 2.3.5 和 2.3.6 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.1.0	軟式

2.3.3 – 2.3.4

下表列出此元件 2.3.3 至 2.3.4 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.1.0	軟式

2.2.8 – 2.3.2

下表列出此元件 2.2.8 至 2.3.2 版本的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.1.0	軟式

2.2.7

下表列出此元件 2.2.7 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.1.0	軟式

2.2.6

下表列出此元件 2.2.6 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.1.0	軟式

2.2.5

下表列出此元件 2.2.5 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.1.0	軟式

2.2.1 - 2.2.4

下表列出此元件 2.2.1-2.2.4 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.1.0	軟式

2.1.3 and 2.2.0

下表列出此元件 2.1.3 和 2.2.0 版本的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.1.0	軟式

2.1.2

下表列出此元件 2.1.2 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.1.0	軟式

2.1.1

下表列出此元件 2.1.1 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.1.0	軟式

2.1.0

下表列出此元件 2.1.0 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.1.0	軟式

2.0.x

下表列出此元件 2.0.x 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.3	軟式

如需有關元件相依性的詳細資訊，請參閱元件方案參考。

組態

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

v2.3.6 – v2.3.7

`logsUploaderConfiguration`

(選擇性) 記錄檔管理員元件上傳的記錄檔組態。此物件包含下列資訊：

`systemLogsConfiguration`

(選擇性) AWS IoT Greengrass 核心軟體系統記錄的組態，其中包括 Greengrass 核心核心和外掛程式元件的記錄。指定此組態可讓記錄管理員元件管理系統記錄檔管理員。此物件包含下列資訊：

`uploadToCloudWatch`

(選擇性) 您可以將系統記錄上傳至 CloudWatch 記錄檔。

預設：`false`

`minimumLogLevel`

(選擇性) 要上傳的記錄訊息的最低層級。只有當您將 Greengrass 核心元件設定為輸出 JSON 格式記錄時，才會套用此最低層級。若要啟用 JSON 格式記錄檔，`JSON`請指定記錄格式參數 (`logging.format`)。

從下列記錄層級中進行選擇，依照層級順序列出：

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

預設：`INFO`

`diskSpaceLimit`

(選擇性) Greengrass 系統記錄檔的大小上限 (以您在中指定的單位表示)。`diskSpaceLimitUnit`在 Greengrass 系統記錄檔的總大小超過此總大小上限之後，AWS IoT Greengrass核心軟體會刪除最舊的 Greengrass 系統記錄檔。

此參數相當於 Greengrass 核元件的記錄檔大小限制參數 (`totalLogsSizeKB`)。AWS IoT Greengrass核心軟體使用兩個值中的最小值做為 Greengrass 系統記錄檔大小的最大總計。

`diskSpaceLimitUnit`

(選擇性) 的單位`diskSpaceLimit`。您可以從以下選項中選擇：

- `KB`— 千字節
- `MB`— 兆字節
- `GB`— 千兆字節

預設：`KB`

`deleteLogFileAfterCloudUpload`

(選擇性) 您可以在記錄管理員元件將記錄檔上傳至「記錄檔」之後刪除 CloudWatch 記錄檔。

預設：`false`

`componentLogsConfigurationMap`

(選擇性) 核心裝置上元件的記錄組態對映。此對映中的每個`componentName`物件都會定義元件或應用程式的記錄組態。記錄管理員元件會將這些元件記錄檔上傳至 CloudWatch 記錄檔。

> **Important**
>
> 我們強烈建議每個元件使用單一組態金鑰。您應該只鎖定只有一個記錄檔的檔案群組，這些記錄檔在使用`logFileRegex`. 不遵循此建議可能會導致重複的記錄檔上傳到 CloudWatch。如果您使用單一規則運算式鎖定多個使用中記錄檔，建議您升級至 log Manager v2.3.1 或更新版本，並考慮使用範例組態變更組態。

> **Note**
>
> 如果您要從 v2.2.0 之前的日誌管理器版本升級，則可以繼續使用該`componentLogsConfiguration`列表而不是。`componentLogsConfigurationMap`但是，我們強烈建議您使用對映格式，以便您可以使用合併和重設更新來修改特定零組件的模型組態。如需有關`componentLogsConfiguration`參數的資訊，請參閱此元件 v2.1.x 的組態參數。

componentName

此記錄組態之*componentName*元件或應用程式的記錄組態。您可以指定 Greengrass 元件的名稱或其他值來識別此記錄群組。

每個物件都包含下列資訊：

`minimumLogLevel`

(選擇性) 要上傳的記錄訊息的最低層級。此最低層級僅適用於此元件的記錄檔使用特定的 JSON 格式，您可以在上的AWS IoT Greengrass記錄模組存放庫中找到 GitHub。

從下列記錄層級中進行選擇，依照層級順序列出：

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

預設：`INFO`

`diskSpaceLimit`

(選擇性) 此元件的所有記錄檔總大小上限 (以您在中指定的單位表示) `diskSpaceLimitUnit`。在此元件的記錄檔總大小超過此最大總大小之後，AWS IoT GreengrassCore 軟體就會刪除此元件最舊的記錄檔。

此參數與 Greengrass 核元件的記錄檔大小限制參數 (`totalLogsSizeKB`) 有關。AWS IoT Greengrass核心軟體會使用兩個值中的最小值做為此元件的最大總記錄大小。

`diskSpaceLimitUnit`

(選擇性) 的單位`diskSpaceLimit`。您可以從以下選項中選擇：

- `KB`— 千字節
- `MB`— 兆字節
- `GB`— 千兆字節

預設：`KB`

`logFileDirectoryPath`

(選擇性) 包含此元件記錄檔之資料夾的路徑。

您不需要為打印到標準輸出（標準輸出）和標準錯誤（stderr）的 Greengrass 組件指定此參數。

預設：*/greengrass/v2*/logs。

`logFileRegex`

(選擇性) 規則運算式，指定元件或應用程式使用的記錄檔名稱格式。記錄管理員元件會使用此規則運算式來識別位於資料夾中的記錄檔`logFileDirectoryPath`。

您不需要為打印到標準輸出（標準輸出）和標準錯誤（stderr）的 Greengrass 組件指定此參數。

如果您的組件或應用程序旋轉日誌文件，請指定與旋轉的日誌文件名稱匹配的正則表達式。例如，您可**`hello_world\\\\w*.log`**以指定為 Hello World 應用程式上傳記錄檔。該`\\\\w*`模式匹配零個或多個單詞字符，其中包括字母數字字符和底線。這個正則表達式匹配名稱中有和沒有時間戳的日誌文件。在此範例中，記錄管理員會上傳下列記錄檔：

- `hello_world.log`— Hello World 應用程式的最新記錄檔。
- `hello_world_2020_12_15_17_0.log`— Hello World 應用程式的較舊記錄檔。

預設值：*componentName*`\\\\w*.log`，其中*####*是此記錄檔組態的元件名稱。

`deleteLogFileAfterCloudUpload`

(選擇性) 您可以在記錄管理員元件將記錄檔上傳至「記錄檔」之後刪除 CloudWatch 記錄檔。

預設：`false`

`multiLineStartPattern`

(選擇性) 識別新行上的記錄訊息何時是新記錄訊息的規則運算式。如果規則運算式不符合新行，則記錄管理員元件會將新行附加至上一行的記錄檔訊息。

根據預設，記錄管理員元件會檢查該行是否以空白字元開頭，例如索引標籤或空格。如果沒有，日誌管理器處理該行作為新的日誌消息。否則，它會將該行附加到當前

日誌消息中。此行為可確保記錄檔管理員元件不會分割跨越多行的郵件，例如堆疊追蹤。

`periodicUploadIntervalSec`

(選擇性) 記錄管理員元件檢查要上傳之新記錄檔的期間 (以秒為單位)。

預設值：`300`(5 分鐘)

最小值：`0.000001` (1 微秒)

`deprecatedVersionSupport`

指出記錄檔管理員是否應該使用記錄檔管理員 v2.3.5 中引入的記錄速度改進。將值設定為以`false`使用改良功能。

如果您將此值設定為，`false`當您從記錄管理員 v2.3.1 升級或更早版本的重複記錄項目時，可能會上傳。

預設值為 `true`。

Example 範例：組態合併更新

下列範例組態指定將系統記錄檔和`com.example.HelloWorld`元件記錄檔上傳至 CloudWatch 記錄檔。

```
{
  "logsUploaderConfiguration": {
    "systemLogsConfiguration": {
      "uploadToCloudWatch": "true",
      "minimumLogLevel": "INFO",
      "diskSpaceLimit": "10",
      "diskSpaceLimitUnit": "MB",
      "deleteLogFileAfterCloudUpload": "false"
    },
    "componentLogsConfigurationMap": {
      "com.example.HelloWorld": {
        "minimumLogLevel": "INFO",
        "diskSpaceLimit": "20",
        "diskSpaceLimitUnit": "MB",
        "deleteLogFileAfterCloudUpload": "false"
      }
    }
```

```
  },
  "periodicUploadIntervalSec": "300",
  "deprecatedVersionSupport": "false"
}
```

Example 範例：使用記錄管理員 v2.3.1 上傳多個使用中記錄檔的組態

如果您要將多個使用中記錄檔鎖定為目標，則建議使用下列範例組態。此範例組態指定您要上傳到哪些使用中記錄檔 CloudWatch。使用此配置範例配置也會上傳符合的任何旋轉檔案`logFileRegex`。日誌管理器 v2.3.1 支持此示例配置。

```
{
  "logsUploaderConfiguration": {
    "componentLogsConfigurationMap": {
      "com.example.A": {
        "logFileRegex": "com.example.A\\w*.log",
        "deleteLogFileAfterCloudUpload": "false"
      }
      "com.example.B": {
        "logFileRegex": "com.example.B\\w*.log",
        "deleteLogFileAfterCloudUpload": "false"
      }
    }
  },
  "periodicUploadIntervalSec": "10"
}
```

v2.3.x

`logsUploaderConfiguration`

(選擇性) 記錄檔管理員元件上傳的記錄檔組態。此物件包含下列資訊：

`systemLogsConfiguration`

(選擇性) AWS IoT Greengrass 核心軟體系統記錄的組態，其中包括 Greengrass 核心核心和外掛程式元件的記錄。指定此組態可讓記錄管理員元件管理系統記錄檔管理員。此物件包含下列資訊：

`uploadToCloudWatch`

(選擇性) 您可以將系統記錄上傳至 CloudWatch 記錄檔。

預設：false

minimumLogLevel

(選擇性) 要上傳的記錄訊息的最低層級。只有當您將 Greengrass 核心元件設定為輸出 JSON 格式記錄時，才會套用此最低層級。若要啟用 JSON 格式記錄檔，JSON請指定記錄格式參數 (logging.format)。

從下列記錄層級中進行選擇，依照層級順序列出：

- DEBUG
- INFO
- WARN
- ERROR

預設：INFO

diskSpaceLimit

(選擇性) Greengrass 系統記錄檔的大小上限 (以您在中指定的單位表示)。diskSpaceLimitUnit在 Greengrass 系統記錄檔的總大小超過此總大小上限之後，AWS IoT Greengrass核心軟體會刪除最舊的 Greengrass 系統記錄檔。

此參數相當於 Greengrass 核元件的記錄檔大小限制參數 (totalLogsSizeKB)。AWS IoT Greengrass核心軟體使用兩個值中的最小值做為 Greengrass 系統記錄檔大小的最大總計。

diskSpaceLimitUnit

(選擇性) 的單位diskSpaceLimit。您可以從以下選項中選擇：

- KB— 千字節
- MB— 兆字節
- GB— 千兆字節

預設：KB

deleteLogFileAfterCloudUpload

(選擇性) 您可以在記錄管理員元件將記錄檔上傳至「記錄檔」之後刪除 CloudWatch 記錄檔。

預設：false

`componentLogsConfigurationMap`

(選擇性) 核心裝置上元件的記錄組態對映。此對映中的每個`componentName`物件都會定義元件或應用程式的記錄組態。記錄管理員元件會將這些元件記錄檔上傳至 CloudWatch 記錄檔。

> **Important**
>
> 我們強烈建議每個元件使用單一組態金鑰。您應該只鎖定只有一個記錄檔的檔案群組，這些記錄檔在使用`logFileRegex`. 不遵循此建議可能會導致重複的記錄檔上傳到 CloudWatch。如果您使用單一規則運算式鎖定多個使用中記錄檔，建議您升級至 log Manager v2.3.1，並考慮使用範例設定來變更組態。

> **Note**
>
> 如果您要從 v2.2.0 之前的日誌管理器版本升級，則可以繼續使用該`componentLogsConfiguration`列表而不是。`componentLogsConfigurationMap`但是，我們強烈建議您使用對映格式，以便您可以使用合併和重設更新來修改特定零組件的模型組態。如需有關`componentLogsConfiguration`參數的資訊，請參閱此元件 v2.1.x 的組態參數。

`componentName`

此記錄組態之*`componentName`*元件或應用程式的記錄組態。您可以指定 Greengrass 元件的名稱或其他值來識別此記錄群組。

每個物件都包含下列資訊：

`minimumLogLevel`

(選擇性) 要上傳的記錄訊息的最低層級。此最低層級僅適用於此元件的記錄檔使用特定的 JSON 格式，您可以在上的AWS IoT Greengrass記錄模組存放庫中找到 GitHub。

從下列記錄層級中進行選擇，依照層級順序列出：

- `DEBUG`
- `INFO`

- WARN
- ERROR

預設：INFO

diskSpaceLimit

(選擇性) 此元件的所有記錄檔總大小上限 (以您在中指定的單位表示) diskSpaceLimitUnit。在此元件的記錄檔總大小超過此最大總大小之後，AWS IoT GreengrassCore 軟體就會刪除此元件最舊的記錄檔。

此參數與 Greengrass 核元件的記錄檔大小限制參數 (totalLogsSizeKB) 有關。AWS IoT Greengrass核心軟體會使用兩個值中的最小值做為此元件的最大總記錄大小。

diskSpaceLimitUnit

(選擇性) 的單位diskSpaceLimit。您可以從以下選項中選擇：

- KB— 千字節
- MB— 兆字節
- GB— 千兆字節

預設：KB

logFileDirectoryPath

(選擇性) 包含此元件記錄檔之資料夾的路徑。

您不需要為打印到標準輸出（標準輸出）和標準錯誤（stderr）的 Greengrass 組件指定此參數。

預設：*/greengrass/v2*/logs。

logFileRegex

(選擇性) 規則運算式，指定元件或應用程式使用的記錄檔名稱格式。記錄管理員元件會使用此規則運算式來識別位於資料夾中的記錄檔logFileDirectoryPath。

您不需要為打印到標準輸出（標準輸出）和標準錯誤（stderr）的 Greengrass 組件指定此參數。

如果您的組件或應用程序旋轉日誌文件，請指定與旋轉的日誌文件名稱匹配的正則表達式。例如，您可**hello_world\\\\w*.log**以指定為 Hello World 應用程式上傳記

錄檔。該\\\\w*模式匹配零個或多個單詞字符，其中包括字母數字字符和底線。這個正則表達式匹配名稱中有和沒有時間戳的日誌文件。在此範例中，記錄管理員會上傳下列記錄檔：

- hello_world.log— Hello World 應用程式的最新記錄檔。
- hello_world_2020_12_15_17_0.log— Hello World 應用程式的較舊記錄檔。

預設值：*componentName*\\\\w*.log，其中####是此記錄檔組態的元件名稱。

deleteLogFileAfterCloudUpload

(選擇性) 您可以在記錄管理員元件將記錄檔上傳至「記錄檔」之後刪除 CloudWatch 記錄檔。

預設：false

multiLineStartPattern

(選擇性) 識別新行上的記錄訊息何時是新記錄訊息的規則運算式。如果規則運算式不符合新行，則記錄管理員元件會將新行附加至上一行的記錄檔訊息。

根據預設，記錄管理員元件會檢查該行是否以空白字元開頭，例如索引標籤或空格。如果沒有，日誌管理器處理該行作為新的日誌消息。否則，它會將該行附加到當前日誌消息中。此行為可確保記錄檔管理員元件不會分割跨越多行的郵件，例如堆疊追蹤。

periodicUploadIntervalSec

(選擇性) 記錄管理員元件檢查要上傳之新記錄檔的期間 (以秒為單位)。

預設值：300(5 分鐘)

最小值：0.000001 (1 微秒)

Example 範例：組態合併更新

下列範例組態指定將系統記錄檔和com.example.HelloWorld元件記錄檔上傳至 CloudWatch 記錄檔。

```
{
  "logsUploaderConfiguration": {
    "systemLogsConfiguration": {
      "uploadToCloudWatch": "true",
      "minimumLogLevel": "INFO",
```

```
      "diskSpaceLimit": "10",
      "diskSpaceLimitUnit": "MB",
      "deleteLogFileAfterCloudUpload": "false"
    },
    "componentLogsConfigurationMap": {
      "com.example.HelloWorld": {
        "minimumLogLevel": "INFO",
        "diskSpaceLimit": "20",
        "diskSpaceLimitUnit": "MB",
        "deleteLogFileAfterCloudUpload": "false"
      }
    }
  },
  "periodicUploadIntervalSec": "300"
}
```

Example 範例：使用記錄管理員 v2.3.1 上傳多個使用中記錄檔的組態

如果您要將多個使用中記錄檔鎖定為目標，則建議使用下列範例組態。此範例組態指定您要上傳到哪些使用中記錄檔 CloudWatch。使用此配置範例配置也會上傳符合的任何旋轉檔案`logFileRegex`。日誌管理器 v2.3.1 支持此示例配置。

```
{
  "logsUploaderConfiguration": {
    "componentLogsConfigurationMap": {
      "com.example.A": {
        "logFileRegex": "com.example.A\\w*.log",
        "deleteLogFileAfterCloudUpload": "false"
      }
      "com.example.B": {
        "logFileRegex": "com.example.B\\w*.log",
        "deleteLogFileAfterCloudUpload": "false"
      }
    }
  },
  "periodicUploadIntervalSec": "10"
}
```

v2.2.x

`logsUploaderConfiguration`

(選擇性) 記錄檔管理員元件上傳的記錄檔組態。此物件包含下列資訊：

`systemLogsConfiguration`

(選擇性) AWS IoT Greengrass 核心軟體系統記錄的組態，其中包括 Greengrass 核心核心和外掛程式元件的記錄。指定此組態可讓記錄管理員元件管理系統記錄檔管理員。此物件包含下列資訊：

`uploadToCloudWatch`

(選擇性) 您可以將系統記錄上傳至 CloudWatch 記錄檔。

預設：`false`

`minimumLogLevel`

(選擇性) 要上傳的記錄訊息的最低層級。只有當您將 Greengrass 核心元件設定為輸出 JSON 格式記錄時，才會套用此最低層級。若要啟用 JSON 格式記錄檔，`JSON`請指定記錄格式參數 (`logging.format`)。

從下列記錄層級中進行選擇，依照層級順序列出：

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

預設：`INFO`

`diskSpaceLimit`

(選擇性) Greengrass 系統記錄檔的大小上限 (以您在中指定的單位表示)。`diskSpaceLimitUnit`在 Greengrass 系統記錄檔的總大小超過此總大小上限之後，AWS IoT Greengrass核心軟體會刪除最舊的 Greengrass 系統記錄檔。

此參數相當於 Greengrass 核元件的記錄檔大小限制參數 (`totalLogsSizeKB`)。AWS IoT Greengrass核心軟體使用兩個值中的最小值做為 Greengrass 系統記錄檔大小的最大總計。

`diskSpaceLimitUnit`

(選擇性) 的單位`diskSpaceLimit`。您可以從以下選項中選擇：

- `KB`— 千字節
- `MB`— 兆字節

- GB— 千兆字節

預設：KB

`deleteLogFileAfterCloudUpload`

(選擇性) 您可以在記錄管理員元件將記錄檔上傳至「記錄檔」之後刪除 CloudWatch 記錄檔。

預設：`false`

`componentLogsConfigurationMap`

(選擇性) 核心裝置上元件的記錄組態對映。此對映中的每個`componentName`物件都會定義元件或應用程式的記錄組態。記錄管理員元件會將這些元件記錄檔上傳至 CloudWatch 記錄檔。

> **Note**
>
> 如果您要從 v2.2.0 之前的日誌管理器版本升級，則可以繼續使用該`componentLogsConfiguration`列表而不是。`componentLogsConfigurationMap`但是，我們強烈建議您使用對映格式，以便您可以使用合併和重設更新來修改特定零組件的模型組態。如需有關`componentLogsConfiguration`參數的資訊，請參閱此元件 v2.1.x 的組態參數。

`componentName`

此記錄組態之*`componentName`*元件或應用程式的記錄組態。您可以指定 Greengrass 元件的名稱或其他值來識別此記錄群組。

每個物件都包含下列資訊：

`minimumLogLevel`

(選擇性) 要上傳的記錄訊息的最低層級。此最低層級僅適用於此元件的記錄檔使用特定的 JSON 格式，您可以在上的AWS IoT Greengrass記錄模組存放庫中找到 GitHub。

從下列記錄層級中進行選擇，依照層級順序列出：

- DEBUG

- INFO
- WARN
- ERROR

預設：INFO

diskSpaceLimit

(選擇性) 此元件的所有記錄檔總大小上限 (以您在中指定的單位表示) diskSpaceLimitUnit。在此元件的記錄檔總大小超過此最大總大小之後，AWS IoT GreengrassCore 軟體就會刪除此元件最舊的記錄檔。

此參數與 Greengrass 核元件的記錄檔大小限制參數 (totalLogsSizeKB) 有關。AWS IoT Greengrass核心軟體會使用兩個值中的最小值做為此元件的最大總記錄大小。

diskSpaceLimitUnit

(選擇性) 的單位diskSpaceLimit。您可以從以下選項中選擇：

- KB— 千字節
- MB— 兆字節
- GB— 千兆字節

預設：KB

logFileDirectoryPath

(選擇性) 包含此元件記錄檔之資料夾的路徑。

您不需要為打印到標準輸出（標準輸出）和標準錯誤（stderr）的 Greengrass 組件指定此參數。

預設：*/greengrass/v2*/logs。

logFileRegex

(選擇性) 規則運算式，指定元件或應用程式使用的記錄檔名稱格式。記錄管理員元件會使用此規則運算式來識別位於資料夾中的記錄檔logFileDirectoryPath。

您不需要為打印到標準輸出（標準輸出）和標準錯誤（stderr）的 Greengrass 組件指定此參數。

如果您的組件或應用程序旋轉日誌文件，請指定與旋轉的日誌文件名稱匹配的正則表達式。例如，您可**hello_world\\\\w*.log**以指定為 Hello World 應用程式上傳記錄檔。該\\\\w*模式匹配零個或多個單詞字符，其中包括字母數字字符和底線。這個正則表達式匹配名稱中有和沒有時間戳的日誌文件。在此範例中，記錄管理員會上傳下列記錄檔：

- hello_world.log— Hello World 應用程式的最新記錄檔。
- hello_world_2020_12_15_17_0.log— Hello World 應用程式的較舊記錄檔。

預設值：*componentName*\\\\w*.log，其中####是此記錄檔組態的元件名稱。

deleteLogFileAfterCloudUpload

(選擇性) 您可以在記錄管理員元件將記錄檔上傳至「記錄檔」之後刪除 CloudWatch 記錄檔。

預設：false

multiLineStartPattern

(選擇性) 識別新行上的記錄訊息何時是新記錄訊息的規則運算式。如果規則運算式不符合新行，則記錄管理員元件會將新行附加至上一行的記錄檔訊息。

根據預設，記錄管理員元件會檢查該行是否以空白字元開頭，例如索引標籤或空格。如果沒有，日誌管理器處理該行作為新的日誌消息。否則，它會將該行附加到當前日誌消息中。此行為可確保記錄檔管理員元件不會分割跨越多行的郵件，例如堆疊追蹤。

periodicUploadIntervalSec

(選擇性) 記錄管理員元件檢查要上傳之新記錄檔的期間 (以秒為單位)。

預設值：300(5 分鐘)

最小值：0.000001 (1 微秒)

Example 範例：組態合併更新

下列範例組態指定將系統記錄檔和com.example.HelloWorld元件記錄檔上傳至 CloudWatch 記錄檔。

```
{
```

```
  "logsUploaderConfiguration": {
    "systemLogsConfiguration": {
      "uploadToCloudWatch": "true",
      "minimumLogLevel": "INFO",
      "diskSpaceLimit": "10",
      "diskSpaceLimitUnit": "MB",
      "deleteLogFileAfterCloudUpload": "false"
    },
    "componentLogsConfigurationMap": {
      "com.example.HelloWorld": {
        "minimumLogLevel": "INFO",
        "diskSpaceLimit": "20",
        "diskSpaceLimitUnit": "MB",
        "deleteLogFileAfterCloudUpload": "false"
      }
    }
  },
  "periodicUploadIntervalSec": "300"
}
```

v2.1.x

`logsUploaderConfiguration`

(選擇性) 記錄檔管理員元件上傳的記錄檔組態。此物件包含下列資訊：

`systemLogsConfiguration`

(選擇性) AWS IoT Greengrass 核心軟體系統記錄的組態，其中包括 Greengrass 核心核心和外掛程式元件的記錄。指定此組態可讓記錄管理員元件管理系統記錄檔管理員。此物件包含下列資訊：

`uploadToCloudWatch`

(選擇性) 您可以將系統記錄上傳至 CloudWatch 記錄檔。

預設：`false`

`minimumLogLevel`

(選擇性) 要上傳的記錄訊息的最低層級。只有當您將 Greengrass 核心元件設定為輸出 JSON 格式記錄時，才會套用此最低層級。若要啟用 JSON 格式記錄檔，JSON請指定記錄格式參數 (`logging.format`)。

從下列記錄層級中進行選擇，依照層級順序列出：

- DEBUG
- INFO
- WARN
- ERROR

預設：INFO

diskSpaceLimit

(選擇性) Greengrass 系統記錄檔的大小上限 (以您在中指定的單位表示)。diskSpaceLimitUnit在 Greengrass 系統記錄檔的總大小超過此總大小上限之後，AWS IoT Greengrass核心軟體會刪除最舊的 Greengrass 系統記錄檔。

此參數相當於 Greengrass 核元件的記錄檔大小限制參數 (totalLogsSizeKB)。AWS IoT Greengrass核心軟體使用兩個值中的最小值做為 Greengrass 系統記錄檔大小的最大總計。

diskSpaceLimitUnit

(選擇性) 的單位diskSpaceLimit。您可以從以下選項中選擇：

- KB— 千字節
- MB— 兆字節
- GB— 千兆字節

預設：KB

deleteLogFileAfterCloudUpload

(選擇性) 您可以在記錄管理員元件將記錄檔上傳至「記錄檔」之後刪除 CloudWatch 記錄檔。

預設：false

componentLogsConfiguration

(選擇性) 核心裝置上元件的記錄組態清單。此清單中的每個組態都會定義元件或應用程式的記錄組態。記錄管理員元件會將這些元件記錄檔上傳至 CloudWatch 記錄

每個物件都包含下列資訊：

`componentName`

此記錄組態的元件或應用程式名稱。您可以指定 Greengrass 元件的名稱或其他值來識別此記錄群組。

`minimumLogLevel`

(選擇性) 要上傳的記錄訊息的最低層級。此最低層級僅適用於此元件的記錄檔使用特定的 JSON 格式，您可以在上的AWS IoT Greengrass記錄模組存放庫中找到 GitHub。

從下列記錄層級中進行選擇，依照層級順序列出：

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

預設：`INFO`

`diskSpaceLimit`

(選擇性) 此元件的所有記錄檔總大小上限 (以您在中指定的單位表示) `diskSpaceLimitUnit`。在此元件的記錄檔總大小超過此最大總大小之後，AWS IoT GreengrassCore 軟體就會刪除此元件最舊的記錄檔。

此參數與 Greengrass 核元件的記錄檔大小限制參數 (`totalLogsSizeKB`) 有關。AWS IoT Greengrass核心軟體會使用兩個值中的最小值做為此元件的最大總記錄大小。

`diskSpaceLimitUnit`

(選擇性) 的單位`diskSpaceLimit`。您可以從以下選項中選擇：

- `KB`— 千字節
- `MB`— 兆字節
- `GB`— 千兆字節

預設：`KB`

`logFileDirectoryPath`

(選擇性) 包含此元件記錄檔之資料夾的路徑。

您不需要為打印到標準輸出（標準輸出）和標準錯誤（stderr）的 Greengrass 組件指定此參數。

預設：*/greengrass/v2*/logs。

`logFileRegex`

(選擇性) 規則運算式，指定元件或應用程式使用的記錄檔名稱格式。記錄管理員元件會使用此規則運算式來識別位於資料夾中的記錄檔`logFileDirectoryPath`。

您不需要為打印到標準輸出（標準輸出）和標準錯誤（stderr）的 Greengrass 組件指定此參數。

如果您的組件或應用程序旋轉日誌文件，請指定與旋轉的日誌文件名稱匹配的正則表達式。例如，您可**`hello_world\\\\w*.log`**以指定為 Hello World 應用程式上傳記錄檔。該`\\\\w*`模式匹配零個或多個單詞字符，其中包括字母數字字符和底線。這個正則表達式匹配名稱中有和沒有時間戳的日誌文件。在此範例中，記錄管理員會上傳下列記錄檔：

- `hello_world.log`— Hello World 應用程式的最新記錄檔。
- `hello_world_2020_12_15_17_0.log`— Hello World 應用程式的較舊記錄檔。

預設值：*componentName*`\\\\w*.log`，其中*####*是此記錄檔組態的元件名稱。

`deleteLogFileAfterCloudUpload`

(選擇性) 您可以在記錄管理員元件將記錄檔上傳至「記錄檔」之後刪除 CloudWatch 記錄檔。

預設：`false`

`multiLineStartPattern`

(選擇性) 識別新行上的記錄訊息何時是新記錄訊息的規則運算式。如果規則運算式不符合新行，則記錄管理員元件會將新行附加至上一行的記錄檔訊息。

根據預設，記錄管理員元件會檢查該行是否以空白字元開頭，例如索引標籤或空格。如果沒有，日誌管理器處理該行作為新的日誌消息。否則，它會將該行附加到當前日誌消息中。此行為可確保記錄檔管理員元件不會分割跨越多行的郵件，例如堆疊追蹤。

`periodicUploadIntervalSec`

(選擇性) 記錄管理員元件檢查要上傳之新記錄檔的期間 (以秒為單位)。

預設值：`300`(5 分鐘)

最小值：`0.000001`（1 微秒）

Example 範例：組態合併更新

下列範例組態指定將系統記錄檔和com.example.HelloWorld元件記錄檔上傳至 CloudWatch 記錄檔。

```
{
  "logsUploaderConfiguration": {
    "systemLogsConfiguration": {
      "uploadToCloudWatch": "true",
      "minimumLogLevel": "INFO",
      "diskSpaceLimit": "10",
      "diskSpaceLimitUnit": "MB",
      "deleteLogFileAfterCloudUpload": "false"
    },
    "componentLogsConfiguration": [
      {
        "componentName": "com.example.HelloWorld",
        "minimumLogLevel": "INFO",
        "diskSpaceLimit": "20",
        "diskSpaceLimitUnit": "MB",
        "deleteLogFileAfterCloudUpload": "false"
      }
    ]
  },
  "periodicUploadIntervalSec": "300"
}
```

v2.0.x

`logsUploaderConfiguration`

(選擇性) 記錄檔管理員元件上傳的記錄檔組態。此物件包含下列資訊：

`systemLogsConfiguration`

(選擇性) AWS IoT Greengrass Core 軟體系統記錄檔的組態。指定此組態可讓記錄管理員元件管理系統記錄檔管理員。此物件包含下列資訊：

`uploadToCloudWatch`

(選擇性) 您可以將系統記錄上傳至 CloudWatch 記錄檔。

預設：`false`

`minimumLogLevel`

(選擇性) 要上傳的記錄訊息的最低層級。只有當您將 Greengrass 核心元件設定為輸出 JSON 格式記錄時，才會套用此最低層級。若要啟用 JSON 格式記錄檔，JSON請指定記錄格式參數 (`logging.format`)。

從下列記錄層級中進行選擇，依照層級順序列出：

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

預設：`INFO`

`diskSpaceLimit`

(選擇性) Greengrass 系統記錄檔的大小上限 (以您在中指定的單位表示)。`diskSpaceLimitUnit`在 Greengrass 系統記錄檔的總大小超過此總大小上限之後，AWS IoT Greengrass核心軟體會刪除最舊的 Greengrass 系統記錄檔。

此參數相當於 Greengrass 核元件的記錄檔大小限制參數 (`totalLogsSizeKB`)。AWS IoT Greengrass核心軟體使用兩個值中的最小值做為 Greengrass 系統記錄檔大小的最大總計。

`diskSpaceLimitUnit`

(選擇性) 的單位`diskSpaceLimit`。您可以從以下選項中選擇：

- `KB`— 千字節
- `MB`— 兆字節
- `GB`— 千兆字節

預設：`KB`

`deleteLogFileAfterCloudUpload`

(選擇性) 您可以在記錄管理員元件將記錄檔上傳至「記錄檔」之後刪除 CloudWatch 記錄檔。

預設：`false`

`componentLogsConfiguration`

(選擇性) 核心裝置上元件的記錄組態清單。此清單中的每個組態都會定義元件或應用程式的記錄組態。記錄管理員元件會將這些元件記錄檔上傳至 CloudWatch 記錄

每個物件都包含下列資訊：

`componentName`

此記錄組態的元件或應用程式名稱。您可以指定 Greengrass 元件的名稱或其他值來識別此記錄群組。

`minimumLogLevel`

(選擇性) 要上傳的記錄訊息的最低層級。此最低層級僅適用於此元件的記錄檔使用特定的 JSON 格式，您可以在上的AWS IoT Greengrass記錄模組存放庫中找到 GitHub。

從下列記錄層級中進行選擇，依照層級順序列出：

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

預設：`INFO`

`diskSpaceLimit`

(選擇性) 此元件的所有記錄檔總大小上限 (以您在中指定的單位表示) `diskSpaceLimitUnit`。在此元件的記錄檔總大小超過此最大總大小之後，AWS IoT GreengrassCore 軟體就會刪除此元件最舊的記錄檔。

此參數與 Greengrass 核元件的記錄檔大小限制參數 (`totalLogsSizeKB`) 有關。AWS IoT Greengrass核心軟體會使用兩個值中的最小值做為此元件的最大總記錄大小。

`diskSpaceLimitUnit`

(選擇性) 的單位`diskSpaceLimit`。您可以從以下選項中選擇：

- `KB`— 千字節
- `MB`— 兆字節
- `GB`— 千兆字節

預設：`KB`

`logFileDirectoryPath`

包含此元件記錄檔之資料夾的路徑。

若要上傳 Greengrass 元件的記錄檔，請指定*/greengrass/v2*/**logs**您的 Greengrass 根資料夾並取代*/greengrass/v2*這些記錄檔。

`logFileRegex`

規則運算式，指定元件或應用程式使用的記錄檔名稱格式。記錄管理員元件會使用此規則運算式來識別位於資料夾中的記錄檔`logFileDirectoryPath`。

若要上傳 Greengrass 元件的記錄檔，請指定符合旋轉記錄檔名稱的正則運算式。例如，您可**com.example.HelloWorld\\w*.log**以指定為 Hello World 元件上傳記錄檔。該`\\w*`模式匹配零個或多個單詞字符，其中包括字母數字字符和底線。這個正則表達式匹配名稱中有和沒有時間戳的日誌文件。在此範例中，記錄管理員會上傳下列記錄檔：

- `com.example.HelloWorld.log`— 您好世界元件的最新記錄檔。
- `com.example.HelloWorld_2020_12_15_17_0.log`— 您好世界元件的較舊記錄檔。Greengrass 核添加了一個旋轉時間戳記到日誌文件。

`deleteLogFileAfterCloudUpload`

(選擇性) 您可以在記錄管理員元件將記錄檔上傳至「記錄檔」之後刪除 CloudWatch 記錄檔。

預設：`false`

`multiLineStartPattern`

(選擇性) 識別新行上的記錄訊息何時是新記錄訊息的規則運算式。如果規則運算式不符合新行，則記錄管理員元件會將新行附加至上一行的記錄檔訊息。

根據預設，記錄管理員元件會檢查該行是否以空白字元開頭，例如索引標籤或空格。如果沒有，日誌管理器處理該行作為新的日誌消息。否則，它會將該行附加到當前日誌消息中。此行為可確保記錄檔管理員元件不會分割跨越多行的郵件，例如堆疊追蹤。

`periodicUploadIntervalSec`

(選擇性) 記錄管理員元件檢查要上傳之新記錄檔的期間 (以秒為單位)。

預設值：`300`(5 分鐘)

最小值：`0.000001`（1 微秒）

Example 範例：組態合併更新

下列範例組態指定將系統記錄檔和com.example.HelloWorld元件記錄檔上傳至 CloudWatch 記錄檔。

```
{
  "logsUploaderConfiguration": {
    "systemLogsConfiguration": {
      "uploadToCloudWatch": "true",
      "minimumLogLevel": "INFO",
      "diskSpaceLimit": "10",
      "diskSpaceLimitUnit": "MB",
      "deleteLogFileAfterCloudUpload": "false"
    },
    "componentLogsConfiguration": [
      {
        "componentName": "com.example.HelloWorld",
        "minimumLogLevel": "INFO",
        "logFileDirectoryPath": "/greengrass/v2/logs",
        "logFileRegex": "com.example.HelloWorld\\w*.log",
        "diskSpaceLimit": "20",
        "diskSpaceLimitUnit": "MB",
        "deleteLogFileAfterCloudUpload": "false"
      }
    ]
  },
  "periodicUploadIntervalSec": "300"
}
```

用量

記錄管理員元件會上傳至下列記錄群組和記錄資料流。

2.1.0 and later

記錄群組名稱

```
/aws/greengrass/componentType/region/componentName
```

記錄群組名稱會使用下列變數：

- componentType— 元件的類型，可以是下列其中一種：

  - `GreengrassSystemComponent`— 此記錄群組包含核心和外掛程式元件的記錄，這些元件與 Greengrass 核心在相同的 JVM 中執行。該組件是 Greengr ass 核的一部分。
  - `UserComponent`— 此日誌群組包含一般元件、Lambda 元件和裝置上其他應用程式的記錄。該組件不是 Greengrass 核的一部分。

  如需詳細資訊，請參閱 元件類型。
- `region`— 核心設備使用的AWS區域。
- `componentName`— 元件的名稱。對於系統記錄檔，此值為`System`。

記錄串流名稱

```
/date/thing/thingName
```

記錄資料流名稱會使用下列變數：

- `date`— 記錄的日期，例如`2020/12/15`。記錄管理員元件會使用此`yyyy/MM/dd`格式。
- `thingName`— 核心裝置的名稱。

> **Note**
>
> 如果物件名稱包含冒號 (`:`)，則記錄管理員會以加號 (`+`) 取代冒號。

2.0.x

記錄群組名稱

```
/aws/greengrass/componentType/region/componentName
```

記錄群組名稱會使用下列變數：

- `componentType`— 元件的類型，可以是下列其中一種：
  - `GreengrassSystemComponent`— 該組件是 Greengr ass 核的一部分.
  - `UserComponent`— 該組件不是 Greengrass 核的一部分. 日誌管理器將此類型用於設備上的 Greengrass 組件和其他應用程序。
- `region`— 核心設備使用的AWS區域。
- `componentName`— 元件的名稱。對於系統記錄檔，此值為`System`。

記錄串流名稱

```
/date/deploymentTargets/thingName
```

記錄資料流名稱會使用下列變數：

- date— 記錄的日期，例如2020/12/15。記錄管理員元件會使用此yyyy/MM/dd格式。
- deploymentTargets— 其部署包含元件的物件。記錄管理員元件會以斜線分隔每個目標。如果元件因為本機部署而在核心裝置上執行，則此值為LOCAL_DEPLOYMENT。

  假設您有一個名為的核心裝置MyGreengrassCore，且核心裝置具有兩個部署的範例：

  - 以核心裝置為目標的部署MyGreengrassCore。
  - 以名稱為MyGreengrassCoreGroup物件群組 (包含核心裝置) 為目標的部署。

  這deploymentTargets個核心設備是thing/MyGreengrassCore/thinggroup/MyGreengrassCoreGroup。
- thingName— 核心裝置的名稱。

記錄項目的格式。

格 Greengrass 核以字符串或 JSON 格式寫入日誌文件。對於系統記錄檔，您可以透過設定logging項目的format欄位來控制格式。您可以在 Greengrass 核組件的配置文件中找到該logging條目。如需詳細資訊，請參閱 Greengrass 核組態。

文字格式為自由格式，可接受任何字串。下列叢集狀態服務訊息是字串格式化記錄的範例：

```
2023-03-26T18:18:27.271Z [INFO] (pool-1-thread-2)
com.aws.greengrass.status.FleetStatusService: fss-status-update-published.
Status update published to FSS. {trigger=CADENCE, serviceName=FleetStatusService,
currentState=RUNNING}
```

如果您想要使用 Greengrass CLI 記錄檔命令檢視記錄檔，或以程式設計方式與記錄檔互動，請使用 JSON 格式。下面的例子概述了 JSON 形狀：

```
{
   "loggerName": <string>,
   "level": <"DEBUG" | "INFO" | "ERROR" | "TRACE" | "WARN">,
   "eventType": <string, optional>,
```

```
    "cause": <string, optional>,
    "contexts": {},
    "thread": <string>,
    "message": <string>,
    "timestamp": <epoch time> # Needs to be epoch time
}
```

若要控制元件記錄檔的輸出，您可以使用minimumLogLevel組態選項。若要使用此選項，您的元件必須以 JSON 格式寫入其記錄項目。您應該使用與系統記錄檔相同的格式。

本機記錄檔

此元件使用與 Greengrass 核元件相同的記錄檔。

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

若要檢視此元件的記錄

- 在核心裝置上執行下列命令，即時檢視此元件的記錄檔。以AWS IoT Greengrass根資料夾的路徑取代*/greengrass/v2*或 *C:\greengrass\v2*。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

變更記錄

下表說明元件每個版本的變更。

版本	變更
2.3.7	版本更新 Greengrass 2.12.0 版本釋放。
2.3.6	錯誤修復和改進 • 調整某些錯誤的日誌級別。
2.3.5	改善項目 提高了日誌上傳速度。 版本更新 Greengrass 2.11.0 版本釋放。
2.3.4	錯誤修復和改進 • 添加對將參數設置為分`periodicUploadIntervalSec` 數值的支援。最小值為 1 微秒。 • 修正記錄管理員不遵守 CloudWatch`putLogEvents` 限制的問題。
2.3.3	版本更新了 Greengrass 2.10.0 版本。
2.3.2	錯誤修復和改進 • 改進了空間管理，以便在上傳之前不會刪除日誌文件。 • 修復了緩存管理的問題。 • 其他小錯誤修復和改進。
2.3.1	錯誤修復和改進 • 修正了具有多個活動日誌文件的目標文件組將重複條目上傳到 CloudWatch的問題。 • 其他小錯誤修復和改進。
2.3.0	**Note** 我們建議您升級至記錄檔管理員 2.3.0 時，升級至 Greengrass 核心 2.9.1。

版本	變更
	新功能 透過處理和直接上傳作用中的記錄檔，而不是等待輪換新檔案來減少記錄延遲。 錯誤修復和改進 • 改進了旋轉具有唯一名稱的文件時對日誌旋轉的支持。 • 其他小錯誤修復和改進。
2.2.8	版本更新 Greengrass 2.9.0 版本釋放。
2.2.7	版本更新 Greengrass 2.8.0 版本的版本。
2.2.6	版本更新了 Greengrass 核 2.7.0 版本釋放。
2.2.5	版本更新 Greengrass 2.6.0 版本發布。
2.2.4	錯誤修復和改進 • 改善處理無效組態時的穩定性。 • 其他小修正和改進。
2.2.3	錯誤修復和改進 • 改善元件重新啟動或發生錯誤的特定情況下的穩定性。 • 修正在特定情況下無法上傳大型記錄訊息和大型記錄檔的問題。 • 修正此元件如何處理組態重設更新的問題。 • 修正組`nulldiskSpaceLimit` 態值造成元件無法部署的問題。
2.2.2	錯誤修復和改進 • 添加對大於 256 KB 的日誌消息的支持。記錄管理員元件會將這些大型記錄檔訊息分割成具有相同記錄事件時間戳記的多個訊息。
2.2.1	版本更新了 Greengrass 核 2.5.0 版本。

版本	變更
2.2.0	新功能 • 新增組componentLogsConfigurationMap 態參數以支援元件記錄組態的對映格式。對映中的每個componentName 物件都會定義元件或應用程式的記錄組態。
2.1.3	版本更新 Greengrass 2.4.0 版本的版本。
2.1.2	版本更新了 Greengrass 核 2.3.0 版本。
2.1.1	錯誤修復和改進 • 修正了某些情況下系統日誌配置未更新的問題。
2.1.0	錯誤修復和改進 • 對於打印到標準輸出（標準輸出）logFileDirectoryPath 和logFileRegex 標準錯誤（stderr）的 Greengrass 組件使用默認值和工作。 • 將記錄檔上傳至 CloudWatch 記錄檔時，透過設定的網路 Proxy 正確路由傳送流量。 • 正確處理記錄資料流名稱中的冒號字元 (:)。 CloudWatch 記錄檔資料流名稱不支援冒號。 • 從記錄串流中移除物件群組名稱，以簡化記錄資料流名稱。 • 移除在正常行為期間列印的錯誤記錄訊息。
2.0.x	初始版本。

機器學習元件

AWS IoT Greengrass提供下列機器學習元件，您可以將這些元件部署到支援的裝置，以執行機器學習推論，使用在 Amazon 訓練的模型 SageMaker 或您自己預先訓練的模型 (儲存在 Amazon S3 中) 執行機器學習推論。

AWS提供下列類別的機器學習元件：

- 模型元件 — 包含做為 Greengrass 人工因素的機器學習模型。
- 執行階段元件 — 包含在 Greengrass 核心裝置上安裝機器學習架構及其相依性的指令碼。

- 推論元件：包含推論程式碼並包含元件相依性，可安裝機器學習架構和下載預先訓練的機器學習模型。

您可以使用AWS提供的機器學習元件中的範例推論程式碼和預先訓練的模型，使用 DLR 和 Lite 執行影像分類和物件偵測。 TensorFlow 若要使用儲存在 Amazon S3 中的自己模型執行自訂機器學習推論，或使用不同的機器學習架構，您可以使用這些公用元件的配方做為範本來建立自訂機器學習元件。如需詳細資訊，請參閱 自訂您的機器學習元件。

AWS IoT Greengrass也包含一個AWS提供的元件，用來管理 Greengrass 核心裝置上 SageMaker Edge Manager 代理程式的安裝和生命週期。透過 SageMaker 邊緣管理員，您可以直接在核心裝置上使用 Amazon SageMaker 新編譯的模型。如需詳細資訊，請參閱 在核心設備上使用 Amazon SageMaker 邊緣管理器。

下表列出中可用的機器學習元件AWS IoT Greengrass。

> **Note**
>
> 幾個AWS提供的組件取決於 Greengrass 核的特定次要版本。由於這種依賴關係，您需要在將 Greengrass 核更新為新的次要版本時更新這些組件。如需每個元件所依賴之特定原子核版本的詳細資訊，請參閱對應的元件主題。如需更新核心的詳細資訊，請參閱更新AWS IoT Greengrass核心軟件（OTA）。

當組件具有泛型和 Lambda 的組件類型時，該組件的當前版本是泛型類型，而先前版本的組件是 Lambda 類型。

元件	描述	元件類型	支援的作業系統	開放原始碼
Lookout for Vision 邊緣代理	在 Greengrass 核心裝置上部署適用於視覺的 Amazon Lookout 執行階段，因此您可以使用電腦視覺來尋找工	一般	Linux	否

元件	描述	元件類型	支援的作業系統	開放原始碼
	業產品中的瑕疵。			
SageMaker 邊緣管理員	在 Greengrass 核心裝置上部署 Amazon SageMaker 邊緣管理器代理程式。	一般	Linux、Windows	否
DLR 影像分類	使用 DLR 影像分類模型存放區和 DLR 執行階段元件做為相依性的推論元件，以安裝 DLR、下載範例影像分類模型，以及在支援的裝置上執行影像分類推論。	一般	Linux、Windows	否

元件	描述	元件類型	支援的作業系統	開放原始碼
DLR 物體偵測	使用 DLR 物件偵測模型存放區和 DLR 執行階段元件做為相依性的推論元件，以安裝 DLR、下載範例物件偵測模型，以及在支援的裝置上執行物件偵測推論。	一般	Linux、Windows	否
DLR 圖像分類模型商店	包含示例 ResNet -50 圖像分類模型作為 Greengrass 工件的模型組件。	一般	Linux、Windows	否
DLR 物件偵測模型商店	包含樣本 Yolov3 對象檢測模型作為綠色工件的模型組件。	一般	Linux、Windows	否
DLR 執行階段	包含用於在 Greengrass 核心裝置上安裝 DLR 及其相依性的安裝指令碼的執行階段元件。	一般	Linux、Windows	否

元件	描述	元件類型	支援的作業系統	開放原始碼
TensorFlow 精簡版圖片分類	使用 TensorFlow Lite 影像分類模型存放區和 Lite 執行階段元件做為相依性的推論元件，以安裝 TensorFlow Lite、下載範例影像分類模型，以及在支援的裝置上執行影像分類推論。TensorFlow	一般	Linux、Windows	否
TensorFlow 精簡型物體偵測	使用 TensorFlow Lite 物件偵測模型存放區和 Lite 執行階段元件做為相依性的推論元件，以安裝 TensorFlow Lite、下載範例物件偵測模型，以及在支援的裝置上執行物件偵測推論。TensorFlow	一般	Linux、Windows	否

元件	描述	元件類型	支援的作業系統	開放原始碼
TensorFlow 精簡版圖片分類模型店	包含做為 Greengrass 人工因素的 MobileNet v1 模型範例的模型元件。	一般	Linux、Windows	否
TensorFlow 精簡型物件偵測模型商店	包含做為 Greengrass 假影的範例單次發射偵測 (SSD) MobileNet 模型的模型元件。	一般	Linux、Windows	否
TensorFlow 精簡版運行	包含用於在 Greengrass 核心裝置上安裝 TensorFlow Lite 及其相依性的安裝指令碼的執行階段元件。	一般	Linux、Windows	否

Lookout for Vision 邊緣代理

Lookout for Vision 邊緣代理元件 (`aws.iot.lookoutvision.EdgeAgent`) 會安裝本機 Amazon Lookout for Vision 執行階段伺服器，該伺服器使用電腦視覺來尋找工業產品中的視覺缺陷。

若要使用此元件，請建立並部署 Lookout for Vision 機器學習模型元件。這些機器學習模型會在您用來訓練模型的影像中尋找圖樣，來預測影像中異常的存在。然後，您可以開發和部署自訂 Greengrass 元件 (稱為用戶端應用程式元件)，將影像和視訊串流提供給此執行階段元件，以便使用機器學習模型偵測異常。

您可以使 Lookout for Vision 邊緣代理 API 與此元件從其他 Greengrass 元件進行交互。此 API 是使用 gRPC 來實作，GrPC 是用來進行遠端程序呼叫的通訊協定。如需詳細資訊，請參閱 Amazon Lookout 視覺開發人員指南中的撰寫用戶端應用程式元件和 Lookout for Vision 邊緣代理程式 API 參考。

如需如何使用此元件的詳細資訊，請參閱下列內容：

- Greengrass out out out out out out out out out out on Vision
- 什麼是 Amazon Lookout for Vision？ 在 Amazon Lookout for Vision 開發人員指南
- 在 Amazon Lookout for Vision 開發人員指南中創 Lookout for Vision 模型。
- 在 Amazon Lookout for Vision 開發人員指南中的邊緣裝置上使 Lookout for Vision 模型。

> **Note**
>
> Lookout for Vision 邊緣代理程式元件僅適用於下列項目 AWS 區域：
>
> - 美國東部 (俄亥俄)
> - 美國東部 (維吉尼亞北部)
> - 美國西部 (奧勒岡)
> - 歐洲 (法蘭克福)
> - 歐洲 (愛爾蘭)
> - 亞太區域 (東京)
> - 亞太區域 (首爾)

主題

- 版本
- Type
- 作業系統
- 要求
- 相依性
- 組態
- 本機記錄檔
- 變更記錄

版本

此元件具有下列版本：

- 1.2.x
- 1.x
- 1.0.x
- 0.1.x

Type

此元件是一般元件 (`aws.greengrass.generic`)。Greengrass 核會執行元件的生命週期指令碼。

如需詳細資訊，請參閱 元件類型。

作業系統

此元件只能安裝在 Linux 核心裝置上。

要求

此元件具有下列需求：

- 格 Greengrass 核心設備必須使用阿姆瓦 8（AArch64）或 x86_64 體系結構。
- 如果您使用版本 1.0.0 或更高版本的這個組件，Python 3.8 或 Python 3.9，包括`pip`，安裝在 Greengrass 核心設備上。

  如果您使用版本 0.1.x 的這個組件，Python 3.7，包括`pip`，安裝在 Greengrass 核心設備上。

  > **Important**
  >
  > 該設備必須具有以下確切版本的 Python 之一。這個組件不支持 Python 的更新版本。

- 若要使用圖形處理單元 (GPU) 推論，核心裝置必須符合下列需求。GPU 推論在此元件的版本 1.1.0 及更新版本中是選用的。
  - 一種支援 CUDA 的圖形處理單元 (GPU)。如需詳細資訊，請參閱 CUDA 工具組文件中的確認您擁有支援 CUDA 的 GPU。
  - 安裝在 Greengrass 核心設備上的 CUDNN，CUDA 和 TensorRT。

- 在 NVIDIA 的傑特森設備，如傑特森納米或傑特森·澤維爾，CuDNN，CUDA 和 TensorRT 都安裝了 NVIDIA。 JetPack您不需要進行任何變更。這個元件支援 JetPack 4.4、JetPack4.5、JetPack 4.5.1 和 JetPack4. 6.1。

  > **Important**
  >
  > 您必須安裝這些版本的其中一個版本， JetPack 而不是安裝其他版本。「觀察視覺」服務會針對這些 JetPack平台編譯電腦視覺模型。

- 在配備具有 NVIDIA 安培微架構 (或 GPU 運算容量為 8.0) 的 GPU 的 x86 裝置上，請執行下列動作：
  - 依照 NVIDIA 安裝指南中的指示來安裝 CUDN N。
  - 依照 NVIDIA CUDA 安裝指南中的指示，安裝 CUDA 11.2 版。
  - 請依照 NVIDIA TensorRT 說明文件中的指示，安裝 TensorRT 版本 8.2.0。
- 在具有在安培之前具有 NVIDIA 架構的 GPU 的 x86 裝置上 (或 GPU 的運算容量小於 8.0)，請執行下列動作：
  - 依照 NVIDIA 安裝指南中的指示來安裝 CUDN N。
  - 依照 NVIDIA CUDA 安裝指南中的指示，安裝 CUDA 10.2 版。
  - 依照 NVIDIA TensorRT 說明文件中的指示，安裝版本 7.1.3 或更新版本，但是早於 8.0.0 版。
- 執行此元件的系統使用者必須是可存取裝置上 GPU 之系統群組的成員。此群組的名稱因作業系統而異。請參閱作業系統和 GPU 的說明文件，以判斷此系統群組的名稱。

  例如，在 NVIDIA Jetson 裝置上，此群組的名稱為`video`，您可以執行下列命令，將系統使用者新增至此群組。以要新增的`####### ggc_` user。

  ```
  sudo usermod -aG video ggc_user
  ```

相依性

這個組件沒有任何依賴關係。

組態

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

`Socket`

(選擇性) Edge 代理程式運作的檔案通訊端。Lookout for Vision 模型元件使用此檔案通訊端與邊緣代理程式通訊。如果您變更此參數，您必須在部署 Lookout for Vision 模型元件時指定相同的值。

預設：`unix:///tmp/aws.iot.lookoutvision.EdgeAgent.sock`

本機記錄檔

此元件使用下列記錄檔。

```
/greengrass/v2/logs/aws.iot.lookoutvision.EdgeAgent.log
```

若要檢視此元件的記錄

- 在核心裝置上執行下列命令，即時檢視此元件的記錄檔。以 AWS IoT Greengrass 根資料夾的路徑取*/greengrass/v2*代。

  ```
  sudo tail -f /greengrass/v2/logs/aws.iot.lookoutvision.EdgeAgent.log
  ```

變更記錄

下表說明元件每個版本的變更。

版本	變更
1.2.0	一般錯誤修正與改進。
1.1.9	一般錯誤修正與改進。
1.1.8	一般錯誤修正與改進。
1.1.7	新功能 • 在 Lookout for Vision 邊緣代理程式虛擬環境中安裝`opencv-python-headless` 套件。 錯誤修復和改進 • 改善置信度分數計算。 • 將熱圖模型遮罩調整為原始檔案大小。

版本	變更
	• 一般錯誤修正與改進。
1.1.6	新功能 增加了新的值的DetectAnomalies 結果。 • anomaly_score — 介於 0.0 到 1.0 之間的數字，表示影像有多異常。 • anomaly_threshold — 在模型訓練期間設定的臨界值，可決定異常影像與一般影像之間的邊界。 一般錯誤修正與改進。
1.1.4	新功能 增加了對 OpenCV 的圖像調整大小的支持（如果可用）。邊緣代理使用枕頭時 OpenCV 不可用。 錯誤修復和改進 一般錯誤修正與改進。
1.1.3	一般錯誤修正與改進。
1.1.1	一般錯誤修正與改進。
1.1.0	新功能 • 添加對圖像分割模型的支持，以識別圖像中的異常。 • 添加對 CPU 推論的支持，因此您可以在沒有 GPU 的核心設備上使用觀察視覺模型。 錯誤修復和改進 • 一般錯誤修正與改進。
1.0.0	這個版本的 Lookout for Vision 邊緣代理程式元件需要不同於版本的 Python 版本 0.1.x。如果您想要從 v0.1.x 升級到 v1.x，您必須升級核心裝置上的 Python 安裝。 錯誤修復和改進 • 一般錯誤修正與改進。

版本	變更
0.1.37	一般錯誤修正與改進。
0.1.36	初始版本。

SageMaker 邊緣管理員

> **Important**
>
> SageMaker 邊緣管理員將於 2024 年 4 月 26 日停止使用。如需有關繼續將模型部署到邊緣裝置的詳細資訊，請參閱SageMaker 邊緣管理員生命週期結束。

Amazon 邊 SageMaker 緣管理器元件 (`aws.greengrass.SageMakerEdgeManager`) 會安裝 SageMaker 邊緣管理器代理程式二進位檔。

SageMaker Edge Manager 為邊緣裝置提供模型管理功能，讓您可以最佳化、保護、監控和維護邊緣裝置叢集上的機器學習模型。 SageMaker Edge 管理員元件會在您的核心裝置上安裝和管理邊 SageMaker 緣管理員代理程式的生命週期。您也可以使用 SageMaker Edge 管理員來封裝並使用 SageMaker 新編譯的模型做為 Greengrass 核心裝置上的模型元件。如需有關在核心裝置上使用 SageMaker Edge Manager 代理程式的詳細資訊，請參閱在核心設備上使用 Amazon SageMaker 邊緣管理器。

SageMaker 邊緣管理員元件 v1.3.x 會安裝邊緣管理員代理程式二進位檔 如需 Edge 管理員代理程式二進位版本的相關資訊，請參閱邊緣管理員代理

> **Note**
>
> 「 SageMaker 邊緣管理員」元件僅適用於下列項目AWS 區域：
>
> - 美國東部 (俄亥俄)
> - 美國東部 (維吉尼亞北部)
> - 美國西部 (奧勒岡)
> - 歐洲 (法蘭克福)
> - 歐洲 (愛爾蘭)
> - 亞太區域 (東京)

主題

版本

此元件具有下列版本：

- 1.3.x
- 1.2.x
- 1.x
- 1.0.x

Type

此元件是一般元件 (`aws.greengrass.generic`)。Greengrass 核會執行元件的生命週期指令碼。

如需詳細資訊，請參閱 元件類型。

作業系統

此元件可安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件具有下列需求：

- 在 Amazon Linux 2，基於 Debian 的 Linux 平台（x86_64 或 Armv8）或視窗（x86_64）上運行的 Greengrass 核心設備。如果您沒有帳戶，請參閱 教學課程：AWS IoT Greengrass V2 入門。
- Python 3.6 或更高版本，包括您pip的 Python 版本，安裝在您的核心設備上。
- 設定了以下內容的 Greengrass 裝置角色：
  - 允許credentials.iot.amazonaws.com和擔任角色sagemaker.amazonaws.com的信任關係，如以下 IAM 政策範例所示。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
       },
      "Action": "sts:AssumeRole"
    },
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "sagemaker.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

  - AmazonSageMakerEdgeDeviceFleetPolicyIAM 受管政策。
  - 動s3:PutObject作，如以下 IAM 政策範例所示。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:PutObject"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
```

```
    }
  ]
}
```

- 在AWS 帳戶與AWS 區域您的 Greengrass 核心裝置相同建立的 Amazon S3 儲存貯體。 SageMaker Edge Manager 需要 S3 儲存貯體來建立邊緣裝置叢集，並存放在裝置上執行推論時所產生的範例資料。如需建立 S3 儲存貯體的相關資訊，請參閱開始使用 Amazon S3。
- SageMaker 邊緣裝置叢集使用與 Greengrass 核心裝置相同的AWS IoT角色別名。如需詳細資訊，請參閱 建立邊緣裝置叢集。
- 您的 Greengrass 核心裝置已註冊為 Edge 裝置群組中的邊 SageMaker 緣裝置。Edge 裝置名稱必須與核心裝置的AWS IoT物件名稱相符。如需詳細資訊，請參閱 註冊 Greengrass 設備。

端點和連接埠

除了基本作業所需的端點和連接埠之外，此元件還必須能夠對下列端點和連接埠執行輸出要求。如需詳細資訊，請參閱 允許裝置流量透過 Proxy 或防火牆。

端點	連線埠	必要	描述
`edge.sagemaker.` *`region`*`.amazonaws.com`	443	是	檢查裝置註冊狀態並將指標傳送至 SageMaker。
`*.s3.amazonaws.com`	443	是	將擷取資料上傳到您指定的 S3 儲存貯體。 您可以*使用上傳資料的每個儲存貯體的名稱來取代。

相依性

部署元件時，AWS IoT Greengrass也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的需求，才能成功部署元件。本節列出此元件之已發行版本的相依性，以及定義每個相依性之元件版本的語意版本條件約束。您也可以在AWS IoT Greengrass主控台中檢視元件每個版本的相依性。在元件詳細資料頁面上，尋找 [相依性] 清單。

1.3.5

下表列出此元件 1.3.5 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
代幣交換服務	>=0.0.0	硬式

1.3.4

下表列出此元件 1.3.4 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
代幣交換服務	>=0.0.0	硬式

1.3.3

下表列出此元件 1.3.3 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
代幣交換服務	>=0.0.0	硬式

1.3.2

下表列出此元件 1.3.2 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
代幣交換服務	>=0.0.0	硬式

1.3.1

下表列出此元件 1.3.1 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.0.0	軟式
代幣交換服務	>=0.0.0	硬式

1.1.1 - 1.3.0

下表列出此元件 1.1.1-1.3.0 版本的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
代幣交換服務	>=0.0.0	硬式

1.1.0

下表列出此元件 1.1.0 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.0.0	軟式

相依性	兼容版本	相依性類型
代幣交換服務	>=0.0.0	硬式

1.0.3

下表列出此元件 1.0.3 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
代幣交換服務	>=0.0.0	硬式

1.0.1 and 1.0.2

下表列出此元件 1.0.1 和 1.0.2 版本的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
代幣交換服務	>=0.0.0	硬式

1.0.0

下表列出此元件 1.0.0 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
代幣交換服務	>=0.0.0	硬式

如需有關元件相依性的詳細資訊，請參閱元件方案參考。

組態

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

> **Note**
>
> 本節說明您在元件中設定的組態參數。如需對應 SageMaker 邊緣管理員組態的詳細資訊，請參閱 Amazon SageMaker 開發人員指南中的邊緣管理員代理程式。

`DeviceFleetName`

包含您 Greengrass 核心裝置之 SageMaker 邊緣管理員裝置叢集的名稱。

部署此元件時，您必須在組態更新中指定此參數的值。

`BucketName`

您上傳擷取的推論資料的 S3 儲存貯體名稱。值區名稱必須包含字串`sagemaker`。

如果您設定`CaptureDataDestination`為`Cloud`，或設定`CaptureDataPeriodicUpload`為`true`，則您必須在部署此元件時在組態更新中指定此參數的值。

> **Note**
>
> 擷取資料是一 SageMaker 項功能，可用來將推論輸入、推論結果和其他推論資料上傳至 S3 儲存貯體或本機目錄，以供 future 分析使用。如需有關透過 SageMaker Edge Manager 使用擷取資料的詳細資訊，請參閱 Amazon SageMaker 開發人員指南中的管理模型。

`CaptureDataBatchSize`

(選擇性) 代理程式處理的一批擷取資料要求大小。此值必須小於您在中指定的緩衝區大小`CaptureDataBufferSize`。我們建議您不要超過緩衝區大小的一半。

代理程式會在緩衝區中的要求數目符合`CaptureDataBatchSize`數目時處理要求批次，或是`CaptureDataPushPeriodSeconds`間隔過去時 (以先發生者為準)。

預設：`10`

`CaptureDataBufferSize`

(選擇性) 儲存在緩衝區中的擷取資料要求數目上限。

預設：`30`

`CaptureDataDestination`

(選擇性) 儲存擷取資料的目的地。此參數可以具有下列值：

- `Cloud`— 將擷取的資料上傳到您在中`BucketName`指定的 S3 儲存貯體。
- `Disk`將擷取的資料寫入元件的工作目錄。

如果您指定`Disk`，您也可以將設定為，選擇定期將擷取的資料上傳`CaptureDataPeriodicUpload`到 S3 儲存貯體`true`。

預設：`Cloud`

`CaptureDataPeriodicUpload`

(選擇性) 字串值，指定是否定期上傳擷取的資料。支援的值為 `true` 和 `false`。

`true`如果您設`CaptureDataDestination`定為`Disk`，並且您也希望代理程式定期上傳您的 S3 儲存貯體擷取的資料，請將此參數設定為。

預設：`false`

`CaptureDataPeriodicUploadPeriodSeconds`

(選擇性) SageMaker Edge Manager 代理程式將擷取的資料上傳到 S3 儲存貯體的間隔 (以秒為單位)。如果設定`CaptureDataPeriodicUpload`為，請使用此參數`true`。

預設：`8`

`CaptureDataPushPeriodSeconds`

(選擇性) SageMaker Edge Manager 代理程式處理來自緩衝區之一批擷取資料要求的間隔 (秒)。

代理程式會在緩衝區中的要求數目符合`CaptureDataBatchSize`數目時處理要求批次，或是`CaptureDataPushPeriodSeconds`間隔過去時 (以先發生者為準)。

預設：`4`

`CaptureDataBase64EmbedLimit`

(選擇性) SageMaker Edge Manager 代理程式上傳的擷取資料大小上限 (以位元組為單位)。

預設：3072

FolderPrefix

(選擇性) 代理程式寫入擷取資料的資料夾名稱。如果設定CaptureDataDestination為Disk，則代理程式會在指定的目錄中建立資料夾CaptureDataDiskPath。如果您設定CaptureDataDestination為Cloud，或設定CaptureDataPeriodicUpload為true，則代理程式會在 S3 儲存貯體中建立資料夾。

預設：sme-capture

CaptureDataDiskPath

此功能在 v1.1.0 及更新版本的「 SageMaker 邊緣管理員」元件中提供。

(選擇性) 代理程式建立擷取的資料資料夾的資料夾路徑。如果設定CaptureDataDestination為Disk，代理程式會在此目錄中建立擷取的資料資料夾。如果未指定此值，則代理程式會在元件的工作目錄中建立擷取的資料資料夾。使用FolderPrefix參數可指定擷取之資料資料夾的名稱。

預設：*/greengrass/v2*/work/aws.greengrass.SageMakerEdgeManager/capture

LocalDataRootPath

此功能可在 v1.2.0 和更新版本的 SageMaker 邊緣管理員元件中使用。

(選擇性) 此元件在核心裝置上儲存下列資料的路徑：

- 當您設DbEnable定為時，執行階段資料的本機資料庫true。
- SageMaker 當您設DeploymentEnable定為時，此元件會自動下載的新編譯模型。true

預設：*/greengrass/v2*/work/aws.greengrass.SageMakerEdgeManager

DbEnable

(選擇性) 您可以啟用此元件將執行階段資料儲存在本機資料庫中，以便在元件故障或裝置中斷電源時保留資料。

此資料庫在核心裝置的檔案系統上需要 5 MB 的儲存空間。

預設：false

DeploymentEnable

此功能可在 v1.2.0 和更新版本的 SageMaker 邊緣管理員元件中使用。

(選擇性) 您可以啟用此元件，自動從您上傳到 Amazon S3 擷取 SageMaker 新編譯的模型。將新模型上傳到 Amazon S3 之後，請使用 SageMaker Studio 或 SageMaker API 將新模型部署到此核心裝置。啟用此功能時，您可以將新模型部署到核心裝置，而不需要建立AWS IoT Greengrass部署。

> **Important**
>
> 若要使用此功能，您必須`DbEnable`將設定為`true`。此功能使用本機資料庫來追蹤從中擷取的模型AWS 雲端。

預設：`false`

`DeploymentPollInterval`

此功能可在 v1.2.0 和更新版本的 SageMaker 邊緣管理員元件中使用。

(選擇性) 此元件檢查要下載的新模型之間的時間量 (以分鐘為單位)。當您設定`DeploymentEnable`為時，會套用此選項`true`。

預設值：`1440`(1 天)

`DLRBackendOptions`

此功能可在 v1.2.0 和更新版本的 SageMaker 邊緣管理員元件中使用。

(選擇性) 要在 DLR 執行階段中設定此組件使用的 DLR 執行階段旗標。您可以設定下列旗標：

- `TVM_TENSORRT_CACHE_DIR`— 啟用快取模型。指定具有讀取/寫入權限的現有資料夾的絕對路徑。
- `TVM_TENSORRT_CACHE_DISK_SIZE_MB`— 指定 TensorRT 模型快取資料夾的上限。當目錄大小超出此限制時，會刪除使用最少的快取引擎。預設值為 512 MB。

例如，您可以將此參數設定為下列值，以啟用 TensorRT 模型快取，並將快取大小限制為 800 MB。

```
TVM_TENSORRT_CACHE_DIR=/data/secured_folder/trt/cache;
 TVM_TENSORRT_CACHE_DISK_SIZE_MB=800
```

`SagemakerEdgeLogVerbose`

(選擇性) 字串值，指定是否啟用偵錯記錄。支援的值為 `true` 和 `false`。

預設：`false`

`UnixSocketName`

(選擇性) SageMaker Edge 管理員通訊端檔案描述元在核心裝置上的位置。

預設：`/tmp/aws.greengrass.SageMakerEdgeManager.sock`

Example 範例：組態合併更新

下列範例組態指定核心裝置是該核心裝置的一部分，*MyEdgeDeviceFleet*且代理程式會將擷取資料同時寫入裝置和 S3 儲存貯體。此設定也會啟用偵錯記錄。

```
{
    "DeviceFleetName": "MyEdgeDeviceFleet",
    "BucketName": "DOC-EXAMPLE-BUCKET",
    "CaptureDataDestination": "Disk",
    "CaptureDataPeriodicUpload": "true",
    "SagemakerEdgeLogVerbose": "true"
}
```

本機記錄檔

此元件會使用下列記錄檔。

Linux

```
/greengrass/v2/logs/aws.greengrass.SageMakerEdgeManager.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.SageMakerEdgeManager.log
```

若要檢視此元件的記錄

- 在核心裝置上執行下列命令，即時檢視此元件的記錄檔。以AWS IoT Greengrass根資料夾的路徑取代*/greengrass/v2*或 *C:\greengrass\v2*。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.SageMakerEdgeManager.log
  ```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\aws.greengrass.SageMakerEdgeManager.log -Tail
 10 -Wait
```

變更記錄

下表說明元件每個版本中的變更。

版本	變更
1.3.5	版本更新了 Greengrass 核 2.12.0 版本釋放。
1.3.4	版本更新了 Greengrass 核 2.11.0 版本釋放。
1.3.3	版本更新了 Greengrass 核 2.10.0 版本。
1.3.2	版本更新 Greengrass 2.9.0 版本釋放。
1.3.1	版本更新 Greengrass 2.8.0 版本的版本。
1.3.0	新功能 • 添加對 TensorRT 緩存磁盤大小管理的支持。 • 將選用TVM_TENSORRT_CACHE_DISK_SIZE_MB 旗標新增至 DLR BackendOptions 參數，以設定磁碟上快取模型的大小限制。 改善項目 • 提供改進的預測並發性。這有助於更好地使用裝置加速器引擎，例如 GPU。
1.2.0	新功能 • 新增對此元件的支援，以自動擷取您上傳到 Amazon S3 的新 SageMaker 編譯模型。啟用此功能時，您可以將新模型部署到核心裝置，而不需要建立AWS IoT Greengrass部署。 • 添加對此組件用於保留運行時數據的備份數據庫的支持，以防組件故障或設備中斷電源。 • 添加了配置此組件時配置 DLR 運行時標誌的支持。

版本	變更
1.1.1	版本更新了 Greengrass 核 2.7.0 版本釋放。
1.1.0	新功能 • 添加對運行 Amazon Linux 2 的 Greengrass 核心設備的支持。 • 加入新的`CaptureDataDiskPath` 組態參數。您可以使用此參數來指定裝置上擷取之資料資料夾的路徑。 錯誤修復和改進 • 版本更新了 Greengrass 核 2.5.0 版本。
1.0.3	版本更新 Greengrass 2.4.0 版本的版本。
1.0.2	錯誤修復和改進 更新元件生命週期中的安裝指令碼。您的核心裝置現在必須在裝置上安裝 Python 3.6 或更新版本 (包括`pip`您的 Python 版本)，才能部署此元件。
1.0.1	版本更新了 Greengrass 核 2.3.0 版本。
1.0.0	初始版本。

DLR 影像分類

DLR 影像分類元件 (`aws.greengrass.DLRImageClassification`) 包含範例推論程式碼，可使用深度學習執行階段和 resnet-50 模型來執行影像分類推論。此元件使用變體DLR 圖像分類模型商店和DLR 執行階段元件作為從屬關係來下載 DLR 和範例模型。

若要將此推論元件與自訂訓練有素的 DLR 模型搭配使用，請建立相依模型存放區元件的自訂版本。若要使用您自己的自訂推論程式碼，您可以使用此元件的方式做為範本來建立自訂推論元件。

主題

- 版本
- Type
- 作業系統
- 要求
- 相依性

- 組態
- 本機記錄檔
- 變更記錄

版本

此元件具有下列版本：

- 2.1.x
- 2.0.x

Type

此元件是一般元件 (`aws.greengrass.generic`)。Greengrass 核會執行元件的生命週期指令碼。

如需詳細資訊，請參閱 元件類型。

作業系統

此元件可安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件具有下列需求：

- 在運行 Amazon Linux 2 或 Ubuntu 18.04 的 Greengrass 核心設備上，GNU C 庫（glibc）2.27 版或更高版本安裝在設備上。
- 在 ARMV7L 設備上，如樹莓派，對於設備上安裝了 OpenCV-Python 的依賴關係。執行下列命令以安裝相依性。

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
 libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- 樹莓派運行樹莓派操作系統靶心設備必須滿足以下要求:
  - NumPy 1.22.4 或更新版本已安裝在裝置上。樹莓派 OS 靶心包括早期版本的 NumPy，因此您可以在設備上運行以下命令進 NumPy 行升級。

```
pip3 install --upgrade numpy
```

- 裝置上啟用的舊式攝影機堆疊。樹莓派 OS Bullseye 包括一個新的相機堆疊，預設情況下啟用且不相容，因此您必須啟用傳統的相機堆疊。

  啟用舊式相機堆疊

  1. 運行以下命令以打開樹莓派配置工具。

     ```
     sudo raspi-config
     ```

  2. 選取介面選項。
  3. 選取「舊式相機」以啟用舊式相機堆疊。
  4. 重新啟動 Raspberry Pi。

相依性

部署元件時，AWS IoT Greengrass也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的需求，才能成功部署元件。本節列出此元件之已發行版本的相依性，以及定義每個相依性之元件版本的語意版本條件約束。您也可以在AWS IoT Greengrass主控台中檢視元件每個版本的相依性。在元件詳細資料頁面上，尋找 [相依性] 清單。

2.1.13

下表列出此元件 2.1.13 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
DLR 圖像分類模型商店	約 2.1.0	硬式
DLR	約 1.6.0	硬式

2.1.12

下表列出此元件 2.1.12 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
DLR 圖像分類模型商店	約 2.1.0	硬式
DLR	約 1.6.0	硬式

2.1.11

下表列出此元件 2.1.11 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
DLR 圖像分類模型商店	約 2.1.0	硬式
DLR	約 1.6.0	硬式

2.1.10

下表列出此元件 2.1.10 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
DLR 圖像分類模型商店	約 2.1.0	硬式
DLR	約 1.6.0	硬式

2.1.9

下表列出此元件 2.1.9 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
DLR 圖像分類模型商店	約 2.1.0	硬式
DLR	約 1.6.0	硬式

2.1.8

下表列出此元件 2.1.8 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
DLR 圖像分類模型商店	約 2.1.0	硬式
DLR	約 1.6.0	硬式

2.1.7

下表列出此元件 2.1.7 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
DLR 圖像分類模型商店	約 2.1.0	硬式
DLR	約 1.6.0	硬式

2.1.6

下表列出此元件 2.1.6 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.0.0	軟式
DLR 圖像分類模型商店	約 2.1.0	硬式
DLR	約 1.6.0	硬式

2.1.4 - 2.1.5

下表列出此元件 2.1.4 至 2.1.5 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
DLR 圖像分類模型商店	約 2.1.0	硬式
DLR	約 1.6.0	硬式

2.1.3

下表列出此元件 2.1.3 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
DLR 圖像分類模型商店	約 2.1.0	硬式
DLR	約 1.6.0	硬式

2.1.2

下表列出此元件 2.1.2 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
DLR 圖像分類模型商店	約 2.1.0	硬式
DLR	約 1.6.0	硬式

2.1.1

下表列出此元件 2.1.1 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
DLR 圖像分類模型商店	約 2.1.0	硬式
DLR	約 1.6.0	硬式

2.0.x

下表列出此元件 2.0.x 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	〜 2.0.0	軟式
DLR 圖像分類模型商店	〜 2.0.0	硬式
DLR	約 1.3.0	軟式

組態

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

2.1.x

`accessControl`

(選擇性) 包含授權原則的物件，可讓元件將訊息發佈至預設通知主題。

預設：

```
{
   "aws.greengrass.ipc.mqttproxy": {
      "aws.greengrass.DLRImageClassification:mqttproxy:1": {
         "policyDescription": "Allows access to publish via topic ml/dlr/image-
classification.",
         "operations": [
            "aws.greengrass#PublishToIoTCore"
         ],
         "resources": [
            "ml/dlr/image-classification"
         ]
      }
   }
}
```

`PublishResultsOnTopic`

(選擇性) 您要發佈推論結果的主題。如果修改此值，則還必須修改`accessControl`參數`resources`中的值以符合您的自訂主題名稱。

預設：`ml/dlr/image-classification`

`Accelerator`

您要使用的加速器。支援的值為 cpu 和 gpu。

相依模型元件中的範例模型僅支援 CPU 加速。若要將 GPU 加速與不同的自訂模型搭配使用，請建立自訂模型元件以覆寫公用模型元件。

預設：`cpu`

`ImageDirectory`

(選擇性) 推論元件讀取影像之設備上的資料夾路徑。您可以將此值修改為裝置上具有讀取/寫入存取權限的任何位置。

預設：*/greengrass/v2*/packages/artifacts-unarchived/*component-name*/image_classification/sample_images/

> **Note**
>
> 如果UseCamera將的值設定為true，則會忽略此組態參數。

ImageName

(選擇性) 推論元件用作進行預測輸入的影像名稱。元件會在中指定的資料夾中尋找影像ImageDirectory。依預設，元件會使用預設映像目錄中的範例影像。 AWS IoT Greengrass支援下列影像格式：jpegjpg、png、和npy。

預設：cat.jpeg

> **Note**
>
> 如果UseCamera將的值設定為true，則會忽略此組態參數。

InferenceInterval

(選擇性) 推論程式碼進行的每個預測之間的時間 (以秒為單位)。範例推論程式碼會無限期執行，並在指定的時間間隔內重複其預測。例如，如果您要使用相機拍攝的影像進行即時預測，可以將其變更為較短的間隔。

預設：3600

ModelResourceKey

(選擇性) 在相依公用模型元件中使用的模型。只有在使用自訂元件覆寫公用模型元件時，才能修改此參數。

預設：

```
{
    "armv7l": "DLR-resnet50-armv7l-cpu-ImageClassification",
    "aarch64": "DLR-resnet50-aarch64-cpu-ImageClassification",
    "x86_64": "DLR-resnet50-x86_64-cpu-ImageClassification",
    "windows": "DLR-resnet50-win-cpu-ImageClassification"
```

```
}
```

`UseCamera`

(選擇性) 字串值，定義是否使用來自連接至 Greengrass 核心裝置之攝影機的影像。支援的值為 `true` 和 `false`。

將此值設定為時`true`，範例推論程式碼會存取裝置上的攝影機，並在擷取的影像本機上執行推論。`ImageName`和`ImageDirectory`參數的值會被忽略。請確定執行此元件的使用者具有相機儲存擷取影像之位置的讀取/寫入存取權。

預設：`false`

> **Note**
>
> 當您檢視此元件的配方時，`UseCamera`組態參數不會顯示在預設組態中。不過，您可以在部署元件時，在組態合併更新中修改此參數的值。
> 當您設定`UseCamera`為時`true`，您還必須建立符號連結，以使推論元件能夠從執行階段元件建立的虛擬環境存取您的攝影機。如需將相機與範例推論元件搭配使用的詳細資訊，請參閱〈更新零組件組態〉。

2.0.x

`MLRootPath`

(選擇性) 推論元件讀取影像和寫入推論結果的 Linux 核心裝置上的資料夾路徑。您可以將此值修改為裝置上執行此元件的使用者具有讀取/寫入存取權限的任何位置。

預設：*/greengrass/v2*/work/variant.DLR/greengrass_ml

預設：*/greengrass/v2*/work/variant.TensorFlowLite/greengrass_ml

`Accelerator`

您要使用的加速器。支援的值為 `cpu` 和 `gpu`。

相依模型元件中的範例模型僅支援 CPU 加速。若要將 GPU 加速與不同的自訂模型搭配使用，請建立自訂模型元件以覆寫公用模型元件。

預設：`cpu`

`ImageName`

(選擇性) 推論元件用作進行預測輸入的影像名稱。元件會在中指定的資料夾中尋找影像`ImageDirectory`。預設位置為*`MLRootPath`*`/images`。 AWS IoT Greengrass支援下列影像格式：`jpeg`jpg、png、和npy。

預設：`cat.jpeg`

`InferenceInterval`

(選擇性) 推論程式碼進行的每個預測之間的時間 (以秒為單位)。範例推論程式碼會無限期執行，並在指定的時間間隔內重複其預測。例如，如果您要使用相機拍攝的影像進行即時預測，可以將其變更為較短的間隔。

預設：`3600`

`ModelResourceKey`

(選擇性) 在相依公用模型元件中使用的模型。只有在使用自訂元件覆寫公用模型元件時，才能修改此參數。

預設：

```
armv7l: "DLR-resnet50-armv7l-cpu-ImageClassification"
x86_64: "DLR-resnet50-x86_64-cpu-ImageClassification"
```

本機記錄檔

此元件使用下列記錄檔。

Linux

```
/greengrass/v2/logs/aws.greengrass.DLRImageClassification.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.DLRImageClassification.log
```

若要檢視此元件的記錄

- 在核心裝置上執行下列命令，即時檢視此元件的記錄檔。以AWS IoT Greengrass根資料夾的路徑取代*/greengrass/v2*或 *C:\greengrass\v2*。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.DLRImageClassification.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.greengrass.DLRImageClassification.log -
  Tail 10 -Wait
  ```

變更記錄

下表說明元件每個版本中的變更。

版本	變更
2.1.13	版本更新了 Greengrass 核 2.12.0 版本釋放。
2.1.12	版本更新了 Greengrass 核 2.11.0 版本釋放。
2.1.11	版本更新了 Greengrass 核 2.10.0 版本釋放。
2.1.10	版本更新 Greengrass 2.9.0 版本釋放。
2.1.9	版本更新 Greengrass 2.8.0 版本的版本。
2.1.8	版本更新了 Greengrass 核 2.7.0 版本釋放。
2.1.7	版本更新 Greengrass 2.6.0 版本的版本。
2.1.6	版本更新了 Greengrass 核 2.5.0 版本。
2.1.5	全部發行的元件AWS 區域。
2.1.4	版本更新 Greengrass 2.4.0 版本的版本。

版本	變更
	此版本不適用於歐洲 (倫敦) (`eu-west-2`)。
2.1.3	版本更新了 Greengrass 核 2.3.0 版本。
2.1.2	版本更新了 Greengrass 核 2.2.0 版本。
2.1.1	新功能 • 使用深度學習執行階段 v1.6.0。 • 在 Arch64 (AArch64) 平台上新增對範例影像分類的支援。這擴展了對運行 NVIDIA 傑特森的 Greengrass 核心設備的機器學習支持，例如傑特森納米。 • 啟用相機整合以進行範例推論。使用新的`UseCamera` 組態參數可啟用範例推論程式碼，以存取 Greengrass 核心裝置上的攝影機，並在擷取的映像上在本機執行推論。 • 新增將推論結果發佈至. AWS 雲端 使用新的`PublishResultsOnTopic` 組態參數來指定您要發佈結果的主題。 • 新增可讓您為要執行推論之影像指定自訂目錄的`ImageDirectory` 組態參數。 錯誤修復和改進 • 將推論結果寫入元件記錄檔，而不是個別的推論檔案。 • 使用AWS IoT Greengrass核心軟體記錄模組來記錄元件輸出。 • 使用讀AWS IoT Device SDK取零組件模型組態並套用模型組態變更。
2.0.4	初始版本。

DLR 物體偵測

DLR 物件偵測元件 (`aws.greengrass.DLRObjectDetection`) 包含範例推論程式碼，可使用深度學習執行階段執行物件偵測推論，以及範例預先訓練的模型。此元件使用變體DLR 物件偵測模型商店和DLR 執行階段元件作為從屬關係來下載 DLR 和範例模型。

若要將此推論元件與自訂訓練有素的 DLR 模型搭配使用，請建立相依模型存放區元件的自訂版本。若要使用您自己的自訂推論程式碼，您可以使用此元件的方式做為範本來建立自訂推論元件。

主題

- 版本
- Type
- 作業系統
- 要求
- 相依性
- 組態
- 本機記錄檔
- 變更記錄

版本

此元件具有下列版本：

- 2.1.x
- 2.0.x

Type

此元件是一般元件 (`aws.greengrass.generic`)。Greengrass 核會執行元件的生命週期指令碼。

如需詳細資訊，請參閱 元件類型。

作業系統

此元件可安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件具有下列需求：

- 在運行 Amazon Linux 2 或 Ubuntu 18.04 的 Greengrass 核心設備上，GNU C 庫（glibc）2.27 版或更高版本安裝在設備上。
- 在 ARMV7L 設備上，如樹莓派，對於設備上安裝了 OpenCV-Python 的依賴關係。執行下列命令以安裝相依性。

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
 libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- 樹莓派運行樹莓派 OS 靶心設備必須滿足以下要求:
  - NumPy 1.22.4 或更新版本已安裝在裝置上。樹莓派 OS 靶心包括早期版本的 NumPy，因此您可以在設備上運行以下命令進 NumPy 行升級。

    ```
    pip3 install --upgrade numpy
    ```

  - 裝置上啟用的舊式攝影機堆疊。樹莓派 OS Bullseye 包括一個新的相機堆疊，預設情況下啟用且不相容，因此您必須啟用傳統的相機堆疊。

    啟用舊式相機堆疊

    1. 運行以下命令以打開樹莓派配置工具。

       ```
       sudo raspi-config
       ```

    2. 選取介面選項。
    3. 選取「舊式相機」以啟用舊式相機堆疊。
    4. 重新啟動 Raspberry Pi。

相依性

部署元件時，AWS IoT Greengrass也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的需求，才能成功部署元件。本節列出此元件之已發行版本的相依性，以及定義每個相依性之元件版本的語意版本條件約束。您也可以在AWS IoT Greengrass主控台中檢視元件每個版本的相依性。在元件詳細資料頁面上，尋找 [相依性] 清單。

2.1.13

下表列出此元件 2.1.13 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
DLR 物件偵測模型商店	約 2.1.0	硬式

相依性	兼容版本	相依性類型
DLR	約 1.6.0	硬式

2.1.12

下表列出此元件 2.1.12 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
DLR 物件偵測模型商店	約 2.1.0	硬式
DLR	約 1.6.0	硬式

2.1.11

下表列出此元件 2.1.11 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
DLR 物件偵測模型商店	約 2.1.0	硬式
DLR	約 1.6.0	硬式

2.1.10

下表列出此元件 2.1.10 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
DLR 物件偵測模型商店	約 2.1.0	硬式

相依性	兼容版本	相依性類型
DLR	約 1.6.0	硬式

2.1.9

下表列出此元件 2.1.9 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
DLR 物件偵測模型商店	約 2.1.0	硬式
DLR	約 1.6.0	硬式

2.1.8

下表列出此元件 2.1.8 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
DLR 物件偵測模型商店	約 2.1.0	硬式
DLR	約 1.6.0	硬式

2.1.7

下表列出此元件 2.1.7 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
DLR 物件偵測模型商店	約 2.1.0	硬式

相依性	兼容版本	相依性類型
DLR	約 1.6.0	硬式

2.1.6

下表列出此元件 2.1.6 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.0.0	軟式
DLR 物件偵測模型商店	約 2.1.0	硬式
DLR	約 1.6.0	硬式

2.1.4 - 2.1.5

下表列出此元件 2.1.4 至 2.1.5 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
DLR 物件偵測模型商店	約 2.1.0	硬式
DLR	約 1.6.0	硬式

2.1.3

下表列出此元件 2.1.3 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
DLR 物件偵測模型商店	約 2.1.0	硬式

相依性	兼容版本	相依性類型
DLR	約 1.6.0	硬式

2.1.2

下表列出此元件 2.1.2 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
DLR 物件偵測模型商店	約 2.1.0	硬式
DLR	約 1.6.0	硬式

2.1.1

下表列出此元件 2.1.1 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
DLR 物件偵測模型商店	約 2.1.0	硬式
DLR	約 1.6.0	硬式

2.0.x

下表列出此元件 2.0.x 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	~ 2.0.0	軟式
DLR 物件偵測模型商店	~ 2.0.0	硬式

相依性	兼容版本	相依性類型
DLR	約 1.3.0	軟式

組態

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

2.1.x

`accessControl`

(選擇性) 包含授權原則的物件，可讓元件將訊息發佈至預設通知主題。

預設：

```
{
   "aws.greengrass.ipc.mqttproxy": {
      "aws.greengrass.DLRObjectDetection:mqttproxy:1": {
         "policyDescription": "Allows access to publish via topic ml/dlr/object-
detection.",
         "operations": [
            "aws.greengrass#PublishToIoTCore"
         ],
         "resources": [
            "ml/dlr/object-detection"
         ]
      }
   }
}
```

`PublishResultsOnTopic`

(選擇性) 您要發佈推論結果的主題。如果修改此值，則還必須修改`accessControl`參數`resources`中的值以符合您的自訂主題名稱。

預設：`ml/dlr/object-detection`

`Accelerator`

您要使用的加速器。支援的值為 cpu 和 gpu。

相依模型元件中的範例模型僅支援 CPU 加速。若要將 GPU 加速與不同的自訂模型搭配使用，請建立自訂模型元件以覆寫公用模型元件。

預設：cpu

`ImageDirectory`

(選擇性) 推論元件讀取影像之設備上的資料夾路徑。您可以將此值修改為裝置上具有讀取/寫入存取權限的任何位置。

預設：*/greengrass/v2*/packages/artifacts-unarchived/*component-name*/object_detection/sample_images/

> **Note**
>
> 如果`UseCamera`將的值設定為`true`，則會忽略此組態參數。

`ImageName`

(選擇性) 推論元件用作進行預測輸入的影像名稱。元件會在中指定的資料夾中尋找影像`ImageDirectory`。依預設，元件會使用預設映像目錄中的範例影像。 AWS IoT Greengrass支援下列影像格式：`jpeg`jpg、png、和npy。

預設：objects.jpg

> **Note**
>
> 如果`UseCamera`將的值設定為`true`，則會忽略此組態參數。

`InferenceInterval`

(選擇性) 推論程式碼進行的每個預測之間的時間 (以秒為單位)。範例推論程式碼會無限期執行，並在指定的時間間隔內重複其預測。例如，如果您要使用相機拍攝的影像進行即時預測，可以將其變更為較短的間隔。

預設：3600

`ModelResourceKey`

(選擇性) 在相依公用模型元件中使用的模型。只有在使用自訂元件覆寫公用模型元件時，才能修改此參數。

預設：

```
{
    "armv7l": "DLR-yolo3-armv7l-cpu-ObjectDetection",
    "aarch64": "DLR-yolo3-aarch64-gpu-ObjectDetection",
    "x86_64": "DLR-yolo3-x86_64-cpu-ObjectDetection",
    "windows": "DLR-resnet50-win-cpu-ObjectDetection"
}
```

`UseCamera`

(選擇性) 字串值，定義是否使用來自連接至 Greengrass 核心裝置之攝影機的影像。支援的值為 `true` 和 `false`。

將此值設定為時`true`，範例推論程式碼會存取裝置上的攝影機，並在擷取的影像本機上執行推論。`ImageName`和`ImageDirectory`參數的值會被忽略。請確定執行此元件的使用者具有相機儲存擷取影像之位置的讀取/寫入存取權。

預設：`false`

> **Note**
>
> 當您檢視此元件的配方時，`UseCamera`組態參數不會顯示在預設組態中。不過，您可以在部署元件時，在組態合併更新中修改此參數的值。
> 當您設定`UseCamera`為時`true`，您還必須建立符號連結，以使推論元件能夠從執行階段元件建立的虛擬環境存取您的攝影機。如需將相機與範例推論元件搭配使用的詳細資訊，請參閱更新零組件組態。

2.0.x

`MLRootPath`

(選擇性) 推論元件讀取影像和寫入推論結果的 Linux 核心裝置上的資料夾路徑。您可以將此值修改為裝置上執行此元件的使用者具有讀取/寫入存取權限的任何位置。

預設：*/greengrass/v2*/work/variant.DLR/greengrass_ml

預設：*/greengrass/v2*/work/variant.TensorFlowLite/greengrass_ml

`Accelerator`

請勿修改。目前，加速器唯一支援的值是`cpu`，因為相依模型元件中的模型僅針對 CPU 加速器編譯。

`ImageName`

(選擇性) 推論元件用作進行預測輸入的影像名稱。元件會在中指定的資料夾中尋找影像`ImageDirectory`。預設位置為*`MLRootPath`*`/images`。 AWS IoT Greengrass支援下列影像格式：`jpeg`jpg、`png`、和`npy`。

預設：`objects.jpg`

`InferenceInterval`

(選擇性) 推論程式碼進行的每個預測之間的時間 (以秒為單位)。範例推論程式碼會無限期執行，並在指定的時間間隔內重複其預測。例如，如果您要使用相機拍攝的影像進行即時預測，可以將其變更為較短的間隔。

預設：`3600`

`ModelResourceKey`

(選擇性) 在相依公用模型元件中使用的模型。只有在使用自訂元件覆寫公用模型元件時，才能修改此參數。

預設：

```
{
    armv7l: "DLR-yolo3-armv7l-cpu-ObjectDetection",
    x86_64: "DLR-yolo3-x86_64-cpu-ObjectDetection"
}
```

本機記錄檔

此元件使用下列記錄檔。

Linux

```
/greengrass/v2/logs/aws.greengrass.DLRObjectDetection.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.DLRObjectDetection.log
```

若要檢視此元件的記錄

- 在核心裝置上執行下列命令，即時檢視此元件的記錄檔。以AWS IoT Greengrass根資料夾的路徑取代*/greengrass/v2*或 *C:\greengrass\v2*。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.DLRObjectDetection.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.greengrass.DLRObjectDetection.log -Tail 10
   -Wait
  ```

變更記錄

下表說明元件每個版本的變更。

版本	變更
2.1.13	版本更新 Greengrass 2.12.0 版本釋放。
2.1.12	版本更新 Greengrass 2.11.0 版本釋放。
2.1.11	版本更新了 Greengrass 2.10.0 版本。
2.1.10	版本更新 Greengrass 2.9.0 版本釋放。
2.1.9	版本更新 Greengrass 2.8.0 版本的版本。
2.1.8	版本更新了 Greengrass 核 2.7.0 版本釋放。
2.1.7	版本更新 Greengrass 2.6.0 版本發布。
2.1.6	版本更新了 Greengrass 核 2.5.0 版本。

版本	變更
2.1.5	全部已發行的元件AWS 區域。
2.1.4	版本更新 Greengrass 2.4.0 版本的版本。 此版本不適用於歐洲 (倫敦) (`eu-west-2`)。
2.1.3	版本更新了 Greengrass 核 2.3.0 版本。
2.1.2	錯誤修復和改進 • 修正在範例 DLR 物件偵測推論結果中造成不正確邊界方塊的影像縮放問題。
2.1.1	新功能 • 使用深度學習執行階段 v1.6.0。 • 添加對在 Armv8（AArch64）平台上進行樣本對象檢測的支持。這擴展了對運行 NVIDIA 傑特森的 Greengrass 核心設備的機器學習支持，例如傑特森納米。 • 啟用相機整合以進行範例推論。使用新的`UseCamera` 組態參數可啟用範例推論程式碼，以存取 Greengrass 核心裝置上的攝影機，並在擷取的映像上在本機執行推論。 • 新增將推論結果發佈至. AWS 雲端 使用新的`PublishResultsOnTopic` 組態參數來指定您要發佈結果的主題。 • 新增可讓您為要執行推論之影像指定自訂目錄的`ImageDirectory` 組態參數。 錯誤修復和改進 • 將推論結果寫入元件記錄檔，而不是個別的推論檔案。 • 使用 AWS IoT Greengrass Core 軟體記錄模組來記錄元件輸出。 • 使用讀AWS IoT Device SDK取零組件模型組態並套用模型組態變更。
2.0.4	初始版本。

DLR 圖像分類模型商店

DLR 影像分類模型存放區是機器學習模型元件，其中包含預先訓練的 ResNet -50 模型做為 Greengrass 工件。此元件中使用的預先訓練過的模型是從 GluonCV 模型 Zoo 擷取，並使用 SageMaker Neo 深度學習執行階段進行編譯。

DLR 圖像分類推論組件使用此組件作為模型源的依賴關係。若要使用自訂訓練的 DLR 模型，請建立此模型元件的自訂版本，並將您的自訂模型納入為元件人工因素。您可以使用此元件的配方做為範本來建立自訂模型元件。

> **Note**
>
> DLR 影像分類模型存放區元件的名稱會根據其版本而有所不同。版本 2.1.x 及更新版本的元件名稱為。`variant.DLR.ImageClassification.ModelStore`版本 2.0.x 的元件名稱為。`variant.ImageClassification.ModelStore`

主題

版本

此元件具有下列版本：

- 2.x () `variant.DLR.ImageClassification.ModelStore`
- 2.x () `variant.ImageClassification.ModelStore`

Type

此元件是一般元件 (`aws.greengrass.generic`)。Greengrass 核會執行元件的生命週期指令碼。

如需詳細資訊，請參閱 元件類型。

作業系統

此元件可安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件具有下列需求：

- 在運行 Amazon Linux 2 或 Ubuntu 18.04 的 Greengrass 核心設備上，GNU C 庫（glibc）2.27 版或更高版本安裝在設備上。
- 在 ARMV7L 設備上，如樹莓派，對於設備上安裝了 OpenCV-Python 的依賴關係。執行下列命令以安裝相依性。

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
 libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- 樹莓派運行樹莓派 OS 靶心設備必須滿足以下要求:
  - NumPy 1.22.4 或更新版本已安裝在裝置上。樹莓派 OS 靶心包括早期版本的 NumPy，因此您可以在設備上運行以下命令進 NumPy 行升級。

    ```
    pip3 install --upgrade numpy
    ```

  - 裝置上啟用的舊式攝影機堆疊。樹莓派 OS Bullseye 包括一個新的相機堆疊，預設情況下啟用且不相容，因此您必須啟用傳統的相機堆疊。

    啟用舊式相機堆疊

    1. 運行以下命令以打開樹莓派配置工具。

       ```
       sudo raspi-config
       ```

    2. 選取介面選項。
    3. 選取「舊式相機」以啟用舊式相機堆疊。
    4. 重新啟動 Raspberry Pi。

相依性

部署元件時，AWS IoT Greengrass也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的需求，才能成功部署元件。本節列出此元件之已發行版本的相依性，以及定義每個相依性之元件版本的語意版本條件約束。您也可以在AWS IoT Greengrass主控台中檢視元件每個版本的相依性。在元件詳細資料頁面上，尋找 [相依性] 清單。

2.1.12

下表列出此元件 2.1.12 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.11

下表列出此元件 2.1.11 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.10

下表列出此元件 2.1.10 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.9

下表列出此元件 2.1.9 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.8

下表列出此元件 2.1.8 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.7

下表列出此元件 2.1.7 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.6

下表列出此元件 2.1.6 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.5

下表列出此元件 2.1.5 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.0.0	軟式

2.1.4

下表列出此元件 2.1.4 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.3

下表列出此元件 2.1.3 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.2

下表列出此元件 2.1.2 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.1

下表列出此元件 2.1.1 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.0.x

下表列出此元件 2.0.x 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	~ 2.0.0	軟式

組態

此元件沒有任何組態參數。

本機記錄檔

此元件不會輸出記錄檔。

變更記錄

下表說明元件每個版本中的變更。

版本	變更
2.1.12	版本更新了 Greengrass 核 2.12.0 版本釋放。
2.1.11	版本更新了 Greengrass 核 2.11.0 版本釋放。
2.1.10	版本更新了 Greengrass 核 2.10.0 版本釋放。
2.1.9	版本更新 Greengrass 2.9.0 版本釋放。
2.1.8	版本更新 Greengrass 2.8.0 版本的版本。
2.1.7	版本更新了 Greengrass 核 2.7.0 版本釋放。
2.1.6	版本更新 Greengrass 2.6.0 版本的版本。
2.1.5	新功能 • 為 Windows 核心裝置新增範例影像分類模型。 • 版本更新了 Greengrass 核 2.5.0 版本。
2.1.4	版本更新 Greengrass 2.4.0 版本的版本。
2.1.3	版本更新了 Greengrass 核 2.3.0 版本。

版本	變更
2.1.2	版本更新了 Greengrass 核 2.2.0 版本。
2.1.1	新功能 • 為 Arch64 平台新增一個範例 ResNet -50 影像分類模型。這擴展了對運行 NVIDIA 傑特森的 Greengrass 核心設備的機器學習支持，例如傑特森納米。
2.0.4	初始版本。

DLR 物件偵測模型商店

DLR 物件偵測模型存放區是一個機器學習模型元件，其中包含預先訓練的 YoLov3 模型做為 Greengrass 工件。此元件中使用的範例模型是從 GluonCV 模型 Zoo 擷取，並使用 SageMaker Neo 深度學習執行階段進行編譯。

該 DLR 對象檢測推斷組件使用此組件作為模型源的依賴關係。若要使用自訂訓練的 DLR 模型，請建立此模型元件的自訂版本，並將您的自訂模型納入為元件人工因素。您可以使用此元件的配方做為範本來建立自訂模型元件。

> **Note**
>
> DLR 物件偵測模型存放區元件的名稱會依其版本而有所不同。版本 2.1.x 及更新版本的元件名稱為。`variant.DLR.ObjectDetection.ModelStore`版本 2.0.x 的元件名稱為。`variant.ObjectDetection.ModelStore`

主題

- 版本
- Type
- 作業系統
- 要求
- 相依性
- 組態

- 本機記錄檔
- 變更記錄

版本

此元件具有下列版本：

- 2.1.x
- 2.0.x

Type

此元件是一般元件 (`aws.greengrass.generic`)。Greengrass 核會執行元件的生命週期指令碼。

如需詳細資訊，請參閱 元件類型。

作業系統

此元件可安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件具有下列需求：

- 在運行 Amazon Linux 2 或 Ubuntu 18.04 的 Greengrass 核心設備上，GNU C 庫（glibc）2.27 版或更高版本安裝在設備上。
- 在 ARMV7L 設備上，如樹莓派，對於設備上安裝了 OpenCV-Python 的依賴關係。執行下列命令以安裝相依性。

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
 libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- 樹莓派運行樹莓派 OS 靶心設備必須滿足以下要求:
  - NumPy 1.22.4 或更新版本已安裝在裝置上。樹莓派 OS 靶心包括早期版本的 NumPy，因此您可以在設備上運行以下命令進 NumPy 行升級。

```
pip3 install --upgrade numpy
```

- 裝置上啟用的舊式攝影機堆疊。樹莓派 OS Bullseye 包括一個新的相機堆疊，預設情況下啟用且不相容，因此您必須啟用傳統的相機堆疊。

  啟用舊式相機堆疊的步驟

  1. 運行以下命令以打開樹莓派配置工具。

     ```
     sudo raspi-config
     ```

  2. 選取介面選項。
  3. 選取「舊式相機」以啟用舊式相機堆疊。
  4. 重新啟動 Raspberry Pi。

相依性

部署元件時，AWS IoT Greengrass也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的需求，才能成功部署元件。本節列出此元件之已發行版本的相依性，以及定義每個相依性之元件版本的語意版本條件約束。您也可以在AWS IoT Greengrass主控台中檢視元件每個版本的相依性。在元件詳細資料頁面上，尋找 [相依性] 清單。

2.1.13

下表列出此元件 2.1.13 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.12

下表列出此元件 2.1.12 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.11

下表列出此元件 2.1.11 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0 <2.11.0	軟式

2.1.10

下表列出此元件 2.1.10 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.9

下表列出此元件 2.1.9 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.8

下表列出此元件 2.1.8 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.7

下表列出此元件 2.1.7 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.5 and 2.1.6

下表列出此元件 2.1.5 和 2.1.6 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.4

下表列出此元件 2.1.4 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.3

下表列出此元件 2.1.3 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.2

下表列出此元件 2.1.2 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.1

下表列出此元件 2.1.1 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.0.x

下表列出此元件 2.0.x 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	~ 2.0.0	軟式

組態

此元件沒有任何組態參數。

本機記錄檔

此元件不會輸出記錄檔。

變更記錄

下表說明元件每個版本中的變更。

版本	變更
2.1.13	版本更新了 Greengrass 核 2.12.0 版本釋放。
2.1.12	版本更新了 Greengrass 核 2.11.0 版本釋放。
2.1.11	版本更新了 Greengrass 2.10.0 版本。
2.1.10	版本更新 Greengrass 2.9.0 版本釋放。
2.1.9	版本更新 Greengrass 2.8.0 版本的版本。

版本	變更
2.1.8	版本更新了 Greengrass 核 2.7.0 版本釋放。
2.1.7	版本更新 Greengrass 2.6.0 版本的版本。
2.1.6	新增 CPU 模型以修正 Arch64 裝置上的問題。
2.1.5	新功能 • 為 Windows 核心裝置新增範例物件偵測模型。 錯誤修復和改進 • 版本更新了 Greengrass 核 2.5.0 版本。
2.1.4	版本更新 Greengrass 2.4.0 版本的版本。
2.1.3	版本更新了 Greengrass 核 2.3.0 版本。
2.1.2	版本更新了 Greengrass 核 2.2.0 版本。
2.1.1	新功能 • 新增適用於 AArch64 平台的範例 Yolov3 物件偵測模型。這擴展了對運行 NVIDIA 傑特森的 Greengrass 核心設備的機器學習支持，例如傑特森納米。
2.0.4	初始版本。

DLR 執行階段

DLR 執行階段元件 (`variant.DLR`) 包含在裝置上虛擬環境中安裝深度學習執行階段 (DLR) 及其相依性的指令碼。DLR 影像分類和DLR 物體偵測元件會使用此元件做為安裝 DLR 的相依性。組件版本 1.6.x 的安裝 DLR v1.6.0 和組件版本 1.3.x 的安裝 DLR V1.3.0。

若要使用不同的執行階段，您可以使用此元件的方案做為範本來建立自訂機器學習元件。

主題

- 版本
- Type
- 作業系統

- 要求
- 相依性
- 組態
- 用量
- 本機記錄檔
- 變更記錄

版本

此元件具有下列版本：

- 1.6.x
- 1.3.x

Type

此元件是一般元件 (`aws.greengrass.generic`)。Greengrass 核會執行元件的生命週期指令碼。

如需詳細資訊，請參閱 元件類型。

作業系統

此元件可安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件具有下列需求：

- 在運行 Amazon Linux 2 或 Ubuntu 18.04 的 Greengrass 核心設備上，GNU C 庫（glibc）2.27 版或更高版本安裝在設備上。
- 在 ARMV7L 設備上，如樹莓派，對於設備上安裝了 OpenCV-Python 的依賴關係。執行下列命令以安裝相依性。

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
 libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- 樹莓派運行樹莓派 OS 靶心設備必須滿足以下要求:
  - NumPy 1.22.4 或更新版本已安裝在裝置上。樹莓派 OS 靶心包括早期版本的 NumPy，因此您可以運行以下命令在設備 NumPy 上升級。

    ```
    pip3 install --upgrade numpy
    ```

  - 裝置上啟用的舊式攝影機堆疊。樹莓派 OS Bullseye 包括一個新的相機堆疊，預設情況下啟用且不相容，因此您必須啟用傳統的相機堆疊。

    啟用舊式相機堆疊的步驟

    1. 運行以下命令以打開樹莓派配置工具。

       ```
       sudo raspi-config
       ```

    2. 選取介面選項。
    3. 選取「舊式相機」以啟用舊式相機堆疊。
    4. 重新啟動 Raspberry Pi。

端點和連接埠

根據預設，此元件會根據核心裝置使用的平台，使用`apt``yum`、`brew`、和`pip`指令，使用安裝程式指令碼來安裝套件。此元件必須能夠對各種套件索引和儲存庫執行輸出要求，才能執行安裝程式指令碼。若要允許此元件透過 Proxy 或防火牆的輸出流量，您必須識別套件索引的端點，以及核心裝置連線以進行安裝的存放庫。

當您識別此元件的安裝指令碼所需的端點時，請考慮下列事項：

- 端點取決於核心設備的平台。例如，執行 Ubuntu 的核心裝置會使用`apt`而不是`yum`或`brew`。此外，使用相同套件索引的裝置可能會有不同的來源清單，因此可能會從不同的軟體庫擷取套件。
- 使用相同套件索引的多個裝置之間的端點可能會有所不同，因為每個裝置都有自己的來源清單來定義擷取套件的位置。
- 端點可能會隨時間變更。每個套裝程式索引都會提供您下載套裝程式的儲存區域 URL，套裝程式的擁有者可以變更套裝程式索引所提供的 URL。

如需有關此元件安裝之相依性以及如何停用安裝程式指令碼的詳細資訊，請參閱UseInstaller組態參數。

如需有關基本作業所需的端點和連接埠的詳細資訊，請參閱允許裝置流量透過 Proxy 或防火牆。

相依性

部署元件時，AWS IoT Greengrass也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的需求，才能成功部署元件。本節列出此元件之已發行版本的相依性，以及定義每個相依性之元件版本的語意版本條件約束。您也可以在AWS IoT Greengrass主控台中檢視元件每個版本的相依性。在元件詳細資料頁面上，尋找 [相依性] 清單。

1.6.11 and 1.6.12

下表列出此元件 1.6.11 和 1.6.12 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

1.6.10

下表列出此元件 1.6.10 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

1.6.9

下表列出此元件 1.6.9 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

1.6.8

下表列出此元件 1.6.8 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

1.6.6 and 1.6.7

下表列出此元件 1.6.6 和 1.6.7 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

1.6.4 and 1.6.5

下表列出此元件 1.6.4 和 1.6.5 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

1.6.3

下表列出此元件 1.6.3 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

1.6.2

下表列出此元件 1.6.2 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

1.6.1

下表列出此元件 1.6.1 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

1.3.x

下表列出此元件 1.3.x 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	~ 2.0.0	軟式

如需有關元件相依性的詳細資訊，請參閱元件方案參考。

組態

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

`MLRootPath`

(選擇性) 推論元件讀取影像和寫入推論結果的 Linux 核心裝置上的資料夾路徑。您可以將此值修改為裝置上執行此元件的使用者具有讀取/寫入存取權限的任何位置。

預設：*/greengrass/v2*/work/variant.DLR/greengrass_ml

`WindowsMLRootPath`

此功能可在此元件 v1.6.6 及更新版本中使用。

(選擇性) 推論元件讀取影像和寫入推論結果的 Windows 核心裝置上的資料夾路徑。您可以將此值修改為裝置上執行此元件的使用者具有讀取/寫入存取權限的任何位置。

預設：*C:\greengrass\v2*\\work\\variant.DLR\\greengrass_ml

`UseInstaller`

(選擇性) 字串值，定義是否使用此元件中的安裝程式指令碼來安裝 DLR 及其相依性。支援的值為 `true` 和 `false`。

`false`如果您要使用自訂指令碼進行 DLR 安裝，或者想要在預先建置的 Linux 映像中包含執行階段相依性，請將此值設定為。若要將此元件與AWS提供的 DLR 推論元件搭配使用，請安裝下列程式庫 (包括任何相依性)，並讓系統使用者使用，例如`ggc_user`執行 ML 元件的程式庫。

- Python 3.7 或更高版本，包括`pip`您的 Python 版本。
- 深度學習執行階段 v1.6.0
- NumPy.
- 開放 CV-Python。
- AWS IoT Device SDK對於 Python 的 V2。
- AWS通用運行時（CRT）Python。
- 皮卡馬拉（僅適用於樹莓派設備）。
- `awscam`模塊（用於AWS DeepLens設備）。
- LibGL (適用於 Linux 設備)

預設：`true`

用量

在將`UseInstaller`組態參數設定為`true`的情況下使用此元件，可在裝置上安裝 DLR 及其相依性。元件會在裝置上設定虛擬環境，其中包含 DLR 所需的 OpenCV 和 NumPy 程式庫。

> **Note**
>
> 此元件中的安裝程式指令碼也會安裝最新版本的其他系統程式庫，以便在您的裝置上設定虛擬環境，以及使用已安裝的機器學習架構。這可能會升級您裝置上現有的系統程式庫。請檢閱下表，以取得此元件為每個支援的作業系統安裝的程式庫清單。如果您要自訂此安裝程序，請將`UseInstaller`組態參數設定為`false`，然後開發您自己的安裝程式指令碼。

平台	裝置系統上安裝的程式庫	安裝在虛擬環境中的程式庫
Armv7l	`build-essential` ，`cmake`，`ca-certificates` ，`git`	`setuptools` ，`wheel`
Amazon Linux 2	`mesa-libGL`	無
Ubuntu	`wget`	無

當您部署推論元件時，此執行階段元件會先驗證您的裝置是否已安裝 DLR 及其相依性，如果沒有，則會為您安裝這些元件。

本機記錄檔

此元件使用下列記錄檔。

Linux

```
/greengrass/v2/logs/variant.DLR.log
```

Windows

```
C:\greengrass\v2\logs\variant.DLR.log
```

若要檢視此元件的記錄

- 在核心裝置上執行下列命令，即時檢視此元件的記錄檔。以AWS IoT Greengrass根資料夾的路徑取代*/greengrass/v2*或 *C:\greengrass\v2*。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/variant.DLR.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\variant.DLR.log -Tail 10 -Wait
  ```

變更記錄

下表說明元件每個版本的變更。

版本	變更
1.6.12	錯誤修復和改進 • 修復了 Windows 操作系統用戶的安裝腳本。
1.6.11	版本更新 Greengrass 2.9.0 版本釋放。

版本	變更
1.6.10	版本更新 Greengrass 2.8.0 版本的版本。
1.6.9	版本更新了 Greengrass 核 2.7.0 版本釋放。
1.6.8	版本更新 Greengrass 2.6.0 版本發布。
1.6.7	錯誤修復和改進 • 更新安`UseInstaller` 裝指令碼以安裝 libGL，在某些 Linux 平台上預設不可用。 • 更新`UseInstaller` 安裝腳本以始終在此組件的虛擬環境中使用 Python 3.9。此變更有助於確保與其他程式庫的相容性。
1.6.6	新功能 • 添加對運行 Windows 的核心設備的支持。 • 新增可用來在 Windows 核心裝置上設定推論結果資料夾的新`WindowsMLRootPath` 組態參數。
1.6.5	新功能 • 新增可用來停用此元件中的安裝指令碼的新`UseInstaller` 組態參數。
1.6.4	版本更新 Greengrass 2.4.0 版本的版本。
1.6.3	版本更新了 Greengrass 核 2.3.0 版本。
1.6.2	版本更新了 Greengrass 核 2.2.0 版本。
1.6.1	新功能 • 安裝深度學習執行階段 v1.6.0 及其相依性。 • 添加對在 ARV8（AArch64）平台上安裝 DLR 的支持。這擴展了對運行 NVIDIA 傑特森的 Greengrass 核心設備的機器學習支持，例如傑特森納米。 錯誤修復和改進 • 在虛擬環境AWS IoT Device SDK中安裝，以讀取元件組態並套用組態變更。 • 其他小錯誤修復和改進。

版本	變更
1.3.2	初始版本。安裝 DLR 版本 1.3.0。

TensorFlow 精簡版圖片分類

TensorFlow Lite 影像分類元件 (`aws.greengrass.TensorFlowLiteImageClassification`) 包含範例推論程式碼，可使用 TensorFlow Lite 執行階段和預先訓練的 MobileNet 1.0 量化模型範例來執行影像分類推論。此組件使用變體TensorFlow 精簡版圖片分類模型店和TensorFlow 精簡版運行組件作為依賴關係來下載 TensorFlow Lite 運行時和示例模型。

若要將此推論元件與自訂訓練的 TensorFlow Lite 模型搭配使用，請建立相依模型存放區元件的自訂版本。若要使用您自己的自訂推論程式碼，您可以使用此元件的方式做為範本來建立自訂推論元件。

主題

版本

此元件具有下列版本：

- 2.1.x

Type

此元件是一般元件 (`aws.greengrass.generic`)。Greengrass 核會執行元件的生命週期指令碼。

如需詳細資訊，請參閱 元件類型。

作業系統

此元件可安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件具有下列需求：

- 在運行 Amazon Linux 2 或 Ubuntu 18.04 的 Greengrass 核心設備上，GNU C 庫（glibc）2.27 版或更高版本安裝在設備上。
- 在 ARMv7L 設備上，如樹莓派，對於設備上安裝了 OpenCV-Python 的依賴關係。執行下列命令以安裝相依性。

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
 libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- 樹莓派運行樹莓派 OS 靶心設備必須滿足以下要求:
  - NumPy 1.22.4 或更新版本已安裝在裝置上。樹莓派 OS 靶心包括早期版本的 NumPy，因此您可以在設備上運行以下命令進 NumPy 行升級。

    ```
    pip3 install --upgrade numpy
    ```

  - 裝置上啟用的舊式攝影機堆疊。樹莓派 OS Bullseye 包括一個新的相機堆疊，預設情況下啟用且不相容，因此您必須啟用傳統的相機堆疊。

    啟用舊式相機堆疊

    1. 運行以下命令以打開樹莓派配置工具。

       ```
       sudo raspi-config
       ```

    2. 選取介面選項。
    3. 選取「舊式相機」以啟用舊式相機堆疊。
    4. 重新啟動 Raspberry Pi。

相依性

部署元件時，AWS IoT Greengrass也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的需求，才能成功部署元件。本節列出此元件之已發行版本的相依性，以及定義每個相依性之元件版本的語意版本條件約束。您也可以在AWS IoT Greengrass主控台中檢視元件每個版本的相依性。在元件詳細資料頁面上，尋找 [相依性] 清單。

2.1.11

下表列出此元件 2.1.11 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
TensorFlow 精簡版圖片分類模型店	> = 2.1.0	硬式
TensorFlow 精簡版	>=2.5.0	硬式

2.1.10

下表列出此元件 2.1.10 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
TensorFlow 精簡版圖片分類模型店	> = 2.1.0	硬式
TensorFlow 精簡版	>=2.5.0	硬式

2.1.9

下表列出此元件 2.1.9 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
TensorFlow 精簡版圖片分類模型店	> = 2.1.0	硬式
TensorFlow 精簡版	>=2.5.0	硬式

2.1.8

下表列出此元件 2.1.8 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
TensorFlow 精簡版圖片分類模型店	> = 2.1.0	硬式
TensorFlow 精簡版	>=2.5.0	硬式

2.1.7

下表列出此元件 2.1.7 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
TensorFlow 精簡版圖片分類模型店	> = 2.1.0	硬式
TensorFlow 精簡版	>=2.5.0	硬式

2.1.6

下表列出此元件 2.1.6 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
TensorFlow 精簡版圖片分類模型店	> = 2.1.0	硬式
TensorFlow 精簡版	>=2.5.0	硬式

2.1.5

下表列出此元件 2.1.5 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
TensorFlow 精簡版圖片分類模型店	> = 2.1.0	硬式
TensorFlow 精簡版	>=2.5.0	硬式

2.1.4

下表列出此元件 2.1.4 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.0.0	軟式
TensorFlow 精簡版圖片分類模型店	> = 2.1.0	硬式
TensorFlow 精簡版	>=2.5.0	硬式

2.1.3

下表列出此元件 2.1.3 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
TensorFlow 精簡版圖片分類模型店	> = 2.1.0	硬式
TensorFlow 精簡版	>=2.5.0	硬式

2.1.2

下表列出此元件 2.1.2 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
TensorFlow 精簡版圖片分類模型店	> = 2.1.0	硬式
TensorFlow 精簡版	>=2.5.0	硬式

2.1.1

下表列出此元件 2.1.1 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
TensorFlow 精簡版圖片分類模型店	> = 2.1.0	硬式
TensorFlow 精簡版	>=2.5.0	硬式

2.1.0

下表列出此元件 2.1.0 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
TensorFlow 精簡版圖片分類模型店	> = 2.1.0	硬式
TensorFlow 精簡版	>=2.5.0	硬式

組態

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

`accessControl`

(選擇性) 包含授權原則的物件，可讓元件將訊息發佈至預設通知主題。

預設：

```
{
   "aws.greengrass.ipc.mqttproxy": {
      "aws.greengrass.TensorFlowLiteImageClassification:mqttproxy:1": {
         "policyDescription": "Allows access to publish via topic ml/tflite/image-
classification.",
         "operations": [
            "aws.greengrass#PublishToIoTCore"
         ],
         "resources": [
            "ml/tflite/image-classification"
         ]
      }
   }
}
```

`PublishResultsOnTopic`

(選擇性) 您要發佈推論結果的主題。如果修改此值，則還必須修改`accessControl`參數`resources`中的值以符合您的自訂主題名稱。

預設：`ml/tflite/image-classification`

`Accelerator`

您要使用的加速器。支援的值為 `cpu` 和 `gpu`。

相依模型元件中的範例模型僅支援 CPU 加速。若要將 GPU 加速與不同的自訂模型搭配使用，請建立自訂模型元件以覆寫公用模型元件。

預設：`cpu`

`ImageDirectory`

(選擇性) 推論元件讀取影像之設備上的資料夾路徑。您可以將此值修改為裝置上具有讀取/寫入存取權限的任何位置。

預設：*`/greengrass/v2`*`/packages/artifacts-unarchived/`*`component-name`*`/image_classification/sample_images/`

> **Note**
>
> 如果`UseCamera`將的值設定為`true`，則會忽略此組態參數。

`ImageName`

(選擇性) 推論元件用作進行預測輸入的影像名稱。元件會在中指定的資料夾中尋找影像`ImageDirectory`。依預設，元件會使用預設映像目錄中的範例影像。 AWS IoT Greengrass支援下列影像格式：`jpeg`jpg、png、和npy。

預設：`cat.jpeg`

> **Note**
>
> 如果`UseCamera`將的值設定為`true`，則會忽略此組態參數。

`InferenceInterval`

(選擇性) 推論程式碼進行的每個預測之間的時間 (以秒為單位)。範例推論程式碼會無限期執行，並在指定的時間間隔內重複其預測。例如，如果您要使用相機拍攝的影像進行即時預測，可以將其變更為較短的間隔。

預設：3600

ModelResourceKey

(選擇性) 在相依公用模型元件中使用的模型。只有在使用自訂元件覆寫公用模型元件時，才能修改此參數。

預設：

```
{
    "model": "TensorFlowLite-Mobilenet"
}
```

UseCamera

(選擇性) 字串值，定義是否使用來自連接至 Greengrass 核心裝置之攝影機的影像。支援的值為 true 和 false。

將此值設定為時true，範例推論程式碼會存取裝置上的攝影機，並在擷取的影像本機上執行推論。ImageName和ImageDirectory參數的值會被忽略。請確定執行此元件的使用者具有相機儲存擷取影像之位置的讀取/寫入存取權。

預設：false

> **Note**
>
> 當您檢視此元件的配方時，UseCamera組態參數不會顯示在預設組態中。不過，您可以在部署元件時，在組態合併更新中修改此參數的值。
> 當您設定UseCamera為時true，您還必須建立符號連結，以使推論元件能夠從執行階段元件建立的虛擬環境存取您的攝影機。如需將相機與範例推論元件搭配使用的詳細資訊，請參閱〈〉更新零組件組態。

本機記錄檔

此元件使用下列記錄檔。

Linux

```
/greengrass/v2/logs/aws.greengrass.TensorFlowLiteImageClassification.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.TensorFlowLiteImageClassification.log
```

若要檢視此元件的記錄

- 在核心裝置上執行下列命令，即時檢視此元件的記錄檔。以AWS IoT Greengrass根資料夾的路徑取代*/greengrass/v2*或 *C:\greengrass\v2*。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/
  aws.greengrass.TensorFlowLiteImageClassification.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs
  \aws.greengrass.TensorFlowLiteImageClassification.log -Tail 10 -Wait
  ```

變更記錄

下表說明元件每個版本中的變更。

版本	變更
2.1.11	版本更新 Greengrass 2.12.0 版本釋放。
2.1.10	版本更新了 Greengrass 核 2.11.0 版本釋放。
2.1.9	版本更新了 Greengrass 2.10.0 版本。
2.1.8	版本更新 Greengrass 2.9.0 版本釋放。
2.1.7	版本更新 Greengrass 2.8.0 版本的版本。
2.1.6	版本更新了 Greengrass 核 2.7.0 版本釋放。
2.1.5	版本更新 Greengrass 2.6.0 版本發布。

版本	變更
2.1.4	版本更新了 Greengrass 核 2.5.0 版本。
2.1.3	版本更新 Greengrass 2.4.0 版本的版本。
2.1.2	版本更新了 Greengrass 核 2.3.0 版本。
2.1.1	版本更新了 Greengrass 核 2.2.0 版本。
2.1.0	初始版本。

TensorFlow 精簡型物體偵測

TensorFlow Lite 物件偵測元件 (`aws.greengrass.TensorFlowLiteObjectDetection`) 包含範例推論程式碼，可使用 TensorFlow Lite 執行物件偵測推論，以及預先訓練的單次射擊偵測 (SSD) MobileNet 1.0 模型範例。此組件使用變體TensorFlow 精簡型物件偵測模型商店和TensorFlow 精簡版運行組件作為依賴關係來下載 TensorFlow Lite 和示例模型。

若要將此推論元件與自訂訓練的 TensorFlow Lite 模型搭配使用，您可以建立相依模型存放區元件的自訂版本。若要使用您自己的自訂推論程式碼，請使用此元件的方式做為範本來建立自訂推論元件。

主題

版本

此元件具有下列版本：

- 2.1.x

Type

此元件是一般元件 (`aws.greengrass.generic`)。Greengrass 核會執行元件的生命週期指令碼。

如需詳細資訊，請參閱 元件類型。

作業系統

此元件可安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件具有下列需求：

- 在運行 Amazon Linux 2 或 Ubuntu 18.04 的 Greengrass 核心設備上，GNU C 庫（glibc）2.27 版或更高版本安裝在設備上。
- 在 ARMV7L 設備上，如樹莓派，對於設備上安裝了 OpenCV-Python 的依賴關係。執行下列命令以安裝相依性。

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
 libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- 樹莓派運行樹莓派 OS 靶心設備必須滿足以下要求:
  - NumPy 1.22.4 或更新版本已安裝在裝置上。樹莓派 OS 靶心包括早期版本的 NumPy，因此您可以運行以下命令在設備 NumPy 上升級。

    ```
    pip3 install --upgrade numpy
    ```

  - 裝置上啟用的舊式攝影機堆疊。樹莓派 OS Bullseye 包括一個新的相機堆疊，預設情況下啟用且不相容，因此您必須啟用傳統的相機堆疊。

    啟用舊式相機堆疊

    1. 運行以下命令以打開樹莓派配置工具。

       ```
       sudo raspi-config
       ```

2. 選取介面選項。
3. 選取「舊式相機」以啟用舊式相機堆疊。
4. 重新啟動 Raspberry Pi。

相依性

部署元件時，AWS IoT Greengrass也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的需求，才能成功部署元件。本節列出此元件之已發行版本的相依性，以及定義每個相依性之元件版本的語意版本條件約束。您也可以在AWS IoT Greengrass主控台中檢視元件每個版本的相依性。在元件詳細資料頁面上，尋找 [相依性] 清單。

2.1.11

下表列出此元件 2.1.11 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
TensorFlow 精簡版圖片分類模型店	> = 2.1.0	硬式
TensorFlow 精簡版	>=2.5.0	硬式

2.1.10

下表列出此元件 2.1.10 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
TensorFlow 精簡版圖片分類模型店	> = 2.1.0	硬式
TensorFlow 精簡版	>=2.5.0	硬式

2.1.9

下表列出此元件 2.1.9 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
TensorFlow 精簡版圖片分類模型店	> = 2.1.0	硬式
TensorFlow 精簡版	>=2.5.0	硬式

2.1.8

下表列出此元件 2.1.8 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
TensorFlow 精簡版圖片分類模型店	> = 2.1.0	硬式
TensorFlow 精簡版	>=2.5.0	硬式

2.1.7

下表列出此元件 2.1.7 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
TensorFlow 精簡版圖片分類模型店	> = 2.1.0	硬式
TensorFlow 精簡版	>=2.5.0	硬式

2.1.6

下表列出此元件 2.1.6 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
TensorFlow 精簡版圖片分類模型店	> = 2.1.0	硬式
TensorFlow 精簡版	>=2.5.0	硬式

2.1.5

下表列出此元件 2.1.5 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
TensorFlow 精簡版圖片分類模型店	> = 2.1.0	硬式
TensorFlow 精簡版	>=2.5.0	硬式

2.1.4

下表列出此元件 2.1.4 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.0.0	軟式
TensorFlow 精簡版圖片分類模型店	> = 2.1.0	硬式
TensorFlow 精簡版	>=2.5.0	硬式

2.1.3

下表列出此元件 2.1.3 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
TensorFlow 精簡版圖片分類模型店	> = 2.1.0	硬式
TensorFlow 精簡版	>=2.5.0	硬式

2.1.2

下表列出此元件 2.1.2 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
TensorFlow 精簡版圖片分類模型店	> = 2.1.0	硬式
TensorFlow 精簡版	>=2.5.0	硬式

2.1.1

下表列出此元件 2.1.1 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
TensorFlow 精簡版圖片分類模型店	> = 2.1.0	硬式
TensorFlow 精簡版	>=2.5.0	硬式

2.1.0

下表列出此元件 2.1.0 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
TensorFlow 精簡版圖片分類模型店	> = 2.1.0	硬式
TensorFlow 精簡版	>=2.5.0	硬式

組態

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

`accessControl`

(選擇性) 包含授權原則的物件，可讓元件將訊息發佈至預設通知主題。

預設：

```
{
   "aws.greengrass.ipc.mqttproxy": {
      "aws.greengrass.TensorFlowLiteObjectDetection:mqttproxy:1": {
         "policyDescription": "Allows access to publish via topic ml/tflite/object-
detection.",
         "operations": [
            "aws.greengrass#PublishToIoTCore"
         ],
         "resources": [
            "ml/tflite/object-detection"
         ]
      }
   }
}
```

`PublishResultsOnTopic`

(選擇性) 您要發佈推論結果的主題。如果修改此值，則還必須修改`accessControl`參數`resources`中的值以符合您的自訂主題名稱。

預設：ml/tflite/object-detection

`Accelerator`

您要使用的加速器。支援的值為 cpu 和 gpu。

相依模型元件中的範例模型僅支援 CPU 加速。若要在不同的自訂模型中使用 GPU 加速，請建立自訂模型元件以覆寫公用模型元件。

預設：cpu

`ImageDirectory`

(選擇性) 推論元件讀取影像之設備上的資料夾路徑。您可以將此值修改為裝置上具有讀取/寫入存取權限的任何位置。

預設：*/greengrass/v2*/packages/artifacts-unarchived/*component-name*/object_detection/sample_images/

> **Note**
>
> 如果UseCamera將的值設定為true，則會忽略此組態參數。

`ImageName`

(選擇性) 推論元件用作進行預測輸入的影像名稱。元件會在中指定的資料夾中尋找影像ImageDirectory。依預設，元件會使用預設映像目錄中的範例影像。 AWS IoT Greengrass支援下列影像格式：jpegjpg、png、和npy。

預設：objects.jpg

> **Note**
>
> 如果UseCamera將的值設定為true，則會忽略此組態參數。

`InferenceInterval`

(選擇性) 推論程式碼進行的每個預測之間的時間 (以秒為單位)。範例推論程式碼會無限期執行，並在指定的時間間隔內重複其預測。例如，如果您要使用相機拍攝的影像進行即時預測，可以將其變更為較短的間隔。

預設：3600

ModelResourceKey

(選擇性) 在相依公用模型元件中使用的模型。只有在使用自訂元件覆寫公用模型元件時，才能修改此參數。

預設：

```
{
    "model": "TensorFlowLite-SSD"
}
```

UseCamera

(選擇性) 字串值，定義是否使用來自連接至 Greengrass 核心裝置之攝影機的影像。支援的值為 true 和 false。

將此值設定為時true，範例推論程式碼會存取裝置上的攝影機，並在擷取的影像本機上執行推論。ImageName和ImageDirectory參數的值會被忽略。請確定執行此元件的使用者具有相機儲存擷取影像之位置的讀取/寫入權限。

預設：false

> **Note**
>
> 當您檢視此元件的配方時，UseCamera組態參數不會顯示在預設組態中。不過，您可以在部署元件時，在組態合併更新中修改此參數的值。
> 當您設定UseCamera為時true，您還必須建立符號連結，以使推論元件能夠從執行階段元件建立的虛擬環境存取您的攝影機。如需將相機與範例推論元件搭配使用的詳細資訊，請參閱更新零組件組態。

> **Note**
>
> 當您檢視此元件的配方時，UseCamera組態參數不會顯示在預設組態中。不過，您可以在部署元件時，在組態合併更新中修改此參數的值。
> 當您設定UseCamera為時true，您還必須建立符號連結，以使推論元件能夠從執行階段元件建立的虛擬環境存取您的攝影機。如需將相機與範例推論元件搭配使用的詳細資訊，請參閱更新零組件組態。

本機記錄檔

此元件會使用下列記錄檔。

Linux

```
/greengrass/v2/logs/aws.greengrass.TensorFlowLiteObjectDetection.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.TensorFlowLiteObjectDetection.log
```

若要檢視此元件的記錄

- 在核心裝置上執行下列命令，即時檢視此元件的記錄檔。以AWS IoT Greengrass根資料夾的路徑取代*/greengrass/v2*或 *C:\greengrass\v2*。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/
  aws.greengrass.TensorFlowLiteObjectDetection.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs
  \aws.greengrass.TensorFlowLiteObjectDetection.log -Tail 10 -Wait
  ```

變更記錄

下表說明元件每個版本中的變更。

版本	變更
2.1.11	版本更新了 Greengrass 核 2.12.0 版本釋放。
2.1.10	版本更新了 Greengrass 核 2.11.0 版本釋放。
2.1.9	版本更新了 Greengrass 2.10.0 版本。

版本	變更
2.1.8	版本更新 Greengrass 2.9.0 版本釋放。
2.1.7	版本更新 Greengrass 2.8.0 版本的版本。
2.1.6	版本更新了 Greengrass 核 2.7.0 版本釋放。
2.1.5	版本更新 Greengrass 2.6.0 版本的版本。
2.1.4	版本更新了 Greengrass 核 2.5.0 版本。
2.1.3	版本更新 Greengrass 2.4.0 版本的版本。
2.1.2	版本更新了 Greengrass 核 2.3.0 版本。
2.1.1	錯誤修復和改進 • 修正影像縮放問題，此問題會導致範例 TensorFlow Lite 物件偵測推論結果中不正確的邊界方塊。
2.1.0	初始版本。

TensorFlow 精簡版圖片分類模型店

TensorFlow Lite 映像分類模型存放區 (`variant.TensorFlowLite.ImageClassification.ModelStore`) 是一種機器學習模型元件，其中包含預先訓練的 MobileNet v1 模型做為 Greengrass 成品。在此組件中使用的示例模型是從TensorFlow集線器獲取並使用TensorFlow 精簡版實現。

TensorFlow 精簡版圖片分類推論元件會使用此元件做為模型來源的相依性。若要使用自訂訓練的 TensorFlow Lite 模型，請建立此模型元件的自訂版本，並將您的自訂模型納入為元件人工因素。您可以使用此元件的配方做為範本來建立自訂模型元件。

主題

- 版本
- Type
- 作業系統
- 要求

- 相依性
- 組態
- 本機記錄檔
- 變更記錄

版本

此元件具有下列版本：

- 2.1.x

Type

此元件是一般元件 (`aws.greengrass.generic`)。Greengrass 核會執行元件的生命週期指令碼。

如需詳細資訊，請參閱 元件類型。

作業系統

此元件可安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件具有下列需求：

- 在運行 Amazon Linux 2 或 Ubuntu 18.04 的 Greengrass 核心設備上，GNU C 庫（glibc）2.27 版或更高版本安裝在設備上。
- 在 ARMv7L 設備上，如樹莓派，對於設備上安裝了 OpenCV-Python 的依賴關係。執行下列命令以安裝相依性。

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
 libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- 樹莓派運行樹莓派 OS 靶心設備必須滿足以下要求:
  - NumPy 1.22.4 或更新版本已安裝在裝置上。樹莓派 OS 靶心包括早期版本的 NumPy，因此您可以運行以下命令在設備 NumPy 上升級。

```
pip3 install --upgrade numpy
```

- 裝置上啟用的舊式攝影機堆疊。樹莓派 OS Bullseye 包括一個新的相機堆疊，預設情況下啟用且不相容，因此您必須啟用傳統的相機堆疊。

  啟用舊式相機堆疊

  1. 運行以下命令以打開樹莓派配置工具。

     ```
     sudo raspi-config
     ```

  2. 選取介面選項。
  3. 選取「舊式相機」以啟用舊式相機堆疊。
  4. 重新啟動 Raspberry Pi。

相依性

部署元件時，AWS IoT Greengrass也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的需求，才能成功部署元件。本節列出此元件之已發行版本的相依性，以及定義每個相依性之元件版本的語意版本條件約束。您也可以在AWS IoT Greengrass主控台中檢視元件每個版本的相依性。在元件詳細資料頁面上，尋找 [相依性] 清單。

2.1.11

下表列出此元件 2.1.11 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.10

下表列出此元件 2.1.10 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.9

下表列出此元件 2.1.9 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.8

下表列出此元件 2.1.8 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.7

下表列出此元件 2.1.7 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.6

下表列出此元件 2.1.6 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.5

下表列出此元件 2.1.5 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.4

下表列出此元件 2.1.4 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.0.0	軟式

2.1.3

下表列出此元件 2.1.3 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.2

下表列出此元件 2.1.2 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.1

下表列出此元件 2.1.1 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.0

下表列出此元件 2.1.0 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

組態

此元件沒有任何組態參數。

本機記錄檔

此元件不會輸出記錄檔。

變更記錄

下表說明元件每個版本的變更。

版本	變更
2.1.11	版本更新了 Greengrass 2.12.0 版本。
2.1.10	版本更新 Greengrass 2.11.0 版本釋放。
2.1.9	版本更新了 Greengrass 2.10.0 版本。
2.1.8	版本更新 Greengrass 2.9.0 版本釋放。
2.1.7	版本更新 Greengrass 2.8.0 版本的版本。
2.1.6	版本更新了 Greengrass 核 2.7.0 版本釋放。
2.1.5	版本更新 Greengrass 2.6.0 版本的版本。
2.1.4	版本更新了 Greengrass 核 2.5.0 版本。
2.1.3	版本更新 Greengrass 2.4.0 版本的版本。
2.1.2	版本更新了 Greengrass 核 2.3.0 版本。

版本	變更
2.1.1	版本更新了 Greengrass 核 2.2.0 版本。
2.1.0	初始版本。

TensorFlow 精簡型物件偵測模型商店

TensorFlow Lite 物件偵測模型存放區 (`variant.TensorFlowLite.ObjectDetection.ModelStore`) 是一種機器學習模型元件，其中包含預先訓練的單次射擊偵測 (SSD) MobileNet 模型，做為 Greengrass 假影。在此組件中使用的示例模型是從TensorFlow 集線器獲取並使用TensorFlow 精簡版實現。

TensorFlow Lite 物件偵測推論元件會使用此元件做為模型來源的相依性。若要使用自訂訓練的 TensorFlow Lite 模型，請建立此模型元件的自訂版本，並將您的自訂模型納入為元件人工因素。您可以使用此元件的配方做為範本來建立自訂模型元件。

主題

版本

此元件具有下列版本：

- 2.1.x

Type

此元件是一般元件 (`aws.greengrass.generic`)。Greengrass 核會執行元件的生命週期指令碼。

如需詳細資訊，請參閱 元件類型。

作業系統

此元件可安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件具有下列需求：

- 在運行 Amazon Linux 2 或 Ubuntu 18.04 的 Greengrass 核心設備上，GNU C 庫（glibc）2.27 版或更高版本安裝在設備上。
- 在 ARMV7L 設備上，如樹莓派，對於設備上安裝了 OpenCV-Python 的依賴關係。執行下列命令以安裝相依性。

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
 libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- 樹莓派運行樹莓派操作系統靶心設備必須滿足以下要求:
  - NumPy 1.22.4 或更新版本已安裝在裝置上。樹莓派 OS 靶心包括早期版本的 NumPy，因此您可以在設備上運行以下命令進 NumPy 行升級。

    ```
    pip3 install --upgrade numpy
    ```

  - 裝置上啟用的舊式攝影機堆疊。樹莓派 OS Bullseye 包括一個新的攝像機堆棧，默認情況下啟用並且不兼容，因此您必須啟用傳統的相機堆棧。

    啟用舊式相機堆疊的步驟

    1. 運行以下命令以打開樹莓派配置工具。

       ```
       sudo raspi-config
       ```

    2. 選取介面選項。
    3. 選取「舊式相機」以啟用舊式相機堆疊。
    4. 重新啟動 Raspberry Pi。

相依性

部署元件時，AWS IoT Greengrass也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的需求，才能成功部署元件。本節列出此元件之已發行版本的相依性，以及定義每個相依性之元件版本的語意版本條件約束。您也可以在AWS IoT Greengrass主控台中檢視元件每個版本的相依性。在元件詳細資料頁面上，尋找 [相依性] 清單。

2.1.11

下表列出此元件 2.1.11 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.10

下表列出此元件 2.1.10 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.9

下表列出此元件 2.1.9 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0 <2.11.0	軟式

2.1.8

下表列出此元件 2.1.8 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.7

下表列出此元件 2.1.7 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.6

下表列出此元件 2.1.6 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.5

下表列出此元件 2.1.5 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.4

下表列出此元件 2.1.4 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.0.0	軟式

2.1.3

下表列出此元件 2.1.3 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.2

下表列出此元件 2.1.2 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.1

下表列出此元件 2.1.1 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.1.0

下表列出此元件 2.1.0 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

組態

此元件沒有任何組態參數。

本機記錄檔

此元件不會輸出記錄檔。

變更記錄

下表說明元件每個版本中的變更。

版本	變更
2.1.11	版本更新了 Greengrass 核 2.12.0 版本釋放。
2.1.10	版本更新了 Greengrass 核 2.11.0 版本釋放。
2.1.9	版本更新了 Greengrass 核 2.10.0 版本釋放。
2.1.8	版本更新 Greengrass 2.9.0 版本釋放。
2.1.7	版本更新 Greengrass 2.8.0 版本的版本。
2.1.6	版本更新了 Greengrass 核 2.7.0 版本釋放。
2.1.5	版本更新 Greengrass 2.6.0 版本發布。
2.1.4	版本更新了 Greengrass 核 2.5.0 版本。
2.1.3	版本更新 Greengrass 2.4.0 版本的版本。
2.1.2	版本更新了 Greengrass 核 2.3.0 版本。
2.1.1	版本更新了 Greengrass 核 2.2.0 版本。
2.1.0	初始版本。

TensorFlow 精簡版運行

TensorFlow Lite 執行階段元件 (`variant.TensorFlowLite`) 包含一個指令碼，可在裝置上的虛擬環境中安裝 TensorFlow Lite 版本 2.5.0 及其相依性。TensorFlow Lite 圖像分類和 TensorFlow Lite 對象檢測組件使用此運行時組件作為安裝 TensorFlow Lite 的依賴項。

> **Note**
>
> TensorFlow Lite 執行階段元件 v2.5.6 及更新版本會重新安裝 TensorFlow Lite 執行階段及其相依性的現有安裝。此重新安裝有助於確保核心設備運行 TensorFlow Lite 及其依賴項的兼容版本。

若要使用不同的執行階段，您可以使用此元件的方案做為範本來建立自訂機器學習元件。

主題

版本

此元件具有下列版本：

- 2.5.x

Type

此元件是一般元件 (`aws.greengrass.generic`)。Greengrass 核會執行元件的生命週期指令碼。

如需詳細資訊，請參閱 元件類型。

作業系統

此元件可安裝在執行下列作業系統的核心裝置上：

- Linux

- Windows

要求

此元件具有下列需求：

- 在運行 Amazon Linux 2 或 Ubuntu 18.04 的 Greengrass 核心設備上，GNU C 庫（glibc）2.27 版或更高版本安裝在設備上。
- 在 ARMV7L 設備上，如樹莓派，對於設備上安裝了 OpenCV-Python 的依賴關係。執行下列命令以安裝相依性。

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
 libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- 樹莓派運行樹莓派 OS 靶心設備必須滿足以下要求:
  - NumPy 1.22.4 或更新版本已安裝在裝置上。樹莓派 OS 靶心包括早期版本的 NumPy，因此您可以在設備上運行以下命令進 NumPy 行升級。

    ```
    pip3 install --upgrade numpy
    ```

  - 裝置上啟用的舊式攝影機堆疊。樹莓派 OS Bullseye 包括一個新的攝像機堆棧，默認情況下啟用並且不兼容，因此您必須啟用傳統的相機堆棧。

    啟用舊式相機堆疊

    1. 運行以下命令以打開樹莓派配置工具。

       ```
       sudo raspi-config
       ```

    2. 選取介面選項。
    3. 選取「舊式相機」以啟用舊式相機堆疊。
    4. 重新啟動 Raspberry Pi。

端點和連接埠

根據預設，此元件會根據核心裝置使用的平台，使用`apt`、`yum`、`brew`、和`pip`指令，使用安裝程式指令碼來安裝套件。此元件必須能夠對各種套件索引和儲存庫執行輸出要求，才能執行安裝程式指令碼。若要允許此元件透過 Proxy 或防火牆的輸出流量，您必須識別套件索引的端點，以及核心裝置連線以進行安裝的存放庫。

當您識別此元件的安裝指令碼所需的端點時，請考慮下列事項：

- 端點取決於核心設備的平台。例如，執行 Ubuntu 的核心裝置會使用`apt`而不是`yum`或`brew`。此外，使用相同套件索引的裝置可能會有不同的來源清單，因此可能會從不同的軟體庫擷取套件。
- 使用相同套件索引的多個裝置之間的端點可能會有所不同，因為每個裝置都有自己的來源清單，用於定義擷取套件的位置。
- 端點可能會隨時間變更。每個套裝程式索引都會提供您下載套裝程式的儲存區域 URL，套裝程式的擁有者可以變更套裝程式索引所提供的 URL。

如需有關此元件安裝之相依性以及如何停用安裝程式指令碼的詳細資訊，請參閱UseInstaller組態參數。

如需有關基本作業所需的端點和連接埠的詳細資訊，請參閱允許裝置流量透過 Proxy 或防火牆。

相依性

部署元件時，AWS IoT Greengrass也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的需求，才能成功部署元件。本節列出此元件之已發行版本的相依性，以及定義每個相依性之元件版本的語意版本條件約束。您也可以在AWS IoT Greengrass主控台中檢視元件每個版本的相依性。在元件詳細資料頁面上，尋找 [相依性] 清單。

2.5.14

下表列出此元件 2.5.14 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.5.13

下表列出此元件 2.5.13 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.5.12

下表列出此元件 2.5.12 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.5.11

下表列出此元件 2.5.11 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.5.10

下表列出此元件 2.5.10 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.5.9

下表列出此元件 2.5.9 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.5.8

下表列出此元件 2.5.8 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.5.5 - 2.5.7

下表列出此元件 2.5.5 到 2.5.7 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.0.0	軟式

2.5.3 and 2.5.4

下表列出此元件 2.5.3 和 2.5.4 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.5.2

下表列出此元件 2.5.2 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.5.1

下表列出此元件 2.5.1 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.5.0

下表列出此元件 2.5.0 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

如需有關元件相依性的詳細資訊，請參閱元件方案參考。

組態

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

`MLRootPath`

(選擇性) 推論元件讀取影像和寫入推論結果的 Linux 核心裝置上的資料夾路徑。您可以將此值修改為裝置上執行此元件的使用者具有讀取/寫入存取權限的任何位置。

預設：*`/greengrass/v2`*`/work/variant.TensorFlowLite/greengrass_ml`

`WindowsMLRootPath`

此功能可在此元件 v1.6.6 及更新版本中使用。

(選擇性) 推論元件讀取影像和寫入推論結果的 Windows 核心裝置上的資料夾路徑。您可以將此值修改為裝置上執行此元件的使用者具有讀取/寫入存取權限的任何位置。

預設：*`C:\greengrass\v2`*`\\work\\variant.DLR\\greengrass_ml`

`UseInstaller`

(選擇性) 字串值，定義是否使用此元件中的安裝程式指令碼來安裝 TensorFlow Lite 及其相依性。支援的值為 `true` 和 `false`。

`false`如果您要使用自訂指令碼進行 TensorFlow Lite 安裝，或者想要在預先建置的 Linux 映像檔中包含執行階段相依性，請將此值設定為。若要將此元件與AWS提供的 TensorFlow Lite 推論元件搭配使用，請安裝下列程式庫 (包括任何相依性)，並讓系統使用者可以使用這些程式庫，例如`ggc_user`執行 ML 元件的系統使用者。

- Python 3.8 或更高版本，包括`pip`您的 Python 版本
- TensorFlow 精簡版

- NumPy
- Python
- AWS IoT Device SDK對於 Python 的 V2
- AWS通用運行時（CRT）Python
- 皮卡馬拉（用於樹莓派設備）
- awscam模塊（用於AWS DeepLens設備）
- LibGL (適用於 Linux 設備)

預設：`true`

用量

在`UseInstaller`組態參數設定為`true`的情況下使用此元件，可在您的裝置上安裝 TensorFlow Lite 及其相依性。該組件在您的設備上設置一個虛擬環境，其中包括精簡 TensorFlow 版所需的 OpenCV 和 NumPy 庫。

> **Note**
>
> 此元件中的安裝程式指令碼也會安裝最新版本的其他系統程式庫，以便在您的裝置上設定虛擬環境，以及使用已安裝的機器學習架構。這可能會升級您裝置上現有的系統程式庫。請檢閱下表，以取得此元件為每個支援的作業系統安裝的程式庫清單。如果您要自訂此安裝程序，請將`UseInstaller`組態參數設定為`false`，然後開發您自己的安裝程式指令碼。

平台	裝置系統上安裝的程式庫	安裝在虛擬環境中的程式庫
Armv7l	`build-essential` ，`cmake`，`ca-certificates` ，`git`	`setuptools` ，`wheel`
Amazon Linux 2	`mesa-libGL`	無
Ubuntu	`wget`	無

部署推論元件時，此執行階段元件會先驗證您的裝置是否已安裝 TensorFlow Lite 及其相依性。如果沒有，則執行階段元件會為您安裝它們。

本機記錄檔

此元件使用下列記錄檔。

Linux

```
/greengrass/v2/logs/variant.TensorFlowLite.log
```

Windows

```
C:\greengrass\v2\logs\variant.TensorFlowLite.log
```

若要檢視此元件的記錄

- 在核心裝置上執行下列命令，即時檢視此元件的記錄檔。以AWS IoT Greengrass根資料夾的路徑取代*/greengrass/v2*或 *C:\greengrass\v2*。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/variant.TensorFlowLite.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\variant.TensorFlowLite.log -Tail 10 -Wait
  ```

變更記錄

下表說明元件每個版本的變更。

版本	變更
2.5.14	版本更新 Greengrass 2.12.0 版本釋放。
2.5.13	版本更新了 Greengrass 核 2.11.0 版本釋放。
2.5.12	版本更新了 Greengrass 2.10.0 版本。
2.5.11	版本更新 Greengrass 2.9.0 版本釋放。

版本	變更
2.5.10	版本更新 Greengrass 2.8.0 版本的版本。
2.5.9	版本更新了 Greengrass 核 2.7.0 版本釋放。
2.5.8	版本更新 Greengrass 2.6.0 版本發布。
2.5.7	錯誤修復和改進 • 更新安`UseInstaller` 裝指令碼以安裝 libGL，此指令碼在某些 Linux 平台上預設不可用。 • 更新`UseInstaller` 安裝腳本以始終在此組件的虛擬環境中使用 Python 3.9。此變更有助於確保與其他程式庫的相容性。
2.5.6	錯誤修復和改進 • 更新此元件以安裝 TensorFlow 精簡版 2.5.0 (`tflite-runtime-2.5.0.post1`) 的最新修補程式，因此您可以在 Python 3.9 中使用此元件。如果此元件無法安裝該修補程式，則會`tflite-runtime-2.5.0` 改為安裝。 • 更新此元件以重新安裝 TensorFlow Lite 及其相依性的現有安裝。此變更有助於確保核心裝置執行 TensorFlow Lite 及其相依性的相容版本。
2.5.5	新功能 • 添加對運行 Windows 的核心設備的支持。 • 新增可用來在 Windows 核心裝置上設定推論結果資料夾的新`WindowsMLRootPath` 組態參數。
2.5.4	新功能 • 新增可讓您停用此元件中的安裝指令碼的新`UseInstaller` 組態參數。
2.5.3	版本更新 Greengrass 2.4.0 版本的版本。
2.5.2	版本更新了 Greengrass 核 2.3.0 版本。
2.5.1	版本更新了 Greengrass 核 2.2.0 版本。
2.5.0	初始版本。

通訊協定介面卡

Modbus-RTU 通訊協定介面卡元件 (`aws.greengrass.Modbus`) 會輪詢來自本地 Modbus RTU 裝置的資訊。

若要從具有此元件的本機 Modbus RTU 裝置要求資訊，請將訊息發佈至此元件訂閱的主題。在訊息中，指定要傳送至裝置的 Modbus RTU 要求。然後，此元件會發佈包含 Modbus RTU 要求結果的回應。

> **Note**
>
> 此元件提供與 V1 中 AWS IoT Greengrass 的 Modbus RTU 通訊協定介面卡連接器類似的功能。如需詳細資訊，請參閱 AWS IoT Greengrass V1 開發人員指南中的 Modbus RTU 通訊協定介面卡連接器。

主題

版本

此元件具有下列版本：

- 2.1.x

- 2.0.x

Type

這個組件是一個 Lambda 組件（`aws.greengrass.lambda`）。Greengrass 核使用 Lambda 啟動器組件運行此組件的 Lambda 函數。

如需詳細資訊，請參閱 元件類型。

作業系統

此元件只能安裝在 Linux 核心裝置上。

要求

此元件具有下列需求：

- 您的核心裝置必須符合執行 Lambda 函數的需求。如果您希望核心裝置執行容器化 Lambda 函數，則裝置必須符合要求才能執行。如需詳細資訊，請參閱 Lambda 函數要求。
- Python 版本 3.7 安裝在核心設備上，並添加到 PATH 環境變量中。
- AWS IoT Greengrass 核心裝置與 Modbus 裝置之間的實體連線。核心設備必須通過串口（例如 USB 端口）實際連接到 Modbus RTU 網絡。
- 若要從此元件接收輸出資料，您必須在部署此元件時，合併舊版訂閱路由器元件的下列組態更新 (`aws.greengrass.LegacySubscriptionRouter`)。此配置指定此組件發布響應的主題。

  Legacy subscription router v2.1.x

  ```
  {
    "subscriptions": {
      "aws-greengrass-modbus": {
        "id": "aws-greengrass-modbus",
        "source": "component:aws.greengrass.Modbus",
        "subject": "modbus/adapter/response",
        "target": "cloud"
      }
    }
  }
  ```

  Legacy subscription router v2.0.x

  ```
  {
  ```

```
  "subscriptions": {
    "aws-greengrass-modbus": {
      "id": "aws-greengrass-modbus",
      "source": "arn:aws:lambda:region:aws:function:aws-greengrass-
modbus:version",
      "subject": "modbus/adapter/response",
      "target": "cloud"
    }
  }
}
```

- 將##替換為您 AWS 區域 使用的區域。
- 將##取代為此元件執行的 Lambda 函數版本。若要尋找 Lambda 函數版本，您必須檢視要部署之此元件版本的方案。在AWS IoT Greengrass 主控台中開啟此元件的詳細資料頁面，然後尋找 Lambda 函數索引鍵值組。此鍵值對包含 Lambda 函數的名稱和版本。

> **Important**
>
> 每次部署此元件時，您都必須更新舊版訂閱路由器上的 Lambda 函數版本。這可確保您針對部署的元件版本使用正確的 Lambda 函數版本。

如需詳細資訊，請參閱 建立部署。

- 支援在 VPC 中執行 Modbus-RTU 通訊協定介面卡。

相依性

部署元件時， AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的需求，才能成功部署元件。本節列出此元件之已發行版本的相依性，以及定義每個相依性之元件版本的語意版本條件約束。您也可以在AWS IoT Greengrass 主控台中檢視元件每個版本的相依性。在元件詳細資料頁面上，尋找 [相依性] 清單。

2.1.8

下表列出此元件 2.1.8 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	硬式

相依性	兼容版本	相依性類型
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
代幣交換服務	^2.0.0	硬式

2.1.7

下表列出此元件 2.1.7 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	硬式
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
代幣交換服務	^2.0.0	硬式

2.1.6

下表列出此元件 2.1.6 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0 <2.11.0	硬式
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
代幣交換服務	^2.0.0	硬式

2.1.4 and 2.1.5

下表列出此元件 2.1.4 和 2.1.5 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	硬式
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
代幣交換服務	^2.0.0	硬式

2.1.3

下表列出此元件 2.1.3 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.0.0	硬式
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
代幣交換服務	^2.0.0	硬式

2.1.2

下表列出此元件 2.1.2 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	硬式
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
代幣交換服務	^2.0.0	硬式

2.1.1

下表列出此元件 2.1.1 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	硬式
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
代幣交換服務	^2.0.0	硬式

2.0.8 and 2.1.0

下表列出此元件 2.0.8 和 2.1.0 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.0.0	硬式
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
代幣交換服務	^2.0.0	硬式

2.0.7

下表列出此元件 2.0.7 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	硬式
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式

相依性	兼容版本	相依性類型
代幣交換服務	^2.0.0	硬式

2.0.6

下表列出此元件 2.0.6 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	硬式
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
代幣交換服務	^2.0.0	硬式

2.0.5

下表列出此元件 2.0.5 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	硬式
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
代幣交換服務	^2.0.0	硬式

2.0.4

下表列出此元件 2.0.4 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	硬式
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
代幣交換服務	^2.0.0	硬式

2.0.3

下表列出此元件 2.0.3 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.3	硬式
发 Lambda 器	>=1.0.0	硬式
Lambda 執行階段	>=1.0.0	軟式
代幣交換服務	>=1.0.0	硬式

如需有關元件相依性的詳細資訊，請參閱元件方案參考。

組態

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

> **Note**
>
> 此元件的預設組態包括 Lambda 函數參數。建議您只編輯下列參數，以便在裝置上設定此元件。

v2.1.x

`lambdaParams`

包含此元件 Lambda 函數之參數的物件。此物件包含下列資訊：

`EnvironmentVariables`

包含 Lambda 函數參數的物件。此物件包含下列資訊：

`ModbusLocalPort`

核心設備上物理 Modbus 串口的絕對路徑，例如`/dev/ttyS2`。

若要在容器中執行此元件，您必須將此路徑定義為元件可存取的系統裝置 (在中`containerParams.devices`)。依預設，此元件會在容器中執行。

> **Note**
>
> 此組件必須具有對設備的讀/寫訪問權限。

`ModbusBaudRate`

(選擇性) 字串值，指定與本機 Modbus TCP 裝置進行序列通訊的傳輸速率。

預設：`9600`

`ModbusByteSize`

(選擇性) 字串值，指定與本機 Modbus TCP 裝置的序列通訊中的位元組大小。選擇`5`、`6`、`7`、或`8`位元。

預設：`8`

`ModbusParity`

(選擇性) 用來驗證與本機 Modbus TCP 裝置的串列通訊中資料完整性的同位檢查模式。

- `E`— 使用偶校驗驗證數據完整性。
- `O`— 使用奇偶校驗驗數據完整性。
- `N`— 不要驗證數據的完整性。

預設：`N`

ModbusStopBits

(選擇性) 字串值，指定在與本機 Modbus TCP 裝置進行序列通訊時指示位元組結尾的位元組數目。

預設：1

containerMode

(選擇性) 此元件的容器化模式。您可以從以下選項中選擇：

- GreengrassContainer— 元件在 AWS IoT Greengrass 容器內的隔離執行階段環境中執行。

  如果您指定此選項，您必須指定一個系統裝置 (incontainerParams.devices)，以便讓容器存取 Modbus 裝置。
- NoContainer— 元件不會在隔離的執行階段環境中執行。

預設：GreengrassContainer

containerParams

(選擇性) 包含此元件之容器參數的物件。如果您指定GreengrassContainer為，則元件會使用這些參數containerMode。

此物件包含下列資訊：

memorySize

(選擇性) 配置給元件的記憶體容量 (以 KB 為單位)。

預設值為 512 MB (525,312 KB)。

devices

(選擇性) 物件，指定元件可在容器中存取的系統裝置。

> **Important**
>
> 若要在容器中執行此元件，您必須指定在ModbusLocalPort環境變數中設定的系統裝置。

此物件包含下列資訊：

0-這是一個數組索引作為一個字符串。

包含下列資訊的物件：

path

核心裝置上系統裝置的路徑。這個值必須與您設定的值相同ModbusLocalPort。

permission

(選擇性) 從容器存取系統裝置的權限。此值必須是rw，指定元件具有系統裝置的讀取/寫入存取權。

預設：rw

addGroupOwner

(選擇性) 是否要新增以系統裝置擁有者身分執行元件的系統群組。

預設：true

pubsubTopics

(選擇性) 包含元件訂閱以接收訊息之主題的物件。您可以指定每個主題，以及元件是否訂閱 MQTT 主題 AWS IoT Core 或本機發佈/訂閱主題。

此物件包含下列資訊：

0-這是一個數組索引作為一個字符串。

包含下列資訊的物件：

type

(選擇性) 此元件用來訂閱訊息的發佈/訂閱訊息類型。您可以從以下選項中選擇：

- PUB_SUB - 訂閱本機發佈/訂閱訊息。如果您選擇此選項，則主題不能包含 MQTT 萬用字元。如需指定此選項時如何從自訂元件傳送郵件的詳細資訊，請參閱發佈/訂閱本地訊息。
- IOT_CORE— 訂閱 AWS IoT Core MQTT 訊息。如果您選擇此選項，主題可以包含 MQTT 萬用字元。如需指定此選項時如何從自訂元件傳送郵件的詳細資訊，請參閱發布/訂閱MQTT 訊 AWS IoT Core 息。

預設：PUB_SUB

`topic`

(選擇性) 元件訂閱接收訊息的主題。如果您指定`IotCore`為`type`，您可以在本主題中使用 MQTT 萬用字元 (+和#)。

Example 範例：組態合併更新 (容器模式)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "ModbusLocalPort": "/dev/ttyS2"
    }
  },
  "containerMode": "GreengrassContainer",
  "containerParams": {
    "devices": {
      "0": {
        "path": "/dev/ttyS2",
        "permission": "rw",
        "addGroupOwner": true
      }
    }
  }
}
```

Example 範例：組態合併更新 (無容器模式)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "ModbusLocalPort": "/dev/ttyS2"
    }
  },
  "containerMode": "NoContainer"
}
```

v2.0.x

`lambdaParams`

包含此元件 Lambda 函數之參數的物件。此物件包含下列資訊：

`EnvironmentVariables`

包含 Lambda 函數參數的物件。此物件包含下列資訊：

`ModbusLocalPort`

核心設備上物理 Modbus 串口的絕對路徑，例如`/dev/ttyS2`。

若要在容器中執行此元件，您必須將此路徑定義為元件可存取的系統裝置 (在中`containerParams.devices`)。依預設，此元件會在容器中執行。

> **Note**
>
> 此組件必須具有對設備的讀/寫訪問權限。

`containerMode`

(選擇性) 此元件的容器化模式。您可以從以下選項中選擇：

- `GreengrassContainer`— 元件在 AWS IoT Greengrass 容器內的隔離執行階段環境中執行。

  如果您指定此選項，您必須指定一個系統裝置 (in`containerParams.devices`)，以便讓容器存取 Modbus 裝置。
- `NoContainer`— 元件不會在隔離的執行階段環境中執行。

預設：`GreengrassContainer`

`containerParams`

(選擇性) 包含此元件之容器參數的物件。如果您指定`GreengrassContainer`為，則元件會使用這些參數`containerMode`。

此物件包含下列資訊：

`memorySize`

(選擇性) 配置給元件的記憶體容量 (以 KB 為單位)。

預設值為 512 MB (525,312 KB)。

`devices`

(選擇性) 物件，指定元件可在容器中存取的系統裝置。

> **Important**
>
> 若要在容器中執行此元件，您必須指定在ModbusLocalPort環境變數中設定的系統裝置。

此物件包含下列資訊：

0-這是一個數組索引作為一個字符串。

包含下列資訊的物件：

path

核心裝置上系統裝置的路徑。這個值必須與您設定的值相同ModbusLocalPort。

permission

(選擇性) 從容器存取系統裝置的權限。此值必須是rw，指定元件具有系統裝置的讀取/寫入存取權。

預設：rw

addGroupOwner

(選擇性) 是否要新增以系統裝置擁有者身分執行元件的系統群組。

預設：true

pubsubTopics

(選擇性) 包含元件訂閱以接收訊息之主題的物件。您可以指定每個主題，以及元件是否訂閱 MQTT 主題 AWS IoT Core 或本機發佈/訂閱主題。

此物件包含下列資訊：

0-這是一個數組索引作為一個字符串。

包含下列資訊的物件：

type

(選擇性) 此元件用來訂閱訊息的發佈/訂閱訊息類型。您可以從以下選項中選擇：

- PUB_SUB - 訂閱本機發佈/訂閱訊息。如果您選擇此選項，則主題不能包含 MQTT 萬用字元。如需指定此選項時如何從自訂元件傳送郵件的詳細資訊，請參閱發佈/訂閱本地訊息。

- `IOT_CORE`— 訂閱 AWS IoT Core MQTT 訊息。如果您選擇此選項，主題可以包含 MQTT 萬用字元。如需指定此選項時如何從自訂元件傳送郵件的詳細資訊，請參閱發布/訂閱MQTT 訊 AWS IoT Core 息。

預設：`PUB_SUB`

`topic`

(選擇性) 元件訂閱接收訊息的主題。如果您指定`IotCore`為`type`，您可以在本主題中使用 MQTT 萬用字元 (+和#)。

Example 範例：組態合併更新 (容器模式)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "ModbusLocalPort": "/dev/ttyS2"
    }
  },
  "containerMode": "GreengrassContainer",
  "containerParams": {
    "devices": {
      "0": {
        "path": "/dev/ttyS2",
        "permission": "rw",
        "addGroupOwner": true
      }
    }
  }
}
```

Example 範例：組態合併更新 (無容器模式)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "ModbusLocalPort": "/dev/ttyS2"
    }
  },
  "containerMode": "NoContainer"
}
```

輸入資料

此元件接受下列主題的 Modbus RTU 要求參數，並將 Modbus RTU 要求傳送至裝置。根據預設，此元件會訂閱本機發佈/訂閱訊息。如需如何從自訂元件將訊息發佈至此元件的詳細資訊，請參閱發佈/訂閱本地訊息。

默認主題（本地發布/訂閱）：`modbus/adapter/request`

該消息接受以下屬性。輸入訊息必須為 JSON 格式。

`request`

要傳送的 Modbus RTU 要求的參數。

請求消息的形狀取決於它所代表的 Modbus RTU 請求的類型。所有請求都需要以下屬性。

類型：`object`包含下列資訊：

`operation`

要執行的作業名稱。例如，指定讀`ReadCoilsRequest`取 Modbus RTU 裝置上的線圈。如需支援作業的詳細資訊，請參閱Modbus RTU 請求和回應。

類型：`string`

`device`

請求的目標裝置。

此值必須是介於`0`和之間的整數`247`。

類型：`integer`

要包含在請求中的其他參數取決於操作。此組件處理循環冗餘檢查（CRC），以驗證數據請求為您。

> **Note**
>
> 如果您要求包含`address`屬性，則必須將其值指定為整數。例如 `"address": 1`。

`id`

請求的任意 ID。使用此屬性可將輸入要求對應至輸出回應。當您指定此屬性時，組件會將回應物件中的`id`屬性設定為此值。

類型：`string`

Example 範例輸入：讀取線圈請求

```
{
  "request": {
    "operation": "ReadCoilsRequest",
    "device": 1,
    "address": 1,
    "count": 1
  },
  "id": "MyRequest"
}
```

輸出資料

依預設，此元件會將回應發佈為下列 MQTT 主題的輸出資料。您必須將此主題指定為舊版訂閱路由器元件的組態`subject`中的。如需如何在自訂元件中訂閱有關此主題之訊息的詳細資訊，請參閱發布/訂閱MQTT 訊 AWS IoT Core 息。

預設主題 (AWS IoT Core MQTT)：`modbus/adapter/response`

響應消息的形狀取決於請求操作和響應狀態。如需範例，請參閱 範例請求和回應。

每個回應含有以下屬性：

`response`

來自 Modbus RTU 設備的響應。

類型：`object`包含下列資訊：

`status`

請求的狀態。狀態可以是下列其中一個值：

- `Success`— 請求有效，組件將請求發送到 Modbus RTU 網絡，Modbus RTU 網絡返回響應。
- `Exception`— 請求有效，組件將請求發送到 Modbus RTU 網絡，Modbus RTU 網絡返回異常。如需詳細資訊，請參閱 回應狀態：例外。
- `No Response`— 請求無效，元件在將請求傳送至 Modbus RTU 網路之前就抓住了錯誤。如需詳細資訊，請參閱 回應狀態：沒有回應。

`operation`

該組件請求的操作。

`device`

元件傳送要求的裝置。

`payload`

來自 Modbus RTU 設備的響應。如果`status`是`No Response`，則此物件僅包含具有錯誤描述的`error`屬性 (例如，`[Input/Output] No Response received from the remote unit`)。

`id`

請求的 ID，您可以使用它來識別哪個響應到哪個請求。

> **Note**
>
> 寫入操作的回應只是請求的回響。雖然寫入回應不包含有意義的資訊，但最好是檢查回應狀態以查看要求是否成功或失敗。

Example 範例輸出：成功

```
{
  "response" : {
    "status" : "success",
    "device": 1,
    "operation": "ReadCoilsRequest",
    "payload": {
      "function_code": 1,
      "bits": [1]
    }
  },
  "id" : "MyRequest"
}
```

Example 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" : "MyRequest"
}
```

如需更多範例，請參閱 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"
}
```

讀取控股寄存器

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"
}
```

讀取輸入寄存器

請求範例：

```
{
  "request": {
    "operation": "ReadInputRegistersRequest",
    "device": 1,
    "address": 1,
    "count": 1
  },
  "id": "TestRequest"
}
```

寫入單線圈

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"
}
```

寫單寄存器

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"
}
```

寫入多個寄存器

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"
}
```

遮罩寫入寄存器

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"
}
```

讀寫多個寄存器

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**
>
> 響應包括組件讀取的寄存器。

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 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"
}
```

本機記錄檔

此元件使用下列記錄檔。

```
/greengrass/v2/logs/aws.greengrass.Modbus.log
```

若要檢視此元件的記錄

- 在核心裝置上執行下列命令，即時檢視此元件的記錄檔。以 AWS IoT Greengrass 根資料夾的路徑取*/greengrass/v2*代。

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.Modbus.log
  ```

授權

此元件包括下列協力廠商軟體/授權：

- Pymodbus/BSD許可證
- 序列號 /BSD 許可證

此元件是根據 Greengrass 核心軟體授權合約發行的。

變更記錄

下表說明元件每個版本的變更。

版本	變更
2.1.8	版本更新 Greengrass 2.12.0 版本釋放。
2.1.7	版本更新 Greengrass 2.11.0 版本釋放。
2.1.6	版本更新了 Greengrass 2.10.0 版本。

版本	變更
2.1.5	錯誤修復和改進 • 修復了`ReadDiscreteInput` 操作問題。
2.1.4	版本更新 Greengrass 2.9.0 版本釋放。
2.1.3	版本更新 Greengrass 2.8.0 版本的版本。
2.1.2	版本更新了 Greengrass 核 2.7.0 版本釋放。
2.1.1	版本更新 Greengrass 2.6.0 版本發布。
2.1.0	新功能 • 新增`ModbusBaudRate` 、、和`ModbusStopBits` 選項 `ModbusByteSize` `ModbusParity` ，您可以指定這些選項來配置與 Modbus RTU 裝置的序列通訊。
2.0.8	版本更新了 Greengrass 核 2.5.0 版本。
2.0.7	版本更新 Greengrass 2.4.0 版本的版本。
2.0.6	版本更新了 Greengrass 核 2.3.0 版本。
2.0.5	版本更新了 Greengrass 核 2.2.0 版本。
2.0.4	版本更新 Greengrass 2.1.0 版本發布。
2.0.3	初始版本。

MQTT 大橋

MQTT 橋接器元件 (`aws.greengrass.clientdevices.mqtt.Bridge`) 會在用戶端裝置、本機 Greengrass 發佈/訂閱和之間轉送 MQTT 訊息。AWS IoT Core您可以使用此元件來處理來自自訂元件中用戶端裝置的 MQTT 訊息，並將用戶端裝置與. AWS 雲端

> **Note**
>
> 用戶端裝置是連線到 Greengrass 核心裝置以傳送 MQTT 訊息和資料進行處理的本機 IoT 裝置。如需詳細資訊，請參閱 與本機 IoT 裝置互動。

您可以使用此元件在下列訊息代理程式之間轉送訊息：

- 本機 MQTT — 本機 MQTT 代理程式可處理用戶端裝置與核心裝置之間的訊息。
- 本機發佈/訂閱 — 本機 Greengrass 訊息代理程式會處理核心裝置上元件之間的訊息。如需有關如何在 Greengrass 元件中與這些訊息互動的詳細資訊，請參閱。發佈/訂閱本地訊息
- AWS IoT Core— AWS IoT Core MQTT 代理程式可處理 IoT 裝置與AWS 雲端目標之間的訊息。如需有關如何在 Greengrass 元件中與這些訊息互動的詳細資訊，請參閱。發布/訂閱MQTT 訊 AWS IoT Core 息

  > **Note**
  >
  > 即使用戶端裝置使用 QoS 0 發佈和訂閱本機 MQTT 代理程式AWS IoT Core，MQTT 橋接器也會使用 QoS 1 來發佈和訂閱。因此，當您將 MQTT 訊息從本機 MQTT 代理程式上的用戶端裝置轉送至時，可能會發現額外的延遲。AWS IoT Core如需核心裝置上 MQTT 組態的詳細資訊，請參閱。設定 MQTT 逾時和快取設定

主題

- 版本
- Type
- 作業系統
- 要求
- 相依性
- 組態
- 本機記錄檔
- 變更記錄

版本

此元件具有下列版本：

- 2.3.x 版本
- 2.2.x 版本
- 2.1.x
- 2.0.x

Type

這個組件是一個插件組件（`aws.greengrass.plugin`）。Greengrass 核在與核相同的 Java 虛擬機（JVM）中運行此組件。當您在核心裝置上變更此元件的版本時，核心會重新啟動。

此組件使用與 Greengrass 核相同的日誌文件。如需詳細資訊，請參閱 監控AWS IoT Greengrass日誌。

如需詳細資訊，請參閱 元件類型。

作業系統

此元件可安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件具有下列需求：

- 如果您將核心裝置的 MQTT 代理程式元件設定為使用預設連接埠 8883 以外的連接埠，則必須使用 MQTT 橋接器 v2.1.0 或更新版本。將其設定為在代理程式運作的連接埠上進行連線。
- 支援 MQTT 橋接器元件在 VPC 中執行。

相依性

部署元件時，AWS IoT Greengrass也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的需求，才能成功部署元件。本節列出此元件之已發行版本的相依性，以及定義每個相依性之元件

版本的語意版本條件約束。您也可以在AWS IoT Greengrass主控台中檢視元件每個版本的相依性。在元件詳細資料頁面上，尋找 [相依性] 清單。

2.3.0

下表列出此元件 2.3.0 版的相依性。

相依性	兼容版本	相依性類型
用戶端裝置驗證	> = 2.2.0	硬式

2.2.5 and 2.2.6

下表列出此元件 2.2.5 和 2.2.6 版的相依性。

相依性	兼容版本	相依性類型
用戶端裝置驗證	> = 2.2.0	硬式

2.2.3 and 2.2.4

下表列出此元件 2.2.3 和 2.2.4 版的相依性。

相依性	兼容版本	相依性類型
用戶端裝置驗證	>=2.2.0	硬式

2.2.0 – 2.2.2

下表列出此元件 2.2.0 至 2.2.2 版的相依性。

相依性	兼容版本	相依性類型
用戶端裝置驗證	>=2.2.0	硬式

2.1.1

下表列出此元件 2.1.1 版的相依性。

相依性	兼容版本	相依性類型
用戶端裝置驗證	> = 2.0.0	硬式

2.0.0 to 2.1.0

下表列出此元件 2.0.0 到 2.1.0 版的相依性。

相依性	兼容版本	相依性類型
用戶端裝置驗證	> = 2.0.0	硬式

如需有關元件相依性的詳細資訊，請參閱元件方案參考。

組態

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

2.3.0

`mqttTopicMapping`

您要橋接的主題對映。此元件會訂閱來源主題上的訊息，並將其接收到的訊息發佈至目標主題。每個主題對映都會定義主題、來源類型和目標類型。

此物件包含下列資訊：

`topicMappingNameKey`

此主題對映的名稱。以可協助您識別此主題對應的名稱取代 *`topicMappingNameKey`*。

此物件包含下列資訊：

`topic`

主題或主題篩選器可在來源代理程式與目標代理程式之間建立橋接

您可以使用+和 # MQTT 主題萬用字元來轉送符合主題篩選器的所有主題的郵件。如需詳細資訊，請參閱AWS IoT Core開發人員指南中的 MQTT 主題。

> **Note**
>
> 若要將 MQTT 主題萬用字元與Pubsub來源代理程式搭配使用，您必須使用 v2.6.0 或更新版本的 Greengrass 核心元件。

`targetTopicPrefix`

當此元件轉送訊息時，要新增至目標主題的前置詞。

`source`

來源訊息代理程式。您可以從以下選項中選擇：

- `LocalMqtt`— 用戶端裝置通訊的本機 MQTT 代理程式。
- `Pubsub`— 本機 Greengrass 發佈/訂閱訊息代理程式。
- `IotCore`— AWS IoT Core MQTT 訊息代理程式。

> **Note**
>
> 即使用戶端裝置使用 QoS 0 發佈和訂閱本機 MQTT 代理程式AWS IoT Core，MQTT 橋接器也會使用 QoS 1 來發佈和訂閱。因此，當您將 MQTT 訊息從本機 MQTT 代理程式上的用戶端裝置轉送至時，可能會發現額外的延遲。AWS IoT Core如需核心裝置上 MQTT 組態的詳細資訊，請參閱。設定 MQTT 逾時和快取設定

`source`並且`target`必須是不同的。

`target`

目標訊息代理程式。您可以從以下選項中選擇：

- `LocalMqtt`— 用戶端裝置通訊的本機 MQTT 代理程式。
- `Pubsub`— 本機 Greengrass 發佈/訂閱訊息代理程式。
- `IotCore`— AWS IoT Core MQTT 訊息代理程式。

> **Note**
>
> 即使用戶端裝置使用 QoS 0 發佈和訂閱本機 MQTT 代理程式AWS IoT Core，MQTT 橋接器也會使用 QoS 1 來發佈和訂閱。因此，當您將 MQTT 訊息從本機 MQTT 代理程式上的用戶端裝置轉送至時，可能會發現額外的延遲。AWS IoT Core如需核心裝置上 MQTT 組態的詳細資訊，請參閱。設定 MQTT 逾時和快取設定

`source`並且`target`必須是不同的。

mqtt5 RouteOptions

(選擇性) 提供用於設定將訊息從來源主題橋接至目標主題的主題對應的選項。

此物件包含下列資訊：

`mqtt5 RouteOptionsNameKey`

主題對映的路由選項名稱。使用欄位中定義的RouteOptionsNameKey相符*`topicMappingName##`*取代 *`mqtt5`*。`mqttTopicMapping`

此物件包含下列資訊：

無本地

(選擇性) 啟用時，橋接器不會轉寄橋接器本身發佈之主題的郵件。使用它來防止循環，如下所示：

```
{
    "mqtt5RouteOptions": {
        "toIoTCore": {
            "noLocal": true
        }
    },
    "mqttTopicMapping": {
        "toIoTCore": {
            "topic": "device",
            "source": "LocalMqtt",
            "target": "IotCore"
        },
        "toLocal": {
            "topic": "device",
```

```
            "source": "IotCore",
            "target": "LocalMqtt"
        }
    }
}
```

noLocal僅支援所在的路source由LocalMqtt。

預設：false

retainAsPublished

(選擇性) 啟用時，橋接器轉寄的訊息會與針對該路由發佈至 Broker 的訊息具有相同的retain旗標。

retainAsPublished僅支援所在的路source由LocalMqtt。

預設：false

mqtt

(選擇性) 用於與本機代理程式通訊的 MQTT 通訊協定設定。

version

(選擇性) 橋接器用來與本機代理程式通訊的 MQTT 通訊協定版本。必須與核心組態中選取的 MQTT 版本相同。

請選擇下列項目：

- mqtt3
- mqtt5

當mqttTopicMapping物件的source或target欄位設定為時，您必須部署 MQTT 代理程式。LocalMqtt如果您選擇該mqtt5選項，則必須使用MQTT 5 經紀商.

預設：mqtt3

ackTimeoutSeconds

(選擇性) 作業失敗之前，等待 PUBACK、SUBACK 或 UNSUBACK 封包的時間間隔。

預設：60

connAckTimeout女士

(選擇性) 在關閉連線之前等待 CONNACK 封包的時間間隔。

預設值：二十秒

pingTimeoutMs

(選擇性) 橋接器等待從本機代理程式接收 PINGACK 訊息的時間量 (毫秒)。如果等待超過逾時，橋接器會關閉，然後重新開啟 MQTT 連線。此值必須小於`keepAliveTimeoutSeconds`。

預設值：3 萬 (30 秒)

keepAliveTimeout秒

(選擇性) 橋接器傳送以維持 MQTT 連線作用中的每個 PING 訊息之間的時間 (以秒為單位)。此值必須大於`pingTimeoutMs`。

預設：60

maxReconnectDelay女士

(選擇性) MQTT 重新連線的時間上限 (以秒為單位)。

預設值：3 萬 (30 秒)

minReconnectDelay女士

(選擇性) MQTT 重新連線的時間下限 (以秒為單位)。

接收最大值

(選擇性) 橋接器可傳送的未確認 QoS1 封包數目上限。

預設：100

maximumPacketSize

用戶端接受 MQTT 封包的最大位元組數。

預設值：空值 (無限制)

sessionExpiryInterval

(選擇性) 您可以要求在橋接器和本機代理程式之間持續工作階段的時間 (以秒為單位)。

預設值：4294967295 (工作階段永遠不會過期)

`brokerUri`

(選擇性) 本機 MQTT 代理程式的 URI。如果您將 MQTT 代理程式設定為使用與預設連接埠 8883 不同的連接埠，則必須指定此參數。`########## MQTT ###############ssl://localhost:port`

預設：`ssl://localhost:8883`

`startupTimeoutSeconds`

(選擇性) 元件啟動的時間上限 (以秒為單位)。`BROKEN`如果超過此逾時，組件的狀態會變更為。

預設：`120`

Example 範例：組態合併更新

下列範例組態更新會指定下列項目：

- 將訊息從用戶端裝置轉送至AWS IoT Core符合主`clients/+/hello/world`題篩選器的主題。
- 在符合主題篩選器的主題上，將訊息從用戶端裝置轉送至本機發佈/訂閱，並將前置`events/input/`詞新增至目標主`clients/+/detections`題。產生的目標主題符合主`events/input/clients/+/detections`題篩選器。
- 將訊息從用戶端裝置轉送至AWS IoT Core符合主`clients/+/status`題篩選器的主題，並將前置`$aws/rules/StatusUpdateRule/`詞新增至目標主題。此範例會將這些訊息直接轉送至名為的AWS IoT規則，`StatusUpdateRule`以使用基本擷取降低成本。

```
{
  "mqttTopicMapping": {
    "ClientDeviceHelloWorld": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "ClientDeviceEvents": {
      "topic": "clients/+/detections",
      "targetTopicPrefix": "events/input/",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ClientDeviceCloudStatusUpdate": {
```

```
      "topic": "clients/+/status",
      "targetTopicPrefix": "$aws/rules/StatusUpdateRule/",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

Example 範例：配置 MQTT 5

下列範例組態會更新下列項目：

- 啟用橋接器與本機代理程式搭配使用 MQTT 5 通訊協定。
- 針對主題對應設定 MQTT 保留為已發佈設定。`ClientDeviceHelloWorld`

```
{
  "mqttTopicMapping": {
    "ClientDeviceHelloWorld": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  },
  "mqtt5RouteOptions": {
    "ClientDeviceHelloWorld": {
      "retainAsPublished": true
    }
  },
  "mqtt": {
    "version": "mqtt5"
  }
}
```

2.2.6

`mqttTopicMapping`

您要橋接的主題對映。此元件會訂閱來源主題上的訊息，並將其接收到的訊息發佈至目標主題。每個主題對映都會定義主題、來源類型和目標類型。

此物件包含下列資訊：

topicMappingNameKey

此主題對映的名稱。以可協助您識別此主題對應的名稱取代 *topicMappingNameKey*。

此物件包含下列資訊：

topic

主題或主題篩選器可在來源代理程式與目標代理程式之間建立橋接

您可以使用+和 # MQTT 主題萬用字元來轉送符合主題篩選器的所有主題的郵件。如需詳細資訊，請參閱AWS IoT Core開發人員指南中的 MQTT 主題。

> **Note**
>
> 若要將 MQTT 主題萬用字元與Pubsub來源代理程式搭配使用，您必須使用 v2.6.0 或更新版本的 Greengrass 核心元件。

targetTopicPrefix

當此元件轉送訊息時，要新增至目標主題的前置詞。

source

來源訊息代理程式。您可以從以下選項中選擇：

- LocalMqtt— 用戶端裝置通訊的本機 MQTT 代理程式。
- Pubsub— 本機 Greengrass 發佈/訂閱訊息代理程式。
- IotCore— AWS IoT Core MQTT 訊息代理程式。

> **Note**
>
> 即使用戶端裝置使用 QoS 0 發佈和訂閱本機 MQTT 代理程式AWS IoT Core，MQTT 橋接器也會使用 QoS 1 來發佈和訂閱。因此，當您將 MQTT 訊息從本機 MQTT 代理程式上的用戶端裝置轉送至時，可能會發現額外的延遲。AWS IoT Core如需核心裝置上 MQTT 組態的詳細資訊，請參閱。設定 MQTT 逾時和快取設定

source並且target必須是不同的。

`target`

目標訊息代理程式。您可以從以下選項中選擇：

- `LocalMqtt`— 用戶端裝置通訊的本機 MQTT 代理程式。
- `Pubsub`— 本機 Greengrass 發佈/訂閱訊息代理程式。
- `IotCore`— AWS IoT Core MQTT 訊息代理程式。

> **Note**
>
> 即使用戶端裝置使用 QoS 0 發佈和訂閱本機 MQTT 代理程式AWS IoT Core，MQTT 橋接器也會使用 QoS 1 來發佈和訂閱。因此，當您將 MQTT 訊息從本機 MQTT 代理程式上的用戶端裝置轉送至時，可能會發現額外的延遲。AWS IoT Core如需核心裝置上 MQTT 組態的詳細資訊，請參閱。設定 MQTT 逾時和快取設定

`source`並且`target`必須是不同的。

`brokerUri`

(選擇性) 本機 MQTT 代理程式的 URI。如果您將 MQTT 代理程式設定為使用與預設連接埠 8883 不同的連接埠，則必須指定此參數。`########## MQTT ###############`ssl://localhost:*port*

預設：`ssl://localhost:8883`

startupTimeoutSeconds

(選擇性) 元件啟動的時間上限 (以秒為單位)。BROKEN如果超過此逾時，組件的狀態會變更為。

預設：`120`

Example 範例：組態合併更新

下列範例組態更新會指定下列項目：

- 將訊息從用戶端裝置轉送至AWS IoT Core符合主`clients/+/hello/world`題篩選器的主題。
- 在符合主題篩選器的主題上，將訊息從用戶端裝置轉送至本機發佈/訂閱，並將前置`events/input/`詞新增至目標主`clients/+/detections`題。產生的目標主題符合主`events/input/clients/+/detections`題篩選器。

- 將訊息從用戶端裝置轉送至AWS IoT Core符合主clients/+/status題篩選器的主題，並將前置$aws/rules/StatusUpdateRule/詞新增至目標主題。此範例會將這些訊息直接轉送至名為的AWS IoT規則，StatusUpdateRule以使用基本擷取降低成本。

```
{
  "mqttTopicMapping": {
    "ClientDeviceHelloWorld": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "ClientDeviceEvents": {
      "topic": "clients/+/detections",
      "targetTopicPrefix": "events/input/",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ClientDeviceCloudStatusUpdate": {
      "topic": "clients/+/status",
      "targetTopicPrefix": "$aws/rules/StatusUpdateRule/",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

2.2.0 - 2.2.5

`mqttTopicMapping`

您要橋接的主題對映。此元件會訂閱來源主題上的訊息，並將其接收到的訊息發佈至目標主題。每個主題對映都會定義主題、來源類型和目標類型。

此物件包含下列資訊：

`topicMappingNameKey`

此主題對映的名稱。以可協助您識別此主題對應的名稱取代 *`topicMappingNameKey`*。

此物件包含下列資訊：

`topic`

主題或主題篩選器可在來源代理程式與目標代理程式之間建立橋接

您可以使用+和 # MQTT 主題萬用字元來轉送符合主題篩選器的所有主題的郵件。如需詳細資訊，請參閱AWS IoT Core開發人員指南中的 MQTT 主題。

> **Note**
>
> 若要將 MQTT 主題萬用字元與Pubsub來源代理程式搭配使用，您必須使用 v2.6.0 或更新版本的 Greengrass 核心元件。

`targetTopicPrefix`

當此元件轉送訊息時，要新增至目標主題的前置詞。

`source`

來源訊息代理程式。您可以從以下選項中選擇：

- `LocalMqtt`— 用戶端裝置通訊的本機 MQTT 代理程式。
- `Pubsub`— 本機 Greengrass 發佈/訂閱訊息代理程式。
- `IotCore`— AWS IoT Core MQTT 訊息代理程式。

> **Note**
>
> 即使用戶端裝置使用 QoS 0 發佈和訂閱本機 MQTT 代理程式AWS IoT Core，MQTT 橋接器也會使用 QoS 1 來發佈和訂閱。因此，當您將 MQTT 訊息從本機 MQTT 代理程式上的用戶端裝置轉送至時，可能會發現額外的延遲。AWS IoT Core如需核心裝置上 MQTT 組態的詳細資訊，請參閱。設定 MQTT 逾時和快取設定

`source`並且`target`必須是不同的。

`target`

目標訊息代理程式。您可以從以下選項中選擇：

- `LocalMqtt`— 用戶端裝置通訊的本機 MQTT 代理程式。
- `Pubsub`— 本機 Greengrass 發佈/訂閱訊息代理程式。
- `IotCore`— AWS IoT Core MQTT 訊息代理程式。

> **Note**
>
> 即使用戶端裝置使用 QoS 0 發佈和訂閱本機 MQTT 代理程式AWS IoT Core，MQTT 橋接器也會使用 QoS 1 來發佈和訂閱。因此，當您將 MQTT 訊息從本機 MQTT 代理程式上的用戶端裝置轉送至時，可能會發現額外的延遲。AWS IoT Core如需核心裝置上 MQTT 組態的詳細資訊，請參閱。設定 MQTT 逾時和快取設定

source並且target必須是不同的。

`brokerUri`

(選擇性) 本機 MQTT 代理程式的 URI。如果您將 MQTT 代理程式設定為使用與預設連接埠 8883 不同的連接埠，則必須指定此參數。`########## MQTT ################`ssl://localhost:*port*

預設：ssl://localhost:8883

Example 範例：組態合併更新

下列範例組態更新會指定下列項目：

- 將訊息從用戶端裝置轉送至AWS IoT Core符合主clients/+/hello/world題篩選器的主題。
- 在符合主題篩選器的主題上，將訊息從用戶端裝置轉送至本機發佈/訂閱，並將前置events/input/詞新增至目標主clients/+/detections題。產生的目標主題符合主events/input/clients/+/detections題篩選器。
- 將訊息從用戶端裝置轉送至AWS IoT Core符合主clients/+/status題篩選器的主題，並將前置$aws/rules/StatusUpdateRule/詞新增至目標主題。此範例會將這些訊息直接轉送至名為的AWS IoT規則，StatusUpdateRule以使用基本擷取降低成本。

```
{
  "mqttTopicMapping": {
    "ClientDeviceHelloWorld": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "ClientDeviceEvents": {
```

```
      "topic": "clients/+/detections",
      "targetTopicPrefix": "events/input/",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ClientDeviceCloudStatusUpdate": {
      "topic": "clients/+/status",
      "targetTopicPrefix": "$aws/rules/StatusUpdateRule/",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

2.1.x

`mqttTopicMapping`

您要橋接的主題對映。此元件會訂閱來源主題上的訊息，並將其接收到的訊息發佈至目標主題。每個主題對映都會定義主題、來源類型和目標類型。

此物件包含下列資訊：

`topicMappingNameKey`

此主題對映的名稱。以可協助您識別此主題對應的名稱取代 *`topicMappingNameKey`*。

此物件包含下列資訊：

`topic`

主題或主題篩選器可在來源代理程式與目標代理程式之間建立橋接

如果您指定`LocalMqtt`或`IotCore`來源代理程式，則可以使用+和 # MQTT 主題萬用字元來轉送符合主題篩選器的所有主題的訊息。如需詳細資訊，請參閱AWS IoT Core開發人員指南中的 MQTT 主題。

`source`

來源訊息代理程式。您可以從以下選項中選擇：

- `LocalMqtt`— 用戶端裝置通訊的本機 MQTT 代理程式。
- `Pubsub`— 本機 Greengrass 發佈/訂閱訊息代理程式。
- `IotCore`— AWS IoT Core MQTT 訊息代理程式。

> **Note**
>
> 即使用戶端裝置使用 QoS 0 發佈和訂閱本機 MQTT 代理程式AWS IoT Core，MQTT 橋接器也會使用 QoS 1 來發佈和訂閱。因此，當您將 MQTT 訊息從本機 MQTT 代理程式上的用戶端裝置轉送至時，可能會發現額外的延遲。AWS IoT Core如需核心裝置上 MQTT 組態的詳細資訊，請參閱。設定 MQTT 逾時和快取設定

`source`並且`target`必須是不同的。

`target`

目標訊息代理程式。您可以從以下選項中選擇：

- `LocalMqtt`— 用戶端裝置通訊的本機 MQTT 代理程式。
- `Pubsub`— 本機 Greengrass 發佈/訂閱訊息代理程式。
- `IotCore`— AWS IoT Core MQTT 訊息代理程式。

> **Note**
>
> 即使用戶端裝置使用 QoS 0 發佈和訂閱本機 MQTT 代理程式AWS IoT Core，MQTT 橋接器也會使用 QoS 1 來發佈和訂閱。因此，當您將 MQTT 訊息從本機 MQTT 代理程式上的用戶端裝置轉送至時，可能會發現額外的延遲。AWS IoT Core如需核心裝置上 MQTT 組態的詳細資訊，請參閱。設定 MQTT 逾時和快取設定

`source`並且`target`必須是不同的。

`brokerUri`

(選擇性) 本機 MQTT 代理程式的 URI。如果您將 MQTT 代理程式設定為使用與預設連接埠 8883 不同的連接埠，則必須指定此參數。`########## MQTT ###############ssl://localhost:port`

預設：`ssl://localhost:8883`

Example 範例：組態合併更新

下列範例組態更新指定將用戶端裝置的訊息轉送至AWS IoT Coreclients/MyClientDevice1/hello/world和clients/MyClientDevice2/hello/world主題。

```
{
  "mqttTopicMapping": {
    "ClientDevice1HelloWorld": {
      "topic": "clients/MyClientDevice1/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "ClientDevice2HelloWorld": {
      "topic": "clients/MyClientDevice2/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

2.0.x

`mqttTopicMapping`

您要橋接的主題對映。此元件會訂閱來源主題上的訊息，並將其接收到的訊息發佈至目標主題。每個主題對映都會定義主題、來源類型和目標類型。

此物件包含下列資訊：

`topicMappingNameKey`

此主題對映的名稱。以可協助您識別此主題對應的名稱取代 *`topicMappingNameKey`*。

此物件包含下列資訊：

`topic`

主題或主題篩選器可在來源代理程式與目標代理程式之間建立橋接

如果您指定`LocalMqtt`或`IotCore`來源代理程式，則可以使用+和 # MQTT 主題萬用字元來轉送符合主題篩選器的所有主題的訊息。如需詳細資訊，請參閱AWS IoT Core開發人員指南中的 MQTT 主題。

`source`

來源訊息代理程式。您可以從以下選項中選擇：

- `LocalMqtt`— 用戶端裝置通訊的本機 MQTT 代理程式。
- `Pubsub`— 本機 Greengrass 發佈/訂閱訊息代理程式。
- `IotCore`— AWS IoT Core MQTT 訊息代理程式。

> **Note**
>
> 即使用戶端裝置使用 QoS 0 發佈和訂閱本機 MQTT 代理程式AWS IoT Core，MQTT 橋接器也會使用 QoS 1 來發佈和訂閱。因此，當您將 MQTT 訊息從本機 MQTT 代理程式上的用戶端裝置轉送至時，可能會發現額外的延遲。AWS IoT Core如需核心裝置上 MQTT 組態的詳細資訊，請參閱。設定 MQTT 逾時和快取設定

`source`並且`target`必須是不同的。

`target`

目標訊息代理程式。您可以從以下選項中選擇：

- `LocalMqtt`— 用戶端裝置通訊的本機 MQTT 代理程式。
- `Pubsub`— 本機 Greengrass 發佈/訂閱訊息代理程式。
- `IotCore`— AWS IoT Core MQTT 訊息代理程式。

> **Note**
>
> 即使用戶端裝置使用 QoS 0 發佈和訂閱本機 MQTT 代理程式AWS IoT Core，MQTT 橋接器也會使用 QoS 1 來發佈和訂閱。因此，當您將 MQTT 訊息從本機 MQTT 代理程式上的用戶端裝置轉送至時，可能會發現額外的延遲。AWS IoT Core如需核心裝置上 MQTT 組態的詳細資訊，請參閱。設定 MQTT 逾時和快取設定

`source`並且`target`必須是不同的。

Example 範例：組態合併更新

下列範例組態更新指定將用戶端裝置的訊息轉送至AWS IoT Core`clients/MyClientDevice1/hello/world`和`clients/MyClientDevice2/hello/world`主題。

```
{
```

```
  "mqttTopicMapping": {
    "ClientDevice1HelloWorld": {
      "topic": "clients/MyClientDevice1/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "ClientDevice2HelloWorld": {
      "topic": "clients/MyClientDevice2/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

本機記錄檔

此元件使用與 Greengrass 核元件相同的記錄檔。

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

若要檢視此元件的記錄

- 在核心裝置上執行下列命令，即時檢視此元件的記錄檔。以AWS IoT Greengrass根資料夾的路徑取代*/greengrass/v2*或 *C:\greengrass\v2*。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

變更記錄

下表說明元件每個版本中的變更。

版本	變更
2.3.1	錯誤修復和改進 修正本機 MQTT 用戶端進入中斷連線迴圈的問題。
2.3.0	新功能 新增 MQTT5 支援，以便在 MQTT 來源AWS IoT Core與本地 MQTT 來源之間進行橋接。
2.2.6	新功能 添加一個新的`startupTimeoutSeconds` 配置選項。
2.2.5	版本更新了客戶端設備身份驗證版本 2.4.0 版本。
2.2.4	版本已針對 Greengrass 客戶端設備身份驗證 2.3.0 版本更新。
2.2.3	此版本包含錯誤修復和改進。
2.2.2	錯誤修復和改進 • 記錄調整。
2.2.1	錯誤修復和改進 修正了可能導致 MQTT 橋接器無法訂閱 MQTT 主題的問題。
2.2.0	新功能 • 當您指定本機發佈/訂閱做為來源訊息代理程式時，新增對 MQTT 主題萬用字元 (#和+) 的支援。 此功能需要 v2.6.0 或更高版 Greeng rass 核成分。 • 新增選`targetTopicPrefix` 項，您可以指定該選項來設定 MQTT 橋接器，以便在轉送郵件時將首碼新增至目標主題。

版本	變更
2.1.1	錯誤修復和改進 • 修正此元件如何處理組態重設更新的問題。 • 當憑證旋轉時，降低 MQTT 用戶端中斷連線的頻率。
2.1.0	新功能 • 新增`brokerUri` 參數，可讓您使用非預設 MQTT 代理程式連接埠。
2.0.1	此版本包括錯誤修復和改進。
2.0.0	初始版本。

MQTT 3.1.1 經紀商 (平均)

Moquette MQTT 代理程式元件 (`aws.greengrass.clientdevices.mqtt.Moquette`) 可處理用戶端裝置與 Greengrass 核心裝置之間的 MQTT 訊息。此元件提供 Moquette MQ TT 代理程式的修改版本。部署此 MQTT 代理程式以執行輕量型 MQTT 代理程式。如需如何選擇 MQTT 代理程式的詳細資訊，請參閱。選擇一個 MQTT 經紀商

該代理商實現了 MQTT 3.1.1 協議。它包含 QoS 0、QoS 1、QoS 2 保留訊息、最後將訊息和持續工作階段的支援。

> **Note**
>
> 用戶端裝置是連線到 Greengrass 核心裝置以傳送 MQTT 訊息和資料進行處理的本機 IoT 裝置。如需詳細資訊，請參閱 與本機 IoT 裝置互動。

主題

- 版本
- Type
- 作業系統
- 要求
- 相依性
- 組態

- 本機記錄檔
- 變更記錄

版本

此元件具有下列版本：

- 2.3.x 版本
- 2.2.x 版本
- 2.1.x
- 2.0.x

Type

這個組件是一個插件組件（`aws.greengrass.plugin`）。Greengrass 核在與核相同的 Java 虛擬機（JVM）中運行此組件。當您在核心裝置上變更此元件的版本時，核心會重新啟動。

此組件使用與 Greengrass 核相同的日誌文件。如需詳細資訊，請參閱 監控AWS IoT Greengrass日誌。

如需詳細資訊，請參閱 元件類型。

作業系統

此元件可安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件具有下列需求：

- 核心裝置必須能夠在 MQTT 代理程式運作的連接埠上接受連線。依預設，此元件會在連接埠 8883 上執行 MQTT 代理程式。您可以在設定此元件時指定不同的連接埠。

  如果您指定不同的連接埠，並使用 MQTT 橋接器元件將 MQTT 訊息轉送給其他代理程式，則必須使用 MQTT 橋接器 v2.1.0 或更新版本。將其設定為使用 MQTT 代理程式運作的連接埠。

如果您指定不同的連接埠，並使用 IP 偵測器元件來管理 MQTT 代理程式端點，則必須使用 IP 偵測器 v2.1.0 或更新版本。將其設定為報告 MQTT 代理程式運作的連接埠。

- 系統支援 MQTT 代理程式元件在 VPC 中執行。

相依性

部署元件時， AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的需求，才能成功部署元件。本節列出此元件之已發行版本的相依性，以及定義每個相依性之元件版本的語意版本條件約束。您也可以在AWS IoT Greengrass 主控台中檢視元件每個版本的相依性。在元件詳細資料頁面上，尋找 [相依性] 清單。

2.3.2 – 2.3.6

下表列出此元件 2.3.2 到 2.3.6 版的相依性。

相依性	兼容版本	相依性類型
用戶端裝置驗證	>=2.2.0	硬式

2.3.0 and 2.3.1

下表列出此元件 2.3.0 和 2.3.1 版的相依性。

相依性	兼容版本	相依性類型
用戶端裝置驗證	>=2.2.0	硬式

2.2.0

下表列出此元件 2.2.0 版的相依性。

相依性	兼容版本	相依性類型
用戶端裝置驗證	>=2.2.0	硬式

2.1.0

下表列出此元件 2.1.0 版的相依性。

相依性	兼容版本	相依性類型
用戶端裝置驗證	> = 2.0.0	硬式

2.0.0 - 2.0.2

下表列出此元件 2.0.0 到 2.0.2 版本的相依性。

相依性	兼容版本	相依性類型
用戶端裝置驗證	> = 2.0.0	硬式

如需有關元件相依性的詳細資訊，請參閱元件方案參考。

組態

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

`moquette`

(選擇性) 要使用的模型 MQTT 代理程式組態。您可以在此元件中配置 Moqeutte 配置選項的子集。如需詳細資訊，請參閱 Moquette 組態檔案中的內嵌註解。

此物件包含下列資訊：

`ssl_port`

(選擇性) MQTT 代理程式運作的連接埠。

> **Note**
>
> 如果您指定不同的連接埠，並使用 MQTT 橋接器元件將 MQTT 訊息轉送給其他代理程式，則必須使用 MQTT 橋接器 v2.1.0 或更新版本。將其設定為使用 MQTT 代理程式運作的連接埠。

如果您指定不同的連接埠，並使用 IP 偵測器元件來管理 MQTT 代理程式端點，則必須使用 IP 偵測器 v2.1.0 或更新版本。將其設定為報告 MQTT 代理程式運作的連接埠。

預設：8883

host

(選擇性) MQTT 代理程式繫結的介面。例如，您可以變更此參數，以便 MQTT 代理程式僅繫結至特定的區域網路。

預設值：0.0.0.0(繫結至所有網路介面)

startupTimeoutSeconds

(選擇性) 元件啟動的時間上限 (以秒為單位)。BROKEN如果超過此逾時，組件的狀態會變更為。

預設：120

Example 範例：組態合併更新

下列範例組態指定在連接埠 443 上操作 MQTT 代理程式。

```
{
  "moquette": {
    "ssl_port": "443"
  }
}
```

本機記錄檔

此元件使用與 Greengrass 核元件相同的記錄檔。

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

若要檢視此元件的記錄

- 在核心裝置上執行下列命令，即時檢視此元件的記錄檔。以 AWS IoT Greengrass 根資料夾的路徑取代*/greengrass/v2*或 *C:\greengrass\v2*。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

變更記錄

下表說明元件每個版本中的變更。

版本	變更
2.3.6	錯誤修復和改進 • 一般錯誤修正與改進。
2.3.5	錯誤修復和改進 • 更新莫奎特到 0.17 版本。
2.3.4	錯誤修復和改進 • 修正用戶端在傳送或接收訊息時，可能會因為用戶端 ID 重複而遇到無效工作階段錯誤的問題。這個問題造成用戶端的工作階段關閉。
2.3.3	新功能 添加一個新的`startupTimeoutSeconds` 配置選項。
2.3.2	版本更新了客戶端設備身份驗證版本 2.4.0 版本。
2.3.1	錯誤修復和改進 • 修正了一個競爭狀況，在嘗試重新連線後，用戶端可能會因為工作階段無效而中斷連線。

版本	變更
2.3.0	添加對證書鏈的支持。
2.2.0	版本更新了客戶端設備身份驗證版本 2.2.0 版本。
2.1.0	錯誤修復和改進 • 更新此組件以使用 Moquette 版本 0.16，這可以提高性能並包括其他一些改進。 • 修正本機 MQTT 伺服器憑證在特定情況下旋轉頻率高於預期的問題。 若要套用此修正程式，您還必須使用 v2.1.0 或更新版本的用戶端裝置驗證元件。
2.0.2	錯誤修復和改進 • 將 MQTT 訊息大小上限從 8,092 位元組增加至 128 KB。由於郵件大小限制包含郵件標頭，因此有效的 MQTT 訊息承載限制略小。 • 添加對參數中整`ssl_port`數值的支援。
2.0.1	版本更新 Greengrass 2.4.0 版本的版本。
2.0.0	初始版本。

MQTT 5 經紀商

EMQX MQTT 代理組件（`aws.greengrass.clientdevices.mqtt.EMQX`）處理客戶端設備和 Greengrass 核心設備之間的 MQTT 消息。此元件提供 EMQX MQ TT 5.0 代理程式的修改版本。部署此 MQTT 代理程式，以便在用戶端裝置與核心裝置之間的通訊中使用 MQTT 5 功能。如需如何選擇 MQTT 代理程式的詳細資訊，請參閱。選擇一個 MQTT 經紀商

此代理程式會實作 MQTT 5.0 通訊協定。它包括對工作階段和訊息到期間隔、使用者屬性、共用訂閱、主題別名等的支援。MQTT 5 與 MQTT 3.1.1 向後兼容，因此，如果您運行 Moquette MQTT 3.1.1 代理商，則可以將其替換為 EMQX MQTT 5 代理商，並且客戶端設備可以繼續連接並像往常一樣運行。

> **Note**
>
> 用戶端裝置是連線到 Greengrass 核心裝置以傳送 MQTT 訊息和資料進行處理的本機 IoT 裝置。如需詳細資訊，請參閱 與本機 IoT 裝置互動。

主題

版本

此元件具有下列版本：

- 2.0.x
- 1.2.x
- 1.x
- 1.0.x

Type

此元件是一般元件 (`aws.greengrass.generic`)。Greengrass 核會執行元件的生命週期指令碼。

如需詳細資訊，請參閱 元件類型。

作業系統

此元件可安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件具有下列需求：

- 核心裝置必須能夠在 MQTT 代理程式運作的連接埠上接受連線。依預設，此元件會在連接埠 8883 上執行 MQTT 代理程式。您可以在設定此元件時指定不同的連接埠。

  如果您指定不同的連接埠，並使用 MQTT 橋接器元件將 MQTT 訊息轉送給其他代理程式，則必須使用 MQTT 橋接器 v2.1.0 或更新版本。將其設定為使用 MQTT 代理程式運作的連接埠。

  如果您指定不同的連接埠，並使用 IP 偵測器元件來管理 MQTT 代理程式端點，則必須使用 IP 偵測器 v2.1.0 或更新版本。將其設定為報告 MQTT 代理程式運作的連接埠。
- 在 Linux 核心設備上，在核心設備上安裝和配置 Docker：
  - 碼頭引擎 1.9.1 或更高版本安裝在 Greengrass 核心設備上。版本 20.10 是經過驗證可與 AWS IoT Greengrass 核心軟件配合使用的最新版本。在部署執行 Docker 容器的元件之前，您必須直接在核心裝置上安裝 Docker。
  - 在您部署此元件之前，Docker 精靈會在核心裝置上啟動並執行。
  - 執行此元件的系統使用者必須具有 root 或管理員權限。或者，您可以在`docker`群組中以系統使用者的身分執行此元件，並將`false`此元件的`requiresPrivileges`選項設定為在沒有權限的情況下執行 EQMX MQTT 代理程式。
- 支援 EMQX MQTT 代理程式元件在 VPC 中執行。
- 平台上不支援 EMQX MQTT 代理程式元件。`armv7`

相依性

部署元件時， AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的需求，才能成功部署元件。本節列出此元件之已發行版本的相依性，以及定義每個相依性之元件版本的語意版本條件約束。您也可以在AWS IoT Greengrass 主控台中檢視元件每個版本的相依性。在元件詳細資料頁面上，尋找 [相依性] 清單。

2.0.0

下表列出此元件 2.0.0 版的相依性。

相依性	兼容版本	相依性類型
用戶端裝置驗證	>=2.2.0	硬式

1.2.2 – 1.2.3

下表列出此元件 1.2.2 至 1.2.3 版的相依性。

相依性	兼容版本	相依性類型
用戶端裝置驗證	>=2.2.0	硬式

1.2.0 and 1.2.1

下表列出此元件 1.2.0 和 1.2.1 版的相依性。

相依性	兼容版本	相依性類型
用戶端裝置驗證	>=2.2.0	硬式

1.0.0 and 1.1.0

下表列出此元件 1.0.0 和 1.1.0 版的相依性。

相依性	兼容版本	相依性類型
用戶端裝置驗證	>=2.2.0	硬式

如需有關元件相依性的詳細資訊，請參閱元件方案參考。

組態

2.0.0

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

> **Important**
>
> 如果您使用 MQTT 5 代理程式 (EMQX) 元件的第 2 版，則必須更新組態檔案。版本 1 設定檔不適用於版本 2。

EMQX 配置

(選擇性) 要使用的 EMQX MQTT 代理程式組態。您可以在此元件中設定 EMQX 組態選項。

當您使用 EMQX 代理程式時，Greengrass 會使用預設設定。除非您使用此欄位進行修改，否則會使用此組態。

修改下列組態設定會導致 EMQX 代理程式元件重新啟動。其他組態變更會套用而不重新啟動元件。

- `emqxConfig/cluster`
- `emqxConfig/node`
- `emqxConfig/rpc`

> **Note**
>
> `aws.greengrass.clientdevices.mqtt.EMQX`可讓您設定敏感安全性選項。其中包括 TLS 設定、驗證和授權提供者。我們建議使用相互 TLS 驗證的預設設定，以及 Greengrass 用戶端裝置驗證提供者。

Example 範例：預設組態

下列範例顯示為 MQTT 5 (EMQX) 代理程式設定的預設值。您可以使用組態設定覆寫這些`emqxConfig`設定。

```
{
  "authorization": {
    "no_match": "deny",
    "sources": []
  },
  "node": {
    "cookie": "<placeholder>"
  },
```

```
  "listeners": {
     "ssl": {
       "default": {
         "ssl_options": {
           "keyfile": "{work:path}\\data\\key.pem",
           "certfile": "{work:path}\\data\\cert.pem",
           "cacertfile": null,
           "verify": "verify_peer",
           "versions": ["tlsv1.3", "tlsv1.2"],
           "fail_if_no_peer_cert": true
         }
       }
     },
     "tcp": {
       "default": {
         "enabled": false
       }
     },
     "ws": {
       "default": {
         "enabled": false
       }
     },
     "wss": {
       "default": {
         "enabled": false
       }
     }
  },
  "plugins": {
    "states": [{"name_vsn": "gg-1.0.0", "enable": true}],
    "install_dir": "plugins"
  }
}
```

认证模式

(選擇性) 設定代理程式的授權提供者。可以是下列其中一個值：

- `enabled`— (預設) 使用 Greengrass 驗證和授權提供者。
- `bypass_on_failure`— 如果 Greengrass 拒絕驗證或授權，請使用 Greengrass 驗證提供者，然後使用 EMQX 提供者鏈中任何剩餘的驗證提供者。
- `bypass`— 格 Greengrass 提供者已停用。身份驗證和授權由 EMQX 提供商鏈處理。

`requiresPrivilege`

(選擇性) 在 Linux 核心裝置上，您可以指定在沒有 root 權限或系統管理員權限的情況下執行 EMQX MQTT 代理程式。如果將此選項設定為`false`，則執行此元件的系統使用者必須是`docker`群組的成員。

預設：`true`

`startupTimeoutSeconds`

(選擇性) EMQX MQTT 代理程式啟動的時間上限 (以秒為單位)。`BROKEN`如果超過此逾時，組件的狀態會變更為。

預設：`90`

`ipcTimeoutSeconds`

(選擇性) 元件等待 Greengrass 核心回應處理序間通訊 (IPC) 要求的時間上限 (以秒為單位)。如果此元件在檢查用戶端裝置是否獲得授權時報告逾時錯誤，請增加此數字。

預設：`5`

`crtLogLevel`

(選擇性)「一 AWS 般執行階段」(CRT) 程式庫的記錄層級。

預設值為 EMQX MQTT 代理程式記錄層級 (在中)。`log.level` `emqx`

`restartIdentifier`

(選擇性) 設定此選項以重新啟動 EMQX MQTT 代理程式。當此組態值變更時，此元件會重新啟動 MQTT 代理程式。您可以使用此選項強制用戶端裝置中斷連線。

`dockerOptions`

(選擇性) 僅在 Linux 作業系統上設定此選項，以將參數新增至 Docker 命令列。例如，若要對應其他連接埠，請使用 `-p` Docker 參數：

```
"-p 1883:1883"
```

Example 範例：將 v1.x 組態檔案更新為 v2.x

下列範例顯示將 v1.x 組態檔更新為 2.x 版所需的變更。

版本 1.x 配置文件：

```
{
    "emqx": {
        "listener.ssl.external": "443",
        "listener.ssl.external.max_connections": "1024000",
        "listener.ssl.external.max_conn_rate": "500",
        "listener.ssl.external.rate_limit": "50KB,5s",
        "listener.ssl.external.handshake_timeout": "15s",
        "log.level": "warning"
    },
    "mergeConfigurationFiles": {
        "etc/plugins/aws_greengrass_emqx_auth.conf": "auth_mode=enabled\n
 use_greengrass_managed_certificates=true\n"
    }
}
```

v2 的對等配置文件：

```
{
    "emqxConfig": {
        "listeners": {
            "ssl": {
                "default": {
                   "bind": "8883",
                   "max_connections": "1024000",
                   "max_conn_rate": "500",
                   "handshake_timeout": "15s"
                }
            }
        },
        "log": {
            "console": {
              "enable": true,
              "level": "warning"
            }
        }
    },
    "authMode": "enabled"
}
```

沒有等同於`listener.ssl.external.rate_limit`組態項目。
組`use_greengrass_managed_certificates`態選項已移除。

Example 範例：為代理程式設定新的連接埠

下列範例會將 MQTT 代理程式運作的連接埠，從預設的 8883 變更為連接埠 1234。如果您使用的是 Linux，請包括dockerOptions欄位。

```
{
  "emqxConfig": {
    "listeners": {
      "ssl": {
        "default": {
          "bind": 1234
        }
      }
    }
  },
  "dockerOptions": "-p 1234:1234"
}
```

Example 範例：調整 MQTT 代理程式的記錄層級

下列範例會將 MQTT 代理程式的記錄層級變更為。debug您可以從下列記錄層級中選擇：

- debug
- info
- notice
- warning
- error
- critical
- alert
- emergency

預設記錄層級為warning。

```
{
  "emqxConfig": {
    "log": {
      "console": {
         "level": "debug"
      }
```

```
    }
  }
}
```

Example 範例：啟用 EMQX 儀表板

以下示例啟用 EMQX 儀表板，以便您可以監視和管理經紀人。如果您使用的是 Linux，請包括`dockerOptions`欄位。

```
{
  "emqxConfig": {
    "dashboard": {
      "listeners": {
        "http": {
          "bind": 18083
        }
      }
    }
  },
  "dockerOptions": "-p 18083:18083"
}
```

1.0.0 - 1.2.2

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

`emqx`

(選擇性) 要使用的 EMQX MQTT 代理程式組態。您可以在此元件中設定 EMQX 組態選項的子集。

此物件包含下列資訊：

`listener.ssl.external`

(選擇性) MQTT 代理程式運作的連接埠。

> **Note**
>
> 如果您指定不同的連接埠，並使用 MQTT 橋接器元件將 MQTT 訊息轉送給其他代理程式，則必須使用 MQTT 橋接器 v2.1.0 或更新版本。將其設定為使用 MQTT 代理程式運作的連接埠。

如果您指定不同的連接埠，並使用 IP 偵測器元件來管理 MQTT 代理程式端點，則必須使用 IP 偵測器 v2.1.0 或更新版本。將其設定為報告 MQTT 代理程式運作的連接埠。

預設：8883

`listener.ssl.external.max_connections`

(選擇性) MQTT 代理程式支援的同時連線數目上限。

預設：1024000

`listener.ssl.external.max_conn_rate`

(選擇性) MQTT 代理程式每秒可接收的新連線數目上限。

預設：500

`listener.ssl.external.rate_limit`

(選擇性) 所有連線至 MQTT 代理程式的頻寬限制。指定該頻寬的頻寬和持續時間，並以逗號 (,) 分隔，格式如下：`bandwidth,duration`例如，您可以指`50KB,5s`定每 5 秒將 MQTT 代理程式限制為 50 KB 的資料。

`listener.ssl.external.handshake_timeout`

(選擇性) MQTT 代理程式等待完成驗證新連線的時間量。

預設：15s

`mqtt.max_packet_size`

(選擇性) MQTT 郵件的大小上限。

預設值：268435455(256 MB 減去 1)

`log.level`

(選擇性) MQTT 代理程式的記錄層級。您可以從以下選項中選擇：

- `debug`
- `info`
- `notice`
- `warning`

- error
- critical
- alert
- emergency

預設記錄層級為warning。

requiresPrivilege

(選擇性) 在 Linux 核心裝置上，您可以指定在沒有 root 權限或系統管理員權限的情況下執行 EMQX MQTT 代理程式。如果將此選項設定為false，則執行此元件的系統使用者必須是docker群組的成員。

預設：true

startupTimeoutSeconds

(選擇性) EMQX MQTT 代理程式啟動的時間上限 (以秒為單位)。BROKEN如果超過此逾時，組件的狀態會變更為。

預設：90

ipcTimeoutSeconds

(選擇性) 元件等待 Greengrass 核心回應處理序間通訊 (IPC) 要求的時間上限 (以秒為單位)。如果此元件在檢查用戶端裝置是否獲得授權時報告逾時錯誤，請增加此數字。

預設：5

crtLogLevel

(選擇性)「一 AWS 般執行階段」(CRT) 程式庫的記錄層級。

預設值為 EMQX MQTT 代理程式記錄層級 (在中)。log.level emqx

restartIdentifier

(選擇性) 設定此選項以重新啟動 EMQX MQTT 代理程式。當此組態值變更時，此元件會重新啟動 MQTT 代理程式。您可以使用此選項強制用戶端裝置中斷連線。

dockerOptions

(選擇性) 僅在 Linux 作業系統上設定此選項，以將參數新增至 Docker 命令列。例如，若要對應其他連接埠，請使用 -p Docker 參數：

```
"-p 1883:1883"
```

`mergeConfigurationFiles`

(選擇性) 設定此選項以新增或覆寫指定 EMQX 組態檔案中的預設值。如需有關組態檔及其格式的資訊，請參閱 EMQX 4.0 文件中的組態。您指定的值會附加到組態檔案中。

下列範例會更新`etc/emqx.conf`檔案。

```
"mergeConfigurationFiles": {
    "etc/emqx.conf": "broker.sys_interval=30s\nbroker.sys_heartbeat=10s"
},
```

除了 EMQX 支持的配置文件，Greengrass 支持配置格雷格拉斯身份驗證插件 EMQX 稱為一個文件。`etc/plugins/aws_greengrass_emqx_auth.conf`有兩個支援的選項，`auth_mode`以及`use_greengrass_managed_certificates`。要使用另一個身份驗證提供程序，請將`auth_mode`選項設置為以下之一：

- `enabled`— (預設) 使用 Greengrass 驗證和授權提供者。
- `bypass_on_failure`— 如果 Greengrass 拒絕驗證或授權，請使用 Greengrass 驗證提供者，然後使用 EMQX 提供者鏈中任何剩餘的驗證提供者。
- `bypass`— 格 Greengrass 提供者已停用。然後由 EMQX 提供商鏈處理身份驗證和授權。

如果`use_greengrass_managed_certificates`是`true`，則此選項表示 Greengrass 管理代理程式 TLS 憑證。如果`false`，則表示您透過其他來源提供憑證。

下列範例會更新組`etc/plugins/aws_greengrass_emqx_auth.conf`態檔案中的預設值。

```
"mergeConfigurationFiles": {
    "etc/plugins/aws_greengrass_emqx_auth.conf": "auth_mode=enabled\n
 use_greengrass_managed_certificates=true\n"
  },
```

> **Note**
>
> `aws.greengrass.clientdevices.mqtt.EMQX`可讓您設定敏感安全性選項。其中包括 TLS 設定、驗證和授權提供者。建議的組態是使用相互 TLS 驗證和 Greengrass 用戶端裝置驗證提供者的預設組態。

`replaceConfigurationFiles`

(選擇性) 設定此選項以取代指定的 EMQX 組態檔。您指定的值會取代整個既有的組態檔案。您無法在此區段中指定`etc/emqx.conf`檔案。您必須使用`mergeConfigurationFile`來修改`etc/emqx.conf`。

Example 範例：組態合併更新

下列範例組態指定在連接埠 443 上操作 MQTT 代理程式。

```
{
  "emqx": {
    "listener.ssl.external": "443",
    "listener.ssl.external.max_connections": "1024000",
    "listener.ssl.external.max_conn_rate": "500",
    "listener.ssl.external.rate_limit": "50KB,5s",
    "listener.ssl.external.handshake_timeout": "15s",
    "log.level": "warning"
  },
  "requiresPrivilege": "true",
  "startupTimeoutSeconds": "90",
  "ipcTimeoutSeconds": "5"
}
```

本機記錄檔

此元件使用下列記錄檔。

Linux

```
/greengrass/v2/logs/aws.greengrass.clientdevices.mqtt.EMQX.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.clientdevices.mqtt.EMQX.log
```

若要檢視此元件的記錄

- 在核心裝置上執行下列命令，即時檢視此元件的記錄檔。以 AWS IoT Greengrass 根資料夾的路徑取代*/greengrass/v2*或 *C:\greengrass\v2*。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.clientdevices.mqtt.EMQX.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.greengrass.clientdevices.mqtt.EMQX.log -
  Tail 10 -Wait
  ```

授權

在 Windows 操作系統上，該軟件包括根據 Microsoft 軟件許可條款分發的代碼-Microsoft 視覺工作室社區 2022。通過下載此軟件，您同意該代碼的許可條款。

此元件根據 Greengrass 核心軟體授權合約發行。

變更記錄

下表說明元件每個版本中的變更。

v2.x

版本	變更
2.0.0	此版本的 MQTT 5 代理程式 (EMQX) 預期的組態參數與版本 1.x 不同。如果您對 1.x 版使用非預設組態，則必須更新元件的 2.x 組態。如需詳細資訊，請參閱 組態。 新功能 • 將 MQTT 代理程式升級至 EMQX 5.1.1。 • 啟用 Broker 組態變更，而不必重新啟動元件。

版本	變更
	更新 • 新增取代`emqx`、`mergeConfigurationFiles` 和`emqxConfig` 組態欄位的新`replaceConfigurationFiles` 組態欄位。

v1.x

版本	變更
1.2.3	錯誤修復和改進 • 修正用戶端先前透過中斷連線並重新驗證用戶端進行驗證後，無法與 EMQX 互動的問題。
1.2.2	版本更新了客戶端設備身份驗證版本 2.4.0 版本。
1.2.1	錯誤修復和改進 • 修正如果 Visual C++ 可再散發程式尚未存在，則元件無法在 Windows 上啟動的問題。 • 將 EMQX 更新為 4.4.14 版本。
1.2.0	添加對證書鏈的支持。
1.1.0	新功能 • 添加對 EMQX 配置的支持，包括代理程序選項和插件。 錯誤修復和改進 • 將 EMQX 更新到 4.4.9 版本。
1.0.1	修正 TLS 握手期間導致某些 MQTT 用戶端無法連線的問題。
1.0.0	初始版本。

核遙測發射器

核心遙測發射器元件 (`aws.greengrass.telemetry.NucleusEmitter`) 會收集系統健康情況遙測資料，並持續將其發佈至本機主題和 MQTT 主題。AWS IoT Core此元件可讓您在 Greengrass 核心裝

置上收集即時系統遙測。如需將系統遙測資料發佈至 Amazon EventBridge 之 Greengrass 遙測代理程式的相關資訊，請參閱。從AWS IoT Greengrass核心裝置收集系統健康狀態遙測資料

根據預設，核心遙測發射器元件會每 60 秒發佈一次遙測資料至下列本機發佈/訂閱主題。

```
$local/greengrass/telemetry
```

核心遙測發射器元件預設不會發佈至 AWS IoT Core MQTT 主題。您可以將此元件設定為在部署 AWS IoT Core MQTT 主題時發佈至 MQTT 主題。使用 MQTT 主題將資料發佈至的需要AWS 雲端AWS IoT Core定價。

AWS IoT Greengrass提供數個社群元件，協助您使用 InfluxDB 和 Grafana 在核心裝置上本機分析和視覺化遙測資料。這些元件使用來自核輻射器元件的遙測資料。如需詳細資訊，請參閱 In fluxDB 發行者元件的讀我檔案。

主題

- 版本
- Type
- 作業系統
- 相依性
- 組態
- 輸出資料
- 用量
- 本機記錄檔
- 變更記錄

版本

此元件具有下列版本：

- 1.0.x

Type

這個組件是一個插件組件（`aws.greengrass.plugin`）。Greengrass 核在與核相同的 Java 虛擬機（JVM）中運行此組件。當您在核心裝置上變更此元件的版本時，核心會重新啟動。

此組件使用與 Greengrass 核相同的日誌文件。如需詳細資訊，請參閱 監控AWS IoT Greengrass日誌。

如需詳細資訊，請參閱 元件類型。

作業系統

此元件可安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

相依性

部署元件時，AWS IoT Greengrass也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的需求，才能成功部署元件。本節列出此元件之已發行版本的相依性，以及定義每個相依性之元件版本的語意版本條件約束。您也可以在AWS IoT Greengrass主控台中檢視元件每個版本的相依性。在元件詳細資料頁面上，尋找 [相依性] 清單。

1.0.8

下表列出此元件 1.0.8 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.4.0	硬式

1.0.7

下表列出此元件 1.0.7 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.4.0	硬式

1.0.6

下表列出此元件 1.0.6 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.4.0	硬式

1.0.5

下表列出此元件 1.0.5 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.4.0	硬式

1.0.4

下表列出此元件 1.0.4 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.4.0	硬式

1.0.3

下表列出此元件 1.0.3 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.4.0	硬式

1.0.2

下表列出此元件 1.0.2 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.4.0	硬式

1.0.1

下表列出此元件 1.0.1 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.4.0	硬式

1.0.0

下表列出此元件 1.0.0 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.4.0	硬式

如需有關元件相依性的詳細資訊，請參閱元件方案參考。

組態

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

`pubSubPublish`

(選擇性) 定義是否將遙測資料發佈至`$local/greengrass/telemetry`主題。支援的值為 `true` 和 `false`。

預設：`true`

`mqttTopic`

(選擇性) 此元件發佈遙測資料的 AWS IoT Core MQTT 主題。

將此值設定為您要發佈遙測資料的 AWS IoT Core MQTT 主題。當此值為空時，核子核發射器不會將遙測資料發佈至. AWS 雲端

> **Note**
>
> 使用 MQTT 主題將資料發佈至的需要AWS 雲端AWS IoT Core定價。

預設：""

`telemetryPublishIntervalMs`

(選擇性) 元件發佈遙測資料之間的時間量 (以毫秒為單位)。如果您將此值設定為低於支援的最小值，元件會改用最小值。

> **Note**
>
> 較低的發佈間隔會導致核心裝置上的 CPU 使用率較高。我們建議您從預設發佈間隔開始，並根據裝置的 CPU 使用率進行調整。

下限：`500`

預設：`60000`

Example 範例：組態合併更新

下列範例顯示組態合併更新範例，此更新可讓您每 5 秒發佈一次遙測資料至`$local/greengrass/telemetry`主題和 `greengrass/myTelemetry` AWS IoT Core MQTT 主題。

```
{
  "pubSubPublish": "true",
  "mqttTopic": "greengrass/myTelemetry",
  "telemetryPublishIntervalMs": 5000
}
```

輸出資料

此元件會根據下列主題將遙測度量發佈為 JSON 陣列。

本地主題：`$local/greengrass/telemetry`

您也可以選擇將遙測指標發佈至 AWS IoT Core MQTT 主題。如需有關主題的詳細資訊，請參閱AWS IoT Core開發人員指南中的 MQTT 主題。

Example 範例資料

```
[
  {
    "A": "Average",
```

```
    "N": "CpuUsage",
    "NS": "SystemMetrics",
    "TS": 1627597331445,
    "U": "Percent",
    "V": 26.21981271562346
  },
  {
    "A": "Count",
    "N": "TotalNumberOfFDs",
    "NS": "SystemMetrics",
    "TS": 1627597331445,
    "U": "Count",
    "V": 7316
  },
  {
    "A": "Count",
    "N": "SystemMemUsage",
    "NS": "SystemMetrics",
    "TS": 1627597331445,
    "U": "Megabytes",
    "V": 10098
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsStarting",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 0
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsInstalled",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 0
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsStateless",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
```

```
    "V": 0
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsStopping",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 0
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsBroken",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 0
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsRunning",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 7
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsErrored",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 0
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsNew",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 0
  },
  {
    "A": "Count",
```

```
    "N": "NumberOfComponentsFinished",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 2
  }
]
```

輸出陣列包含具有下列屬性的度量清單：

A

測量結果的聚總類型。

對於CpuUsage測量結果，此特性會設為，Average因為測量結果的已發佈值是自上次發佈事件以來的平均 CPU 使用量。

對於所有其他度量，核子核發射器不會彙總度量值，而且此屬性設定為。Count

N

指標的名稱

NS

測量結果命名空間。

TS

收集資料的時間戳記。

U

公制值的單位。

V

指標值。

核發射器發布以下指標：

名稱	描述	
系統		

名稱	描述	
SystemMemUsage	Greengrass 核心裝置 (包括作業系統) 上所有應用程式目前使用的記憶體容量。	
CpuUsage	Greengrass 核心裝置 (包括作業系統) 上所有應用程式目前正在使用的 CPU 數量。	
TotalNumberOfFDs	Greengrass 核心裝置作業系統所儲存的檔案描述元數目。一個文件描述符唯一標識一個打開的文件。	
Greengrass 核		
NumberOfComponentsRunning	在 Greengrass 核心裝置上執行的元件數目。	
NumberOfComponentsErrored	Greengrass 核心裝置上處於錯誤狀態的元件數目。	
NumberOfComponentsInstalled	安裝在 Greengrass 核心裝置上的元件數目。	
NumberOfComponentsStarting	在 Greengrass 核心裝置上啟動的元件數目。	
NumberOfComponentsNew	Greengrass 核心裝置上新增的元件數目。	
NumberOfComponentsStopping	在 Greengrass 核心裝置上停止的元件數目。	
NumberOfComponentsFinished	在 Greengrass 核心裝置上完成的元件數目。	

名稱	描述	
NumberOfComponentsBroken	Greengrass 核心裝置上損壞的元件數目。	
NumberOfComponentsStateless	Greengrass 核心裝置上無狀態的元件數目。	

用量

若要使用系統健康情況遙測資料，您可以建立自訂元件，以訂閱核心發射器發佈遙測資料的主題，並視需要回應該資料。由於核心發射器元件提供將遙測資料發佈到本機主題的選項，因此您可以訂閱該主題，並使用已發佈的資料在核心裝置上在本機上執行動作。然後，即使與雲端的連線有限，核心裝置也可以對遙測資料做出反應。

例如，您可以設定偵聽遙測資料`$local/greengrass/telemetry`主題的元件，並將資料傳送至串流管理員元件，以將資料串流至. AWS 雲端 如需建立此類元件的詳細資訊，請參閱發佈/訂閱本地訊息和建立使用串流管理員的自訂元件。

本機記錄檔

此元件使用與 Greengrass 核元件相同的記錄檔。

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

若要檢視此元件的記錄

- 在核心裝置上執行下列命令，即時檢視此元件的記錄檔。以AWS IoT Greengrass根資料夾的路徑取代*/greengrass/v2*或 *C:\greengrass\v2*。

Linux

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
```

變更記錄

下表說明元件每個版本中的變更。

版本	變更
1.0.8	版本更新 Greengrass 2.12.0 版本釋放。
1.0.7	版本更新 Greengrass 2.11.0 版本釋放。
1.0.6	版本更新了 Greengrass 2.10.0 版本。
1.0.5	版本更新 Greengrass 2.9.0 版本釋放。
1.0.4	版本更新 Greengrass 2.8.0 版本的版本。
1.0.3	版本更新了 Greengrass 核 2.7.0 版本釋放。
1.0.2	版本更新 Greengrass 2.6.0 版本發布。
1.0.1	版本更新了 Greengrass 核 2.5.0 版本。
1.0.0	初始版本。

PKCS #11 供應商

PKCS #11 提供者元件 (`aws.greengrass.crypto.Pkcs11Provider`) 可讓您將AWS IoT Greengrass核心軟體設定為透過 PK CS #11 介面使用硬體安全性模組 (HSM)。此元件可讓您安全地儲存憑證和私密金鑰檔案，以便在軟體中不會公開或複製這些檔案。如需詳細資訊，請參閱 硬體安全整合。

若要佈建將憑證和私密金鑰儲存在 HSM 中的 Greengrass 核心裝置，您必須在安裝 Core 軟體時將此元件指定為佈建外掛程式。AWS IoT Greengrass如需詳細資訊，請參閱 透過手動佈建資源安裝 AWS IoT Greengrass 核心軟體。

AWS IoT Greengrass將此元件提供為 JAR 檔案，您可以下載該檔案，以便在安裝期間指定為佈建外掛程式。您可以透過下列網址下載最新版本的元件 JAR 檔案：https://d2s8p88vqu9w66.cloudfront.net/releases/Pkcs11Provider/aws.greengrass.crypto.Pkcs11Provider-latest.jar.

主題

版本

此元件具有下列版本：

- 2.0.x

Type

這個組件是一個插件組件（`aws.greengrass.plugin`）。Greengrass 核在與核相同的 Java 虛擬機（JVM）中運行此組件。當您在核心裝置上變更此元件的版本時，核心會重新啟動。

此組件使用與 Greengrass 核相同的日誌文件。如需詳細資訊，請參閱 監控AWS IoT Greengrass日誌。

如需詳細資訊，請參閱 元件類型。

作業系統

此元件只能安裝在 Linux 核心裝置上。

要求

此元件具有下列需求：

- 一種硬體安全性模組，支援 PKCS #1 v1.5 簽章配置，以及具有 RSA-2048 金鑰大小 (或更大) 或 ECC 金鑰的 RSA 金鑰。

  > **Note**
  >
  > 若要搭配 ECC 金鑰使用硬體安全性模組，您必須使用 Greengrass 核心 v2.5.6 或更新版本。
  > 要使用硬件安全模塊和密碼管理器，您必須使用帶有 RSA 密鑰的硬件安全模塊。

- AWS IoT Greengrass核心軟體可以在執行階段 (使用 libdl) 載入的 PKCS #11 提供者程式庫，以呼叫 PKCS #11 函數。PKCS #11 提供者程式庫必須實作下列 PKCS #11 API 作業：
  - `C_Initialize`
  - `C_Finalize`
  - `C_GetSlotList`
  - `C_GetSlotInfo`
  - `C_GetTokenInfo`
  - `C_OpenSession`
  - `C_GetSessionInfo`
  - `C_CloseSession`
  - `C_Login`
  - `C_Logout`
  - `C_GetAttributeValue`
  - `C_FindObjectsInit`
  - `C_FindObjects`
  - `C_FindObjectsFinal`
  - `C_DecryptInit`
  - `C_Decrypt`
  - `C_DecryptUpdate`
  - `C_DecryptFinal`
  - `C_SignInit`

  - C_Sign
  - C_SignUpdate
  - C_SignFinal
  - C_GetMechanismList
  - C_GetMechanismInfo
  - C_GetInfo
  - C_GetFunctionList
- 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。
- 您必須將私密金鑰和憑證儲存在相同的插槽中，而且如果 HSM 支援物件 ID，則必須使用相同的物件標籤和物件 ID。
- 憑證和私密金鑰必須可透過物件標籤來解析。
- 私密金鑰必須具有下列權限：
  - sign
  - decrypt
- (選擇性) 若要使用密碼管理員元件，您必須使用 2.1.0 或更新版本，且私密金鑰必須具有下列權限：
  - unwrap
  - wrap
- (選擇性) 如果您使用 TPM2 程式庫並將 Greengrass 核心作為服務執行，則必須提供環境變數，其中包含 PKCS #11 儲存區的位置。下列範例是具有必要環境變數的 systemd 服務檔案：

```
[Unit]
Description=Greengrass Core
After=network.target

[Service]
Type=simple
PIDFile=/var/run/greengrass.pid
Environment=TPM2_PKCS11_STORE=/path/to/store/directory
RemainAfterExit=no
Restart=on-failure
RestartSec=10
ExecStart=/bin/sh /greengrass/v2/alts/current/distro/bin/loader

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

相依性

部署元件時，AWS IoT Greengrass也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的需求，才能成功部署元件。本節列出此元件之已發行版本的相依性，以及定義每個相依性之元件版本的語意版本條件約束。您也可以在AWS IoT Greengrass主控台中檢視元件每個版本的相依性。在元件詳細資料頁面上，尋找 [相依性] 清單。

2.0.7

下表列出此元件 2.0.7 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.5.3	軟式

2.0.6

下表列出此元件 2.0.6 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.5.3	軟式

2.0.5

下表列出此元件 2.0.5 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.5.3	軟式

2.0.4

下表列出此元件 2.0.4 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.5.3	軟式

2.0.3

下表列出此元件 2.0.3 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.5.3 <2.9.0	軟式

2.0.2

下表列出此元件 2.0.2 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.5.3	軟式

2.0.1

下表列出此元件 2.0.1 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.5.3	軟式

2.0.0

下表列出此元件 2.0.0 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.5.3	軟式

如需有關元件相依性的詳細資訊，請參閱元件方案參考。

組態

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

`name`

PKCS #11 組態的名稱。

`library`

PKCS #11 實作程式庫的絕對檔案路徑，AWS IoT Greengrass核心軟體可以使用 libdl 載入。

`slot`

包含私密金鑰和裝置憑證的插槽識別碼。此值與槽索引或槽標籤不同。

`userPin`

用來存取插槽的使用者 PIN 碼。

Example 範例：組態合併更新

```
{
  "name": "softhsm_pkcs11",
  "library": "/usr/lib/softhsm/libsofthsm2.so",
  "slot": 1,
  "userPin": "1234"
}
```

本機記錄檔

此元件使用與 Greengrass 核元件相同的記錄檔。

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

若要檢視此元件的記錄

- 在核心裝置上執行下列命令，即時檢視此元件的記錄檔。以AWS IoT Greengrass根資料夾的路徑取代*/greengrass/v2*或 *C:\greengrass\v2*。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

變更記錄

下表說明元件每個版本中的變更。

版本	變更
2.0.7	版本更新了 Greengrass 核 2.12.0 版本釋放。
2.0.6	版本更新了 Greengrass 核 2.11.0 版本釋放。
2.0.5	版本更新了 Greengrass 2.10.0 版本。
2.0.4	版本更新 Greengrass 2.9.0 版本釋放。
2.0.3	版本更新 Greengrass 2.8.0 版本的版本。
2.0.2	版本更新了 Greengrass 核 2.7.0 版本釋放。
2.0.1	版本更新 Greengrass 2.6.0 版本的版本。
2.0.0	初始版本。

秘密經理

密碼管理員元件 (`aws.greengrass.SecretManager`) 會將密碼從 AWS Secrets Manager Greengrass 核心裝置部署。使用此元件可在 Greengrass 核心裝置的自訂元件中安全地使用認證，例

如密碼。如需有關 Secrets Manager 的詳細資訊，請參閱什麼是AWS Secrets Manager？ 在《AWS Secrets Manager使用者指南》中。

若要在自訂 Greengrass 元件中存取此元件的密碼，請使GetSecretValue用. AWS IoT Device SDK 如需詳細資訊，請參閱 使AWS IoT Device SDK用與 Greengrass 核、其他元件和通訊 AWS IoT Core 及擷取秘密值。

此元件會加密核心裝置上的密碼，以確保您的憑證和密碼安全，直到您需要使用它們為止。它使用核心裝置的私密金鑰來加密和解密密碼。

主題

版本

此元件具有下列版本：

- 2.1.x
- 2.0.x

Type

這個組件是一個插件組件（`aws.greengrass.plugin`）。Greengrass 核在與核相同的 Java 虛擬機（JVM）中運行此組件。當您在核心裝置上變更此元件的版本時，核心會重新啟動。

此組件使用與 Greengrass 核相同的日誌文件。如需詳細資訊，請參閱 監控AWS IoT Greengrass日誌。

如需詳細資訊，請參閱 元件類型。

作業系統

此元件可安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件具有下列需求：

- Greengrass 裝置角色必須允許`secretsmanager:GetSecretValue`動作，如下列 IAM 政策範例所示。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "secretsmanager:GetSecretValue"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:secretsmanager:region:123456789012:secret:MySecret"
      ]
    }
  ]
}
```

> **Note**
>
> 如果您使用客戶管理的AWS Key Management Service金鑰來加密密碼，則裝置角色也必須允許該`kms:Decrypt`處理行動。

如需有關 Secrets Manager 的身分與存取權管理政策的詳細資訊，請參閱AWS Secrets Manager使用者指南中的下列

- 驗證和存取控制 AWS Secrets Manager
- 您可以在 IAM 政策或秘密政策中使用的動作、資源和內容金鑰 AWS Secrets Manager

- 自訂元件必須定義授權原則，aws.greengrass#GetSecretValue以取得您儲存在此元件的密碼。在此授權原則中，您可以限制元件對特定機密的存取。如需詳細資訊，請參閱密碼管理員 IPC 授權。
- (選擇性) 如果將核心裝置的私密金鑰和憑證儲存在硬體安全性模組 (HSM) 中，則 HSM 必須支援 RSA 金鑰、私密金鑰必須具有unwrap權限，且公開金鑰必須具有權限。wrap

端點和連接埠

除了基本作業所需的端點和連接埠之外，此元件還必須能夠對下列端點和連接埠執行輸出要求。如需詳細資訊，請參閱 允許裝置流量透過 Proxy 或防火牆。

端點	連線埠	必要	描述
secretsmanager. *region*.amazonaws.com	443	是	將秘密下載到核心設備。

相依性

部署元件時，AWS IoT Greengrass也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的需求，才能成功部署元件。本節列出此元件之已發行版本的相依性，以及定義每個相依性之元件版本的語意版本條件約束。您也可以在AWS IoT Greengrass主控台中檢視元件每個版本的相依性。在元件詳細資料頁面上，尋找 [相依性] 清單。

2.1.7

下表列出此元件 2.1.7 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.5.0	軟式

2.1.6

下表列出此元件 2.1.6 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.5.0	軟式

2.1.5

下表列出此元件 2.1.5 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.5.0	軟式

2.1.4

下表列出此元件 2.1.4 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.5.0	軟式

2.1.3

下表列出此元件 2.1.3 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.5.0	軟式

2.1.2

下表列出此元件 2.1.2 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.5.0	軟式

2.1.1

下表列出此元件 2.1.1 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.5.0	軟式

2.1.0

下表列出此元件 2.1.0 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.5.0	軟式

2.0.9

下表列出此元件 2.0.9 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.0.8

下表列出此元件 2.0.8 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.0.7

下表列出此元件 2.0.7 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.0.6

下表列出此元件 2.0.6 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

2.0.4 and 2.0.5

下表列出此元件 2.0.4 和 2.0.5 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.3	軟式

如需有關元件相依性的詳細資訊，請參閱元件方案參考。

組態

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

`cloudSecrets`

要部署到核心裝置的密碼清單。您可以指定標籤來定義要部署的每個密碼版本。如果您未指定版本，則此元件會部署AWSCURRENT附加測試標籤的版本。如需詳細資訊，請參閱AWS Secrets Manager使用指南中的暫存標籤。

密碼管理員元件會在本機快取密碼。如果秘密值在 Secrets Manager 中發生變更，此元件不會自動擷取新值。若要更新本機副本，請為機密提供新標籤，並將此元件設定為擷取新標籤所識別的密碼。

每個物件都包含下列資訊：

`arn`

要部署之密碼的 ARN。秘密的 ARN 可以是完整的 ARN 或部分 ARN。我們建議您指定完整的 ARN，而不是部分 ARN。如需詳細資訊，請參閱從部分 ARN 尋找密碼。以下是完整 ARN 和部分 ARN 的示例：

- 全 ARN: `arn:aws:secretsmanager:us-east-2:111122223333:secret:SecretName-abcdef`
- 部分 ARN：`arn:aws:secretsmanager:us-east-2:111122223333:secret:SecretName`

`labels`

(選擇性) 標籤清單，用於識別要部署到核心裝置的密碼版本。

每個標籤必須是一個字符串。

Example 範例：組態合併更新

```
{
  "cloudSecrets": [
    {
      "arn": "arn:aws:secretsmanager:us-west-2:123456789012:secret:MyGreengrassSecret-
abcdef"
    }
  ]
}
```

本機記錄檔

此元件使用與 Greengrass 核元件相同的記錄檔。

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

若要檢視此元件的記錄

- 在核心裝置上執行下列命令，即時檢視此元件的記錄檔。以AWS IoT Greengrass根資料夾的路徑取代*/greengrass/v2*或 *C:\greengrass\v2*。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

變更記錄

下表說明元件每個版本的變更。

版本	變更
2.1.7	版本更新 Greengrass 2.12.0 版本釋放。
2.1.6	版本更新了 Greengrass 核 2.11.0 版本釋放。
2.1.5	版本更新了 Greengrass 2.10.0 版本。
2.1.4	錯誤修復和改進 修正部署秘密管理員且 Greengrass 核心重新啟動時，會移除快取密碼的問題。 版本更新 Greengrass 2.9.0 版本釋放。
2.1.3	版本更新 Greengrass 2.8.0 版本的版本。
2.1.2	版本更新了 Greengrass 核 2.7.0 版本釋放。
2.1.1	版本更新 Greengrass 2.6.0 版本發布。

版本	變更
2.1.0	新功能 • 添加對硬件安全集成的支持。秘密管理員元件可以使用您儲存在硬體安全性模組 (HSM) 中的私密金鑰來加密和解密密碼。如需詳細資訊，請參閱硬體安全整合。 錯誤修復和改進 • 版本更新了 Greengrass 核 2.5.0 版本。
2.0.9	版本更新 Greengrass 2.4.0 版本的版本。
2.0.8	版本更新了 Greengrass 核 2.3.0 版本。
2.0.7	版本更新了 Greengrass 核 2.2.0 版本。
2.0.6	版本更新 Greengrass 2.1.0 版本發布。
2.0.5	改善項目 • 新增對AWS中國地區和AWS GovCloud (US)地區的支援。
2.0.4	初始版本。

安全隧道

使用`aws.greengrass.SecureTunneling`元件，您可以與位於受限防火牆後方的 Greengrass 核心裝置建立安全的雙向通訊。

例如，假設您在防火牆後面有一個 Greengrass 核心設備，該設備禁止所有傳入連接。安全通道使用 MQTT 將存取權杖傳輸到裝置，然後使用 WebSockets 透過防火牆與裝置建立 SSH 連線。使用此AWS IoT託管通道，您可以打開設備所需的 SSH 連接。如需使用AWS IoT安全通道連線至遠端裝置的詳細資訊，請參閱開發人員指南中的AWS IoT安全通道。AWS IoT

此元件會根據`$aws/things/greengrass-core-device/tunnels/notify`主題訂閱 AWS IoT Core MQTT 訊息代理程式，以接收安全通道通知。

主題

- 版本
- Type

版本

此元件具有下列版本：

- 1.0.x

Type

此元件是一般元件 (`aws.greengrass.generic`)。Greengrass 核會執行元件的生命週期指令碼。

如需詳細資訊，請參閱 元件類型。

作業系統

此元件只能安裝在 Linux 核心裝置上。

架構：

- 阿尔姆维 71 号
- Armv8 (AArch64)
- x86_64

要求

此元件具有下列需求：

- 安全通道元件可用的最少 32 MB 磁碟空間。此要求不包括 Greengrass 核心軟體或在相同裝置上執行的其他元件。
- 安全通道元件至少有 16 MB RAM 可用。此要求不包括 Greengrass 核心軟體或在相同裝置上執行的其他元件。如需詳細資訊，請參閱 使用 JVM 選項控制內存分配。
- 安全通道元件版本 1.0.12 及以上版本需要使用 GNU C 程式庫 (glibc) 2.25 版或更高版本，且 Linux 核心為 3.2 或更高版本。不支援超過其長期支援生命週期結束日期的作業系統和程式庫版本。您應該使用具有長期支持的操作系統和庫。
- 作業系統和 Java 執行階段都必須安裝為 64 位元。
- Python 3.5 或更高版本安裝在核心設備上，並添加到 PATH 環境變量中。
- `libcrypto.so.1.1`安裝在 Greengrass 核心裝置上，並新增至 PATH 環境變數。
- 在 Greengrass 核心裝置上的連接埠 443 上開啟輸出流量。
- 開啟您要用來與 Greengrass 核心裝置通訊的通訊服務支援。例如，若要開啟與裝置的 SSH 連線，您必須在該裝置上開啟 SSH。

端點和連接埠

除了基本作業所需的端點和連接埠之外，此元件還必須能夠對下列端點和連接埠執行輸出要求。如需詳細資訊，請參閱 允許裝置流量透過 Proxy 或防火牆。

端點	連線埠	必要	描述
`data.tunneling.iot.`*`region`*`.amazonaws.com`	443	是	建立安全的隧道。

相依性

部署元件時，AWS IoT Greengrass也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的需求，才能成功部署元件。本節列出此元件之已發行版本的相依性，以及定義每個相依性之元件版本的語意版本條件約束。您也可以在AWS IoT Greengrass主控台中檢視元件每個版本的相依性。在元件詳細資料頁面上，尋找 [相依性] 清單。

1.0.18

下表列出此元件 1.0.18 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

1.0.16 – 1.0.17

下表列出此元件 1.0.16 至 1.0.17 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

1.0.14 – 1.0.15

下表列出此元件 1.0.14 至 1.0.15 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

1.0.11 – 1.0.13

下表列出此元件 1.0.11 — 1.0.13 版本的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

1.0.10

下表列出此元件 1.0.10 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.0.0	軟式

1.0.9

下表列出此元件 1.0.9 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

1.0.8

下表列出此元件 1.0.8 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

1.0.5 - 1.0.7

下表列出此元件 1.0.5 到 1.0.7 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.0.0	軟式

1.0.4

下表列出此元件 1.0.4 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

1.0.3

下表列出此元件 1.0.3 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

1.0.2

下表列出此元件 1.0.2 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

1.0.1

下表列出此元件 1.0.1 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

1.0.0

下表列出此元件 1.0.0 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.3	軟式

如需有關元件相依性的詳細資訊，請參閱元件方案參考。

組態

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

`OS_DIST_INFO`

(選擇性) 核心裝置的作業系統。根據預設，元件會嘗試自動識別核心裝置上執行的作業系統。如果元件無法以預設值啟動，請使用此值來指定作業系統。如需此元件支援的作業系統清單，請參閱裝置要求。

此值可以是下列其中一項：`auto`、`ubuntu`、`amzn2`、`raspberrypi`。

預設：`auto`

`accessControl`

(選擇性) 包含授權原則的物件，可讓元件訂閱安全通道通知主題。

> **Note**
>
> 如果您的部署以物件群組為目標，請勿修改此組態參數。如果您的部署以個別核心裝置為目標，而您想要將其訂閱限制為裝置主題，請指定核心裝置的物件名稱。在裝置授權原則的`resources`值中，將 MQTT 主題萬用字元取代為裝置的物件名稱。

```
{
  "aws.greengrass.ipc.mqttproxy": {
    "aws.iot.SecureTunneling:mqttproxy:1": {
      "policyDescription": "Access to tunnel notification pubsub topic",
      "operations": [
        "aws.greengrass#SubscribeToIoTCore"
      ],
      "resources": [
        "$aws/things/+/tunnels/notify"
      ]
    }
  }
}
```

Example 範例：組態合併更新

下列範例組態指定允許此元件在執行 Ubuntu 的核心裝置上開啟安全通道。**MyGreengrassCore**

```
{
```

```
  "OS_DIST_INFO": "ubuntu",
  "accessControl": {
    "aws.greengrass.ipc.mqttproxy": {
      "aws.iot.SecureTunneling:mqttproxy:1": {
        "policyDescription": "Access to tunnel notification pubsub topic",
        "operations": [
          "aws.greengrass#SubscribeToIoTCore"
        ],
        "resources": [
          "$aws/things/MyGreengrassCore/tunnels/notify"
        ]
      }
    }
  }
}
```

本機記錄檔

此元件會使用下列記錄檔。

```
/greengrass/v2/logs/aws.greengrass.SecureTunneling.log
```

若要檢視此元件的記錄

- 在核心裝置上執行下列命令，即時檢視此元件的記錄檔。以AWS IoT Greengrass根資料夾的路徑取*/greengrass/v2*代。

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.SecureTunneling.log
  ```

授權

此元件包括下列協力廠商軟體/授權：

- AWS IoT裝置用戶@@ 端 /阿帕奇授權 2.0
- 適用於 JAVA 的 AWS IoT Device SDK/Apache License 2.0
- 克森/阿帕奇許可證 2.0
- 阿帕奇許可證 2.0
- 阿帕奇 2.0 許可證

用量

若要使用裝置上的安全通道元件，請執行下列動作：

1. 將安全通道元件部署到您的裝置。
2. 開啟 AWS IoT主控台。從左側功能表中選擇 [遠端動作]，然後選擇 [安全通道]。
3. 創建一個通往您的綠色設備的隧道。
4. 下載源訪問令牌。
5. 使用本地代理與源訪問令牌連接到您的目的地。如需詳細資訊，請參閱AWS IoT開發人員指南中的如何使用本機 Proxy。

另請參閱

- AWS IoT開發人員指南中的AWS IoT安全通道
- 如何在AWS IoT開發人員指南中使用本地代理

變更記錄

下表說明元件每個版本中的變更。

版本	變更
1.0.18	版本更新了 Greengrass 核 2.12.0 版本釋放。
1.0.17	錯誤修復和改進 • 修復了阻止用戶創建隧道的線程清理問題。此元件現在會在收到 CloseTunnel 訊號後或通道在 12 小時後過期時清除執行緒。
1.0.16	版本更新了 Greengrass 核 2.11.0 版本釋放。
1.0.15	錯誤修復和改進 • 修正裝置上沒有主目錄的使用者的啟動問題。安全通道組件現在不為陰影文檔創建目錄啟動。
1.0.14	版本更新了 Greengrass 核 2.10.0 版本釋放。

版本	變更
1.0.13	錯誤修復和改進 • 修正孤立用戶端程序會阻止多個通道鎖定裝置的問題。
1.0.12	錯誤修復和改進 • 在樹莓派操作系統上運行時添加對 x86_64（AMD64）和 ARMv8（Aarch64）的支持。
1.0.11	版本更新 Greengrass 2.9.0 版本釋放。
1.0.10	版本更新 Greengrass 2.8.0 版本的版本。
1.0.9	版本更新了 Greengrass 核 2.7.0 版本釋放。
1.0.8	版本更新 Greengrass 2.6.0 版本的版本。
1.0.7	錯誤修復和改進 • 修正透過 SCP 傳輸大型檔案時，元件中斷連線的問題。
1.0.6	此版本包含錯誤修復。
1.0.5	版本更新了 Greengrass 核 2.5.0 版本。
1.0.4	版本更新 Greengrass 2.4.0 版本的版本。
1.0.3	版本更新了 Greengrass 核 2.3.0 版本。
1.0.2	版本更新了 Greengrass 核 2.2.0 版本。
1.0.1	版本更新 Greengrass 2.1.0 版本的版本。
1.0.0	初始版本。

陰影管理

陰影管理員元件 (`aws.greengrass.ShadowManager`) 會在核心裝置上啟用本機陰影服務。本機陰影服務可讓元件使用處理序間通訊與本機陰影互動。陰影管理員元件會管理本機陰影文件的儲存空間，也會處理本機陰影狀態與 AWS IoT Device Shadow 服務的同步處理。

如需 Greengrass 核心裝置如何與陰影互動的詳細資訊，請參閱。與裝置陰影互動

主題

版本

此元件具有下列版本：

- 2.3.x 版本
- 2.2.x 版本
- 2.1.x
- 2.0.x

Type

這個組件是一個插件組件（`aws.greengrass.plugin`）。Greengrass 核在與核相同的 Java 虛擬機（JVM）中運行此組件。當您在核心裝置上變更此元件的版本時，核心會重新啟動。

此組件使用與 Greengrass 核相同的日誌文件。如需詳細資訊，請參閱 監控AWS IoT Greengrass日誌。

如需詳細資訊，請參閱 元件類型。

作業系統

此元件可安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件具有下列需求：

- (選擇性) 若要將陰影同步至 AWS IoT Device Shadow 服務，Greengrass 核心裝置的 AWS IoT 原則必須允許下列 AWS IoT Core 陰影原則動作：
  - `iot:GetThingShadow`
  - `iot:UpdateThingShadow`
  - `iot:DeleteThingShadow`

  如需這些政 AWS IoT Core 策的詳細資訊，請參閱AWS IoT 開發人員指南中的AWS IoT Core 政策動作。

  如需最小 AWS IoT 原則的詳細資訊，請參閱 AWS IoT Greengrass V2核心裝置的最低AWS IoT原則
- 支援在 VPC 中執行陰影管理員元件。

相依性

部署元件時， AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的需求，才能成功部署元件。本節列出此元件之已發行版本的相依性，以及定義每個相依性之元件版本的語意版本條件約束。您也可以在AWS IoT Greengrass 主控台中檢視元件每個版本的相依性。在元件詳細資料頁面上，尋找 [相依性] 清單。

2.3.5 – 2.3.7

下表列出此元件 2.3.5 到 2.3.7 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.5.0	軟式

2.3.3 and 2.3.4

下表列出此元件 2.3.3 和 2.3.4 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.5.0	軟式

2.3.2

下表列出此元件 2.3.2 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.5.0	軟式

2.3.0 and 2.3.1

下表列出此元件 2.3.0 和 2.3.1 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.5.0	軟式

2.2.3 and 2.2.4

下表列出此元件 2.2.3 和 2.2.4 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.2.0	軟式

2.2.2

下表列出此元件 2.2.2 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.2.0	軟式

2.2.1

下表列出此元件 2.2.1 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.2.0	軟式

2.1.1 and 2.2.0

下表列出此元件 2.1.1 和 2.2.0 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.2.0	軟式

2.0.5 - 2.1.0

下表列出此元件 2.0.5 到 2.1.0 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.2.0	軟式

2.0.3 and 2.0.4

下表列出此元件 2.0.3 和 2.0.4 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.2.0	軟式

2.0.1 and 2.0.2

下表列出此元件 2.0.1 和 2.0.2 版本的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.2.0	軟式

2.0.0

下表列出此元件 2.0.0 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.2.0	軟式

如需有關元件相依性的詳細資訊，請參閱元件方案參考。

組態

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

2.3.x

`strategy`

(選擇性) 此元件用來在 AWS IoT Core 與核心裝置之間同步陰影的策略。

此物件包含下列資訊。

`type`

(選擇性) 此元件用來同步處理 AWS IoT Core 與核心裝置之間陰影的策略類型。您可以從以下選項中選擇：

- `realTime`— AWS IoT Core 每次陰影更新時都會與陰影同步。
- `periodic`— 與您使用 AWS IoT Core `delay`組態參數指定的定期間隔同步陰影。

預設：`realTime`

`delay`

(選擇性) 當您指定`periodic`同步策略時 AWS IoT Core，此元件會與陰影同步的間隔 (以秒為單位)。

> **Note**
>
> 如果您指定`periodic`同步策略，則需要此參數。

`synchronize`

(選擇性) 決定陰影與的同步化設定 AWS 雲端。

> **Note**
>
> 您必須使用此屬性建立模型組態更新，才能與陰影同步 AWS 雲端。

此物件包含下列資訊。

`coreThing`

(選擇性) 要同步的核心裝置陰影。此物件包含下列資訊。

`classic`

(選擇性) 根據預設，陰影管理員會將核心裝置的傳統陰影的本機狀態與. AWS 雲端如果您不想同步傳統裝置陰影，請將其設定為`false`。

預設：`true`

`namedShadows`

(選擇性) 要同步的具名核心裝置陰影清單。您必須指定陰影的確切名稱。

> **Warning**
>
> 此 AWS IoT Greengrass 服務會使用`AWSManagedGreengrassV2Deployment`具名的陰影來管理以個別核心裝置為目標的部署。此具名陰影會保留供 AWS IoT Greengrass 服務使用。請勿更新或刪除此命名陰影。

`shadowDocumentsMap`

(選擇性) 要同步的其他裝置陰影。使用此組態參數可讓您更輕鬆地指定陰影文件。我們建議您使用此參數而不是`shadowDocuments`物件。

> **Note**
>
> 如果您指定shadowDocumentsMap物件，則不得指定shadowDocuments物件。

每個物件都包含下列資訊：

thingName

此陰影組態之 *thingName* 的陰影組態。

classic

(選擇性) 如果您不想同步裝置的傳統裝置陰影，請將此設定為false。thingName

namedShadows

您要同步的已命名陰影清單。您必須指定陰影的確切名稱。

shadowDocuments

(選擇性) 要同步的其他裝置陰影清單。我們建議您改用shadowDocumentsMap參數。

> **Note**
>
> 如果您指定shadowDocuments物件，則不得指定shadowDocumentsMap物件。

此清單中的每個物件都包含下列資訊。

thingName

要同步陰影之裝置的物件名稱。

classic

(選擇性) 如果您不想同步裝置的傳統裝置陰影，請將此設定為false。thingName

預設：true

namedShadows

(選擇性) 您要同步的已命名裝置陰影清單。您必須指定陰影的確切名稱。

direction

(選擇性) 在本機陰影服務與 AWS 雲端. 您可以設定此選項以減少頻寬和與 AWS 雲端. 您可以從以下選項中選擇：

- `betweenDeviceAndCloud`— 同步處理本機陰影服務與 AWS 雲端.
- `deviceToCloud`— 將陰影更新從本機陰影服務傳送至 AWS 雲端，並忽略來自 AWS 雲端.
- `cloudToDevice`— 從接收陰影更新 AWS 雲端，並且不要將陰影更新從本機陰影服務傳送到 AWS 雲端.

預設：`BETWEEN_DEVICE_AND_CLOUD`

`rateLimits`

(選擇性) 決定陰影服務要求之速率限制的設定。

此物件包含下列資訊。

`maxOutboundSyncUpdatesPerSecond`

(選擇性) 裝置每秒傳輸的同步處理要求數目上限。

預設值：每秒 100 個要求

`maxTotalLocalRequestsRate`

(選擇性) 每秒傳送至核心裝置的本機 IPC 要求數目上限。

預設值：每秒 200 個要求

`maxLocalRequestsPerSecondPerThing`

(選擇性) 針對每個連線 IoT 物件，每秒傳送的本機 IPC 要求數目上限。

預設值：每個物件每秒 20 次要求

> **Note**
>
> 這些速率限制參數定義本機陰影服務每秒的要求數目上限。 AWS IoT Device Shadow 服務每秒的要求數目上限取決於您的 AWS 區域。如需詳細資訊，AWS IoT Device Shadow 參閱 Amazon Web Services 一般參考.

`shadowDocumentSizeLimitBytes`

(選擇性) 本機陰影每個 JSON 狀態文件允許的大小上限。

如果增加此值，您也必須增加雲陰影 JSON 狀態文件的資源限制。如需詳細資訊，AWS IoT Device Shadow 參閱 Amazon Web Services 一般參考.

預設值：8192 個位元組

最大位元組

Example 範例：組態合併更新

下列範例顯示範例組態合併更新，其中包含陰影管理員元件的所有可用組態參數。

```
{
   "strategy":{
      "type":"periodic",
      "delay":300
   },
   "synchronize":{
      "shadowDocumentsMap":{
         "MyDevice1":{
            "classic":false,
            "namedShadows":[
               "MyShadowA",
               "MyShadowB"
            ]
         },
         "MyDevice2":{
            "classic":true,
            "namedShadows":[]
         }
      },
      "direction":"betweenDeviceAndCloud"
   },
   "rateLimits":{
      "maxOutboundSyncUpdatesPerSecond":100,
      "maxTotalLocalRequestsRate":200,
      "maxLocalRequestsPerSecondPerThing":20
   },
   "shadowDocumentSizeLimitBytes":8192
}
```

2.2.x

`strategy`

(選擇性) 此元件用來在 AWS IoT Core 與核心裝置之間同步陰影的策略。

此物件包含下列資訊。

`type`

(選擇性) 此元件用來同步處理 AWS IoT Core 與核心裝置之間陰影的策略類型。您可以從以下選項中選擇：

- `realTime`— AWS IoT Core 每次陰影更新時都會與陰影同步。
- `periodic`— 與您使用 AWS IoT Core `delay`組態參數指定的定期間隔同步陰影。

預設：`realTime`

`delay`

(選擇性) 當您指定`periodic`同步策略時 AWS IoT Core，此元件會與陰影同步的間隔 (以秒為單位)。

> **Note**
>
> 如果您指定`periodic`同步策略，則需要此參數。

`synchronize`

(選擇性) 決定陰影與的同步化設定 AWS 雲端。

> **Note**
>
> 您必須使用此屬性建立模型組態更新，才能與陰影同步 AWS 雲端。

此物件包含下列資訊。

`coreThing`

(選擇性) 要同步的核心裝置陰影。此物件包含下列資訊。

`classic`

(選擇性) 根據預設，陰影管理員會將核心裝置的傳統陰影的本機狀態與. AWS 雲端如果您不想同步傳統裝置陰影，請將其設定為`false`。

預設：`true`

namedShadows

(選擇性) 要同步的具名核心裝置陰影清單。您必須指定陰影的確切名稱。

> **Warning**
>
> 此 AWS IoT Greengrass 服務會使用AWSManagedGreengrassV2Deployment具名的陰影來管理以個別核心裝置為目標的部署。此具名陰影會保留供 AWS IoT Greengrass 服務使用。請勿更新或刪除此命名陰影。

shadowDocumentsMap

(選擇性) 要同步的其他裝置陰影。使用此組態參數可讓您更輕鬆地指定陰影文件。我們建議您使用此參數而不是shadowDocuments物件。

> **Note**
>
> 如果您指定shadowDocumentsMap物件，則不得指定shadowDocuments物件。

每個物件都包含下列資訊：

thingName

此陰影組態之 *thingName* 的陰影組態。

classic

(選擇性) 如果您不想同步裝置的傳統裝置陰影，請將此設定為false。thingName

namedShadows

您要同步的已命名陰影清單。您必須指定陰影的確切名稱。

shadowDocuments

(選擇性) 要同步的其他裝置陰影清單。我們建議您改用shadowDocumentsMap參數。

> **Note**
>
> 如果您指定shadowDocuments物件，則不得指定shadowDocumentsMap物件。

此清單中的每個物件都包含下列資訊。

`thingName`

要同步陰影之裝置的物件名稱。

`classic`

(選擇性) 如果您不想同步裝置的傳統裝置陰影，請將此設定為`false`。`thingName`

預設：`true`

`namedShadows`

(選擇性) 您要同步的已命名裝置陰影清單。您必須指定陰影的確切名稱。

`direction`

(選擇性) 在本機陰影服務與 AWS 雲端. 您可以設定此選項以減少頻寬和與 AWS 雲端. 您可以從以下選項中選擇：

- `betweenDeviceAndCloud`— 同步處理本機陰影服務與 AWS 雲端.
- `deviceToCloud`— 將陰影更新從本機陰影服務傳送至 AWS 雲端，並忽略來自 AWS 雲端.
- `cloudToDevice`— 從接收陰影更新 AWS 雲端，並且不要將陰影更新從本機陰影服務傳送到 AWS 雲端.

預設：`BETWEEN_DEVICE_AND_CLOUD`

`rateLimits`

(選擇性) 決定陰影服務要求之速率限制的設定。

此物件包含下列資訊。

`maxOutboundSyncUpdatesPerSecond`

(選擇性) 裝置每秒傳輸的同步處理要求數目上限。

預設值：每秒 100 個要求

`maxTotalLocalRequestsRate`

(選擇性) 每秒傳送至核心裝置的本機 IPC 要求數目上限。

預設值：每秒 200 個要求

`maxLocalRequestsPerSecondPerThing`

(選擇性) 針對每個連線 IoT 物件，每秒傳送的本機 IPC 要求數目上限。

預設值：每個物件每秒 20 次要求

> **Note**
>
> 這些速率限制參數定義本機陰影服務每秒的要求數目上限。 AWS IoT Device Shadow 服務每秒的要求數目上限取決於您的 AWS 區域。如需詳細資訊，AWS IoT Device Shadow 參閱 Amazon Web Services 一般參考.

`shadowDocumentSizeLimitBytes`

(選擇性) 本機陰影每個 JSON 狀態文件允許的大小上限。

如果增加此值，您也必須增加雲陰影 JSON 狀態文件的資源限制。如需詳細資訊，AWS IoT Device Shadow 參閱 Amazon Web Services 一般參考.

預設值：8192 個位元組

最大位元組

Example 範例：組態合併更新

下列範例顯示範例組態合併更新，其中包含陰影管理員元件的所有可用組態參數。

```
{
   "strategy":{
      "type":"periodic",
      "delay":300
   },
   "synchronize":{
      "shadowDocumentsMap":{
         "MyDevice1":{
            "classic":false,
            "namedShadows":[
               "MyShadowA",
               "MyShadowB"
            ]
```

```
         },
         "MyDevice2":{
            "classic":true,
            "namedShadows":[]
         }
      },
      "direction":"betweenDeviceAndCloud"
   },
   "rateLimits":{
      "maxOutboundSyncUpdatesPerSecond":100,
      "maxTotalLocalRequestsRate":200,
      "maxLocalRequestsPerSecondPerThing":20
   },
   "shadowDocumentSizeLimitBytes":8192
}
```

2.1.x

`strategy`

(選擇性) 此元件用來在 AWS IoT Core 與核心裝置之間同步陰影的策略。

此物件包含下列資訊。

`type`

(選擇性) 此元件用來同步處理 AWS IoT Core 與核心裝置之間陰影的策略類型。您可以從以下選項中選擇：

- `realTime`— AWS IoT Core 每次陰影更新時都會與陰影同步。
- `periodic`— 與您使用 AWS IoT Core `delay`組態參數指定的定期間隔同步陰影。

預設：`realTime`

`delay`

(選擇性) 當您指定`periodic`同步策略時 AWS IoT Core，此元件會與陰影同步的間隔 (以秒為單位)。

> **Note**
>
> 如果您指定`periodic`同步策略，則需要此參數。

`synchronize`

(選擇性) 決定陰影與的同步化設定 AWS 雲端。

> **Note**
>
> 您必須使用此屬性建立模型組態更新，才能與陰影同步 AWS 雲端。

此物件包含下列資訊。

`coreThing`

(選擇性) 要同步的核心裝置陰影。此物件包含下列資訊。

`classic`

(選擇性) 根據預設，陰影管理員會將核心裝置的傳統陰影的本機狀態與. AWS 雲端如果您不想同步傳統裝置陰影，請將其設定為`false`。

預設：`true`

`namedShadows`

(選擇性) 要同步的具名核心裝置陰影清單。您必須指定陰影的確切名稱。

> **Warning**
>
> 此 AWS IoT Greengrass 服務會使用`AWSManagedGreengrassV2Deployment`具名的陰影來管理以個別核心裝置為目標的部署。此具名陰影會保留供 AWS IoT Greengrass 服務使用。請勿更新或刪除此命名陰影。

`shadowDocumentsMap`

(選擇性) 要同步的其他裝置陰影。使用此組態參數可讓您更輕鬆地指定陰影文件。我們建議您使用此參數而不是`shadowDocuments`物件。

> **Note**
>
> 如果您指定`shadowDocumentsMap`物件，則不得指定`shadowDocuments`物件。

每個物件都包含下列資訊：

thingName

此陰影組態之 *thingName* 的陰影組態。

classic

(選擇性) 如果您不想同步裝置的傳統裝置陰影，請將此設定為false。thingName

namedShadows

您要同步的已命名陰影清單。您必須指定陰影的確切名稱。

shadowDocuments

(選擇性) 要同步的其他裝置陰影清單。我們建議您改用shadowDocumentsMap參數。

> **Note**
> 如果您指定shadowDocuments物件，則不得指定shadowDocumentsMap物件。

此清單中的每個物件都包含下列資訊。

thingName

要同步陰影之裝置的物件名稱。

classic

(選擇性) 如果您不想同步裝置的傳統裝置陰影，請將此設定為false。thingName

預設：true

namedShadows

(選擇性) 您要同步的已命名裝置陰影清單。您必須指定陰影的確切名稱。

rateLimits

(選擇性) 決定陰影服務要求之速率限制的設定。

此物件包含下列資訊。

maxOutboundSyncUpdatesPerSecond

(選擇性) 裝置每秒傳輸的同步處理要求數目上限。

預設值：每秒 100 個要求

`maxTotalLocalRequestsRate`

(選擇性) 每秒傳送至核心裝置的本機 IPC 要求數目上限。

預設值：每秒 200 個要求

`maxLocalRequestsPerSecondPerThing`

(選擇性) 針對每個連線 IoT 物件，每秒傳送的本機 IPC 要求數目上限。

預設值：每個物件每秒 20 次要求

> **Note**
>
> 這些速率限制參數定義本機陰影服務每秒的要求數目上限。 AWS IoT Device Shadow 服務每秒的要求數目上限取決於您的 AWS 區域。如需詳細資訊，AWS IoT Device Shadow 參閱 Amazon Web Services 一般參考.

`shadowDocumentSizeLimitBytes`

(選擇性) 本機陰影每個 JSON 狀態文件允許的大小上限。

如果增加此值，您也必須增加雲陰影 JSON 狀態文件的資源限制。如需詳細資訊，AWS IoT Device Shadow 參閱 Amazon Web Services 一般參考.

預設值：8192 個位元組

最大位元組

Example 範例：組態合併更新

下列範例顯示範例組態合併更新，其中包含陰影管理員元件的所有可用組態參數。

```
{
   "strategy":{
      "type":"periodic",
      "delay":300
   },
   "synchronize":{
      "shadowDocumentsMap":{
         "MyDevice1":{
```

```
            "classic":false,
            "namedShadows":[
               "MyShadowA",
               "MyShadowB"
            ]
         },
         "MyDevice2":{
            "classic":true,
            "namedShadows":[]
         }
      },
      "direction":"betweenDeviceAndCloud"
   },
   "rateLimits":{
      "maxOutboundSyncUpdatesPerSecond":100,
      "maxTotalLocalRequestsRate":200,
      "maxLocalRequestsPerSecondPerThing":20
   },
   "shadowDocumentSizeLimitBytes":8192
}
```

2.0.x

`synchronize`

(選擇性) 決定陰影與的同步化設定 AWS 雲端。

> **Note**
>
> 您必須使用此屬性建立模型組態更新，才能與陰影同步 AWS 雲端。

此物件包含下列資訊。

`coreThing`

(選擇性) 要同步的核心裝置陰影。此物件包含下列資訊。

`classic`

(選擇性) 根據預設，陰影管理員會將核心裝置的傳統陰影的本機狀態與. AWS 雲端如果您不想同步傳統裝置陰影，請將其設定為`false`。

預設：`true`

namedShadows

(選擇性) 要同步的具名核心裝置陰影清單。您必須指定陰影的確切名稱。

> **Warning**
>
> 此 AWS IoT Greengrass 服務會使用AWSManagedGreengrassV2Deployment具名的陰影來管理以個別核心裝置為目標的部署。此具名陰影會保留供 AWS IoT Greengrass 服務使用。請勿更新或刪除此命名陰影。

shadowDocumentsMap

(選擇性) 要同步的其他裝置陰影。使用此組態參數可讓您更輕鬆地指定陰影文件。我們建議您使用此參數而不是shadowDocuments物件。

> **Note**
>
> 如果您指定shadowDocumentsMap物件，則不得指定shadowDocuments物件。

每個物件都包含下列資訊：

thingName

此陰影組態之 *thingName* 的陰影組態。

classic

(選擇性) 如果您不想同步裝置的傳統裝置陰影，請將此設定為false。thingName

namedShadows

您要同步的已命名陰影清單。您必須指定陰影的確切名稱。

shadowDocuments

(選擇性) 要同步的其他裝置陰影清單。我們建議您改用shadowDocumentsMap參數。

> **Note**
>
> 如果您指定shadowDocuments物件，則不得指定shadowDocumentsMap物件。

此清單中的每個物件都包含下列資訊。

`thingName`

要同步陰影之裝置的物件名稱。

`classic`

(選擇性) 如果您不想同步裝置的傳統裝置陰影，請將此設定為`false`。`thingName`

預設：`true`

`namedShadows`

(選擇性) 您要同步的已命名裝置陰影清單。您必須指定陰影的確切名稱。

`rateLimits`

(選擇性) 決定陰影服務要求之速率限制的設定。

此物件包含下列資訊。

`maxOutboundSyncUpdatesPerSecond`

(選擇性) 裝置每秒傳輸的同步處理要求數目上限。

預設值：每秒 100 個要求

`maxTotalLocalRequestsRate`

(選擇性) 每秒傳送至核心裝置的本機 IPC 要求數目上限。

預設值：每秒 200 個要求

`maxLocalRequestsPerSecondPerThing`

(選擇性) 針對每個連線 IoT 物件，每秒傳送的本機 IPC 要求數目上限。

預設值：每個物件每秒 20 次要求

> **Note**
>
> 這些速率限制參數定義本機陰影服務每秒的要求數目上限。 AWS IoT Device Shadow 服務每秒的要求數目上限取決於您的 AWS 區域。如需詳細資訊，AWS IoT Device Shadow 參閱 Amazon Web Services 一般參考.

`shadowDocumentSizeLimitBytes`

(選擇性) 本機陰影每個 JSON 狀態文件允許的大小上限。

如果增加此值，您也必須增加雲陰影 JSON 狀態文件的資源限制。如需詳細資訊，AWS IoT Device Shadow 參閱 Amazon Web Services 一般參考.

預設值：8192 個位元組

最大位元組

Example 範例：組態合併更新

下列範例顯示範例組態合併更新，其中包含陰影管理員元件的所有可用組態參數。

```
{
  "synchronize": {
    "coreThing": {
      "classic": true,
      "namedShadows": [
        "MyCoreShadowA",
        "MyCoreShadowB"
      ]
    },
    "shadowDocuments": [
      {
        "thingName": "MyDevice1",
        "classic": false,
        "namedShadows": [
          "MyShadowA",
          "MyShadowB"
        ]
      },
      {
        "thingName": "MyDevice2",
        "classic": true,
        "namedShadows": []
      }
    ]
  },
  "rateLimits": {
    "maxOutboundSyncUpdatesPerSecond": 100,
    "maxTotalLocalRequestsRate": 200,
```

```
    "maxLocalRequestsPerSecondPerThing": 20
  },
  "shadowDocumentSizeLimitBytes": 8192
}
```

本機記錄檔

此元件使用與 Greengrass 核元件相同的記錄檔。

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

若要檢視此元件的記錄

- 在核心裝置上執行下列命令，即時檢視此元件的記錄檔。以 AWS IoT Greengrass 根資料夾的路徑取代*/greengrass/v2*或 *C:\greengrass\v2*。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

變更記錄

下表說明元件每個版本的變更。

版本	變更
2.3.7	錯誤修復和改進 • 修正陰影管理員在陰影管理員同步處理期間定期記錄`NullPointerException` 錯誤的問題。
2.3.6	錯誤修復和改進 • 修正當裝置離線時透過 AWS 雲端 更新刪除的陰影屬性在重新取得連線後，仍會繼續存在於本機陰影中的問題。
2.3.5	版本更新了 Greengrass 核 2.12.0 版本釋放。
2.3.4	錯誤修復和改進 • 添加對空白和空陰影狀態文檔的支持。
2.3.3	版本更新了 Greengrass 核 2.11.0 版本釋放。
2.3.2	錯誤修復和改進 • 修正當本機陰影資料庫損毀時，陰影管理員進入`BROKEN`狀態的問題。 • 版本更新了 Greengrass 核 2.10.0 版本釋放。
2.3.1	錯誤修復和改進 • 修正可能導致雲陰影更新無法同步的情況。 • 修正具名陰影同步組態的變更只會套用至一個具名陰影的問題。
2.3.0	錯誤修復和改進 • 修正當 Greengrass 裝置私密金鑰儲存在硬體安全性模組中時，可能無法同步陰影的問題。
2.2.4	錯誤修復和改進 • 修正更新本機陰影文件時，陰影大小驗證與雲端不一致的問題。 • 修正當部署`RESET`在組態節點上執行時，陰影管理員會停止接聽組態更新的問題。
2.2.3	版本更新 Greengrass 2.9.0 版本釋放。
2.2.2	版本更新 Greengrass 2.8.0 版本的版本。

版本	變更
2.2.1	版本更新了 Greengrass 核 2.7.0 版本釋放。
2.2.0	新功能 • 通過本地發布/訂閱界面添加對本地陰影服務的支持。您現在可以在陰影 MQTT 主題上與本機發佈/訂閱訊息代理程式進行通訊，以取得、更新和刪除核心裝置上的陰影。此功能可讓您使用 MQTT 橋接器在用戶端裝置與本機發佈/訂閱介面之間轉送陰影主題的訊息，將用戶端裝置連線至本機陰影服務。 此功能需要 v2.6.0 或更高版 Greeng rass 核成分。若要將用戶端裝置連線到本機陰影服務，您也必須使用 v2.2.0 或更新版本的 MQ TT 橋接器元件。 • 新增您可以設定的`direction` 選項，以自訂方向，以便在本機陰影服務與之間同步陰影 AWS 雲端。您可以設定此選項以減少頻寬和與 AWS 雲端.
2.1.1	錯誤修復和改進 • 修正了 JSON 裝置陰影狀態文件`desired`和`reported`區段中最大深度為 4 個層級而非 5 個層級的問題。 • 版本更新 Greengrass 2.6.0 版本的版本。
2.1.0	新功能 • 新增對定期陰影同步處理間隔的支援，因此您可以設定核心裝置以減少頻寬使用量和費用。
2.0.6	此版本包含錯誤修復和改進。
2.0.5	版本更新了 Greengrass 核 2.5.0 版本。
2.0.4	錯誤修復和改進 • 修正造成陰影管理員刪除先前刪除之任何陰影的新建立版本的問題。 • 更新 `DeleteThingShadow` IPC 作業，以在呼叫時增加陰影版本。
2.0.3	版本更新 Greengrass 2.4.0 版本的版本。

版本	變更
2.0.2	錯誤修復和改進 • 修正在同步陰影狀態時，陰影管理員無法辨識`delta`屬性的問題 AWS IoT Core。 • 修正有時會導致陰影的同步處理要求合併不正確的問題。
2.0.1	版本更新了 Greengrass 核 2.3.0 版本。
2.0.0	初始版本。

Amazon SNS

Amazon SNS 元件 (`aws.greengrass.SNS`) 會將訊息發佈到 Amazon Simple Notification Service (Amazon SNS) 主題。您可以使用此元件將事件從 Greengrass 核心裝置傳送至 Web 伺服器、電子郵件地址和其他訊息訂閱者。如需詳細資訊，請參閱什麼是 Amazon SNS？ 在 Amazon 簡單通知服務開發人員指南中。

若要使用此元件發佈到 Amazon SNS 主題，請將訊息發佈到此元件訂閱的主題。依預設，此元件會訂閱`sns/message`本機發佈/訂閱主題。您可以在部署此元件時指定其他主題，包括 AWS IoT Core MQTT 主題。

在您的自訂元件中，您可能想要實作篩選或格式化邏輯，以處理來自其他來源的郵件，然後再將它們發佈到此元件。這可讓您將訊息處理邏輯集中在單一元件上。

> **Note**
>
> 此元件提供與 AWS IoT Greengrass V1 中 Amazon SNS 連接器類似的功能。如需詳細資訊，請參閱 AWS IoT Greengrass V1 開發人員指南中的 Amazon SNS 連接器。

主題

- 相依性
- 組態
- 輸入資料
- 輸出資料
- 本機記錄檔
- 授權
- 變更記錄

版本

此元件具有下列版本：

- 2.1.x
- 2.0.x

Type

這個組件是一個 Lambda 組件（`aws.greengrass.lambda`）。Greengrass 核使用 Lambda 啟動器組件運行此組件的 Lambda 函數。

如需詳細資訊，請參閱 元件類型。

作業系統

此元件只能安裝在 Linux 核心裝置上。

要求

此元件具有下列需求：

- 您的核心裝置必須符合執行 Lambda 函數的需求。如果您希望核心裝置執行容器化 Lambda 函數，則該裝置必須符合要求才能執行此操作。如需詳細資訊，請參閱 Lambda 函數要求。
- Python 版本 3.7 安裝在核心設備上，並添加到 PATH 環境變量中。
- Amazon 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。

- Greengrass 裝置角色必須允許`sns:Publish`動作，如下列 IAM 政策範例所示。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "sns:Publish"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:sns:region:account-id:topic-name"
      ]
    }
  ]
}
```

  您可以動態覆寫此元件之輸入訊息承載中的預設主題。如果您的應用程式使用此功能，IAM 政策必須包含所有目標主題作為資源。您可以為資源授予精細或條件式存取 (例如，使用萬用字元 * 命名機制)。

- 若要從此元件接收輸出資料，您必須在部署此元件時，合併舊版訂閱路由器元件的下列組態更新 (`aws.greengrass.LegacySubscriptionRouter`)。此配置指定此組件發布響應的主題。

  Legacy subscription router v2.1.x

```
{
  "subscriptions": {
    "aws-greengrass-sns": {
      "id": "aws-greengrass-sns",
      "source": "component:aws.greengrass.SNS",
      "subject": "sns/message/status",
      "target": "cloud"
    }
  }
}
```

  Legacy subscription router v2.0.x

```
{
  "subscriptions": {
    "aws-greengrass-sns": {
      "id": "aws-greengrass-sns",
```

```
      "source": "arn:aws:lambda:region:aws:function:aws-greengrass-sns:version",
      "subject": "sns/message/status",
      "target": "cloud"
    }
  }
}
```

- 將*##*替換為您 AWS 區域 使用的區域。
- 將*##*取代為此元件執行的 Lambda 函數版本。若要尋找 Lambda 函數版本，您必須檢視要部署之此元件版本的方案。在AWS IoT Greengrass 主控台中開啟此元件的詳細資料頁面，然後尋找 Lambda 函數索引鍵值組。此鍵值對包含 Lambda 函數的名稱和版本。

> **Important**
>
> 每次部署此元件時，您都必須更新舊版訂閱路由器上的 Lambda 函數版本。這可確保您針對部署的元件版本使用正確的 Lambda 函數版本。

如需詳細資訊，請參閱 建立部署。

- 支援 Amazon SNS 元件在虛擬私人 VPC 中執行。若要在 VPC 中部署此元件，需要下列項目。
  - Amazon SNS 元件必須具有的 VPC 擬私人雲端端點的`com.amazonaws.us-east-1.sns`連線能力。`sns.region.amazonaws.com`

端點和連接埠

除了基本作業所需的端點和連接埠之外，此元件還必須能夠對下列端點和連接埠執行輸出要求。如需詳細資訊，請參閱 允許裝置流量透過 Proxy 或防火牆。

端點	連線埠	必要	描述
`sns.region.amazonaws.com`	443	是	將訊息發佈到 Amazon SNS。

相依性

部署元件時， AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的需求，才能成功部署元件。本節列出此元件之已發行版本的相依性，以及定義每個相依性之元件版本的語意版本條件約束。您也可以在AWS IoT Greengrass 主控台中檢視元件每個版本的相依性。在元件詳細資料頁面上，尋找 [相依性] 清單。

2.1.7

下表列出此元件 2.1.7 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	硬式
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
代幣交換服務	^2.0.0	硬式

2.1.6

下表列出此元件 2.1.6 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	硬式
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
代幣交換服務	^2.0.0	硬式

2.1.5

下表列出此元件 2.1.5 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	硬式
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
代幣交換服務	^2.0.0	硬式

2.1.4

下表列出此元件 2.1.4 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	硬式
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
代幣交換服務	^2.0.0	硬式

2.1.3

下表列出此元件 2.1.3 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	硬式
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
代幣交換服務	^2.0.0	硬式

2.1.2

下表列出此元件 2.1.2 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	硬式
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
代幣交換服務	^2.0.0	硬式

2.1.1

下表列出此元件 2.1.1 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	硬式
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
代幣交換服務	^2.0.0	硬式

2.0.8 - 2.1.0

下表列出此元件 2.0.8 和 2.1.0 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	硬式
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式

相依性	兼容版本	相依性類型
代幣交換服務	^2.0.0	硬式

2.0.7

下表列出此元件 2.0.7 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	硬式
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
代幣交換服務	^2.0.0	硬式

2.0.6

下表列出此元件 2.0.6 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	硬式
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
代幣交換服務	^2.0.0	硬式

2.0.5

下表列出此元件 2.0.5 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	硬式
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
代幣交換服務	^2.0.0	硬式

2.0.4

下表列出此元件 2.0.4 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	硬式
发 Lambda 器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
代幣交換服務	^2.0.0	硬式

2.0.3

下表列出此元件 2.0.3 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.3	硬式
发 Lambda 器	>	硬式
Lambda 執行階段	>	軟式
代幣交換服務	>	硬式

如需有關元件相依性的詳細資訊，請參閱元件方案參考。

組態

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

> **Note**
>
> 此元件的預設組態包括 Lambda 函數參數。建議您只編輯下列參數，以便在裝置上設定此元件。

`lambdaParams`

包含此元件 Lambda 函數之參數的物件。此物件包含下列資訊：

`EnvironmentVariables`

包含 Lambda 函數參數的物件。此物件包含下列資訊：

`DEFAULT_SNS_ARN`

此元件會在其中發佈訊息的預設 Amazon SNS 主題的 ARN。您可以使用輸入訊息承載中的`sns_topic_arn`屬性覆寫目標主題。

`containerMode`

(選擇性) 此元件的容器化模式。您可以從以下選項中選擇：

- `NoContainer`— 元件不會在隔離的執行階段環境中執行。
- `GreengrassContainer`— 元件在 AWS IoT Greengrass 容器內的隔離執行階段環境中執行。

預設：`GreengrassContainer`

`containerParams`

(選擇性) 包含此元件之容器參數的物件。如果您指定`GreengrassContainer`為，則元件會使用這些參數`containerMode`。

此物件包含下列資訊：

`memorySize`

(選擇性) 配置給元件的記憶體容量 (以 KB 為單位)。

預設值為 512 MB (525,312 KB)。

`pubsubTopics`

(選擇性) 包含元件訂閱以接收訊息之主題的物件。您可以指定每個主題，以及元件是否訂閱 MQTT 主題 AWS IoT Core 或本機發佈/訂閱主題。

此物件包含下列資訊：

`0`-這是一個數組索引作為一個字符串。

包含下列資訊的物件：

`type`

(選擇性) 此元件用來訂閱訊息的發佈/訂閱訊息類型。您可以從以下選項中選擇：

- PUB_SUB - 訂閱本機發佈/訂閱訊息。如果您選擇此選項，則主題不能包含 MQTT 萬用字元。如需指定此選項時如何從自訂元件傳送郵件的詳細資訊，請參閱發佈/訂閱本地訊息。
- `IOT_CORE`— 訂閱 AWS IoT Core MQTT 訊息。如果您選擇此選項，主題可以包含 MQTT 萬用字元。如需指定此選項時如何從自訂元件傳送郵件的詳細資訊，請參閱發布/訂閱 MQTT 訊 AWS IoT Core 息。

預設：PUB_SUB

`topic`

(選擇性) 元件訂閱接收訊息的主題。如果您指定`IotCore`為`type`，您可以在本主題中使用 MQTT 萬用字元 (+和#)。

Example 範例：組態合併更新 (容器模式)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "DEFAULT_SNS_ARN": "arn:aws:sns:us-west-2:123456789012:mytopic"
    }
  },
  "containerMode": "GreengrassContainer"
}
```

Example 範例：組態合併更新 (無容器模式)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
```

```
      "DEFAULT_SNS_ARN": "arn:aws:sns:us-west-2:123456789012:mytopic"
    }
  },
  "containerMode": "NoContainer"
}
```

輸入資料

此元件接受有關以下主題的訊息，並將訊息依原樣發佈到目標 Amazon SNS 主題。根據預設，此元件會訂閱本機發佈/訂閱訊息。如需如何從自訂元件將訊息發佈至此元件的詳細資訊，請參閱發佈/訂閱本地訊息。

默認主題（本地發布/訂閱）：`sns/message`

該消息接受以下屬性。輸入訊息必須為 JSON 格式。

`request`

要傳送至 Amazon SNS 主題之訊息的相關資訊。

類型：`object`包含下列資訊：

`message`

以字串形式顯示的訊息內容。

若要傳送 JSON 物件，請將其序列化為字串，然後`json`為`message_structure`屬性指定。

類型：`string`

`subject`

(選擇性) 郵件的主旨。

類型：`string`

主旨可以是 ASCII 文字和最多 100 個字元。它必須以字母、數字或標點符號開頭。它不能包含換行符或控制字符。

`sns_topic_arn`

(選擇性) 此元件在其中發佈訊息的 Amazon SNS 主題的 ARN。指定此屬性可覆寫預設的 Amazon SNS 主題。

類型：`string`

`message_structure`

(選擇性) 訊息的結構。指定`json`以傳送您在`content`屬性中序列化為字串的 JSON 訊息。

類型：`string`

有效值：`json`

`id`

請求的任意 ID。使用此屬性可將輸入要求對應至輸出回應。當您指定此屬性時，組件會將回應物件中的`id`屬性設定為此值。

類型：`string`

> **Note**
>
> 郵件大小最多可以是 256 KB。

Example 範例輸入：字串訊息

```
{
  "request": {
    "subject": "Message subject",
    "message": "Message data",
    "sns_topic_arn": "arn:aws:sns:region:account-id:topic2-name"
  },
  "id": "request123"
}
```

Example 範例輸入：JSON 訊息

```
{
  "request": {
    "subject": "Message subject",
    "message": "{ \"default\": \"Message data\" }",
    "message_structure": "json"
  },
  "id": "request123"
}
```

輸出資料

依預設，此元件會將回應發佈為下列 MQTT 主題的輸出資料。您必須將此主題指定為舊版訂閱路由器元件的組態subject中的。如需如何在自訂元件中訂閱有關此主題之訊息的詳細資訊，請參閱發布/訂閱MQTT 訊 AWS IoT Core 息。

預設主題 (AWS IoT Core MQTT)：sns/message/status

Example 範例輸出：成功

```
{
  "response": {
    "sns_message_id": "f80a81bc-f44c-56f2-a0f0-d5af6a727c8a",
    "status": "success"
  },
  "id": "request123"
}
```

Example 範例輸出：失敗

```
{
  "response" : {
    "error": "InvalidInputException",
    "error_message": "SNS Topic Arn is invalid",
    "status": "fail"
  },
  "id": "request123"
}
```

本機記錄檔

此元件使用下列記錄檔。

```
/greengrass/v2/logs/aws.greengrass.SNS.log
```

若要檢視此元件的記錄

- 在核心裝置上執行下列命令，即時檢視此元件的記錄檔。以 AWS IoT Greengrass 根資料夾的路徑取*/greengrass/v2*代。

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.SNS.log
  ```

授權

此元件包括下列協力廠商軟體/授權：

- AWS SDK for Python (Boto3)/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

此元件是根據 Greengrass 核心軟體授權合約發行的。

變更記錄

下表說明元件每個版本的變更。

版本	變更
2.1.7	版本更新 Greengrass 2.12.0 版本釋放。
2.1.6	版本更新 Greengrass 2.11.0 版本釋放。
2.1.5	版本更新了 Greengrass 2.10.0 版本。
2.1.4	版本更新 Greengrass 2.9.0 版本釋放。
2.1.3	版本更新 Greengrass 2.8.0 版本的版本。
2.1.2	版本更新了 Greengrass 核 2.7.0 版本釋放。
2.1.1	版本更新 Greengrass 2.6.0 版本發布。
2.1.0	新功能 • 添加對 HTTPS 網絡代理配置的支持。如需詳細資訊，請參閱 連線至連接埠 443 或透過網路代理 及 讓核心裝置信任 HTTPS 代理。

版本	變更
2.0.8	版本更新了 Greengrass 核 2.5.0 版本。
2.0.7	版本更新 Greengrass 2.4.0 版本的版本。
2.0.6	版本更新了 Greengrass 核 2.3.0 版本。
2.0.5	版本更新了 Greengrass 核 2.2.0 版本。
2.0.4	版本更新 Greengrass 2.1.0 版本發布。
2.0.3	初始版本。

串流管理員

串流管理員元件 (`aws.greengrass.StreamManager`) 可讓您處理資料串流，以便 AWS 雲端 從 Greengrass 核心裝置傳輸到。

如需如何在自訂元件中設定和使用串流管理員的詳細資訊，請參閱管理核心裝 Greengrass 資料串流。

主題

- 版本
- Type
- 作業系統
- 要求
- 相依性
- 組態
- 本機記錄檔
- 變更記錄

版本

此元件具有下列版本：

- 2.1.x

- 2.0.x

> **Note**
>
> 如果您使用串流管理員將資料匯出至雲端，則無法將串流管理員元件的 2.0.7 版升級為 v2.0.8 和 v2.0.11 之間的版本。如果您是第一次部署串流管理員，強烈建議您部署最新版本的串流管理員元件。

Type

此元件是一般元件 (`aws.greengrass.generic`)。Greengrass 核會執行元件的生命週期指令碼。

如需詳細資訊，請參閱 元件類型。

作業系統

此元件可安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件具有下列需求：

- T oken 交換角色必須允許存取您搭配串流管理員使用的 AWS 雲端 目的地。如需詳細資訊，請參閱：
  - the section called “AWS IoT Analytics 頻道”
  - the section called “Amazon Kinesis 數據流”
  - the section called “AWS IoT SiteWise資產性質”
  - the section called “Amazon S3 對象”
- 支援串流管理員元件在 VPC 中執行。若要在 VPC 中部署此元件，需要下列項目。
  - 串流管理員元件必須與您發佈資料的目標 AWS 服務具有連線能力。
    - Amazon S3：`com.amazonaws.region.s3`
    - Amazon Kinesis Data Streams：`com.amazonaws.region.kinesis-streams`

  - AWS IoT SiteWise: `com.amazonaws.region.iotsitewise.data`
- 如果您將資料發佈到`us-east-1`區域中的 Amazon S3，則此元件預設會嘗試使用 S3 全球端點；不過，此端點無法透過 Amazon S3 VPC 界面端點使用。如需詳細資訊，請參閱 Amazon S3 AWS PrivateLink 的限制和限制。若要解決此問題，您可以從下列選項中進行選擇。
  - 透過提供`AWS_S3_US_EAST_1_REGIONAL_ENDPOINT=regional`環境變數，將串流管理員元件設定為使用`us-east-1`區域中的區域 S3 端點。
  - 建立 Amazon S3 閘道 VPC 端點，而不是 Amazon S3 界面 VPC 端點。S3 閘道端點支援對 S3 全球端點的存取。如需詳細資訊，請參閱建立閘道端點。

端點和連接埠

除了基本作業所需的端點和連接埠之外，此元件還必須能夠對下列端點和連接埠執行輸出要求。如需詳細資訊，請參閱 允許裝置流量透過 Proxy 或防火牆。

端點	連線埠	必要	描述
`iotanalytics.region.amazonaws.com`	443	否	如果您將資料發佈到，則為必要 AWS IoT Analytics。
`kinesis.region.amazonaws.com`	443	否	如果您將資料發佈至 Firehose，則需要此項。
`data.iotsitewise.region.amazonaws.com`	443	否	如果您將資料發佈到，則為必要 AWS IoT SiteWise。
`*.s3.amazonaws.com`	443	否	如果將資料發佈到 S3

端點	連線埠	必要	描述
			儲存貯體，則為必要項 您可以*用發佈資料的每個值區名稱取代。

相依性

部署元件時， AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的需求，才能成功部署元件。本節列出此元件之已發行版本的相依性，以及定義每個相依性之元件版本的語意版本條件約束。您也可以在AWS IoT Greengrass 主控台中檢視元件每個版本的相依性。在元件詳細資料頁面上，尋找 [相依性] 清單。

2.1.11

下表列出此元件 2.1.11 至 2.1.10 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
代幣交換服務	>	硬式

2.1.9 – 2.1.10

下表列出此元件 2.1.9 至 2.1.10 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
代幣交換服務	>	硬式

2.1.5 – 2.1.8

下表列出此元件 2.1.5 至 2.1.8 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
代幣交換服務	>	硬式

2.1.2 – 2.1.4

下表列出此元件 2.1.2 至 2.1.4 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
代幣交換服務	>	硬式

2.1.1

下表列出此元件 2.1.1 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
代幣交換服務	>	硬式

2.1.0

下表列出此元件 2.1.0 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式

相依性	兼容版本	相依性類型
代幣交換服務	>	硬式

2.0.15

下表列出此元件 2.0.15 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
代幣交換服務	>	硬式

2.0.13 and 2.0.14

下表列出此元件 2.0.13 和 2.0.14 版本的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	>=2.0.0	軟式
代幣交換服務	>	硬式

2.0.11 and 2.0.12

下表列出此元件 2.0.11 和 2.0.12 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
代幣交換服務	>	硬式

2.0.10

下表列出此元件 2.0.10 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
代幣交換服務	>	硬式

2.0.9

下表列出此元件 2.0.9 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
代幣交換服務	>	硬式

2.0.8

下表列出此元件 2.0.8 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.0	軟式
代幣交換服務	>	硬式

2.0.7

下表列出此元件 2.0.7 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.0.3	軟式
代幣交換服務	>	硬式

如需有關元件相依性的詳細資訊，請參閱元件方案參考。

組態

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

`STREAM_MANAGER_STORE_ROOT_DIR`

(選擇性) 用來儲存串流之本機目錄的絕對路徑。此值必須以正斜線開頭 (例如 `/data`)。

您必須指定現有的資料夾，而且執行串流管理員元件的系統使用者必須擁有讀取和寫入此資料夾的權限。例如，您可以執行下列命令來建立和設定資料夾`/var/greengrass/streams`，您可以將其指定為串流管理員根資料夾。這些指令允許預設系統使用者讀取和寫入此資料夾。`ggc_user`

```
sudo mkdir /var/greengrass/streams
sudo chown ggc_user /var/greengrass/streams
sudo chmod 700 /var/greengrass/streams
```

預設：*/greengrass/v2*/work/aws.greengrass.StreamManager

`STREAM_MANAGER_SERVER_PORT`

(選擇性) 用來與串流管理員通訊的本機連接埠號碼。

您可以指定`0`使用隨機可用的連接埠。

預設：8088

`STREAM_MANAGER_AUTHENTICATE_CLIENT`

(選擇性) 您可以強制要求用戶端進行驗證，然後才能與串流管理員互動。串流管理員 SDK 可控制用戶端和串流管理員之間的互動。這個參數決定哪些用戶端可以呼叫串流管理員 SDK 來處理串流。如需詳細資訊，請參閱串流管理員用戶端驗證。

如果您指定`true`，串流管理員 SDK 只允許 Greengrass 元件做為用戶端。

如果您指定`false`，串流管理員 SDK 會允許核心裝置上的所有處理序做為用戶端。

預設：`true`

`STREAM_MANAGER_EXPORTER_MAX_BANDWIDTH`

(選擇性) 串流管理員可用來匯出資料的平均最大頻寬 (以每秒 KB 為單位)。

預設：無限制

`STREAM_MANAGER_EXPORTER_THREAD_POOL_SIZE`

(選擇性) 串流管理員可用來匯出資料的作用中繫線數目上限。

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。每個串流都會耗用硬體資源，因此請嘗試限制受限裝置上的匯出串流數量。

預設值：5 個執行緒

`STREAM_MANAGER_EXPORTER_S3_DESTINATION_MULTIPART_UPLOAD_MIN_PART_SIZE_BYTES`

(選擇性) 多部分上傳至 Amazon S3 時，零件的大小下限 (以位元組為單位)。流管理器使用此設置和輸入文件的大小來確定如何批處理多部分 PUT 請求中的數據。

> **Note**
>
> 串流管理員使用串流`sizeThresholdForMultipartUploadBytes`屬性來判斷是以單一或多部分上傳方式匯出到 Amazon S3。 AWS IoT Greengrass 元件可以在建立匯出至 Amazon S3 的串流時設定此閾值。

預設值：`5242880`(5 MB)。這也是最小值。

`LOG_LEVEL`

(選擇性) 元件的記錄層級。從下列記錄層級中進行選擇，依層級順序列示在此處：

- `TRACE`
- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

預設：`INFO`

`JVM_ARGS`

(選擇性) 要在啟動時傳遞至串流管理員的自訂 Java 虛擬機器引數。用空格分隔多個引數。

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。

Example 範例：組態合併更新

下列範例組態指定使用非預設連接埠。

```
{
  "STREAM_MANAGER_SERVER_PORT": "18088"
}
```

本機記錄檔

此元件會使用下列記錄檔。

Linux

```
/greengrass/v2/logs/aws.greengrass.StreamManager.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.StreamManager.log
```

若要檢視此元件的記錄

- 在核心裝置上執行下列命令，即時檢視此元件的記錄檔。以 AWS IoT Greengrass 根資料夾的路徑取代*/greengrass/v2*或 *C:\greengrass\v2*。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.StreamManager.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.greengrass.StreamManager.log -Tail 10 -
  Wait
  ```

變更記錄

下表說明元件每個版本中的變更。

版本	變更
2.1.12	錯誤修復和改進 更新證明資料的使用順序，讓 Greengrass 證明資料偏好用於服務要求 AWS 。
2.1.11	版本更新了 Greengrass 核 2.12.0 版本釋放。
2.1.10	錯誤修復和改進 修正 HTTPS 代理伺服器設定不信任 Greengrass 憑證授權單位 (CA) 憑證鏈結的問題。
2.1.9	版本更新了 Greengrass 核 2.11.0 版本釋放。
2.1.8	錯誤修復和改進 修復了流管理器無限重試 SiteWise 導出失敗的問題。`InvalidRequestException`
2.1.7	錯誤修復和改進 修正串流管理員無法正確讀取 Proxy 組態的問題。
2.1.6	錯誤修復和改進 修正了在某些 ARMv8 處理器（包括 Jetson Nano）上啟動時可能導致當機的問題。
2.1.5	版本更新了 Greengrass 核 2.10.0 版本釋放。
2.1.4	錯誤修復和改進 • 修正 SiteWise API 的單一批次傳回`ConflictingOperationException` 中具有相同時間戳記的相同屬性資產項目，導致串流管理員持續重試的問題。

版本	變更
	• 將預設連線逾時從 3 秒更新為 1 分鐘。
2.1.3	錯誤修復和改進 修復以系統用戶身份運行時 Windows 操作系統上的啟動問題。
2.1.2	錯誤修復和改進 • 修正 Windows 作業系統上使用非英文語言的問題。 • 版本更新 Greengrass 2.9.0 版本釋放。
2.1.1	版本更新 Greengrass 2.8.0 版本的版本。
2.1.0	新功能 • 更新此元件以自動將遙測指標傳送至 Amazon EventBridge。如需詳細資訊，請參閱 從AWS IoT Greengrass核心裝置收集系統健康狀態遙測資料。 此功能需要 v2.7.0 或更高版 Greeng rass 核成分。 • 版本更新了 Greengrass 核 2.7.0 版本釋放。
2.0.15	版本更新 Greengrass 2.6.0 版本的版本。
2.0.14	此版本包含錯誤修復和改進。
2.0.13	版本更新了 Greengrass 核 2.5.0 版本。
2.0.12	錯誤修復和改進 修正無法將串流管理員 v2.0.7 升級至 v2.0.8 和 v2.0.11 之間的版本的問題。如果您使用串流管理員將資料匯出至雲端，您現在可以升級至 v2.0.12。
2.0.11	版本更新 Greengrass 2.4.0 版本的版本。
2.0.10	版本更新了 Greengrass 核 2.3.0 版本。
2.0.9	版本更新 Greengrass 2.2.0 版本的版本。

版本	變更
2.0.8	版本更新 Greengrass 2.1.0 版本的版本。
2.0.7	初始版本。

Systems Manager 代理

AWS Systems Manager 代理程式元件 (`aws.greengrass.SystemsManagerAgent`) 會安裝系統管理員代理程式，讓您可以使用系統管理員來管理核心裝置。Systems Manager 是可用來檢視和控制基礎設施的 AWS 服務 AWS，包括 Amazon EC2 執行個體、現場部署伺服器和虛擬機器 (VM) 和邊緣裝置。Systems Manager 可讓您檢視作業資料、將作業工作自動化，以及維護安全性與合規性。如需詳細資訊，請參閱什麼是 AWS Systems Manager？ 和AWS Systems Manager 使用者指南中的關於系統管理員代理程式。

Systems Manager 的工具和功能稱為功能。Greengrass 核心設備支持所有 Systems Manager 功能。如需有關這些功能以及如何使用 Systems Manager 來管理核心裝置的詳細資訊，請參閱使用AWS Systems Manager 者指南中的 Systems Manager 功能。

主題

版本

此元件具有下列版本：

- 1.x

- 1.0.x

Type

此元件是一般元件 (`aws.greengrass.generic`)。Greengrass 核會執行元件的生命週期指令碼。

如需詳細資訊，請參閱 元件類型。

作業系統

此元件只能安裝在 Linux 核心裝置上。

要求

此元件具有下列需求：

- 在 64 位元 Linux 平台上執行的格林格拉斯核心裝置：
- 您必須具有 Systems Manager 可以擔任的 AWS Identity and Access Management (IAM) 服務角色。此角色必須包含 AmazonSSM ManagedInstanceCore 受管原則或定義對等權限的自訂原則。如需詳細資訊，請參閱使AWS Systems Manager 用指南中的為邊緣裝置建立 IAM 服務角色。

  部署此元件時，您必須為`SSMRegistrationRole`組態參數指定此角色的名稱。
- Greengrass 裝置角色必須允許和動作`ssm:AddTagsToResource`。`ssm:RegisterManagedInstance`裝置角色還必須允許符合先前需求的 IAM 服務角色`iam:PassRole`執行動作。下列 IAM 政策範例會授予這些權限。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "iam:PassRole"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:iam::account-id:role/SSMServiceRole"
      ]
    },
    {
      "Action": [
```

```
        "ssm:AddTagsToResource",
        "ssm:RegisterManagedInstance"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

端點和連接埠

除了基本作業所需的端點和連接埠之外，此元件還必須能夠對下列端點和連接埠執行輸出要求。如需詳細資訊，請參閱 允許裝置流量透過 Proxy 或防火牆。

端點	連線埠	必要	描述
ec2messages. *region*.amazonaws.com	443	是	與中的系 Systems Manager 服務通訊 AWS 雲端。
ssm.*region*.amazonaws.com	443	是	將核心裝置註冊為系統管理的節點。
ssmmessages. *region*.amazonaws.com	443	是	與中的工作階段管理員 (Systems Manager 的功能) 通訊 AWS 雲端。

有關更多信息，請參閱用戶指南中的參考：ec2消息，ssm消息和其他 API 調用。AWS Systems Manager

相依性

部署元件時， AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的需求，才能成功部署元件。本節列出此元件之已發行版本的相依性，以及定義每個相依性之元件版本的語意版本條件約束。您也可以在AWS IoT Greengrass 主控台中檢視元件每個版本的相依性。在元件詳細資料頁面上，尋找 [相依性] 清單。

下表列出此元件 1.0.0 至 1.2.4 版的相依性。

相依性	兼容版本	相依性類型
代幣交換服務	^2.0.0	軟式

如需有關元件相依性的詳細資訊，請參閱元件方案參考。

組態

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

`SSMRegistrationRole`

Systems Manager 可以承擔的 IAM 服務角色，其中包括 AmazonSSM ManagedInstanceCore 受管政策或定義等效許可的自訂政策。如需詳細資訊，請參閱使AWS Systems Manager 用指南中的為邊緣裝置建立 IAM 服務角色。

`SSMOverrideExistingRegistration`

(選擇性) 如果核心裝置已執行以混合式啟動方式註冊的系統管理員代理程式，您可以覆寫裝置現有的系統管理員代理程式註冊。設定此選項可使用此元件提供的 Systems Manager 代理程式，`true`將核心裝置註冊為受管理節點。

> **Note**
>
> 此選項僅適用於透過混合式啟用註冊的裝置。如果核心裝置在已安裝系統管理員代理程式且已設定執行個體設定檔角色的 Amazon EC2 執行個體上執行，則 Amazon EC2 執行個體現有的受管節點 ID 會以開頭`i-`。當您安裝系統管理員代理程式元件時，系統管理員代理程式會註冊一個新的受管理節點，其 ID 開頭為，`mi-`而不是`i-`。然後，您可以使用 ID 開頭的受管理節點，透過系統管理員`mi-`來管理核心裝置。

預設：`false`

`SSMResourceTags`

(選擇性) 要新增至「系統管理員」管理節點的標籤，此元件會為核心裝置建立。您可以使用這些標記，透過系統管理員來管理核心裝置群組。例如，您可以在具有指定標籤的所有裝置上執行指令。

指定一個清單，其中每個標籤都是具有`Key`和的物件`Value`。例如，下列的值`SSMResourceTags`指示此元件在核心裝置的受管理節點**richard-roe**上將**Owner**標籤設定為。

```
[
  {
    "Key": "Owner",
    "Value": "richard-roe"
  }
]
```

如果受管理節點已存在且`SSMOverrideExistingRegistration`存在，則此元件會忽略這些標籤`false`。

Example 範例：組態合併更新

下列範例組態會指定使用名為的服務角色，`SSMServiceRole`以允許核心裝置註冊並與 Systems Manager 進行通訊。

```
{
  "SSMRegistrationRole": "SSMServiceRole",
  "SSMOverrideExistingRegistration": false,
  "SSMResourceTags": [
    {
      "Key": "Owner",
      "Value": "richard-roe"
    },
    {
      "Key": "Team",
      "Value": "solar"
    }
  ]
}
```

本機記錄檔

系統管理員代理程式軟體會將記錄寫入 Greengrass 根資料夾外的資料夾。如需詳細資訊，請參閱AWS Systems Manager 使用指南中的檢視系統管理員代理程式記錄。

系統管理員代理程式元件會使用 shell 指令碼來安裝、啟動及停止系統管理員代理程式。您可以在下列記錄檔中找到這些指令碼的輸出。

```
/greengrass/v2/logs/aws.greengrass.SystemsManagerAgent.log
```

若要檢視此元件的記錄

- 在核心裝置上執行下列命令，即時檢視此元件的記錄檔。以 AWS IoT Greengrass 根資料夾的路徑取*/greengrass/v2*代。

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.SystemsManagerAgent.log
  ```

另請參閱

- 使用 Greengrass 管理核心裝置 AWS Systems Manager
- 《AWS Systems Manager 使用者指南》中的什麼是 AWS Systems Manager？
- 關於AWS Systems Manager 使用指南中的系統管理員代理程式

變更記錄

下表說明元件每個版本的變更。

版本	變更
1.2.4	錯誤修復和改進 更新此元件以取得代理程式版本 3.2.2303.0。
1.2.3	錯誤修復和改進 • 使用 Greengrass 上的快照添加代理程式元件安裝的重試次數。 • 更新代理程式元件的組態，以便只使用 Greengrass 中的 Onprem 身分識別。

版本	變更
	• 只有在安裝的代理程式版本與 Greengrass SSM 代理程式元件的版本不相符時，才更新此元件才會更新代理程式。
1.1.0	此版本包含錯誤修復和改進。
1.0.0	初始版本。

代幣交換服務

權杖交換服務元件 (aws.greengrass.TokenExchangeService) 提供AWS認證，您可以用來與自訂元件中的AWS服務互動。

令牌交換服務將 Amazon Elastic Container Service (Amazon ECS) 容器執行個體做為本機伺服器執行個體。此本機伺服器會使用您在 Greengrass 核心核心元件中設定的AWS IoT角色別名連線至AWS IoT認證提供者。該組件提供了兩個環境變量，AWS_CONTAINER_CREDENTIALS_FULL_URI和AWS_CONTAINER_AUTHORIZATION_TOKEN。AWS_CONTAINER_CREDENTIALS_FULL_URI定義此本機伺服器的 URI。當元件建立 AWS SDK 用戶端時，用戶端會辨識此 URI 環境變數，並使用中的權杖連線AWS_CONTAINER_AUTHORIZATION_TOKEN至 Token 交換服務並擷取認AWS證。這可讓 Greengrass 核心裝置呼叫服務AWS作業。如需如何在自訂元件中使用此元件的詳細資訊，請參閱與AWS服務互動。

> **Important**
>
> 以這種方式取得AWS認證的 Support 已於 2016 年 7 月 13 日新增至 AWS SDK。您的元件必須使用在該日期或之後建立的 AWS SDK 版本。如需詳細資訊，請參閱 Amazon 彈性容器服務開發人員指南中的使用支援的 AWS SDK。

主題

- 版本
- Type
- 作業系統
- 相依性
- 組態

- 本機記錄檔
- 變更記錄

版本

此元件具有下列版本：

- 2.0.x

Type

此元件是一般元件 (`aws.greengrass.generic`)。Greengrass 核會執行元件的生命週期指令碼。

如需詳細資訊，請參閱 元件類型。

作業系統

此元件可安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

相依性

這個組件沒有任何依賴關係。

組態

此元件沒有任何組態參數。

本機記錄檔

此元件使用與 Greengrass 核元件相同的記錄檔。

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

若要檢視此元件的記錄

- 在核心裝置上執行下列命令，即時檢視此元件的記錄檔。以AWS IoT Greengrass根資料夾的路徑取代*/greengrass/v2*或 *C:\greengrass\v2*。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

變更記錄

下表說明元件每個版本的變更。

版本	變更
2.0.3	初始版本。

IoT 集 SiteWise 電極

IoT SiteWise OPC-UA 收集器元件 (`aws.iot.SiteWiseEdgeCollectorOpcua`) 可讓 AWS IoT SiteWise 閘道器從本機 OPC-UA 伺服器收集資料。

使用此元件， AWS IoT SiteWise 閘道器可以連接到多個 OPC-UA 伺服器。如需有 AWS IoT SiteWise 關設備的詳細資訊，請參閱《使用 AWS IoT SiteWise 指南》中的「在邊緣AWS IoT SiteWise使用」。

主題

- 版本
- Type

版本

此元件具有下列版本：

- 2.4.x 版本
- 2.3.x 版本
- 2.2.x 版本
- 2.1.x
- 2.0.x

Type

此元件是一般元件 (`aws.greengrass.generic`)。Greengrass 核會執行元件的生命週期指令碼。

如需詳細資訊，請參閱 元件類型。

作業系統

此元件可安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件具有下列需求：

- Greengrass 核心裝置必須在下列其中一個平台上執行：
  - 操作系統:Ubuntu 18.04 或更高版本

    體系結構：64（AMD64）或阿爾姆 8（建築 64）
  - 作業系統:紅帽企業版 (RHEL) 8

    體系結構：64（AMD64）或阿爾姆 8（建築 64）
  - 操作系統:Amazon Linux 2

    體系結構：64（AMD64）或阿爾姆 8（建築 64）
  - 操作系統:Debian

    體系結構：64（AMD64）或阿爾姆 8（建築 64）
  - 作業系統:視窗伺服器 2019 或更新版

    系統架構：四十四 (AMD64)
- Greengrass 核心裝置必須允許輸出網路連線至 OPC-UA 伺服器。

相依性

部署元件時， AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的需求，才能成功部署元件。本節列出此元件之已發行版本的相依性，以及定義每個相依性之元件版本的語意版本條件約束。您也可以在AWS IoT Greengrass 主控台中檢視元件每個版本的相依性。在元件詳細資料頁面上，尋找 [相依性] 清單。

下表列出此元件所有版本的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.3.0	硬式
串流管理員	>2.0.10 <3.0.0	硬式
秘密經理	> = 2.0.8	硬式

如需有關元件相依性的詳細資訊，請參閱元件方案參考。

組態

此元件沒有任何組態參數。

您可以使用主 AWS IoT SiteWise 控台或 API 來設定 IoT SiteWise OPC-UA 收集器元件。有關詳情，請參閱《AWS IoT SiteWise 使用指南》中的步驟 4：新增資料來源-可選。

輸入資料

該組件僅接受以下格式的數據，其他所有將被忽略和丟棄。下表將 OPC UA 資料類型對應至 SiteWise 相等資料類型。

SiteWise 資料類型	數據類型	Description
STRING	String Guid XmlElement	最大長度為 1024 個字節的字符串。
INTEGER	SByte Byte Int16 UInt16 Int32 UInt32* Int64*	範圍為的有符號 32 位元整數-2,147,483,648 to 2,147,483,647 。
DOUBLE	UInt32* Int64* Float	具有範圍從-10^100 to 10^100和IEEE 754雙精度的浮點數。

SiteWise 資料類型	數據類型	Description
	Double	
BOOLEAN	Boolean	true 或 false

* 對於 OPC UA 數據類型 UInt32Int64，其 SiteWise 數據類型將 SiteWise 是INTEGER如果能夠表示其值，否則將DOUBLE是。

輸出資料

此組件將BatchPutAssetPropertyValue消息寫入 AWS IoT Greengrass 流管理器。如需詳細資訊，請參閱 AWS IoT SiteWise API 參考中的 BatchPutAssetPropertyValue。

本機記錄檔

此元件會使用下列記錄檔。

Linux

```
/greengrass/v2/logs/aws.iot.SiteWiseEdgeCollectorOpcua.log
```

Windows

```
C:\greengrass\v2\logs\aws.iot.SiteWiseEdgeCollectorOpcua.log
```

若要檢視此元件的記錄

- 在核心裝置上執行下列命令，即時檢視此元件的記錄檔。以 AWS IoT Greengrass 根資料夾的路徑取代*/greengrass/v2*或 *C:\greengrass\v2*。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.iot.SiteWiseEdgeCollectorOpcua.log
  ```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\aws.iot.SiteWiseEdgeCollectorOpcua.log -Tail
 10 -Wait
```

疑難排解和偵錯

此元件包含新的事件記錄檔，可協助客戶識別並修正問題。記錄檔與本機記錄檔不同，可在下列位置找到。以 AWS IoT Greengrass 根資料夾的路徑取代*/greengrass/v2*或 *C:\greengrass\v2*。

Linux

```
/greengrass/v2/work/aws.iot.SiteWiseEdgeCollectorOpcua/logs/
IotSiteWiseOpcUaCollectorEvents.log
```

Windows

```
C:\greengrass\v2\work\aws.iot.SiteWiseEdgeCollectorOpcua\logs
\IotSiteWiseOpcUaCollectorEvents.log
```

此記錄檔包含詳細資訊和疑難排解指示。疑難排解資訊會與診斷程式一起提供，其中包含如何解決問題的說明，有時還會提供進一步資訊的連結。診斷資訊包括下列項目：

- 嚴重性等級
- 時間戳記
- 其他事件特定資訊

Example 範例日誌

```
dataSourceConnectionSuccess:
  Summary: Successfully connected to OpcUa server
  Level: INFO
  Timestamp: '2023-06-15T21:04:16.303Z'
  Description: Successfully connected to the data source.
  AssociatedMetrics:
  - Name: FetchedDataStreams
    Description: The number of fetched data streams for this data source
```

```
    Value: 1.0
    Namespace: IoTSiteWise
    Dimensions:
    - Name: SourceName
      Value: SourceName{value=OPC-UA Server}
    - Name: ThingName
      Value: test-core
  AssociatedData:
  - Name: DataSourceTrace
    Description: Name of the data source
    Data:
    - OPC-UA Server
  - Name: EndpointUri
    Description: The endpoint to which the connection was attempted.
    Data:
    - '"opc.tcp://10.0.0.1:1234"'
```

授權

此元件根據 Greengrass 核心軟體授權合約發行。

變更記錄

下表說明元件每個版本的變更。

版本	變更
2.4.2	錯誤修復和改進 • 修正 OPC UA 伺服器探索期間可能多次探索節點的問題。 • 修正快照功能，以確保每個快照資料點的時間戳記都是新的。
2.4.1	錯誤修復和改進 • 修復了與代理支持相關的問題。 • 修復了線程清理失敗並導致數據阻塞的問題。
2.4.0	新功能 • 新增事件記錄檔，以便更輕鬆地識別和修復問題。

版本	變更
	錯誤修復和改進 • 修正 OPC-UA 用戶端連線至使用 OPC-UA 規格 1.05 版的 OPC-UA 伺服器時造成憑證錯誤的問題。
2.3.0	新功能 • 添加對 Linux 上 Greengrass HTTP 代理配置的支援。 錯誤修復和改進 • 修正了一個安全性問題 (CVE-2019-19135)。
2.2.0	新功能 • 添加對在 Linux ARMv8 體系結構上安裝數據收集包的支持。 • 對於 Linux 的最低要求： • 記憶體：4 GB • 中央處理器：臂皮質-A72 或同等規格 錯誤修復和改進 • 改進了節點發現過程中指標的記錄。 • 改進了對不支持數據類型的處理。 • 改進了數據流錯誤的日誌記錄。
2.1.3	新功能 • 添加對視窗服務器 2019 或更高版本的支持。 錯誤修復和改進 • 改善在不支援的裝置上部署此元件時的錯誤訊息。

版本	變更
2.1.1	新功能 • 添加對配置以下訂閱屬性的支持： • DataChangeTrigger-您可以定義啟動資料變更警示的條件。 • QueueSize-特定測量結果的 OPC-UA 伺服器上佇列深度，其中「受監督項目」的通知會排入佇列。 • PublishingIntervalMilliseconds-建立訂閱時指定的發佈週期間隔 (以毫秒為單位)。 • SnapshotFrequencyMilliseconds -您可以設定快照頻率逾時設定，以確保 AWS IoT SiteWise Edge 擷取穩定的資料流。 • 此版本支援擷取BAD品質資料，並根據下列資料品質篩選資料： • UNCERTAIN 品質資料 • BAD品質資料 錯誤修復和改進 • 客戶指標的改進。 • 修正連線至啟用加密的伺服器時，有時會造成問題的安全性編碼。 • 修正內容群組無法更新的問題。
2.0.3	錯誤修復和改進。
2.0.2	錯誤修復和改進了資產優先級與邊緣同步。
2.0.1	初始版本。

另請參閱

- 什麼是 AWS IoT SiteWise？ 在《AWS IoT SiteWise 使用者指南》中。

IoT SiteWise OPC-UA 數據源模擬器

IoT SiteWise OPC-UA 資料來源模擬器元件 (`aws.iot.SiteWiseEdgeOpcuaDataSourceSimulator`) 會啟動產生樣本資料的本機 OPC-UA 伺服器。使用此 OPC-UA 伺服器模擬閘道器上 IoT SiteWise OPC-UA 收集器元件讀取的資料

來源。AWS IoT SiteWise然後，您可以使用此範例資料探索AWS IoT SiteWise功能。如需有AWS IoT SiteWise關設備的詳細資訊，請參閱《使用AWS IoT SiteWise指南》中的「在邊緣AWS IoT SiteWise使用」。

主題

版本

此元件具有下列版本：

- 1.0.x

Type

此元件是一般元件 (`aws.greengrass.generic`)。Greengrass 核會執行元件的生命週期指令碼。

如需詳細資訊，請參閱 元件類型。

作業系統

此元件可安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件具有下列需求：

- Greengrass 核心裝置必須能夠在本機主機上使用連接埠 4840。此元件的本機 OPC-UA 伺服器會在此連接埠上執行。

相依性

部署元件時，AWS IoT Greengrass也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的需求，才能成功部署元件。本節列出此元件之已發行版本的相依性，以及定義每個相依性之元件版本的語意版本條件約束。您也可以在AWS IoT Greengrass主控台中檢視元件每個版本的相依性。在元件詳細資料頁面上，尋找 [相依性] 清單。

下表列出此元件所有版本的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.3.0	硬式

如需有關元件相依性的詳細資訊，請參閱元件方案參考。

組態

此元件沒有任何組態參數。

本機記錄檔

此元件使用下列記錄檔。

Linux

```
/greengrass/v2/logs/aws.iot.SiteWiseEdgeOpcuaDataSourceSimulator.log
```

Windows

```
C:\greengrass\v2\logs\aws.iot.SiteWiseEdgeOpcuaDataSourceSimulator.log
```

若要檢視此元件的記錄

- 在核心裝置上執行下列命令，即時檢視此元件的記錄檔。以AWS IoT Greengrass根資料夾的路徑取代*/greengrass/v2*或 *C:\greengrass\v2*。

Linux

```
sudo tail -f /greengrass/v2/logs/
aws.iot.SiteWiseEdgeOpcuaDataSourceSimulator.log
```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs
\aws.iot.SiteWiseEdgeOpcuaDataSourceSimulator.log -Tail 10 -Wait
```

變更記錄

下表說明元件每個版本的變更。

版本	變更
1.0.0	初始版本。 添加對視窗服務器 2016 或更高版本的支持。

另請參閱

- 什麼是AWS IoT SiteWise？ 在《AWS IoT SiteWise使用者指南》中。

IoT SiteWise 出版

IoT SiteWise 發行者元件 (`aws.iot.SiteWiseEdgePublisher`) 可讓 AWS IoT SiteWise 閘道將資料從邊緣匯出到 AWS 雲端.

如需有 AWS IoT SiteWise 關設備的詳細資訊，請參閱《使用 AWS IoT SiteWise 指南》中的「在邊緣AWS IoT SiteWise使用」。

主題

- 版本
- Type
- 作業系統

版本

此元件具有下列版本：

- 3.1.x 版本
- 3.0.x
- 2.4.x 版本
- 2.3.x 版本
- 2.2.x 版本
- 2.1.x
- 2.0.x

Type

此元件是一般元件 (`aws.greengrass.generic`)。Greengrass 核會執行元件的生命週期指令碼。

如需詳細資訊，請參閱 元件類型。

作業系統

此元件可安裝在執行下列作業系統的核心裝置上：

- Linux

- Windows

要求

此元件具有下列需求：

- Greengrass 核心裝置必須在下列其中一個平台上執行：
  - 操作系統:Ubuntu 18.04 或更高版本

    體系結構：64（AMD64）或阿爾姆 8（建築 64）
  - 作業系統:紅帽企業版 (RHEL) 8

    體系結構：64（AMD64）或阿爾姆 8（建築 64）
  - 操作系統:Amazon Linux 2

    體系結構：64（AMD64）或阿爾姆 8（建築 64）
  - 操作系統:Debian

    體系結構：64（AMD64）或阿爾姆 8（建築 64）
  - 作業系統:視窗伺服器 2019 或更新版

    系統架構：四十四 (AMD64)
- Greengrass 核心設備必須連接到互聯網。
- Greengrass 核心裝置必須獲得授權才能執行動作。`iotsitewise:BatchPutAssetPropertyValue`如需詳細資訊，請參閱[授權核心裝置與 AWS 服務互動](#)。

  Example 許可政策

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iotsitewise:BatchPutAssetPropertyValue",
            "Resource": "*"
        }
    ]
}
```

端點和連接埠

除了基本作業所需的端點和連接埠之外，此元件還必須能夠對下列端點和連接埠執行輸出要求。如需詳細資訊，請參閱 允許裝置流量透過 Proxy 或防火牆。

端點	連線埠	必要	描述
data.iotsitewise. *region*.amazonaws.com	443	是	將資料發佈至 AWS IoT SiteWise。

相依性

部署元件時， AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的需求，才能成功部署元件。本節列出此元件之已發行版本的相依性，以及定義每個相依性之元件版本的語意版本條件約束。您也可以在AWS IoT Greengrass 主控台中檢視元件每個版本的相依性。在元件詳細資料頁面上，尋找 [相依性] 清單。

下表列出此元件 2.0.x 至 2.x 版的相依性。

相依性	兼容版本	相依性類型
Greengrass 核	> = 2.3.0<3.0.0	硬式
串流管理員	>=2.0.10	硬式

如需有關元件相依性的詳細資訊，請參閱元件方案參考。

組態

此元件沒有任何組態參數。

您可以使用主 AWS IoT SiteWise 控台或 API 來設定 IoT SiteWise 發行者元件。如需詳細資訊，請參閱《AWS IoT SiteWise 使用指南》中的步驟 3：設定發行者 -選用。

輸入資料

此組件從 AWS IoT Greengrass 流管理器讀取`PutAssetPropertyValueEntry`消息。如需詳細資訊，請參閱 AWS IoT SiteWise API 參考[PutAssetPropertyValueEntry](#)中的。

本機記錄檔

此元件會使用下列記錄檔。

Linux

```
/greengrass/v2/logs/aws.iot.SiteWiseEdgePublisher.log
```

Windows

```
C:\greengrass\v2\logs\aws.iot.SiteWiseEdgePublisher.log
```

若要檢視此元件的記錄

- 在核心裝置上執行下列命令，即時檢視此元件的記錄檔。以 AWS IoT Greengrass 根資料夾的路徑取代*/greengrass/v2*或 *C:\greengrass\v2*。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.iot.SiteWiseEdgePublisher.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.iot.SiteWiseEdgePublisher.log -Tail 10 -
  Wait
  ```

疑難排解和偵錯

此元件包含新的事件記錄檔，可協助客戶識別並修正問題。記錄檔與本機記錄檔不同，可在下列位置找到。以 AWS IoT Greengrass 根資料夾的路徑取代*/greengrass/v2*或 *C:\greengrass\v2*。

Linux

```
/greengrass/v2/work/aws.iot.SiteWiseEdgePublisher/logs/
IotSiteWisePublisherEvents.log
```

Windows

```
C:\greengrass\v2\work\aws.iot.SiteWiseEdgePublisher\logs
\IotSiteWisePublisherEvents.log
```

此記錄檔包含詳細資訊和疑難排解指示。疑難排解資訊會與診斷程式一起提供，其中包含如何解決問題的說明，有時還會提供進一步資訊的連結。診斷資訊包括下列項目：

- 嚴重性等級
- 時間戳記
- 其他事件特定資訊

Example 範例日誌

```
accountBeingThrottled:
  Summary: Data upload speed slowed due to quota limits
  Level: WARN
  Timestamp: '2023-06-09T21:30:24.654Z'
  Description: The IoT SiteWise Publisher is limited to the "Rate of data points
 ingested"
    quota for a customers account. See the associated documentation and associated
    metric for the number of requests that were limited for more information. Note
    that this may be temporary and not require any change, although if the issue
 continues
    you may need to request an increase for the mentioned quota.
  FurtherInformation:
  - https://docs.aws.amazon.com/iot-sitewise/latest/userguide/quotas.html
  - https://docs.aws.amazon.com/iot-sitewise/latest/userguide/troubleshooting-
gateway.html#gateway-issue-data-streams
  AssociatedMetrics:
  - Name: TotalErrorCount
    Description: The total number of errors of this type that occurred.
    Value: 327724.0
  AssociatedData:
  - Name: AggregatePropertyAliases
    Description: The aggregated property aliases of the throttled data.
    FileLocation: /greengrass/v2/work/aws.iot.SiteWiseEdgePublisher/./logs/data/
AggregatePropertyAliases_1686346224654.log
```

授權

此元件是根據 Greengrass 核心軟體授權合約發行的。

變更記錄

下表說明元件每個版本中的變更。

版本	變更
3.1.2	錯誤修復和改進 • 修復了 3.1.1 版中引入的 CPU 使用率過高的問題。
3.1.1	錯誤修復和改進 • 新增其他記錄，以在發生錯誤時識別受影響的資料別名。 • 針對擷取的資料年齡新增 AWS IoT SiteWise API 限制的本機強制執行功能。 • 修正當有多個 Amazon S3 目的地時，發佈者混合 StreamManager 串流的檢查點的問題。 • 修正發行者從 StreamManager 串流讀取方式的效能瓶頸。
3.1.0	新功能 • 添加對將數據作為實木複合地板文件發布到 Amazon S3 的支持。 • 添加對 AWS IoT SiteWise 緩衝擷取的支援。
3.0.0	錯誤修復和改進 • 修復了與代理支持相關的問題。 新功能 • 支援從 MQTT 代理程式擷取資料。
2.4.1	錯誤修復和改進 • 啟用元件與 Java 編輯器 11 版本 11.0.20.8.1 及更高版本一起使用。組件版本 2.4.0 和 2.3.3 顯示`"Could not find or load main class"`錯誤消息時，與 Java Corretto 版本 11.0.20.8.1 使用。
2.4.0	新功能 • 添加新的事件日誌，以便更容易識別和修復問題。

版本	變更
	錯誤修復和改進 • 改善發行者檢查點復原。
2.3.3	錯誤修復和改進 • 改善支援高輸送量的能力。
2.3.2	錯誤修復和改進 • 修復了下載發布者配置時的 HTTP 代理支持。
2.3.1	新功能 • 添加對在 Linux ARMv8 體系結構上安裝數據收集包的支持。 • 對於 Linux 的最低要求： • 記憶體：4 GB • 中央處理器：臂皮質-A72 或同等規格
2.2.3	錯誤修復和改進 • 移除不在可重新擷取例外清單中的一般例外重試。
2.2.2	錯誤修復和改進 • AWS IoT SiteWise 透過 HTTP 代理伺服器重新引入資料上傳支援。
2.2.1	Note 此版本不支援 HTTP 代理伺服器組態。2.2.2 及更高版本重新引入了對此功能的支持。 新功能 • 添加對此組件的支持，以在將數據上傳到時切換壓縮 AWS IoT SiteWise。

版本	變更

2.2.0	**Note**
	此版本不支援 HTTP 代理伺服器組態。2.2.2 及更高版本重新引入了對此功能的支持。
	
	新功能
	• 更新此組件以在將數據發送到 AWS IoT SiteWise 服務之前壓縮數據。
	  • 在大多數情況下，與此元件的舊版相比，這項變更可減少 75% 的頻寬使用量。
	  • 在大多數情況下，這項變更會將 CPU 使用率增加最多 5%。在處理大量資料的閘道上，此變更最多可將 CPU 使用率增加 15%。
	  • 這項變更不會影響 AWS IoT SiteWise 服務費用或服務配額使用量。
	  • 添加對視窗服務器 2019 或更高版本的支持。
	錯誤修復和改進
	• 修正檢查點檔案損毀時造成此元件無法啟動的問題。

2.1.4	錯誤修復和改進 • 修復了與 Java 版本 8 的兼容性。
2.1.3	**Warning** 除了美國東部 (俄亥俄)、加拿大 (中部) 和 AWS GovCloud (美國東部) 區域外，已不再提供此版本。此元件版本需要執行 Java 11 或更新版本。此版本中的改進功能在此元件的更新版本中提供。 錯誤修復和改進 • 改善在不支援的裝置上部署此元件時的錯誤訊息。 • 資料上傳失敗時記錄錯誤的更新。
2.1.2	錯誤修復和改進 • 更新以在資料到期時叫用過期的資料匯出功能。

版本	變更
2.1.1	錯誤修復和改進。
2.1.0	新功能 • 添加對首先將最新數據發布到雲的支持。 • 添加對不將過期數據發布到雲的支持。 • 添加對本地存儲過期數據的支持。 錯誤修復和改進 • 減少磁碟 I/O 和對應的延遲。
2.0.2	錯誤修復和改進。
2.0.1	初始版本。

另請參閱

- 什麼是 AWS IoT SiteWise？ 在《AWS IoT SiteWise 使用者指南》中。

IoT SiteWise 處理

IoT 處 SiteWise 理器元件 (`aws.iot.SiteWiseEdgeProcessor`) 可讓 AWS IoT SiteWise 閘道在邊緣處理資料。

有了這個元件，AWS IoT SiteWise 閘道可以使用資產模型和資產來處理閘道裝置上的資料。如需有 AWS IoT SiteWise 關設備的詳細資訊，請參閱《使用 AWS IoT SiteWise 指南》中的「在邊緣AWS IoT SiteWise使用」。

主題

- 版本
- Type
- 作業系統
- 要求
- 相依性
- 組態

- 本機記錄檔
- 授權
- 變更記錄
- 另請參閱

版本

此元件具有下列版本：

- 3.2.x
- 3.1.x 版本
- 3.0.x
- 2.2.x 版本
- 2.1.x
- 2.0.x

Type

此元件是一般元件 (`aws.greengrass.generic`)。Greengrass 核會執行元件的生命週期指令碼。

如需詳細資訊，請參閱 元件類型。

作業系統

此元件可安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件具有下列需求：

- Greengrass 核心裝置必須在下列其中一個平台上執行：
  - 操作系統:

系統架構：四十四 (AMD64)

- 作業系統:紅帽企業版 (RHEL) 8

  系統架構：四十四 (AMD64)

- 操作系統:Amazon Linux 2

  系統架構：四十四 (AMD64)

- 作業系統:視窗伺服器 2019 或更新版

  系統架構：四十四 (AMD64)

- Greengrass 核心裝置必須允許連接埠 443 上的輸入流量。
- Greengrass 核心裝置必須允許連接埠 443 和 8883 上的輸出流量。
- 下列連接埠保留供使用 AWS IoT SiteWise：
  80、443、3001、4569、4572、8000、8082、8084、8085、8086、8445、9000、9500、11080 和 50010。為流量使用保留的連接埠可能會導致連線終止。

  > **Note**
  >
  > 只有此元件的版本 2.0.15 及更新版本才需要連接埠 8087。

- Greengrass 裝置角色必須具有允許您在裝置上使用 AWS IoT SiteWise 閘道的權限。 AWS IoT Greengrass V2 若要取得更多資訊，請參閱AWS IoT SiteWise 使用指南中的「需求」。

端點和連接埠

除了基本作業所需的端點和連接埠之外，此元件還必須能夠對下列端點和連接埠執行輸出要求。如需詳細資訊，請參閱 允許裝置流量透過 Proxy 或防火牆。

端點	連線埠	必要	描述
`model.iotsitewise.region.amazonaws.com`	443	是	取得資產和資 AWS IoT SiteWise 產模型的相關資訊。

端點	連線埠	必要	描述
edge.iotsitewise. *region*.amazonaws.com	443	是	取得核心裝置 AWS IoT SiteWise 閘道組態的相關資訊。
ecr.*region*.amazonaws.com	443	是	從 Amazon 彈性容器註冊表下載 AWS IoT SiteWise 邊緣網關碼頭映像。
iot.*region*.amazonaws.com	443	是	取得適用於您的 AWS 帳戶.
sts.*region*.amazonaws.com	443	是	取得您的 AWS 帳戶.
monitor.iotsitewise. *region*.amazonaws.com	443	否	如果您存取核心裝置上的 AWS IoT SiteWise Monitor 入口網站，則為必要項

相依性

部署元件時， AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的需求，才能成功部署元件。本節列出此元件之已發行版本的相依性，以及定義每個相依性之元

件版本的語意版本條件約束。您也可以在AWS IoT Greengrass 主控台中檢視元件每個版本的相依性。在元件詳細資料頁面上，尋找 [相依性] 清單。

下表列出此元件 2.0.x 至 2.1.x 版的相依性。

相依性	兼容版本	相依性類型
代幣交換服務	> = 2.0.3	硬式
串流管理員	>=2.0.10 <3.0.0	硬式
Greengrass	> = 2.3.0	硬式

如需有關元件相依性的詳細資訊，請參閱元件方案參考。

組態

此元件沒有任何組態參數。

本機記錄檔

此元件會使用下列記錄檔。

Linux

```
/greengrass/v2/logs/aws.iot.SiteWiseEdgeProcessor.log
```

Windows

```
C:\greengrass\v2\logs\aws.iot.SiteWiseEdgeProcessor.log
```

若要檢視此元件的記錄

- 在核心裝置上執行下列命令，即時檢視此元件的記錄檔。以 AWS IoT Greengrass 根資料夾的路徑取代*/greengrass/v2*或 *C:\greengrass\v2*。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.iot.SiteWiseEdgeProcessor.log
  ```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\aws.iot.SiteWiseEdgeProcessor.log -Tail 10 -
Wait
```

授權

此元件包括下列協力廠商軟體/授權：

- 阿帕奇 -2.0
- 理工學
- BSD-條款
- BSD-条款
- 卡德的 -1.0
- 卡德的 -1.1
- ISC
- 茲利卜
- 具有 GCC 異常的 GPL-3.0
- 公共領域
- 蟒蛇 2.0
- 德国联合国足球协会
- BSD-1-條款
- OpenSSL
- EPL-1.0
- EPL-2.0
- GPL-2.0-with-classpath-exception
- 邁普爾 -2.0
- C0-1.0
- JSON

此元件是根據 Greengrass 核心軟體授權合約發行的。

變更記錄

下表說明元件每個版本中的變更。

版本	變更
3.2.0	效能提升 • 優化 API 服務以減少內存佔用空間，並且需要更少的磁盤空間來安裝 • 這樣可減少 2 GB 的初始記憶體使用量 (現在啟動時使用 7.5 GB 的記憶體，但仍建議使用 16 GB)，整個元件的下載大小減少 500 MB (現在需要下載 1.4 GB)。 新功能 • `GetAssetPropertyValueAggregates` API 現在支援邊緣上 15 分鐘的聚合視窗。 • 連接埠 8081 和 8082 不再需要可用，此元件才能正確執行。 **Note** AWS IoT SiteWise 資料平面 API 的本機端點 (例如`get-asset-property-value`) 正在從變更`http://localhost:8081` 為`http://localhost:11080/data`。AWS IoT SiteWise 控制平面 API 的本機端點 (例如`list-asset-models`) 已從變更`http://localhost:11080` 為`http://localhost:11080/control`。AWS 一律建議您使用邊 SiteWise 緣閘道 HTTPS 端點。這些端點尚未變更。 錯誤修復和改進 • 現在，如果先前的同步中斷，從同步處理 AWS IoT SiteWise 會將資源轉換為有效狀態。這將修復強制重新啟動後某些資源損壞的問題。 • 修復了在同步期間修改資源時邊緣上可能損壞的罕見情況。現在，如果偵測到此情況，同步將會失敗，並在下次同步中重試資源。 • 修正可能允許 API 的 HTTP 端點在外部呼叫的問題。現在只能使用 HTTPS 來呼叫本機回送位址以外的 API。 • `ListAssets` API 現在會針對儲存在邊緣的資產顯示資產階層。 • 修正資料處理套件無法在 Windows 上重新啟動、升級或降級的問題。

版本	變更
	• 修正 Windows 作業系統資料處理套件中的錯誤，此錯誤會導致客戶無法使用認證與 MQTT 代理人連線。
3.1.3	錯誤修復和改進 • 修正當某些資源實際失敗時，資料處理套件不正確回報成功同步處理的問題。 • 允許多個資產具有相同的名稱，只要它們沒有相同的父項。
3.1.1	錯誤修復和改進 • 修復了由於時區不匹配而導致 Sigv4 請求失敗的問題。 • 修正轉換和量度屬性在重新啟動後依賴屬性時停止計算的問題。 • 啟用自訂串流管理員連接埠設定的支援。 • 修正同步至邊緣的屬性可能會停止更新的問題。
3.1.0	錯誤修復和改進 • 修復 `ListAssetModels` API 無法生成下一個令牌的問題。
3.0.0	新功能 • 支援從 MQTT 代理程式擷取資料。
2.2.1	錯誤修復和改進 • 調整同步處理程序，使控制平面資料儲存與雲端運作方式更加一致。這會稍微影響升級。 **Note** 在 2.2.1 版或更高版本上同步的控制平面資料將與舊版不相容。要降級到以前的版本，您需要完成全新安裝。這不會影響升級，在舊版上同步的資料將適用於 2.2.1 版。 • 對 AWS 憑證鏈結進行其他修改以排定 AWS IoT Greengrass V2 憑證的優先順序

版本	變更
2.1.37	錯誤修復和改進 • 棄用 dependency-routing-service 過程並將其功能移動到該 property-state-service 過程中，以減少通信過程中的資源使用量。 • 將 `get-asset-property-value-history` API 的最大結果限制增加到 20,000，以符合所使用的限制 AWS IoT SiteWise。 • 修正未指定最大結果限制時，`get-asset-property-value-history` API 的分頁結果中未提供下一個權杖的問題。
2.1.35	錯誤修復和改進 • 修改 AWS 認證鏈結以排定認 AWS IoT Greengrass 證的優先順序。 • 修正部署為 AWS IoT 物件群組的一部分時，帳戶偵測的問題。
2.1.34	錯誤修復和改進 • 調整度量/轉換計算以在 Linux 上使用多執行緒。Windows 會繼續執行單執行緒運算，以確保相容性。 • 修正某些計算視窗會遺失度量計算的問題。
2.1.33	錯誤修復和改進 • 修正了向 Greengrass 控制台報告錯誤狀態的問題。
2.1.32	錯誤修復和改進 • 添加對自定義用戶名和組的支持。
2.1.31	錯誤修復和改進 • 添加支援，以計算在中建模的資料的時間加權平均值和時間加權標準差。AWS IoT SiteWise
2.1.29	錯誤修復和改進 • 添加對邊緣功能過濾資產的支持。
2.1.28	錯誤修復和改進 • 最佳化資源同步處理，讓大量資產能夠從邊 AWS 雲端 緣同步。

版本	變更
2.1.24	錯誤修復和改進 • 修正第二次同步資源時導致儀表板消失的問題。
2.1.23	錯誤修復和改進 • 為aws.iot.SiteWiseEdgeProcessor 安裝過程添加了超時，以避免 Internet 連接速度慢時安裝失敗。 • 優化資源同步以提高雲端和邊緣之間的同步效率。
2.1.21	⚠ Warning 從 2.0.x 升級到 2.1.x 將導致本地數據丟失。 新功能 • 添加對視窗服務器 2019 或更高版本的支持。 • 移除基於 Linux 的操作系統的 docker。
2.0.16	此版本包含錯誤修復和改進。
2.0.15	錯誤修復和改進 • 將此元件用於資源同步處理 API 作業的連接埠從 8085 變更為 8087。因此，此元件現在需要連接埠 8087 才能使用。此元件仍需要連接埠 8085 才能使用。 • 更新 AWS OpsHub 驗證以在登錄期間拒絕未經授權的用戶，而不是當用戶嘗試調用 API 操作時。
2.0.14	此版本包含錯誤修復和改進。
2.0.13	錯誤修復和改進 • 修正問題，以便當此元件向 Amazon 指 CloudWatch標報告資料時，現在可正確指出哪些資料未建模。

版本	變更
2.0.9	錯誤修復和改進 • 改善在核心裝置上建立和更新 AWS IoT SiteWise 資源的可靠性。 • 新增其他本機 API 作業，您可用來監視核心裝置上安裝的元件、每個元件的版本，以及每個元件的狀態。您可以在核心裝置上的 AWS IoT SiteWise 應用程式的 [設定] 索引標籤上檢視此資訊。 AWS OpsHub • 為此元件執行的 Docker 容器新增健全狀況狀態。您可以執行命`docker ps`令來檢視容器的健全狀況狀態。
2.0.7	錯誤修復和改進 • 修復了對在核心設備上查看 AWS IoT SiteWise Monitor 門戶的支持。
2.0.6	錯誤修復和改進 • 修正 AWS IoT SiteWise `statetime()` 此元件在核心裝置上計算的`earliest()` 、和`latest()`功能。
2.0.5	錯誤修復和改進 • 添加對此組件在核心設備上計算的轉換 AWS IoT SiteWise `pretrigger()` 函數的支持。 • 變更此元件儲存輕量型目錄存取通訊協定 (LDAP) 組態以進行驗證的路徑。
2.0.2	初始版本。

另請參閱

- 什麼是 AWS IoT SiteWise？ 在《AWS IoT SiteWise 使用者指南》中。

出版商支援的元件

發行者支援的元件位於的預覽版本中，AWS IoT Greengrass且可能會變更。這些元件不受支援AWS。如果每個元件有任何問題，您必須聯絡「發行者」。

Greengrass 發行者支援的元件是由第三方元件廠商開發、提供和服務。第三方元件廠商來自AWS Partner裝置目錄、AWS英雄或社群廠商。您可以直接聯絡第三方元件廠商，以購買此目錄中的元件。

Greengrass 發行者支援的元件包括下列項目：

主題

- 輔助屏蔽邊緣
- AI EdgeLabs 感測器
- Greengrass S3 技巧

輔助屏蔽邊緣

該組件由博世提供支持的 AIShield 開發和支持。利用人工智慧邊緣提升您的人工智慧安全性。此元件旨在將威脅知情、量身打造的防禦系統無縫部署到邊緣裝置，保護您的裝置免受 AI 攻擊。

此元件提供下列優點：

- 利用 AIShield AI 防護從弱點分析無縫過渡到內部強化的邊緣防禦 AWS
- 輕鬆在多個邊緣裝置上部署量身打造的防禦
- 針對各種 AI 設定量身打造的廣泛保護，支援各種模型類型和架構
- 通過無縫集成到 Greengrass 工作流程中Amazon SageMaker保持最新狀態
- 立即深入瞭解潛在威脅，並將資料直接傳送至 AWS IoT Core
- 透過 Marketplace 上的 AIShield AI 安全防護在邊緣進行防禦部署的有凝聚力的人工智慧安全性途徑 AWS

此元件必須在下列平台上執行：

- 作業系統:Linux

如果您有興趣購買此元件，請聯絡博世軟體和數位解決方案：<AIShield.Contact@bosch.com>。

AI EdgeLabs 感測器

該組件是由 AI 開發並支持的 EdgeLabs。AI EdgeLabs Sensor 是一種基於容器的應用程序，其中包含基於 AI 的威脅檢測和預防功能。人工智能傳感器包裝到 Greengrass 組件中，並與其他 Greengrass 組件一起部署為核心設備上的獨立容器。

此目前元件是以容器為基礎的代理程式，可持續驗證網路通訊，並在 Edge Host 或 IoT 閘道上執行的軟體中尋找威脅模式。此元件使用 EBPF、程序頻寬的行為驗證，以及主機型組態。此元件的主要功能是以 NDR/IPS 和 EDR 功能為基礎。

此元件提供下列優點：

- 基於 AI 的威脅檢測，防止網絡攻擊和惡意軟件（EDR/NDR）
- 自動化 AI 型事件回應 (IPS)
- 主機本機威脅情報，以最少的資料傳輸到外部
- 使用碼頭和 Greengrass 進行輕量級部署

此元件必須在下列其中一個平台上執行：

- 作業系統:Linux

如果您有興趣購買此元件，請聯絡 AI EdgeLabs：<contact@edgelabs.ai>。

Greengrass S3 技巧

這個組件是開發的，由內森·格洛弗支持。Greengrass S3 Ingestor 元件是設計用來搭配串流管理員元件使用。該組件從流管理器中獲取 JSON 消息的行分隔流，並將其批處理到 GZIP 文件中。此元件可有效率地將資料擷取到 Amazon S3，以進行進一步處理或儲存。此組件不支持實時將數據發送到。AWS 雲端

此元件必須在下列其中一個平台上執行：

- 作業系統:Linux
- 操作系統:視窗

如果您有興趣購買此組件，請聯繫內森·格洛弗：<nathan@glovers.id.au>。

社群元件

Greengrass 軟件目錄是由 Greengrass 社區開發的 Greengrass 組件的索引。您可以從此目錄下載、修改和部署元件，以建立 Greengrass 應用程式。您可以通過以下鏈接查看目錄：https://github.com/aws-greengrass/ aws-greengrass-software-catalog.

每個組件都有一個公共 GitHub 存儲庫，您可以瀏覽。檢視上的 Greengrass 軟體目錄 GitHub 以尋找完整的社群元件清單。例如，此目錄包括下列元件：

- Amazon Kinesis Video Streams

  此元件會從使用即時串流通訊協定 (RTSP) 的本機攝影機擷取音訊和視訊串流。然後，元件會將音訊和視訊串流上傳到 Amazon Kinesis Video Streams。
- 藍牙 IoT 閘道

  此元件使用可與藍牙低功耗 (LE) 裝置進行通訊的程式BluePy庫，以建立藍牙 LE 用戶端介面。
- 憑證旋轉器

  此元件提供跨叢集大規模旋轉 AWS IoT Greengrass 核心裝置憑證和私密金鑰的方法。
- 容器化安全隧道

  此元件提供 Docker 容器，用於安全通道，其中包含不依賴特定主機作業系統的可重複使用配方中的所有相依性和相符程式庫。
- Grafana

  此元件可讓您在 Greengrass 核心裝置上託管 Grafana 伺服器。您可以使用 Grafana 儀表板來視覺化並管理核心裝置上的資料。
- Amazon Lookout for Vision 的 GStreamer

  此元件提供 GStreamer 外掛程式，因此您可以在自訂 GStreamer 管道中執行檢測 Lookout for Vision 異常偵測。
- 家庭助理

  此元件可讓客戶使用「家庭助理」提供智慧家庭裝置的本機控制。它提供與邊緣和雲端 AWS 服務的整合，以提供可擴展「家庭助理」的家庭自動化解決方案。
- 導入儀表板

  這個元件提供了一鍵式體驗來設定 InfluxDB 和 Grafana 元件。它將 InfluxDB 連接到 Grafana 和自動化的本地 Grafana 儀表板的設置，呈現遙測在實時。 AWS IoT Greengrass
- 資源分貝

  此元件在 Greengrass 核心裝置上提供 InfluxDB 時間序列資料庫。您可以使用此元件來處理來自 IoT 感應器的資料、即時分析資料，以及監控邊緣的作業。
- 發行者

此元件會將 AWS IoT Greengrass 系統健康狀態遙測從 Nucleus 發射器外掛程式轉送至 InfluxDB。此元件也可以將自訂遙測轉寄至 InfluxDB。

- IoT 發布框架

  此架構提供應用程式架構、範本程式碼和可部署的範例，協助改善使用 v2 自訂元件的分散式事件導向 IoT pubsub 應用 AWS IoT Greengrass 程式的程式碼品質。如需詳細資訊，請參閱 建立 AWS IoT Greengrass 元件。

- 木普特實驗室

  此元件會部署 JupyterLab 至 AWS IoT Greengrass 核心裝置。Jupyter 環境可以訪問由設置的過程和環境變量資源 AWS IoT Greengrass，簡化測試和開發用 Python 編寫的組件的過程。

- 本機網頁伺服器

  此元件可讓您在 Greengrass 核心裝置上建立本機 Web 使用者介面。您可以建立本機 Web 使用者介面，以便設定裝置和應用程式設定或監視裝置，例如。

- LoRaWaN 協議適配器

  此元件會從使用 LoRaWa N 通訊協定的本機無線裝置擷取資料，此通訊協定是低功率廣域網路 (LPWAN) 通訊協定。此元件可讓您在本機分析資料並對其採取行動，而無需與雲端通訊。

- https://github.com/awslabs/aws-greengrass-labs-modbus-tcp-protocol-adapter

  此元件會使用 ModBbustCP 通訊協定從本機裝置收集資料，並將其發佈至選取的資料串流。

- 紅色節點

  該組件使用 NPM 在 AWS IoT Greengrass 核心設備上安裝節點紅色。該組件取決於必須明確部署和配置的節點紅色 Auth 組件。您可以使用 Greengrass 的節點紅色 CLI 將節點紅色流程部署到裝置。 AWS IoT Greengrass

- 紅色節點碼頭

  該組件使用官方的節點紅色 Docker 容器在 AWS IoT Greengrass 核心設備上安裝節點紅色。該組件取決於必須明確部署和配置的節點紅色 Auth 組件。您可以使用 Greengrass 的節點紅色 CLI 將節點紅色流程部署到裝置。 AWS IoT Greengrass

- 節點紅色身份驗證

  此元件會設定使用者名稱和密碼，以保護核心裝置上執行的 Node-red 執行個體。 AWS IoT Greengrass

- OpenThread邊界路由器

此元件會部署 OpenThread 邊界路由器 Docker 容器。該組件有助於組成包括線程邊界路由器的問題設備。

- OSI Pi 串流資料連接器

  此元件提供從 OSI Pi 資料封存到現代資料架構的串流即時資料擷取。 AWS它集成了 OSI Pi 資產框架，該框架 AWS IoT PubSub 通過消息傳遞集中管理。

- 剖析提供者

  該組件使 AWS IoT Greengrass 設備能夠使用雲原生計算基金會（CNCF）的開源 Parsec 項目集成硬件安全解決方案。

- PostgreSQL 数据库

  此元件在邊緣提供 PostgreSQL 關聯式資料庫的支援。客戶可以使用此元件在泊塢視窗容器內佈建和管理本機 PostgreSQL 執行個體。

- S3 文件上傳器

  此元件會監控目錄中是否有新檔案，將它們上傳到 Amazon Simple Storage Service (Amazon S3)，然後在成功上傳後刪除這些檔案。

- Secrets Manager 客戶端

  這個元件提供了一個 CLI 工具，可供需要從配方生命週期指令碼中從 Secrets Manager 元件擷取密碼的其他元件使用。

- TES 路由至貨櫃

  此元件會在 AWS IoT Greengrass 裝置上設定 nftables 或 iptables，以便它可以將元件與容器搭配使用。代幣交換服務

- WebRTC

  此元件會從連接至 AWS IoT Greengrass 核心裝置的 RTSP 攝影機擷取音訊和視訊串流。然後元件會透過 Amazon Kinesis 視訊串流將音訊和視 peer-to-peer訊串流轉換為通訊或中繼。

若要要求某項功能或回報錯誤，請在該元件的儲存庫上開啟 GitHub 問題。 AWS 不提供社區組件的支持。如需詳細資訊，請參閱每個元件儲存庫中的CONTRIBUTING.md檔案。

幾個 AWS提供的元件也是開放原始碼。如需更多詳細資訊，請參閱 開放原始碼AWS IoT Greengrass 核心軟體。

AWS IoT Greengrass開發工具

使用AWS IoT Greengrass開發工具來建立、測試、建置、發佈和部署自訂 Greengrass 元件。

- Greengrass 開發工具包 CLI

  使用本機AWS IoT Greengrass開發環境中的開發套件命令列介面 (GDK CLI)，從 Greengr ass 軟體目錄中的範本和社群元件建立元件。您可以使用 GDK CLI 建置元件，並將元件發佈至AWS IoT Greengrass服務，做為. AWS 帳戶
- 綠色命令行界面

  使用 Greengrass 核心設備上的命令行界面（Greengrass CLI）來部署和調試 Greengrass 組件。Greengrass CLI 是您可以部署到核心裝置的元件，以建立本機部署、檢視有關已安裝元件的詳細資料，以及瀏覽記錄檔。
- 本機除錯主控台

  使用 Greengrass 核心裝置上的本機偵錯主控台，使用本機儀表板 Web 介面來部署和偵錯 Greengrass 元件。本機偵錯主控台是您可以部署至核心裝置的元件，以建立本機部署並檢視有關已安裝元件的詳細資料。

AWS IoT Greengrass還提供了以下可用於自定義 Greengrass 組件的 SDK：

- 的AWS IoT Device SDK，其中包含進程間通信（IPC）庫。如需詳細資訊，請參閱 使AWS IoT Device SDK用與 Greengrass 核、其他元件和通訊 AWS IoT Core。
- 串流管理員 SDK，您可以使用此 SDK 將資料串流傳輸到AWS 雲端. 如需更多詳細資訊，請參閱 管理核心裝 Greengrass 資料串流。

主題

- AWS IoT Greengrass開發套件命令列介面
- 綠色命令行界面
- 使用 AWS IoT Greengrass 測試框架

AWS IoT Greengrass開發套件命令列介面

開AWS IoT Greengrass發套件命令列介面 (GDK CLI) 提供的功能可協助您開發自訂 Greeng rass 元件。您可以使用 GDK CLI 來建立、建置和發佈自訂元件。使用 GDK CLI 建立元件存放庫時，您可以

從 Greengr ass 軟體目錄中的範本或社群元件開始。然後，您可以選擇將文件打包為 ZIP 存檔的構建系統，使用 Maven 或 Gradle 構建腳本，或運行自定義構建命令。建立元件之後，您可以使用 GDK CLI 將其發佈到AWS IoT Greengrass服務，因此您可以使用AWS IoT Greengrass主控台或 API 將元件部署到 Greengrass 核心裝置。

當您在沒有 GDK CLI 的情況下開發 Greengrass 元件時，每次建立元件的新版本時，都必須更新元件方案檔案中的版本和成品 URI。當您使用 GDK CLI 時，它可以在您每次發佈新版本的元件時為您自動更新版本和成品 URI。

GDK CLI 是開放原始碼的，可在上 GitHub使用。您可以自訂和擴充 GDK CLI，以滿足您的元件開發需求。我們邀請您在 GitHub 存儲庫上打開問題並提取請求。您可以通過以下鏈接找到 GDK CLI 源代碼：https://github.com/aws-greengrass/ aws-greengrass-gdk-cli.

必要條件

要安裝和使用 Greengrass 開發工具包 CLI，您需要以下內容：

- AWS 帳戶。如果您沒有帳戶，請參閱 設置一個 AWS 帳戶。
- 具有網際網路連線的視窗、macOS 或類似 UNIX 的開發電腦。
- 對於 GDK CLI 1.1.0 或更新版本，Python 3.6 或更新版本已安裝在您的開發計算機上。

  對於開發計算機上安裝的 GDK CLI 版本 1.0.0，Python 3.8 或更高版本。
- Git 安裝在您的開發計算機上。
- AWS Command Line Interface（AWS CLI）在您的開發計算機上安裝和配置憑據。如需詳細資訊，請參閱《AWS Command Line Interface使用指南》AWS CLI中的〈安裝、更新AWS CLI和解除安裝〉和〈設定〉。

  > **Note**
  >
  > 如果您使用樹莓派或其他 32 位 ARM 設備，請安裝 AWS CLI V1。 AWS CLIV2 不適用於 32 位元 ARM 裝置。如需詳細資訊，請參閱安裝、更新和解除安裝AWS CLI版本 1。
- 若要使用 GDK CLI 將元件發行至AWS IoT Greengrass服務，您必須具備下列權限：
  - `s3:CreateBucket`
  - `s3:GetBucketLocation`
  - `s3:PutObject`
  - `greengrass:CreateComponentVersion`

  - `greengrass:ListComponentVersions`
- 若要使用 GDK CLI 建置其成品存在於 S3 儲存貯體中而非本機檔案系統中的元件，您必須具備下列權限：
  - `s3:ListBucket`

  此功能適用於 GDK CLI 1.1.0 版及更新版本。

變更記錄

下表說明 GDK CLI 每個版本中的變更。如需詳細資訊，請參閱上的 GDK CLI 發行版本頁面 GitHub。

版本	變更
1.6.2	錯誤修復和改進 • 修復了 Windows gradlew.bat 由於相對路徑而無法正常工作的問題。 • 對日誌記錄，測試和打包進行小幅改進。
1.6.1	錯誤修復和改進 • 新增 CLI 引數剖析的安全性修正程式。 • 使 GDK 獲取最新的 Greengrass 測試框架（GTF）發行名稱作為默認的 GTF 版本。 • 讓 GDK 建議使用舊版 GTF 的客戶，將其更新為最新版本。
1.6.0	新功能 • 在和指令期間，針對 Greengrass 方案結構描述新增配方驗證檢查。`component build` `component publish`此更新可協助開發人員在元件建立程序的早期元件配方中找出可行動的問題。 • 將置信度測試套件添加到可以通過`test-e2e init`命令拉下的模板。此置信度測試套件包括八個通用測試，可以使用和擴展以滿足基本組件測試需求。 錯誤修復和改進 • 將`test-e2e`命令使用的默認 Greengrass 測試框架（GTF）版本更新為 1.2.0 版本。

版本	變更
1.5.0	錯誤修復和改進 如果是，則更新excludes建置選項識別的模build_system 式zip。此版本現在將根據通配符符識別匹配路徑名的 glob 模式。這會啟用要排除哪些目錄的自訂規格。
1.4.0	新功能 • 添加一個新config命令，該命令啟動交互式提示以修改現有 GDK 配置文件中的字段。 • 在繼續之前，修改gdk component build和gdk component publish命令以驗證方案大小是否在 Greengrass 要求範圍內（<= 16000 字節）。 錯誤修復和改進 • 當配方語法錯誤阻止構建完成以進行感知時，在gdk component build命令的輸出中添加其他日誌記錄。 • 由於otf-version 將開放測試框架重命名為 Greengrass 測試框架，因此將gtf-options 和gtf-version 分別重命名為和。otf-options
1.3.0	新功能 • 添加一個新test-e2e命令，以支持使用開放 end-to-end 測試框架組件的測試。 • 加入新的組態選項zip_name，以透過 zip 建置系統支援可設定的 zip 檔案名稱。 • 將 GDK 配置檔案中的region屬性設為可選。 錯誤修復和改進 • 修正使用引數初始化 GDK 專案時，即使指定的範本或儲存庫不存在，也會建立新目錄的--name問題。
1.2.3	錯誤修復和改進 • 修正值區建立因錯誤處理不正確而失敗的問題。 • 修復了組件配方中的列表結構被刪除的問題。

版本	變更
1.2.2	錯誤修復和改進 • 配方鍵不再區分大小寫。 • 在建立新值區之前，新增檢查以判斷值區中是否存在值區中，AWS 區域且使用者可以存取。要求使用者擁有`GetBucketLocation` 權限。 • 修正 GDK CLI 組態檔中`excludes`關鍵字的問題。
1.2.1	錯誤修復和改進 • 接受`gdk-config.json` 檔案中區域組態項目AWS 區域中的加拿大 (中部`ca-central-1`) ()。 • 修正`publish`指令的 `--region` GDK CLI 引數的問題。
1.2.0	新功能 • 將`options`項目新增至 GDK CLI `build` 組態檔中的組態。支持`excludes`在`options`使用`zip`構建系統時從 zip 工件中排除某些文件。 • 添加`gradlew`構建系統以使用搖籃包裝來構建組件。 • 為構建選`gradle`項添加對 Kotlin DSL 構建文件的支持。 • 將項`options`目新增至 GDK CLI `publish` 組態檔中的組態。在將檔案上傳`options`到 Amazon S3 時，支援`file_upload_args` 下列項目以提供額外的引數。 錯誤修復和改進 • 修復了 Gradle 構建在運行構建命令之前未清理的問題。 • 修復了構建命令失敗時構建未退出的問題。 • 改進了`gdk component list`命令的輸出格式。

版本	變更
1.1.0	新功能 • 添加對搖籃構建系統的支持。 • 添加對 Windows 設備上的 Maven 構建系統的支持。 • 將--bucket引數加入至元件發佈指令。您可以使用此引數來指定 GDK CLI 上傳元件成品的確切值區。 • 將--name參數添加到組件 init 命令。您可以使用此選項來指定 GDK CLI 初始化元件的資料夾。 • 添加對 S3 存儲桶中但不存在於本地組件構建文件夾中的組件成品的支持。您可以使用此功能來降低大型元件成品的頻寬成本，例如機器學習模型。 錯誤修復和改進 • 更新元件發佈指令，以檢查元件是否在發佈元件之前建置。如果未建立元件，此指令現在會為您建置元件。 • 修正 ZIP 檔案名稱包含大寫字母時，zip 建置系統無法在 Windows 裝置上建置的問題。 • 改進了日誌消息格式，並將默認日誌級別更改為INFO在運行 Python 版本比 3.8 以前的設備上。 • 將最低版本要求更改為 Python 3.6。
1.0.0	初始版本。

安裝或更新AWS IoT Greengrass開發套件命令列介面

開AWS IoT Greengrass發套件命令列介面 (GDK CLI) 是建立在 Python 上，因此您可以使用pip它安裝在您的開發電腦上。

> **Tip**
>
> 您也可以將 GDK CLI 安裝在 Python 虛擬環境中，例如 VEN V。如需詳細資訊，請參閱 Python 3 文件中的虛擬環境和套件。

若要安裝或更新 GDK CLI

1. 執行下列命令，從其GitHub儲存庫安裝最新版本的 GDK CLI。

```
python3 -m pip install -U git+https://github.com/aws-greengrass/aws-greengrass-gdk-cli.git@v1.6.2
```

> **Note**
>
> 若要安裝特定版本的 GDK CLI，請將####取代為要安裝的版本標籤。您可以在其GitHub存放庫中檢視 GDK CLI 的版本標籤。
>
> ```
> python3 -m pip install -U git+https://github.com/aws-greengrass/aws-greengrass-gdk-cli.git@versionTag
> ```

2. 執行下列命令以驗證 GDK CLI 是否已成功安裝。

```
gdk --help
```

如果找不到該gdk命令，請將其文件夾添加到 PATH。

- 在 Linux 設備上，添加/home/*MyUser*/.local/bin到 PATH，並替換為您*MyUser*的用戶的名稱。
- 在 Windows 設備上，添加*PythonPath*\\Scripts到路徑，並*PythonPath*替換為設備上 Python 文件夾的路徑。

您現在可以使用 GDK CLI 來建立、建置和發佈 Greengrass 元件。如需如何使用 GDK CLI 的詳細資訊，請參閱AWS IoT Greengrass開發套件命令列介面命令。

AWS IoT Greengrass開發套件命令列介面命令

AWS IoT Greengrass開發套件命令列介面 (GDK CLI) 提供了一個命令列介面，您可以用來在開發電腦上建立、建置和發佈 Greengrass 元件。GDK CLI 命令使用以下格式。

```
gdk <command> <subcommand> [arguments]
```

當您安裝 GDK CLI 時，安裝程式會新增gdk至 PATH，以便您可以從命令列執行 GDK CLI。

您可以將下列引數與任何命令搭配使用：

- `--help`如需 GDK CLI 命令的相關資訊，請使用`-h`或。
- 使用`-v`或`--version`查看安裝的 GDK CLI 版本。
- 使用`-d`或輸出可用`--debug`來偵錯 GDK CLI 的詳細資訊記錄。

本節說明 GDK CLI 命令，並提供每個命令的範例。每個指令的摘要都會顯示其引數及其用法。可選參數顯示在方括號中。

可用命令

- 元件
- config
- 測試 E2E

元件

使用開AWS IoT Greengrass發套件`component`命令列介面 (GDK CLI) 中的命令來建立、建置和發佈自訂 Greengrass 元件。

子命令

- init
- build
- 發布
- 列出

init

從元件範本或社群元件初始化 Greengrass 元件資料夾。

GDK CLI 會從 Greengrass 軟體目錄擷取社群元件，並從上的元件範本存放庫擷取元AWS IoT Greengrass件範本。 GitHub

> **Note**
>
> 如果您使用 GDK CLI v1.0.0，則必須在空白資料夾中執行此命令。GDK CLI 會將範本或社群元件下載到目前的資料夾。

如果您使用 GDK CLI v1.1.0 或更新版本，您可以指定`--name`引數來指定 GDK CLI 下載範本或社群元件的資料夾。如果您使用此引數，請指定不存在的資料夾。GDK CLI 會為您建立資料夾。如果您未指定此引數，GDK CLI 會使用目前的資料夾，該資料夾必須為空白。
如果元件使用 zip 建置系統，GDK CLI 會將元件資料夾中的某些檔案壓縮為與元件資料夾名稱相同的 zip 檔案。例如，如果元件資料夾的名稱是`HelloWorld`，GDK CLI 會建立名`HelloWorld.zip`為的 zip 檔案。在元件方案中，zip 人工因素名稱必須與元件資料夾的名稱相符。如果您在 Windows 裝置上使用 GDK CLI 1.0.0 版，元件資料夾和壓縮檔案名稱必須只包含小寫字母。
如果您將使用 zip 建置系統的範本或社群元件初始化到名稱不同於範本或元件的資料夾，您必須變更元件方案中的 zip 成品名稱。更新`Artifacts`和定`Lifecycle`義，使 zip 檔案名稱與元件資料夾的名稱相符。下列範例會反白`Artifacts`和`Lifecycle`定義中的 zip 檔案名稱。

JSON

```
{
  ...
  "Manifests": [
    {
      "Platform": {
        "os": "all"
      },
      "Artifacts": [
        {
          "URI": "s3://BUCKET_NAME/COMPONENT_NAME/
COMPONENT_VERSION/HelloWorld.zip",
          "Unarchive": "ZIP"
        }
      ],
      "Lifecycle": {
        "run": "python3 -u {artifacts:decompressedPath}/HelloWorld/main.py
 {configuration:/Message}"
      }
    }
  ]
}
```

YAML

```
---
...
Manifests:
```

```
  - Platform:
      os: all
    Artifacts:
      - URI: "s3://BUCKET_NAME/COMPONENT_NAME/
COMPONENT_VERSION/HelloWorld.zip"
        Unarchive: ZIP
    Lifecycle:
      run: "python3 -u {artifacts:decompressedPath}/HelloWorld/main.py
 {configuration:/Message}"
```

概要

```
$ gdk component init
    [--language]
    [--template]
    [--repository]
    [--name]
```

參數（從組件模板初始化）

- `-l`、`--language` — 用於指定範本的程式設計語言。

  您必須指定`--repository`或`--language`和`--template`。
- `-t`、`--template` — 用於本機元件專案的元件範本。若要檢視可用的樣板，請使用 list 指令。

  您必須指定`--repository`或`--language`和`--template`。
- `-n`, `--name` — (選擇性) GDK CLI 初始化元件的本機資料夾名稱。指定不存在的資料夾。GDK CLI 會為您建立資料夾。

  此功能適用於 GDK CLI 1.1.0 版及更新版本。

參數（從社區組件初始化）

- `-r`、`--repository` — 要簽出至本機資料夾的社群元件。若要檢視可用的社群元件，請使用 list 指令。

  您必須指定`--repository`或`--language`和`--template`。
- `-n`, `--name` — (選擇性) GDK CLI 初始化元件的本機資料夾名稱。指定不存在的資料夾。GDK CLI 會為您建立資料夾。

  此功能適用於 GDK CLI 1.1.0 版及更新版本。

輸出

下面的例子顯示當您運行此命令從 Python Hello World 模板初始化組件文件夾產生的輸出。

```
$ gdk component init -l python -t HelloWorld
[2021-11-29 12:51:40] INFO - Initializing the project directory with a python
 component template - 'HelloWorld'.
[2021-11-29 12:51:40] INFO - Fetching the component template 'HelloWorld-python'
 from Greengrass Software Catalog.
```

下列範例顯示執行此命令以從社群元件初始化元件資料夾時產生的輸出。

```
$ gdk component init -r aws-greengrass-labs-database-influxdb
[2022-01-24 15:44:33] INFO - Initializing the project directory with a component
 from repository catalog - 'aws-greengrass-labs-database-influxdb'.
[2022-01-24 15:44:33] INFO - Fetching the component repository 'aws-greengrass-labs-
database-influxdb' from Greengrass Software Catalog.
```

build

將元件的來源建置到可發佈至AWS IoT Greengrass服務的方案和成品中。GDK CLI 會執行您在 GDK CLI 設定檔中指定的建置系統。`gdk-config.json`您必須在`gdk-config.json`檔案所在的相同資料夾中執行此命令。

當您執行此命令時，GDK CLI 會在元件資料夾的`greengrass-build`資料夾中建立方案和成品。GDK CLI 會將配方儲存在`greengrass-build/recipes`資料夾中，並將成品儲存在`greengrass-build/artifacts/`*`componentName`*`/`*`componentVersion`*資料夾中。

如果您使用 GDK CLI v1.1.0 或更新版本，元件方案可以指定存在於 S3 儲存貯體中但不存在於本機元件建置資料夾中的成品。當您開發含有大型成品的元件 (例如機器學習模型) 時，您可以使用此功能來減少頻寬使用量。

構建組件之後，您可以執行以下操作之一來在 Greengrass 核心設備上對其進行測試：

- 如果您在與執行 AWS IoT Greengrass Core 軟體的裝置不同的裝置上進行開發，則必須發佈元件以將其部署到 Greengrass 核心裝置。將元件發佈到AWS IoT Greengrass服務，並將其部署到 Greengrass 核心裝置。如需詳細資訊，請參閱發佈命令和建立部署。
- 如果您在執行 AWS IoT Greengrass Core 軟體的相同裝置上進行開發，則可以將元件發佈到要部署的AWS IoT Greengrass服務，或者您可以建立本機部署以安裝並執行元件。若要建立本機部署，請使用 Greengrass CLI。如需詳細資訊，請參閱 綠色命令行界面 及 使用本機部署測試AWS IoT

Greengrass元件。當您建立本機部署時，請指定greengrass-build/recipes為 recipe 資料夾和greengrass-build/artifacts成品資料夾。

概要

```
$ gdk component build
```

Arguments (引數)

無

輸出

下列範例顯示執行此命令時產生的輸出。

```
$ gdk component build
[2021-11-29 13:18:49] INFO - Getting project configuration from gdk-config.json
[2021-11-29 13:18:49] INFO - Found component recipe file 'recipe.yaml' in the
 project directory.
[2021-11-29 13:18:49] INFO - Building the component 'com.example.PythonHelloWorld'
 with the given project configuration.
[2021-11-29 13:18:49] INFO - Using 'zip' build system to build the component.
[2021-11-29 13:18:49] WARNING - This component is identified as using 'zip' build
 system. If this is incorrect, please exit and specify custom build command in the
 'gdk-config.json'.
[2021-11-29 13:18:49] INFO - Zipping source code files of the component.
[2021-11-29 13:18:49] INFO - Copying over the build artifacts to the greengrass
 component artifacts build folder.
[2021-11-29 13:18:49] INFO - Updating artifact URIs in the recipe.
[2021-11-29 13:18:49] INFO - Creating component recipe in 'C:\Users\MyUser\Documents
\greengrass-components\python\HelloWorld\greengrass-build\recipes'.
```

發布

將此元件發佈至AWS IoT Greengrass服務。此命令會將組建成品上傳到 S3 儲存貯體、更新方案中的成品 URI，並從方案建立新版本的元件。GDK CLI 會使用您在 GDK CLI 組態檔案中指定的 S3 儲存貯體和AWS區域。gdk-config.json您必須在gdk-config.json檔案所在的相同資料夾中執行此命令。

如果您使用 GDK CLI v1.1.0 或更新版本，則可以指定--bucket引數來指定 GDK CLI 上傳元件成品的 S3 儲存貯體。如果您未指定此引數，GDK CLI 會上傳至名稱為的 S3 儲存貯

體*bucket*-*region*-*accountId*，其中儲存#體和##是您在中指定的值gdk-config.json，而 *accountId* 就是您AWS 帳戶的 ID。GDK CLI 會建立值區 (如果值區不存在)。

如果您使用 GDK CLI v1.2.0 或更新版本，您可以使用參數覆寫 GDK CLI 組態檔中AWS 區域指定的。--region您也可以使用--options參數指定其他選項。如需可用選項的清單，請參閱Greengrass 開發工具包 CLI 配置文件。

當您執行此命令時，GDK CLI 會以您在方案中指定的版本發行元件。如果您指定NEXT_PATCH，GDK CLI 會使用下一個尚未存在的修補程式版本。語義版本使用一個主要的。 未成年人。 補丁編號系統。如需詳細資訊，請參閱語意版本規格。

> **Note**
>
> 如果您使用 GDK CLI v1.1.0 或更新版本，當您執行此命令時，GDK CLI 會檢查元件是否已建置。如果未建置元件，GDK CLI 會在發佈元件之前建立元件。

概要

```
$ gdk component publish
    [--bucket] [--region] [--options]
```

Arguments (引數)

- -b，--bucket— (選擇性) 指定 GDK CLI 發佈元件成品所在 S3 儲存貯體的名稱。

  如果您未指定此引數，GDK CLI 會上傳至名稱為的 S3 儲存貯體*bucket*-*region*-*accountId*，其中儲存#體和##是您在中指定的值gdk-config.json，而 *accountId* 就是您AWS 帳戶的 ID。GDK CLI 會建立值區 (如果值區不存在)。

  GDK CLI 會建立值區 (如果值區不存在)。

  此功能適用於 GDK CLI 1.1.0 版及更新版本。

- -r，--region— (選擇性) 指定建立元AWS 區域件時的目標名稱。此引數會覆寫 GDK CLI 組態中的區域名稱。

  此功能適用於 GDK CLI 1.2.0 版及更新版本。

- -o，--options(選擇性) 指定用於發佈元件的選項清單。引數必須是包含發佈選項的 JSON 檔案的有效 JSON 字串或檔案路徑。此引數會覆寫 GDK CLI 組態中的選項。

  此功能適用於 GDK CLI 1.2.0 版及更新版本。

輸出

下列範例顯示執行此命令時產生的輸出。

```
$ gdk component publish
[2021-11-29 13:45:29] INFO - Getting project configuration from gdk-config.json
[2021-11-29 13:45:29] INFO - Found component recipe file 'recipe.yaml' in the
 project directory.
[2021-11-29 13:45:29] INFO - Found credentials in shared credentials file: ~/.aws/
credentials
[2021-11-29 13:45:30] INFO - Publishing the component 'com.example.PythonHelloWorld'
 with the given project configuration.
[2021-11-29 13:45:30] INFO - No private version of the component
 'com.example.PythonHelloWorld' exist in the account. Using '1.0.0' as the next
 version to create.
[2021-11-29 13:45:30] INFO - Uploading the component built artifacts to s3 bucket.
[2021-11-29 13:45:30] INFO - Uploading component artifacts to S3 bucket: {bucket}.
 If this is your first time using this bucket, add the 's3:GetObject' permission
 to each core device's token exchange role to allow it to download the component
 artifacts. For more information, see https://docs.aws.amazon.com/greengrass/v2/
developerguide/device-service-role.html.
[2021-11-29 13:45:30] INFO - Not creating an artifacts bucket as it already exists.
[2021-11-29 13:45:30] INFO - Updating the component recipe
 com.example.PythonHelloWorld-1.0.0.
[2021-11-29 13:45:30] INFO - Creating a new greengrass component
 com.example.PythonHelloWorld-1.0.0
[2021-11-29 13:45:30] INFO - Created private version '1.0.0' of the component in the
 account.'com.example.PythonHelloWorld'.
```

列出

擷取可用元件範本和社群元件的清單。

GDK CLI 會從 Greengrass 軟體目錄擷取社群元件，並從上的元件範本存放庫擷取元AWS IoT Greengrass件範本。 GitHub

您可以將此命令的輸出傳遞給 init 命令，以從模板和社區組件初始化組件存儲庫。

概要

```
$ gdk component list
    [--template]
```

```
    [--repository]
```

Arguments (引數)

- `-t, --template` — (選擇性) 指定此引數以列出可用的元件範本。此指令會以格式輸出每個範本的名稱和語言*name*-*language*。例如，在中`HelloWorld-python`，範本名稱為`HelloWorld`，語言為`python`。
- `-r, --repository` — (選擇性) 指定此引數以列出可用的社群元件存放庫。

輸出

下列範例顯示執行此命令時產生的輸出。

```
$ gdk component list --template
[2021-11-29 12:29:04] INFO - Listing all the available component templates from
 Greengrass Software Catalog.
[2021-11-29 12:29:04] INFO - Found '2' component templates to display.
1. HelloWorld-python
2. HelloWorld-java
```

config

使用AWS IoT Greengrass開發套件`config`命令列介面 (GDK CLI) 中的命令來修改設定檔中 GDK 的組態。`gdk-config.json`

子命令

- update

update

啟動互動式提示以修改現有 GDK 組態檔案中的欄位。

概要

```
$ gdk config update
    [--component]
```

Arguments (引數)

- `-c`，`--component`— 更新檔案中與元件相關的欄位。`gdk-config.json`此參數是必需的，因為它是唯一的選擇。

輸出

下列範例顯示執行此命令來設定元件時所產生的輸出。

```
$ gdk config update --component
Current value of the REQUIRED component_name is (default:
 com.example.PythonHelloWorld):
Current value of the REQUIRED author is (default: author):
Current value of the REQUIRED version is (default: NEXT_PATCH):
Do you want to change the build configurations? (y/n)
Do you want to change the publish configurations? (y/n)
[2023-09-26 10:19:48] INFO - Config file has been updated. Exiting...
```

測試 E2E

使用開AWS IoT Greengrass發套件`test-e2e`命令列介面 (GDK CLI) 中的命令，在 GDK 專案中初始化、建置和執行 end-to-end 測試模組。

子命令

- init
- build
- run

init

使用使用 Greengrass 測試框架（GTF）的測試模塊初始化現有的 GDK CLI 項目。

默認情況下，GDK CLI 檢索從組AWS IoT Greengrass件模板存儲庫上 GitHub的 Maven 模塊模板。這個 Maven 模塊帶有對 `aws-greengrass-testing-standalone` JAR 文件的依賴關係。

這個命令創建一個名為 GDK 項目`gg-e2e-tests`內的新目錄。如果 test 模組目錄已存在且不為空，則指令會結束而不執行任何動作。此`gg-e2e-tests`文件夾包含在 Maven 項目結構的黃瓜功能和步驟定義。

默認情況下，該命令將嘗試使用最新版本的 GTF。

概要

```
$ gdk test-e2e init
    [--gtf-version]
```

Arguments (引數)

- `-ov, --gtf-version` — (選用) GDK 專案中與 end-to-end 測試模組搭配使用的 GTF 版本。該值必須是發行版本的 GTF 版本之一。此引數會覆寫 GDK CLI 組態`gtf_version`中的。

輸出

下面的例子顯示當您運行此命令與測試模塊初始化 GDK 項目產生的輸出。

```
$ gdk test-e2e init
[2023-12-06 12:20:28] INFO - Using the GTF version provided in the GDK test config
 1.2.0
[2023-12-06 12:20:28] INFO - Downloading the E2E testing template from GitHub into
 gg-e2e-tests directory...
```

build

> **Note**
>
> 在構建 end-to-end 測試模塊gdk component build之前，您必須通過運行構建組件。

構建測 end-to-end 試模塊。GDK CLI 會使用您在 GDK CLI 組態檔案中指定的建置系統，`gdk-config.json`在屬性下建置測試模組。`test-e2e`您必須在`gdk-config.json`檔案所在的相同資料夾中執行此命令。

默認情況下，GDK CLI 使用 Maven 構建系統來構建測試模塊。Maven 需要運行該`gdk test-e2e build`命令。

如果測試功能文件具有類似`GDK_COMPONENT_NAME`和插值的變量，則必須在構建測試模塊gdk-component-build之前運行構`GDK_COMPONENT_RECIPE_FILE`建組件。

當您執行此命令時，GDK CLI 會插入 GDK 專案組態中的所有變數，並建置`gg-e2e-tests`模組以產生最終的測試 JAR 檔案。

概要

```
$ gdk test-e2e build
```

Arguments (引數)

無

輸出

下列範例顯示執行此命令時產生的輸出。

```
$ gdk test-e2e build
[2023-07-20 15:36:48] INFO - Updating feature file: file:///path/to//
HelloWorld/greengrass-build/gg-e2e-tests/src/main/resources/greengrass/features/
component.feature
[2023-07-20 15:36:48] INFO - Creating the E2E testing recipe file:///path/to/
HelloWorld/greengrass-build/recipes/e2e_test_recipe.yaml
[2023-07-20 15:36:48] INFO - Building the E2E testing module
[2023-07-20 15:36:48] INFO - Running the build command 'mvn package'
.........
```

run

使用 GDK 配置文件中的測試選項運行測試模塊。

> **Note**
>
> 在運行測試gdk test-e2e build之前，您必須通過運行來構建測 end-to-end 試模塊。

概要

```
$ gdk test-e2e run
    [--gtf-options]
```

Arguments (引數)

- `-oo, --gtf-options` — (選擇性) 指定用於執行 end-to-end 測試的選項清單。引數必須是有效的 JSON 字串或包含 GTF 選項之 JSON 檔案的檔案路徑。組態檔案中提供的選項會與命令引數中提供的選項合併。如果兩個選項都存在，則參數中的一個選項會優先於配置文件中的選項。

  如果未在此命令中指定該`tags`選項，GDK 將用`Sample`於標籤。如果`ggc-archive`未指定，GDK 會下載最新版本的 Greengrass 核壓縮檔。

輸出

下列範例顯示執行此命令時產生的輸出。

```
$ gdk test-e2e run
```

```
[2023-07-20 16:35:53] INFO - Downloading latest nucleus archive from url https://
d2s8p88vqu9w66.cloudfront.net/releases/greengrass-latest.zip
[2023-07-20 16:35:57] INFO - Running test jar with command java -jar /path/to/
greengrass-build/gg-e2e-tests/target/uat-features-1.0.0.jar –ggc-archive=/path/to/
aws-greengrass-gdk-cli/HelloWorld/greengrass-build/greengrass-nucleus-latest.zip –
tags=Sample

16:35:59.693 [] [] [] [INFO]
 com.aws.greengrass.testing.modules.GreengrassContextModule - Extracting /path/
to/workplace/aws-greengrass-gdk-cli/HelloWorld/greengrass-build/greengrass-
nucleus-latest.zip into /var/folders/7g/ltzcb_3s77nbtmkzfb6brwv40000gr/T/gg-
testing-7718418114158172636/greengrass
16:36:00.534 [gtf-1.1.0-SNAPSHOT] [] [] [INFO]
 com.aws.greengrass.testing.features.LoggerSteps - GTF Version is gtf-1.1.0-SNAPSHOT
.......
```

Greengrass 開發工具包 CLI 配置文件

開AWS IoT Greengrass發套件命令列介面 (GDK CLI) 會從名為建置和發佈元件`gdk-config.json`的設定檔讀取。此組態檔案必須存在於元件儲存庫的根目錄中。您可以使用 GDK CLI init 指令，以此設定檔初始化元件儲存庫。

主題

- GDK CLI 設定檔案格式
- GDK CLI 設定檔範例

GDK CLI 設定檔案格式

當您為元件定義 GDK CLI 組態檔時，請以 JSON 格式指定下列資訊。

`gdk_version`

與此元件相容的 GDK CLI 的最低版本。此值必須是發行版本的其中一個 GDK CLI 版本。

`component`

此零組件的模型組態。

`componentName`

`author`

元件的作者或發行者。

`version`

元件的版本。請指定下列其中一項：

- `NEXT_PATCH`— 當您選擇此選項時，GDK CLI 會在您發佈元件時設定版本。GDK CLI 會查詢AWS IoT Greengrass服務，以識別元件的最新發佈版本。然後，它會將版本設定為該版本之後的下一個修補程式版本。如果您之前尚未發佈元件，GDK CLI 會使用版本`1.0.0`。

  如果您選擇此選項，則無法使用 Greengrass CLI 在本機部署元件，並將元件測試到執行 Core 軟體的本機開發電腦。AWS IoT Greengrass若要啟用本機部署，您必須改為指定語意版本。

- 語義版本，例如**`1.0.0`**. 語義版本使用一個主要的。 未成年人。 補丁編號系統。如需詳細資訊，請參閱語意版本規格。

  如果您在要部署和測試元件的 Greengrass 核心裝置上開發元件，請選擇此選項。您必須使用特定版本建立元件，才能使用 Greengr ass CLI 建立本機部署。

`build`

用來將此元件的來源建置為成品的組態。此物件包含下列資訊：

`build_system`

要使用的建置系統。您可以從以下選項中選擇：

- `zip`— 將元件的資料夾封裝至 ZIP 檔案，以定義為元件的唯一人工因素。針對下列類型的元件選擇此選項：
  - 使用解譯式程式設計語言的元件，例如 Python 或 JavaScript.
  - 封裝程式碼以外檔案的元件，例如機器學習模型或其他資源。

  GDK CLI 會將元件的資料夾壓縮成與元件資料夾名稱相同的 zip 檔案。例如，如果元件資料夾的名稱是`HelloWorld`，GDK CLI 會建立一個名`HelloWorld.zip`為的 zip 檔案。

  > **Note**
  >
  > 如果您在 Windows 裝置上使用 GDK CLI 1.0.0 版，元件資料夾和壓縮檔案名稱必須只包含小寫字母。

  當 GDK CLI 將元件的資料夾壓縮至 zip 檔案時，會略過下列檔案：

  - gdk-config.json 檔案
  - 配方文件（recipe.json或recipe.yaml）
  - 建置資料夾，例如 greengrass-build
- maven— 執行指mvn clean package令，將元件的來源建置為成品。針對使用 Maven 的元件 (例如 Java 元件) 選擇此選項。

  在視窗裝置上，此功能適用於 GDK CLI 1.1.0 版及更新版本。
- gradle— 執行指gradle build令，將元件的來源建置為成品。對於使用搖籃的組件，請選擇此選項。此功能適用於 GDK CLI 1.1.0 版及更新版本。

  gradle構建系統支持 Kotlin DSL 作為構建文件。此功能適用於 GDK CLI 1.2.0 版及更新版本。
- gradlew— 執行指gradlew令，將元件的來源建置為成品。針對使用 Gradle 包裝函式的元件選擇此選項。

  此功能適用於 GDK CLI 1.2.0 版及更新版本。
- custom— 執行自訂命令，將元件的來源建置到配方和成品中。在custom_build_command參數中指定自訂指令。

custom_build_command

(選擇性) 要針對自訂建置系統執行的自訂建置命令。如果您指定custom為，則必須指定此參數build_system。

> **Important**
>
> 此指令必須在元件資料夾內的下列資料夾中建立方案和人工因素。當您執行元件建置命令時，GDK CLI 會為您建立這些資料夾。
>
> - 食譜資料夾：greengrass-build/recipes
> - 成品資料夾：greengrass-build/artifacts/*componentName*/*componentVersion*
>
>   以####取代元件名稱，並以元件版本或取#####。NEXT_PATCH

您可以指定單一字串或字串清單，其中每個字串都是指令中的一個字。例如，若要執行 C++ 元件的自訂建置命令，您可以指定**cmake --build build --config Release**或**["cmake", "--build", "build", "--config", "Release"]**。

要查看自定義構建系統的示例，請參閱（詳見）GitHub。aws.greengrass.labs.LocalWebServer community component

`options`

(選擇性) 在元件建置程序期間使用的其他組態選項。

此功能適用於 GDK CLI 1.2.0 版及更新版本。

`excludes`

glob 模式清單，用於定義在構建 zip 文件時從組件目錄中排除哪些文件。只有在是時才`build_system`有效`zip`。

> **Note**
>
> 在 GDK CLI 版本 1.4.0 及更早版本中，任何符合排除清單中項目的檔案都會從元件的所有子目錄中排除。若要在 GDK CLI 1.5.0 版及更新版本中達`**/`到相同的行為，請在排除清單中的現有項目前加上。例如，`*.txt`將僅從目錄中排除文本文件; `**/*.txt` 將從所有目錄和子目錄中排除文本文件。
> 在 GDK CLI 版本 1.5.0 及更新版本中，如果在 GDK 配置檔案中定義，您可能會在元件建置期間看到警告。`excludes`若要停用此警告，請將環境變數設定`GDK_EXCLUDES_WARN_IGNORE`為`true`。

GDK CLI 一律會從壓縮檔案中排除下列檔案：

- `gdk-config.json` 檔案
- 配方文件（`recipe.json`或`recipe.yaml`）
- 建置資料夾，例如 `greengrass-build`

依預設，會排除下列檔案。但是，您可以使用`excludes`此選項控制要排除哪些檔案。

- 以前綴「test」(`test*`) 開頭的任何文件夾
- 所有隱藏文件
- `node_modules` 資料夾

如果您指定`excludes`選項，GDK CLI 只會排除您使用該`excludes`選項設定的檔案。如果您未指定`excludes`選項，GDK CLI 會排除先前指出的預設檔案和資料夾。

`zip_name`

在建置程序期間建立 zip 加工品時要使用的 zip 檔案名稱。只有在是時才`build_system`有效`zip`。如果`build_system`為空，則會將元件名稱用於 zip 檔案名稱。

`publish`

用來將此元件發行至AWS IoT Greengrass服務的組態。

如果您使用 GDK CLI v1.1.0 或更新版本，則可以指定`--bucket`引數來指定 GDK CLI 上傳元件成品的 S3 儲存貯體。如果您未指定此引數，GDK CLI 會上傳至名稱為的 S3 儲存貯體*`bucket`*`-`*`region`*`-`*`accountId`*，其中儲存`#`體和`##`是您在中指定的值`gdk-config.json`，而 *`accountId`* 是您AWS 帳戶的 ID。GDK CLI 會建立值區 (如果值區不存在)。

此物件包含下列資訊：

`bucket`

用於託管元件成品的 S3 儲存貯體名稱。

`region`

GDK CLI 發行此元件的AWS 區域位置。

如果您使用的是 GDK CLI v1.3.0 或更新版本，則此屬性是可選的。

`options`

(選擇性) 建立元件版本期間使用的其他組態選項。

此功能適用於 GDK CLI 1.2.0 版及更新版本。

`file_upload_args`

JSON 結構，其中包含將檔案上傳至儲存貯體時傳送至 Amazon S3 的引數，例如中繼資料和加密機制。如需允許引數的清單，請參閱 Boto3 文件中的`S3Transfer`類別。 。

`test-e2e`

(選擇性) 元件 end-to-end 測試期間要使用的組態。此功能適用於 GDK CLI 版本 1.3.0 及更新版本。

`build`

`build_system`— 要使用的建置系統。預設選項為`maven`。您可以從以下選項中選擇：

- maven-運行命`mvn package`令以構建測試模塊。選擇此選項以建立使用 Maven 的測試模組。
- gradle-運行命`gradle build`令以構建測試模塊。為使用 Gradle 的測試模塊選擇此選項。

`gtf_version`

（可選）當您使用 GTF 初始化 GDK 項目時，Greengrass 測試框架（GTF）的版本用作 end-to-end 測試模塊的依賴項。該值必須是發行版本的 GTF 版本之一。默認設置為 GTF 版本 1.1.0。

`gtf_options`

(選擇性) 在 end-to-end 測試元件期間使用的其他組態選項。

下列清單包含您可以在 GTF 1.1.0 版中使用的選項。

- `additional-plugins`-（可選）其他黃瓜插件
- `aws-region`— 針對AWS服務的特定區域端點。默認為 AWS SDK 發現的內容。
- `credentials-path`— 選擇性AWS設定檔認證路徑。預設為在主機環境中探索到的認證。
- `credentials-path-rotation`— AWS 認證的可選輪換持續時間。預設值為 15 分鐘或`PT15M`。
- `csr-path`— 將用來產生裝置憑證的 CSR 路徑。
- `device-mode`— 被測的目標設備。預設為本機裝置。
- `env-stage`— 以 Greengrass 的部署環境為目標。預設為生產。
- `existing-device-cert-arn`— 您要用作 Greengrass 裝置憑證的現有憑證的 arn。
- `feature-path`— 包含其他功能檔案的檔案或目錄。預設值是不使用其他功能檔案。
- `gg-cli-version`-覆寫格 Greengrass CLI 的版本。預設為在中找到的值`ggc.version`。
- `gg-component-bucket`— 容納 Greengrass 元件的現有 Amazon S3 儲存貯體的名稱。
- `gg-component-overrides`-綠色組件覆蓋的列表。
- `gg-persist`-測試運行後要持續存在的測試元素列表。默認行為是什麼都不堅持。接受的值為：`aws.resourcesinstalled.software`、和`generated.files`。
- `gg-runtime`— 影響測試與測試資源互動方式的值清單。這些值會取代參`gg.persist`數。如果默認值為空，則假定所有測試資源都由測試用例管理，包括安裝的 Greengrass 運行時。接受的值為：`aws.resourcesinstalled.software`、和`generated.files`。
- `ggc-archive`— 路徑歸檔的 Greengrass 核組件.
- `ggc-install-root`— 安裝 Greengrass 核組件的目錄. 默認為測試 .temp.path 和測試運行文件夾。

- `ggc-log-level`— 為測試運行設置 Greengrass 核日誌級別。預設值為「資訊」。
- `ggc-tes-rolename`— AWS IoT Greengrass 核心將承擔存取AWS服務的 IAM 角色。如果具有給定名稱的角色不存在，則將創建一個並默認訪問策略。
- `ggc-trusted-plugins`— 需要添加到 Greengrass 的受信任插件的路徑（在主機上）的逗號分隔列表。要提供 DUT 本身的路徑，請在路徑前加上「dut：」
- `ggc-user-name`— 使用者：群組 Greengrass 核的 PosixUser 值。預設為目前登入的使用者名稱。
- `ggc-version`— 覆寫正在執行的 Greengrass 核元件的版本。默認為在 ggc.archive 中找到的值。
- `log-level`— 測試運行的日誌級別。默認為「信息」。
- `parallel-config`— 設置批次索引和批次數作為 JSON 字符串。批處理索引的默認值為 0，批次數為 1。
- `proxy-url`— 配置所有測試以通過此 URL 路由流量。
- `tags`— 僅執行功能標籤。可以與 '&' 相交
- `test-id-prefix`— 應用於所有測試特定資源（包括資AWS源名稱和標籤）的通用前綴。預設值為「gg」前置詞。
- `test-log-path`— 將包含整個測試運行結果的目錄。默認為「測試結果」。
- `test-results-json`— 用於確定產生的黃瓜 JSON 報告是否產生寫入磁碟的旗標。預設為 true。
- `test-results-log`— 用於確定控制台輸出是否生成寫入磁盤的標誌。預設為 false。
- `test-results-xml`— 用於判斷產生的 JUnit XML 報告是否已產生寫入磁碟的旗標。預設為 true。
- `test-temp-path`— 產生本機測試成品的目錄。默認為以 gg-test 前綴的隨機臨時目錄。
- `timeout-multiplier`— 提供給所有測試超時的乘數。預設值為 1.0。

GDK CLI 設定檔範例

您可以參考下列 GDK CLI 設定檔範例，以協助您設定 Greengrass 元件環境。

你好世界 (Python)

下面的 GDK CLI 配置文件支持運行一個 Python 腳本你好世界組件。此組態檔案會使用`zip`建置系統，將元件的 Python 指令碼封裝成 ZIP 檔案，讓 GDK CLI 以成品的形式上傳。

```
{
```

```
  "component": {
    "com.example.PythonHelloWorld": {
      "author": "Amazon",
      "version": "NEXT_PATCH",
      "build": {
        "build_system" : "zip",
        "options": {
           "excludes": [".*"]
        }
      },
      "publish": {
        "bucket": "greengrass-component-artifacts",
        "region": "us-west-2",
        "options": {
           "file_upload_args": {
              "Metadata": {
                 "some-key": "some-value"
              }
           }
        }
      }
    },
  "test-e2e":{
    "build":{
        "build_system": "maven"
    },
    "gtf_version": "1.1.0",
    "gtf_options": {
         "tags": "Sample"
     }
  },
  "gdk_version": "1.6.1"
  }
}
```

你好世界

下面的 GDK CLI 配置文件支持運行 Java 應用程序的你好世界組件。此組態檔案會使用maven建置系統，將元件的 Java 原始程式碼封裝到 GDK CLI 上傳為成品的 JAR 檔案中。

```
{
  "component": {
    "com.example.JavaHelloWorld": {
```

```
      "author": "Amazon",
      "version": "NEXT_PATCH",
      "build": {
        "build_system" : "maven"
      },
      "publish": {
        "bucket": "greengrass-component-artifacts",
        "region": "us-west-2",
        "options": {
           "file_upload_args": {
              "Metadata": {
                 "some-key": "some-value"
              }
           }
        }
      }
  },
  "test-e2e":{
    "build":{
        "build_system": "maven"
    },
    "gtf_version": "1.1.0",
    "gtf_options": {
         "tags": "Sample"
     }
  },
  "gdk_version": "1.6.1"
  }
}
```

社群元件

Greengrass 軟體目錄中的幾個社群元件都使用 GDK CLI。您可以瀏覽這些元件儲存庫中的 GDK CLI 組態檔案。

若要檢視社群元件的 GDK CLI 組態檔

1. 執行下列命令以列出使用 GDK CLI 的社群元件。

```
gdk component list --repository
```

回應會列出使用 GDK CLI 之每個社群元件的 GitHub 存放庫名稱。每個存放庫都存在於awslabs組織中。

```
[2022-02-22 17:27:31] INFO - Listing all the available component repositories from
 Greengrass Software Catalog.
[2022-02-22 17:27:31] INFO - Found '6' component repositories to display.
1. aws-greengrass-labs-database-influxdb
2. aws-greengrass-labs-telemetry-influxdbpublisher
3. aws-greengrass-labs-dashboard-grafana
4. aws-greengrass-labs-dashboard-influxdb-grafana
5. aws-greengrass-labs-local-web-server
6. aws-greengrass-labs-lookoutvision-gstreamer
```

2. 在下列 URL 開啟社群元件的 GitHub 儲存庫。取代*community-component-name*為上一個步驟中社群元件的名稱。

```
https://github.com/awslabs/community-component-name
```

綠色命令行界面

Greengrass 命令列介面 (CLI) 可讓您與裝置上的 AWS IoT Greengrass Core 互動，以便在本機開發元件和偵錯問題。例如，您可以使用 Greengrass CLI 建立本機部署，並重新啟動核心裝置上的元件。

部署綠色 CLI 元件 (aws.greengrass.Cli) 以在您的核心裝置上安裝 Greengrass CLI。

> **Important**
>
> 我們建議您僅在開發環境中使用此元件，而不是在生產環境中使用。此元件可讓您存取通常在生產環境中不需要的資訊和作業。只將此元件部署到您需要的核心裝置，以遵循最低權限原則。

主題

- 安裝 Greengrass CLI
- Greengrass CLI 命令

安裝 Greengrass CLI

您可以使用下列其中一種方式來安裝綠色 CLI：

- 首次在裝置上設定 AWS IoT Greengrass Core 軟體時，請使用`--deploy-dev-tools`引數。您也必須指`--provision true`定套用此引數。
- 在您的裝置上部署綠色 CLI 元件 (`aws.greengrass.Cli`)。

本節說明部署綠色 CLI 元件的步驟。如需在初始設定期間安裝 Greengrass CLI 的相關資訊，請參閱。教學課程：AWS IoT Greengrass V2 入門

必要條件

若要部署 Greengrass CLI 元件，您必須符合下列需求：

- AWS IoT Greengrass 在您的核心裝置上安裝並設定核心軟體。如需詳細資訊，請參閱 教學課程：AWS IoT Greengrass V2 入門。
- 若要使用 AWS CLI 來部署 Greengrass CLI，您必須已安裝並設定. AWS CLI如需詳細資訊，請參閱《AWS Command Line Interface 使用者指南》中的設定 AWS CLI。
- 您必須獲得授權才能使用 Greengrass CLI 與核心軟體進行互動 AWS IoT Greengrass 。執行下列其中一項動作以使用 Greengrass CLI：
  - 使用執行 AWS IoT Greengrass Core 軟體的系統使用者。
  - 使用具有 root 權限或管理權限的使用者。在 Linux 核心裝置上，您可以使用`sudo`來取得根權限。
  - 部署元件時，請使用位於`AuthorizedPosixGroups`或組`AuthorizedWindowsGroups`態參數中指定之群組中的系統使用者。如需詳細資訊，請參閱 Greengrass CLI 元件組態。

部署綠色 CLI 元件

完成下列步驟，即可將 Greengrass CLI 元件部署到您的核心裝置：

若要部署綠色 CLI 元件 (主控台)

1. 登入 AWS IoT Greengrass 主控台。
2. 在導覽功能表中，選擇 [元件]。
3. 在元面頁面上的公用元件索引標籤上，選擇 `aws.greengrass.Cli`。
4. 在 aws.greengrass.Cli 頁面中，選擇部署。
5. 從 [新增至部署] 中，選擇 [建立新部署]。

6. 在「指定目標」頁面的「建置目標」下的「目標名稱」清單中，選擇您要建置的 Greengrass 群組，然後選擇下一步。
7. 在 [選取元件] 頁面上，確認已選取aws.greengrass.Cli元件，然後選擇 [下一步]。
8. 在 [設定元件] 頁面上，保留預設組態設定，然後選擇 [下一步]。
9. 在 [設定進階設定] 頁面上，保留預設組態設定，然後選擇 [下一步]。
10. 在「複查」頁面上，按一下部署

若要部署綠色 CLI 元件 ()AWS CLI

1. 在您的裝置上，建立deployment.json檔案以定義 Greengrass CLI 元件的部署組態。此檔案應如下所示：

```
{
  "targetArn":"targetArn",
  "components": {
    "aws.greengrass.Cli": {
      "componentVersion": "2.12.3",
      "configurationUpdate": {
        "merge": "{\"AuthorizedPosixGroups\":\"<group1>,<group2>,...,<groupN>\",
\"AuthorizedWindowsGroups\":\"<group1>,<group2>,...,<groupN>\"}"
      }
    }
  }
}
```

- 在 target 欄位中，*targetArn*以下列格式將物件或物群組的 Amazon Resource Name (ARN) 取代為目標部署：
  - 物件：arn:aws:iot:*region*:*account-id*:thing/*thingName*
  - 物件群組：arn:aws:iot:*region*:*account-id*:thinggroup/*thingGroupName*
- 在aws.greengrass.Cli組件物件中，依下列方式指定值：

  `version`

  Greengrass CLI 組件的版本。

  `configurationUpdate.AuthorizedPosixGroups`

  (選擇性) 包含以逗號分隔的系統群組清單的字串。您授權這些系統群組使用 Greengrass CLI 與核心軟體互動。 AWS IoT Greengrass 您可以指定群組名稱或群組 ID。例

如，group1,1002,group3授權三個系統群組 (group11002、和group3) 使用 Greengrass CLI。

如果您沒有指定任何要授權的群組，您可以使用 Greengrass CLI 做為根使用者 (sudo) 或執行 Core 軟體的系統使用者。 AWS IoT Greengrass

`configurationUpdate.AuthorizedWindowsGroups`

(選擇性) 包含以逗號分隔的系統群組清單的字串。您授權這些系統群組使用 Greengrass CLI 與核心軟體互動。 AWS IoT Greengrass 您可以指定群組名稱或群組 ID。例如，group1,1002,group3授權三個系統群組 (group11002、和group3) 使用 Greengrass CLI。

如果您未指定任何要授權的群組，您可以使用 Greengrass CLI 做為系統管理員或執行 Core 軟體的系統使用者。 AWS IoT Greengrass

2. 執行下列命令，以在裝置上部署 Greengrass CLI 元件：

```
$ aws greengrassv2 create-deployment --cli-input-json file://path/
to/deployment.json
```

在安裝期間，元件會在裝置上的*/greengrass/v2*/bin資料夾greengrass-cli中新增符號連結，然後從此路徑執行 Greengrass CLI。要在沒有絕對路徑的情況下運行 Greengrass CLI，請將您的*/greengrass/v2*/bin文件夾添加到 PATH 變量中。若要驗證 Greengrass CLI 安裝，請執行下列命令：

Linux or Unix

```
/greengrass/v2/bin/greengrass-cli help
```

Windows

```
C:\greengrass\v2\bin\greengrass-cli help
```

您應該會看到下列輸出：

```
Usage: greengrass-cli [-hV] [--ggcRootPath=<ggcRootPath>] [COMMAND]
Greengrass command line interface
```

```
      --ggcRootPath=<ggcRootPath>
                  The AWS IoT Greengrass V2 root directory.
  -h, --help      Show this help message and exit.
  -V, --version   Print version information and exit.
Commands:
  help                Show help information for a command.
  component           Retrieve component information and stop or restart
                        components.
  deployment          Create local deployments and retrieve deployment status.
  logs                Analyze Greengrass logs.
  get-debug-password  Generate a password for use with the HTTP debug view
                        component.
```

如果找`greengrass-cli`不到，則部署可能無法安裝 Greengrass CLI。如需更多詳細資訊，請參閱[疑難排 AWS IoT Greengrass V2](#)。

Greengrass CLI 命令

Greengrass CLI 提供了一個命令行界面，可以在本地與您的AWS IoT Greengrass核心設備進行交互。綠色 CLI 命令使用以下格式。

```
$ greengrass-cli <command> <subcommand> [arguments]
```

默認情況下，文件夾中的`greengrass-cli`可執行文件與文件*/greengrass/v2*/bin/夾中運行的AWS IoT Greengrass核心軟件的版本進行*/greengrass/v2*交互。如果您呼叫未放置在此位置的可執行檔，或者您想要與位於不同位置的 AWS IoT Greengrass Core 軟體互動，則必須使用下列其中一種方法，明確指定您要與之互動的 AWS IoT Greengrass Core 軟體的根路徑：

- 將 GGC_ROOT_PATH 環境變數設為 */greengrass/v2*。
- 如下列範例所示，將`--ggcRootPath` */greengrass/v2*引數新增至您的命令。

  ```
  greengrass-cli --ggcRootPath /greengrass/v2 <command> <subcommand> [arguments]
  ```

您可以將下列引數與任何命令搭配使用：

- 用`--help`於取得有關特定 Greengrass CLI 命令的資訊。
- 用`--version`於取得有關 Greengrass 版本的資訊。

本節介紹了 Greengrass CLI 命令，並提供了這些命令的範例。每個指令的摘要都會顯示其引數及其用法。可選參數顯示在方括號中。

可用命令

- 元件
- 部署
- 日誌
- get-debug-password

元件

使用指component令與核心裝置上的本機元件互動。

子命令

- details
- list
- 重新開始
- stop

details

擷取一個元件的版本、狀態和組態。

概要

```
greengrass-cli component details --name <component-name>
```

Arguments (引數)

`--name`，`-n`。 元件名稱。

輸出

下列範例顯示執行此命令時產生的輸出。

```
$ sudo greengrass-cli component details --name MyComponent
```

```
Component Name: MyComponent
Version: 1.0.0
State: RUNNING
Configuration: null
```

list

擷取裝置上安裝之每個元件的名稱、版本、狀態和組態。

概要

```
greengrass-cli component list
```

Arguments (引數)

無

輸出

下列範例顯示執行此命令時產生的輸出。

```
$ sudo greengrass-cli component list

Components currently running in Greengrass:
Component Name: FleetStatusService
Version: 0.0.0
State: RUNNING
Configuration: {"periodicUpdateIntervalSec":86400.0}
Component Name: UpdateSystemPolicyService
Version: 0.0.0
State: RUNNING
Configuration: null
Component Name: aws.greengrass.Nucleus
Version: 2.0.0
State: FINISHED
Configuration: {"awsRegion":"region","runWithDefault":
{"posixUser":"ggc_user:ggc_group"},"telemetry":{}}
Component Name: DeploymentService
Version: 0.0.0
State: RUNNING
Configuration: null
Component Name: TelemetryAgent
```

```
Version: 0.0.0
State: RUNNING
Configuration: null
Component Name: aws.greengrass.Cli
Version: 2.0.0
State: RUNNING
Configuration: {"AuthorizedPosixGroups":"ggc_user"}
```

重新開始

重新啟動元件。

概要

```
greengrass-cli component restart --names <component-name>,...
```

Arguments (引數)

`--names`，`-n`。 元件名稱。至少需要一個元件名稱。您可以指定其他元件名稱，並以逗號分隔每個名稱。

輸出

無

stop

停止執行元件。

概要

```
greengrass-cli component stop --names <component-name>,...
```

Arguments (引數)

`--names`，`-n`。 元件名稱。至少需要一個元件名稱。如果需要，您可以指定其他元件名稱，並以逗號分隔每個名稱。

輸出

無

部署

使用指deployment令與核心裝置上的本機元件互動。

若要監視本機部署的進度，請使用status子指令。您無法使用主控台監視本機部署的進度。

子命令

- 建立
- 取消
- 列出
- status

建立

使用指定的元件配方、成品和執行階段引數來建立或更新本機部署。

概要

```
greengrass-cli deployment create
    --recipeDir path/to/component/recipe
    [--artifactDir path/to/artifact/folder ]
    [--update-config {component-configuration}]
    [--groupId <thing-group>]
    [--merge "<component-name>=<component-version>"]...
    [--runWith "<component-name>:posixUser=<user-name>[:<group-name>]"]...
    [--systemLimits "{component-system-resource-limits}]"]...
    [--remove <component-name>,...]
    [--failure-handling-policy <policy name[ROLLBACK, DO_NOTHING]>]
```

Arguments (引數)

- `--recipeDir`，`-r`。 包含元件 recipe 檔案之資料夾的完整路徑。
- `--artifactDir`，`-a`。 包含您要包含在部署中的成品檔案之資料夾的完整路徑。人工因素資料夾必須包含下列目錄結構：

  ```
  /path/to/artifact/folder/<component-name>/<component-version>/<artifacts>
  ```

- `--update-config`，`-c`。 部署的組態引數，以 JSON 字串或 JSON 檔案形式提供。JSON 字串應採用下列格式：

```
{ \
  "componentName": { \
    "MERGE": {"config-key": "config-value"}, \
    "RESET": ["path/to/reset/"] \
  } \
}
```

MERGE和區RESET分大小寫，且必須為大寫。

- --groupId，-g。 部署的目標物件群組。
- --merge，-m。 您要新增或更新的目標元件的名稱和版本。您必須以格式提供元件資訊*<component>*=*<version>*。為每個要指定的其他元件使用不同的引數。如果需要，請使用--runWith引數來posixUser提供執行元件的posixGroup、和windowsUser資訊。
- --runWith。 執行一般或 Lambda 元件的posixUserposixGroup、和windowsUser資訊。您必須以格式提供此資訊*<component>*:{posixUser|windowsUser}=*<user>*[:*<=posixGroup>*]。例如，您可以指定**HelloWorld:posixUser=ggc_user:ggc_group**或**HelloWorld:windowsUser=ggc_user**。為每個要指定的其他選項使用單獨的引數。

  如需詳細資訊，請參閱 設定執行元件的使用者。
- --systemLimits。 系統資源限制適用於核心裝置上的一般和非容器化 Lambda 元件的程序。您可以設定每個元件的處理序可以使用的 CPU 和 RAM 使用量上限。指定序列化的 JSON 物件或 JSON 檔案的檔案路徑。JSON 物件必須具有下列格式。

```
{  \
  "componentName": { \
    "cpus": cpuTimeLimit, \
    "memory": memoryLimitInKb \
  } \
}
```

您可以為每個元件設定下列系統資源限制：

- cpus— 此元件的處理序可以在核心裝置上使用的 CPU 時間上限。核心裝置的 CPU 總時間等於裝置的 CPU 核心數。例如，在具有 4 個 CPU 核心的核心裝置上，您可以將此值設定為2將此元件的處理序限制為每個 CPU 核心的 50% 使用率。在具有 1 個 CPU 核心的裝置上，您可以將此值設定0.25為將此元件的處理序限制為 CPU 使用率 25%。如果您將此值設定為大於 CPU 核心數目的數字， AWS IoT Greengrass 核心軟體就不會限制元件的 CPU 使用率。

- memory— 此元件的處理序可在核心裝置上使用的最大 RAM 量 (以 KB 為單位)。

  如需詳細資訊，請參閱 設定元件的系統資源限制。

  此功能適用於 Linux 核心裝置上的 Greengrass 核心元件和 Greengrass CLI 的 v2.4.0 及更新版本。 AWS IoT Greengrass 目前在 Windows 核心裝置上不支援此功能。

- --remove。 您要從本機部署移除的目標元件名稱。若要移除已從雲端部署合併的元件，您必須以下列格式提供目標物群組的群組 ID：

  Greengrass nucleus v2.4.0 and later

  ```
  --remove <component-name> --groupId <group-name>
  ```

  Earlier than v2.4.0

  ```
  --remove <component-name> --groupId thinggroup/<group-name>
  ```

- --failure-handling-policy。 定義部署失敗時採取的動作。您可以指定兩個動作：

  - ROLLBACK –
  - DO_NOTHING –

  此功能適用於 v2.11.0 及更高版本的。Greengrass 核

輸出

下列範例顯示執行此命令時產生的輸出。

```
$ sudo greengrass-cli deployment create \
    --merge MyApp1=1.0.0 \
    --merge MyApp2=1.0.0 --runWith MyApp2:posixUser=ggc_user \
    --remove MyApp3 \
    --recipeDir recipes/ \
    --artifactDir artifacts/

Local deployment has been submitted! Deployment Id: 44d89f46-1a29-4044-
ad89-5151213dfcbc
```

取消

取消指定的部署。

概要

```
greengrass-cli deployment cancel
    -i <deployment-id>
```

引數

-i。 要取消之部署的唯一識別碼。部署 ID 會在create命令的輸出中傳回。

輸出

- 無

列出

擷取最近 10 個本機部署的狀態。

概要

```
greengrass-cli deployment list
```

Arguments (引數)

無

輸出

下列範例顯示執行此命令時產生的輸出。根據部署狀態而定，輸出會顯示下列其中一個狀態值：IN_PROGRESSSUCCEEDED、或FAILED。

```
$ sudo greengrass-cli deployment list

44d89f46-1a29-4044-ad89-5151213dfcbc: SUCCEEDED
Created on: 6/27/23 11:05 AM
```

status

擷取特定部署的狀態。

概要

```
greengrass-cli deployment status -i <deployment-id>
```

Arguments (引數)

`-i`。 部署的識別碼。

輸出

下列範例顯示執行此命令時產生的輸出。根據部署狀態而定，輸出會顯示下列其中一個狀態值：`IN_PROGRESS`、`SUCCEEDED`、或`FAILED`。

```
$ sudo greengrass-cli deployment status -i 44d89f46-1a29-4044-ad89-5151213dfcbc

44d89f46-1a29-4044-ad89-5151213dfcbc: FAILED
Created on: 6/27/23 11:05 AM
Detailed Status: <Detailed deployment status>
Deployment Error Stack: List of error codes
Deployment Error Types: List of error types
Failure Cause: Cause
```

日誌

使用命`logs`令分析核心裝置上的 Greengrass 記錄檔。

子命令

- get
- 列表關鍵字
- list-log-files

get

收集、篩選和視覺化 Greengrass 記錄檔。此命令僅支援 JSON 格式的記錄檔。您可以在核心組態中指定記錄格式。

概要

```
greengrass-cli logs get
    [--log-dir path/to/a/log/folder]
    [--log-file path/to/a/log/file]
    [--follow true | false ]
```

```
    [--filter <filter> ]
    [--time-window <start-time>,<end-time> ]
    [--verbose ]
    [--no-color ]
    [--before <value> ]
    [--after <value> ]
    [--syslog ]
    [--max-long-queue-size <value> ]
```

Arguments (引數)

- `--log-dir`，`-ld`。 檢查記錄檔的目錄路徑，例如**/greengrass/v2/logs**。請勿與配合使用`--syslog`。為每個要指定的其他目錄使用單獨的引數。您必須至少使用其中一個`--log-dir`或`--log-file`。您也可以在單一命令中使用這兩個引數。
- `--log-file`，`-lf`。 您要使用之記錄目錄的路徑。為每個要指定的其他目錄使用單獨的引數。您必須至少使用其中一個`--log-dir`或`--log-file`。您也可以在單一命令中使用這兩個引數。
- `--follow`，`-fol`。 在發生記錄檔更新時顯示。CLI Greengrass 繼續執行並從指定的記錄中讀取。如果您指定時間範圍，則 Greengrass CLI 會在所有時間視窗結束後停止監視記錄檔。
- `--filter`，`-f`。 用作篩選器的關鍵字、規則運算式或索引鍵值組。將此值提供為字串、規則運算式或索引鍵值配對。為要指定的每個其他篩選器使用不同的引數。

  評估時，在單一引數中指定的多個篩選器會以 OR 運算子分隔，而在其他引數中指定的篩選器會與 AND 運算子結合。例如，如果您的命令包括`--filter "installed" --filter "name=alpha,name=beta"`，則 Greengrass CLI 將過濾並顯示包含關鍵字`installed`和具有`name`值或的鍵的日誌消息。`alpha beta`
- `--time-window`，`-t`。 顯示記錄資訊的時間範圍。您可以同時使用確切的時間戳記和相對偏移量。您必須以格式提供此資訊*<begin-time>*,*<end-time>*。如果您未指定開始時間或結束時間，則該選項的值會預設為目前的系統日期和時間。為每個要指定的其他時間範圍使用不同的引數。

  格 Greengrass CLI 支持以下格式的時間戳：

  - `yyyy-MM-DD`，例如，`2020-06-30`. 使用此格式時，時間預設為 00:00:00。

    `yyyyMMDD`，例如，`20200630`. 使用此格式時，時間預設為 00:00:00。

    `HH:mm:ss`，例如，`15:30:45`. 當您使用此格式時，日期預設為目前的系統日期。

    `HH:mm:ssSSS`，例如，`15:30:45`. 當您使用此格式時，日期會預設目前的系統日期。

    `YYYY-MM-DD'T'HH:mm:ss'Z'`，例如，`2020-06-30T15:30:45Z`.

YYYY-MM-DD'T'HH:mm:ss，例如，2020-06-30T15:30:45.

yyyy-MM-dd'T'HH:mm:ss.SSS，例如，2020-06-30T15:30:45.250.

相對位移指定從目前系統時間偏移的時間週期。Greengrass CLI 支持相對偏移以下格式：。+|-[<*value*>h|hr|hours][*value*m|min|minutes][*value*]s|sec|seconds

例如，下列引數用於指定目前時間之前 1 小時到 2 小時 15 分鐘之間的時間範圍--time-window -2h15min,-1hr。

- --verbose。 顯示記錄訊息中的所有欄位。請勿與配合使用--syslog。
- --no-color，-nc。 移除顏色編碼。日誌消息的默認顏色編碼使用紅色粗體文本。僅支持類 Unix 的終端，因為它使用 ANSI 轉義序列。
- --before，-b。 相符記錄項目前要顯示的行數。預設值為 0。
- --after，-a。 符合記錄項目之後要顯示的行數。預設值為 0。
- --syslog。 使用 RFC3164 定義的系統記錄通訊協定處理所有記錄檔。請勿與--log-dir和一起使用--verbose。系統記錄通訊協定使用下列格式：。"<$Priority>$Timestamp $Host $Logger ($Class): $Message"如果您未指定記錄檔，則 Greengrass CLI 會從下列位置讀取記錄訊息：/var/log/messages/var/log/syslog、或. /var/log/system.log

  AWS IoT Greengrass目前在 Windows 核心裝置上不支援此功能。
- --max-log-queue-size，-m。 要配置給記憶體的記錄項目數目上限。使用此選項可最佳化記憶體使用量。預設值為 100。

輸出

下列範例顯示執行此命令時產生的輸出。

```
$ sudo greengrass-cli logs get --verbose \
    --log-file /greengrass/v2/logs/greengrass.log \
    --filter deployment,serviceName=DeploymentService \
    --filter level=INFO \
    --time-window 2020-12-08T01:11:17,2020-12-08T01:11:22

2020-12-08T01:11:17.615Z [INFO] (pool-2-thread-14)
 com.aws.greengrass.deployment.DeploymentService: Current deployment finished.
 {DeploymentId=44d89f46-1a29-4044-ad89-5151213dfcbc, serviceName=DeploymentService,
 currentState=RUNNING}
2020-12-08T01:11:17.675Z [INFO] (pool-2-thread-14)
 com.aws.greengrass.deployment.IotJobsHelper: Updating status of persisted
```

```
 deployment. {Status=SUCCEEDED, StatusDetails={detailed-deployment-
status=SUCCESSFUL}, ThingName=MyThing, JobId=22d89f46-1a29-4044-ad89-5151213dfcbc
```

列表關鍵字

顯示可用來篩選記錄檔的建議關鍵字。

概要

```
greengrass-cli logs list-keywords [arguments]
```

Arguments (引數)

無

輸出

下列範例顯示執行此命令時產生的輸出。

```
$ sudo greengrass-cli logs list-keywords

Here is a list of suggested keywords for Greengrass log:
level=$str
thread=$str
loggerName=$str
eventType=$str
serviceName=$str
error=$str
```

```
$ sudo greengrass-cli logs list-keywords --syslog

Here is a list of suggested keywords for syslog:
priority=$int
host=$str
logger=$str
class=$str
```

list-log-files

顯示位於指定目錄中的記錄檔。

概要

```
greengrass-cli logs list-log-files [arguments]
```

Arguments (引數)

`--log-dir`，`-ld`。 檢查記錄檔案的目錄路徑。

輸出

下列範例顯示執行此命令時產生的輸出。

```
$ sudo greengrass-cli logs list-log-files -ld /greengrass/v2/logs/

/greengrass/v2/logs/aws.greengrass.Nucleus.log
/greengrass/v2/logs/main.log
/greengrass/v2/logs/greengrass.log
Total 3 files found.
```

get-debug-password

使用`get-debug-password`命令打印隨機生成的密碼本地調試控制台組件(`aws.greengrass.LocalDebugConsole`。密碼會在產生 8 小時後過期。

概要

```
greengrass-cli get-debug-password
```

Arguments (引數)

無

輸出

以下範例顯示執行此命令時產生的輸出。

```
$ sudo greengrass-cli get-debug-password

Username: debug
Password: bEDp3MOHdj8ou2w5de_sCBI2XAaguy3a8XxREXAMPLE
Password expires at: 2021-04-01T17:01:43.921999931-07:00
```

```
The local debug console is configured to use TLS security. The certificate is self-
signed so you will need to bypass your web browser's security warnings to open the
 console.
Before you bypass the security warning, verify that the certificate fingerprint
 matches the following fingerprints.
SHA-256: 15 0B 2C E2 54 8B 22 DE 08 46 54 8A B1 2B 25 DE FB 02 7D 01 4E 4A 56 67 96
 DA A6 CC B1 D2 C4 1B
SHA-1: BC 3E 16 04 D3 80 70 DA E0 47 25 F9 90 FA D6 02 80 3E B5 C1
```

使用 AWS IoT Greengrass 測試框架

Greengrass 測試框架（GTF）是支持從客戶角度 end-to-end自動化的構建模塊的集合。GTF 使用黃瓜作為功能驅動程序。 AWS IoT Greengrass 使用相同的建置區塊來限定各種裝置上的軟體變更。有關更多信息，請參閱 Greengrass itHub 上的綠色測試框架。

GTF 是使用黃瓜，用於運行自動化測試的工具來實現，以鼓勵組件的行為驅動開發（BDD）。在黃瓜，這個系統的功能在一個名為特殊類型的文件概述`feature`。每個功能被稱為場景是可以轉換成自動化測試規範的人類可讀格式描述。每個場景被概述為一系列使用稱為小黃瓜特定領域的語言定義該系統的相互作用和結果的步驟。一個小黃瓜步驟被鏈接到使用一種稱為步驟定義的方法，其硬線規範到測試流程的編程代碼。GTF 中的步驟定義是使用 Java 實現的。

主題

- 運作方式
- 變更記錄
- 綠色測試框架配置選項
- 教程：使用 Greengrass 測 end-to-end 試框架和 Greengrass 開發工具包運行測試
- 教學課程：使用置信度測試套件中的可信度測試

運作方式

AWS IoT Greengrass 將 GTF 作為由多個 Java 模塊組成的獨立 JAR 進行分發。要使用 GTF end-to-end 測試組件，您必須在 Java 項目中實現測試。在 Java 項目中添加測試可站立的 JAR 作為依賴項使您可以使用 GTF 的現有功能，並通過編寫自己的自定義測試用例來擴展它。要運行自定義測試用例，您可以構建 Java 項目並使用中描述的配置選項運行目標 JAR 綠色測試框架配置選項。

GTF 獨立式 JAR

使用雲端作為 Maven 存儲庫來託管不同版本的 GTF 獨立 JAR。有關 GTF 版本的完整列表，請參閱 GTF 版本。

GTF 獨立 JAR 包括以下模塊。它不僅限於這些模塊。您可以在專案中分別挑選並選擇每個依賴項，或者使用測試獨立 JAR 文件一次包含所有這些依賴項。

- `aws-greengrass-testing-resources`：此模塊提供抽象，用於在測試過程中管理 AWS 資源的生命週期。您可以使用它來使用`ResourceSpec`抽象來定義您的自定義 AWS 資源，以便 GTF 可以為您創建和刪除這些資源。
- `aws-greengrass-testing-platform`：此模塊為測試生命週期期間被測設備提供平台級抽象。它包含用於與獨立於平台的操作系統進行交互的 API，並且可用於模擬在設備外殼中運行的命令。
- `aws-greengrass-testing-components`：此模組包含用於測試 Greengrass 核心功能 (例如部署、IPC 和其他功能) 的範例元件。
- `aws-greengrass-testing-features`：此模塊由可重複使用的通用步驟及其定義組成，這些步驟用於在 Greengrass 環境中進行測試。

主題

變更記錄

下表描述了每個 GTF 版本中的變更。如需詳細資訊，請參閱中的 GTF 發行版本頁面 GitHub。

版本	變更
1.2.0	新功能 • 新增與網路相關的步驟，以在測試期間設定 MQTT 和網際網路連線。 • 添加系統度量步驟以監視設備 RAM 和 CPU 使用情況。

版本	變更
	錯誤修復和改進 • 重試 CLI 本地部署步驟，直到它成功為止。 • 測試優雅地停止 Greengrass 核，而不是殺死它。 • 新增 GTF 輪詢 AWS IoT 認證端點的改進，直到可擷取物件和角色別名的認證為止。 • 修復丟失的工件和配方目錄。此版本還修復了缺少的組件版本。 • 修復了如果碼頭映像不存在，則在 docker 映像清理期間 GTF 失敗的問題。 • 添加當前關鍵字作為組件的版本。
1.1.0	新功能 • 添加了使用配置安裝自定義組件的功能。這需要自定義組件的配方。 • 添加使用自訂規劃更新本機部署的功能。 錯誤修復和改進 • 修復了日誌上下文 GTF 版本不一致的問題。
1.0.0	初始版本。

綠色測試框架配置選項

GTF 組態選項

Greengrass 測試架構 (GTF) 可讓您在啟動期間設定某些參數 end-to-end 測試過程以協調測試流程。您可以將這些組態選項指定為 GTF 獨立 JAR 的 CLI 引數。

GTF 1.1.0 及更新版本提供下列組態選項。

- `additional-plugins-`（可選）其他黃瓜插件
- `aws-region`— 針對特定的區域端點AWS服務。默認為什麼AWSSDK 會發現。
- `credentials-path`— 可選AWS設定檔認證路徑。預設為在主機環境中探索到的認證。
- `credentials-path-rotation`— 可選的旋轉持續時間AWS認證。預設值為 15 分鐘或`PT15M`。
- `csr-path`— 將用來產生裝置憑證的 CSR 路徑。
- `device-mode`— 被測的目標設備。預設為本機裝置。

- env-stage— 以格林格拉斯的部署環境為目標。預設為生產。
- existing-device-cert-arn— 您要用作 Greengrass 裝置憑證的現有憑證的 arn。
- feature-path— 包含其他功能檔案的檔案或目錄。預設值是不使用其他功能檔案。
- gg-cli-version-覆寫格林格拉斯 CLI 的版本。預設為在中找到的值ggc.version。
- gg-component-bucket— 容納 Greengrass 元件的現有亞馬遜 S3 儲存貯體的名稱。
- gg-component-overrides-綠色組件覆蓋的列表。
- gg-persist-測試運行後要持續存在的測試元素列表。默認行為是什麼都不堅持。接受的值為：aws.resources,installed.software，以及generated.files。
- gg-runtime— 影響測試與測試資源互動方式的值清單。這些值取代gg.persist參數。如果默認值為空，則假定所有測試資源都由測試用例管理，包括安裝的 Greengrass 運行時。接受的值為：aws.resources,installed.software，以及generated.files。
- ggc-archive— 路徑歸檔的 Greengrass 核組件.
- ggc-install-root— 安裝綠核組件的目錄. 默認為測試 .temp.path 和測試運行文件夾。
- ggc-log-level— 為測試運行設置 Greengrass 核日誌級別。預設值為「資訊」。
- ggc-tes-rolename— IAM 的角色AWS IoT Greengrass核心將假設訪問AWS服務。如果具有給定名稱的角色不存在，則將創建一個並默認訪問策略。
- ggc-trusted-plugins— 需要添加到 Greengrass 的受信任插件的路徑（在主機上）的逗號分隔列表。要提供 DUT 本身的路徑，請在路徑前加上「dut：」
- ggc-user-name— 使用者：群組 Greengrass 核的 PosixUser 值。預設為目前登入的使用者名稱。
- ggc-version— 覆寫正在執行的 Greengrass 核元件的版本。默認為在 ggc.archive 中找到的值。
- log-level— 測試運行的日誌級別。默認為「信息」。
- parallel-config— 設置批次索引和批次數作為 JSON 字符串。批處理索引的默認值為 0，批次數為 1。
- proxy-url— 配置所有測試以通過此 URL 路由流量。
- tags— 僅執行功能標籤。可以與 '&' 相交
- test-id-prefix— 應用於所有測試特定資源的通用前綴，包括AWS資源名稱和標籤。預設值為「gg」前置詞。
- test-log-path— 將包含整個測試運行結果的目錄。默認為「測試結果」。
- test-results-json— 用於確定產生的黃瓜 JSON 報告是否產生寫入磁碟的旗標。預設為 true。
- test-results-log— 用於確定控制台輸出是否生成寫入磁盤的標誌。預設為 false。

- `test-results-xml`— 用於判斷產生的 JUnit XML 報告是否已產生寫入磁碟的旗標。預設為 true。
- `test-temp-path`— 產生本機測試成品的目錄。默認為以 gg-test 前綴的隨機臨時目錄。
- `timeout-multiplier`— 提供給所有測試超時的乘數。預設值為 1.0。

教程：使用 Greengrass 測 end-to-end 試框架和 Greengrass 開發工具包運行測試

AWS IoT Greengrass測試框架（GTF）和 Greengrass 開發工具包（GDK）為開發人員提供了運行測試的方法。 end-to-end 您可以完成本教程以初始化 GDK 項目的組件，使用 end-to-end 測試模塊初始化 GDK 項目，並構建自定義測試用例。在構建自定義測試用例後，您可以運行測試。

在此教學課程中，您將執行下列操作：

1. 使用元件初始化 GDK 專案。
2. 使用 end-to-end 測試模塊初始化 GDK 項目。
3. 建立自訂測試案例。
4. 將標籤添加到新的測試用例。
5. 建置測試 JAR。
6. 執行 測試。

主題

必要條件

為了完成本教學，您需要以下項目：

- GDK 版本 1.3.0 或更新版本
- Java
- Maven
- Git

第 1 步：使用組件初始化 GDK 項目

- 使用 GDK 專案初始化空白資料夾。通過運行以下命令下載在 Python 中實現的HelloWorld組件。

  ```
  gdk component init -t HelloWorld -l python -n HelloWorld
  ```

  這個命令會在目前的目錄HelloWorld中建立一個名為的新目錄。

第 2 步：使用 end-to-end測試模塊初始化 GDK 項目

- GDK 使您能夠下載由功能和步驟實現組成的測試模塊模板。運行以下命令來打開HelloWorld目錄，並使用測試模塊初始化現有的 GDK 項目。

  ```
  cd HelloWorld
  gdk test-e2e init
  ```

  此命令會在目錄gg-e2e-tests中建立名為的新HelloWorld目錄。這個測試目錄是一個 Maven 項目，它對 Greengrass 測試獨立 JAR 的依賴關係。

步驟 3：建立自訂測試案例

編寫自定義測試用例大致包括兩個步驟：創建具有測試場景的功能文件並實現步驟定義。如需建立自訂測試案例的範例，請參閱示例：構建自定義測試用例。使用以下步驟來構建自定義測試用例：

1. 使用測試場景創建功能文件

   功能通常描述正在測試的軟體的特定功能。在黃瓜，每個功能被指定為一個標題，詳細描述，並稱為場景特定情況的一個或多個例子單獨的功能文件。每個案例都包含一個標題、詳細描述，以及定義互動和預期結果的一系列步驟。案例是使用「給定」、「何時」和「then」關鍵字以結構化格式撰寫。
2. 實作步驟定義

步驟定義鏈接簡單的語言小黃瓜步驟的編程代碼。當黃瓜標識在場景中的小黃瓜步驟，它會尋找一個匹配的步驟定義運行。

步驟 4：將標籤添加到新的測試用例

- 您可以將標籤分配給功能和方案以組織測試過程。您可以使用標籤來分類案例的子集，也可以有條件地選取要執行的掛接。功能和案例可以有多個以空格分隔的標籤。

  在這個例子中，我們使用的HelloWorld組件。

  在特徵檔案中，在標籤@HelloWorld旁加入名為的新@Sample標籤。

  ```
  @Sample @HelloWorld
  Scenario: As a developer, I can create a component and deploy it on my device
  ....
  ```

步驟 5：建立測試 JAR

1. 建置元件。您必須在構建測試模塊之前構建組件。

   ```
   gdk component build
   ```

2. 使用以下命令構建測試模塊。此命令將在greengrass-build文件夾中構建測試 JAR。

   ```
   gdk test-e2e build
   ```

步驟 6：執行測試

當您運行自定義測試用例時，GTF 會自動化測試的生命周期以及管理在測試過程中創建的資源。它首先將被測設備（DUT）佈建為一個AWS IoT東西，並在其上安裝 Greengrass 核心軟件。然後，它將創建一個HelloWorld使用該路徑中指定的配方命名的新組件。然後，HelloWorld元件會透過 Greengrass 物件部署到核心裝置上。然後，如果部署成功，它將被驗證。如果部署成功，部署狀態將變更為 3 分鐘COMPLETED內。

1. 轉到項目目錄中的gdk-config.json文件以使用HelloWorld標籤定位測試。使用以下命令更新test-e2e密鑰。

```
"test-e2e":{
  "gtf_options" : {
       "tags":"HelloWorld"
   }
}
```

2. 在執行測試之前，您必須提供主機裝置的AWS認證。GTF 在測試過程中使用這些憑據來管理AWS資源。確保您提供的角色具有自動化測試中包含的必要操作的權限。

   執行下列命令以提供AWS認證。

   - Linux or Unix

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

     Windows Command Prompt (CMD)

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

     PowerShell

     ```
     $env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
     $env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
     ```

3. 使用以下命令運行測試。

   ```
   gdk test-e2e run
   ```

   此命令會下載`greengrass-build`資料夾中 Greengrass 核的最新版本，並使用它執行測試。此命令也只針對具有`HelloWorld`標籤的案例，並產生這些案例的報表。您將看到在此測試期間創建的AWS資源在測試結束時被丟棄。

示例：構建自定義測試用例

Example

GDK 項目中下載的測試模塊由示例功能和步驟實現文件組成。

在下面的例子中，我們創建一個功能文件來測試 Greengrass 軟件的物件部署功能。我們使用透過 Greeng AWS 雲端 rass 執行元件部署的案例來部分測試此功能的功能。這是一系列步驟，可協助我們瞭解此使用案例的互動和預期結果。

1. 建立特徵檔

   導覽至目前目錄中的gg-e2e-tests/src/main/resources/greengrass/features資料夾。您可以找到如下範component.feature例所示的範例。

   在此功能檔案中，您可以測試 Greengrass 軟體的物件部署功能。您可以使用透過 Greengrass 雲端執行元件部署的案例來部分測試此功能的功能。案例是一系列步驟，可協助您瞭解此使用案例的互動和預期結果。

   ```
   Feature: Testing features of Greengrassv2 component

   Background:
       Given my device is registered as a Thing
       And my device is running Greengrass

   @Sample
   Scenario: As a developer, I can create a component and deploy it on my device
       When I create a Greengrass deployment with components
           HelloWorld | /path/to/recipe/file
       And I deploy the Greengrass deployment configuration
       Then the Greengrass deployment is COMPLETED on the device after 180 seconds
       And I call my custom step
   ```

   GTF 包含下列所有步驟的步驟定義，名為:And I call my custom step的步驟除外。

2. 實作步驟定義

   GTF 獨立 JAR 包含除了一個步驟以外的所有步驟的步驟定義：And I call my custom step。您可以在測試模塊中實現此步驟。

   導航到測試文件的源代碼。您可以使用下列指令，使用步驟定義來連結自訂步驟。

   ```
   @And("I call my custom step")
   public void customStep() {
       System.out.println("My custom step was called ");
   }
   ```

教學課程：使用置信度測試套件中的可信度測試

AWS IoT Greengrass測試框架（GTF）和 Greengrass 開發工具包（GDK）為開發人員提供了運行測試的方法。 end-to-end 您可以完成本教程來初始化 GDK 項目與組件，初始化 GDK 項目與 end-to-end 測試模塊，並使用從可信度測試套件的置信度測試。在構建自定義測試用例後，您可以運行測試。

置信度測試是 Greengrass 提供的通用測試，用於驗證基本組件行為。這些測試可以進行修改或擴展，以滿足更具體的組件需求。

在本教程中，我們將使用一個 HelloWorld 組件。如果您正在使用其他元件，請用您的 HelloWorld 元件取代該元件。

在此教學課程中，您將執行下列操作：

1. 使用元件初始化 GDK 專案。
2. 使用 end-to-end 測試模塊初始化 GDK 項目。
3. 使用可信度測試套件中的測試。
4. 將標籤添加到新的測試用例。
5. 建置測試 JAR。
6. 執行 測試。

主題

必要條件

為了完成本教學，您需要以下項目：

- GDK 版本 1.6.0 或更新版本
- Java
- Maven
- Git

第 1 步：使用組件初始化 GDK 項目

- 使用 GDK 專案初始化空白資料夾。通過運行以下命令下載在 Python 中實現的`HelloWorld`組件。

  ```
  gdk component init -t HelloWorld -l python -n HelloWorld
  ```

  這個命令會在目前的目錄`HelloWorld`中建立一個名為的新目錄。

第 2 步：使用 end-to-end測試模塊初始化 GDK 項目

- GDK 使您能夠下載由功能和步驟實現組成的測試模塊模板。運行以下命令來打開`HelloWorld`目錄，並使用測試模塊初始化現有的 GDK 項目。

  ```
  cd HelloWorld
  gdk test-e2e init
  ```

  這個命令會在目錄中建立一`gg-e2e-tests`個名為的新`HelloWorld`目錄。這個測試目錄是一個 Maven 項目，它對 Greengrass 測試獨立 JAR 的依賴關係。

步驟 3：使用可信度測試套件中的測試

編寫一個置信度測試用例包括使用所提供的功能文件，並在需要時修改方案。如需使用置信度測試的範例，請參閱示例：構建自定義測試用例。請使用下列步驟來使用可靠度測試：

- 使用提供的功能文件。

  導覽至目前目錄中的`gg-e2e-tests/src/main/resources/greengrass/features`資料夾。開啟範例`confidenceTest.feature`檔案以使用置信度測試。

步驟 4：將標籤添加到新的測試用例

- 您可以將標籤分配給功能和方案以組織測試過程。您可以使用標籤來分類案例的子集，也可以有條件地選取要執行的勾點。功能和案例可以有多個以空格分隔的標籤。

  在這個例子中，我們使用的HelloWorld組件。

  每個案例都以標記@ConfidenceTest。如果您只想運行測試套件的子集，請更改或添加標籤。每個測試案例都在每個可信度測試的頂部進行描述。該方案是一系列的步驟，有助於理解每個測試用例的交互和預期結果。您可以通過添加自己的步驟或修改現有步驟來擴展這些測試。

```
@ConfidenceTest
Scenario: As a Developer, I can deploy GDK_COMPONENT_NAME to my device and see it
 is working as expected
....
```

步驟 5：建立測試 JAR

1. 建置元件。您必須在構建測試模塊之前構建組件。

```
gdk component build
```

2. 使用以下命令構建測試模塊。此命令將在greengrass-build文件夾中構建測試 JAR。

```
gdk test-e2e build
```

步驟 6：執行測試

當您運行置信度測試時，GTF 會自動化測試的生命周期以及管理在測試過程中創建的資源。它首先將被測設備（DUT）佈建為一個AWS IoT東西，並在其上安裝 Greengrass 核心軟件。然後，它將創建一個HelloWorld使用該路徑中指定的配方命名的新組件。然後，HelloWorld元件會透過 Greengrass 物件部署到核心裝置上。然後，如果部署成功，它將被驗證。如果部署成功，部署狀態將變更為 3 分鐘COMPLETED內。

1. 轉到項目目錄中的gdk-config.json文件，以使用在步驟 4 中指定的ConfidenceTest標籤或任何標籤 yo8u 定位測試。使用以下命令更新test-e2e密鑰。

```
  "test-e2e":{
```

```
    "gtf_options" : {
         "tags":"ConfidenceTest"
     }
  }
```

2. 在執行測試之前，您必須提供主機裝置的AWS認證。GTF 在測試過程中使用這些憑據來管理AWS資源。確保您提供的角色具有自動化測試中包含的必要操作的權限。

   執行下列命令以提供AWS認證。

   - Linux or Unix

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

     Windows Command Prompt (CMD)

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

     PowerShell

     ```
     $env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
     $env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
     ```

3. 使用以下命令運行測試。

   ```
   gdk test-e2e run
   ```

   此命令會下載`greengrass-build`資料夾中 Greengrass 核的最新版本，並使用它執行測試。此命令也只針對具有`ConfidenceTest`標籤的案例，並產生這些案例的報表。您將看到在此測試期間創建的AWS資源在測試結束時被丟棄。

範例：使用可信度測試

Example

在 GDK 項目下載的測試模塊由提供的功能文件。

在下列範例中，我們使用功能檔案來測試 Greengrass 軟體的物件部署功能。我們使用透過 Greeng AWS 雲端 rass 執行元件部署的案例來部分測試此功能的功能。這是一系列步驟，可協助我們瞭解此使用案例的互動和預期結果。

- 使用提供的功能文件。

  導覽至目前目錄中的gg-e2e-tests/src/main/resources/greengrass/features資料夾。您可以找到如下範confidenceTest.feature例所示的範例。

```
Feature: Confidence Test Suite

Background:
    Given my device is registered as a Thing
    And my device is running Greengrass

@ConfidenceTest
Scenario: As a Developer, I can deploy GDK_COMPONENT_NAME to my device and see it
 is working as expected
    When I create a Greengrass deployment with components
      | GDK_COMPONENT_NAME | GDK_COMPONENT_RECIPE_FILE |
      | aws.greengrass.Cli | LATEST                    |
    And I deploy the Greengrass deployment configuration
    Then the Greengrass deployment is COMPLETED on the device after 180 seconds
    # Update component state accordingly. Possible states: {RUNNING, FINISHED,
 BROKEN, STOPPING}
    And I verify the GDK_COMPONENT_NAME component is RUNNING using the greengrass-
cli
```

  每個測試案例都在每個可信度測試的頂部進行描述。該方案是一系列的步驟，有助於理解每個測試用例的交互和預期結果。您可以通過添加自己的步驟或修改現有步驟來擴展這些測試。每個案例都包含可協助您進行這些調整的註解。

開發AWS IoT Greengrass元件

您可以在 Greengrass 核心裝置上開發和測試元件。因此，您可以建立並重複使用您的AWS IoT Greengrass軟體，而無需與AWS 雲端. 完成元件的某個版本後，您可以將其上傳到AWS IoT Greengrass雲端，以便您和您的團隊可以將元件部署到叢集中的其他裝置。如需如何部署元件的詳細資訊，請參閱將AWS IoT Greengrass元件部署到裝置。

每個組件都由配方和文物組成。

- 食譜

  每個組件都包含一個 recipe 文件，該文件定義了其元數據。方案也會指定元件的組態參數、元件相依性、生命週期和平台相容性。元件生命週期會定義安裝、執行和關閉元件的命令。如需詳細資訊，請參閱 AWS IoT Greengrass元件配方參考。

  您可以定義 JSON 或 YAML 格式的配方。

- 文物

  組件可以有任意數量的加工品，這是組件二進製文件。成品可以包括指令碼、已編譯的程式碼、靜態資源，以及元件使用的任何其他檔案。元件也可以從元件相依性取用人工因素。

AWS IoT Greengrass提供預先建置的元件，您可以在應用程式中使用這些元件並部署到您的裝置。例如，您可以使用串流管理員元件將資料上傳到各種AWS服務，或者使用 CloudWatch 指標元件將自訂指標發佈到 Amazon CloudWatch。如需詳細資訊，請參閱 AWS-提供的組件。

AWS IoT Greengrass策劃 Greengrass 組件的索引，稱為 Greengrass 軟件目錄。此目錄追蹤 Greengrass 社群所開發的 Greengrass 元件。您可以從此目錄下載、修改和部署元件，以建立 Greengrass 應用程式。如需詳細資訊，請參閱 社群元件。

AWS IoT Greengrass核心軟體會以系統使用者和群組的身分執行元件 `ggc_user`和`ggc_group`，例如您在核心裝置上設定的和。這表示元件具有該系統使用者的權限。如果您使用的是沒有主目錄的系統使用者，則元件將無法使用 run 命令或使用主目錄的程式碼。這意味著您不能使用命令`pip install some-library --user`來安裝例如 Python 軟件包。如果您依照入門教學課程設定核心裝置，則您的系統使用者沒有主目錄。如需如何設定執行元件的使用者和群組的詳細資訊，請參閱設定執行元件的使用者。

> **Note**
>
> AWS IoT Greengrass使用語義版本的組件。語義版本遵循一個主要的。 未成年人。 修補程式編號系統。例如，版本`1.0.0`代表元件的第一個主要發行版本。如需詳細資訊，請參閱語意版本規格。

主題

- 元件生命週
- 元件類型
- 建立 AWS IoT Greengrass 元件

- 使用本機部署測試AWS IoT Greengrass元件
- 發佈元件以部署至核心裝置
- 與AWS服務互動
- 運行碼頭容器
- AWS IoT Greengrass元件配方參考
- 元件環境變數參考

元件生命週

元件生命週期定義了 AWS IoT Greengrass Core 軟體用來安裝和執行元件的階段。每個階段都會定義指令碼，以及指定組件行為方式的其他資訊。例如，當您安裝元件時，AWS IoT Greengrass核心軟體會執行該元件的`Install`生命週期指令碼。核心裝置上的元件具有下列生命週期狀態：

- NEW— 元件的配方和成品會載入到核心裝置上，但未安裝元件。元件進入此狀態後，便會執行其安裝指令碼。
- INSTALLED— 元件安裝在核心裝置上。元件會在執行安裝指令碼之後進入此狀態。
- STARTING— 元件在核心裝置上啟動。元件會在執行啟動指令碼時進入此狀態。如果啟動成功，元件就會進入狀RUNNING態。
- RUNNING— 元件正在核心裝置上執行。當元件執行其執行指令碼，或從其啟動指令碼具有作用中的背景處理程序時，就會進入此狀態。
- FINISHED— 元件執行成功並完成其執行。
- STOPPING— 元件正在停止。元件會在執行關機指令碼時進入此狀態。
- ERRORED— 元件遇到錯誤。當元件進入此狀態時，它會執行其復原指令碼。然後，組件重新啟動以嘗試返回正常使用狀態。如果元件進入ERRORED狀態三次而未成功執行，則元件會變成BROKEN。
- BROKEN— 元件多次遇到錯誤，無法復原。您必須再次部署元件才能修正它。

元件類型

元件類型會指定AWS IoT Greengrass核心軟體執行元件的方式。元件可以有下列類型：

- 原子核 () `aws.greengrass.nucleus`

  Greengrass 核是提供核心軟件的最小功能的組件。AWS IoT Greengrass如需詳細資訊，請參閱 Greengrass 核。

- 插件（`aws.greengrass.plugin`）

  Greengrass 核在與核相同的 Java 虛擬機（JVM）中運行一個插件組件。當您變更核心裝置上外掛程式元件的版本時，核心會重新啟動。要安裝和運行插件組件，您必須配置 Greengrass 核作為系統服務運行。如需詳細資訊，請參閱 將 Greengrass 核配置為系統服務。

  由提供的幾個組件AWS是插件組件，這使它們能夠直接與 Greengrass 核接口。插件組件使用與 Greengrass 核相同的日誌文件。如需詳細資訊，請參閱 監控AWS IoT Greengrass日誌。
- 一般 (`aws.greengrass.generic`)

  Greengrass 核會執行一般元件的生命週期指令碼 (如果元件定義生命週期)。

  此類型是自訂元件的預設類型。
- Lambda (`aws.greengrass.lambda`)

  Greengrass 核運行使用 Lambda 啟動器組件的 Lambda 函數組件。

  當您從 Lambda 函數建立元件時，該組件具有此類型。如需詳細資訊，請參閱 執行AWS Lambda函數。

> **Note**
>
> 我們不建議您在方案中指定元件類型。 AWS IoT Greengrass在建立元件時為您設定類型。

建立 AWS IoT Greengrass 元件

您可以在本機開發電腦或 Greengrass 核心裝置上開發自訂 AWS IoT Greengrass 元件。 AWS IoT Greengrass 提供AWS IoT Greengrass 開發套件命令列介面 (GDK CLI)，協助您從預先定義的元件範本和社群元件建立、建置和發佈元件。您也可以執行內建 shell 命令來建立、建置和發佈元件。從下列選項中選擇以建立自訂 Greengrass 元件：

- 使用格 Greengrass 開發工具包 CLI

  使用 GDK CLI 在本機開發電腦上開發元件。GDK CLI 會建置元件原始程式碼，並將其封裝到方案和成品中，您可以將其作為私有元件發佈至 AWS IoT Greengrass 服務。您可以將 GDK CLI 設定為在發佈元件時自動更新元件的版本和成品 URI，因此您不需要每次都更新配方。若要使用 GDK CLI 開發元件，您可以從 Greengr ass 軟體目錄中的範本或社群元件開始。如需詳細資訊，請參閱 AWS IoT Greengrass開發套件命令列介面。

- 執行內建殼層命令

  您可以執行內建的殼層命令，在本機開發電腦或 Greengrass 核心裝置上開發元件。您可以使用 shell 命令將組件源代碼複製或構建到成品中。每次建立元件的新版本時，都必須使用新的元件版本來建立或更新方案。當您將元件發佈至 AWS IoT Greengrass 服務時，必須更新方案中每個元件人工因素的 URI。

主題

建立元件 (GDK CLI)

請遵循本節中的指示，使用 GDK CLI 建立和建置元件。

若要開發一個 Greengrass 元件 (GDK CLI)

1. 如果您尚未安裝，請在開發電腦上安裝 GDK CLI。如需詳細資訊，請參閱 安裝或更新AWS IoT Greengrass開發套件命令列介面。
2. 切換至要在其中建立元件資料夾的資料夾。

   Linux or Unix

   ```
   mkdir ~/greengrassv2
   cd ~/greengrassv2
   ```

   Windows Command Prompt (CMD)

   ```
   mkdir %USERPROFILE%\greengrassv2
   cd %USERPROFILE%\greengrassv2
   ```

   PowerShell

   ```
   mkdir ~/greengrassv2
   cd ~/greengrassv2
   ```

3. 選擇要下載的元件範本或社群元件。GDK CLI 會下載範本或社群元件，因此您可以從功能範例開始。使用元件清單指令擷取可用樣板或社群元件的清單。

- 若要列示元件樣板，請執行下列命令。回應中的每一行都包含範本的名稱和程式設計語言。

```
gdk component list --template
```

- 若要列出社群元件，請執行下列命令。

```
gdk component list --repository
```

4. 建立並變更為 GDK CLI 下載範本或社群元件的元件資料夾。以元件*HelloWorld*的名稱或其他可協助您識別此元件資料夾的名稱取代。

   Linux or Unix

```
mkdir HelloWorld
cd HelloWorld
```

   Windows Command Prompt (CMD)

```
mkdir HelloWorld
cd HelloWorld
```

   PowerShell

```
mkdir HelloWorld
cd HelloWorld
```

5. 將範本或社群元件下載到目前的資料夾。使用組件初始化命令。

   - 若要從樣板建立元件資料夾，請執行下列指令。*HelloWorld*替換為模板的名稱，並用編程語言的名稱替換 *python*。

```
gdk component init --template HelloWorld --language python
```

   - 若要從社群元件建立元件資料夾，請執行下列指令。*ComponentName*以社群元件的名稱取代。

```
gdk component init --repository ComponentName
```

> **Note**
>
> 如果您使用 GDK CLI v1.0.0，則必須在空白資料夾中執行此命令。GDK CLI 會將範本或社群元件下載到目前的資料夾。
> 如果您使用 GDK CLI v1.1.0 或更新版本，您可以指定`--name`引數來指定 GDK CLI 下載範本或社群元件的資料夾。如果您使用此引數，請指定不存在的資料夾。GDK CLI 會為您建立資料夾。如果您未指定此引數，GDK CLI 會使用目前的資料夾，該資料夾必須為空白。

6. GDK CLI 會從名為的 GDK CLI 組態檔讀取`gdk-config.json`，以建置和發佈元件。此組態檔案存在於元件資料夾的根目錄中。上一步會為您建立此檔案。在此步驟中，您會更新`gdk-config.json`元件的相關資訊。請執行下列操作：

   a. 在文字編輯器中開啟 `gdk-config.json`。

   b. (選擇性) 變更元件的名稱。元件名稱是`component`物件中的關鍵字。

   c. 變更元件的作者。

   d. (選擇性) 變更元件的版本。請指定下列其中一項：

      - `NEXT_PATCH`— 當您選擇此選項時，GDK CLI 會在您發佈元件時設定版本。GDK CLI 會查詢 AWS IoT Greengrass 服務，以識別元件的最新發佈版本。然後，它會將版本設定為該版本之後的下一個修補程式版本。如果您之前尚未發佈元件，GDK CLI 會使用版本`1.0.0`。

        如果您選擇此選項，則無法使用 Greengrass CLI 在本機部署元件，並將元件測試到執行 Core 軟體的本機開發電腦。 AWS IoT Greengrass 若要啟用本機部署，您必須改為指定語意版本。

      - 語義版本，例如**`1.0.0`**. 語義版本使用一個主要的。 未成年人。 補丁編號系統。如需詳細資訊，請參閱語意版本規格。

        如果您在要部署和測試元件的 Greengrass 核心裝置上開發元件，請選擇此選項。您必須使用特定版本建立元件，才能使用 Greengr ass CLI 建立本機部署。

   e. (選擇性) 變更元件的組建組態。組建組態會定義 GDK CLI 如何將元件的來源建置成品。從下列選項中選擇`build_system`：

      - `zip`— 將元件的資料夾封裝至 ZIP 檔案，以定義為元件的唯一人工因素。針對下列類型的元件選擇此選項：

- 使用解譯式程式設計語言的元件，例如 Python 或 JavaScript.
- 封裝程式碼以外檔案的元件，例如機器學習模型或其他資源。

GDK CLI 會將元件的資料夾壓縮成與元件資料夾名稱相同的 zip 檔案。例如，如果元件資料夾的名稱是`HelloWorld`，GDK CLI 會建立名`HelloWorld.zip`為的 zip 檔案。

> **Note**
>
> 如果您在 Windows 裝置上使用 GDK CLI 1.0.0 版，元件資料夾和壓縮檔案名稱必須只包含小寫字母。

當 GDK CLI 將元件的資料夾壓縮至 zip 檔案時，會略過下列檔案：

- `gdk-config.json` 檔案
- 配方文件（`recipe.json`或`recipe.yaml`）
- 建置資料夾，例如 `greengrass-build`

- `maven`— 執行指`mvn clean package`令，將元件的來源建置為成品。針對使用 Maven 的元件 (例如 Java 元件) 選擇此選項。

  在視窗裝置上，此功能適用於 GDK CLI 1.1.0 版及更新版本。
- `gradle`— 執行指`gradle build`令，將元件的來源建置為成品。對於使用搖籃的組件，請選擇此選項。此功能適用於 GDK CLI 1.1.0 版及更新版本。

  `gradle`構建系統支持 Kotlin DSL 作為構建文件。此功能適用於 GDK CLI 1.2.0 版及更新版本。
- `gradlew`— 執行指`gradlew`令，將元件的來源建置為成品。針對使用 Gradle 包裝函式的元件，請選擇此選項。

  此功能適用於 GDK CLI 1.2.0 版及更新版本。
- `custom`— 執行自訂命令，將元件的來源建置到配方和成品中。
  在`custom_build_command`參數中指定自訂指令。

f. 如果您指定`custom`為`build_system`，請將新增`custom_build_command`至`build`物件。在中`custom_build_command`，指定單一字串或字串清單，其中每個字串都是指令中的一個字。例如，若要執行 C++ 元件的自訂建置命令，您可以指定**`["cmake", "--build", "build", "--config", "Release"]`**。

g. 如果您使用 GDK CLI v1.1.0 或更新版本，則可以指定`--bucket`引數來指定 GDK CLI 上傳元件成品的 S3 儲存貯體。如果您未指定此引數，GDK CLI 會上傳至名稱為的 S3 儲存貯體*bucket*-*region*-*accountId*，其中儲存#體和##是您在中指定的值`gdk-config.json`，而 *accountId* 就是您 AWS 帳戶 的 ID。GDK CLI 會建立值區 (如果值區不存在)。

變更元件的發佈規劃。請執行下列操作：

i. 指定用於託管元件成品的 S3 儲存貯體名稱。

ii. 指定 GDK CLI 發佈元件的 AWS 區域 位置。

完成此步驟後，`gdk-config.json`檔案看起來可能類似於下列範例。

```
{
  "component": {
    "com.example.PythonHelloWorld": {
      "author": "Amazon",
      "version": "NEXT_PATCH",
      "build": {
        "build_system" : "zip"
      },
      "publish": {
        "bucket": "greengrass-component-artifacts",
        "region": "us-west-2"
      }
    }
  },
  "gdk_version": "1.0.0"
}
```

7. 更新名為`recipe.yaml`或的元件方案檔案`recipe.json`。請執行下列操作：

a. 如果您下載了使用組`zip`建系統的範本或社群元件，請檢查 zip 成品名稱是否與元件資料夾的名稱相符。GDK CLI 會將元件資料夾壓縮成與元件資料夾名稱相同的 zip 檔案。方案會在元件人工因素清單以及使用 zip 加工品中檔案的生命週期指令碼中包含 zip 人工因素名稱。更新`Artifacts`和定`Lifecycle`義，使 zip 檔案名稱與元件資料夾的名稱相符。下列部分方案範例會強調顯示`Artifacts`和`Lifecycle`定義中的 zip 檔案名稱。

JSON

```
{
  ...
  "Manifests": [
    {
      "Platform": {
        "os": "all"
      },
      "Artifacts": [
        {
          "URI": "s3://{COMPONENT_NAME}/{COMPONENT_VERSION}/HelloWorld.zip",
          "Unarchive": "ZIP"
        }
      ],
      "Lifecycle": {
        "run": "python3 -u {artifacts:decompressedPath}/HelloWorld/main.py
 {configuration:/Message}"
      }
    }
  ]
}
```

YAML

```
---
...
Manifests:
  - Platform:
      os: all
    Artifacts:
      - URI: "s3://BUCKET_NAME/COMPONENT_NAME/
COMPONENT_VERSION/HelloWorld.zip"
        Unarchive: ZIP
    Lifecycle:
      run: "python3 -u {artifacts:decompressedPath}/HelloWorld/main.py
 {configuration:/Message}"
```

b. (選擇性) 更新元件說明、預設組態、人工因素、生命週期指令碼和平台支援。如需詳細資訊，請參閱 AWS IoT Greengrass元件配方參考。

完成此步驟後，recipe 檔案看起來可能與下列範例類似。

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "{COMPONENT_NAME}",
  "ComponentVersion": "{COMPONENT_VERSION}",
  "ComponentDescription": "This is a simple Hello World component written in
 Python.",
  "ComponentPublisher": "{COMPONENT_AUTHOR}",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "Message": "World"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "all"
      },
      "Artifacts": [
        {
          "URI": "s3://{COMPONENT_NAME}/{COMPONENT_VERSION}/HelloWorld.zip",
          "Unarchive": "ZIP"
        }
      ],
      "Lifecycle": {
        "run": "python3 -u {artifacts:decompressedPath}/HelloWorld/main.py
 {configuration:/Message}"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: "2020-01-25"
ComponentName: "{COMPONENT_NAME}"
ComponentVersion: "{COMPONENT_VERSION}"
```

```
ComponentDescription: "This is a simple Hello World component written in
 Python."
ComponentPublisher: "{COMPONENT_AUTHOR}"
ComponentConfiguration:
  DefaultConfiguration:
    Message: "World"
Manifests:
  - Platform:
      os: all
    Artifacts:
      - URI: "s3://BUCKET_NAME/COMPONENT_NAME/COMPONENT_VERSION/HelloWorld.zip"
        Unarchive: ZIP
    Lifecycle:
      run: "python3 -u {artifacts:decompressedPath}/HelloWorld/main.py
 {configuration:/Message}"
```

8. 開發和構建 Greengrass 組件。元件建置指令會在元件資料夾的`greengrass-build`資料夾中產生配方和成品。執行下列命令。

```
gdk component build
```

當您準備好測試元件時，請使用 GDK CLI 將其發佈到 AWS IoT Greengrass 服務。然後，您可以將該組件部署到 Greengrass 核心設備。如需詳細資訊，請參閱 發佈元件以部署至核心裝置。

建立元件 (薄殼指令)

請遵循本節中的指示，建立包含多個元件之原始程式碼和人工因素的方案和人工因素資料夾。

若要開發 Greengrass 元件 (殼層指令)

1. 使用配方和成品的子資料夾，為您的元件建立資料夾。在 Greengrass 核心裝置上執行下列命令，以建立這些資料夾並變更為元件資料夾。將 *~/greengrassv2 #% ######%\ Greengrassv2 ############*料夾路徑。

   Linux or Unix

```
mkdir -p ~/greengrassv2/{recipes,artifacts}
cd ~/greengrassv2
```

Windows Command Prompt (CMD)

```
mkdir %USERPROFILE%\greengrassv2\\recipes, %USERPROFILE%\greengrassv2\\artifacts
cd %USERPROFILE%\greengrassv2
```

PowerShell

```
mkdir ~/greengrassv2/recipes, ~/greengrassv2/artifacts
cd ~/greengrassv2
```

2. 使用文字編輯器建立 recipe 檔案，以定義元件的中繼資料、參數、相依性、生命週期和平台功能。在 recipe 檔案名稱中包含元件版本，以便識別哪個方案反映哪個元件版本。您可以為您的食譜選擇 YAML 或 JSON 格式。

   例如，在 Linux 系統上，您可以執行下列命令來使用 GNU nano 建立檔案。

   JSON

   ```
   nano recipes/com.example.HelloWorld-1.0.0.json
   ```

   YAML

   ```
   nano recipes/com.example.HelloWorld-1.0.0.yaml
   ```

   > **Note**
   >
   > AWS IoT Greengrass 使用語義版本的組件。語義版本遵循一個主要的。 未成年人。 修補程式編號系統。例如，版本`1.0.0`代表元件的第一個主要發行版本。如需詳細資訊，請參閱語意版本規格。

3. 定義元件的配方。如需詳細資訊，請參閱 AWS IoT Greengrass元件配方參考。

   您的食譜看起來可能類似於下列 Hello World 範例配方。

   JSON

   ```
   {
     "RecipeFormatVersion": "2020-01-25",
   ```

```
  "ComponentName": "com.example.HelloWorld",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "My first AWS IoT Greengrass component.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "Message": "world"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "run": "python3 -u {artifacts:path}/hello_world.py {configuration:/
Message}"
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "run": "py -3 -u {artifacts:path}/hello_world.py {configuration:/
Message}"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.HelloWorld
ComponentVersion: '1.0.0'
ComponentDescription: My first AWS IoT Greengrass component.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    Message: world
Manifests:
```

```
  - Platform:
      os: linux
    Lifecycle:
      run: |
        python3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
  - Platform:
      os: windows
    Lifecycle:
      run: |
        py -3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
```

這個配方會執行 Hello World Python 指令碼，看起來可能類似於下列範例指令碼。

```
import sys

message = "Hello, %s!" % sys.argv[1]

# Print the message to stdout, which Greengrass saves in a log file.
print(message)
```

4. 為要開發的元件版本建立資料夾。建議您為每個元件版本的成品使用不同的資料夾，以便識別每個元件版本的成品。執行下列命令。

   Linux or Unix

   ```
   mkdir -p artifacts/com.example.HelloWorld/1.0.0
   ```

   Windows Command Prompt (CMD)

   ```
   mkdir artifacts/com.example.HelloWorld/1.0.0
   ```

   PowerShell

   ```
   mkdir artifacts/com.example.HelloWorld/1.0.0
   ```

   Important

   人工因素資料夾路徑必須使用下列格式。包括您在方案中指定的元件名稱和版本。

```
artifacts/componentName/componentVersion/
```

5. 在您在上一個步驟中建立的資料夾中建立元件的成品。成品可以包括軟體、影像，以及您的元件使用的任何其他二進位檔案。

   當您的組件準備就緒時，請測試您的組件。

使用本機部署測試AWS IoT Greengrass元件

如果您在核心裝置上開發 Greengrass 元件，您可以建立本機部署以進行安裝和測試。請遵循本節中的步驟來建立本機部署。

如果您在不同的電腦 (例如本機開發電腦) 上開發元件，則無法建立本機部署。而是將元件發佈至AWS IoT Greengrass服務，以便您可以將其部署到 Greengrass 核心裝置以進行測試。如需詳細資訊，請參閱 發佈元件以部署至核心裝置 及 將AWS IoT Greengrass元件部署到裝置。

若要在 Greengrass 核心裝置上測試元件

1. 核心裝置會記錄元件更新等事件。您可以檢視此記錄檔，以探索元件的錯誤並進行疑難排解，例如無效的方案。此記錄檔也會顯示元件列印為標準輸出 (stdout) 的訊息。我們建議您在核心裝置上開啟額外的終端機工作階段，以即時觀察新的記錄訊息。開啟新的終端機工作階段，例如透過 SSH，然後執行下列命令以檢視記錄。以AWS IoT Greengrass根資料夾的路徑取*/greengrass/v2*代。

   Linux or Unix

   ```
   sudo tail -f /greengrass/v2/logs/greengrass.log
   ```

   PowerShell

   ```
   gc C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
   ```

   您也可以檢視元件的記錄檔。

Linux or Unix

```
sudo tail -f /greengrass/v2/logs/com.example.HelloWorld.log
```

PowerShell

```
gc C:\greengrass\v2\logs\com.example.HelloWorld.log -Tail 10 -Wait
```

2. 在原始終端機工作階段中，執行下列命令以使用元件更新核心裝置。以AWS IoT Greengrass根資料夾*/greengrass/v2*的路徑取代，並將 *~/greengrassv2* 取代為本機開發資料夾的路徑。

   Linux or Unix

   ```
   sudo /greengrass/v2/bin/greengrass-cli deployment create \
     --recipeDir ~/greengrassv2/recipes \
     --artifactDir ~/greengrassv2/artifacts \
     --merge "com.example.HelloWorld=1.0.0"
   ```

   Windows Command Prompt (CMD)

   ```
   C:\greengrass\v2\bin\greengrass-cli deployment create ^
     --recipeDir %USERPROFILE%\greengrassv2\recipes ^
     --artifactDir %USERPROFILE%\greengrassv2\artifacts ^
     --merge "com.example.HelloWorld=1.0.0"
   ```

   PowerShell

   ```
   C:\greengrass\v2\bin\greengrass-cli deployment create `
     --recipeDir ~/greengrassv2/recipes `
     --artifactDir ~/greengrassv2/artifacts `
     --merge "com.example.HelloWorld=1.0.0"
   ```

   > **Note**
   >
   > 您也可以使用`greengrass-cli deployment create`指令來設定元件組態參數的值。如需詳細資訊，請參閱 建立。

3. 使用`greengrass-cli deployment status`命令來監視元件部署的進度。

Unix or Linux

```
sudo /greengrass/v2/bin/greengrass-cli deployment status \
  -i deployment-id
```

Windows Command Prompt (CMD)

```
C:\greengrass\v2\bin\greengrass-cli deployment status ^
  -i deployment-id
```

PowerShell

```
C:\greengrass\v2\bin\greengrass-cli deployment status `
  -i deployment-id
```

4. 測試您的組件，因為它在 Greengrass 核心設備上運行。當您完成此版本的元件時，您可以將其上傳至AWS IoT Greengrass服務。然後，您可以將元件部署到其他核心裝置。如需更多詳細資訊，請參閱 發佈元件以部署至核心裝置。

發佈元件以部署至核心裝置

建置或完成元件版本之後，您可以將其發佈至AWS IoT Greengrass服務。然後，您可以將其部署到 Greengrass 核心設備。

如果您使用 Greengrass 開發套件 CLI (GDK CLI) 來開發和建置元件，您可以使用 GDK CLI 將元件發行到. AWS 雲端 否則，請使用內建的 shell 命令和AWS CLI來發佈元件。

您也可以使AWS CloudFormation用從範本建立元件和其他AWS資源。如需詳細資訊，請參閱什麼是 AWS CloudFormation？ 以及AWS::GreengrassV2::ComponentVersion在《用AWS CloudFormation戶指南》中。

主題

- 發佈元件 (GDK CLI)
- 發佈元件 (薄殼指令)

發佈元件 (GDK CLI)

請遵循本節中的指示，使用 GDK CLI 發佈元件。GDK CLI 會將組建成品上傳到 S3 儲存貯體、更新方案中的成品 URI，並從配方建立元件。您可以指定要在 GDK CLI 組態檔中使用的 S3 儲存貯體和區域。

如果您使用 GDK CLI v1.1.0 或更新版本，則可以指定`--bucket`引數來指定 GDK CLI 上傳元件成品的 S3 儲存貯體。如果您未指定此引數，GDK CLI 會上傳至名稱為的 S3 儲存貯體*bucket*-*region*-*accountId*，其中儲存#體和##是您在中指定的值`gdk-config.json`，而*accountId* 就是您AWS 帳戶的 ID。GDK CLI 會建立值區 (如果值區不存在)。

> **Important**
>
> 核心裝置角色預設不允許存取 S3 儲存貯體。如果這是您第一次使用此 S3 儲存貯體，則必須向角色新增許可，以允許核心裝置從此 S3 儲存貯體擷取元件成品。如需詳細資訊，請參閱 允許存取 S3 儲存貯體的存取。

若要發佈 Greengrass 元件 (GDK CLI)，請執行下列步驟：

1. 在指令提示字元或端子中開啟元件資料夾。
2. 如果你還沒有，建立 Greengrass 組件。元件建置指令會在元件資料夾的`greengrass-build`資料夾中產生配方和成品。執行下列命令。

   ```
   gdk component build
   ```

3. 將元件發佈到AWS 雲端. 元件發佈命令會將元件的成品上傳到 Amazon S3，並使用每個成品的 URI 更新元件的配方。然後，它會在AWS IoT Greengrass服務中建立元件。

   > **Note**
   >
   > AWS IoT Greengrass當您建立元件時，會計算每個成品的摘要。這表示您無法在建立元件之後修改 S3 儲存貯體中的成品檔案。如果這樣做，包含此元件的部署將會失敗，因為檔案摘要不相符。如果修改人工因素檔案，則必須建立元件的新版本。

   如果您在 GDK CLI 組態檔中指定`NEXT_PATCH`元件版本，GDK CLI 會使用服務中尚未存在的下一個修補程式版本。AWS IoT Greengrass

執行下列命令。

```
gdk component publish
```

輸出會告訴您 GDK CLI 建立的元件版本。

發佈元件之後，您可以將元件部署到核心裝置。如需詳細資訊，請參閱 將AWS IoT Greengrass元件部署到裝置。

發佈元件 (薄殼指令)

請使用下列程序，使用 shell 指令和 AWS Command Line Interface (AWS CLI) 來發佈元件。發佈元件時，請執行下列操作：

1. 將元件成品發佈到 S3 儲存貯體。
2. 將每個成品的 Amazon S3 URI 新增至元件配方。
3. 從元件方案AWS IoT Greengrass中建立元件版本。

> **Note**
>
> 您上傳的每個元件版本都必須是唯一的。請務必上傳正確的元件版本，因為上傳後就無法編輯它。

您可以按照以下步驟從開發計算機或 Greengrass 核心設備發布組件。

發佈元件 (薄殼指令) 的步驟

1. 如果元件使用AWS IoT Greengrass服務中存在的版本，則您必須變更元件的版本。在文字編輯器中開啟配方，增加版本，然後儲存檔案。選擇反映您對元件所做變更的新版本。

   > **Note**
   >
   > AWS IoT Greengrass使用語義版本的組件。語義版本遵循一個主要的。 未成年人。 修補程式編號系統。例如，版本`1.0.0`代表元件的第一個主要發行版本。如需詳細資訊，請參閱語意版本規格。

2. 如果您的元件有人工因素，請執行下列動作：

   a. 將元件的成品發佈到AWS 帳戶.

   > **Tip**
   >
   > 建議您在 S3 儲存貯體中的成品路徑中包含元件名稱和版本。此命名配置可協助您維護舊版元件所使用的人工因素，以便您可以繼續支援先前的元件版本。

   執行下列命令，將成品檔案發佈到 S3 儲存貯體。*##########################/######HelloWorld/1.0.0/## .py ##*件文件的路徑。

   ```
   aws s3 cp artifacts/com.example.HelloWorld/1.0.0/artifact.py s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.HelloWorld/1.0.0/artifact.py
   ```

   > **Important**
   >
   > 核心裝置角色預設不允許存取 S3 儲存貯體。如果這是您第一次使用此 S3 儲存貯體，則必須向角色新增許可，以允許核心裝置從此 S3 儲存貯體擷取元件成品。如需詳細資訊，請參閱 允許存取 S3 儲存貯體的存取。

   b. 添加一個名為的列表`Artifacts`到組件配方，如果它不存在。此`Artifacts`清單會顯示在每個資訊清單中，其中定義元件在其支援的每個平台上的需求 (或元件對所有平台的預設需求)。

   c. 將每個人工因素新增至人工因素清單，或更新現有成品的 URI。Amazon S3 URI 由儲存貯體名稱和儲存貯體中成品物件的路徑組成。您人工品的 Amazon S3 URI 看起來應該類似於下列範例。

   ```
   s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.HelloWorld/1.0.0/artifact.py
   ```

   完成這些步驟後，您的方案應該會有如下所示的`Artifacts`清單。

   JSON

   ```
   {
     ...
     "Manifests": [
       {
   ```

```
      "Lifecycle": {
        ...
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/MyGreengrassComponent/1.0.0/
artifact.py",
          "Unarchive": "NONE"
        }
      ]
    }
  ]
}
```

> **Note**
>
> 您可以新增 ZIP 人工因素的"Unarchive": "ZIP"選項，以便將AWS IoT Greengrass核心軟體設定為在元件部署時解壓縮成品。

YAML

```
...
Manifests:
  - Lifecycle:
      ...
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/MyGreengrassComponent/1.0.0/
artifact.py
        Unarchive: NONE
```

> **Note**
>
> 您可以使用Unarchive: ZIP此選項來設定AWS IoT Greengrass核心軟體，以便在部署元件時解壓縮 ZIP 加工品。如需如何在組件中使用 ZIP 加工品的詳細資訊，請參閱人工因素：解壓縮路徑方法變數。

如需配方的詳細資訊，請參閱AWS IoT Greengrass元件配方參考。

3. 使用AWS IoT Greengrass主控台從 recipe 檔案建立元件。

   執行下列指令，從 recipe 檔案建立元件。此指令會建立元件，並將其發佈為AWS 帳戶. AWS IoT Greengrass 用 recipe 文件的路*#####/#/####*。

   ```
   aws greengrassv2 create-component-version --inline-recipe fileb://path/to/
   recipeFile
   ```

   arn從響應複製以在下一個步驟中檢查組件的狀態。

   > **Note**
   >
   > AWS IoT Greengrass當您建立元件時，會計算每個成品的摘要。這表示您無法在建立元件之後修改 S3 儲存貯體中的成品檔案。如果這樣做，包含此元件的部署將會失敗，因為檔案摘要不相符。如果修改人工因素檔案，則必須建立元件的新版本。

4. AWS IoT Greengrass服務中的每個元件都有一個狀態。執行下列命令，以確認您在此程序中發佈的元件版本狀態。例如替換*# HelloWorld*和 *1.0.0* 與要查詢的組件版本。arn使用上一個步驟中的 ARN 取代。

   ```
   aws greengrassv2 describe-component --arn "arn:aws:greengrass:region:account-
   id:components:com.example.HelloWorld:versions:1.0.0"
   ```

   此作業會傳回包含元件中繼資料的回應。中繼資料包含一個status物件，其中包含元件狀態和任何錯誤 (如果適用)。

   當元件狀態為時DEPLOYABLE，您可以將元件部署到裝置。如需更多詳細資訊，請參閱 將AWS IoT Greengrass元件部署到裝置。

與AWS服務互動

Greengrass 核心裝置使用 X.509 憑證來連線到AWS IoT Core使用 TLS 相互驗證通訊協定。這些憑證可讓裝置在沒有AWS認證的AWS IoT情況下進行互動，這些憑證通常包含存取金鑰 ID 和秘密存取金鑰。其他AWS服務需要AWS憑證而非 X.509 憑證，才能在服務端點呼叫 API 作業。 AWS IoT Core具有認證提供者，可讓裝置使用其 X.509 憑證來驗證AWS要求。AWS IoT憑證提供者會使用 X.509 憑證驗證裝置，並以臨時、有限權限的安全性權杖形式發出AWS認證。設備可以使用此令牌來簽署和驗證任何AWS請求。這樣就不需要在 Greengrass 核心設備上存儲AWS憑據。如需詳細資訊，請參閱AWS IoT Core開發人員指南中的授權直接呼叫AWS服務。

若要從 AWS IoT Greengrass 擷取認證，核心裝置會使用指向 IAM AWS IoT 角色的角色別名。此 IAM 角色稱為權杖交換角色。您可以在安裝 AWS IoT Greengrass Core 軟體時建立角色別名和權杖交換角色。若要指定核心裝置使用的角色別名，請設定的`iotRoleAlias`參數Greengrass 核。

AWS IoT憑據提供程序代表您承擔令牌交換角色，以向核心設備提供AWS憑據。您可以將適當的 IAM 政策附加到此角色，以允許核心裝置存取您的AWS資源，例如 S3 儲存貯體中的元件成品。如需如何設定 Token 交換角色的詳細資訊，請參閱授權核心設備與AWS服務。

Greengrass 核心裝置會將AWS認證儲存在記憶體中，依預設，認證會在一小時後過期。如果AWS IoT Greengrass核心軟體重新啟動，它必須再次擷取認證。您可以使用此UpdateRoleAlias作業來設定認證有效的持續時間。

AWS IoT Greengrass提供一個公共組件，即令牌交換服務組件，您可以將其定義為自定義組件中的依賴項以與AWS服務進行交互。Token 交換服務會為您的元件提供環境變數`AWS_CONTAINER_CREDENTIALS_FULL_URI`，該變數會定義提供認AWS證的本機伺服器的 URI。當您建立 AWS SDK 用戶端時，用戶端會檢查此環境變數，並連線至本機伺服器以擷取AWS認證，並使用它們簽署 API 要求。這可讓您使用 AWS SDK 和其他工具來呼叫元件中的AWS服務。如需詳細資訊，請參閱 代幣交換服務。

> **Important**
>
> 以這種方式取得AWS認證的 Support 已於 2016 年 7 月 13 日新增至 AWS SDK。您的元件必須使用在該日期或之後建立的 AWS SDK 版本。如需詳細資訊，請參閱 Amazon 彈性容器服務開發人員指南中的使用支援的 AWS SDK。

若要取得自訂元件中的AWS認證，請在元件方案中定義`aws.greengrass.TokenExchangeService`為相依性。下列範例方法定義了安裝 boto3 並執行 Python 指令碼的元件，該指令碼使用權杖交換服務的AWS登入資料列出 Amazon S3 儲存貯體。

> **Note**
>
> 若要執行此範例元件，您的裝置必須具有`s3:ListAllMyBuckets`權限。如需詳細資訊，請參閱 授權核心設備與AWS服務。

JSON

```
{
```

```
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.ListS3Buckets",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that uses the token exchange service to list
 S3 buckets.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.TokenExchangeService": {
      "VersionRequirement": "^2.0.0",
      "DependencyType": "HARD"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "pip3 install --user boto3",
        "run": "python3 -u {artifacts:path}/list_s3_buckets.py"
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "pip3 install --user boto3",
        "run": "py -3 -u {artifacts:path}/list_s3_buckets.py"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.ListS3Buckets
ComponentVersion: '1.0.0'
ComponentDescription: A component that uses the token exchange service to list S3
 buckets.
ComponentPublisher: Amazon
```

```
ComponentDependencies:
  aws.greengrass.TokenExchangeService:
    VersionRequirement: '^2.0.0'
    DependencyType: HARD
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install:
        pip3 install --user boto3
      run: |-
        python3 -u {artifacts:path}/list_s3_buckets.py
  - Platform:
      os: windows
    Lifecycle:
      install:
        pip3 install --user boto3
      run: |-
        py -3 -u {artifacts:path}/list_s3_buckets.py
```

此範例元件會執行下列列出 Amazon S3 儲存貯體的 Python 指令碼。`list_s3_buckets.py`

```
import boto3
import os

try:
    print("Creating boto3 S3 client...")
    s3 = boto3.client('s3')
    print("Successfully created boto3 S3 client")
except Exception as e:
    print("Failed to create boto3 s3 client. Error: " + str(e))
    exit(1)

try:
    print("Listing S3 buckets...")
    response = s3.list_buckets()
    for bucket in response['Buckets']:
        print(f'\t{bucket["Name"]}')
    print("Successfully listed S3 buckets")
except Exception as e:
    print("Failed to list S3 buckets. Error: " + str(e))
    exit(1)
```

運行碼頭容器

您可以將AWS IoT Greengrass元件設定為從儲存在下列位置的映像執行 Docker 容器：

- Amazon Elastic Container Registry (Amazon ECR) 中的公共和私有映像存儲庫
- 公共碼頭集線器存儲庫
- 公共碼頭信任的註冊表
- S3 儲存貯體

在您的自訂元件中，包含 Docker 映像 URI 作為成品，以擷取映像並在核心裝置上執行。對於 Amazon ECR 和 Docker Hub 映像，您可以使用 Doc ker 應用程式管理員元件下載映像並管理私有 Amazon ECR 儲存庫的登入資料。

主題

要求

要在組件中運行 Docker 容器，您需要以下內容：

- 一個 Greengrass 核心設備。如果您沒有帳戶，請參閱 教學課程：AWS IoT Greengrass V2 入門。
- 碼頭引擎 1.9.1 或更高版本安裝在 Greengrass 核心設備上。版本 20.10 是經過驗證可與AWS IoT Greengrass核心軟件配合使用的最新版本。在部署執行 Docker 容器的元件之前，您必須直接在核心裝置上安裝 Docker。

> **Tip**
>
> 您也可以將核心裝置設定為在元件安裝時安裝 Docker Engine。例如，下列安裝指令碼會在載入 Docker 映像檔之前先安裝 Docker 引擎。此安裝腳本適用於基於 Debian 的 Linux 發行

版，例如 Ubuntu。如果您將元件設定為使用此命令安裝 Docker Engine，您可能需要在生命週期指令碼`true`中`RequiresPrivilege`將設定為，才能執行安裝和命`docker`令。如需詳細資訊，請參閱 AWS IoT Greengrass元件配方參考。

```
apt-get install docker-ce docker-ce-cli containerd.io && docker load -i
 {artifacts:path}/hello-world.tar
```

- 執行 Docker 容器元件的系統使用者必須具有根或系統管理員權限，或者您必須將 Docker 設定為以非 root 使用者或非管理員使用者的身分執行。
  - 在 Linux 裝置上，您可以將使用者新增至`docker`群組，以便在不使用的情況下呼叫`docker`指令`sudo`。
  - 在 Windows 裝置上，您可以將使用者新增至`docker-users`群組，以便在沒有系統管理員權限的情況下呼叫`docker`命令。

  Linux or Unix

  若要將`ggc_user`您用來執行 Docker 容器元件的非 root 使用者新增至`docker`群組，請執行下列命令。

  ```
  sudo usermod -aG docker ggc_user
  ```

  如需詳細資訊，請參閱以非 root 使用者身分管理 Docker。

  Windows Command Prompt (CMD)

  若要將`ggc_user`或您用來執行 Docker 容器元件的使用者新增至`docker-users`群組，請以系統管理員身分執行下列命令。

  ```
  net localgroup docker-users ggc_user /add
  ```

  Windows PowerShell

  若要將`ggc_user`或您用來執行 Docker 容器元件的使用者新增至`docker-users`群組，請以系統管理員身分執行下列命令。

  ```
  Add-LocalGroupMember -Group docker-users -Member ggc_user
  ```

- 由裝載為 Docker 容器容器中的磁碟區的 Docker 容器元件存取的檔案。

- 如果您將 AWS IoT GreengrassCore 軟體設定為使用網路 Proxy，則必須將 Docker 設定為使用相同的 Proxy 伺服器。

除了這些需求之外，如果這些需求適用於您的環境，您還必須符合下列需求：

- 若要使用 Docker 撰寫來建立和啟動您的 Docker 容器，請在 Greengrass 核心裝置上安裝 Docker 撰寫，然後將您的碼頭撰寫檔案上傳到 S3 儲存貯體。您必須將撰寫檔案存放在 S3 儲存貯體中AWS帳戶與AWS 區域元件相同的儲存貯體中。若要取得在自訂元件中使用`docker-compose up`指令的範例，請參閱〈〉從 Amazon ECR 或碼頭集線器中的公共映像運行碼頭容器。
- 如果您在網路代理伺服器AWS IoT Greengrass後方執行，請將 Docker 常駐程式設定為使用代理伺服器。
- 如果您的 Docker 映像檔儲存在 Amazon ECR 或 Docker 集線器中，請在 Docker 容器元件中加入 Docker 元件管理員元件做為相依性。在部署元件之前，您必須在核心裝置上啟動 Docker 精靈。

  此外，請將影像 URI 納入為元件加工品。影像 URI 的格式必須`docker:`*`registry`*`/`*`image`*`[:`*`tag`*`|@`*`digest`*`]`如下列範例所示：

  - 私人 Amazon ECR 圖片：`docker:`*`account-id`*`.dkr.ecr.`*`region`*`.amazonaws.com/`*`repository`*`/`*`image`*`[:`*`tag`*`|@`*`digest`*`]`
  - 公共 Amazon ECR 圖片：`docker:public.ecr.aws/`*`repository`*`/`*`image`*`[:`*`tag`*`|@`*`digest`*`]`
  - 公共碼頭集線器映像：`docker:`*`name`*`[:`*`tag`*`|@`*`digest`*`]`

  有關從存儲在公共存儲庫中的映像運行 Docker 容器的更多信息，請參閱從 Amazon ECR 或碼頭集線器中的公共映像運行碼頭容器。
- 如果您的 Docker 映像檔儲存在 Amazon ECR 私有儲存庫中，則您必須將權杖交換服務元件作為 Docker 容器元件的相依性包含在 Docker 容器元件中。此外，Greengrass 裝置角色必須允許`ecr:GetAuthorizationToken`、和`ecr:GetDownloadUrlForLayer`動作`ecr:BatchGetImage`，如下列範例 IAM 政策所示。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "ecr:GetAuthorizationToken",
        "ecr:BatchGetImage",
        "ecr:GetDownloadUrlForLayer"
      ],
      "Resource": [
```

```
        "*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

如需從 Amazon ECR 私有儲存庫中儲存的映像執行 Docker 容器的相關資訊，請參閱。從 Amazon ECR 中的私有映像運行碼頭集裝箱

- 若要使用存放在 Amazon ECR 私有儲存庫中的 Docker 映像，私有存放庫必須與核心裝置位於AWS區域相同的位置。
- 如果您的 Docker 映像或撰寫檔案儲存在 S3 儲存貯體中，Greengrass 裝置角色必須允許許`s3:GetObject`可允許核心裝置將映像下載為元件成品，如下列範例 IAM 政策所示。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

如需從 Amazon S3 中存放的映像執行 Docker 容器的相關資訊，請參閱從 Amazon S3 中的映像運行碼頭集裝箱。

- 若要在 Docker 容器元件中使用處理序間通訊 (IPC)、AWS認證或串流管理員，您必須在執行 Docker 容器時指定其他選項。如需詳細資訊，請參閱下列內容：
  - 在 Docker 容器組件中使用進程間通信
  - 在碼頭容器組件中使用AWS憑據（Linux）
  - 在碼頭容器組件中使用流管理器（Linux）

從 Amazon ECR 或碼頭集線器中的公共映像運行碼頭容器

本節說明如何建立使用碼頭構成的自訂元件，從存放 Amazon ECR 和碼頭集線器的 Docker 映像執行碼頭容器。

使用碼頭構成運行碼頭容器

1. 建立碼頭撰寫檔案並將其上傳到 Amazon S3 儲存貯體。請確定 Greengrass 裝置角色允許允許裝置存取撰寫檔案的`s3:GetObject`權限。下列範例中顯示的範例撰寫檔案包括來自 Amazon ECR 的 Amazon CloudWatch 代理程式映像以及來自碼頭集線器的 MySQL 映像。

   ```
   version: "3"
   services:
     cloudwatchagent:
       image: "public.ecr.aws/cloudwatch-agent/cloudwatch-agent:latest"
     mysql:
       image: "mysql:8.0"
   ```

2. 在AWS IoT Greengrass核心裝置上建立自訂元件。下列範例中顯示的範例配方具有下列屬性：

   - Docker 應用程序管理器組件作為依賴關係。此元件可AWS IoT Greengrass讓您從公有的 Amazon ECR 和碼頭集線器儲存庫下載映像檔。
   - 在公用 Amazon ECR 儲存庫中指定碼頭映像的元件成品。
   - 在公用 Docker Hub 儲存庫中指定 Docker 映像檔的元件人工因素。
   - 一種元件人工因素，指定 Docker 撰寫檔案，其中包含您要執行的 Docker 映像的容器。
   - 一個生命週期運行腳本，使用 docker-compose 最多從指定的映像創建和啟動容器。

   JSON

   ```
   {
     "RecipeFormatVersion": "2020-01-25",
     "ComponentName": "com.example.MyDockerComposeComponent",
     "ComponentVersion": "1.0.0",
     "ComponentDescription": "A component that uses Docker Compose to run images
    from public Amazon ECR and Docker Hub.",
     "ComponentPublisher": "Amazon",
     "ComponentDependencies": {
       "aws.greengrass.DockerApplicationManager": {
         "VersionRequirement": "~2.0.0"
       }
   ```

```
  },
  "Manifests": [
    {
      "Platform": {
        "os": "all"
      },
      "Lifecycle": {
        "run": "docker-compose -f {artifacts:path}/docker-compose.yaml up"
      },
      "Artifacts": [
        {
          "URI": "docker:public.ecr.aws/cloudwatch-agent/cloudwatch-
agent:latest"
        },
        {
          "URI": "docker:mysql:8.0"
        },
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/folder/docker-compose.yaml"
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.MyDockerComposeComponent
ComponentVersion: '1.0.0'
ComponentDescription: 'A component that uses Docker Compose to run images from
 public Amazon ECR and Docker Hub.'
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.DockerApplicationManager:
    VersionRequirement: ~2.0.0
Manifests:
  - Platform:
      os: all
    Lifecycle:
        run: docker-compose -f {artifacts:path}/docker-compose.yaml up
    Artifacts:
```

```
      - URI: "docker:public.ecr.aws/cloudwatch-agent/cloudwatch-agent:latest"
      - URI: "docker:mysql:8.0"
      - URI: "s3://DOC-EXAMPLE-BUCKET/folder/docker-compose.yaml"
```

> **Note**
>
> 若要在 Docker 容器元件中使用處理序間通訊 (IPC)、AWS認證或串流管理員，您必須在執行 Docker 容器時指定其他選項。如需詳細資訊，請參閱下列內容：
>
> - 在 Docker 容器組件中使用進程間通信
> - 在碼頭容器組件中使用AWS憑據（Linux）
> - 在碼頭容器組件中使用流管理器（Linux）

3. 測試元件以確認其如預期般運作。

> **Important**
>
> 您必須先安裝並啟動 Docker 精靈，才能部署元件。

在本機部署元件之後，您可以執行 docker 容器 ls 命令來確認您的容器是否執行。

```
docker container ls
```

4. 當元件準備就緒時，請上傳元件以部署AWS IoT Greengrass到其他核心裝置。如需詳細資訊，請參閱 發佈元件以部署至核心裝置。

從 Amazon ECR 中的私有映像運行碼頭集裝箱

本節說明如何從存放在 Amazon ECR 的私有儲存庫中的 Docker 映像建立執行 Docker 容器的自訂元件。

若要執行泊塢視窗容器

1. 在AWS IoT Greengrass核心裝置上建立自訂元件。使用下列範例配方，它具有下列屬性：
   - Docker 應用程序管理器組件作為依賴關係。此元件可AWS IoT Greengrass讓您管理認證，以便從專用儲存庫下載映像。

- 令牌交換服務組件作為依賴項。此元件可讓您擷AWS IoT Greengrass取AWS登入資料以與 Amazon ECR 互動。
- 在私有 Amazon ECR 儲存庫中指定碼頭映像的元件成品。
- 使用 docker run 從映像建立和啟動容器的生命週期執行指令碼。

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.MyPrivateDockerComponent",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that runs a Docker container from a
 private Amazon ECR image.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.DockerApplicationManager": {
      "VersionRequirement": "~2.0.0"
    },
    "aws.greengrass.TokenExchangeService": {
      "VersionRequirement": "~2.0.0"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "all"
      },
      "Lifecycle": {
        "run": "docker run account-
id.dkr.ecr.region.amazonaws.com/repository[:tag|@digest]"
      },
      "Artifacts": [
        {
          "URI": "docker:account-
id.dkr.ecr.region.amazonaws.com/repository[:tag|@digest]"
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.MyPrivateDockerComponent
ComponentVersion: '1.0.0'
ComponentDescription: 'A component that runs a Docker container from a private
 Amazon ECR image.'
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.DockerApplicationManager:
    VersionRequirement: ~2.0.0
  aws.greengrass.TokenExchangeService:
    VersionRequirement: ~2.0.0
Manifests:
  - Platform:
      os: all
    Lifecycle:
        run: docker run account-id.dkr.ecr.region.amazonaws.com/repository[:tag|
@digest]
    Artifacts:
      - URI: "docker:account-id.dkr.ecr.region.amazonaws.com/repository[:tag|
@digest]"
```

> **Note**
>
> 若要在 Docker 容器元件中使用處理序間通訊 (IPC)、AWS認證或串流管理員，您必須在執行 Docker 容器時指定其他選項。如需詳細資訊，請參閱下列內容：
>
> - 在 Docker 容器組件中使用進程間通信
> - 在碼頭容器組件中使用AWS憑據（Linux）
> - 在碼頭容器組件中使用流管理器（Linux）

2. 測試元件以確認其如預期般運作。

> **Important**
>
> 您必須先安裝並啟動 Docker 精靈，才能部署元件。

在本機部署元件之後，您可以執行 docker 容器 ls 命令來確認您的容器是否執行。

```
docker container ls
```

3. 將元件上傳AWS IoT Greengrass至部署至其他核心裝置。如需詳細資訊，請參閱 發佈元件以部署至核心裝置。

從 Amazon S3 中的映像運行碼頭集裝箱

本節說明如何從存放在 Amazon S3 中的 Docker 映像執行元件中的 Docker 容器。

從 Amazon S3 中的映像執行元件中的 Docker 容器

1. 運行 docker 保存命令來創建一個 Docker 容器的備份。您可以將此備份提供為元件加工品，以便在其上執行容器AWS IoT Greengrass。以映像*##########*，並將 *hello-world.tar* 取代為要建立的封存檔案的名稱。

```
docker save hello-world > artifacts/com.example.MyDockerComponent/1.0.0/hello-world.tar
```

2. 在AWS IoT Greengrass核心裝置上建立自訂元件。使用下列範例配方，它具有下列屬性：

   - 一種生命週期安裝腳本，使用碼 docker 加載從歸檔中加載 Docker 映像。
   - 使用 docker run 從映像建立和啟動容器的生命週期執行指令碼。`--rm`此選項會在容器結束時清除容器。

   JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.MyS3DockerComponent",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that runs a Docker container from an
 image in an S3 bucket.",
  "ComponentPublisher": "Amazon",
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
```

```
      },
      "Lifecycle": {
        "install": {
          "Script": "docker load -i {artifacts:path}/hello-world.tar"
        },
        "run": {
          "Script": "docker run --rm hello-world"
        }
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.MyS3DockerComponent
ComponentVersion: '1.0.0'
ComponentDescription: 'A component that runs a Docker container from an image in
 an S3 bucket.'
ComponentPublisher: Amazon
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install:
        Script: docker load -i {artifacts:path}/hello-world.tar
      run:
        Script: docker run --rm hello-world
```

> **Note**
>
> 若要在 Docker 容器元件中使用處理序間通訊 (IPC)、AWS認證或串流管理員，您必須在執行 Docker 容器時指定其他選項。如需詳細資訊，請參閱下列內容：
>
> - 在 Docker 容器組件中使用進程間通信
> - 在碼頭容器組件中使用AWS憑據（Linux）
> - 在碼頭容器組件中使用流管理器（Linux）

3. 測試元件以確認其如預期般運作。

   在本機部署元件之後，您可以執行 docker 容器 ls 命令來確認您的容器是否執行。

   ```
   docker container ls
   ```

4. 當元件準備就緒時，請將 Docker 映像存檔上傳到 S3 儲存貯體，並將其 URI 新增至元件方案。然後，您可以將元件上傳AWS IoT Greengrass到部署到其他核心裝置。如需詳細資訊，請參閱 發佈元件以部署至核心裝置。

   當您完成時，組件配方應如下列範例所示。

   JSON

   ```
   {
     "RecipeFormatVersion": "2020-01-25",
     "ComponentName": "com.example.MyS3DockerComponent",
     "ComponentVersion": "1.0.0",
     "ComponentDescription": "A component that runs a Docker container from an
    image in an S3 bucket.",
     "ComponentPublisher": "Amazon",
     "Manifests": [
       {
         "Platform": {
           "os": "linux"
         },
         "Lifecycle": {
           "install": {
             "Script": "docker load -i {artifacts:path}/hello-world.tar"
           },
           "run": {
             "Script": "docker run --rm hello-world"
           }
         },
         "Artifacts": [
           {
             "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
   com.example.MyDockerComponent/1.0.0/hello-world.tar"
           }
         ]
       }
     ]
   ```

```
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.MyS3DockerComponent
ComponentVersion: '1.0.0'
ComponentDescription: 'A component that runs a Docker container from an image in
 an S3 bucket.'
ComponentPublisher: Amazon
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install:
        Script: docker load -i {artifacts:path}/hello-world.tar
      run:
        Script: docker run --rm hello-world
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.MyDockerComponent/1.0.0/hello-world.tar
```

在 Docker 容器組件中使用進程間通信

您可以使用中的 Greengrass 處理序間通訊 (IPC) 程式庫與 Greengrass 核心、其他 Greengrass 元件和通訊。AWS IoT Device SDK AWS IoT Core如需詳細資訊，請參閱 使AWS IoT Device SDK用與 Greengrass 核、其他元件和通訊 AWS IoT Core。

若要在 Docker 容器元件中使用 IPC，您必須使用下列參數執行 Docker 容器：

- 將 IPC 插槽安裝在容器中。Greengrass 核在環境變量中提供了 IPC 套接字文件路徑。`AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT`
- 將`SVCUID`和`AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT`環境變量設置為 Greengrass 核提供給組件的值。您的組件使用這些環境變量來驗證與 Greengrass 核的連接。

Example 方法範例：將 MQTT 訊息發佈至 AWS IoT Core (Python)

下列方案定義了將 MQTT 訊息發佈至的範例 Docker 容器元件。AWS IoT Core此配方具有下列屬性：

- 允許元件AWS IoT Core在所有主題上將 MQTT 訊息發佈到的授權原則 (`accessControl`)。如需詳細資訊，請參閱授權元件執行 IPC 作業和 AWS IoT CoreMQTT I PC 授權。
- 在 Amazon S3 中將 Docker 映像指定為 TAR 存檔的元件成品。
- 從 TAR 歸檔載入 Docker 映像檔的生命週期安裝指令碼。
- 從映像執行 Docker 容器的生命週期執行指令碼。碼頭運行命令具有以下參數：
  - 該`-v`參數掛載 Greengrass IPC 套接字在容器中。
  - 前兩個`-e`引數會在 Docker 容器中設定必要的環境變數。
  - 其他`-e`引數會設定此範例所使用的環境變數。
  - 該`--rm`參數在退出時清理容器。

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.python.docker.PublishToIoTCore",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Uses interprocess communication to publish an MQTT
 message to IoT Core.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "topic": "test/topic/java",
      "message": "Hello, World!",
      "qos": "1",
      "accessControl": {
        "aws.greengrass.ipc.mqttproxy": {
          "com.example.python.docker.PublishToIoTCore:pubsub:1": {
            "policyDescription": "Allows access to publish to IoT Core on all
 topics.",
            "operations": [
              "aws.greengrass#PublishToIoTCore"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
```

```
  },
  "Manifests": [
    {
      "Platform": {
        "os": "all"
      },
      "Lifecycle": {
        "install": "docker load -i {artifacts:path}/publish-to-iot-core.tar",
        "run": "docker run -v $AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT:
$AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT -e SVCUID -e
 AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT -e MQTT_TOPIC=
\"{configuration:/topic}\" -e MQTT_MESSAGE=\"{configuration:/message}\" -e MQTT_QOS=
\"{configuration:/qos}\" --rm publish-to-iot-core"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.python.docker.PublishToIoTCore/1.0.0/publish-to-iot-core.tar"
        }
      ]
    }
  ]
}
```

YAML

```
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.python.docker.PublishToIoTCore
ComponentVersion: 1.0.0
ComponentDescription: Uses interprocess communication to publish an MQTT message to
 IoT Core.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    topic: 'test/topic/java'
    message: 'Hello, World!'
    qos: '1'
    accessControl:
      aws.greengrass.ipc.mqttproxy:
        'com.example.python.docker.PublishToIoTCore:pubsub:1':
          policyDescription: Allows access to publish to IoT Core on all topics.
          operations:
            - 'aws.greengrass#PublishToIoTCore'
```

```
          resources:
            - '*'
Manifests:
  - Platform:
      os: all
    Lifecycle:
      install: 'docker load -i {artifacts:path}/publish-to-iot-core.tar'
      run: |
        docker run \
          -v $AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT:
$AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT \
          -e SVCUID \
          -e AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT \
          -e MQTT_TOPIC="{configuration:/topic}" \
          -e MQTT_MESSAGE="{configuration:/message}" \
          -e MQTT_QOS="{configuration:/qos}" \
          --rm publish-to-iot-core
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.python.docker.PublishToIoTCore/1.0.0/publish-to-iot-core.tar
```

在碼頭容器組件中使用AWS憑據（Linux）

您可以使用權杖交換服務元件與 Greengrass 元件中的AWS服務進行互動。此元件會使用本機容器伺服器，從核心裝置的 Token 交換角色提供AWS認證。如需詳細資訊，請參閱 與AWS服務互動。

> **Note**
>
> 本節中的範例僅適用於 Linux 核心裝置。

若要在 Docker 容器元件中使用權杖交換服務的AWS認證，您必須使用下列參數執行 Docker 容器：

- 使用`--network=host`引數提供對主機網路的存取。此選項可讓 Docker 容器連線至本機權杖交換服務，以擷取AWS認證。這個論點僅適用於 Linux 的泊塢窗。

> **Warning**
>
> 此選項可讓容器存取主機上的所有區域網路介面，因此此選項比在沒有主機網路存取權的情況下執行 Docker 容器的安全性較低。當您開發並執行使用此選項的 Docker 容器元件時，請考慮這一點。如需詳細資訊，請參閱 Docker 文件中的網路：主機。

- 將AWS_CONTAINER_CREDENTIALS_FULL_URI和AWS_CONTAINER_AUTHORIZATION_TOKEN環境變量設置為 Greengrass 核提供給組件的值。 AWSSDK 使用這些環境變數來擷取AWS認證。

Example 範例配方：列出碼頭容器元件 (Python) 中的 S3 儲存貯體

下列方案定義了列出. 中 S3 儲存貯體的 AWS 帳戶 Docker 容器元件範例 此配方具有下列屬性：

- 令牌交換服務組件作為依賴項。此相依性可讓元件擷取AWS認證，以便與其他AWS服務互動。
- 在 Amazon S3 中將 Docker 映像指定為 tar 存檔的元件成品。
- 從 TAR 歸檔載入 Docker 映像檔的生命週期安裝指令碼。
- 從映像執行 Docker 容器的生命週期執行指令碼。碼頭運行命令具有以下參數：
  - 該--network=host引數提供了對主機網絡的容器訪問，因此容器可以連接到令牌交換服務。
  - -e引數會在 Docker 容器中設定必要的環境變數。
  - 該--rm參數在退出時清理容器。

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.python.docker.ListS3Buckets",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Uses the token exchange service to lists your S3
 buckets.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.TokenExchangeService": {
      "VersionRequirement": "^2.0.0",
      "DependencyType": "HARD"
    }
  },
  "Manifests": [
```

```
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "docker load -i {artifacts:path}/list-s3-buckets.tar",
        "run": "docker run --network=host -e AWS_CONTAINER_AUTHORIZATION_TOKEN -e
 AWS_CONTAINER_CREDENTIALS_FULL_URI --rm list-s3-buckets"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.python.docker.ListS3Buckets/1.0.0/list-s3-buckets.tar"
        }
      ]
    }
  ]
}
```

YAML

```
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.python.docker.ListS3Buckets
ComponentVersion: 1.0.0
ComponentDescription: Uses the token exchange service to lists your S3 buckets.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.TokenExchangeService:
    VersionRequirement: ^2.0.0
    DependencyType: HARD
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: 'docker load -i {artifacts:path}/list-s3-buckets.tar'
      run: |
        docker run \
          --network=host \
          -e AWS_CONTAINER_AUTHORIZATION_TOKEN \
          -e AWS_CONTAINER_CREDENTIALS_FULL_URI \
          --rm list-s3-buckets
    Artifacts:
```

```
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.python.docker.ListS3Buckets/1.0.0/list-s3-buckets.tar
```

在碼頭容器組件中使用流管理器（Linux）

您可以使用串流管理員元件來管理 Greengrass 元件中的資料串流。此元件可讓您處理資料串流，並將大量 IoT 資料傳輸至. AWS 雲端 AWS IoT Greengrass提供您用來與串流管理員元件互動的串流管理員 SDK。如需詳細資訊，請參閱 管理核心裝 Greengrass 資料串流。

> **Note**
>
> 本節中的範例僅適用於 Linux 核心裝置。

若要在 Docker 容器元件中使用串流管理員 SDK，您必須使用下列參數執行 Docker 容器：

- 使用`--network=host`引數提供對主機網路的存取。此選項可讓 Docker 容器透過本機 TLS 連線與串流管理員元件互動。這個參數只適用於 Linux 的碼頭工具

  > **Warning**
  >
  > 此選項可讓容器存取主機上的所有區域網路介面，因此此選項比在沒有主機網路存取權的情況下執行 Docker 容器的安全性較低。當您開發並執行使用此選項的 Docker 容器元件時，請考慮這一點。如需詳細資訊，請參閱 Docker 文件中的網路：主機。

- 如果您將串流管理員元件設定為需要驗證 (這是預設行為)，請將`AWS_CONTAINER_CREDENTIALS_FULL_URI`環境變數設定為 Greengrass 核心提供給元件的值。如需詳細資訊，請參閱串流管理員組態。
- 如果您將串流管理員元件設定為使用非預設連接埠，請使用處理序間通訊 (IPC)，從串流管理員元件組態取得連接埠。您必須使用其他選項來執行 Docker 容器，才能使用 IPC。如需詳細資訊，請參閱下列內容：
  - 在應用程式程式碼中 Connect 至串流管
  - 在 Docker 容器組件中使用進程間通信

Example 方法範例：將檔案串流至碼頭容器元件 (Python) 中的 S3 儲存貯體

下列方法定義了建立檔案並將其串流至 S3 儲存貯體的範例 Docker 容器元件。此配方具有下列屬性：

- 流管理器組件作為依賴關係。這種依賴關係使組件能夠使用流管理器 SDK 與流管理器組件進行交互。
- 在 Amazon S3 中將 Docker 映像指定為 TAR 存檔的元件成品。
- 從 TAR 歸檔載入 Docker 映像檔的生命週期安裝指令碼。
- 從映像執行 Docker 容器的生命週期執行指令碼。碼頭運行命令具有以下參數：
  - 該`--network=host`引數提供了對主機網絡的容器訪問，因此容器可以連接到流管理器組件。
  - 第一個`-e`引數會在 Docker 容器中設定所需的`AWS_CONTAINER_AUTHORIZATION_TOKEN`環境變數。
  - 其他`-e`引數會設定此範例所使用的環境變數。
  - `-v`引數會在容器中掛載元件的工作資料夾。此範例會在工作資料夾中建立一個檔案，以使用串流管理員將該檔案上傳到 Amazon S3。
  - 該`--rm`參數在退出時清理容器。

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.python.docker.StreamFileToS3",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Creates a text file and uses stream manager to stream the
 file to S3.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.StreamManager": {
      "VersionRequirement": "^2.0.0",
      "DependencyType": "HARD"
    }
  },
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "bucketName": ""
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
```

```
        "install": "docker load -i {artifacts:path}/stream-file-to-s3.tar",
        "run": "docker run --network=host -e AWS_CONTAINER_AUTHORIZATION_TOKEN
 -e BUCKET_NAME=\"{configuration:/bucketName}\" -e WORK_PATH=\"{work:path}\" -v
 {work:path}:{work:path} --rm stream-file-to-s3"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.python.docker.StreamFileToS3/1.0.0/stream-file-to-s3.tar"
        }
      ]
    }
  ]
}
```

YAML

```
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.python.docker.StreamFileToS3
ComponentVersion: 1.0.0
ComponentDescription: Creates a text file and uses stream manager to stream the file
 to S3.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.StreamManager:
    VersionRequirement: ^2.0.0
    DependencyType: HARD
ComponentConfiguration:
  DefaultConfiguration:
    bucketName: ''
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: 'docker load -i {artifacts:path}/stream-file-to-s3.tar'
      run: |
        docker run \
          --network=host \
          -e AWS_CONTAINER_AUTHORIZATION_TOKEN \
          -e BUCKET_NAME="{configuration:/bucketName}" \
          -e WORK_PATH="{work:path}" \
          -v {work:path}:{work:path} \
          --rm stream-file-to-s3
```

```
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.python.docker.StreamFileToS3/1.0.0/stream-file-to-s3.tar
```

AWS IoT Greengrass元件配方參考

元件方案是定義元件詳細資訊、相依性、人工因素和生命週期的檔案。例如，元件生命週期會指定要執行以安裝、執行和關閉元件的命令。AWS IoT Greengrass核心會使用您在方案中定義的生命週期來安裝和執行元件。當您部署元件時，AWS IoT Greengrass服務會使用方案來識別要部署到核心裝置的相依性和成品。

在方案中，您可以為元件支援的每個平台定義唯一的相依性和生命週期。您可以使用此功能將元件部署到具有不同需求的多個平台的裝置。您也可以使用此功能AWS IoT Greengrass來防止在不支援元件的裝置上安裝元件。

每個配方都包含清單列表。每個資訊清單都會指定一組平台需求，以及用於平台符合這些需求的核心裝置的生命週期和成品。核心設備使用具有設備滿足的平台要求的第一個清單。指定沒有任何平台需求的清單以匹配任何核心設備。

您也可以指定不在資訊清單中的全域生命週期。在全域生命週期中，您可以使用選取鍵來識別生命週期的子區段。然後，您可以在資訊清單中指定這些選取鍵，除了資訊清單的生命週期之外，還可以使用全域生命週期的這些區段。僅當清單未定義生命週期時，核心設備才使用清單的選擇鍵。您可以使用資訊清單中的`all`選取項來比對全域生命週期的區段，而不需要選取索引鍵。

AWS IoT Greengrass核心軟體選取符合核心裝置的資訊清單之後，會執行下列動作來識別要使用的生命週期步驟：

- 如果選取的資訊清單定義了生命週期，則核心裝置會使用該生命週期。
- 如果選取的資訊清單未定義生命週期，則核心裝置會使用全域生命週期。核心裝置會執行下列動作，以識別要使用的全域生命週期中的哪些區段：
  - 如果清單定義了選擇鍵，則核心設備將使用包含清單選擇鍵的全局生命週期的部分。
  - 如果資訊清單未定義選取金鑰，核心裝置會使用沒有選取金鑰的全域生命週期區段。此行為等同於定義`all`選取範圍的資訊清單。

> **Important**
>
> 核心裝置必須符合至少一個資訊清單的平台需求，才能安裝元件。如果沒有資訊清單符合核心裝置，則 AWS IoT Greengrass Core 軟體不會安裝元件，且部署失敗。

您可以定義 JSON 或 YAML 格式的配方。食譜示例部分包括每種格式的食譜。

主題

配方驗證

建立元件版本時，Greengrass 會驗證 JSON 或 YAML 元件方案。此方案驗證會檢查您的 JSON 或 YAML 元件方案是否有常見錯誤，以避免潛在的部署問題。驗證會檢查配方是否有常見錯誤 (例如，缺少逗號、大括號和欄位)，並確保配方格式正確。

如果您收到配方驗證錯誤訊息，請檢查您的方案是否有遺漏逗號、大括號或欄位。通過查看配方格式來確認您沒有缺少任何字段。

食譜格式

當您定義元件的方案時，您可以在 recipe 文件中指定下列資訊。相同的結構適用於 YAML 和 JSON 格式的配方。

`RecipeFormatVersion`

該配方的模板版本。選擇下列選項：

- `2020-01-25`

`ComponentName`

此方案定義的元件名稱。您在每個區域中的AWS 帳戶元件名稱必須是唯一的。

> **提示**
>
> - 使用反向域名格式來避免公司內的名稱衝突。例如，如果您的公司擁有`example.com`並且您從事太陽能專案，您可以為 Hello World 元件命名`com.example.solar.HelloWorld`。這有助於避免公司內部的元件名稱衝突。
> - 避免在組件名稱中使用`aws.greengrass`前綴。 AWS IoT Greengrass將此前綴用於其提供的公共組件。如果您選擇與公共組件相同的名稱，則組件將替換該組件。然後，當它部署具有依賴於該公共組件的組件時，AWS IoT Greengrass提供您的組件而不是公共組件。此功能可讓您覆寫公用元件的行為，但如果您不打算覆寫公用元件，它也可能會中斷其他元件。

`ComponentVersion`

元件的版本。主要、次要和修正程式值的最大值為 999999。

> **Note**
>
> AWS IoT Greengrass使用語義版本的組件。語義版本遵循一個主要的。 未成年人。 修補程式編號系統。例如，版本`1.0.0`代表元件的第一個主要發行版本。如需詳細資訊，請參閱語意版本規格。

`ComponentDescription`

(選擇性) 元件的說明。

`ComponentPublisher`

元件的發行者或作者。

`ComponentConfiguration`

(選擇性) 定義元件組態或參數的物件。您可以定義預設組態，然後在部署元件時，指定要提供給元件的組態物件。元件組態支援巢狀參數和結構。此物件包含下列資訊：

`DefaultConfiguration`

定義元件預設組態的物件。您可以定義此物件的結構。

> **Note**
>
> AWS IoT Greengrass使用 JSON 作為組態值。JSON 指定數字類型，但不區分整數和浮點數。因此，組態值可能會轉換為中AWS IoT Greengrass的浮點數。若要確保您的元件使用正確的資料類型，建議您將數值組態值定義為字串。然後，讓你的組件將它們解析為整數或浮點數。如此可確保您的組態值在組態和核心裝置上具有相同的類型。

`ComponentDependencies`

(選擇性) 物件字典，每個物件都會定義元件的元件相依性。每個物件的索引鍵會識別元件相依性的名稱。 AWS IoT Greengrass安裝元件時會安裝元件相依性。 AWS IoT Greengrass等待依賴關係在啟動組件之前啟動。每個物件都包含下列資訊：

`VersionRequirement`

npm-style 語意版本約束，用於定義此相依性的相容元件版本。您可以指定版本或版本範圍。如需詳細資訊，請參閱 npm 語意版本計算器。

`DependencyType`

(選擇性) 此相依性的類型。您可以從以下選項中選擇。

- `SOFT` - 在相依性變更狀態時，元件不會重新啟動。
- `HARD` - 在相依性變更狀態時，元件會重新啟動。

預設為 `HARD`。

`ComponentType`

(選擇性) 元件的類型。

> **Note**
>
> 我們不建議您在方案中指定元件類型。 AWS IoT Greengrass在建立元件時為您設定類型。

該類型可以是以下類型之一：

- `aws.greengrass.generic`— 元件會執行命令或提供加工品。
- `aws.greengrass.lambda`— 元件使用 Lambda 啟動器元件執行 Lambda 函數。此`ComponentSource`參數會指定此元件執行的 Lambda 函數的 ARN。

我們不建議您使用此選項，因為它是在您從 Lambda 函數建立元件AWS IoT Greengrass時設定的。如需詳細資訊，請參閱 執行AWS Lambda函數。

- `aws.greengrass.plugin`— 元件在與 Greengrass 核心相同的 Java 虛擬機器 (JVM) 中執行。如果您部署或重新啟動外掛程式元件，Greengrass 核心會重新啟動。

  插件組件使用與 Greengrass 核相同的日誌文件。如需詳細資訊，請參閱 監控AWS IoT Greengrass日誌。

  我們不建議您在組件配方中使用此選項，因為它適用於使用 Java 編寫且直接與 Greengrass 核心連接的AWS提供組件。如需有關哪些公用元件是外掛程式的詳細資訊，請參閱AWS-提供的組件。

- `aws.greengrass.nucleus`— 細胞核成分. 如需詳細資訊，請參閱 Greengrass 核。

  我們不建議您在組件配方中使用此選項。它適用於 Greengrass 核組件，它提供了核心軟件的最小功能。AWS IoT Greengrass

`aws.greengrass.generic`當您從配方建立元件，或從 Lambda 函數建立元件`aws.greengrass.lambda`時，預設為。

如需詳細資訊，請參閱 元件類型。

`ComponentSource`

(選擇性) 元件執行的 Lambda 函數的 ARN。

我們不建議您在方案中指定元件來源。 AWS IoT Greengrass當您從 Lambda 函數建立元件時，會為您設定此參數。如需詳細資訊，請參閱 執行AWS Lambda函數。

`Manifests`

一份物件清單，每個物件都會定義元件的生命週期、參數和平台需求。如果核心裝置符合多個資訊清單的平台需求，請AWS IoT Greengrass使用核心裝置符合的第一個資訊清單。若要確保核心裝置使用正確的資訊清單，請先定義具有更嚴格平台需求的資訊清單。適用於所有平台的資訊清單必須是清單中的最後一個資訊清單。

> **Important**
>
> 核心裝置必須符合至少一個資訊清單的平台需求，才能安裝元件。如果沒有資訊清單符合核心裝置，則 AWS IoT Greengrass Core 軟體不會安裝元件，且部署失敗。

每個物件都包含下列資訊：

`Name`

(選擇性) 此資訊清單定義之平台的易記名稱。

如果您省略此參數，請從平台`os`和AWS IoT Greengrass建立名稱`architecture`。

`Platform`

(選擇性) 定義套用此資訊清單之平台的物件。省略此參數，以定義適用於所有平台的資訊清單。

此物件會指定有關核心裝置執行之平台的索引鍵值配對。當您部署此元件時，AWS IoT GreengrassCore 軟體會比較這些索引鍵值配對與核心裝置上的平台屬性。AWS IoT GreengrassCore 軟件始終定義`os`和`architecture`，並且它可能會定義其他屬性。您可以在部署 Greengrass 核心元件時指定核心裝置的自訂平台屬性。如需詳細資訊，請參閱 Greengrass 核心元件的平台覆寫參數。

對於每個鍵值對，您可以指定下列其中一個值：

- 精確的值，例如`linux`或`windows`。精確值必須以字母或數字開頭。
- `*`，符合任何值。當一個值不存在時，這也匹配。
- Java 樣式的規則運算式，例如. `/windows|linux/` 規則運算式必須以斜線字元 (/) 開頭和結尾。例如，規則運算式會`/.+/`比對任何非空白值。

此物件包含下列資訊：

`os`

(選擇性) 此資訊清單支援之平台的作業系統名稱。一般平台包含下列值：

- `linux`
- `windows`
- `darwin` (macOS)

`architecture`

(選擇性) 此資訊清單支援之平台的處理器架構。通用架構包括以下值：

- `amd64`
- `arm`
- `aarch64`

- x86

`architecture.detail`

(選擇性) 此資訊清單支援之平台的處理器架構詳細資料。一般架構詳細資料包括下列值：

- arm61
- arm71
- arm81

key

(選擇性) 您為此資訊清單定義的平台屬性。將#替換為平台屬性的名稱。AWS IoT Greengrass核心軟體會將此平台屬性與您在 Greengrass 核心元件組態中指定的索引鍵值配對相符。如需詳細資訊，請參閱 Greengrass 核心元件的平台覆寫參數。

> **Tip**
>
> 使用反向域名格式來避免公司內的名稱衝突。例如，如果您的公司擁有`example.com`並且您在某個無線電專案上工作，您可以命名自訂平台屬性`com.example.radio.RadioModule`。這有助於避免公司內的平台屬性名稱衝突。

例如，您可以定義 platform 屬性`com.example.radio.RadioModule`，以根據核心裝置上可用的無線電模組來指定不同的資訊清單。每個資訊清單可以包含套用至不同硬體組態的不同成品，以便您將最小的軟體組合部署到核心裝置。

`Lifecycle`

定義如何在此資訊清單定義的平台上安裝及執行元件的物件或字串。您也可以定義適用於所有平台的全域生命週期。只有當要使用的資訊清單未指定生命週期時，核心裝置才會使用全域生命週期。

> **Note**
>
> 您可以在資訊清單中定義此生命週期。您在此處指定的生命週期步驟僅適用於此資訊清單定義的平台。您也可以定義適用於所有平台的全域生命週期。

此物件或字串包含下列資訊：

`Setenv`

(選擇性) 提供給所有生命週期指令碼的環境變數字典。您可以`Setenv`在每個生命週期指令碼中覆寫這些環境變數。

`install`

(選擇性) 定義元件安裝時要執行之指令碼的物件或字串。AWS IoT Greengrass核心軟體也會在每次軟體啟動時執行此生命週期步驟。

如果指`install`令碼以成功代碼結束，元件就會進入`INSTALLED`狀態。

此物件或字串包含下列資訊：

`Script`

要執行的指令碼。

`RequiresPrivilege`

(選擇性) 您可以使用 root 權限執行命令檔。如果您將此選項設定為`true`，則 AWS IoT Greengrass Core 軟體會以 root 身分執行此生命週期指令碼，而不是以您設定來執行此元件的系統使用者身分執行此生命週期指令碼。預設為 `false`。

`Skipif`

(選擇性) 決定是否要執行指令碼的檢查。您可以定義檢查可執行文件是否在路徑上或文件是否存在。如果輸出為 true，則AWS IoT Greengrass核心軟體會略過該步驟。選擇下列其中一項檢查：

- `onpath` *`runnable`*-檢查系統路徑上是否可運行。例如，如果 Python 3 可**`onpath python3`**用，則使用跳過此生命週期步驟。
- `exists` *`file`*-檢查文件是否存在。例如，如果`/tmp/my-configuration.db`存在**`exists /tmp/my-configuration.db`**，則使用跳過此生命週期步驟。

`Timeout`

(選擇性) 在 AWS IoT Greengrass Core 軟體終止處理程序之前，指令碼可以執行的時間上限 (以秒為單位)。

預設值：120 秒

`Setenv`

(選擇性) 要提供給指令集的環境變數字典。這些環境變數會覆寫您在中提供的變數`Lifecycle.Setenv`。

`run`

(選擇性) 定義元件啟動時要執行之指令碼的物件或字串。

執行此生命週期步驟時，元件會進入RUNNING狀態。如果指run令碼以成功代碼結束，元件就會進入STOPPING狀態。如果指定了指shutdown令碼，它就會執行；否則，元件就會進入FINISHED狀態。

依賴此元件的元件會在此生命週期步驟執行時啟動。若要執行背景處理序 (例如相依元件所使用的服務)，請改用startup生命週期步驟。

當您部署具有run生命週期的元件時，核心裝置可以在此生命週期指令碼執行後立即報告部署完成。因此，即使run生命週期指令碼在執行後不久失敗，部署也可以完成且成功。如果您希望部署狀態取決於元件啟動指令碼的結果，請改用startup生命週期步驟。

> **Note**
>
> 您只能定義一個startup或run生命週期。

此物件或字串包含下列資訊：

`Script`

要執行的指令碼。

`RequiresPrivilege`

(選擇性) 您可以使用 root 權限執行命令檔。如果您將此選項設定為true，則 AWS IoT Greengrass Core 軟體會以 root 身分執行此生命週期指令碼，而不是以您設定來執行此元件的系統使用者身分執行此生命週期指令碼。預設為 false。

`Skipif`

(選擇性) 決定是否要執行指令碼的檢查。您可以定義檢查可執行文件是否在路徑上或文件是否存在。如果輸出為 true，則AWS IoT Greengrass核心軟體會略過該步驟。選擇下列其中一項檢查：

- onpath *runnable*-檢查系統路徑上是否可運行。例如，如果 Python 3 可**onpath python3**用，則使用跳過此生命週期步驟。
- exists *file*-檢查文件是否存在。例如，如果/tmp/my-configuration.db存在**exists /tmp/my-configuration.db**，則使用跳過此生命週期步驟。

`Timeout`

(選擇性) 在 AWS IoT Greengrass Core 軟體終止處理程序之前，指令碼可以執行的時間上限 (以秒為單位)。

依預設，此生命週期步驟不會逾時。如果您省略此逾時，`run`指令碼會執行直到結束為止。

`Setenv`

(選擇性) 要提供給指令集的環境變數字典。這些環境變數會覆寫您在中提供的變數`Lifecycle.Setenv`。

`startup`

(選擇性) 定義元件啟動時要執行的背景處理程序的物件或字串。

用`startup`於執行必須成功結束的指令，或將元件的狀態更新為，`RUNNING`才能啟動相依元件。使用 UpdateStateIPC 作業將元件的狀態設定為`RUNNING`或`ERRORED`當元件啟動未結束的指令碼時。例如，您可以定義一個啟動 MySQL 處理程序的`startup`步驟`/etc/init.d/mysqld start`。

執行此生命週期步驟時，元件會進入`STARTING`狀態。如果指`startup`令碼以成功代碼結束，元件就會進入`RUNNING`狀態。然後，從屬元件即可啟動。

當您部署具有`startup`生命週期的元件時，核心裝置可以在此生命週期指令碼結束或報告其狀態之後，將部署報告為完成。換句話說，部署的狀態是`IN_PROGRESS`直到所有元件的啟動指令碼結束或報告狀態為止。

> **Note**
>
> 您只能定義一個`startup`或`run`生命週期。

此物件或字串包含下列資訊：

`Script`

要執行的指令碼。

`RequiresPrivilege`

(選擇性) 您可以使用 root 權限執行命令檔。如果您將此選項設定為`true`，則 AWS IoT Greengrass Core 軟體會以 root 身分執行此生命週期指令碼，而不是以您設定來執行此元件的系統使用者身分執行此生命週期指令碼。預設為 `false`。

`Skipif`

(選擇性) 決定是否要執行指令碼的檢查。您可以定義檢查可執行文件是否在路徑上或文件是否存在。如果輸出為 true，則AWS IoT Greengrass核心軟體會略過該步驟。選擇下列其中一項檢查：

- `onpath` *`runnable`*-檢查系統路徑上是否可運行。例如，如果 Python 3 可**`onpath python3`**用，則使用跳過此生命週期步驟。
- `exists` *`file`*-檢查文件是否存在。例如，如果`/tmp/my-configuration.db`存在**`exists /tmp/my-configuration.db`**，則使用跳過此生命週期步驟。

`Timeout`

(選擇性) 在 AWS IoT Greengrass Core 軟體終止處理程序之前，指令碼可以執行的時間上限 (以秒為單位)。

預設值：120 秒

`Setenv`

(選擇性) 要提供給指令集的環境變數字典。這些環境變數會覆寫您在中提供的變數`Lifecycle.Setenv`。

`shutdown`

(選擇性) 定義元件關閉時要執行之指令碼的物件或字串。使用關閉生命週期來執行您要在元件處於`STOPPING`狀態時執行的程式碼。關閉生命週期可用來停止由`startup`或`run`指令碼啟動的處理程序。

如果您在中啟動背景處理程序`startup`，請使用`shutdown`步驟在元件關閉時停止該程序。例如，您可以定義停止 MySQL 處理程序的`shutdown`步驟`/etc/init.d/mysqld stop`。

指`shutdown`令碼會在元件進入`STOPPING`狀態後執行。如果指令碼順利完成，元件就會進入`FINISHED`狀態。

此物件或字串包含下列資訊：

`Script`

要執行的指令碼。

`RequiresPrivilege`

(選擇性) 您可以使用 root 權限執行命令檔。如果您將此選項設定為`true`，則 AWS IoT Greengrass Core 軟體會以 root 身分執行此生命週期指令碼，而不是以您設定來執行此元件的系統使用者身分執行此生命週期指令碼。預設為 `false`。

`Skipif`

(選擇性) 決定是否要執行指令碼的檢查。您可以定義檢查可執行文件是否在路徑上或文件是否存在。如果輸出為 true，則AWS IoT Greengrass核心軟體會略過該步驟。選擇下列其中一項檢查：

- `onpath` *`runnable`*-檢查系統路徑上是否可運行。例如，如果 Python 3 可**`onpath python3`**用，則使用跳過此生命週期步驟。
- `exists` *`file`*-檢查文件是否存在。例如，如果`/tmp/my-configuration.db`存在**`exists /tmp/my-configuration.db`**，則使用跳過此生命週期步驟。

`Timeout`

(選擇性) 在 AWS IoT Greengrass Core 軟體終止處理程序之前，指令碼可以執行的時間上限 (以秒為單位)。

預設值：15 秒。

`Setenv`

(選擇性) 要提供給指令集的環境變數字典。這些環境變數會覆寫您在中提供的變數`Lifecycle.Setenv`。

`recover`

(選擇性) 定義元件發生錯誤時要執行之指令碼的物件或字串。

此步驟會在元件進入`ERRORED`狀態時執行。如果元件變成`ERRORED`三次而未成功復原，則元件會變更為`BROKEN`狀態。若要修正`BROKEN`元件，您必須重新部署它。

此物件或字串包含下列資訊：

`Script`

要執行的指令碼。

`RequiresPrivilege`

(選擇性) 您可以使用 root 權限執行命令檔。如果您將此選項設定為`true`，則 AWS IoT Greengrass Core 軟體會以 root 身分執行此生命週期指令碼，而不是以您設定來執行此元件的系統使用者身分執行此生命週期指令碼。預設為 `false`。

`Skipif`

(選擇性) 決定是否要執行指令碼的檢查。您可以定義檢查可執行文件是否在路徑上或文件是否存在。如果輸出為 true，則AWS IoT Greengrass核心軟體會略過該步驟。選擇下列其中一項檢查：

- `onpath` *`runnable`*-檢查系統路徑上是否可運行。例如，如果 Python 3 可**`onpath python3`**用，則使用跳過此生命週期步驟。
- `exists` *`file`*-檢查文件是否存在。例如，如果`/tmp/my-configuration.db`存在**`exists /tmp/my-configuration.db`**，則使用跳過此生命週期步驟。

`Timeout`

(選擇性) 在 AWS IoT Greengrass Core 軟體終止處理程序之前，指令碼可以執行的時間上限 (以秒為單位)。

預設：60 秒。

`Setenv`

(選擇性) 要提供給指令集的環境變數字典。這些環境變數會覆寫您在中提供的變數`Lifecycle.Setenv`。

`bootstrap`

(選擇性) 定義需要AWS IoT Greengrass核心軟體或核心裝置重新啟動的指令碼的物件或字串。例如，這可讓您開發元件，在安裝作業系統更新或執行階段更新之後執行重新啟動。

> **Note**
>
> 若要安裝不需要AWS IoT Greengrass核心軟體或裝置重新啟動的更新或相依性，請使用安裝生命週期。

此生命週期步驟會在安裝生命週期步驟之前執行，在下列情況下，AWS IoT Greengrass核心軟體部署元件時：

- 元件會首次部署至核心裝置。
- 元件版本會變更。
- 啟動程序指令碼會隨著元件組態更新而變更。

在AWS IoT Greengrass核心軟體完成部署中具有啟動程序步驟之所有元件的啟動程序步驟之後，軟體便會重新啟動。

> **Important**
>
> 您必須將 AWS IoT Greengrass Core 軟體設定為系統服務，才能重新啟動 AWS IoT Greengrass Core 軟體或核心裝置。如果您未將 AWS IoT Greengrass Core 軟體設定為系統服務，軟體就不會重新啟動。如需詳細資訊，請參閱 將 Greengrass 核配置為系統服務。

此物件或字串包含下列資訊：

`BootstrapOnRollback`

> **Note**
>
> 啟用此功能時，只`BootstrapOnRollback`會針對已完成或嘗試執行啟動程序生命週期步驟作為失敗目標部署的一部分的元件執行。此功能適用於 Greengrass 2.12.0 及更高版本。

(選擇性) 您可以執行啟動程序生命週期步驟，做為復原部署的一部分。如果將此選項設定為`true`，則會執行倒回部署中定義的啟動程序生命週期步驟。當部署失敗時，先前版本的元件啟動程序生命週期將在復原部署期間再次執行。

預設為 `false`。

`Script`

要執行的指令碼。這個腳本的退出代碼定義了 restart 指令。使用以下退出代碼：

- `0`— 不要重新啟動 AWS IoT Greengrass Core 軟件或核心設備。AWS IoT Greengrass核心軟件在所有組件引導後仍然重新啟動。
- `100`— 要求重新啟動AWS IoT Greengrass核心軟體。

- 101— 請求重新啟動核心設備。

結束代碼 100 到 199 會保留用於特殊行為。其他結束代碼代表指令碼錯誤。

`RequiresPrivilege`

(選擇性) 您可以使用 root 權限執行命令檔。如果您將此選項設定為`true`，則 AWS IoT Greengrass Core 軟體會以 root 身分執行此生命週期指令碼，而不是以您設定來執行此元件的系統使用者身分執行此生命週期指令碼。預設為 `false`。

`Timeout`

(選擇性) 在 AWS IoT Greengrass Core 軟體終止處理程序之前，指令碼可以執行的時間上限 (以秒為單位)。

預設值：120 秒

`Setenv`

(選擇性) 要提供給指令集的環境變數字典。這些環境變數會覆寫您在中提供的變數`Lifecycle.Setenv`。

`Selections`

(選擇性) 選取索引鍵清單，指定要針對此資訊清單執行的全域生命週期區段。在全域生命週期中，您可以使用任何層級的選取鍵來定義生命週期步驟，以選取生命週期的子區段。然後，核心設備使用與此清單中的選擇鍵匹配的那些部分。如需詳細資訊，請參閱全域生命週期範例。

> **Important**
>
> 只有在此資訊清單未定義生命週期時，核心裝置才會使用全域生命週期中的選取項目。

您可以指定`all`選取鍵來執行沒有選取鍵的全域生命週期區段。

`Artifacts`

(選擇性) 物件清單，每個物件都會定義此資訊清單所定義之平台上元件的二進位成品。例如，您可以將程式碼或影像定義為人工因素。

部署元件時，AWS IoT Greengrass核心軟體會將成品下載至核心裝置上的資料夾。您也可以將人工因素定義為軟體下載後擷取的封存檔案。

您可以使用 recipe 變數取得成品安裝在核心裝置上的資料夾路徑。

- 一般檔案 — 使用人工因素:path recipe 變數取得包含成品之資料夾的路徑。例如，在方案`{artifacts:path}/my_script.py`中指定以取得具有 URI 之成品的路徑`s3://DOC-EXAMPLE-BUCKET/path/to/my_script.py`。
- 已擷取的存檔 — 使用「人工因素:解壓縮路徑 recipe」變數，取得包含已解壓縮歸檔加工品之資料夾的路徑。AWS IoT Greengrass核心軟體會將每個歸檔解壓縮到與歸檔名稱相同的資料夾中。例如，在方案`{artifacts:decompressedPath}/my_archive/my_script.py`中指定要在具有 URI `my_script.py` 的歸檔加工品中取得路徑`s3://DOC-EXAMPLE-BUCKET/path/to/my_archive.zip`。

> **Note**
>
> 當您在本機核心裝置上開發具有封存人工因素的元件時，您可能沒有該成品的 URI。若要使用擷取人工因素的`Unarchive`選項測試元件，請指定檔案名稱與歸檔人工因素檔案名稱相符的 URI。您可以指定預期上傳歸檔人工因素的 URI，也可以指定新的預留位置 URI。例如，若要在本機部署期間擷取`my_archive.zip`成品，您可以指定`s3://DOC-EXAMPLE-BUCKET/my_archive.zip`。

每個物件都包含下列資訊：

`URI`

S3 儲存貯體中成品的 URI。AWS IoT Greengrass核心軟體會在元件安裝時從此 URI 擷取成品，除非該成品已存在於裝置上。每個成品在每個資訊清單中都必須有唯一的檔案名稱。

`Unarchive`

(選擇性) 要解壓縮的歸檔類型。您可以從以下選項中選擇：

- `NONE`-該文件不是要解壓縮的存檔。AWS IoT Greengrass核心軟體會將成品安裝至核心裝置上的資料夾。您可以使用人工因素:path recipe 變數來取得此資料夾的路徑。
- `ZIP`— 檔案為 ZIP 封存檔。AWS IoT Greengrass核心軟體會將歸檔解壓縮到與歸檔名稱相同的資料夾中。您可以使用人工因素:解壓縮 Path recipe 變數來取得包含此資料夾之資料夾的路徑。

預設為 `NONE`。

`Permission`

(選擇性) 定義要為此成品檔案設定之存取權限的物件。您可以設定讀取權限和執行權限。

> **Note**
>
> 您無法設定寫入權限，因為 AWS IoT Greengrass Core 軟體不允許元件編輯成品資料夾中的成品檔案。若要編輯元件中的人工因素檔案，請將其複製到其他位置，或發佈並部署新的人工因素檔案。

如果您將人工因素定義為要解壓縮的歸檔，則 AWS IoT Greengrass Core 軟體會對其從歸檔解壓縮的檔案設定這些存取權限。AWS IoT Greengrass核心軟體會將資料夾的存取權限設定ALL為Read和Execute。這允許組件查看文件夾中解壓縮的文件。若要設定封存中個別檔案的權限，您可以在安裝生命週期指令碼中設定權限。

此物件包含下列資訊：

`Read`

(選擇性) 要為此成品檔案設定的讀取權限。若要允許其他元件存取此人工因素 (例如相依於此元件的元件)，請指定ALL。您可以從以下選項中選擇：

- NONE— 檔案無法讀取。
- OWNER— 您配置為執行此元件的系統使用者可讀取檔案。
- ALL— 檔案可供所有使用者讀取。

預設為 OWNER。

`Execute`

(選擇性) 要為此成品檔案設定的執行權限。Execute權限意味著Read權限。例如，如果您指定ALL為Execute，則所有使用者都可以讀取並執行此成品檔案。

您可以從以下選項中選擇：

- NONE— 檔案無法執行。
- OWNER— 檔案可由您配置為執行元件的系統使用者執行。
- ALL— 檔案可由所有使用者執行。

預設為 NONE。

`Digest`

(唯讀) 成品的密碼編譯摘要雜湊。建立元件時，AWS IoT Greengrass會使用雜湊演算法來計算人工因素檔案的雜湊值。然後，當您部署元件時，Greengrass 核心會計算下載成品的雜湊

值，並將雜湊與此摘要進行比較，以便在安裝前驗證成品。如果雜湊不符合摘要，部署會失敗。

如果設定此參數，則AWS IoT Greengrass會取代您在建立元件時設定的值。

`Algorithm`

(唯讀) AWS IoT Greengrass 用來計算成品摘要雜湊的雜湊演算法。

如果設定此參數，則AWS IoT Greengrass會取代您在建立元件時設定的值。

`Lifecycle`

定義如何安裝和執行元件的物件。只有當要使用的資訊清單未指定生命週期時，核心裝置才會使用全域生命週期。

> **Note**
>
> 您可以在資訊清單外定義此生命週期。您也可以定義適用於符合該資訊清單的平台的資訊清單生命週期。

在全域生命週期中，您可以指定針對在每個資訊清單中指定的特定選取鍵執行的生命週期。選取索引鍵是字串，可識別要針對每個資訊清單執行的全域生命週期區段。

`all`選取鍵是沒有選取鍵的任何區段上的預設值。這表示您可以在資訊清單中指定`all`選取鍵，以執行全域生命週期的區段，而不需要選取索引鍵。您不需要在全域生命週期中指定`all`選取鍵。

如果清單未定義生命週期或選擇鍵，則核心設備默認使用該`all`選擇。這表示在這種情況下，核心裝置會使用不使用選取金鑰的全域生命週期區段。

此物件包含與資訊清單生命週期相同的資訊，但您可以在任何層級指定選取鍵來選取生命週期的子區段。

> **Tip**
>
> 建議您對每個選取鍵僅使用小寫字母，以避免選取鍵與生命週期金鑰之間發生衝突。生命週期金鑰以大寫字母開頭。

Example 具有最上層選取鍵的全域生命週期

```
Lifecycle:
```

```
  key1:
    install:
      Skipif: either onpath executable or exists file
      Script: command1
  key2:
    install:
      Script: command2
  all:
    install:
      Script: command3
```

Example 具有底層選取鍵的全域生命週期範例

```
Lifecycle:
  install:
    Script:
      key1: command1
      key2: command2
      all: command3
```

Example 具有多個選取鍵層級的全域生命週期範例

```
Lifecycle:
  key1:
    install:
      Skipif: either onpath executable or exists file
      Script: command1
  key2:
    install:
      Script: command2
  all:
    install:
      Script:
        key3: command3
        key4: command4
        all: command5
```

配方變數

配方變數會公開來自目前元件和原子核的資訊，供您在食譜中使用。例如，您可以使用 recipe 變數，將元件組態參數傳遞至您在生命週期指令碼中執行的應用程式。

您可以在組件配方的以下部分中使用 recipe 變量：

- 生命週期定義。
- 零組件模型組態定義，如果您使用 Greengrass 核 v2.6.0 或更新版本，並將組態選項設定為。interpolateComponentConfigurationtrue您也可以在部署元件組態更新時使用方法變數。

配方變量使用{recipe_variable}語法。大括號表示方法變數。

AWS IoT Greengrass支援下列配方變數：

component_dependency_name:configuration:*json_pointer*

此方案所定義之元件的組態參數值，或此元件相依之元件的組態參數值。

您可以使用此變數，為在元件生命週期中執行的指令碼提供參數。

> **Note**
>
> AWS IoT Greengrass僅在元件生命週期定義中支援此 recipe 變數。

此 recipe 變數具有下列輸入：

- component_dependency_name— (選擇性) 要查詢的元件相依性名稱。省略此區段，即可查詢此方案定義的元件。您只能指定直接相依性。
- json_pointer— 要評估之組態值的 JSON 指標。JSON 指針以正斜線開頭/。要識別巢狀零組件模型組態中的值，請使用正斜線 (/) 來分隔模型組態中每個層級的關鍵字。您可以使用數字作為鍵來指定列表中的索引。如需詳細資訊，請參閱 JSON 指標規格。

  AWS IoT Greengrass核心使用 JSON 指針在 YAML 格式的配方。

JSON 指針可以引用以下節點類型：

- 值節點。 AWS IoT Greengrass核心替換配方變量與值的字符串表示。Null 值會轉換null為字串。
- 物件節點。 AWS IoT Greengrass核心替換配方變量與該對象的序列化 JSON 字符串表示。
- 沒有節點。 AWS IoT Greengrass核心不會取代配方變數。

例如，{configuration:/Message}recipe 變數會擷取元件組態中的索Message引鍵值。{com.example.MyComponentDependency:configuration:/server/port}recipe 變數會擷取元件相依性port之server組態物件中的值。

component_dependency_name:artifacts:path

此方案所定義之元件的成品根路徑，或此元件相依之元件的根路徑。

安裝元件時，會將元件的成品AWS IoT Greengrass複製到此變數公開的資料夾中。例如，您可以使用此變數來識別要在元件生命週期中執行的指令碼位置。

此路徑上的資料夾是唯讀的。若要修改人工因素檔案，請將檔案複製到其他位置，例如目前工作目錄 (`$PWD`或`.`)。然後，在那裡修改文件。

若要從元件相依性讀取或執行成品，該成品`Read`或`Execute`權限必須為`ALL`。如需詳細資訊，請參閱您在元件方案中定義的成品權限。

此 recipe 變數具有下列輸入：

- component_dependency_name— (選擇性) 要查詢的元件相依性名稱。省略此區段，即可查詢此方案定義的元件。您只能指定直接相依性。

component_dependency_name:artifacts:decompressedPath

此方案定義之元件或此元件相依之元件的解壓縮歸檔人工因素的根路徑。

安裝元件時，將元AWS IoT Greengrass件的封存加工品解壓至此變數公開的資料夾。例如，您可以使用此變數來識別要在元件生命週期中執行的指令碼位置。

每個成品都會解壓縮至解壓縮路徑內的資料夾，其中資料夾的名稱與成品相同，減去其副檔名。例如，名為的 ZIP 人工因素會`models.zip`解壓縮至`{artifacts:decompressedPath}/models`資料夾。

此路徑上的資料夾是唯讀的。若要修改人工因素檔案，請將檔案複製到其他位置，例如目前工作目錄 (`$PWD`或`.`)。然後，在那裡修改文件。

若要從元件相依性讀取或執行成品，該成品`Read`或`Execute`權限必須為`ALL`。如需詳細資訊，請參閱您在元件方案中定義的成品權限。

此 recipe 變數具有下列輸入：

- component_dependency_name— (選擇性) 要查詢的元件相依性名稱。省略此區段，即可查詢此方案定義的元件。您只能指定直接相依性。

component_dependency_name:work:path

此功能適用於 v2.0.4 及更高版 Greeng rass 核組件。

此方案所定義之元件的工作路徑，或此元件相依之元件的工作路徑。此 recipe 變數的值等同於從元件內容執行時的`$PWD`環境變數和 pwd 命令的輸出。

您可以使用此 recipe 變數在元件和相依性之間共用檔案。

此路徑上的資料夾可由此配方定義的元件以及以相同使用者和群組身分執行的其他元件讀取和寫入。

此 recipe 變數具有下列輸入：

- `component_dependency_name`— (選擇性) 要查詢的元件相依性名稱。省略此區段，即可查詢此方案定義的元件。您只能指定直接相依性。

`kernel:rootPath`

AWS IoT Greengrass核心根路徑。

`iot:thingName`

此功能適用於 v2.3.0 及更高版 Greeng rass 核組件。

核心裝置的名稱。AWS IoT

食譜示例

您可以參考下列配方範例，以協助您建立元件的配方。

AWS IoT Greengrass策劃 Greengrass 組件的索引，稱為 Greengrass 軟件目錄。此目錄追蹤 Greengrass 社群所開發的 Greengrass 元件。您可以從此目錄下載、修改和部署元件，以建立 Greengrass 應用程式。如需詳細資訊，請參閱 社群元件。

主題

- 你好世界組件食譜
- Python 運行時組件示例
- 指定多個字段的組件配方

你好世界組件食譜

下列配方說明執行 Python 指令碼的「你好世界」元件。此元件支援所有平台，並接受作為引`Message`數AWS IoT Greengrass傳遞給 Python 指令碼的參數。這是在入門教程中 Hello World 組件的配方。

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.HelloWorld",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "My first AWS IoT Greengrass component.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "Message": "world"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "run": "python3 -u {artifacts:path}/hello_world.py {configuration:/Message}"
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "run": "py -3 -u {artifacts:path}/hello_world.py {configuration:/Message}"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.HelloWorld
ComponentVersion: '1.0.0'
ComponentDescription: My first AWS IoT Greengrass component.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
```

```
    Message: world
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      run: |
        python3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
  - Platform:
      os: windows
    Lifecycle:
      run: |
        py -3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
```

Python 運行時組件示例

下面的配方描述了一個安裝 Python 的組件。此元件支援 64 位元 Linux 裝置。

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PythonRuntime",
  "ComponentDescription": "Installs Python 3.7",
  "ComponentPublisher": "Amazon",
  "ComponentVersion": "3.7.0",
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
        "architecture": "amd64"
      },
      "Lifecycle": {
        "install": "apt-get update\napt-get install python3.7"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
```

```
ComponentName: com.example.PythonRuntime
ComponentDescription: Installs Python 3.7
ComponentPublisher: Amazon
ComponentVersion: '3.7.0'
Manifests:
  - Platform:
      os: linux
      architecture: amd64
    Lifecycle:
      install: |
        apt-get update
        apt-get install python3.7
```

指定多個字段的組件配方

下列元件方案使用數個方案欄位。

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.FooService",
  "ComponentDescription": "Complete recipe for AWS IoT Greengrass components",
  "ComponentPublisher": "Amazon",
  "ComponentVersion": "1.0.0",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "TestParam": "TestValue"
    }
  },
  "ComponentDependencies": {
    "BarService": {
      "VersionRequirement": "^1.1.0",
      "DependencyType": "SOFT"
    },
    "BazService": {
      "VersionRequirement": "^2.0.0"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
```

```
        "architecture": "amd64"
      },
      "Lifecycle": {
        "install": {
          "Skipif": "onpath git",
          "Script": "sudo apt-get install git"
        },
        "Setenv": {
          "environment_variable1": "variable_value1",
          "environment_variable2": "variable_value2"
        }
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/hello_world.zip",
          "Unarchive": "ZIP"
        },
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/hello_world_linux.py"
        }
      ]
    },
    {
      "Lifecycle": {
        "install": {
          "Skipif": "onpath git",
          "Script": "sudo apt-get install git",
          "RequiresPrivilege": "true"
        }
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/hello_world.py"
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
```

```
ComponentName: com.example.FooService
ComponentDescription: Complete recipe for AWS IoT Greengrass components
ComponentPublisher: Amazon
ComponentVersion: 1.0.0
ComponentConfiguration:
  DefaultConfiguration:
    TestParam: TestValue
ComponentDependencies:
  BarService:
    VersionRequirement: ^1.1.0
    DependencyType: SOFT
  BazService:
    VersionRequirement: ^2.0.0
Manifests:
  - Platform:
      os: linux
      architecture: amd64
    Lifecycle:
      install:
        Skipif: onpath git
        Script: sudo apt-get install git
      Setenv:
        environment_variable1: variable_value1
        environment_variable2: variable_value2
    Artifacts:
      - URI: 's3://DOC-EXAMPLE-BUCKET/hello_world.zip'
        Unarchive: ZIP
      - URI: 's3://DOC-EXAMPLE-BUCKET/hello_world_linux.py'
  - Lifecycle:
      install:
        Skipif: onpath git
        Script: sudo apt-get install git
        RequiresPrivilege: 'true'
    Artifacts:
      - URI: 's3://DOC-EXAMPLE-BUCKET/hello_world.py'
```

元件環境變數參考

AWS IoT Greengrass核心軟體會在執行元件的生命週期指令碼時設定環境變數。您可以在組件中獲取這些環境變量以獲取物件名稱AWS 區域，和 Greengrass 核版本。此軟體也會設定元件使用處理序間通訊 SDK 以及與AWS服務互動所需的環境變數。

您也可以為元件的生命週期指令碼設定自訂環境變數。如需詳細資訊，請參閱 Setenv。

AWS IoT Greengrass核心軟體設定下列環境變數：

`AWS_IOT_THING_NAME`

代表此 Greengrass 核心裝置的AWS IoT物件名稱。

`AWS_REGION`

這AWS 區域個 Greengrass 核心設備的運行位置。

AWSSDK 使用此環境變數來識別要使用的預設區域。此變數相當於`AWS_DEFAULT_REGION`。

`AWS_DEFAULT_REGION`

這AWS 區域個 Greengrass 核心設備的運行位置。

AWS CLI使用此環境變數來識別要使用的預設「區域」。此變數相當於`AWS_REGION`。

`GGC_VERSION`

在此 Greengrass 核心裝置上執行的 Greengrass 核心元件的版本。

`GG_ROOT_CA_PATH`

此功能適用於 v2.5.5 及更高版 Greengrass 核組件。

Greengrass 核心使用的根憑證授權機構 (CA) 憑證的根憑證授權機構 (CA) 憑證。

`AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT`

元件用來與AWS IoT Greengrass Core 軟體通訊的 IPC 通訊端路徑。如需詳細資訊，請參閱使 AWS IoT Device SDK用與 Greengrass 核、其他元件和通訊 AWS IoT Core。

`SVCUID`

組件用於連接到 IPC 套接字並與AWS IoT Greengrass Core 軟件進行通信的密鑰令牌。如需詳細資訊，請參閱使AWS IoT Device SDK用與 Greengrass 核、其他元件和通訊 AWS IoT Core。

`AWS_CONTAINER_AUTHORIZATION_TOKEN`

元件用來從權杖交換服務元件擷取認證的秘密權杖。

`AWS_CONTAINER_CREDENTIALS_FULL_URI`

元件要求從權杖交換服務元件擷取認證的 URI。

將AWS IoT Greengrass元件部署到裝置

您可以使用AWS IoT Greengrass將元件部署到裝置或裝置群組。您可以使用部署來定義傳送至裝置的元件和組態。 AWS IoT Greengrass部署至代表 Greengrass 核心裝置的目標、AWS IoT物件或物群組。 AWS IoT Greengrass使用AWS IoT Core工作部署到您的核心裝置。您可以設定如何將工作推出至您的裝置。

核心裝置部署

每個核心裝置都會執行該裝置的部署元件。相同目標的新部署會覆寫先前的部署至目標。建立部署時，您可以定義要套用至核心裝置現有軟體的元件和組態。

修訂目標的部署時，您可以使用新修訂版中的元件取代先前版序中的元件。例如，您可以將日誌管理器和秘密經理元件部署到物群組TestGroup。然後您建立另一個僅指TestGroup定祕密管理員元件的部署。因此，該群組中的核心裝置將不再執行記錄管理員。

平台相依性解析

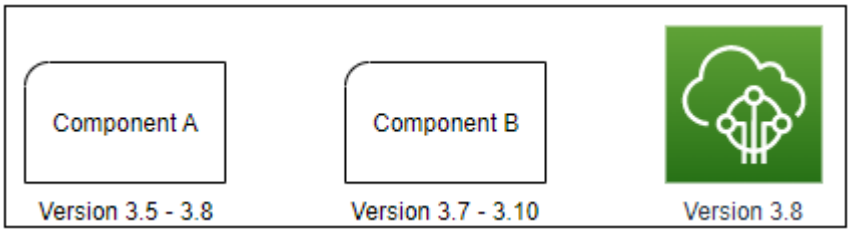
當核心裝置收到部署時，它會檢查以確定元件與核心裝置相容。例如，如果您將部署Firehose到 Windows 目標，則部署將會失敗。

元件相依性解析

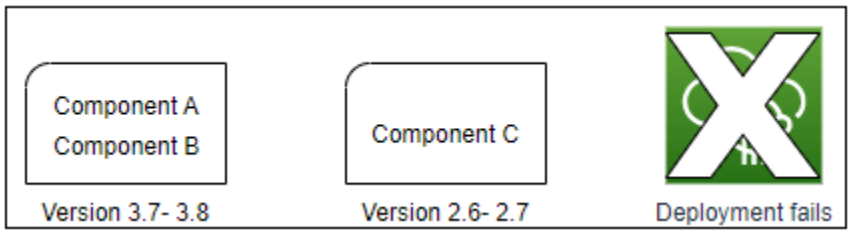
核心裝置也會檢查每個元件相依性是否與將其他元件部署至此物件群組的版本限制相容。當元件的版本限制重疊時，Greengrass 會使用最高適用的元件版本。例如：

- 您將元件 A 部署到TestGroup。元件 A 取決於元件com.example.PythonRuntime版本 3.5-3.10。
- 然後，您可以將元件 B 部署到TestGroup. 元件 B 取決於元件com.example.PythonRuntime版本 3.7 至 3.8。

因此，中的核心裝置會TestGroup判斷它們可以部署com.example.PythonRuntime元件 3.8 版，因為此版本是版本限制重疊的最高適用版本。



然後，您將元件 C 部署到`TestGroup`. 組件 C 取決於組件`com.example.PythonRuntime`版本 2.6-2.7。此部署失敗，因為沒有符合條件約束 2.6-2.7 和 3.7-3.8 的元件版本。



從物件群組移除裝置

當您從物件群組中移除核心裝置時，元件部署行為取決於核心裝置執行的 Greengrass 核心版本。

2.5.1 and later

當您從物件群組移除核心裝置時，行為會視AWS IoT原則是否授與`greengrass:ListThingGroupsForCoreDevice`權限而定。如需有關此權限和核心裝置AWS IoT原則的詳細資訊，請參閱AWS IoT Greengrass 的裝置身分驗證和授權。

- 如果政AWS IoT策授予此權限

  當您從物件群組中AWS IoT Greengrass移除核心裝置時，會在下次對裝置進行部署時移除該物群組的元件。如果設備上的某個元件包含在下一個部署中，則不會從設備中移除該元件。

- 如果原AWS IoT則未授與此權限

  當您從物件群組中移除核心裝置時，AWS IoT Greengrass不會從裝置刪除該物件群組的元件。

  若要從裝置移除元件，請使用 Greengrass CLI 的部署建立指令。使用引數指定要移除的元件，並使用`--remove`引`--groupId`數指定物件群組。

2.5.0

當您從物件群組中AWS IoT Greengrass移除核心裝置時，會在下次對裝置進行部署時移除該物群組的元件。如果設備上的某個元件包含在下一個部署中，則不會從設備中移除該元件。

此行為需要核心裝置的AWS IoT策略授與`greengrass:ListThingGroupsForCoreDevice`權限。如果核心裝置沒有此權限，核心裝置將無法套用部署。如需詳細資訊，請參閱 AWS IoT Greengrass 的裝置身分驗證和授權。

2.0.x - 2.4.x

當您從物件群組中移除核心裝置時，AWS IoT Greengrass不會從裝置刪除該物件群組的元件。

若要從裝置移除元件，請使用 Greengrass CLI 的部署建立指令。使用引數指定要移除的元件，並使用`--remove`引`--groupId`數指定物件群組。

部署

部署是連續的。建立部署時，將部署AWS IoT Greengrass推出至線上的目標裝置。如果目標裝置未連線，則會在下次連線時接收部署AWS IoT Greengrass。當您將核心裝置新增至目標物件群組時，AWS IoT Greengrass會將該物件群組的最新部署傳送給設備。

核心裝置部署元件之前，依預設會通知裝置上的每個元件。Greengrass 元件可以回應延遲部署的通知。如果裝置的電池電量不足或執行無法中斷的程序，您可能會想要延遲部署。如需詳細資訊，請參閱 教學課程：開發延遲元件更新的 Greengrass 元件。建立部署時，您可以將其設定為部署而不通知元件。

每個目標物件或物群組一次可以有一個部署。這表示當您為目標建立部署時，AWS IoT Greengrass不再部署該目標部署的先前修訂版。

部署選項

部署提供數個選項，可讓您控制哪些裝置接收更新，以及更新部署的方式。建立部署時，您可以設定下列選項：

- AWS IoT Greengrass元件

  定義要在目標裝置上安裝和執行的元件。 AWS IoT Greengrass元件是您在 Greengrass 核心裝置上部署和執行的軟體模組。只有當元件支援裝置的平台時，裝置才會接收元件。這可讓您部署到裝置群組，即使目標裝置在多個平台上執行。如果組件不支持設備的平台，則該組件不會部署到設備。

  您可以將自訂元件和AWS提供的元件部署到您的裝置。當您部署元件時，請AWS IoT Greengrass識別任何元件相依性並加以部署。如需詳細資訊，請參閱 開發AWS IoT Greengrass元件 及 AWS-提供的組件。

  您可以定義要為每個元件部署的版本和組態更新。組態更新會指定如何修改核心裝置上元件的現有組態，或如果核心裝置上沒有元件，則修改元件的預設組態。您可以指定要重設為預設值的組態值，以及要合併至核心裝置的新組態值。當核心裝置接收不同目標的部署，且每個部署都指定了相容的元件

版本時，核心裝置會根據您建立部署時的時間戳記，依序套用組態更新。如需詳細資訊，請參閱 更新零組件組態。

> **Important**
>
> 當您部署元件時，AWS IoT Greengrass會安裝該元件所有相依性的最新受支援版本。因此，如果您將新裝置新增至物件群組，或更新以這些裝置為目標的部署，則AWS提供之公用元件的新修補程式版本可能會自動部署到核心裝置。某些自動更新 (例如核心更新) 可能會導致裝置意外重新啟動。
> 若要避免對裝置上執行的元件進行意外更新，建議您在建立部署時直接包含該元件的偏好版本。如需有關 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱更新AWS IoT Greengrass核心軟件（OTA）。

- 部署原則

  定義何時安全部署設定，以及部署失敗時該如何處理。您可以指定是否等待元件回報它們可以更新。如果設備套用失敗的部署，您也可以指定是否將設備還原為其先前的組態。

- 停止組態

  定義停止部署的時間和方式。如果符合您定義的準則，部署就會停止並失敗。例如，您可以將部署規劃為在最少數量的設備接收到部署後，若有一定百分比的設備無法套用該部署，則可以停止部署。

- 推展組態

  定義部署至目標裝置的速率。您可以設定具有最小和最大速率界限的指數速率增加。

- 逾時設定

  定義每個設備套用部署的時間上限。如果設備超過您指定的持續時間，則設備將無法套用部署。

> **Important**
>
> 自訂元件可以定義 S3 儲存貯體中的成品。AWS IoT Greengrass核心軟體部署元件時，會從. AWS 雲端 核心裝置角色預設不允許存取 S3 儲存貯體。若要部署在 S3 儲存貯體中定義成品的自訂元件，核心裝置角色必須授與從該儲存貯體下載成品的權限。如需更多詳細資訊，請參閱允許存取 S3 儲存貯體的存取。

主題

建立部署

您可以建立以物件或物群組為目標的部署。

建立部署時，您可以規劃要部署的軟體元件，以及如何將部署工作推出至目標裝置。您可以在提供給的 JSON 檔案中定義部署AWS CLI。

部署目標會決定您要在其上執行元件的裝置。若要部署至一個核心裝置，請指定物件。若要部署到多個核心裝置，請指定包含這些裝置的物件群組。如需有關如何設定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與動態物件群組。

請遵循本節中的步驟來建立目標的部署。如需如何在具有部署之目標上更新軟體元件的詳細資訊，請參閱修訂部署。

> **Warning**
>
> 該CreateDeployment操作可以從核心設備卸載組件。如果元件存在於先前的部署中，而不是新部署中，則核心裝置會解除安裝該元件。若要避免解除安裝元件，請先使用此ListDeployments作業來檢查部署的目標是否已有現有部署。然後，在建立新部署時，使用該GetDeployment作業從該現有部署開始。

若要建立部署 (AWS CLI)

1. 建立名為的檔案`deployment.json`，然後將下列 JSON 物件複製到檔案中。以要*########AWS IoT####### ARN ## Tar* Tarn。物件與物件群組 ARN 具有下列格式：

   - 物件：`arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`thingName`*
   - 物件群組：`arn:aws:iot:`*`region`*`:`*`account-id`*`:thinggroup/`*`thingGroupName`*

   ```
   {
   ```

```
  "targetArn": "targetArn"
}
```

2. 檢查部署目標是否具有您要修訂的現有部署。請執行下列操作：

   a. 執行下列命令以列出部署目標的部署。以目標AWS IoT物件或物### ARN #### targe tArn n。

   ```
   aws greengrassv2 list-deployments --target-arn targetArn
   ```

   回應包含一份包含目標最新部署的清單。如果回應為空白，表示目標沒有現有的部署，您可以跳到Step 3。否則，請deploymentId從響應中復制以在下一個步驟中使用。

   > **Note**
   >
   > 您也可以修訂目標的最新修訂版以外的部署。指定--history-filter ALL引數以列出目標的所有部署。然後，複製您要修訂的部署 ID。

   b. 執行下列命令以取得部署的詳細資料。這些詳細資料包括中繼資料、元件和工作組態。使用上一個步驟的 ID 取代## ID。

   ```
   aws greengrassv2 get-deployment --deployment-id deploymentId
   ```

   回應包含部署的詳細資料。

   c. 將以下任何鍵-值對從上一個命令的響應複製到deployment.json。您可以變更新部署的這些值。

   - deploymentName— 部署的名稱。
   - components— 部署的元件。若要解除安裝元件，請將其從此物件中移除。
   - deploymentPolicies— 部署的原則。
   - iotJobConfiguration— 部署的工作組態。
   - tags— 部署的標籤。

3. (選擇性) 定義部署的名稱。將####取代為部署的名稱。

```
{
  "targetArn": "targetArn",
  "deploymentName": "deploymentName"
```

```
}
```

4. 新增每個元件以部署目標裝置。若要這麼做，請將索引鍵值配對新增至`components`物件，其中索引鍵是元件名稱，而值是包含該元件詳細資訊的物件。為您新增的每個元件指定下列詳細資訊：

   - `version`— 要部署的元件版本。
   - `configurationUpdate`— 要部署的組態更新。更新是修補程式作業，會修改元件在每個目標裝置上的現有組態，或修改元件的預設組態 (如果目標裝置上不存在)。您可以指定下列組態更新：
     - 重設更新 (`reset`) — (選用) JSON 指標清單，用於定義要在目標裝置上重設為預設值的組態值。AWS IoT Greengrass Core 軟體會先套用重設更新，然後再套用合併更新。如需詳細資訊，請參閱 重設更新。
     - 合併更新 (`merge`) — (選用) 定義要合併到目標裝置的組態值的 JSON 文件。您必須將 JSON 文檔序列化為字符串。如需詳細資訊，請參閱 合併更新。
   - `runWith`— (選擇性) AWS IoT Greengrass Core 軟體用來在核心裝置上執行此元件處理程序的系統處理程序選項。如果您省略`runWith`物件中的參數，AWS IoT Greengrass核心軟體會使用您在 Greengrass 核心元件上設定的預設值。

     您可以指定下列任何選項：

     - `posixUser`— 用於在 Linux 核心裝置上執行此元件的 POSIX 系統使用者和 (選擇性) 群組。使用者和群組 (若有指定) 必須存在於每個 Linux 核心裝置上。以下列格式指定使用者和群組，並以冒號 (`:`) 分隔：`user:group`。群組為選用項目。如果您未指定群組，AWS IoT GreengrassCore 軟體會使用使用者的主要群組。如需詳細資訊，請參閱 設定執行元件的使用者。
     - `windowsUser`— 用來在 Windows 核心裝置上執行此元件的 Windows 使用者。使用者必須存在於每個 Windows 核心裝置上，且其名稱和密碼必須儲存在 LocalSystem 帳戶的認證管理員執行個體中。如需詳細資訊，請參閱 設定執行元件的使用者。

       此功能適用於 v2.5.0 及更高版 Greeng rass 核組件。
     - `systemResourceLimits`— 系統資源限制套用至此元件的處理序。您可以將系統資源限制套用至一般和非容器化 Lambda 元件。如需詳細資訊，請參閱 設定元件的系統資源限制。

       您可以指定下列任何選項：

       - `cpus`— 此元件的處理序可以在核心裝置上使用的 CPU 時間上限。核心裝置的 CPU 總時間等於裝置的 CPU 核心數。例如，在具有 4 個 CPU 核心的核心裝置上，您可以將此值設定為2將此元件的處理序限制為每個 CPU 核心的 50% 使用率。在具有 1 個 CPU 核心的

裝置上，您可以將此值設定0.25為將此元件的處理序限制為 CPU 使用率 25%。如果您將此值設定為大於 CPU 核心數目的數字，AWS IoT Greengrass核心軟體就不會限制元件的 CPU 使用率。

- memory— 此元件的處理序可在核心裝置上使用的最大 RAM 量 (以 KB 為單位)。

此功能適用於 v2.4.0 和更高版 Greeng rass 核組件。 AWS IoT Greengrass目前在 Windows 核心裝置上不支援此功能。

Example 範例基本組態更新

下列範例components物件指定要部署需要名為的組態參數的元件pythonVersion。com.example.PythonRuntime

```
{
  "targetArn": "targetArn",
  "deploymentName": "deploymentName",
  "components": {
    "com.example.PythonRuntime": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "merge": "{\"pythonVersion\":\"3.7\"}"
      }
    }
  }
}
```

Example 包含重設和合併更新的範例組態更新

請考慮具有下列預設組態的工業儀表板元件範例。com.example.IndustrialDashboard

```
{
  "name": null,
  "mode": "REQUEST",
  "network": {
    "useHttps": true,
    "port": {
      "http": 80,
      "https": 443
    },
```

```
  },
  "tags": []
}
```

下列組態更新會指定下列指示：

1. 將 HTTPS 設定重設為預設值 (true)。
2. 將工業標籤清單重設為空白清單。
3. 合併工業標籤清單，以識別兩個鍋爐的溫度和壓力資料流。

```
{
  "reset": [
    "/network/useHttps",
    "/tags"
  ],
  "merge": {
    "tags": [
      "/boiler/1/temperature",
      "/boiler/1/pressure",
      "/boiler/2/temperature",
      "/boiler/2/pressure"
    ]
  }
}
```

下列範例components物件指定要部署此工業儀表板元件和組態更新。

```
{
  "targetArn": "targetArn",
  "deploymentName": "deploymentName",
  "components": {
    "com.example.IndustrialDashboard": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "reset": [
          "/network/useHttps",
          "/tags"
        ],
        "merge": "{\"tags\":[\"/boiler/1/temperature\",\"/boiler/1/pressure\",\"/
boiler/2/temperature\",\"/boiler/2/pressure\"]}"
```

```
      }
    }
  }
}
```

5. (選擇性) 定義部署的部署原則。您可以設定核心裝置何時可以安全地套用部署，或是核心裝置無法套用部署時該如何處理。若要這樣做，請將deploymentPolicies物件新增至deployment.json，然後執行下列任一項作業：

   1. (選擇性) 指定元件更新原則 (componentUpdatePolicy)。此原則定義部署是否允許元件延遲更新，直到準備好更新為止。例如，元件可能需要清理資源或完成重要動作，才能重新啟動以套用更新。此原則也會定義元件必須回應更新通知的時間量。

      此原則是具有下列參數的物件：

      - action— (選用) 是否要通知元件，並等待它們在準備更新時報告。您可以從以下選項中選擇：
        - NOTIFY_COMPONENTS – 部署會在停止前向每個元件發出通知並更新該元件。元件可以使用 SubscribeToComponentUpdates IPC 作業來接收這些通知。
        - SKIP_NOTIFY_COMPONENTS – 部署不會向元件發出通知，也不會等到可安全更新元件時才更新。

        預設為 NOTIFY_COMPONENTS。
      - timeoutInSeconds每個元件必須透過 DeferComponentUpdate IPC 作業回應更新通知的時間 (以秒為單位)。如果元件沒有在這段時間內回應，則部署會在核心裝置上繼續進行。

        預設值為 60 秒。

   2. (選擇性) 指定組態驗證原則 (configurationValidationPolicy)。此原則定義每個元件必須驗證部署中組態更新的時間長度。元件可以使用 SubscribeToValidateConfigurationUpdates IPC 作業來訂閱其自身組態更新的通知。然後，組件可以使用 SendConfigurationValidityReport IPC 操作來告訴 AWS IoT Greengrass Core 軟件配置更新是否有效。如果組態更新無效，則部署會失敗。

      此原則是具有下列參數的物件：

      - timeoutInSeconds(選擇性) 每個元件驗證組態更新所需的時間 (秒)。如果元件沒有在這段時間內回應，則部署會在核心裝置上繼續進行。

        預設值為 30 秒。

3. (選擇性) 指定失敗處理原則 (`failureHandlingPolicy`)。此原則是一個字串，用來定義部署失敗時是否要復原裝置。您可以從以下選項中選擇：

   - `ROLLBACK`— 如果核心裝置上的部署失敗，則 AWS IoT Greengrass Core 軟體會將該核心裝置回復為先前的組態。
   - `DO_NOTHING`— 如果核心裝置上的部署失敗，則 AWS IoT Greengrass Core 軟體會保留新的組態。如果新的組態無效，這可能會導致損毀的元件。

   預設為 `ROLLBACK`。

您在中的部署看起來`deployment.json`可能類似下列範例：

```
{
  "targetArn": "targetArn",
  "deploymentName": "deploymentName",
  "components": {
    "com.example.IndustrialDashboard": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "reset": [
          "/network/useHttps",
          "/tags"
        ],
        "merge": "{\"tags\":[\"/boiler/1/temperature\",\"/boiler/1/pressure\",\"/
boiler/2/temperature\",\"/boiler/2/pressure\"]}"
      }
    }
  },
  "deploymentPolicies": {
    "componentUpdatePolicy": {
      "action": "NOTIFY_COMPONENTS",
      "timeoutInSeconds": 30
    },
    "configurationValidationPolicy": {
      "timeoutInSeconds": 60
    },
    "failureHandlingPolicy": "ROLLBACK"
  }
}
```

6. (選擇性) 定義部署停止、推出或逾時的方式。 AWS IoT Greengrass使用AWS IoT Core工作將部署傳送至核心裝置，因此這些選項與AWS IoT Core工作的組態選項相同。如需詳細資訊，請參閱AWS IoT開發人員指南中的 Job 推出和中止設定。

   若要定義工作選項，請將iotJobConfiguration物件新增至deployment.json。然後，定義要設定的選項。

   您在中的部署看起來deployment.json可能類似下列範例：

```
{
  "targetArn": "targetArn",
  "deploymentName": "deploymentName",
  "components": {
    "com.example.IndustrialDashboard": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "reset": [
          "/network/useHttps",
          "/tags"
        ],
        "merge": "{\"tags\":[\"/boiler/1/temperature\",\"/boiler/1/pressure\",\"/
boiler/2/temperature\",\"/boiler/2/pressure\"]}"
      }
    }
  },
  "deploymentPolicies": {
    "componentUpdatePolicy": {
      "action": "NOTIFY_COMPONENTS",
      "timeoutInSeconds": 30
    },
    "configurationValidationPolicy": {
      "timeoutInSeconds": 60
    },
    "failureHandlingPolicy": "ROLLBACK"
  },
  "iotJobConfiguration": {
    "abortConfig": {
      "criteriaList": [
        {
          "action": "CANCEL",
          "failureType": "ALL",
          "minNumberOfExecutedThings": 100,
          "thresholdPercentage": 5
```

```
        }
      ]
    },
    "jobExecutionsRolloutConfig": {
      "exponentialRate": {
        "baseRatePerMinute": 5,
        "incrementFactor": 2,
        "rateIncreaseCriteria": {
          "numberOfNotifiedThings": 10,
          "numberOfSucceededThings": 5
        }
      },
      "maximumPerMinute": 50
    },
    "timeoutConfig":  {
      "inProgressTimeoutInMinutes": 5
    }
  }
}
```

7. (選擇性) 為部署新增標籤 (tags)。如需詳細資訊，請參閱 標記您的 AWS IoT Greengrass Version 2 資源。
8. 執行下列命令以從中建立部署deployment.json。

```
aws greengrassv2 create-deployment --cli-input-json file://deployment.json
```

回應包括識別deploymentId此部署的。您可以使用部署 ID 來檢查部署的狀態。如需更多詳細資訊，請參閱 檢查部署狀態。

更新零組件組態

組件配置是定義每個組件參數的 JSON 對象。每個元件的配方都會定義其預設組態，您可以在將元件部署到核心裝置時修改這些組態。

建立部署時，您可以指定要套用至每個元件的規劃更新。組態更新是修補程式作業，這表示更新會修改核心裝置上存在的元件組態。如果核心裝置沒有元件，則組態更新會修改並套用該部署的預設組態。

組態更新會定義重設更新和合併更新。重設更新定義要重設為預設值或移除的組態值。合併更新會定義要為零組件設定的新組態值。當您部署組態更新時， AWS IoT Greengrass 核心軟體會在合併更新之前執行重設更新。

元件可以驗證您部署的組態更新。元件會訂閱在部署變更其組態時接收通知，而且可以拒絕不支援的組態。如需詳細資訊，請參閱 與組件配置互動。

主題

- 重設更新
- 合併更新
- 範例

重設更新

重設更新會定義要在核心裝置上將哪些組態值重設為其預設值。如果組態值沒有預設值，則重設更新會從元件的組態中移除該值。這可協助您修正因組態無效而中斷的元件。

使用 JSON 指標清單來定義要重設的組態值。JSON 指針以正斜線開頭/。要識別巢狀零組件模型組態中的值，請使用正斜線 (/) 來分隔模型組態中每個層級的關鍵字。如需詳細資訊，請參閱 JSON 指標規格。

> **Note**
>
> 您只能將整個清單重設為預設值。您無法使用重設更新來重設清單中的個別元素。

若要將組件的整個組態重設為預設值，請指定單一空字串作為重設更新。

```
"reset": [""]
```

合併更新

合併更新定義要插入核心元件組態中的組態值。合併更新是一個 JSON 物件， AWS IoT Greengrass 核心軟體會在重設更新中指定的路徑中重設值之後合併。當您使用 AWS CLI 或 AWS SDK 時，您必須將此 JSON 物件序列化為字串。

您可以合併元件預設組態中不存在的索引鍵值配對。您也可以合併具有與具有相同索引鍵之值不同類型的鍵值配對。新值會取代舊值。這表示您可以變更配置物件的結構。

您可以合併 null 值和空字串、清單和物件。

> **Note**
>
> 您不能將合併更新用於將元素插入或附加到列表中。您可以取代整個清單，也可以定義每個元素都有唯一索引鍵的物件。
>
> AWS IoT Greengrass 使用 JSON 作為組態值。JSON 指定了一個數字類型，但不區分整數和浮點數。因此，組態值可能會轉換為中 AWS IoT Greengrass的浮點數。若要確保您的元件使用正確的資料類型，建議您將數值組態值定義為字串。然後，讓你的組件將它們解析為整數或浮點數。如此可確保您的組態值在組態和核心裝置上具有相同的類型。

在合併更新中使用配方變數

此功能適用於 v2.6.0 及更高版 Greeng rass 核組件。

如果您將 Greengrass 核心的interpolateComponentConfiguration組態選項設定為`true`，則可以在合併更新中使用 recipe 變數以外的*component_dependency_name*`:configuration:`*json_pointer*配方變數。例如，您可以在合併更新中使用 `{iot:thingName}` recipe 變數，將核心裝置的 AWS IoT 物件名稱包含在元件組態值中，例如處理序間通訊 (IPC) 授權原則。

範例

下列範例示範具有下列預設組態之儀表板元件的組態更新。此範例元件顯示有關工業設備的資訊。

```
{
  "name": null,
  "mode": "REQUEST",
  "network": {
    "useHttps": true,
    "port": {
      "http": 80,
      "https": 443
    },
  },
  "tags": []
}
```

工業儀表板元件配方

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.IndustrialDashboard",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Displays information about industrial equipment.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "name": null,
      "mode": "REQUEST",
      "network": {
        "useHttps": true,
        "port": {
          "http": 80,
          "https": 443
        },
      },
      "tags": []
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "run": "python3 -u {artifacts:path}/industrial_dashboard.py"
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "run": "py -3 -u {artifacts:path}/industrial_dashboard.py"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.IndustrialDashboard
ComponentVersion: '1.0.0'
ComponentDescription: Displays information about industrial equipment.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    name: null
    mode: REQUEST
    network:
      useHttps: true
      port:
        http: 80
        https: 443
    tags: []
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      run: |
        python3 -u {artifacts:path}/industrial_dashboard.py
  - Platform:
      os: windows
    Lifecycle:
      run: |
        py -3 -u {artifacts:path}/industrial_dashboard.py
```

Example 範例 1：合併更新

您可以建立套用下列規劃更新的部署，該更新會指定合併更新，但不指定重設更新。此組態更新會告訴元件在 HTTP 連接埠 8080 上顯示儀表板，其中包含來自兩個鍋爐的資料。

Console

要合併的組態

```
{
  "name": "Factory 2A",
  "network": {
```

```
    "useHttps": false,
    "port": {
      "http": 8080
    }
  },
  "tags": [
    "/boiler/1/temperature",
    "/boiler/1/pressure",
    "/boiler/2/temperature",
    "/boiler/2/pressure"
  ]
}
```

AWS CLI

下列指令會建立核心裝置的部署。

```
aws greengrassv2 create-deployment --cli-input-json file://dashboard-deployment.json
```

該文dashboard-deployment.json件包含以下 JSON 文檔。

```
{
  "targetArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
  "deploymentName": "Deployment for MyGreengrassCore",
  "components": {
    "com.example.IndustrialDashboard": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "merge": "{\"name\":\"Factory 2A\",\"network\":{\"useHttps\":false,\"port
\":{\"http\":8080}},\"tags\":[\"/boiler/1/temperature\",\"/boiler/1/pressure\",\"/
boiler/2/temperature\",\"/boiler/2/pressure\"]}"
      }
    }
  }
}
```

Greengrass CLI

以下 Greengrass CLI 命令會在核心裝置上建立本機部署。

```
sudo greengrass-cli deployment create \
```

```
  --recipeDir recipes \
  --artifactDir artifacts \
  --merge "com.example.IndustrialDashboard=1.0.0" \
  --update-config dashboard-configuration.json
```

該文dashboard-configuration.json件包含以下 JSON 文檔。

```
{
  "com.example.IndustrialDashboard": {
    "MERGE": {
      "name": "Factory 2A",
      "network": {
        "useHttps": false,
        "port": {
          "http": 8080
        }
      },
      "tags": [
        "/boiler/1/temperature",
        "/boiler/1/pressure",
        "/boiler/2/temperature",
        "/boiler/2/pressure"
      ]
    }
  }
}
```

此更新之後，儀表板元件具有下列組態。

```
{
  "name": "Factory 2A",
  "mode": "REQUEST",
  "network": {
    "useHttps": false,
    "port": {
      "http": 8080,
      "https": 443
    }
  },
  "tags": [
    "/boiler/1/temperature",
    "/boiler/1/pressure",
```

```
    "/boiler/2/temperature",
    "/boiler/2/pressure"
  ]
}
```

Example 範例 2：重設和合併更新

然後，您會建立套用下列規劃更新的部署，該更新會指定重設更新和合併更新。這些更新會指定在預設 HTTPS 連接埠上顯示儀表板，其中包含來自不同鍋爐的資料。這些更新會修改前一個範例中組態更新所產生的組態。

Console

重設路徑

```
[
  "/network/useHttps",
  "/tags"
]
```

要合併的組態

```
{
  "tags": [
    "/boiler/3/temperature",
    "/boiler/3/pressure",
    "/boiler/4/temperature",
    "/boiler/4/pressure"
  ]
}
```

AWS CLI

下列指令會建立核心裝置的部署。

```
aws greengrassv2 create-deployment --cli-input-json file://dashboard-
deployment2.json
```

該文dashboard-deployment2.json件包含以下 JSON 文檔。

```
{
```

```
  "targetArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
  "deploymentName": "Deployment for MyGreengrassCore",
  "components": {
    "com.example.IndustrialDashboard": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "reset": [
          "/network/useHttps",
          "/tags"
        ],
        "merge": "{\"tags\":[\"/boiler/3/temperature\",\"/boiler/3/pressure\",\"/
boiler/4/temperature\",\"/boiler/4/pressure\"]}"
      }
    }
  }
}
```

Greengrass CLI

以下 Greengrass CLI 命令會在核心裝置上建立本機部署。

```
sudo greengrass-cli deployment create \
  --recipeDir recipes \
  --artifactDir artifacts \
  --merge "com.example.IndustrialDashboard=1.0.0" \
  --update-config dashboard-configuration2.json
```

該文`dashboard-configuration2.json`件包含以下 JSON 文檔。

```
{
  "com.example.IndustrialDashboard": {
    "RESET": [
      "/network/useHttps",
      "/tags"
    ],
    "MERGE": {
      "tags": [
        "/boiler/3/temperature",
        "/boiler/3/pressure",
        "/boiler/4/temperature",
        "/boiler/4/pressure"
      ]
    }
```

```
    }
  }
```

此更新之後，儀表板元件具有下列組態。

```
{
  "name": "Factory 2A",
  "mode": "REQUEST",
  "network": {
    "useHttps": true,
    "port": {
      "http": 8080,
      "https": 443
    }
  },
  "tags": [
    "/boiler/3/temperature",
    "/boiler/3/pressure",
    "/boiler/4/temperature",
    "/boiler/4/pressure",
  ]
}
```

建立子部署

> **Note**
>
> 在 Greengrass 核 2.9.0 版及更新版本上提供子部署功能。您無法將組態部署至具有較早版本 Greengrass 核心元件的子部署。

子部署是以父系部署中較小的裝置子集為目標的部署。您可以使用子部署將組態部署到較小的裝置子集。您也可以建立子部署，以便在父系部署中的一或多個裝置失敗時重試失敗的父系部署。使用此功能，您可以選取該父系部署中失敗的設備，並建立子部署以測試組態，直到子部署成功為止。一旦子部署成功，您可以將該組態重新部署至父系部署。

請遵循本節中的步驟來建立子部署並檢查其狀態。如需如何建立部署的相關資訊，請參閱建立部署。

若要建立子部署 () AWS CLI

1. 執行下列命令以擷取物群組的最新部署。將指令中的 ARN 取代為要查詢之物件群組的 ARN。設定`--history-filter`為**`LATEST_ONLY`**以查看該物件群組的最新部署。

   ```
   aws greengrassv2 list-deployments --target-arn arn:aws:iot:region:account-
   id:thinggroup/thingGroupName --history-filter LATEST_ONLY
   ```

2. 將回應`deploymentId`複製到要在下一個步驟中使用的list-deployments指令。
3. 執行下列命令以擷取部署狀態。以要查詢的部署 ID 取*`deploymentId`*代。

   ```
   aws greengrassv2 get-deployment --deployment-id deploymentId
   ```

4. 將回應複製到要在下一步中使用的get-deployment指令。`iotJobId`
5. 執行下列命令以擷取指定工作的工作執行項目清單。用上一步`iotJobId`中的替換 *`Jobid`*。將*`#`*`#`替換為您要過濾的狀態。您可以篩選具有下列狀態的結果：

   - `QUEUED`
   - `IN_PROGRESS`
   - `SUCCEEDED`
   - `FAILED`
   - `TIMED_OUT`
   - `REJECTED`
   - `REMOVED`
   - `CANCELED`

   ```
   aws iot list-job-executions-for-job --job-id jobID --status status
   ```

6. 為子部署建立新的AWS IoT物件群組，或使用現有的物件群組。然後，將AWS IoT物件新增至此物件群組。您可以使用物件群組來管理 Greengrass 核心裝置的叢集。將軟體元件部署到裝置時，您可以針對個別裝置或裝置群組。您可以使用作用中 Greengrass 部署將設備新增至物件群組。新增之後，您就可以將該物件群組的軟體元件部署到該裝置。

   欲建立新物件群組並將裝置新增至該群組，請執行下列動作：

   a. 建立物AWS IoT件群組。*`MyGreengrassCoreGroup`*以新物件群組的名稱取代。您不能在物件群組名稱中使用冒號 (:)。

> **Note**
>
> 如果子部署的物件群組與其中一個物件群組搭配使用parentTargetArn，則無法在不同的父系叢集中重複使用該物件群組。如果已使用物件群組為另一個叢集建立子部署，則 API 會傳回錯誤。

```
aws iot create-thing-group --thing-group-name MyGreengrassCoreGroup
```

如果要求成功，回應看起來會類似下列範例：

```
{
  "thingGroupName": "MyGreengrassCoreGroup",
  "thingGroupArn": "arn:aws:iot:us-
west-2:123456789012:thinggroup/MyGreengrassCoreGroup",
  "thingGroupId": "4df721e1-ff9f-4f97-92dd-02db4e3f03aa"
}
```

b. 將佈建的 Greengrass 核心新增至您的物件群組。使用這些參數執行下列命令：

- 替換*MyGreengrassCore*為您佈建的 Greengrass 核心的名稱。
- *MyGreengrassCoreGroup*以物件群組的名稱取代。

```
aws iot add-thing-to-thing-group --thing-name MyGreengrassCore --thing-group-
name MyGreengrassCoreGroup
```

如果請求成功，命令沒有任何輸出。

7. 建立名為的檔案deployment.json，然後將下列 JSON 物件複製到檔案中。以要*########## AWS IoT##### ARN ## Tar* n。子部署目標只能是物群組。物件群組 ARN 具有下列格式：

- 物件群組 — arn:aws:iot:*region*:*account-id*:thinggroup/*thingGroupName*

```
{
  "targetArn": "targetArn"
}
```

8. 再次執行下列命令以取得原始部署的詳細資料。這些詳細資料包括中繼資料、元件和工作組態。使用來*## ID ####* ID。Step 1您可以使用此部署規劃來設定子部署，並視需要進行變更。

```
aws greengrassv2 get-deployment --deployment-id deploymentId
```

回應包含部署的詳細資料。將以下任何鍵值對從get-deployment命令的響應複製到deployment.json。您可以變更子部署的這些值。若要取得有關此指令的詳細資訊，請參閱〈〉GetDeployment。

- components— 部署的元件。若要解除安裝元件，請將其從此物件中移除。
- deploymentName— 部署的名稱。
- deploymentPolicies— 部署的原則。
- iotJobConfiguration— 部署的工作組態。
- parentTargetArn— 父系部署的目標。
- tags— 部署的標籤。

9. 執行下列命令以建立子部署。deployment.json將*#####*取代為子部署的名稱。

```
aws greengrassv2 create-deployment --deployment-name subdeploymentName --cli-input-
json file://deployment.json
```

回應包括識別deploymentId此子部署的。您可以使用部署 ID 來檢查部署的狀態。如需詳細資訊，請參閱檢查部署狀態。

10. 如果子部署成功，您可以使用其組態來修訂父項部署。複製您deployment.json在上一步中使用的。以父系部署的 ARN 取代 JSON 檔案targetArn中的，並執行下列命令，以使用此新組態建立父系部署。

> **Note**
>
> 如果您建立父系叢集的新部署修訂版本，它會取代該父系部署的所有部署修訂版本和子部署。如需詳細資訊，請參閱版本修訂部署。

```
aws greengrassv2 create-deployment --cli-input-json file://deployment.json
```

回應包括識別deploymentId此部署的。您可以使用部署 ID 來檢查部署的狀態。如需更多詳細資訊，請參閱 檢查部署狀態。

修訂部署

每個目標物件或物群組一次可以有一個作用中部署。當您為已有部署的目標建立部署時，新部署中的軟體元件會取代先前部署中的軟體元件。如果新部署未定義先前部署所定義的元件，則 AWS IoT Greengrass Core 軟體會從目標核心裝置移除該元件。您可以修訂現有部署，這樣就不會將核心裝置上執行的元件從先前的部署移除至目標。

若要修訂部署，您可以建立從先前部署中存在的相同元件和規劃開始的部署。您可以使用CreateDeployment作業，這與建立部署時所使用的作業相同。

若要修訂部署 (AWS CLI)

1. 執行下列命令以列出部署目標的部署。將目標 *targetArn* 取代為目標AWS IoT物件或物群組的 ARN。

   ```
   aws greengrassv2 list-deployments --target-arn targetArn
   ```

   回應包含一份包含目標最新部署的清單。複製來deploymentId自響應以在下一個步驟中使用。

   > **Note**
   >
   > 您也可以修訂目標的最新修訂版以外的部署。指定--history-filter ALL引數以列出目標的所有部署。然後，複製您要修訂的部署 ID。

2. 執行下列命令以取得部署的詳細資料。這些詳細資料包括中繼資料、元件和工作組態。使用上一個步驟的 ID 取代*##* ID。

   ```
   aws greengrassv2 get-deployment --deployment-id deploymentId
   ```

   回應包含部署的詳細資料。

3. 建立名為 deployment.json 的文件，並將前一個命令的回應複製到檔案中。
4. 從 JSON 物件的 deployment.json 中移除以下鍵值組：

   - deploymentId

- revisionId
- iotJobId
- iotJobArn
- creationTimestamp
- isLatestForTarget
- deploymentStatus

作CreateDeployment業需要具有下列結構的有效負載。

```
{
  "targetArn": "String",
  "components": Map of components,
  "deploymentPolicies": DeploymentPolicies,
  "iotJobConfiguration": DeploymentIoTJobConfiguration,
  "tags": Map of tags
}
```

5. 在 deployment.json 中，執行下列任何一項：
   - 變更部署名稱 (deploymentName)。
   - 變更部署的元件 (components)。
   - 變更部署的原則 (deploymentPolicies)。
   - 變更部署的工作組態 (iotJobConfiguration)。
   - 變更部署的標籤 (tags)。

   如需如何定義這些部署詳細資訊的詳細資訊，請參閱建立部署。
6. 執行下列命令以從中建立部署deployment.json。

```
aws greengrassv2 create-deployment --cli-input-json file://deployment.json
```

回應包括識別deploymentId此部署的。您可以使用部署 ID 來檢查部署的狀態。如需更多詳細資訊，請參閱 檢查部署狀態。

取消部署

您可以取消作用中部署，以防止其軟體元件安裝在AWS IoT Greengrass核心裝置上。如果您取消以物件群組為目標的部署，您新增至群組的核心裝置將不會收到該連續部署。如果核心裝置已執行部署，則取消部署時不會變更該裝置上的元件。您必須建立新部署或修訂部署，才能修改在接收已取消部署的核心裝置上執行的元件。

取消部署的步驟 (AWS CLI)

1. 執行下列命令以尋找目標之最新部署修訂版本的識別碼。最新修訂版是目標唯一可以處於作用中狀態的部署，因為先前的部署會在您建立新修訂時取消。以目 *targetArn* 項目或項目群組的 ARN 取代目標項目AWS IoT或項目群組的 ARN。

   ```
   aws greengrassv2 list-deployments --target-arn targetArn
   ```

   回應包含一份包含目標最新部署的清單。複製回應，以便在下一個步驟中使用。deploymentId
2. 執行下列命令以取消部署。以上一個步驟的 *ID ####* ID。

   ```
   aws greengrassv2 cancel-deployment --deployment-id deploymentId
   ```

   如果項目成功，部署狀態會變更為CANCELED。

檢查部署狀態

您可以檢查已在中建立的部署的狀態AWS IoT Greengrass。您也可以檢查將部署部署至每個核心裝置的AWS IoT工作狀態。部署處於作用中狀態時，AWS IoT工作的狀態為IN_PROGRESS。建立部署的新修訂版本後，先前修訂的AWS IoT工作狀態會變更為CANCELLED。

主題

- 檢查部署狀態
- 檢查裝置部署狀態

檢查部署狀態

您可以檢查透過部署的目標或其 ID 識別的部署狀態。

若要依目標檢查部署狀態 (AWS CLI)

- 執行以下命令來擷取目標的最新部署的狀態。將 *TargetArn* 取代為部署目標的AWS IoT物件或物件群組的 Amazon Resource Name (ARN)。

```
aws greengrassv2 list-deployments --target-arn targetArn
```

回應包含一份包含目標最新部署的清單。此部署物件包含部署的狀態。

若要依 ID (AWS CLI) 檢查部署狀態

- 執行以下命令來擷取部署的狀態。將部 *deploymentId* ID 取代為要查詢的部署 ID。

```
aws greengrassv2 get-deployment --deployment-id deploymentId
```

回應包含部署的狀態。

檢查裝置部署狀態

您可以檢查套用至個別核心裝置的部署任務的狀態。您也可以檢查物件群組部署的部署工作狀態。

若要檢查核心裝置的部署工作狀態 (AWS CLI)

- 執行以下命令來擷取核心裝置的所有部署任務的狀態。以要查詢的核心裝置名稱取*coreDeviceName*代。

```
aws greengrassv2 list-effective-deployments --core-device-thing-name coreDeviceName
```

回應包含核心裝置的部署工作清單。您可以透過工作的`deploymentId`或來識別部署工作`targetArn`。每個部署任務都會包含核心裝置上任務的狀態。

若要檢查物件群組的部署狀態 (AWS CLI)

1. 執行以下命令來擷取現有部署的 ID。*####### ARN #### Arn#*

```
aws greengrassv2 list-deployments --target-arn targetArn
```

回應包含一份包含目標最新部署的清單。複製回應以在下一個步驟中使用的回應。deploymentId

> **Note**
>
> 您也可以列出目標的最新部署以外的部署。指定--history-filter ALL引數以列出目標的所有部署。然後，複製您要檢查其狀態的部署 ID。

2. 執行以下命令來取得部署的詳細資訊。使用上一個步驟的 *ID ####* ID。

```
aws greengrassv2 get-deployment --deployment-id deploymentId
```

回應會包含部署的相關資訊。iotJobId從響應複製以在以下步驟中使用。

3. 執行下列命令，說明核心裝置針對部署的工作執行。使用上一個步驟中的作業 ID 以及要檢查其狀態的核心裝置取*iotJobId*代並*coreDeviceThing#*名。

```
aws iot describe-job-execution --job-id iotJobId --thing-name coreDeviceThingName
```

回應包含核心裝置的部署工作執行狀態，以及狀態的詳細資料。包detailsMap含下列資訊：

- detailed-deployment-status— 部署結果狀態，可以是下列其中一個數值：
  - SUCCESSFUL— 部署成功。
  - FAILED_NO_STATE_CHANGE— 核心裝置準備套用部署時，部署失敗。
  - FAILED_ROLLBACK_NOT_REQUESTED— 部署失敗，且部署未指定回復至先前的運作中設定，因此核心裝置可能無法正常運作。
  - FAILED_ROLLBACK_COMPLETE— 部署失敗，核心裝置成功還原為先前的工作組態。
  - FAILED_UNABLE_TO_ROLLBACK— 部署失敗，核心裝置無法回復到先前的可運作組態，因此核心裝置可能無法正常運作。

  如果部署失敗，請檢查deployment-failure-cause值和核心裝置的記錄檔，以識別問題。如需存取核心裝置的記錄檔的詳細資訊，請參閱監控AWS IoT Greengrass日誌。
- deployment-failure-cause— 錯誤訊息，提供有關工作執行失敗原因的其他詳細資訊。

回應看起來類似以下範例。

```
{
  "execution": {
    "jobId": "2cc2698a-5175-48bb-adf2-1dd345606ebd",
    "status": "FAILED",
    "statusDetails": {
      "detailsMap": {
        "deployment-failure-cause": "No local or cloud component version
 satisfies the requirements. Check whether the version constraints conflict
 and that the component exists in your AWS ## with a version that matches the
 version constraints. If the version constraints conflict, revise deployments
 to resolve the conflict. Component com.example.HelloWorld version constraints:
 LOCAL_DEPLOYMENT requires =1.0.0, thinggroup/MyGreengrassCoreGroup requires
 =1.0.1.",
        "detailed-deployment-status": "FAILED_NO_STATE_CHANGE"
      }
    },
    "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
    "queuedAt": "2022-02-15T14:45:53.098000-08:00",
    "startedAt": "2022-02-15T14:46:05.670000-08:00",
    "lastUpdatedAt": "2022-02-15T14:46:20.892000-08:00",
    "executionNumber": 1,
    "versionNumber": 3
  }
}
```

AWS IoT Greengrass 中的日誌記錄和監控

監控是維護 AWS IoT Greengrass 及您 AWS 解決方案可靠性、可用性和效能的重要部分。您應該收集 AWS 解決方案各方面的監控資料，以便在發生多點失敗的情況下，更輕鬆地偵錯。在開始監控 AWS IoT Greengrass 之前，應先建立監控計畫，為下列問題提供解答：

- 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？

主題

監控工具

AWS 提供可讓您用來監控 AWS IoT Greengrass 的工具。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監視工具來監視AWS IoT Greengrass和報告問題：

- Amazon CloudWatch 日誌 — 監控、存放和存取來自AWS CloudTrail或其他來源的日誌檔。如需詳細資訊，請參閱 Amazon CloudWatch 使用者指南中的監控日誌檔。
- AWS CloudTrail記錄監控 — 在帳戶之間共用記錄檔、即時監控記 CloudTrail 錄檔案，方法是將記錄檔傳送至 CloudWatch 記錄檔、使用 Java 撰寫記錄處理應用程式，以及驗證記錄檔在傳送之後是否未變更 CloudTrail。若要取得更多資訊，請參閱《使用指南》中的〈AWS CloudTrail使用 CloudTrail 記錄檔〉。

- Greengrass 系統健康狀態遙測 — 訂閱以接收 Greengrass 核心傳送的遙測資料。如需詳細資訊，請參閱 the section called “收集系統健康狀態遙測資”。
- 裝置運作狀態通知使用 Amazon 建立事件， EventBridge 以接收有關部署和元件的狀態更新。如需詳細資訊，請參閱 取得部署和元件健康狀態通知。
- 車隊狀態服務 — 使用車隊狀態 API 操作來檢查核心設備及其 Greengrass 組件的狀態。您也可以在主控台中檢視叢集狀態資AWS IoT Greengrass訊。如需更多詳細資訊，請參閱 檢查核心設備狀態。

監控AWS IoT Greengrass日誌

AWS IoT Greengrass 由雲端服務和 AWS IoT Greengrass Core 軟體組成。AWS IoT Greengrass核心軟體可以將日誌寫入 Amazon CloudWatch 日誌和核心裝置的本機檔案系統。在核心裝置上執行的 Greengrass 元件也可以將記錄寫入記錄檔和本機 CloudWatch 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 日誌項目皆含有時間戳記、日誌等級和事件資料。

根據預設，AWS IoT GreengrassCore 軟體只會將記錄寫入本機檔案系統。您可以即時檢視檔案系統記錄，以便偵錯您開發和部署的 Greengrass 元件。您還可以配置核心設備將日誌寫入日 CloudWatch 誌，以便在不訪問本地文件系統的情況下對核心設備進行故障排除。如需詳細資訊，請參閱 啟用記錄 CloudWatch 檔的記錄。

主題

- 存取檔案系統記錄
- 存取 CloudWatch 記錄
- 存取系統服務記錄
- 啟用記錄 CloudWatch 檔的記錄
- 設定 AWS IoT Greengrass 的日誌記錄
- AWS CloudTrail 日誌

存取檔案系統記錄

AWS IoT Greengrass核心軟件將日誌存儲在核心設備上的文件*/greengrass/v2*/logs夾中，其中*/greengrass/v2*是AWS IoT Greengrass根文件夾的路徑。log 資料夾具有以下結構。

```
/greengrass/v2
### logs
    ### greengrass.log
    ### greengrass_2021_09_14_15_0.log
```

```
### ComponentName.log
### ComponentName_2021_09_14_15_0.log
### main.log
```

- greengrass.log— AWS IoT Greengrass 核心軟體記錄檔。使用此記錄檔可檢視有關元件和部署的即時資訊。此日誌文件包括 Greengrass 核心，這是核心軟件的AWS IoT Greengrass核心，和插件組件，如日誌管理器和秘密管理器日誌。
- *ComponentName*.log-綠色組件日誌文件。使用元件記錄檔可檢視核心裝置上執行之 Greengrass 元件的即時資訊。一般元件和 Lambda 元件會將標準輸出 (標準輸出) 和標準錯誤 (標準錯誤) 寫入這些記錄檔。
- main.log— 處理元件生命週期之main服務的記錄檔。此記錄檔永遠是空的。

如需外掛程式、一般元件和 Lambda 元件之間差異的詳細資訊，請參閱元件類型。

使用檔案系統日誌時有下列考量：

- 根使用者權限

  您必須具有在檔案系統上讀取 AWS IoT Greengrass 日誌的根許可。

- 日誌文件旋轉

  AWS IoT Greengrass核心軟體每小時或超過檔案大小限制時，都會旋轉記錄檔。旋轉的記錄檔在其檔案名稱中包含時間戳記。例如，旋轉後的 AWS IoT Greengrass Core 軟體記錄檔可能會命名為greengrass_2021_09_14_15_0.log。預設的檔案大小限制為 1,024 KB (1 MB)。您可以在 Greengrass 核組件上配置文件大小限制。

- 記錄檔刪除

  當AWS IoT Greengrass核心軟體記錄檔或 Greengrass 元件記錄檔 (包括旋轉的記錄檔) 的大小超過磁碟空間限制時，AWS IoT Greengrass核心軟體會清除較早的記錄檔。AWS IoT Greengrass核心軟體記錄檔和每個元件記錄檔的預設磁碟空間限制為 10,240 KB (10 MB)。您可以在 Greengrass 核心元件或記錄檔管理員元件上設定核AWS IoT Greengrass心軟體記錄檔磁碟空間限制。您可以在記錄檔管理員元件上設定每個元件的記錄磁碟空間限制。

若要檢視AWS IoT Greengrass核心軟體記錄檔

- 執行下列命令以即時檢視記錄檔。以AWS IoT Greengrass根資料夾的路徑取*/greengrass/v2*代。

Linux or Unix

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

Windows Command Prompt (CMD)

```
type C:\greengrass\v2\logs\com.example.HelloWorld.log
```

該type命令將文件的內容寫入終端。多次執行此命令以觀察檔案中的變更。

PowerShell

```
gc C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
```

若要檢視元件的記錄檔

- 執行下列命令以即時檢視記錄檔。以AWS IoT Greengrass根資料夾的路徑取代*/greengrass/v2*或 *C:\greengrass\v2*，並取代 *com.example# HelloWorld*與組件的名稱。

  Linux or Unix

  ```
  sudo tail -f /greengrass/v2/logs/com.example.HelloWorld.log
  ```

  PowerShell

  ```
  gc C:\greengrass\v2\logs\com.example.HelloWorld.log -Tail 10 -Wait
  ```

您也可以使用 Greengrass CLI 的logs命令來分析核心裝置上的 Greengrass 記錄。若要使用此logs命令，您必須設定 Greengrass 核來輸出 JSON 格式的記錄檔。如需詳細資訊，請參閱 綠色命令行界面 及 日誌。

存取 CloudWatch 記錄

您可以部署記錄管理員元件，將核心裝置設定為寫入 CloudWatch 記錄檔。如需詳細資訊，請參閱 啟用記錄 CloudWatch 檔的記錄。然後，您可以在 Amazon CloudWatch 主控台的「日誌」頁面或使用 CloudWatch 日誌 API 檢視日誌。

記錄群組名稱

```
/aws/greengrass/componentType/region/componentName
```

記錄群組名稱使用下列變數：

- componentType— 元件的類型，可以是下列其中一種：
  - GreengrassSystemComponent— 此記錄群組包含核心和外掛程式元件的記錄，這些元件與 Greengrass 核心在相同的 JVM 中執行。該組件是 Greengr ass 核的一部分。
  - UserComponent— 此日誌群組包含一般元件、Lambda 元件和裝置上其他應用程式的記錄。該組件不是 Greengrass 核的一部分。

  如需詳細資訊，請參閱 元件類型。
- region— 核心設備使用的AWS區域。
- componentName— 元件的名稱。對於系統記錄檔，此值為System。

記錄串流名稱

```
/date/thing/thingName
```

記錄資料流名稱會使用下列變數：

- date— 記錄的日期，例如2020/12/15。記錄管理員元件會使用此yyyy/MM/dd格式。
- thingName— 核心裝置的名稱。

> **Note**
>
> 如果物件名稱包含冒號 (:)，則記錄管理員會以加號 (+) 取代冒號。

當您使用記錄檔管理員元件寫入 CloudWatch 記錄檔時，需要考量下列事項：

- 記錄延遲

> **Note**
>
> 我們建議您升級至記錄檔管理員 2.3.0 版，以減少輪換和使用中記錄檔的記錄延遲。當您升級至記錄檔管理員 2.3.0 時，我們建議您也升級至 Greengrass 核心 2.9.1。

記錄管理員元件 2.2.8 版 (及更早版本) 只會從輪換的記錄檔處理和上傳記錄檔。根據預設，AWS IoT Greengrass核心軟體會每小時或在 1,024 KB 之後輪換記錄檔。因此，記錄管理員元件只會在AWS IoT Greengrass核心軟體或 Greengrass 元件寫入超過 1,024 KB 的記錄檔之後上傳記錄檔。您可以設定較低的記錄檔大小限制，使記錄檔更頻繁地旋轉。這會導致記錄檔管理員元件更頻繁地將記錄檔上傳至 CloudWatch 記錄檔。

記錄管理員元件 2.3.0 版 (及更新版本) 會處理並上傳所有記錄檔。當您寫入新的記錄檔時，記錄檔管理員 2.3.0 版 (及更新版本) 會處理並直接上傳該使用中的記錄檔，而不是等待輪換。這意味著您可以在 5 分鐘或更短的時間內查看新日誌。

記錄管理員元件會定期上傳新的記錄檔。根據預設，記錄管理員元件會每 5 分鐘上傳一次新的記錄檔。您可以設定較低的上傳間隔，如此一來，記錄管理員元件會將記錄檔上傳至 CloudWatch 記錄檔的頻率，方法是設定`periodicUploadIntervalSec`. 如需如何設定此週期間隔的相關資訊，請參閱組態。

日誌可以從相同的 Greengrass 文件系統以近乎即時的方式上傳。如果您需要即時觀察記錄，請考慮使用檔案系統記錄檔。

> **Note**
>
> 如果您使用不同的檔案系統來寫入記錄檔，記錄檔管理員會還原為記錄檔管理員元件 2.2.8 及更早版本中的行為。如需存取檔案系統記錄檔的相關資訊，請參閱存取檔案系統記錄。

- 時鐘偏斜

  記錄管理員元件會使用標準簽章第 4 版簽署程序來建立對 CloudWatch 記錄的 API 要求。如果核心裝置上的系統時間不同步超過 15 分鐘，則 CloudWatch Logs 會拒絕要求。如需詳細資訊，請參閱 AWS 一般參考 中的 Signature 第 4 版簽署程序。

存取系統服務記錄

如果您將 AWS IoT GreengrassCore 軟體設定為系統服務，則可以檢視系統服務記錄檔來疑難排解問題，例如軟體無法啟動。

若要檢視系統服務記錄檔 (CLI)

1. 執行下列命令以檢視AWS IoT Greengrass核心軟體系統服務記錄檔。

Linux or Unix (systemd)

```
sudo journalctl -u greengrass.service
```

Windows Command Prompt (CMD)

```
type C:\greengrass\v2\logs\greengrass.wrapper.log
```

PowerShell

```
gc C:\greengrass\v2\logs\greengrass.wrapper.log
```

2. 在 Windows 設備上，AWS IoT Greengrass核心軟件為系統服務錯誤創建一個單獨的日誌文件。執行下列命令以檢視系統服務錯誤記錄檔。

   Windows Command Prompt (CMD)

   ```
   type C:\greengrass\v2\logs\greengrass.err.log
   ```

   PowerShell

   ```
   gc C:\greengrass\v2\logs\greengrass.err.log
   ```

在 Windows 裝置上，您也可以使用事件檢視器應用程式來檢視系統服務記錄檔。

若要檢視 Windows 服務記錄檔 (事件檢視器)

1. 開啟事件檢視器應用程式。
2. 選取視窗記錄以展開它。
3. 選擇應用程式以檢視應用程式服務記錄
4. 尋找並開啟「來源」為的事件記錄檔greengrass。

啟用記錄 CloudWatch 檔的記錄

您可以部署記錄管理員元件，以設定核心裝置將記錄寫入 CloudWatch 記錄檔。您可以啟用AWS IoT Greengrass核心軟體 CloudWatch 記錄檔的記錄檔，也可以為特定 Greengrass 元件啟用 CloudWatch 記錄檔。

> **Note**
>
> Greengrass 核心裝置的權杖交換角色必須允許核心裝置寫入 CloudWatch 記錄，如下列範例 IAM 政策所示。如果您使用自動資源佈建來安裝 AWS IoT Greengrass Core 軟體，則您的核心裝置具有這些權限。
>
> ```
> {
>   "Version": "2012-10-17",
>   "Statement": [
>     {
>       "Action": [
>         "logs:CreateLogGroup",
>         "logs:CreateLogStream",
>         "logs:PutLogEvents",
>         "logs:DescribeLogStreams"
>       ],
>       "Effect": "Allow",
>       "Resource": "arn:aws:logs:*:*:*"
>     }
>   ]
> }
> ```

若要設定核心裝置將AWS IoT Greengrass核心軟體記錄檔寫入 CloudWatch 記錄檔，`true`請建立一個部署，以指定為`aws.greengrass.LogManager`元件設`uploadToCloudWatch`定的組態更新。AWS IoT Greengrass核心軟件日誌包括 Greengrass 核心和插件組件的日誌。

```
{
  "logsUploaderConfiguration": {
    "systemLogsConfiguration": {
      "uploadToCloudWatch": "true"
    }
  }
}
```

若要設定核心裝置將 Greengrass 元件的記錄檔寫入記錄檔，請建立指定組態更新的部署，以便將元件新增至元件 CloudWatch 記錄組態清單。當您將元件新增至此清單時，記錄管理員元件會將其記錄寫入 CloudWatch 記錄檔。元件記錄包括一般元件和 Lambda 元件的記錄。

```
{
  "logsUploaderConfiguration": {
    "componentLogsConfigurationMap": {
      "com.example.HelloWorld": {

      }
    }
  }
}
```

部署記錄管理員元件時，您也可以設定磁碟空間限制，以及核心裝置是否在將記錄檔寫入記錄檔後刪除 CloudWatch 記錄檔。如需詳細資訊，請參閱 設定 AWS IoT Greengrass 的日誌記錄。

設定 AWS IoT Greengrass 的日誌記錄

您可以設定下列選項來自訂 Greengrass 核心裝置的記錄。若要設定這些選項，請建立一個部署，以指定 Greengrass 核心或記錄檔管理員元件的組態更新。

- 將記錄寫入記 CloudWatch 錄檔

  若要遠端疑難排解核心裝置，您可以設定核心裝置將AWS IoT Greengrass核心軟體和元件記錄寫入 CloudWatch 記錄檔。若要這麼做，請部署和設定記錄檔管理員元件。如需詳細資訊，請參閱 啟用記錄 CloudWatch 檔的記錄。
- 刪除上傳的記錄檔

  若要減少磁碟空間使用量，您可以將核心裝置設定為在將記錄檔寫入記錄檔後刪除 CloudWatch 記錄檔。如需詳細資訊，請參閱記錄管理員元件的`deleteLogFileAfterCloudUpload`參數，您可以為AWS IoT Greengrass核心軟體記錄檔和元件記錄檔指定此參數。
- 記錄磁碟空間限制

  若要限制磁碟空間使用量，您可以在核心裝置上設定每個記錄檔的最大磁碟空間，包括其旋轉的記錄檔。例如，您可以為`greengrass.log`和旋轉`greengrass.log`檔案設定合併磁碟空間上限。如需詳細資訊，請參閱 Greengrass 核心元件的`logging.totalLogsSizeKB`參數和記錄檔管理員元件的參數，您可以為AWS IoT Greengrass核心軟體記錄檔和元件記錄檔指定此`diskSpaceLimit`參數。

- 記錄檔大小限制

  您可以設定每個記錄檔的檔案大小上限。記錄檔超過此檔案大小限制之後，AWS IoT GreengrassCore 軟體會建立新的記錄檔。記錄管理員元件 2.28 版 (及更早版本) 只會將輪替的記錄檔寫入 CloudWatch 記錄檔，因此您可以指定較低的檔案大小限制，以便更頻繁地將記錄寫入 CloudWatch 記錄檔。記錄檔管理員元件 2.3.0 版 (及更新版本) 會處理並上傳所有記錄檔，而不是等待它們輪換。如需詳細資訊，請參閱 Greengrass 核心元件的記錄檔大小限制參數 ()。`logging.fileSizeKB`

- 最低記錄層級

  您可以設定 Greengrass 核心元件寫入檔案系統記錄的最低記錄層級。例如，您可以指定DEBUG層級記錄檔以協助進行疑難排解，也可以指定ERROR層級記錄檔以減少核心裝置建立的記錄數量。如需詳細資訊，請參閱 Greengrass 核心元件的記錄層級參數 ()。`logging.level`

  您也可以設定記錄檔管理員元件寫入記錄檔的最低 CloudWatch 記錄層級。例如，您可以指定較高的記錄層級以降低記錄成本。如需詳細資訊，請參閱記錄管理員元件的`minimumLogLevel`參數，您可以為AWS IoT Greengrass核心軟體記錄檔和元件記錄檔指定此參數。

- 檢查記錄檔寫入記錄檔的間隔 CloudWatch

  若要增加或減少記錄管理員元件將記錄寫入 CloudWatch 記錄檔的頻率，您可以設定檢查要寫入的新記錄檔的間隔。例如，您可以指定較小的間隔，以便在記錄檔中檢視 CloudWatch 記錄檔的時間比預設的 5 分鐘間隔要快。您可以指定較高的間隔來降低成本，因為記錄管理員元件會將記錄檔批次處理成較少的要求。如需詳細資訊，請參閱記錄管理員元件的上傳間隔參數 (`periodicUploadIntervalSec`)。

- 日誌格式

  您可以選擇 AWS IoT Greengrass Core 軟體是以文字還是 JSON 格式寫入記錄。如果您讀取記錄檔，請選擇文字格式；如果您使用應用程式讀取或剖析記錄，請選擇 JSON 格式。如需詳細資訊，請參閱 Greengrass 核心元件的記錄格式參數 ()。`logging.format`

- 本機檔案系統記錄資料夾

  您可以將 logs 文件夾從更改*`/greengrass/v2`*`/logs`為核心設備上的另一個文件夾。如需詳細資訊，請參閱 Greengrass 核心元件的輸出目錄參數 ()。`logging.outputDirectory`

AWS CloudTrail 日誌

AWS IoT Greengrass與整合AWS CloudTrail，提供使用者、角色或AWS 服務中所採取之動作記錄的服務AWS IoT Greengrass。如需更多詳細資訊，請參閱 使用記錄 AWS IoT Greengrass V2 API 呼叫 AWS CloudTrail。

使用記錄 AWS IoT Greengrass V2 API 呼叫 AWS CloudTrail

AWS IoT Greengrass V2 與 (提供中的使用者 AWS CloudTrail、角色或服務所採取的動作記錄) 的 AWS 服務整合 AWS IoT Greengrass Version 2。 CloudTrail 擷取 AWS IoT Greengrass 作為事件的所有 API 呼叫。擷取的呼叫包括來自 AWS IoT Greengrass 主控台的呼叫和 AWS IoT Greengrass API 作業的程式碼呼叫。

如果您建立追蹤，您可以啟用 CloudTrail 事件持續傳遞至 S3 儲存貯體，包括 AWS IoT Greengrass. 如果您未設定追蹤，您仍然可以在 [事件歷程記錄] 中檢視 CloudTrail 主控台中最近的事件。使用收集的資訊 CloudTrail，您可以判斷提出的要求 AWS IoT Greengrass、提出要求的 IP 位址、提出要求的人員、提出要求的時間，以及其他詳細資訊。

若要取得有關的更多資訊 CloudTrail，請參閱AWS CloudTrail 使用者指南。

AWS IoT Greengrass V2 中的資訊 CloudTrail

CloudTrail 在您創建帳戶 AWS 帳戶 時啟用。當活動發生在中時 AWS IoT Greengrass，該活動會與事件歷史記錄中的其他 AWS 服務 CloudTrail 事件一起記錄在事件中。您可以檢視、搜尋和下載 AWS 帳戶的最新事件。如需詳細資訊，請參閱使用 CloudTrail 事件歷程記錄檢視事件。

對於您的事件的持續記錄 AWS 帳戶，包括事件 AWS IoT Greengrass，請創建一個跟踪。追蹤可 CloudTrail 將日誌檔傳遞至 S3 儲存貯體。根據預設，當您在主控台中建立追蹤時，追蹤會套用至所有 AWS 區域項目。追蹤記錄來自 AWS 分割區中所有區域的事件，並將日誌檔傳送到您指定的 S3 儲存貯體。此外，您還可以設定其他 AWS 服務，以進一步分析 CloudTrail 記錄中收集的事件資料並採取行動。如需詳細資訊，請參閱下列內容：

- 建立追蹤的概觀
- CloudTrail 支援的服務與整合
- 設定 Amazon SNS 通知 CloudTrail
- 從多個區域接收 CloudTrail 日誌文件並從多個帳戶接收 CloudTrail 日誌文件

所有 AWS IoT Greengrass V2 動作均由「API 參考」記錄 CloudTrail 並記錄在「AWS IoT Greengrass V2 API 參考」中。例如，呼叫`CreateDeployment`和`CancelDeployment`動作會`CreateComponentVersion`在 CloudTrail 記錄檔中產生項目。

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：

- 要求是使用根使用者登入資料還是 AWS Identity and Access Management (IAM) 使用者登入資料提出。
- 提出該請求時，是否使用了特定角色或聯合身分使用者的暫時安全憑證。
- 請求是否由其他 AWS 服務提出。

如需詳細資訊，請參閱 CloudTrail userIdentity 元素。

AWS IoT Greengrass 資料事件 CloudTrail

資料事件提供在資源上或在資源中執行之資源作業的相關資訊 (例如，取得元件版本或部署的組態)。這些也稱為資料平面操作。資料事件通常是大量資料的活動。依預設， CloudTrail 不會記錄資料事件。CloudTrail 事件歷史記錄不會記錄數據事件。

資料事件需支付額外的費用。如需有關 CloudTrail 定價的詳細資訊，請參閱AWS CloudTrail 定價。

您可以使用 CloudTrail 主控台或 CloudTrail API 作業記錄 AWS IoT Greengrass 資源類型的資料事件。 AWS CLI此段落中的表格顯示可用的資源類型 AWS IoT Greengrass。

- 若要使用 CloudTrail 主控台記錄資料事件，請建立追蹤或事件資料存放區以記錄資料事件，或更新現有追蹤或事件資料存放區以記錄資料事件。
  1. 選擇 [資料事件] 以記錄資料事件。
  2. 從 [資料事件類型] 清單中，選擇您要記錄其資料事件的資源類型。
  3. 選擇您要使用的記錄選取器範本。您可以記錄資源類型的所有資料事件、記錄所有`readOnly`事件、記錄所有`writeOnly`事件，或建立自訂記錄選取器範本以篩選`readOnly`、`eventName`、和`resources.ARN`欄位。
- 若要使用記錄資料事件 AWS CLI，請配置`--advanced-event-selectors`參數以將`eventCategory`欄位設定為等於，`Data`且`resources.type`欄位等於資源類型值 (請參閱表格)。您可以加入條件以篩選`readOnly`、`eventName`、和`resources.ARN`欄位的值。
  - 若要設定追蹤以記錄資料事件，請執行put-event-selectors命令。如需詳細資訊，請參閱使用 AWS CLI.

- 若要規劃事件資料倉庫以記錄資料事件，請執行create-event-data-store指令以建立新的事件資料倉庫以記錄資料事件，或執行update-event-data-store指令來更新現有的事件資料倉庫。如需詳細資訊，請參閱使用 AWS CLI.

下表列出了資 AWS IoT Greengrass 源類型。[資料事件類型 (主控台)] 欄顯示可從主控台的 [資料事件類型 CloudTrail] 清單中選擇的值。resource .type 值欄會顯示**`resources.type`**值，您可以在使用或 API 設定進階事件選取器時指定這個值。 AWS CLI CloudTrail 記錄到資料 CloudTrail欄中的資料 API 會顯示 CloudTrail針對資源類型記錄的 API 呼叫。

資料事件類型 (主控台)	resources.type 值	記錄到的資料 API CloudTrail
IoT 證書	`AWS::IoT::Certificate`	• VerifyClientDeviceIdentity • VerifyClientDeviceIoTCertificateAssociation
IoT Greengrass 件版本	`AWS::GreengrassV2::ComponentVersion`	• ResolveComponentCandidates
IoT 環境部署	`AWS::GreengrassV2::Deployment`	• GetDeploymentConfiguration
IoT 的事	`AWS::IoT::Thing`	• ListThingGroupsForCoreDevices • PutCertificateAuthorities • VerifyClientDeviceIoTCertificateAssociation

> **Note**
>
> Greengrass 不會記錄訪問被拒絕的事件。

您可以設定進階事件選取器來篩選`eventName`、`readOnly`、和`resources.ARN`欄位，以僅記錄對您很重要的事件。

在上新增篩選器`eventName`以包含或排除特定資料 API。

如需有關這些欄位的詳細資訊，請參閱 AdvancedFieldSelector。

下列範例顯示如何使用設定進階選取器 AWS CLI。用您自己的信息替換*TrailName*和*##*。

Example — 記錄 IoT 事件的數據事件

```
aws cloudtrail put-event-selectors --trail-name TrailName --region region \
--advanced-event-selectors \
'[
    {
            "Name": "Log all thing data events",
            "FieldSelectors": [
                { "Field": "eventCategory", "Equals": ["Data"] },
                { "Field": "resources.type", "Equals": ["AWS::IoT::Thing"] }
            ]
        }
]'
```

Example — 在特定的 IoT 事物 API 上進行過濾

```
aws cloudtrail put-event-selectors --trail-name TrailName --region region \
--advanced-event-selectors \
'[
    {
            "Name": "Log IoT Greengrass PutCertificateAuthorities API calls",
            "FieldSelectors": [
                { "Field": "eventCategory", "Equals": ["Data"] },
                { "Field": "resources.type", "Equals": ["AWS::IoT::Thing"] },
                { "Field": "eventName", "Equals": ["PutCertificateAuthorities"] }
            ]
    }
]'
```

Example -記錄所有 Greengrass 數據事件

```
aws cloudtrail put-event-selectors --trail-name TrailName --region region \
--advanced-event-selectors \
'[
    {
        "Name": "Log all certificate data events",
        "FieldSelectors": [
            {
                "Field": "eventCategory",
```

```
                "Equals": [
                    "Data"
                ]
            },
            {
                "Field": "resources.type",
                "Equals": [
                    "AWS::IoT::Certificate"
                ]
            }
        ]
    },
    {
        "Name": "Log all component version data events",
        "FieldSelectors": [
            {
                "Field": "eventCategory",
                "Equals": [
                    "Data"
                ]
            },
            {
                "Field": "resources.type",
                "Equals": [
                    "AWS::GreengrassV2::ComponentVersion"
                ]
            }
        ]
    },
    {
        "Name": "Log all deployment version",
        "FieldSelectors": [
            {
                "Field": "eventCategory",
                "Equals": [
                    "Data"
                ]
            },
            {
                "Field": "resources.type",
                "Equals": [
                    "AWS::GreengrassV2::Deployment"
                ]
            }
```

```
        ]
    },
    {
        “Name”: “Log all thing data events”,
        “FieldSelectors”: [
            {
                “Field”: “eventCategory”,
                “Equals”: [
                    “Data”
                ]
            },
            {
                “Field”: “resources.type”,
                “Equals”: [
                    “AWS::IoT::Thing”
                ]
            }
        ]
    }
]’
```

AWS IoT Greengrass 管理事件 CloudTrail

管理事件提供有關對您 AWS 帳戶中資源執行之管理作業的相關資訊。這些也稱為控制平面操作。依預設，會 CloudTrail 記錄管理事件。

AWS IoT Greengrass 將所有 AWS IoT Greengrass 控制平面作業記錄為管理事件。如需記 AWS IoT Greengrass 錄到的 AWS IoT Greengrass 控制平面作業清單 CloudTrail，請參閱 AWS IoT Greengrass API 參考資料版本 2。

瞭解 AWS IoT Greengrass V2 記錄檔項目

追蹤是一種組態，可讓事件以日誌檔的形式傳遞至您指定的 S3 儲存貯體。 CloudTrail 記錄檔包含一或多個記錄項目。事件代表來自任何來源的單一請求。它包括請求的動作、動作的日期和時間、請求參數等相關資訊。 CloudTrail 日誌文件不是公共 API 調用的有序堆棧跟踪，因此它們不會以任何特定順序顯示。

下列範例顯示示範`CreateDeployment`動作的 CloudTrail 記錄項目。

```
{
    "eventVersion": "1.08",
    "userIdentity": {
```

```
        "type": "IAMUser",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/Administrator",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "Administrator"
    },
    "eventTime": "2021-01-06T02:38:05Z",
    "eventSource": "greengrass.amazonaws.com",
    "eventName": "CreateDeployment",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "203.0.113.0",
    "userAgent": "aws-cli/2.1.9 Python/3.7.9 Windows/10 exe/AMD64 prompt/off command/
greengrassv2.create-deployment",
    "requestParameters": {
        "deploymentPolicies": {
            "failureHandlingPolicy": "DO_NOTHING",
            "componentUpdatePolicy": {
                "timeoutInSeconds": 60,
                "action": "NOTIFY_COMPONENTS"
            },
            "configurationValidationPolicy": {
                "timeoutInSeconds": 60
            }
        },
        "deploymentName": "Deployment for MyGreengrassCoreGroup",
        "components": {
            "aws.greengrass.Cli": {
                "componentVersion": "2.0.3"
            }
        },
        "iotJobConfiguration": {},
        "targetArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/
MyGreengrassCoreGroup"
    },
    "responseElements": {
        "iotJobArn": "arn:aws:iot:us-west-2:123456789012:job/fdfeba1d-ac6d-44ef-
ab28-54f684ea578d",
        "iotJobId": "fdfeba1d-ac6d-44ef-ab28-54f684ea578d",
        "deploymentId": "4196dddc-0a21-4c54-a985-66a525f6946e"
    },
    "requestID": "311b9529-4aad-42ac-8408-c06c6fec79a9",
    "eventID": "c0f3aa2c-af22-48c1-8161-bad4a2ab1841",
    "readOnly": false,
```

```
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "eventCategory": "Management",
    "recipientAccountId": "123456789012"
}
```

從AWS IoT Greengrass核心裝置收集系統健康狀態遙測資料

系統健康情況遙測資料是診斷資料，可協助您監視 Greengrass 核心裝置上關鍵作業的效能。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。流程工程師等領域專家可以使用這些應用程式來深入瞭解機隊健康狀況。

您可以使用下列方法從 Greengrass 核心裝置收集遙測資料：

- Nucleus 遙測發射器元件 — Greengrass 核心裝置上的核心遙測發射器元件 (`aws.greengrass.telemetry.NucleusEmitter`) 預設會將遙測資料發佈至主題。`$local/greengrass/telemetry`即使您的裝置與雲端的連線能力有限，您也可以使用發佈到本主題的資料在核心裝置上執行本機操作。您也可以選擇將元件設定為將遙測資料發佈到您選擇的 AWS IoT Core MQTT 主題。

  您必須將核心發射器元件部署到核心裝置，才能發佈遙測資料。將遙測資料發佈到本機主題不需要任何費用。但AWS 雲端是，使用 MQTT 主題將資料發佈至的需要AWS IoT Core定價。

  AWS IoT Greengrass提供數個社群元件，協助您使用 InfluxDB 和 Grafana 在核心裝置上本機分析和視覺化遙測資料。這些元件使用來自核輻射器元件的遙測資料。如需詳細資訊，請參閱 In fluxDB 發行者元件的讀我檔案。
- 遙測代理程式 — Greengrass 核心裝置上的遙測代理程式會收集本機遙測資料，並將其發佈到 Amazon， EventBridge 而不需要任何客戶互動。核心裝置會以 EventBridge 最佳方式將遙測資料發佈至。例如，核心裝置可能無法在離線時傳送遙測資料。

  根據預設，所有 Greengrass 核心裝置都會啟用遙測代理程式功能。一旦您設定 Greengrass 核心裝置，就會自動開始接收資料。除了您的數據鏈接成本外，從核心設備傳輸到的數據AWS IoT Core是免費的。這是因為代理程式會發佈至AWS保留主題。不過，視您的使用案例而定，接收或處理資料時可能會產生費用。

> **Note**
>
> Amazon EventBridge 是一種事件匯流排服務，可用來將應用程式與來自各種來源的資料 (例如 Greengrass 核心裝置) 連接。如需詳細資訊，請參閱什麼是 Amazon EventBridge？ 在 Amazon 用 EventBridge 戶指南。

為了確保AWS IoT Greengrass核心軟件正常運行，AWS IoT Greengrass使用數據進行開發和質量改進目的。此功能也有助於提供新的和增強的邊緣功能。 AWS IoT Greengrass最多可保留 7 天的遙測資料。

本節說明如何設定及使用遙測代理程式。如需設定核心遙測發射器元件的相關資訊，請參閱。核遙測發射器

主題

- 遙測指標
- 設定遙測代理程式設
- 訂閱遙測資料 EventBridge

遙測指標

下表說明遙測代理程式所發行的度量。

名稱	描述	
系統		
`SystemMemUsage`	Greengrass 核心裝置 (包括作業系統) 上所有應用程式目前使用的記憶體容量。	
`CpuUsage`	Greengrass 核心裝置 (包括作業系統) 上所有應用程式目前正在使用的 CPU 數量。	
`TotalNumberOfFDs`	Greengrass 核心裝置作業系統所儲存的檔案描述元數目。一	

名稱	描述
	個文件描述符唯一標識一個打開的文件。
Greengrass 核	
NumberOfComponentsRunning	Greengrass 核心裝置上執行的元件數目。
NumberOfComponentsErrored	Greengrass 核心裝置上處於錯誤狀態的元件數目。
NumberOfComponentsInstalled	安裝在 Greengrass 核心裝置上的元件數目。
NumberOfComponentsStarting	在 Greengrass 核心裝置上啟動的元件數目。
NumberOfComponentsNew	Greengrass 核心裝置上新增的元件數目。
NumberOfComponentsStopping	在 Greengrass 核心裝置上停止的元件數目。
NumberOfComponentsFinished	在 Greengrass 核心裝置上完成的元件數目。
NumberOfComponentsBroken	Greengrass 核心裝置上損壞的元件數目。
NumberOfComponentsStateless	Greengrass 核心裝置上無狀態的元件數目。
客戶端設備身份驗證 — 此功能需要 v2.4.0 或更高版本的客戶端設備身份驗證組件。	
VerifyClientDeviceIdentity.Success	驗證用戶端裝置身分是否成功的次數。

名稱	描述	
VerifyClientDeviceIdentity.Failure	驗證用戶端裝置身分識別失敗的次數。	
AuthorizeClientDeviceActions.Success	授權用戶端裝置完成要求動作的次數。	
AuthorizeClientDeviceActions.Failure	用戶端裝置未獲授權完成要求動作的次數。	
GetClientDeviceAuthToken.Success	成功驗證用戶端裝置的次數。	
GetClientDeviceAuthToken.Failure	無法驗證用戶端裝置的次數。	
SubscribeToCertificateUpdates.Success	成功訂閱憑證更新的數目。	
SubscribeToCertificateUpdates.Failure	嘗試訂閱憑證更新失敗的次數。	
ServiceError	用戶端裝置驗證中未處理的內部錯誤數目。	
串流管理員 — 此功能需要 v2.7.0 或更新版本的 Greengrass 核元件。		
BytesAppended	附加到流管理器的數據的字節數。	
BytesUploadedToIoTAnalytics	串流管理員匯出至頻道的資料位元組數AWS IoT Analytics。	
BytesUploadedToKinesis	串流管理員匯出至 Amazon Kinesis 資料串流中串流的資料位元組數。	

名稱	描述	
BytesUploadedToIoTSiteWise	串流管理員匯出至資產屬性的資料位元組數AWS IoT SiteWise。	
BytesUploadedToS3	串流管理員匯出至 Amazon S3 中物件的資料位元組數。	

設定遙測代理程式設

遙測代理程式會使用下列預設設定：

- 遙測代理程式每小時會彙總一次遙測資料。
- 遙測代理程式會每 24 小時發佈一次遙測訊息。

遙測代理程式會使用服務品質 (QoS) 層級為 0 的 MQTT 通訊協定發佈資料，這表示它不會確認傳遞或重試發佈嘗試。遙測訊息與目的地訂閱的其他郵件共用 MQTT 連線。AWS IoT Core

除了您的數據鏈接成本之外，從核心到的數據傳輸AWS IoT Core是免費的。這是因為代理程式會發佈至AWS保留主題。不過，視您的使用案例而定，接收或處理資料時可能會產生費用。

您可以為每個 Greengrass 核心裝置啟用或停用遙測代理程式功能。您也可以設定核心裝置彙總和發佈資料的間隔。若要設定遙測，請在部署 Greengrass 核心元件時自訂遙測組態參數。

訂閱遙測資料 EventBridge

您可以在 Amazon 中建立規則，以 EventBridge 定義如何處理從 Greengrass 核心裝置上的遙測代理程式發佈的遙測資料。 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、儲存事件資訊、採取更正動作或叫用其他事件。

遙測事件

遙測事件使用下列格式。

```
{
  "version": "0",
  "id": "a09d303e-2f6e-3d3c-a693-8e33f4fe3955",
  "detail-type": "Greengrass Telemetry Data",
```

```
  "source": "aws.greengrass",
  "account": "123456789012",
  "time": "2020-11-30T20:45:53Z",
  "region": "us-east-1",
  "resources": [],
  "detail": {
    "ThingName": "MyGreengrassCore",
    "Schema": "2020-07-30",
    "ADP": [
      {
        "TS": 1602186483234,
        "NS": "SystemMetrics",
        "M": [
          {
            "N": "TotalNumberOfFDs",
            "Sum": 6447.0,
            "U": "Count"
          },
          {
            "N": "CpuUsage",
            "Sum": 15.458333333333332,
            "U": "Percent"
          },
          {
            "N": "SystemMemUsage",
            "Sum": 10201.0,
            "U": "Megabytes"
          }
        ]
      },
      {
        "TS": 1602186483234,
        "NS": "GreengrassComponents",
        "M": [
          {
            "N": "NumberOfComponentsStopping",
            "Sum": 0.0,
            "U": "Count"
          },
          {
            "N": "NumberOfComponentsStarting",
            "Sum": 0.0,
            "U": "Count"
          },
```

```
          {
            "N": "NumberOfComponentsBroken",
            "Sum": 0.0,
            "U": "Count"
          },
          {
            "N": "NumberOfComponentsFinished",
            "Sum": 1.0,
            "U": "Count"
          },
          {
            "N": "NumberOfComponentsInstalled",
            "Sum": 0.0,
            "U": "Count"
          },
          {
            "N": "NumberOfComponentsRunning",
            "Sum": 7.0,
            "U": "Count"
          },
          {
            "N": "NumberOfComponentsNew",
            "Sum": 0.0,
            "U": "Count"
          },
          {
            "N": "NumberOfComponentsErrored",
            "Sum": 0.0,
            "U": "Count"
          },
          {
            "N": "NumberOfComponentsStateless",
            "Sum": 0.0,
            "U": "Count"
          }
        ]
      },
      {
        "TS": 1602186483234,
        "NS": "aws.greengrass.ClientDeviceAuth",
        "M": [
          {
            "N": "VerifyClientDeviceIdentity.Success",
            "Sum": 3.0,
```

```
            "U": "Count"
          },
          {
            "N": "VerifyClientDeviceIdentity.Failure",
            "Sum": 1.0,
            "U": "Count"
          },
          {
            "N": "AuthorizeClientDeviceActions.Success",
            "Sum": 20.0,
            "U": "Count"
          },
          {
            "N": "AuthorizeClientDeviceActions.Failure",
            "Sum": 5.0,
            "U": "Count"
          },
          {
            "N": "GetClientDeviceAuthToken.Success",
            "Sum": 5.0,
            "U": "Count"
          },
          {
            "N": "GetClientDeviceAuthToken.Failure",
            "Sum": 2.0,
            "U": "Count"
          },
          {
            "N": "SubscribeToCertificateUpdates.Success",
            "Sum": 10.0,
            "U": "Count"
          },
          {
            "N": "SubscribeToCertificateUpdates.Failure",
            "Sum": 1.0,
            "U": "Count"
          },
          {
            "N": "ServiceError",
            "Sum": 3.0,
            "U": "Count"
          }
        ]
      },
```

```
      {
        "TS": 1602186483234,
        "NS": "aws.greengrass.StreamManager",
        "M": [
          {
            "N": "BytesAppended",
            "Sum": 157745524.0,
            "U": "Bytes"
          },
          {
            "N": "BytesUploadedToIoTAnalytics",
            "Sum": 149012.0,
            "U": "Bytes"
          },
          {
            "N": "BytesUploadedToKinesis",
            "Sum": 12192.0,
            "U": "Bytes"
          },
          {
            "N": "BytesUploadedToIoTSiteWise",
            "Sum": 13321.0,
            "U": "Bytes"
          },
          {
            "N": "BytesUploadedToS3",
            "Sum": 12213.0,
            "U": "Bytes"
          }
        ]
      }
    ]
  }
}
```

ADP陣列包含具有下列屬性的彙總資料點清單：

TS

收集資料的時間戳記。

NS

測量結果命名空間。

M

量度清單。測量結果包含下列特性：

N

指標的名稱

Sum

此遙測事件中度量值的總和。

U

公制值的單位。

如需有關每個量度的詳細資訊，請參閱遙測指標。

建立 EventBridge規則的先決條件

在建立的 EventBridge 規則之前AWS IoT Greengrass，您應該執行下列動作：

- 熟悉中的事件、規則和目標。 EventBridge
- 建立並設定 EventBridge 規則呼叫的目標。規則可以叫用許多類型的目標，例如 Amazon Kinesis 串流、AWS Lambda函數、Amazon SNS 主題和 Amazon SQS 佇列。

  您的 EventBridge 規則和關聯的目標必須AWS 區域位於您建立 Greengrass 資源的位置。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，請參閱什麼是 Amazon EventBridge？ 並在 Amazon 用 EventBridge 戶指南EventBridge中開始使用 Amazon。

建立事件規則以取得遙測資料 (主控台)

使用下列步驟來建立AWS Management Console接收 Greengrass 核心裝置所發佈之遙測資料的 EventBridge 規則。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 使用者指南中的建立可從AWS資源觸發事件的 EventBridge規則。

1. 開啟 Amazon 主 EventBridge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. 在 Define pattern (定義模式)，設定規則模式。

   a. 選擇 Event pattern (事件模式)。

   b. 選擇Pre-defined pattern by service (依服務預先定義模式)。

   c. 在 Service provider (服務提供者)，選擇 AWS。

   d. 在 Service Name (服務名稱)，選擇 Greengrass。

   e. 針對 [事件類型]，選取 [Greengrass 遙測資料]。

4. 在 Select event bus (選取事件匯流排)，保留預設的事件匯流排選項。

5. 在 Select targets (選取目標)，設定您的目標。下列範例使用 Amazon SQS 佇列，但您可以設定其他目標類型。

   a. 針對「目標」，選擇 SQS 佇列。

   b. 在佇列 * 中，選擇您的目標佇列。

6. 在 Tags - optional (標籤 - 選用)，定義規則的標籤，或將欄位留白。

7. 選擇建立。

建立事件規則以取得遙測資料 (CLI)

使用下列步驟來建立AWS CLI接收 Greengrass 核心裝置所發佈之遙測資料的 EventBridge 規則。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。

   - 將##########備的物件名稱。

   Linux or Unix

   ```
   aws events put-rule \
     --name MyGreengrassTelemetryEventRule \
     --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail\": {\"ThingName
   \": [\"thing-name\"]}}"
   ```

   Windows Command Prompt (CMD)

   ```
   aws events put-rule ^
     --name MyGreengrassTelemetryEventRule ^
   ```

```
  --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail\": {\"ThingName
\": [\"thing-name\"]}}"
```

PowerShell

```
aws events put-rule `
  --name MyGreengrassTelemetryEventRule `
  --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。

   - 使用 *Amazon SQS ### ARN* 取代佇列中的佇列。

   Linux or Unix

   ```
   aws events put-targets \
     --rule MyGreengrassTelemetryEventRule \
     --targets "Id"="1","Arn"="queue-arn"
   ```

   Windows Command Prompt (CMD)

   ```
   aws events put-targets ^
     --rule MyGreengrassTelemetryEventRule ^
     --targets "Id"="1","Arn"="queue-arn"
   ```

   PowerShell

   ```
   aws events put-targets `
     --rule MyGreengrassTelemetryEventRule `
     --targets "Id"="1","Arn"="queue-arn"
   ```

> **Note**
>
> 若要允許 Amazon 叫 EventBridge 用您的目標佇列，您必須在主題中新增以資源為基礎的政策。如需詳細資訊，請參閱 Amazon EventBridge 使用者指南中的 Amazon SQS 許可。

如需詳細資訊，請參閱 Amazon EventBridge 使用者指南 EventBridge中的事件和事件模式。

取得部署和元件健康狀態通知

Amazon EventBridge 事件規則會針對裝置收到的 Greengrass 部署以及裝置上已安裝元件的狀態變更提供通知。 EventBridge 提供描述 AWS 資源變更的近乎即時的系統事件串流。 AWS IoT Greengrass 將這些事件發送到盡最大努力的基礎 EventBridge 上。這表示 AWS IoT Greengrass 嘗試將所有事件傳送至， EventBridge 但在某些極少數情況下，事件可能無法傳遞。此外， AWS IoT Greengrass 可能會傳送給定事件的多個副本，這表示您的事件偵聽程式可能不會按照事件發生的順序接收事件。

> **Note**
>
> Amazon EventBridge 是一種事件匯流排服務，可用來將應用程式與來自各種來源的資料連接起來，例如 Greengrass 核心裝置以及部署和元件通知。如需詳細資訊，請參閱什麼是 Amazon EventBridge？ 在 Amazon 用 EventBridge 戶指南。

主題

部署狀態變更事件

AWS IoT Greengrass 當部署進入下列狀態時，會發出事件：FAILED、SUCCEEDED和COMPLETED。您可以建立針對所有狀態轉換或轉換至您指定的狀態執行的 EventBridge 規則。當部署進入啟動規則的狀態時，會 EventBridge 呼叫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：

- 啟動部署後作業，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。

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 V2 Effective Deployment Status Change",
    "source":"aws.greengrass",
    "account":"123456789012",
    "region":"us-west-2",
    "time":"2018-03-22T00:38:11Z",
    "resources":["arn:aws:greengrass:us-
east-1:123456789012:coreDevices:MyGreengrassCore"],
    "detail":{
        "deploymentId": "4f38f1a7-3dd0-42a1-af48-EXAMPLE09681",
        "coreDeviceExecutionStatus": "FAILED|SUCCEEDED|COMPLETED",
        "statusDetails": {
          "errorStack": ["DEPLOYMENT_FAILURE", "ARTIFACT_DOWNLOAD_ERROR", "S3_ERROR",
 "S3_ACCESS_DENIED", "S3_HEAD_OBJECT_ACCESS_DENIED"],
          "errorTypes": ["DEPENDENCY_ERROR", "PERMISSION_ERROR"],
        },
        "reason": "S3_HEAD_OBJECT_ACCESS_DENIED: FAILED_NO_STATE_CHANGE: Failed to
 download artifact name: 's3://pentest27/nucleus/281/aws.greengrass.nucleus.zip' for
 component aws.greengrass.Nucleus-2.8.1, reason: S3 HeadObject returns 403 Access
 Denied. Ensure the IAM role associated with the core device has a policy granting
 s3:GetObject. null (Service: S3, Status Code: 403, Request ID: HR94ZNT2161DAR58,
 Extended Request ID: wTX4DDI+qigQt3uzwl9rlnQiYlBgwvPm/KJFWeFAn9t1mnGXTms/
luLCYANgq08RIH+x2H+hEKc=)"
     }
}
```

您可以建立規則和事件，以便更新部署狀態。當部署完成為、OR 時FAILED，SUCCEEDED便會啟動事件COMPLETED。如果核心裝置上的部署失敗，您將收到詳細的回應，說明部署失敗的原因。如需部署錯誤碼的更多資訊，請參閱詳細的部署錯誤代碼。

部署狀態

- FAILED。 部署失敗。
- SUCCEEDED。 以物件群組為目標的部署已成功完成。
- COMPLETED。 以物件為目標的部署已成功完成。

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。

如需和中錯誤代碼的完整清單 errorStackserrorTypes，請參閱詳細的部署錯誤代碼和詳細的元件狀態代碼。

元件狀態變更事件

對於 2.12.2 及更早 AWS IoT Greengrass 版本，Greengrass 會在組件進入下列狀態時發出事件：和。ERRORED BROKEN對於 Greengrass 核 2.12.3 及更新版本，Greengrass 會在元件進入下列狀態時發出事件：、、和。ERRORED BROKEN RUNNING FINISHEDGreengrass 也會在部署完成時發出事件。您可以建立針對所有狀態轉換或轉換至您指定的狀態執行的 EventBridge 規則。當安裝的元件進入啟動規則的狀態時，會 EventBridge 呼叫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。

元件狀態變更的事件使用下列格式：

Greengrass nucleus v2.12.2 and earlier

<title>元件狀態：ERRORED或 BROKEN</title>

```
{
    "version":"0",
    "id":" cd4d811e-ab12-322b-8255-EXAMPLEb1bc8",
    "detail-type":"Greengrass V2 Installed Component Status Change",
    "source":"aws.greengrass",
    "account":"123456789012",
    "region":"us-west-2",
    "time":"2018-03-22T00:38:11Z",
    "resources":["arn:aws:greengrass:us-
east-1:123456789012:coreDevices:MyGreengrassCore"],
```

```
    "detail": {
       "components": [
          {
            "componentName": "MyComponent",
            "componentVersion": "1.0.0",
            "root": true,
            "lifecycleState": "ERRORED|BROKEN",
            "lifecycleStatusCodes": ["STARTUP_ERROR"],
            "lifecycleStateDetails": "An error occurred during startup. The startup
 script exited with code 1."
          }
        ]
    }
}
```

Greengrass nucleus v2.12.3 and later

<title>元件狀態：ERRORED或 BROKEN</title>

```
{
    "version":"0",
    "id":" cd4d811e-ab12-322b-8255-EXAMPLEb1bc8",
    "detail-type":"Greengrass V2 Installed Component Status Change",
    "source":"aws.greengrass",
    "account":"123456789012",
    "region":"us-west-2",
    "time":"2018-03-22T00:38:11Z",
    "resources":["arn:aws:greengrass:us-
east-1:123456789012:coreDevices:MyGreengrassCore"],
    "detail": {
       "components": [
          {
            "componentName": "MyComponent",
            "componentVersion": "1.0.0",
            "root": true,
            "lifecycleState": "ERRORED|BROKEN",
            "lifecycleStatusCodes": ["STARTUP_ERROR"],
            "lifecycleStateDetails": "An error occurred during startup. The startup
 script exited with code 1."
          }
        ]
    }
}
```

<title>元件狀態：RUNNING或 FINISHED</title>

```
{
    "version":"0",
    "id":" cd4d811e-ab12-322b-8255-EXAMPLEb1bc8",
    "detail-type":"Greengrass V2 Installed Component Status Change",
    "source":"aws.greengrass",
    "account":"123456789012",
    "region":"us-west-2",
    "time":"2018-03-22T00:38:11Z",
    "resources":["arn:aws:greengrass:us-
east-1:123456789012:coreDevices:MyGreengrassCore"],
    "detail": {
       "components": [
          {
            "componentName": "MyComponent",
            "componentVersion": "1.0.0",
            "root": true,
            "lifecycleState": "RUNNING|FINISHED",
            "lifecycleStateDetails": null
          }
        ]
    }
}
```

您可以建立規則和事件，讓您瞭解已安裝元件的狀態。當元件變更裝置上的狀態時，就會啟動事件。您將收到詳細的回應，說明元件錯誤或損壞的原因。您也會收到一個狀態碼，指出失敗的原因。若要取得有關元件狀態碼的更多資訊，請參閱詳細的元件狀態代碼。

建立 EventBridge 規則的先決條件

在建立的 EventBridge 規則之前 AWS IoT Greengrass，請執行下列動作：

- 熟悉中的事件、規則和目標。 EventBridge
- 建立並設定 EventBridge 規則呼叫的目標。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，請參閱什麼是 Amazon EventBridge？ 並在 Amazon 用 EventBridge 戶指南EventBridge中開始使用 Amazon。

設定裝置健全狀況通知 (主控台)

使用下列步驟建立 EventBridge 規則，以便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 使用者指南中的建立可從 AWS 資源觸發事件的 EventBridge 規則。

1. 打開 Amazon EventBridge 控制台。
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。當您帳戶中的某個 AWS 服務發出活動時，它始終會進入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，選擇 AWS 事件。
9. 對於事件模式，選擇AWS 服務。
10. 對於AWS 服務，請選擇 Greengrass。
11. 對於事件類型，請從下列項目中進行選擇：

    - 對於部署事件，請選擇 Greengrass V2 有效部署狀態變更。
    - 對於元件事件，請選擇 Greengrass V2 已安裝的元件狀態變更。
12. 選擇下一步。
13. 在目標類型欄位中，選擇 AWS 服務。
14. 對於選取目標，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以傳送通知。

    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。

15. 選擇下一步。
16. 檢閱規則的詳細資訊，然後選擇建立規則。

設定裝置健全狀況通知 (CLI)

使用下列步驟建立 EventBridge 規則，以在發生 Greengrass 狀態變更事件時發佈 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。

   - 適用於部署狀態變更事件。

   ```
   aws events put-rule \
     --name TestRule \
     --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail-type\":
    [\"Greengrass V2 Effective Deployment Status Change\"]}"
   ```

   - 對於元件狀態變更事件。

   ```
   aws events put-rule \
     --name TestRule \
     --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail-type\":
    [\"Greengrass V2 Installed Component Status Change\"]}"
   ```

   模式省略的屬性會遭到忽略。
2. 新增主題作為規則目標。

   - 用您*# Amazon SNS ### ARN* 替換主題 arn。

   ```
   aws events put-targets \
     --rule TestRule \
     --targets "Id"="1","Arn"="topic-arn"
   ```

   > **Note**
   >
   > 為了允許 Amazon 調 EventBridge 用您的目標主題，您必須在主題中添加基於資源的政策。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 EventBridge中的事件和事件模式。

設定裝置健全狀況通知 (AWS CloudFormation)

使用 AWS CloudFormation 範本建立 EventBridge 規則，以傳送有關 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

- 檢查裝置部署狀態
- 什麼是 Amazon EventBridge？ 在 Amazon 用 EventBridge 戶指南

檢查核心設備狀態

Greengrass 核心裝置會將其軟體元件的狀態回報給。 AWS IoT Greengrass您可以檢查每個裝置的健全狀況摘要，也可以檢查每個裝置上每個元件的狀態。

核心裝置具有下列健全狀態：

- HEALTHY— AWS IoT Greengrass 核心軟件和所有組件在核心設備上運行沒有問題。
- UNHEALTHY— AWS IoT Greengrass 核心軟體或元件在核心裝置上處於錯誤狀態。

> **Note**
>
> AWS IoT Greengrass 依賴個別裝置將狀態更新傳送至 AWS 雲端. 如果 AWS IoT Greengrass Core 軟件未在設備上運行，或者設備未連接到 AWS 雲端，則該設備的報告狀態可能不會反映其當前狀態。狀態時間戳記會指出裝置狀態上次更新的時間。
> 核心裝置會在下列時間傳送狀態更新：
>
> - AWS IoT Greengrass 核心軟體啟動時
> - 當核心裝置接收來自 AWS 雲端
> - 對於 Greengrass 核心 2.12.2 及更早版本，當核心裝置上任何元件的狀態變為或時，核心裝置會傳送狀態更新 ERRORED BROKEN
> - 對於 Greengrass 核心 2.12.3 及更新版本，當核心裝置上任何元件的狀態變成、、或時，核心裝置會傳送狀態更新 ERRORED BROKEN RUNNING FINISHED

- 您可以設定的固定間隔，預設為 24 小時

對於 AWS IoT Greengrass Core v2.7.0 及更新版本，核心裝置會在發生本機部署和雲端部署時傳送狀態更新

主題

檢查核心裝置的健全狀況

您可以檢查個別核心裝置的狀態。

若要檢查核心裝置的狀態 (AWS CLI)

- 執行下列命令以擷取裝置的狀態。以要查詢的核心裝置名稱取*coreDeviceName*代。

```
aws greengrassv2 get-core-device --core-device-thing-name coreDeviceName
```

回應包含核心裝置的相關資訊，包括其狀態。

檢查核心裝置群組的健全狀況

您可以檢查核心裝置群組 (物件群組) 的狀態。

若要檢查裝置群組的狀態 (AWS CLI)

- 執行下列命令以擷取多個核心裝置的狀態。將指令中的 ARN 取代為要查詢之物件群組的 ARN。

```
aws greengrassv2 list-core-devices --thing-group-arn "arn:aws:iot:region:account-id:thinggroup/thingGroupName"
```

回應包含物件群組中的核心裝置清單。清單中的每個項目都包含核心裝置的狀態。

檢查核心裝置元件狀態

您可以檢查核心裝置上軟體元件的狀態，例如生命週期狀態。如需元件生命週期狀態的更多資訊，請參閱開發AWS IoT Greengrass元件。

檢查核心裝置上元件的狀態 (AWS CLI)

- 執行下列命令以擷取核心裝置上元件的狀態。以要查詢的核心裝置名稱取*coreDeviceName*代。

```
aws greengrassv2 list-installed-components --core-device-thing-name coreDeviceName
```

回應包含核心裝置上執行的元件清單。清單中的每個項目都包含元件的生命週期狀態，包括資料的目前狀態，以及 Greengrass 核心裝置上次傳送包含特定元件的訊息到雲端的時間。回應也會包含將元件帶到 Greengrass 核心裝置的最新部署來源。

> **Note**
>
> 此命令檢索 Greengrass 核心設備運行的組件的分頁列表。根據預設，此清單不包含部署為其他元件相依性的元件。您可以將`topologyFilter`參數設定為，在回應中包含相依性`ALL`。

執行AWS Lambda函數

> **Note**
>
> AWS IoT Greengrass目前在 Windows 核心裝置上不支援此功能。

您可以將AWS Lambda函數匯入為AWS IoT Greengrass核心裝置上執行的元件。在以下情況下，您可能需要執行此操作：

- 您在 Lambda 函數中擁有要部署到核心裝置的應用程式程式碼。
- 您有想要在AWS IoT Greengrass V2核心裝置上執行的 AWS IoT Greengrass V1 應用程式。如需詳細資訊，請參閱 步驟 2：建立和部署 AWS IoT Greengrass V2 元件以移轉 AWS IoT Greengrass V1 應用程式。

Lambda 函數包括下列元件的相依性。當您匯入函數時，您不需要將這些元件定義為相依性。當您部署 Lambda 函數元件時，部署會包含這些 Lambda 元件相依性。

- Lambda 啟動器元件 (`aws.greengrass.LambdaLauncher`) 會處理程序和環境組態。
- Lambda 管理員元件 (`aws.greengrass.LambdaManager`) 會處理處理序間通訊和擴充。
- Lambda 執行階段元件 (`aws.greengrass.LambdaRuntimes`) 會為每個受支援的 Lambda 執行階段提供成品。

主題

要求

您的核心裝置和 Lambda 函數必須符合下列需求，才能在AWS IoT Greengrass核心軟體上執行函數：

- 您的核心裝置必須符合執行 Lambda 函數的需求。如果您希望核心裝置執行容器化 Lambda 函數，裝置必須符合要求才能執行。如需詳細資訊，請參閱 Lambda 函數要求。
- 您必須在核心裝置上安裝 Lambda 函數使用的程式設計語言。

> **Tip**
>
> 您可以建立安裝程式設計語言的元件，然後將該元件指定為 Lambda 函數元件的相依性。Greengrass 支持所 Lambda 支持的 Python，Node.js 和 Java 運行時的版本。Greengrass 不會對已淘汰的 Lambda 執行階段版本套用任何其他限制。您可以在上執行使用這些已取代執行階段的 Lambda 函數AWS IoT Greengrass，但無法在中AWS Lambda建立它們。如需 Lambda 執行階段AWS IoT Greengrass支援的詳細資訊，請參閱執行AWS Lambda函數。

設定函 Lambda 生命週期

Greengrass Lambda 函數生命週期會決定函數的啟動時間，以及它如何建立和使用容器。生命週期也決定AWS IoT Greengrass核心軟體如何保留函數處理常式之外的變數和預處理邏輯。

AWS IoT Greengrass支援隨需 (預設) 和長壽命週期：

- 隨需函數在調用時啟動，並在沒有任務要運行時停止。每次呼叫函式都會建立個別容器 (也稱為沙箱) 來處理呼叫，除非現有的容器可供重複使用。任何容器都可能會處理您傳送至函數的資料。

  隨需函數的多個調用可以同時運行。

  建立新容器時，不會保留您在函數處理常式外部定義的變數和預處理邏輯。
- 長壽命 (或固定) 功能會在AWS IoT Greengrass核心軟體啟動並在單一容器中執行時啟動。同一個容器會處理您傳送至函數的所有資料。

  多個調用排隊，直到 AWS IoT Greengrass Core 軟件運行較早的調用。

  每次呼叫處理常式時，都會保留您在函數處理常式外部定義的變數和預處理邏輯。

  當您需要在沒有任何初始輸入的情況下開始工作時，請使用長壽命的 Lambda 函數 例如，長期使用的函數可以載入並開始處理機器學習模型，以便在函數接收裝置資料時準備就緒。

> **Note**
>
> 長壽命函數具有與其處理程序的每次調用相關聯的超時。如果要叫用無限期執行的程式碼，您必須在處理常式之外啟動它。請確定處理常式之外沒有封鎖程式碼，可能會阻止函式初始化。
>
> 除非AWS IoT Greengrass核心軟體停止 (例如在部署或重新開機期間)，否則這些功能會執行。如果函數遇到未捕獲的異常，超過其內存限制或進入錯誤狀態（例如處理程序超時），這些函數將不會運行。

如需容器重複使用的詳細資訊，請參閱了解 AWSCompute 部落格AWS Lambda中的容器重複使用。

設定 Lambda 函數容器化

根據預設，Lambda 函數會在AWS IoT Greengrass容器內部執行。Greengrass 容器在您的函數和主機之間提供隔離。這種隔離可增加主機和容器中功能的安全性。

我們建議您在 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存取的方式。

在下列情況下，您可以在不使用容器化的情況下執行 Lambda 函數：

- 您想要AWS IoT Greengrass在不支援容器模式的裝置上執行。一個例子是，如果你想使用一個特殊的 Linux 發行版，或者有一個過期的早期內核版本。
- 您想要使用自己的 OverlayFS 函式在另一個容器環境中執行 Lambda 函數，但是在 Greengrass 容器中執行時，會遇到覆寫檔案衝突。
- 您需要存取具有無法在部署時判斷路徑的本機資源，或部署後路徑可能變更的路徑。此資源的一個例子是可插拔的設備。
- 您有一個較早的應用程序被編寫為進程，並且在 Greengrass 容器中運行它時遇到問題。

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AWS IoT Greengrass有功能都可以使用。

容器化	備註
	• 在 Greengrass 容器中執行的 Lambda 函數無法存取其他 Lambda 函數的已部署程式碼，即使它們使用相同的系統群組執行也是如此。換句話說，您的 Lambda 函數在彼此之間的隔離增加的情況下運行。 • 由於AWS IoT Greengrass核心軟體會在與 Lambda 函數相同的容器中執行所有子處理序，因此當 Lambda 函數停止時，子程序就會停止。

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。
	  • 本機裝置和磁碟區資源。您必須使用核心裝置上的檔案路徑來存取這些資源，而非 Lambda 函數資源。
	• 如果您的非容器化 Lambda 函數存取機器學習資源，則必須識別資源擁有者並設定資源的存取權限，而不是 Lambda 函數。
	• 非容器化 Lambda 函數對於使用相同系統群組執行的其他 Lambda 函數的已部署程式碼具有唯讀存取權。

如果您在部署 Lambda 函數時變更其容器化，則該函數可能無法如預期般運作。如果 Lambda 函數使用新容器化設定無法再使用的本機資源，部署會失敗。

- 當您將 Lambda 函數從在 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。在部署 Lambda 函數之前，您必須先移除所有連接的資源。
- 當您將 Lambda 函數從沒有容器化的情況下執行變更為在容器中執行時，Lambda 函數會失去檔案系統的直接存取權。您必須為每個功能定義記憶體限制，或接受預設的 16 MB 記憶體限制。您可以在部署每個 Lambda 函數時為其設定這些設定。

若要變更 Lambda 函數元件的容器化設定，請在部署元件時將`containerMode`組態參數的值設定為下列其中一個選項。

- `NoContainer`— 元件不會在隔離的執行階段環境中執行。
- `GreengrassContainer`— 元件在AWS IoT Greengrass容器內的隔離執行階段環境中執行。

如需如何部署和設定元件的詳細資訊，請參閱將AWS IoT Greengrass元件部署到裝置和更新零組件組態。

將 Lambda 函數匯入為元件 (主控台)

當您使用AWS IoT Greengrass主控台建立 Lambda 函數元件時，您會匯入現有AWS Lambda函數，然後將其設定為建立在 Greengrass 裝置上執行的元件。

在開始之前，請先檢閱在 Greengrass 裝置上執行 Lambda 函數的需求。

任務

- 步驟 1：選擇要匯入的 Lambda 函數
- 步驟 2：設定 Lambda 函數參數
- 步驟 3：(選擇性) 為 Lambda 函數指定支援的平台
- 步驟 4：(選擇性) 指定 Lambda 函數的元件相依性
- 步驟 5：(選用) 在容器中執行 Lambda 函數
- 步驟 6：建立 Lambda 函數元件

步驟 1：選擇要匯入的 Lambda 函數

1. 在AWS IoT Greengrass主控台瀏覽功能表中，選擇 [元件]。
2. 在 [元件] 頁面上，選擇 [建立元件]。
3. 在 [建立元件] 頁面的 [元件資訊] 下，選擇 [匯入 Lambda 函數]。
4. 在 Lambda 函數中，搜尋並選擇您要匯入的 Lambda 函數。

   AWS IoT Greengrass建立具有 Lambda 函數名稱的元件。
5. 在 Lambda 函數版本中，選擇要匯入的版本。您無法選擇 Lambda 別名，例如`$LATEST`。

   AWS IoT Greengrass將 Lambda 函數的版本建立為有效語意版本的元件。例如，如果您的函數版本為 `3`，則元件版本會變成 `3.0.0`。

步驟 2：設定 Lambda 函數參數

在 [建立元件] 頁面的 Lambda 函數組態下，設定下列參數以用來執行 Lambda 函數。

1. (選擇性) 新增 Lambda 函數訂閱工作訊息的事件來源清單。您可以指定事件來源，將此函數訂閱到本機發佈/訂閱訊息和 AWS IoT Core MQTT 訊息。當接收來自事件來源的訊息時，就會呼叫 Lambda 函數。

   > **Note**
   >
   > 若要訂閱此函數以接收來自其他 Lambda 函數或元件的訊息，請在部署此 Lambda 函數元件時部署舊版訂閱路由器元件。部署舊版訂閱路由器元件時，請指定 Lambda 函數使用的訂閱。

   在「事件來源」下，執行下列動作以新增事件來源：

   a. 針對您新增的每個事件來源，指定下列選項：

      - 主題 — 要訂閱訊息的主題。
      - 類型 — 事件來源的類型。您可以從以下選項中選擇：
        - 本機發佈/訂閱 — 訂閱本機發佈/訂閱訊息。

          如果您使用 Greengrass 核心 v2.6.0 或更新版本和 Lambda 管理員 v2.2.5 或更新版本，則可以在指定此類型時在主題中使用 MQTT 主題萬用字元 (**+**和)。**#**
        - AWS IoT CoreMQTT — 訂閱 AWS IoT Core MQTT 訊息。

          指定此類型時，您可以在「主題」中使用 MQTT 主題萬用字元 (+和#)。

   b. 若要新增其他事件來源，請選擇 [新增事件來源]，然後重複上一個步驟。若要移除事件來源，請在您要移除的事件來源旁邊選擇 [移除]。

2. 針對逾時 (秒)，輸入非釘選 Lambda 函數在逾時之前可執行的時間上限 (以秒為單位)。預設為 3 秒。
3. 對於「釘接」，請選擇是否釘選 Lambda 函數元件。預設值為「真」。

   - 固定（或長壽命）Lambda 函數在啟動時AWS IoT Greengrass啟動，並在自己的容器中繼續運行。

- 非釘選 (或隨選) Lambda 函數只會在收到工作項目時啟動，並在閒置超過指定的最大閒置時間後結束。如果此函數具有多個工作項目，AWS IoT Greengrass Core 軟體會建立此函數的多個執行個體。

4. (選擇性) 在其他參數下，設定下列 Lambda 函數參數。

   - 狀態逾時 (秒) — Lambda 函數元件將狀態更新傳送至 Lambda 管理員元件的間隔 (秒)。此參數僅適用於固定的函數。預設值為 60 秒。
   - 佇列大小上限 — Lambda 函數元件的訊息佇列大小上限。AWS IoT Greengrass核心軟體會將訊息儲存在 FIFO (先進先出) 佇列中，直到它可以執行 Lambda 函數來使用每則訊息為止。預設值為 1,000 則訊息。
   - 執行個體數目上限 — 非固定 Lambda 函數可同時執行的執行個體數目上限。預設值為 100 個執行個體。
   - 最大閒置時間 (秒) — 非釘選 Lambda 函數在 AWS IoT Greengrass Core 軟體停止處理程序之前可閒置的最大時間 (秒)。預設值為 60 秒。
   - 編碼類型 — Lambda 函數支援的有效負載類型。您可以從以下選項中選擇：
     - JSON
     - ：二進位

     預設值為 JSON。

5. (選擇性) 指定執行 Lambda 函數時要傳遞給 Lambda 函數的命令列引數清單。

   a. 在其他參數的處理引數下，選擇新增引數。
   b. 針對您新增的每個引數，輸入要傳遞至函數的引數。
   c. 若要移除引數，請選擇您要移除之引數旁邊的 [移除]。

6. (選擇性) 指定 Lambda 函數執行時可用的環境變數。環境變數可讓您儲存和更新組態設定，而不需要變更函數程式碼。

   a. 在其他參數的環境變數下，選擇新增環境變數。
   b. 針對您新增的每個環境變數，指定下列選項：
      - Key — 變數名稱。
      - 值 — 此變數的預設值。
   c. 若要移除環境變數，請在您要移除的環境變數旁邊選擇「移除」。

步驟 3：(選擇性) 為 Lambda 函數指定支援的平台

所有核心裝置都具有作業系統和架構的屬性。部署 Lambda 函數元件時，AWS IoT Greengrass核心軟體會將您指定的平台值與核心裝置上的平台屬性進行比較，以判斷該裝置是否支援 Lambda 函數。

> **Note**
>
> 當您將 Greengrass 核心元件部署到核心裝置時，您也可以指定自訂平台屬性。如需詳細資訊，請參閱 Greengrass 核心元件的平台覆寫參數。

在 Lambda 函數組態下的其他參數「平台」下執行下列動作，以指定此 Lambda 函數支援的平台。

1. 針對每個平台，指定下列選項：
   - 作業系統 — 平台的作業系統名稱。目前，僅支援的值為 `linux`。
   - 架構 — 平台的處理器架構。支援的值如下：
     - `amd64`
     - `arm`
     - `aarch64`
     - `x86`
2. 若要新增其他平台，請選擇 [新增平台]，然後重複上一個步驟。若要移除支援的平台，請在您要移除的平台旁邊選擇「移除」。

步驟 4：(選擇性) 指定 Lambda 函數的元件相依性

組件依賴關係識別您的函數使用的其他AWS提供的組件或自定義組件。部署 Lambda 函數元件時，部署會包含這些相依性供您的函數執行。

> **Important**
>
> 若要匯入您建立要在 AWS IoT Greengrass V1 上執行的 Lambda 函數，您必須針對函數使用的功能 (例如秘密、本機陰影和串流管理員) 定義個別元件相依性。將這些元件定義為硬式相依性，以便 Lambda 函數元件在相依性變更狀態時重新啟動。如需詳細資訊，請參閱 匯入 V1 Lambda 函數。

在 Lambda 函數組態「其他參數」「元件相依性」下，完成下列步驟，以指定 Lambda 函數的元件相依性。

1. 選擇新增相依性。
2. 針對您新增的每個元件相依性，指定下列選項：
   - 元件名稱 — 元件名稱。例如，輸入**aws.greengrass.StreamManager**以包含串流管理員元件。
   - 版本需求 — 識別此元件相依性的相容版本的 npm-style 語意版本條件約束。您可以指定單一版本或版本範圍。例如，輸入**^1.0.0**以指定此 Lambda 函數取決於串流管理員元件第一個主要版本中的任何版本。如需語意版本條件約束的詳細資訊，請參閱 npm semver 計算器。
   - 「類型」— 相依性的類型。您可以從以下選項中選擇：
     - 硬體 — 如果相依性變更狀態，Lambda 函數元件會重新啟動。這是預設選項。
     - 軟體 — 如果相依性變更狀態，Lambda 函數元件不會重新啟動。
3. 若要移除元件相依性，請選擇元件相依性旁邊的移除

步驟 5：(選用) 在容器中執行 Lambda 函數

根據預設，Lambda 函數會在AWS IoT Greengrass核心軟體內的隔離執行階段環境中執行。您也可以選擇在沒有任何隔離的情況下以程序的方式執行 Lambda 函數 (也就是在「無容器」模式下)。

在 Linux 程序組態下，針對隔離模式，從下列選項中選擇以選取 Lambda 函數的容器化：

- 容器-Lambda 函數在容器中運行。這是預設選項。
- 無容器 — Lambda 函數在沒有任何隔離的情況下以程序的形式執行。

如果您在容器中執行 Lambda 函數，請完成下列步驟以設定 Lambda 函數的程序組態。

1. 設定要提供給容器使用的記憶體數量和系統資源 (例如磁碟區和裝置)。

   在「容器參數」下，執行下列操作。

   a. 在記憶體大小中，輸入您要配置給容器的記憶體大小。您可以指定以 MB 或 k B 為單位的記憶體大小。

   b. 對於唯讀 sys 資料夾，請選擇容器是否可以讀取設備資/sys料夾中的資訊。預設值為「假」。

2. (選擇性) 設定容器化 Lambda 函數可存取的本機磁碟區。在您定義磁碟區時，AWS IoT Greengrass Core 軟體會將來源檔案掛載至容器內的目的地。

   a. 在磁碟區下，選擇新增磁碟區。

   b. 針對您新增的每個磁碟區，指定下列選項：

      - 實體磁碟區 — 核心裝置上來源資料夾的路徑。
      - 邏輯磁碟區 — 容器中目標資料夾的路徑。
      - 權限 — (選用) 從容器存取來源資料夾的權限。您可以從以下選項中選擇：
        - 唯讀 — Lambda 函數對來源資料夾具有唯讀存取權。這是預設選項。
        - 讀寫 — Lambda 函數具有來源資料夾的讀取/寫入存取權。
      - 新增群組擁有者 — (選用) 是否要新增執行 Lambda 函數元件的系統群組，做為來源資料夾的擁有者。預設值為「假」。

   c. 若要移除卷宗，請選擇您要移除的磁碟區旁邊的 [移除]。

3. (選擇性) 設定容器化 Lambda 函數可存取的本機系統裝置。

   a. 在 [裝置] 下，選擇 [新增裝置]

   b. 針對您新增的每個裝置，指定下列選項：

      - 掛載路徑 — 核心裝置上系統裝置的路徑。
      - 權限 — (選用) 從容器存取系統裝置的權限。您可以從以下選項中選擇：
        - 唯讀 — Lambda 函數對系統裝置具有唯讀存取權。這是預設選項。
        - 讀寫 — Lambda 函數具有來源資料夾的讀取/寫入存取權。
      - 新增群組擁有者 — (選用) 是否新增以系統裝置擁有者身分執行 Lambda 函數元件的系統群組。預設值為「假」。

步驟 6：建立 Lambda 函數元件

設定 Lambda 函數元件的設定後，請選擇建立以完成新元件的建立。

若要在核心裝置上執行 Lambda 函數，您可以接著將新元件部署到核心裝置。如需更多詳細資訊，請參閱 將AWS IoT Greengrass元件部署到裝置。

將 Lambda 函數匯入為元件 (AWS CLI)

使用此CreateComponentVersion作業從 Lambda 函數建立元件。呼叫此作業時，請指定匯`lambdaFunction`入 Lambda 函數。

任務

- 步驟 1：定義 Lambda 函數組態
- 步驟 2：建立 Lambda 函數元件

步驟 1：定義 Lambda 函數組態

1. 建立名為的檔案`lambda-function-component.json`，然後將下列 JSON 物件複製到檔案中。以要匯入的 Lambda 函數的 ARN 取代。`lambdaArn`

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1"
  }
}
```

> **Important**
>
> 您必須指定包含要匯入之函數版本的 ARN。您不能使用 $LATEST 之類的版本別名。

2. (選擇性) 指定元件的名稱 (`componentName`)。如果您省略此參數，請使用 Lambda 函數的名稱AWS IoT Greengrass建立元件。

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda"
  }
}
```

3. (選擇性) 指定元件的版本 (`componentVersion`)。如果您省略此參數，則AWS IoT Greengrass會建立具有 Lambda 函數版本作為有效語意版本的元件。例如，如果您的函數版本為 `3`，則元件版本會變成 `3.0.0`。

> **Note**
>
> 您上傳的每個元件版本都必須是唯一的。請務必上傳正確的元件版本，因為上傳後就無法編輯它。
>
> AWS IoT Greengrass使用語義版本的組件。語義版本遵循一個主要的。 未成年人。 修補程式編號系統。例如，版本1.0.0代表元件的第一個主要發行版本。如需詳細資訊，請參閱語意版本規格。

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0"
  }
}
```

4. (選擇性) 指定此 Lambda 函數支援的平台。每個平台都包含可識別平台的屬性對映。所有核心裝置都具有作業系統 (os) 和架構 (architecture) 的屬性。AWS IoT Greengrass核心軟體可能會新增其他平台屬性。當您將 Greengrass 核心元件部署到核心裝置時，您也可以指定自訂平台屬性。請執行下列操作：

   a. 將平台清單 (componentPlatforms) 新增至中的 Lambda 函數lambda-function-component.json。

   ```
   {
     "lambdaFunction": {
       "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
       "componentName": "com.example.HelloWorldLambda",
       "componentVersion": "1.0.0",
       "componentPlatforms": [

       ]
     }
   }
   ```

   b. 將每個支援的平台新增至清單。每個平台都有一name個友好的識別它和屬性映射。下列範例會指定此函數支援執行 Linux 的 x86 裝置。

```
{
  "name": "Linux x86",
  "attributes": {
    "os": "linux",
    "architecture": "x86"
  }
}
```

您lambda-function-component.json可能包含類似下列範例的文件。

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ]
  }
}
```

5. (選擇性) 指定 Lambda 函數的元件相依性。部署 Lambda 函數元件時，部署會包含這些相依性供您的函數執行。

> **Important**
>
> 若要匯入您建立要在 AWS IoT Greengrass V1 上執行的 Lambda 函數，您必須針對函數使用的功能 (例如秘密、本機陰影和串流管理員) 定義個別元件相依性。將這些元件定義為硬式相依性，以便 Lambda 函數元件在相依性變更狀態時重新啟動。如需詳細資訊，請參閱 匯入 V1 Lambda 函數。

請執行下列操作：

a. 將元件相依性 (componentDependencies) 的對應新增至中的 Lambda 函數lambda-function-component.json。

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {

    }
  }
}
```

b. 將每個組件依賴關係添加到映射中。指定元件名稱做為索引鍵，並使用下列參數指定物件：

- versionRequirement— 識別元件相依性的相容版本的 npm-style 語意版本條件約束。您可以指定單一版本或版本範圍。如需語意版本條件約束的詳細資訊，請參閱 npm semver 計算器。
- dependencyType— (選用) 相依性的類型。請選擇下列項目：
  - SOFT— 如果相依性變更狀態，Lambda 函數元件不會重新啟動。
  - HARD— 如果相依性變更狀態，Lambda 函數元件會重新啟動。

  預設值為 HARD。

下列範例會指定此 Lambda 函數依賴串流管理員元件第一個主要版本中的任何版本。Lambda 函數元件會在串流管理員重新啟動或更新時重新啟

```
{
  "aws.greengrass.StreamManager": {
```

```
    "versionRequirement": "^1.0.0",
    "dependencyType": "HARD"
  }
}
```

您lambda-function-component.json可能包含類似下列範例的文件。

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    }
  }
}
```

6. (選擇性) 設定用來執行函數的 Lambda 函數參數。您可以設定環境變數、訊息事件來源、逾時和容器設定等選項。請執行下列操作：

   a. 將 Lambda 參數物件 (componentLambdaParameters) 新增至中的 Lambda 函數lambda-function-component.json。

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
```

```
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {

    }
  }
}
```

b. (選擇性) 指定 Lambda 函數訂閱工作訊息的事件來源。您可以指定事件來源，將此函數訂閱到本機發佈/訂閱訊息和 AWS IoT Core MQTT 訊息。當接收來自事件來源的訊息時，就會呼叫 Lambda 函數。

> **Note**
>
> 若要訂閱此函數以接收來自其他 Lambda 函數或元件的訊息，請在部署此 Lambda 函數元件時部署舊版訂閱路由器元件。部署舊版訂閱路由器元件時，請指定 Lambda 函數使用的訂閱。

請執行下列操作：

i. 將事件來源清單 (`eventSources`) 新增至 Lambda 函數參數。

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
```

```
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [

      ]
    }
  }
}
```

ii. 將每個事件來源新增至清單。每個事件來源都有下列參數：

- `topic`— 要訂閱訊息的主題。
- `type`— 事件來源的類型。您可以從以下選項中選擇：
  - PUB_SUB - 訂閱本機發佈/訂閱訊息。

    如果您使用 Greengrass 核心 v2.6.0 或更新版本和 Lambda 管理員 v2.2.5 或更新版本，則可以在指定此類型時，在中使用 MQTT 主題萬用字元 (和)。+ # `topic`
  - IOT_CORE – 訂閱 AWS IoT Core MQTT 訊息。

    `topic`當您指定此類型時，您可以在中使用 MQTT 主題萬用字元 (+和#)。

下列範例會針對符合主題篩選`hello/world/+`器的主題訂閱 AWS IoT Core MQTT。

```
{
  "topic": "hello/world/+",
  "type": "IOT_CORE"
}
```

您的外觀`lambda-function-component.json`可能類似於下列範例。

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-
id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
      ]
    }
  }
}
```

c. (選擇性) 在 Lambda 函數參數物件中指定下列任一參數：

- `environmentVariables`— Lambda 函數執行時可用的環境變數對應。
- `execArgs`— 執行時傳遞給 Lambda 函數的引數清單。
- `inputPayloadEncodingType`— Lambda 函數支援的有效負載類型。您可以從以下選項中選擇：
  - `json`
  - `binary`

預設：json

- pinned— 無論 Lambda 函數是否已固定。預設值為 true。
  - 固定（或長壽命）Lambda 函數在啟動時AWS IoT Greengrass啟動，並在自己的容器中繼續運行。
  - 非釘選 (或隨選) Lambda 函數只會在收到工作項目時啟動，並在閒置超過指定的最大閒置時間後結束。如果此函數具有多個工作項目，AWS IoT Greengrass Core 軟體會建立此函數的多個執行個體。

  用maxIdleTimeInSeconds於設定函數的最大閒置時間。
- timeoutInSeconds— Lambda 函數在逾時之前可以執行的時間上限 (以秒為單位)。預設為 3 秒。
- statusTimeoutInSeconds— Lambda 函數元件將狀態更新傳送至 Lambda 管理員元件的間隔 (秒)。此參數僅適用於固定的函數。預設值為 60 秒。
- maxIdleTimeInSeconds— AWS IoT Greengrass 核心軟體停止處理程序之前，非釘選 Lambda 函數可閒置的最大時間 (以秒為單位)。預設值為 60 秒。
- maxInstancesCount— 非固定 Lambda 函數可同時執行的執行個體數目上限。預設值為 100 個執行個體。
- maxQueueSize— Lambda 函數元件的訊息佇列大小上限。AWS IoT Greengrass核心軟體會將訊息儲存在 FIFO (first-in-first-out) 佇列中，直到它可以執行 Lambda 函數來使用每個訊息為止。預設值為 1,000 則訊息。

您lambda-function-component.json可能包含類似下列範例的文件。

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
```

```
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
      ],
      "environmentVariables": {
        "LIMIT": "300"
      },
      "execArgs": [
        "-d"
      ],
      "inputPayloadEncodingType": "json",
      "pinned": true,
      "timeoutInSeconds": 120,
      "statusTimeoutInSeconds": 30,
      "maxIdleTimeInSeconds": 30,
      "maxInstancesCount": 50,
      "maxQueueSize": 500
    }
  }
}
```

d. (選擇性) 設定 Lambda 函數的容器設定。根據預設，Lambda 函數會在AWS IoT Greengrass 核心軟體內的隔離執行階段環境中執行。您也可以選擇在沒有任何隔離的情況下以程序的形式執行 Lambda 函數。如果您在容器中執行 Lambda 函數，則需要設定容器的記憶體大小，以及 Lambda 函數可使用的系統資源。請執行下列操作：

   i. 將 Linux 程序參數物件 (`linuxProcessParams`) 新增至中的 Lambda 參數物件`lambda-function-component.json`。

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
```

```
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
      ],
      "environmentVariables": {
        "LIMIT": "300"
      },
      "execArgs": [
        "-d"
      ],
      "inputPayloadEncodingType": "json",
      "pinned": true,
      "timeoutInSeconds": 120,
      "statusTimeoutInSeconds": 30,
      "maxIdleTimeInSeconds": 30,
      "maxInstancesCount": 50,
      "maxQueueSize": 500,
      "linuxProcessParams": {

      }
    }
  }
}
```

ii. (選擇性) 指定 Lambda 函數是否在容器中執行。將isolationMode參數新增至流程參數物件，然後從下列選項中進行選擇：

- GreengrassContainer-Lambda 函數在容器中運行。
- NoContainer-Lambda 函數作為一個過程運行，沒有任何隔離。

預設值為 GreengrassContainer。

iii. (選擇性) 如果您在容器中執行 Lambda 函數，您可以設定可供容器使用的記憶體數量和系統資源 (例如磁碟區和裝置)。請執行下列操作：

A. 將容器參數物件 (containerParams) 新增至中的 Linux 程序參數物件lambda-function-component.json。

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-
id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
      ],
```

```
      "environmentVariables": {
        "LIMIT": "300"
      },
      "execArgs": [
        "-d"
      ],
      "inputPayloadEncodingType": "json",
      "pinned": true,
      "timeoutInSeconds": 120,
      "statusTimeoutInSeconds": 30,
      "maxIdleTimeInSeconds": 30,
      "maxInstancesCount": 50,
      "maxQueueSize": 500,
      "linuxProcessParams": {
        "containerParams": {

        }
      }
    }
  }
}
```

B. (選擇性) 新增`memorySizeInKB`參數以指定容器的記憶體大小。預設值為 16,384 KB。

C. (選擇性) 新增`mountROSysfs`參數以指定容器是否可以從設備資`/sys`料夾讀取資訊。預設值為 `false`。

D. (選擇性) 設定容器化 Lambda 函數可存取的本機磁碟區。在您定義磁碟區時，AWS IoT Greengrass Core 軟體會將來源檔案掛載至容器內的目的地。請執行下列操作：

   I. 將磁碟區清單 (`volumes`) 新增至容器參數。

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-
id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
```

```
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
      ],
      "environmentVariables": {
        "LIMIT": "300"
      },
      "execArgs": [
        "-d"
      ],
      "inputPayloadEncodingType": "json",
      "pinned": true,
      "timeoutInSeconds": 120,
      "statusTimeoutInSeconds": 30,
      "maxIdleTimeInSeconds": 30,
      "maxInstancesCount": 50,
      "maxQueueSize": 500,
      "linuxProcessParams": {
        "containerParams": {
          "memorySizeInKB": 32768,
          "mountROSysfs": true,
          "volumes": [

          ]
        }
      }
    }
  }
}
```

II. 將每個磁碟區新增至清單。每個磁碟區都有下列參數：

- `sourcePath`— 核心裝置上來源資料夾的路徑。
- `destinationPath`— 容器中目標資料夾的路徑。
- `permission`— (選用) 從容器存取來源資料夾的權限。您可以從以下選項中選擇：
  - `ro`— Lambda 函數對來源資料夾具有唯讀存取權。
  - `rw`— Lambda 函數具有來源資料夾的讀寫存取權限。

  預設值為 `ro`。
- `addGroupOwner`— (選用) 是否新增執行 Lambda 函數元件的系統群組，做為來源資料夾的擁有者。預設值為 `false`。

您`lambda-function-component.json`可能包含類似下列範例的文件。

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-
id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
```

```
        }
      ],
      "environmentVariables": {
        "LIMIT": "300"
      },
      "execArgs": [
        "-d"
      ],
      "inputPayloadEncodingType": "json",
      "pinned": true,
      "timeoutInSeconds": 120,
      "statusTimeoutInSeconds": 30,
      "maxIdleTimeInSeconds": 30,
      "maxInstancesCount": 50,
      "maxQueueSize": 500,
      "linuxProcessParams": {
        "containerParams": {
          "memorySizeInKB": 32768,
          "mountROSysfs": true,
          "volumes": [
            {
              "sourcePath": "/var/data/src",
              "destinationPath": "/var/data/dest",
              "permission": "rw",
              "addGroupOwner": true
            }
          ]
        }
      }
    }
  }
}
```

E. (選擇性) 設定容器化 Lambda 函數可存取的本機系統裝置。請執行下列操作：

I. 將系統裝置 (devices) 清單新增至容器參數。

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-
id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
```

```
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
      ],
      "environmentVariables": {
        "LIMIT": "300"
      },
      "execArgs": [
        "-d"
      ],
      "inputPayloadEncodingType": "json",
      "pinned": true,
      "timeoutInSeconds": 120,
      "statusTimeoutInSeconds": 30,
      "maxIdleTimeInSeconds": 30,
      "maxInstancesCount": 50,
      "maxQueueSize": 500,
      "linuxProcessParams": {
        "containerParams": {
          "memorySizeInKB": 32768,
          "mountROSysfs": true,
          "volumes": [
            {
              "sourcePath": "/var/data/src",
              "destinationPath": "/var/data/dest",
              "permission": "rw",
              "addGroupOwner": true
```

```
            }
          ],
          "devices": [

          ]
        }
      }
    }
  }
}
```

II. 將每個系統裝置新增至清單。每個系統設備具有以下參數：

- path— 核心裝置上系統裝置的路徑。
- permission— (選用) 從容器存取系統裝置的權限。您可以從以下選項中選擇：
  - ro— Lambda 函數對系統裝置具有唯讀存取權限。
  - rw-Lambda 函數具有對系統設備的讀寫訪問權限。

  預設值為 ro。
- addGroupOwner— (選用) 是否新增以系統裝置擁有者身分執行 Lambda 函數元件的系統群組。預設值為 false。

您lambda-function-component.json可能包含類似下列範例的文件。

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-
id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
```

```
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
      ],
      "environmentVariables": {
        "LIMIT": "300"
      },
      "execArgs": [
        "-d"
      ],
      "inputPayloadEncodingType": "json",
      "pinned": true,
      "timeoutInSeconds": 120,
      "statusTimeoutInSeconds": 30,
      "maxIdleTimeInSeconds": 30,
      "maxInstancesCount": 50,
      "maxQueueSize": 500,
      "linuxProcessParams": {
        "containerParams": {
          "memorySizeInKB": 32768,
          "mountROSysfs": true,
          "volumes": [
            {
              "sourcePath": "/var/data/src",
              "destinationPath": "/var/data/dest",
              "permission": "rw",
              "addGroupOwner": true
            }
          ],
          "devices": [
            {
              "path": "/dev/sda3",
              "permission": "rw",
              "addGroupOwner": true
            }
          ]
```

```
        }
      }
    }
  }
}
```

7. (選擇性) 為元件加入標籤 (tags)。如需詳細資訊，請參閱 [標記您的 AWS IoT Greengrass Version 2 資源](#)。

步驟 2：建立 Lambda 函數元件

1. 執行下列命令以從中建立 Lambda 函數元件`lambda-function-component.json`。

```
aws greengrassv2 create-component-version --cli-input-json file://lambda-function-component.json
```

如果要求成功，回應看起來會類似下列範例。

```
{
  "arn":
 "arn:aws:greengrass:region:123456789012:components:com.example.HelloWorldLambda:versions:1
  "componentName": "com.example.HelloWorldLambda",
  "componentVersion": "1.0.0",
  "creationTimestamp": "Mon Dec 15 20:56:34 UTC 2020",
  "status": {
    "componentState": "REQUESTED",
    "message": "NONE",
    "errors": {}
  }
}
```

`arn`從輸出複製，以在下一個步驟中檢查組件的狀態。

2. 當您建立元件時，其狀態為`REQUESTED`。然後，AWS IoT Greengrass驗證元件是否可部署。您可以執行下列命令來查詢元件狀態，並確認元件是否可部署。`arn`使用上一個步驟中的 ARN 取代。

```
aws greengrassv2 describe-component \
  --arn "arn:aws:greengrass:region:account-id:components:com.example.HelloWorldLambda:versions:1.0.0"
```

如果組件驗證，則響應指示組件狀態為DEPLOYABLE。

```
{
  "arn": "arn:aws:greengrass:region:account-
id:components:com.example.HelloWorldLambda:versions:1.0.0",
  "componentName": "com.example.HelloWorldLambda",
  "componentVersion": "1.0.0",
  "creationTimestamp": "2020-12-15T20:56:34.376000-08:00",
  "publisher": "AWS Lambda",
  "status": {
    "componentState": "DEPLOYABLE",
    "message": "NONE",
    "errors": {}
  },
  "platforms": [
    {
      "name": "Linux x86",
      "attributes": {
        "architecture": "x86",
        "os": "linux"
      }
    }
  ]
}
```

在元件完成之後DEPLOYABLE，您可以將 Lambda 函數部署到核心裝置。如需更多詳細資訊，請參閱 將AWS IoT Greengrass元件部署到裝置。

使AWS IoT Device SDK用與 Greengrass 核、其他元件和通訊 AWS IoT Core

核心裝置上執行的元件可以使用中的核AWS IoT Greengrass心處理序間通訊 (IPC) 程式庫，與AWS IoT Greengrass核心和其他 Greengrass 元件進行通訊。AWS IoT Device SDK若要開發和執行使用 IPC 的自訂元件，您必須使用連線AWS IoT Device SDK至 AWS IoT Greengrass Core IPC 服務並執行 IPC 作業。

IPC 介面支援兩種類型的操作：

- 請求/回應

  元件會傳送要求至 IPC 服務，並接收包含要求結果的回應。
- 訂閱

  元件會將訂閱要求傳送至 IPC 服務，並預期事件訊息串流以作回應。組件提供處理事件消息，錯誤和流關閉的訂閱處理程序。AWS IoT Device SDK包括一個處理常式介面，其中包含每個 IPC 作業的正確回應和事件類型。如需詳細資訊，請參閱 訂閱 IPC 事件串流。

主題

- 驗證和授權用戶端裝置

IPC 用戶端版本

在 Java 和 Python 軟件開發套件的更新版本中，AWS IoT Greengrass提供了 IPC 客戶端的改進版本，稱為 IPC 客戶端 V2。IPC 客戶端 V2：

- 減少使用 IPC 作業所需撰寫的程式碼量，並協助避免 IPC 用戶端 V1 可能發生的常見錯誤。
- 在單獨的執行緒中呼叫訂閱處理常式回呼，因此您現在可以在訂閱處理常式回呼中執行封鎖程式碼，包括其他 IPC 函數呼叫。IPC 用戶端 V1 使用相同的執行緒與 IPC 伺服器通訊，並呼叫訂閱處理常式回呼。
- 可讓您使用 Lambda 運算式 (Java) 或函數 (Python) 來呼叫訂閱作業。IPC 用戶端 V1 需要您定義訂閱處理常式類別。
- 提供每個 IPC 操作的同步和非同步版本。IPC 用戶端 V1 僅提供每個作業的非同步版本。

我們建議您使用 IPC 用戶端 V2 來利用這些改良功能。不過，本文件和某些線上內容中的許多範例只會示範如何使用 IPC Client V1。您可以使用下列範例和教學課程來查看使用 IPC 用戶端 V2 的範例元件：

- PublishToTopic例子
- SubscribeToTopic例子
- 教學課程：開發延遲元件更新的 Greengrass 元件
- 教學課程：透過 MQTT 與本地 IoT 裝置互動

目前，用AWS IoT Device SDK於 C ++ v2 的僅支持 IPC 客戶端 V1。

支援處理序間通訊的 SDK

AWS IoT Greengrass酷睿 IPC 庫包含在以下AWS IoT Device SDK版本中。

SDK	最低版本	用量	
AWS IoT Device SDK對於爪哇 V2	v1.6.0	請參閱 用AWS IoT Device SDK於 Java V2（IPC 客戶端 V2）	

SDK	最低版本	用量	
AWS IoT Device SDK 對於 Python V2	v1.9.0	請參閱 用AWS IoT Device SDK於 Python V2（IPC 客戶端 V2）	
AWS IoT Device SDK 對於 C ++ 第 2	v1.17.0	請參閱 用AWS IoT Device SDK於 C ++ 第 2	
AWS IoT Device SDK 對於 JavaScript V2	v1.12.0	請參閱 使AWS IoT Device SDK用 JavaScript V2 (IPC 用戶端 V1)	

Connect 到AWS IoT Greengrass酷睿 IPC 服務

若要在自訂元件中使用處理序間通訊，您必須建立與 AWS IoT Greengrass Core 軟體執行之 IPC 伺服器通訊端的連線。完成下列任務以下載並使AWS IoT Device SDK用您選擇的語言。

用AWS IoT Device SDK於 Java V2（IPC 客戶端 V2）

若要使AWS IoT Device SDK用 Java V2 (IPC 用戶端 V2)

1. 下載AWS IoT Device SDK適用於 Java V2（版本 1.6.0 或更高版本）。
2. 執行下列其中一項作業，在元件中執行自訂程式碼：
   - 將元件建置為包含的 JAR 檔案AWS IoT Device SDK，並在元件方案中執行此 JAR 檔案。
   - 將 AWS IoT Device SDK JAR 定義為元件人工因素，並在元件方案中執行應用程式時，將該成品新增至類別路徑。
3. 使用下列程式碼建立 IPC 用戶端。

```
try (GreengrassCoreIPCClientV2 ipcClient =
 GreengrassCoreIPCClientV2.builder().build()) {
    // Use client.
} catch (Exception e) {
    LOGGER.log(Level.SEVERE, "Exception occurred when using IPC.", e);
    System.exit(1);
```

```
}
```

用AWS IoT Device SDK於 Python V2（IPC 客戶端 V2）

若要使AWS IoT Device SDK用 Python V2 (IPC 用戶端 V2)

1. 下載AWS IoT Device SDK適用於 Python（版本 1.9.0 或更高版本）。
2. 在元件的配方中，將 SDK 的安裝步驟新增至安裝生命週期。
3. 建立與AWS IoT Greengrass核心 IPC 服務的連線。使用下列程式碼建立 IPC 用戶端。

```
from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2

try:
    ipc_client = GreengrassCoreIPCClientV2()
    # Use IPC client.
except Exception:
    print('Exception occurred when using IPC.', file=sys.stderr)
    traceback.print_exc()
    exit(1)
```

用AWS IoT Device SDK於 C ++ 第 2

要為 C ++ 構建 AWS IoT Device SDK v2，設備必須具有以下工具：

- C ++ 11 或更高版本
- 製造 3.1 或更高版本
- 下列其中一個編譯器：
  - 海灣合作委員會 4.8 或
  - 鐺 3.9 或更高版本
  - 無線電視廣告 2015 或更新版

若要使用用AWS IoT Device SDK於 C++ V2

1. 下載AWS IoT Device SDK適用於 C ++ 第 2 版（v 1.17.0 或更高版本）。
2. 按照自述文件中的安裝說明從源代碼構建 C ++ v2。AWS IoT Device SDK

3. 在您的 C ++ 構建工具中，鏈接您在上一步中構建的 Greengrass IPC 庫。AWS::GreengrassIpc-cpp下列CMakeLists.txt範例會將 Greengrass IPC 程式庫連結到您使用 CMake 建置的專案。

```
cmake_minimum_required(VERSION 3.1)
project (greengrassv2_pubsub_subscriber)

file(GLOB MAIN_SRC
        "*.h"
        "*.cpp"
        )
add_executable(${PROJECT_NAME} ${MAIN_SRC})

set_target_properties(${PROJECT_NAME} PROPERTIES
        LINKER_LANGUAGE CXX
        CXX_STANDARD 11)
find_package(aws-crt-cpp PATHS ~/sdk-cpp-workspace/build)
find_package(EventstreamRpc-cpp PATHS ~/sdk-cpp-workspace/build)
find_package(GreengrassIpc-cpp PATHS ~/sdk-cpp-workspace/build)
target_link_libraries(${PROJECT_NAME} AWS::GreengrassIpc-cpp)
```

4. 在元件程式碼中，建立與 AWS IoT Greengrass Core IPC 服務的連線，以建立 IPC 用戶端 () Aws::Greengrass::GreengrassCoreIpcClient。您必須定義處理 IPC 連線、中斷連線和錯誤事件的 IPC 連線生命週期處理常式。下列範例會建立 IPC 用戶端和 IPC 連線生命週期處理常式，此處理常式會在 IPC 用戶端連線、中斷連線及發生錯誤時列印。

```
#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        std::cout << "OnConnectCallback" << std::endl;
    }

    void OnDisconnectCallback(RpcError error) override {
        std::cout << "OnDisconnectCallback: " << error.StatusToString() <<
 std::endl;
```

```
        exit(-1);
    }

    bool OnErrorCallback(RpcError error) override {
        std::cout << "OnErrorCallback: " << error.StatusToString() << std::endl;
        return true;
    }
};

int main() {
    // Create the IPC client.
    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
 connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    // Use the IPC client to create an operation request.

    // Activate the operation request.
    auto activate = operation.Activate(request, nullptr);
    activate.wait();

    // Wait for Greengrass Core to respond to the request.
    auto responseFuture = operation.GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
 std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from
 Greengrass Core." << std::endl;
        exit(-1);
    }

    // Check the result of the request.
    auto response = responseFuture.get();
    if (response) {
        std::cout << "Successfully published to topic: " << topic << std::endl;
    } else {
```

```
        // An error occurred.
        std::cout << "Failed to publish to topic: " << topic << std::endl;
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            std::cout << "Operation error: " << error->GetMessage().value() <<
 std::endl;
        } else {
            std::cout << "RPC error: " << response.GetRpcError() << std::endl;
        }
        exit(-1);
    }

    return 0;
}
```

5. 若要在元件中執行自訂程式碼，請將程式碼建置為二進位成品，然後在元件方案中執行二進位成品。將成品的`Execute`權限設定`OWNER`為允許AWS IoT Greengrass核心軟體執行二進位成品。

   您的組件配方的`Manifests`部分看起來可能類似於以下示例。

   JSON

   ```
   {
     ...
     "Manifests": [
       {
         "Lifecycle": {
           "run": "{artifacts:path}/greengrassv2_pubsub_subscriber"
         },
         "Artifacts": [
           {
             "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
   com.example.PubSubSubscriberCpp/1.0.0/greengrassv2_pubsub_subscriber",
             "Permission": {
               "Execute": "OWNER"
             }
           }
         ]
       }
     ]
   }
   ```

YAML

```
...
Manifests:
  - Lifecycle:
      run: {artifacts:path}/greengrassv2_pubsub_subscriber
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.PubSubSubscriberCpp/1.0.0/greengrassv2_pubsub_subscriber
        Permission:
          Execute: OWNER
```

使AWS IoT Device SDK用 JavaScript V2 (IPC 用戶端 V1)

要構建 JavaScript v2 以與 NodeJS 一起使用，設備必須具有以下工具：AWS IoT Device SDK

- NodeJS JS 10.0 或更高版本
  - 執行node -v以檢查節點版本。
- 製造 3.1 或更高版本

若要使AWS IoT Device SDK用 JavaScript V2 (IPC 用戶端 V1)

1. 下載AWS IoT Device SDK適用於第 JavaScript 2 版 (1.12.10 版或更新版本)。
2. 按照 README 中的安裝說明從源代碼構建 AWS IoT Device SDK for JavaScript v2。
3. 建立與AWS IoT Greengrass核心 IPC 服務的連線。完成下列步驟以建立 IPC 用戶端並建立連線。
4. 使用下列程式碼建立 IPC 用戶端。

```
import * as greengrascoreipc from 'aws-iot-device-sdk-v2';

let client = greengrascoreipc.createClient();
```

5. 使用下面的代碼來建立從組件到 Greengrass 核的連接。

```
await client.connect();
```

授權元件執行 IPC 作業

若要允許自訂元件使用某些 IPC 作業，您必須定義允許元件在特定資源上執行作業的授權原則。每個授權原則都會定義作業清單以及原則允許的資源清單。例如，發佈/訂閱訊息 IPC 服務會定義主題資源的發佈和訂閱作業。您可以使用*萬用字元來允許存取所有作業或所有資源。

您可以使用`accessControl`組態參數來定義授權原則，您可以在元件方案中或部署元件時設定這些參數。`accessControl`物件會將 IPC 服務識別碼對應至授權原則清單。您可以為每個 IPC 服務定義多個授權原則，以控制存取。每個授權策略都有一個策略 ID，它在所有元件中必須是唯一的。

> **Tip**
>
> 若要建立唯一的原則識別碼，您可以結合元件名稱、IPC 服務名稱和計數器。例如，名為的元件`com.example.HelloWorld`可能會定義兩個具有下列 ID 的發佈/訂閱授權原則：
>
> - `com.example.HelloWorld:pubsub:1`
> - `com.example.HelloWorld:pubsub:2`

授權原則使用下列格式。這個對象是`accessControl`配置參數。

JSON

```
{
  "IPC service identifier": {
    "policyId": {
      "policyDescription": "description",
      "operations": [
        "operation1",
        "operation2"
      ],
      "resources": [
        "resource1",
        "resource2"
      ]
    }
  }
}
```

YAML

```
IPC service identifier:
  policyId:
    policyDescription: description
    operations:
      - operation1
      - operation2
    resources:
      - resource1
      - resource2
```

授權原則中的萬用字元

您可以在 IPC 授權原則的`resources`元素中使用`*`萬用字元，以允許存取單一授權原則中的多個資源。

- 在 Greengrass 核心的所有版本中，您可以指定單一`*`字元作為資源，以允許存取所有資源。
- 在 Greengrass 核 v2.6.0 及更高版本中，您可以在資源中指定字符以匹配任意`*`字符組合。例如，您可以指定`factory/1/devices/Thermostat*/status`允許存取工廠中所有恆溫器裝置的狀態主題，其中每個裝置的名稱開頭為`Thermostat`。

當您定義 AWS IoT Core MQTT IPC 服務的授權原則時，您也可以使用 MQTT 萬用字元 (+和#) 來比對多個資源。如需詳細資訊，請參閱 MQTT IPC 授權原則中的 AWS IoT Core MQTT 萬用字元。

授權政策中的配方變數

如果您使用 Greengrass 核心 v2.6.0 或更新版本，並將 `true` Greengrass 核心的`interpolateComponentConfiguration`組態選項設定為，您可以在授權原則中使用 recipe 變數。`{iot:thingName}`當您需要包含核心裝置名稱的授權原則時 (例如針對 MQTT 主題或裝置陰影)，您可以使用此 recipe 變數為一組核心裝置設定單一授權原則。例如，您可以允許元件存取下列資源，以進行陰影 IPC 作業。

```
$aws/things/{iot:thingName}/shadow/
```

授權政策中的特殊字元

若要在授權原則中指定常值*或?字元，您必須使用逸出序列。下列逸出序列會指示AWS IoT Greengrass核心軟體使用常值，而非字元的特殊意義。例如，*字元是萬用字元，符合任何字元組合。

常值字元	逸出序列	備註
*	${*}	
?	${?}	AWS IoT Greengrass目前不支援符合任何單一字?元的萬用字元。
$	${$}	使用此逸出序列來比對包含的資源${。例如，若要符合名為的資源${resourceName} ，您必須指定${$}{resourceName} 。否則，若要比對包含的資源$，您可以使用常值$，例如允許存取以開頭的主題$aws。

授權政策範例

您可以參考下列授權原則範例，協助您設定元件的授權原則。

Example 含授權原則的範例元件配方

下列範例元件方案包含定義授權原則的accessControl物件。此原則會授權com.example.HelloWorld元件發佈至主test/topic題。

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.HelloWorld",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that publishes messages.",
  "ComponentPublisher": "Amazon",
```

```
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.HelloWorld:pubsub:1": {
            "policyDescription": "Allows access to publish to test/topic.",
            "operations": [
              "aws.greengrass#PublishToTopic"
            ],
            "resources": [
              "test/topic"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Lifecycle": {
        "run": "java -jar {artifacts:path}/HelloWorld.jar"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.HelloWorld
ComponentVersion: '1.0.0'
ComponentDescription: A component that publishes messages.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        "com.example.HelloWorld:pubsub:1":
          policyDescription: Allows access to publish to test/topic.
          operations:
            - "aws.greengrass#PublishToTopic"
          resources:
```

```
            - "test/topic"
Manifests:
  - Lifecycle:
      run: |-
        java -jar {artifacts:path}/HelloWorld.jar
```

Example 使用授權原則更新元件組態範例

下列部署中的組態更新範例會指定使用定義授權原則的accessControl物件來設定元件。此原則會授權com.example.HelloWorld元件發佈至主test/topic題。

Console

要合併的組態

```
{
  "accessControl": {
    "aws.greengrass.ipc.pubsub": {
      "com.example.HelloWorld:pubsub:1": {
        "policyDescription": "Allows access to publish to test/topic.",
        "operations": [
          "aws.greengrass#PublishToTopic"
        ],
        "resources": [
          "test/topic"
        ]
      }
    }
  }
}
```

AWS CLI

下列指令會建立核心裝置的部署。

```
aws greengrassv2 create-deployment --cli-input-json file://hello-world-
deployment.json
```

該文hello-world-deployment.json件包含以下 JSON 文檔。

```
{
```

```
  "targetArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
  "deploymentName": "Deployment for MyGreengrassCore",
  "components": {
    "com.example.HelloWorld": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "merge": "{\"accessControl\":{\"aws.greengrass.ipc.pubsub\":
{\"com.example.HelloWorld:pubsub:1\":{\"policyDescription\":\"Allows access to
 publish to test/topic.\",\"operations\":[\"aws.greengrass#PublishToTopic\"],
\"resources\":[\"test/topic\"]}}}}"
      }
    }
  }
}
```

Greengrass CLI

以下 Greengrass CLI 命令會在核心裝置上建立本機部署。

```
sudo greengrass-cli deployment create \
  --recipeDir recipes \
  --artifactDir artifacts \
  --merge "com.example.HelloWorld=1.0.0" \
  --update-config hello-world-configuration.json
```

該文hello-world-configuration.json件包含以下 JSON 文檔。

```
{
  "com.example.HelloWorld": {
    "MERGE": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.HelloWorld:pubsub:1": {
            "policyDescription": "Allows access to publish to test/topic.",
            "operations": [
              "aws.greengrass#PublishToTopic"
            ],
            "resources": [
              "test/topic"
            ]
          }
        }
      }
```

```
    }
  }
}
```

訂閱 IPC 事件串流

您可以使用 IPC 操作來訂閱 Greengrass 核心設備上的事件流。若要使用訂閱作業，請定義訂閱處理常式，並建立 IPC 服務的要求。然後，每當核心裝置將事件訊息串流至您的元件時，IPC 用戶端就會執行訂閱處理常式的函數。

您可以關閉訂閱以停止處理事件訊息。若要這麼做，請`closeStream()`在您用來開啟訂閱的訂閱作業物件上呼叫 `Close()` (Java)、(Python) 或 (C++)。`close()`

AWS IoT Greengrass酷睿 IPC 服務支援下列訂閱作業：

- SubscribeToTopic
- SubscribeToIoTCore
- SubscribeToComponentUpdates
- SubscribeToConfigurationUpdate
- SubscribeToValidateConfigurationUpdates

主題

- 定義訂閱處理器
- 訂閱處理常式範

定義訂閱處理器

若要定義訂閱處理常式，請定義處理事件訊息、錯誤和串流結束的回呼函數。如果您使用 IPC 用戶端 V1，則必須在類別中定義這些函數。如果您使用 IPC 用戶端 V2 (在較新版本的 Java 和 Python SDK 中提供)，您可以在不建立訂閱處理常式類別的情況下定義這些函數。

Java

如果您使用 IPC 用戶端 V1，則必須實作一般`software.amazon.awssdk.eventstreamrpc.StreamResponseHandler<`*`StreamEventType`*`>`

面。*StreamEventType*是訂閱作業的事件訊息類型。定義下列函數以處理事件訊息、錯誤和資料流結束。

如果您使用 IPC 用戶端 V2，您可以在訂閱處理常式類別之外定義這些函數，或使用 Lambda 運算式。

```
void onStreamEvent(StreamEventType event)
```

IPC 用戶端在收到事件訊息 (例如 MQTT 訊息或元件更新通知) 時呼叫的回呼。

```
boolean onStreamError(Throwable error)
```

發生串流錯誤時，IPC 用戶端呼叫的回呼。

傳回 true 以因錯誤而關閉訂閱串流，或傳回 false 以保持串流開啟。

```
void onStreamClosed()
```

當串流關閉時，IPC 用戶端呼叫的回呼。

Python

如果您使用 IPC 用戶端 V1，則必須擴充與訂閱作業對應的串流回應處理常式類別。AWS IoT Device SDK包含每個訂閱作業的訂閱處理常式類別。*StreamEventType*是訂閱作業的事件訊息類型。定義下列函數以處理事件訊息、錯誤和資料流結束。

如果您使用 IPC 用戶端 V2，您可以在訂閱處理常式類別之外定義這些函數，或使用 Lambda 運算式。

```
def on_stream_event(self, event: StreamEventType) -> None
```

IPC 用戶端在收到事件訊息 (例如 MQTT 訊息或元件更新通知) 時呼叫的回呼。

```
def on_stream_error(self, error: Exception) -> bool
```

發生串流錯誤時，IPC 用戶端呼叫的回呼。

傳回 true 以因錯誤而關閉訂閱串流，或傳回 false 以保持串流開啟。

```
def on_stream_closed(self) -> None
```

當串流關閉時，IPC 用戶端呼叫的回呼。

C++

實作衍生自對應於訂閱作業之串流回應處理常式類別的類別。AWS IoT Device SDK包含每個訂閱作業的訂閱處理常式基底類別。*StreamEventType*是訂閱作業的事件訊息類型。定義下列函數以處理事件訊息、錯誤和資料流結束。

```
void OnStreamEvent(StreamEventType *event)
```

IPC 用戶端在收到事件訊息 (例如 MQTT 訊息或元件更新通知) 時呼叫的回呼。

```
bool OnStreamError(OperationError *error)
```

發生串流錯誤時，IPC 用戶端呼叫的回呼。

傳回 true 以因錯誤而關閉訂閱串流，或傳回 false 以保持串流開啟。

```
void OnStreamClosed()
```

當串流關閉時，IPC 用戶端呼叫的回呼。

JavaScript

實作衍生自對應於訂閱作業之串流回應處理常式類別的類別。AWS IoT Device SDK包含每個訂閱作業的訂閱處理常式基底類別。*StreamEventType*是訂閱作業的事件訊息類型。定義下列函數以處理事件訊息、錯誤和資料流結束。

```
on(event: 'ended', listener: StreamingOperationEndedListener)
```

當串流關閉時，IPC 用戶端呼叫的回呼。

```
on(event: 'streamError', listener: StreamingRpcErrorListener)
```

發生串流錯誤時，IPC 用戶端呼叫的回呼。

傳回 true 以因錯誤而關閉訂閱串流，或傳回 false 以保持串流開啟。

```
on(event: 'message', listener: (message: InboundMessageType) => void)
```

IPC 用戶端在收到事件訊息 (例如 MQTT 訊息或元件更新通知) 時呼叫的回呼。

訂閱處理常式範

下列範例示範如何使用SubscribeToTopic作業和訂閱處理常式來訂閱本機發佈/訂閱訊息。

Java (IPC client V2)

Example 範例：訂閱本機發佈/訂閱訊息

```
package com.aws.greengrass.docs.samples.ipc;

import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClientV2;
import software.amazon.awssdk.aws.greengrass.SubscribeToTopicResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.*;

import java.nio.charset.StandardCharsets;
import java.util.Optional;

public class SubscribeToTopicV2 {

    public static void main(String[] args) {
        String topic = args[0];
        try (GreengrassCoreIPCClientV2 ipcClient =
 GreengrassCoreIPCClientV2.builder().build()) {
            SubscribeToTopicRequest request = new
 SubscribeToTopicRequest().withTopic(topic);
            GreengrassCoreIPCClientV2.StreamingResponse<SubscribeToTopicResponse,
                    SubscribeToTopicResponseHandler> response =
                    ipcClient.subscribeToTopic(request,
 SubscribeToTopicV2::onStreamEvent,
                            Optional.of(SubscribeToTopicV2::onStreamError),
                            Optional.of(SubscribeToTopicV2::onStreamClosed));
            SubscribeToTopicResponseHandler responseHandler =
 response.getHandler();
            System.out.println("Successfully subscribed to topic: " + topic);

            // Keep the main thread alive, or the process will exit.
            try {
                while (true) {
                    Thread.sleep(10000);
                }
            } catch (InterruptedException e) {
                System.out.println("Subscribe interrupted.");
            }

            // To stop subscribing, close the stream.
            responseHandler.closeStream();
        } catch (Exception e) {
            if (e.getCause() instanceof UnauthorizedError) {
```

```
                System.err.println("Unauthorized error while publishing to topic: "
 + topic);
            } else {
                System.err.println("Exception occurred when using IPC.");
            }
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static void onStreamEvent(SubscriptionResponseMessage
 subscriptionResponseMessage) {
        try {
            BinaryMessage binaryMessage =
 subscriptionResponseMessage.getBinaryMessage();
            String message = new String(binaryMessage.getMessage(),
 StandardCharsets.UTF_8);
            String topic = binaryMessage.getContext().getTopic();
            System.out.printf("Received new message on topic %s: %s%n", topic,
 message);
        } catch (Exception e) {
            System.err.println("Exception occurred while processing subscription
 response " +
                    "message.");
            e.printStackTrace();
        }
    }

    public static boolean onStreamError(Throwable error) {
        System.err.println("Received a stream error.");
        error.printStackTrace();
        return false; // Return true to close stream, false to keep stream open.
    }

    public static void onStreamClosed() {
        System.out.println("Subscribe to topic stream closed.");
    }
}
```

Python (IPC client V2)

Example 範例：訂閱本機發佈/訂閱訊息

```
import sys
```

```
import time
import traceback

from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2
from awsiot.greengrasscoreipc.model import (
    SubscriptionResponseMessage,
    UnauthorizedError
)


def main():
    args = sys.argv[1:]
    topic = args[0]

    try:
        ipc_client = GreengrassCoreIPCClientV2()
        # Subscription operations return a tuple with the response and the
 operation.
        _, operation = ipc_client.subscribe_to_topic(topic=topic,
 on_stream_event=on_stream_event,

 on_stream_error=on_stream_error, on_stream_closed=on_stream_closed)
        print('Successfully subscribed to topic: ' + topic)

        # Keep the main thread alive, or the process will exit.
        try:
            while True:
                time.sleep(10)
        except InterruptedError:
            print('Subscribe interrupted.')

        # To stop subscribing, close the stream.
        operation.close()
    except UnauthorizedError:
        print('Unauthorized error while subscribing to topic: ' +
              topic, file=sys.stderr)
        traceback.print_exc()
        exit(1)
    except Exception:
        print('Exception occurred', file=sys.stderr)
        traceback.print_exc()
        exit(1)
```

```
def on_stream_event(event: SubscriptionResponseMessage) -> None:
    try:
        message = str(event.binary_message.message, 'utf-8')
        topic = event.binary_message.context.topic
        print('Received new message on topic %s: %s' % (topic, message))
    except:
        traceback.print_exc()


def on_stream_error(error: Exception) -> bool:
    print('Received a stream error.', file=sys.stderr)
    traceback.print_exc()
    return False  # Return True to close stream, False to keep stream open.


def on_stream_closed() -> None:
    print('Subscribe to topic stream closed.')


if __name__ == '__main__':
    main()
```

C++

Example 範例：訂閱本機發佈/訂閱訊息

```
#include <iostream>

#include </crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class SubscribeResponseHandler : public SubscribeToTopicStreamHandler {
    public:
        virtual ~SubscribeResponseHandler() {}

    private:
        void OnStreamEvent(SubscriptionResponseMessage *response) override {
            auto jsonMessage = response->GetJsonMessage();
            if (jsonMessage.has_value() &&
 jsonMessage.value().GetMessage().has_value()) {
```

```
                auto messageString =
 jsonMessage.value().GetMessage().value().View().WriteReadable();
                // Handle JSON message.
            } else {
                auto binaryMessage = response->GetBinaryMessage();
                if (binaryMessage.has_value() &&
 binaryMessage.value().GetMessage().has_value()) {
                    auto messageBytes = binaryMessage.value().GetMessage().value();
                    std::string messageString(messageBytes.begin(),
 messageBytes.end());
                    // Handle binary message.
                }
            }
        }

        bool OnStreamError(OperationError *error) override {
            // Handle error.
            return false; // Return true to close stream, false to keep stream open.
        }

        void OnStreamClosed() override {
            // Handle close.
        }
};

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        // Handle connection to IPC service.
    }

    void OnDisconnectCallback(RpcError error) override {
        // Handle disconnection from IPC service.
    }

    bool OnErrorCallback(RpcError error) override {
        // Handle IPC service connection error.
        return true;
    }
};

int main() {
    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
```

```
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    String topic("my/topic");
    int timeout = 10;

    SubscribeToTopicRequest request;
    request.SetTopic(topic);

    //SubscribeResponseHandler streamHandler;
    auto streamHandler = MakeShared<SubscribeResponseHandler>(DefaultAllocator());
    auto operation = ipcClient.NewSubscribeToTopic(streamHandler);
    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from Greengrass
Core." << std::endl;
        exit(-1);
    }

    auto response = responseFuture.get();
    if (!response) {
        // Handle error.
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            (void)error;
            // Handle operation error.
        } else {
            // Handle RPC error.
        }
        exit(-1);
    }
```

```
    // Keep the main thread alive, or the process will exit.
    while (true) {
        std::this_thread::sleep_for(std::chrono::seconds(10));
    }

    operation->Close();
    return 0;
}
```

JavaScript

Example 範例：訂閱本機發佈/訂閱訊息

```
import * as greengrasscoreipc from "aws-iot-device-sdk-v2/dist/greengrasscoreipc";
import {SubscribeToTopicRequest, SubscriptionResponseMessage} from "aws-iot-device-
sdk-v2/dist/greengrasscoreipc/model";
import {RpcError} from "aws-iot-device-sdk-v2/dist/eventstream_rpc";

class SubscribeToTopic {
    private ipcClient : greengrasscoreipc.Client
    private readonly topic : string;

    constructor() {
        // define your own constructor, e.g.
        this.topic = "<define_your_topic>";
        this.subscribeToTopic().then(r => console.log("Started workflow"));
    }

    private async subscribeToTopic() {
        try {
            this.ipcClient = await getIpcClient();

            const subscribeToTopicRequest : SubscribeToTopicRequest = {
                topic: this.topic,
            }

            const streamingOperation =
 this.ipcClient.subscribeToTopic(subscribeToTopicRequest, undefined); //
 conditionally apply options

            streamingOperation.on("message", (message: SubscriptionResponseMessage)
 => {
                // parse the message depending on your use cases, e.g.
```

```
                if(message.binaryMessage && message.binaryMessage.message) {
                    const receivedMessage =
 message.binaryMessage?.message.toString();
                }
            });

            streamingOperation.on("streamError", (error : RpcError) => {
                // define your own error handling logic
            })

            streamingOperation.on("ended", () => {
                // define your own logic
            })

            await streamingOperation.activate();

            // Keep the main thread alive, or the process will exit.
            await new Promise((resolve) => setTimeout(resolve, 10000))
        } catch (e) {
            // parse the error depending on your use cases
            throw e
        }
    }
}

export async function getIpcClient(){
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

// starting point
const subscribeToTopic = new SubscribeToTopic();
```

IPC 最佳做法

在自訂元件中使用 IPC 的最佳做法會因 IPC 用戶端 V1 和 IPC 用戶端 V2 而有所不同。請遵循您使用的 IPC 用戶端版本的最佳作法。

IPC client V2

IPC 用戶端 V2 會在個別的執行緒中執行回呼函式，因此與 IPC 用戶端 V1 相比，當您使用 IPC 和撰寫訂閱處理常式函數時，需要遵循的準則較少。

- 重複使用一個 IPC 用戶端

  建立 IPC 用戶端之後，請將其保持開啟狀態，並將其重複使用於所有 IPC 作業。創建多個客戶端會使用額外的資源，並可能導致資源洩漏。

- 處理異常

  IPC 用戶端 V2 會在訂閱處理常式函式中記錄未捕獲的例外狀況。您應該在處理常式函數中 catch 例外狀況，以處理程式碼中發生的錯誤。

IPC client V1

IPC 用戶端 V1 使用與 IPC 伺服器通訊的單一執行緒，並呼叫訂閱處理常式。撰寫訂閱處理常式函數時，必須考慮這種同步行為。

- 重複使用一個 IPC 用戶端

  建立 IPC 用戶端之後，請將其保持開啟狀態，並將其重複使用於所有 IPC 作業。創建多個客戶端會使用額外的資源，並可能導致資源洩漏。

- 非同步執行封鎖程式碼

  IPC 用戶端 V1 無法傳送新的要求或處理新的事件訊息，而執行緒被封鎖。您應該在從處理常式函數執行的個別執行緒中執行封鎖程式碼。封鎖程式碼包括`sleep`呼叫、持續執行的迴圈，以及需要時間才能完成的同步 I/O 要求。

- 以非同步方式傳送新的 IPC 要求

  IPC 用戶端 V1 無法從訂閱處理常式函式中傳送新的要求，因為如果您等待回應，要求會封鎖處理常式函式。您應該在從處理程序函數運行的單獨線程中發送 IPC 請求。

- 處理異常

IPC 用戶端 V1 不會處理訂閱處理常式函數中未捕捉到的例外狀況。如果您的處理常式函數擲回例外狀況，訂閱就會關閉，且例外狀況不會出現在元件記錄中。您應該在處理常式函數中 catch 例外狀況，以保持訂閱開啟狀態，並記錄程式碼中發生的錯誤。

發佈/訂閱本地訊息

發佈/訂閱 (pubsub) 訊息可讓您傳送和接收主題的訊息。組件可以將消息發佈到主題以將消息發送到其他組件。接著，訂閱該主題的元件就可以對其收到的訊息採取行動。

> **Note**
>
> 您不能使用此發布/訂閱 IPC 服務來發布或訂閱 MQTT。AWS IoT Core如需如何與 AWS IoT Core MQTT 交換郵件的詳細資訊，請參閱。發布/訂閱MQTT 訊 AWS IoT Core 息

主題

最低 SDK 版本

下表列出在本機主題中發行及訂閱訊息時，必須使用的最低版本。AWS IoT Device SDK

SDK	最低版本	
AWS IoT Device SDK對於 Java V2	v1.2.10	
AWS IoT Device SDK對於 Python V2	V1.5.3	

SDK	最低版本	
AWS IoT Device SDK對於 C++ 第 2	v1.17.0	
適用於 JavaScript 的 AWS IoT Device SDK v2	v1.12.0	

授權

若要在自訂元件中使用本機發佈/訂閱訊息，您必須定義允許元件傳送和接收訊息至主題的授權原則。如需定義授權原則的資訊，請參閱授權元件執行 IPC 作業。

發佈/訂閱訊息的授權原則具有下列屬性。

IPC 服務識別碼：`aws.greengrass.ipc.pubsub`

操作	描述	資源
`aws.greengrass#PublishToTopic`	允許元件將訊息發佈到您指定的主題。	主題字串，例如`test/topic` 。使用*來比對主題中任何字元組合。 此主題字串不支援 MQTT 主題萬用字元 (#和)。+
`aws.greengrass#SubscribeToTopic`	允許元件訂閱您指定主題的訊息。	主題字串，例如`test/topic` 。使用*來比對主題中任何字元組合。 在 Greengrass 核心 v2.6.0 及更新版本中，您可以訂閱包含 MQTT 主題萬用字元 (和) 的主題。# +此主題字串支援 MQTT 主題萬用字元做為常值字元。例如，如果元件的授權原則授與存取權`test/topic/#` ，則該元件可以訂閱`test/topi`

操作	描述	資源
		c/# ，但無法訂閱test/topic/filter 。
*	允許元件針對您指定的主題發佈和訂閱訊息。	主題字串，例如test/topic 。使用*來比對主題中任何字元組合。 在 Greengrass 核心 v2.6.0 及更新版本中，您可以訂閱包含 MQTT 主題萬用字元 (和) 的主題。# +此主題字串支援 MQTT 主題萬用字元做為常值字元。例如，如果元件的授權原則授與存取權test/topic/# ，則該元件可以訂閱test/topic/# ，但無法訂閱test/topic/filter 。

授權政策範例

您可以參考下列授權原則範例，協助您設定元件的授權原則。

Example 授權政策範例

下列範例授權原則可讓元件發佈和訂閱所有主題。

```
{
  "accessControl": {
    "aws.greengrass.ipc.pubsub": {
      "com.example.MyLocalPubSubComponent:pubsub:1": {
        "policyDescription": "Allows access to publish/subscribe to all topics.",
        "operations": [
          "aws.greengrass#PublishToTopic",
          "aws.greengrass#SubscribeToTopic"
        ],
        "resources": [
          "*"
        ]
```

```
      }
    }
  }
}
```

PublishToTopic

發布訊息至主題。

請求

此操作的請求具有以下參數：

`topic`

要發佈郵件的目標主題。

`publishMessage`(Python:`publish_message`)

要發佈的訊息。此物件`PublishMessage`包含下列資訊。您必須指定`jsonMessage`與之一`binaryMessage`。

`jsonMessage`(Python:`json_message`)

(選擇性) JSON 訊息。此物件`JsonMessage`包含下列資訊：

`message`

JSON 消息作為一個對象。

`context`

訊息的內容，例如發佈郵件的主題。

此功能適用於 v2.6.0 及更高版 Greeng rass 核組件。下表列出存取訊息前後關聯AWS IoT Device SDK所必須使用的最低版本。

SDK	最低版本	
AWS IoT Device SDK對於 Java V2	v1.9.3	
AWS IoT Device SDK對於 Python V2	v1.11.3	

SDK	最低版本	
AWS IoT Device SDK對於 C ++ 第 2	v1.18.4	
適用於 JavaScript 的 AWS IoT Device SDK v2	v1.12.0	

> **Note**
>
> AWS IoT Greengrass核心軟體在PublishToTopic和SubscribeToTopic作業中使用相同的訊息物件。當您訂閱時，AWS IoT GreengrassCore 軟體會在訊息中設定此前後關聯物件，並在您發佈的訊息中忽略此前後關聯物件。

此物件MessageContext包含下列資訊：

`topic`

發佈郵件的主題。

`binaryMessage`(Python:`binary_message`)

(選擇性) 二進位訊息。此物件BinaryMessage包含下列資訊：

`message`

二進制消息作為一個 blob。

`context`

訊息的內容，例如發佈郵件的主題。

此功能適用於 v2.6.0 及更高版 Greeng rass 核組件。下表列出存取訊息前後關聯AWS IoT Device SDK所必須使用的最低版本。

SDK	最低版本	
AWS IoT Device SDK對於 Java V2	v1.9.3	

SDK	最低版本	
AWS IoT Device SDK對於 Python V2	v1.11.3	
AWS IoT Device SDK對於 C ++ 第 2	v1.18.4	
適用於 JavaScript 的 AWS IoT Device SDK v2	v1.12.0	

> **Note**
>
> AWS IoT Greengrass核心軟體在PublishToTopic和SubscribeToTopic作業中使用相同的訊息物件。當您訂閱時，AWS IoT GreengrassCore 軟體會在訊息中設定此前後關聯物件，並在您發佈的訊息中忽略此前後關聯物件。

此物件MessageContext包含下列資訊：

`topic`

發佈郵件的主題。

回應

此操作在其響應中不提供任何信息。

範例

下列範例示範如何在自訂元件程式碼中呼叫此作業。

Java (IPC client V2)

Example 範例：發佈二進位訊息

```
package com.aws.greengrass.docs.samples.ipc;

import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClientV2;
import software.amazon.awssdk.aws.greengrass.model.BinaryMessage;
```

```
import software.amazon.awssdk.aws.greengrass.model.PublishMessage;
import software.amazon.awssdk.aws.greengrass.model.PublishToTopicRequest;
import software.amazon.awssdk.aws.greengrass.model.PublishToTopicResponse;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;

import java.nio.charset.StandardCharsets;

public class PublishToTopicV2 {

    public static void main(String[] args) {
        String topic = args[0];
        String message = args[1];
        try (GreengrassCoreIPCClientV2 ipcClient =
 GreengrassCoreIPCClientV2.builder().build()) {
            PublishToTopicV2.publishBinaryMessageToTopic(ipcClient, topic,
 message);
            System.out.println("Successfully published to topic: " + topic);
        } catch (Exception e) {
            if (e.getCause() instanceof UnauthorizedError) {
                System.err.println("Unauthorized error while publishing to topic: "
 + topic);
            } else {
                System.err.println("Exception occurred when using IPC.");
            }
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static PublishToTopicResponse publishBinaryMessageToTopic(
            GreengrassCoreIPCClientV2 ipcClient, String topic, String message)
 throws InterruptedException {
        BinaryMessage binaryMessage =
                new
 BinaryMessage().withMessage(message.getBytes(StandardCharsets.UTF_8));
        PublishMessage publishMessage = new
 PublishMessage().withBinaryMessage(binaryMessage);
        PublishToTopicRequest publishToTopicRequest =
                new
 PublishToTopicRequest().withTopic(topic).withPublishMessage(publishMessage);
        return ipcClient.publishToTopic(publishToTopicRequest);
    }
}
```

Python (IPC client V2)

Example 範例：發佈二進位訊息

```
import sys
import traceback

from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2
from awsiot.greengrasscoreipc.model import (
    PublishMessage,
    BinaryMessage
)


def main():
    args = sys.argv[1:]
    topic = args[0]
    message = args[1]

    try:
        ipc_client = GreengrassCoreIPCClientV2()
        publish_binary_message_to_topic(ipc_client, topic, message)
        print('Successfully published to topic: ' + topic)
    except Exception:
        print('Exception occurred', file=sys.stderr)
        traceback.print_exc()
        exit(1)


def publish_binary_message_to_topic(ipc_client, topic, message):
    binary_message = BinaryMessage(message=bytes(message, 'utf-8'))
    publish_message = PublishMessage(binary_message=binary_message)
    return ipc_client.publish_to_topic(topic=topic,
 publish_message=publish_message)


if __name__ == '__main__':
    main()
```

C++

Example 範例：發佈二進位訊息

```
#include <iostream>
```

```
#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        // Handle connection to IPC service.
    }

    void OnDisconnectCallback(RpcError error) override {
        // Handle disconnection from IPC service.
    }

    bool OnErrorCallback(RpcError error) override {
        // Handle IPC service connection error.
        return true;
    }
};

int main() {
    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
 connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    String topic("my/topic");
    String message("Hello, World!");
    int timeout = 10;

    PublishToTopicRequest request;
    Vector<uint8_t> messageData({message.begin(), message.end()});
    BinaryMessage binaryMessage;
    binaryMessage.SetMessage(messageData);
```

```
    PublishMessage publishMessage;
    publishMessage.SetBinaryMessage(binaryMessage);
    request.SetTopic(topic);
    request.SetPublishMessage(publishMessage);

    auto operation = ipcClient.NewPublishToTopic();
    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
 std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from Greengrass
 Core." << std::endl;
        exit(-1);
    }

    auto response = responseFuture.get();
    if (!response) {
        // Handle error.
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            (void)error;
            // Handle operation error.
        } else {
            // Handle RPC error.
        }
    }
    return 0;
}
```

JavaScript

Example 範例：發佈二進位訊息

```
import * as greengrasscoreipc from "aws-iot-device-sdk-v2/dist/greengrasscoreipc";
import {BinaryMessage, PublishMessage, PublishToTopicRequest} from "aws-iot-device-
sdk-v2/dist/greengrasscoreipc/model";

class PublishToTopic {
    private ipcClient : greengrasscoreipc.Client
    private readonly topic : string;
```

```
    private readonly messageString : string;

    constructor() {
        // define your own constructor, e.g.
        this.topic = "<define_your_topic>";
        this.messageString = "<define_your_message_string>";
        this.publishToTopic().then(r => console.log("Started workflow"));
    }

    private async publishToTopic() {
        try {
            this.ipcClient = await getIpcClient();

            const binaryMessage : BinaryMessage = {
                message: this.messageString
            }

            const publishMessage : PublishMessage = {
                binaryMessage: binaryMessage
            }

            const request : PublishToTopicRequest = {
                topic: this.topic,
                publishMessage: publishMessage
            }

            this.ipcClient.publishToTopic(request).finally(() =>
 console.log(`Published message ${publishMessage.binaryMessage?.message} to topic`))

        } catch (e) {
            // parse the error depending on your use cases
            throw e
        }
    }
}


export async function getIpcClient(){
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
```

```
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

// starting point
const publishToTopic = new PublishToTopic();
```

SubscribeToTopic

訂閱有關主題的訊息。

此作業是訂閱作業，您可以在其中訂閱事件訊息串流。若要使用此作業，請定義具有處理事件訊息、錯誤和資料流結束之函數的串流回應處理常式。如需詳細資訊，請參閱 訂閱 IPC 事件串流。

事件訊息類型：`SubscriptionResponseMessage`

請求

此操作的請求具有以下參數：

`topic`

要訂閱的主題。

> **Note**
>
> 在 Greengrass 核 v2.6.0 及更新版本中，本主題支援 MQTT 主題萬用字元 (和)。# +

`receiveMode`(Python:`receive_mode`)

(選擇性) 指定元件是否從本身接收訊息的行為。您可以變更此行為，以允許元件對自己的訊息執行動作。預設行為取決於主題是否包含 MQTT 萬用字元。您可以從以下選項中選擇：

- `RECEIVE_ALL_MESSAGES`— 接收符合主題的所有訊息，包括來自訂閱元件的訊息。

  當您訂閱不包含 MQTT 萬用字元的主題時，此模式為預設選項。

- `RECEIVE_MESSAGES_FROM_OTHERS`— 接收符合主題的所有訊息，但來自訂閱元件的訊息除外。

  當您訂閱包含 MQTT 萬用字元的主題時，此模式是預設選項。

此功能適用於 v2.6.0 及更高版 Greeng rass 核組件。下表列出設定接收模式AWS IoT Device SDK所必須使用的最低版本。

SDK	最低版本	
AWS IoT Device SDK對於 Java V2	v1.9.3	
AWS IoT Device SDK對於 Python V2	v1.11.3	
AWS IoT Device SDK對於 C++ 第 2	v1.18.4	
AWS IoT Device SDK對於 JavaScript V2	v1.12.0	

回應

此作業的回應包含下列資訊：

`messages`

訊息串流。此物件`SubscriptionResponseMessage`包含下列資訊。每個訊息都包含`jsonMessage`或`binaryMessage`。

`jsonMessage`(Python:`json_message`)

(選擇性) JSON 訊息。此物件`JsonMessage`包含下列資訊：

`message`

JSON 消息作為一個對象。

`context`

訊息的內容，例如發佈郵件的主題。

此功能適用於 v2.6.0 及更高版 Greeng rass 核組件。下表列出存取訊息前後關聯AWS IoT Device SDK所必須使用的最低版本。

SDK	最低版本	
AWS IoT Device SDK對於 Java V2	v1.9.3	
AWS IoT Device SDK對於 Python V2	v1.11.3	
AWS IoT Device SDK對於 C ++ 第 2	v1.18.4	
適用於 JavaScript 的 AWS IoT Device SDK v2	v1.12.0	

> **Note**
>
> AWS IoT Greengrass核心軟體在PublishToTopic和SubscribeToTopic作業中使用相同的訊息物件。當您訂閱時，AWS IoT GreengrassCore 軟體會在訊息中設定此前後關聯物件，並在您發佈的訊息中忽略此前後關聯物件。

此物件MessageContext包含下列資訊：

`topic`

發佈郵件的主題。

`binaryMessage`(Python:`binary_message`)

(選擇性) 二進位訊息。此物件BinaryMessage包含下列資訊：

`message`

二進制消息作為一個 blob。

`context`

訊息的內容，例如發佈郵件的主題。

此功能適用於 v2.6.0 及更高版 Greeng rass 核組件。下表列出存取訊息前後關聯AWS IoT Device SDK所必須使用的最低版本。

SDK	最低版本	
AWS IoT Device SDK對於 Java V2	v1.9.3	
AWS IoT Device SDK對於 Python V2	v1.11.3	
AWS IoT Device SDK對於 C ++ 第 2	v1.18.4	
適用於 JavaScript 的 AWS IoT Device SDK v2	v1.12.0	

> **Note**
>
> AWS IoT Greengrass核心軟體在PublishToTopic和SubscribeToTopic作業中使用相同的訊息物件。當您訂閱時，AWS IoT GreengrassCore 軟體會在訊息中設定此前後關聯物件，並在您發佈的訊息中忽略此前後關聯物件。

此物件MessageContext包含下列資訊：

`topic`

發佈郵件的主題。

`topicName`(Python:`topic_name`)

郵件發行目標的主題。

> **Note**
>
> 此屬性目前未使用。在 Greengrass 核 v2.6.0 及更高版本中，您可以從 a SubscriptionResponseMessage 中獲取(jsonMessage|binaryMessage).context.topic值以獲取發布消息的主題。

範例

下列範例示範如何在自訂元件程式碼中呼叫此作業。

Java (IPC client V2)

Example 範例：訂閱本地發佈/訂閱訊息

```
package com.aws.greengrass.docs.samples.ipc;

import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClientV2;
import software.amazon.awssdk.aws.greengrass.SubscribeToTopicResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.*;

import java.nio.charset.StandardCharsets;
import java.util.Optional;

public class SubscribeToTopicV2 {

    public static void main(String[] args) {
        String topic = args[0];
        try (GreengrassCoreIPCClientV2 ipcClient =
 GreengrassCoreIPCClientV2.builder().build()) {
            SubscribeToTopicRequest request = new
 SubscribeToTopicRequest().withTopic(topic);
            GreengrassCoreIPCClientV2.StreamingResponse<SubscribeToTopicResponse,
                    SubscribeToTopicResponseHandler> response =
                    ipcClient.subscribeToTopic(request,
 SubscribeToTopicV2::onStreamEvent,
                            Optional.of(SubscribeToTopicV2::onStreamError),
                            Optional.of(SubscribeToTopicV2::onStreamClosed));
            SubscribeToTopicResponseHandler responseHandler =
 response.getHandler();
            System.out.println("Successfully subscribed to topic: " + topic);

            // Keep the main thread alive, or the process will exit.
            try {
                while (true) {
                    Thread.sleep(10000);
                }
            } catch (InterruptedException e) {
                System.out.println("Subscribe interrupted.");
            }
```

```
            // To stop subscribing, close the stream.
            responseHandler.closeStream();
        } catch (Exception e) {
            if (e.getCause() instanceof UnauthorizedError) {
                System.err.println("Unauthorized error while publishing to topic: "
 + topic);
            } else {
                System.err.println("Exception occurred when using IPC.");
            }
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static void onStreamEvent(SubscriptionResponseMessage
 subscriptionResponseMessage) {
        try {
            BinaryMessage binaryMessage =
 subscriptionResponseMessage.getBinaryMessage();
            String message = new String(binaryMessage.getMessage(),
 StandardCharsets.UTF_8);
            String topic = binaryMessage.getContext().getTopic();
            System.out.printf("Received new message on topic %s: %s%n", topic,
 message);
        } catch (Exception e) {
            System.err.println("Exception occurred while processing subscription
 response " +
                    "message.");
            e.printStackTrace();
        }
    }

    public static boolean onStreamError(Throwable error) {
        System.err.println("Received a stream error.");
        error.printStackTrace();
        return false; // Return true to close stream, false to keep stream open.
    }

    public static void onStreamClosed() {
        System.out.println("Subscribe to topic stream closed.");
    }
}
```

Python (IPC client V2)

Example 範例：訂閱本地發佈/訂閱訊息

```
import sys
import time
import traceback

from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2
from awsiot.greengrasscoreipc.model import (
    SubscriptionResponseMessage,
    UnauthorizedError
)


def main():
    args = sys.argv[1:]
    topic = args[0]

    try:
        ipc_client = GreengrassCoreIPCClientV2()
        # Subscription operations return a tuple with the response and the
 operation.
        _, operation = ipc_client.subscribe_to_topic(topic=topic,
 on_stream_event=on_stream_event,

 on_stream_error=on_stream_error, on_stream_closed=on_stream_closed)
        print('Successfully subscribed to topic: ' + topic)

        # Keep the main thread alive, or the process will exit.
        try:
            while True:
                time.sleep(10)
        except InterruptedError:
            print('Subscribe interrupted.')

        # To stop subscribing, close the stream.
        operation.close()
    except UnauthorizedError:
        print('Unauthorized error while subscribing to topic: ' +
              topic, file=sys.stderr)
        traceback.print_exc()
        exit(1)
    except Exception:
```

```
        print('Exception occurred', file=sys.stderr)
        traceback.print_exc()
        exit(1)


def on_stream_event(event: SubscriptionResponseMessage) -> None:
    try:
        message = str(event.binary_message.message, 'utf-8')
        topic = event.binary_message.context.topic
        print('Received new message on topic %s: %s' % (topic, message))
    except:
        traceback.print_exc()


def on_stream_error(error: Exception) -> bool:
    print('Received a stream error.', file=sys.stderr)
    traceback.print_exc()
    return False  # Return True to close stream, False to keep stream open.


def on_stream_closed() -> None:
    print('Subscribe to topic stream closed.')


if __name__ == '__main__':
    main()
```

C++

Example 範例：訂閱本地發佈/訂閱訊息

```
#include <iostream>

#include </crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class SubscribeResponseHandler : public SubscribeToTopicStreamHandler {
    public:
        virtual ~SubscribeResponseHandler() {}

    private:
```

```
        void OnStreamEvent(SubscriptionResponseMessage *response) override {
            auto jsonMessage = response->GetJsonMessage();
            if (jsonMessage.has_value() &&
 jsonMessage.value().GetMessage().has_value()) {
                auto messageString =
 jsonMessage.value().GetMessage().value().View().WriteReadable();
                // Handle JSON message.
            } else {
                auto binaryMessage = response->GetBinaryMessage();
                if (binaryMessage.has_value() &&
 binaryMessage.value().GetMessage().has_value()) {
                    auto messageBytes = binaryMessage.value().GetMessage().value();
                    std::string messageString(messageBytes.begin(),
 messageBytes.end());
                    // Handle binary message.
                }
            }
        }

        bool OnStreamError(OperationError *error) override {
            // Handle error.
            return false; // Return true to close stream, false to keep stream open.
        }

        void OnStreamClosed() override {
            // Handle close.
        }
};

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        // Handle connection to IPC service.
    }

    void OnDisconnectCallback(RpcError error) override {
        // Handle disconnection from IPC service.
    }

    bool OnErrorCallback(RpcError error) override {
        // Handle IPC service connection error.
        return true;
    }
};
```

```
int main() {
    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
 connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    String topic("my/topic");
    int timeout = 10;

    SubscribeToTopicRequest request;
    request.SetTopic(topic);

    //SubscribeResponseHandler streamHandler;
    auto streamHandler = MakeShared<SubscribeResponseHandler>(DefaultAllocator());
    auto operation = ipcClient.NewSubscribeToTopic(streamHandler);
    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
 std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from Greengrass
 Core." << std::endl;
        exit(-1);
    }

    auto response = responseFuture.get();
    if (!response) {
        // Handle error.
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            (void)error;
            // Handle operation error.
        } else {
            // Handle RPC error.
```

```
        }
        exit(-1);
    }

    // Keep the main thread alive, or the process will exit.
    while (true) {
        std::this_thread::sleep_for(std::chrono::seconds(10));
    }

    operation->Close();
    return 0;
}
```

JavaScript

Example 範例：訂閱本地發佈/訂閱訊息

```
import * as greengrasscoreipc from "aws-iot-device-sdk-v2/dist/greengrasscoreipc";
import {SubscribeToTopicRequest, SubscriptionResponseMessage} from "aws-iot-device-
sdk-v2/dist/greengrasscoreipc/model";
import {RpcError} from "aws-iot-device-sdk-v2/dist/eventstream_rpc";

class SubscribeToTopic {
    private ipcClient : greengrasscoreipc.Client
    private readonly topic : string;

    constructor() {
        // define your own constructor, e.g.
        this.topic = "<define_your_topic>";
        this.subscribeToTopic().then(r => console.log("Started workflow"));
    }

    private async subscribeToTopic() {
        try {
            this.ipcClient = await getIpcClient();

            const subscribeToTopicRequest : SubscribeToTopicRequest = {
                topic: this.topic,
            }

            const streamingOperation =
 this.ipcClient.subscribeToTopic(subscribeToTopicRequest, undefined); //
 conditionally apply options
```

```
            streamingOperation.on("message", (message: SubscriptionResponseMessage)
 => {
                // parse the message depending on your use cases, e.g.
                if(message.binaryMessage && message.binaryMessage.message) {
                    const receivedMessage =
 message.binaryMessage?.message.toString();
                }
            });

            streamingOperation.on("streamError", (error : RpcError) => {
                // define your own error handling logic
            })

            streamingOperation.on("ended", () => {
                // define your own logic
            })

            await streamingOperation.activate();

            // Keep the main thread alive, or the process will exit.
            await new Promise((resolve) => setTimeout(resolve, 10000))
        } catch (e) {
            // parse the error depending on your use cases
            throw e
        }
    }
}

export async function getIpcClient(){
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}
```

```
// starting point
const subscribeToTopic = new SubscribeToTopic();
```

範例

使用下列範例，瞭解如何在元件中使用發佈/訂閱 IPC 服務。

發布/訂閱發佈者範例 (Java、IPC 用戶端 V1)

下列範例方案可讓元件發佈至所有主題。

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubPublisherJava",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that publishes messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubPublisherJava:pubsub:1": {
            "policyDescription": "Allows access to publish to all topics.",
            "operations": [
              "aws.greengrass#PublishToTopic"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Lifecycle": {
        "run": "java -jar {artifacts:path}/PubSubPublisher.jar"
      }
    }
  ]
```

```
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PubSubPublisherJava
ComponentVersion: '1.0.0'
ComponentDescription: A component that publishes messages.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        'com.example.PubSubPublisherJava:pubsub:1':
          policyDescription: Allows access to publish to all topics.
          operations:
            - 'aws.greengrass#PublishToTopic'
          resources:
            - '*'
Manifests:
  - Lifecycle:
      run: |-
        java -jar {artifacts:path}/PubSubPublisher.jar
```

下列範例 Java 應用程式示範如何使用發佈/訂閱 IPC 服務，將訊息發佈至其他元件。

```
/* Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: Apache-2.0 */

package com.example.ipc.pubsub;

import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.model.*;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;

import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;
```

```
public class PubSubPublisher {

    public static void main(String[] args) {
        String message = "Hello from the pub/sub publisher (Java).";
        String topic = "test/topic/java";

        try (EventStreamRPCConnection eventStreamRPCConnection =
 IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient = new
 GreengrassCoreIPCClient(eventStreamRPCConnection);

            while (true) {
                PublishToTopicRequest publishRequest = new PublishToTopicRequest();
                PublishMessage publishMessage = new PublishMessage();
                BinaryMessage binaryMessage = new BinaryMessage();
                binaryMessage.setMessage(message.getBytes(StandardCharsets.UTF_8));
                publishMessage.setBinaryMessage(binaryMessage);
                publishRequest.setPublishMessage(publishMessage);
                publishRequest.setTopic(topic);
                CompletableFuture<PublishToTopicResponse> futureResponse = ipcClient
                        .publishToTopic(publishRequest,
 Optional.empty()).getResponse();

                try {
                    futureResponse.get(10, TimeUnit.SECONDS);
                    System.out.println("Successfully published to topic: " + topic);
                } catch (TimeoutException e) {
                    System.err.println("Timeout occurred while publishing to topic: " +
 topic);
                } catch (ExecutionException e) {
                    if (e.getCause() instanceof UnauthorizedError) {
                        System.err.println("Unauthorized error while publishing to
 topic: " + topic);
                    } else {
                        System.err.println("Execution exception while publishing to
 topic: " + topic);
                    }
                    throw e;
                }
                Thread.sleep(5000);
            }
        } catch (InterruptedException e) {
            System.out.println("Publisher interrupted.");
```

```
        } catch (Exception e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }
}
```

發布/訂閱用戶的示例（Java，IPC 客戶端 V1）

下列範例方案可讓元件訂閱所有主題。

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubSubscriberJava",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that subscribes to messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubSubscriberJava:pubsub:1": {
            "policyDescription": "Allows access to subscribe to all topics.",
            "operations": [
              "aws.greengrass#SubscribeToTopic"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Lifecycle": {
        "run": "java -jar {artifacts:path}/PubSubSubscriber.jar"
      }
    }
  ]
```

```
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PubSubSubscriberJava
ComponentVersion: '1.0.0'
ComponentDescription: A component that subscribes to messages.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        'com.example.PubSubSubscriberJava:pubsub:1':
          policyDescription: Allows access to subscribe to all topics.
          operations:
            - 'aws.greengrass#SubscribeToTopic'
          resources:
            - '*'
Manifests:
  - Lifecycle:
      run: |-
        java -jar {artifacts:path}/PubSubSubscriber.jar
```

下列範例 Java 應用程式示範如何使用發佈/訂閱 IPC 服務來訂閱其他元件的訊息。

```
/* Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: Apache-2.0 */

package com.example.ipc.pubsub;

import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.SubscribeToTopicResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.SubscribeToTopicRequest;
import software.amazon.awssdk.aws.greengrass.model.SubscribeToTopicResponse;
import software.amazon.awssdk.aws.greengrass.model.SubscriptionResponseMessage;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;
import software.amazon.awssdk.eventstreamrpc.StreamResponseHandler;

import java.nio.charset.StandardCharsets;
```

```
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class PubSubSubscriber {

    public static void main(String[] args) {
        String topic = "test/topic/java";

        try (EventStreamRPCConnection eventStreamRPCConnection =
 IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient = new
 GreengrassCoreIPCClient(eventStreamRPCConnection);

            SubscribeToTopicRequest subscribeRequest = new SubscribeToTopicRequest();
            subscribeRequest.setTopic(topic);
            SubscribeToTopicResponseHandler operationResponseHandler = ipcClient
                    .subscribeToTopic(subscribeRequest, Optional.of(new
 SubscribeResponseHandler()));
            CompletableFuture<SubscribeToTopicResponse> futureResponse =
 operationResponseHandler.getResponse();

            try {
                futureResponse.get(10, TimeUnit.SECONDS);
                System.out.println("Successfully subscribed to topic: " + topic);
            } catch (TimeoutException e) {
                System.err.println("Timeout occurred while subscribing to topic: " +
 topic);
                throw e;
            } catch (ExecutionException e) {
                if (e.getCause() instanceof UnauthorizedError) {
                    System.err.println("Unauthorized error while subscribing to topic:
 " + topic);
                } else {
                    System.err.println("Execution exception while subscribing to topic:
 " + topic);
                }
                throw e;
            }

            // Keep the main thread alive, or the process will exit.
            try {
```

```
                while (true) {
                    Thread.sleep(10000);
                }
            } catch (InterruptedException e) {
                System.out.println("Subscribe interrupted.");
            }
        } catch (Exception e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }

    private static class SubscribeResponseHandler implements
 StreamResponseHandler<SubscriptionResponseMessage> {

        @Override
        public void onStreamEvent(SubscriptionResponseMessage
 subscriptionResponseMessage) {
            try {
                String message = new
 String(subscriptionResponseMessage.getBinaryMessage()
                        .getMessage(), StandardCharsets.UTF_8);
                System.out.println("Received new message: " + message);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }

        @Override
        public boolean onStreamError(Throwable error) {
            System.err.println("Received a stream error.");
            error.printStackTrace();
            return false; // Return true to close stream, false to keep stream open.
        }

        @Override
        public void onStreamClosed() {
            System.out.println("Subscribe to topic stream closed.");
        }
    }
}
```

發布/訂閱發布的示例（Python，IPC 客戶端 V1）

下列範例方案可讓元件發佈至所有主題。

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubPublisherPython",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that publishes messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubPublisherPython:pubsub:1": {
            "policyDescription": "Allows access to publish to all topics.",
            "operations": [
              "aws.greengrass#PublishToTopic"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "python3 -m pip install --user awsiotsdk",
        "run": "python3 -u {artifacts:path}/pubsub_publisher.py"
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
```

```
        "install": "py -3 -m pip install --user awsiotsdk",
        "run": "py -3 -u {artifacts:path}/pubsub_publisher.py"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PubSubPublisherPython
ComponentVersion: 1.0.0
ComponentDescription: A component that publishes messages.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        com.example.PubSubPublisherPython:pubsub:1:
          policyDescription: Allows access to publish to all topics.
          operations:
            - aws.greengrass#PublishToTopic
          resources:
            - "*"
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: python3 -m pip install --user awsiotsdk
      run: python3 -u {artifacts:path}/pubsub_publisher.py
  - Platform:
      os: windows
    Lifecycle:
      install: py -3 -m pip install --user awsiotsdk
      run: py -3 -u {artifacts:path}/pubsub_publisher.py
```

以下示例 Python 應用程序演示瞭如何使用發布/訂閱 IPC 服務將消息發布到其他組件。

```
import concurrent.futures
import sys
import time
```

```
import traceback

import awsiot.greengrasscoreipc
from awsiot.greengrasscoreipc.model import (
    PublishToTopicRequest,
    PublishMessage,
    BinaryMessage,
    UnauthorizedError
)


topic = "test/topic/python"
message = "Hello from the pub/sub publisher (Python)."
TIMEOUT = 10

try:
    ipc_client = awsiot.greengrasscoreipc.connect()

    while True:
        request = PublishToTopicRequest()
        request.topic = topic
        publish_message = PublishMessage()
        publish_message.binary_message = BinaryMessage()
        publish_message.binary_message.message = bytes(message, "utf-8")
        request.publish_message = publish_message
        operation = ipc_client.new_publish_to_topic()
        operation.activate(request)
        future_response = operation.get_response()

        try:
            future_response.result(TIMEOUT)
            print('Successfully published to topic: ' + topic)
        except concurrent.futures.TimeoutError:
            print('Timeout occurred while publishing to topic: ' + topic,
 file=sys.stderr)
        except UnauthorizedError as e:
            print('Unauthorized error while publishing to topic: ' + topic,
 file=sys.stderr)
            raise e
        except Exception as e:
            print('Exception while publishing to topic: ' + topic, file=sys.stderr)
            raise e
        time.sleep(5)
except InterruptedError:
```

```
    print('Publisher interrupted.')
except Exception:
    print('Exception occurred when using IPC.', file=sys.stderr)
    traceback.print_exc()
    exit(1)
```

發布/訂閱用戶的示例（Python，IPC 客戶端 V1）

下列範例方案可讓元件訂閱所有主題。

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubSubscriberPython",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that subscribes to messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubSubscriberPython:pubsub:1": {
            "policyDescription": "Allows access to subscribe to all topics.",
            "operations": [
              "aws.greengrass#SubscribeToTopic"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "python3 -m pip install --user awsiotsdk",
        "run": "python3 -u {artifacts:path}/pubsub_subscriber.py"
      }
```

```
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "py -3 -m pip install --user awsiotsdk",
        "run": "py -3 -u {artifacts:path}/pubsub_subscriber.py"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PubSubSubscriberPython
ComponentVersion: 1.0.0
ComponentDescription: A component that subscribes to messages.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        com.example.PubSubSubscriberPython:pubsub:1:
          policyDescription: Allows access to subscribe to all topics.
          operations:
            - aws.greengrass#SubscribeToTopic
          resources:
            - "*"
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: python3 -m pip install --user awsiotsdk
      run: python3 -u {artifacts:path}/pubsub_subscriber.py
  - Platform:
      os: windows
    Lifecycle:
      install: py -3 -m pip install --user awsiotsdk
      run: py -3 -u {artifacts:path}/pubsub_subscriber.py
```

以下示例 Python 應用程序演示瞭如何使用發布/訂閱 IPC 服務來訂閱消息到其他組件。

```
import concurrent.futures
import sys
import time
import traceback

import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import (
    SubscribeToTopicRequest,
    SubscriptionResponseMessage,
    UnauthorizedError
)

topic = "test/topic/python"
TIMEOUT = 10


class StreamHandler(client.SubscribeToTopicStreamHandler):
    def __init__(self):
        super().__init__()

    def on_stream_event(self, event: SubscriptionResponseMessage) -> None:
        try:
            message = str(event.binary_message.message, "utf-8")
            print("Received new message: " + message)
        except:
            traceback.print_exc()

    def on_stream_error(self, error: Exception) -> bool:
        print("Received a stream error.", file=sys.stderr)
        traceback.print_exc()
        return False  # Return True to close stream, False to keep stream open.

    def on_stream_closed(self) -> None:
        print('Subscribe to topic stream closed.')


try:
    ipc_client = awsiot.greengrasscoreipc.connect()

    request = SubscribeToTopicRequest()
    request.topic = topic
```

```
    handler = StreamHandler()
    operation = ipc_client.new_subscribe_to_topic(handler)
    operation.activate(request)
    future_response = operation.get_response()

    try:
        future_response.result(TIMEOUT)
        print('Successfully subscribed to topic: ' + topic)
    except concurrent.futures.TimeoutError as e:
        print('Timeout occurred while subscribing to topic: ' + topic,
 file=sys.stderr)
        raise e
    except UnauthorizedError as e:
        print('Unauthorized error while subscribing to topic: ' + topic,
 file=sys.stderr)
        raise e
    except Exception as e:
        print('Exception while subscribing to topic: ' + topic, file=sys.stderr)
        raise e

    # Keep the main thread alive, or the process will exit.
    try:
        while True:
            time.sleep(10)
    except InterruptedError:
        print('Subscribe interrupted.')
except Exception:
    print('Exception occurred when using IPC.', file=sys.stderr)
    traceback.print_exc()
    exit(1)
```

發布/訂閱發佈者範例 (C++)

下列範例方案可讓元件發佈至所有主題。

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubPublisherCpp",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that publishes messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
```

```
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubPublisherCpp:pubsub:1": {
            "policyDescription": "Allows access to publish to all topics.",
            "operations": [
              "aws.greengrass#PublishToTopic"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Lifecycle": {
        "run": "{artifacts:path}/greengrassv2_pubsub_publisher"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.PubSubPublisherCpp/1.0.0/greengrassv2_pubsub_publisher",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PubSubPublisherCpp
ComponentVersion: 1.0.0
ComponentDescription: A component that publishes messages.
ComponentPublisher: Amazon
ComponentConfiguration:
```

```
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        com.example.PubSubPublisherCpp:pubsub:1:
          policyDescription: Allows access to publish to all topics.
          operations:
            - aws.greengrass#PublishToTopic
          resources:
            - "*"
Manifests:
  - Lifecycle:
      run: "{artifacts:path}/greengrassv2_pubsub_publisher"
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.PubSubPublisherCpp/1.0.0/
greengrassv2_pubsub_publisher
        Permission:
          Execute: OWNER
```

下列範例 C++ 應用程式示範如何使用發佈/訂閱 IPC 服務，將訊息發佈至其他元件。

```
#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        std::cout << "OnConnectCallback" << std::endl;
    }

    void OnDisconnectCallback(RpcError error) override {
        std::cout << "OnDisconnectCallback: " << error.StatusToString() << std::endl;
        exit(-1);
    }

    bool OnErrorCallback(RpcError error) override {
        std::cout << "OnErrorCallback: " << error.StatusToString() << std::endl;
        return true;
    }
```

```
};

int main() {
    String message("Hello from the pub/sub publisher (C++).");
    String topic("test/topic/cpp");
    int timeout = 10;

    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
 connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    while (true) {
        PublishToTopicRequest request;
        Vector<uint8_t> messageData({message.begin(), message.end()});
        BinaryMessage binaryMessage;
        binaryMessage.SetMessage(messageData);
        PublishMessage publishMessage;
        publishMessage.SetBinaryMessage(binaryMessage);
        request.SetTopic(topic);
        request.SetPublishMessage(publishMessage);

        auto operation = ipcClient.NewPublishToTopic();
        auto activate = operation->Activate(request, nullptr);
        activate.wait();

        auto responseFuture = operation->GetResult();
        if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
 std::future_status::timeout) {
            std::cerr << "Operation timed out while waiting for response from
 Greengrass Core." << std::endl;
            exit(-1);
        }

        auto response = responseFuture.get();
        if (response) {
```

```
            std::cout << "Successfully published to topic: " << topic << std::endl;
        } else {
            // An error occurred.
            std::cout << "Failed to publish to topic: " << topic << std::endl;
            auto errorType = response.GetResultType();
            if (errorType == OPERATION_ERROR) {
                auto *error = response.GetOperationError();
                std::cout << "Operation error: " << error->GetMessage().value() <<
 std::endl;
            } else {
                std::cout << "RPC error: " << response.GetRpcError() << std::endl;
            }
            exit(-1);
        }

        std::this_thread::sleep_for(std::chrono::seconds(5));
    }

    return 0;
}
```

發布/訂閱用戶示例（C ++）

下列範例方案可讓元件訂閱所有主題。

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubSubscriberCpp",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that subscribes to messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubSubscriberCpp:pubsub:1": {
            "policyDescription": "Allows access to subscribe to all topics.",
            "operations": [
              "aws.greengrass#SubscribeToTopic"
            ],
            "resources": [
              "*"
```

```
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Lifecycle": {
        "run": "{artifacts:path}/greengrassv2_pub_sub_subscriber"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.PubSubSubscriberCpp/1.0.0/greengrassv2_pub_sub_subscriber",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PubSubSubscriberCpp
ComponentVersion: 1.0.0
ComponentDescription: A component that subscribes to messages.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        com.example.PubSubSubscriberCpp:pubsub:1:
          policyDescription: Allows access to subscribe to all topics.
          operations:
            - aws.greengrass#SubscribeToTopic
          resources:
            - "*"
Manifests:
```

```
  - Lifecycle:
      run: "{artifacts:path}/greengrassv2_pub_sub_subscriber"
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.PubSubSubscriberCpp/1.0.0/greengrassv2_pub_sub_subscriber
        Permission:
          Execute: OWNER
```

下列範例 C++ 應用程式示範如何使用發佈/訂閱 IPC 服務來訂閱其他元件的訊息。

```
#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class SubscribeResponseHandler : public SubscribeToTopicStreamHandler {
    public:
        virtual ~SubscribeResponseHandler() {}

    private:
        void OnStreamEvent(SubscriptionResponseMessage *response) override {
            auto jsonMessage = response->GetJsonMessage();
            if (jsonMessage.has_value() &&
 jsonMessage.value().GetMessage().has_value()) {
                auto messageString =
 jsonMessage.value().GetMessage().value().View().WriteReadable();
                std::cout << "Received new message: " << messageString << std::endl;
            } else {
                auto binaryMessage = response->GetBinaryMessage();
                if (binaryMessage.has_value() &&
 binaryMessage.value().GetMessage().has_value()) {
                    auto messageBytes = binaryMessage.value().GetMessage().value();
                    std::string messageString(messageBytes.begin(),
 messageBytes.end());
                    std::cout << "Received new message: " << messageString <<
 std::endl;
                }
            }
        }
```

```
        bool OnStreamError(OperationError *error) override {
            std::cout << "Received an operation error: ";
            if (error->GetMessage().has_value()) {
                std::cout << error->GetMessage().value();
            }
            std::cout << std::endl;
            return false; // Return true to close stream, false to keep stream open.
        }

        void OnStreamClosed() override {
            std::cout << "Subscribe to topic stream closed." << std::endl;
        }
};

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        std::cout << "OnConnectCallback" << std::endl;
    }

    void OnDisconnectCallback(RpcError error) override {
        std::cout << "OnDisconnectCallback: " << error.StatusToString() << std::endl;
        exit(-1);
    }

    bool OnErrorCallback(RpcError error) override {
        std::cout << "OnErrorCallback: " << error.StatusToString() << std::endl;
        return true;
    }
};

int main() {
    String topic("test/topic/cpp");
    int timeout = 10;

    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
```

```
        std::cerr << "Failed to establish IPC connection: " <<
connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    SubscribeToTopicRequest request;
    request.SetTopic(topic);
    auto streamHandler = MakeShared<SubscribeResponseHandler>(DefaultAllocator());
    auto operation = ipcClient.NewSubscribeToTopic(streamHandler);
    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from Greengrass
Core." << std::endl;
        exit(-1);
    }

    auto response = responseFuture.get();
    if (response) {
        std::cout << "Successfully subscribed to topic: " << topic << std::endl;
    } else {
        // An error occurred.
        std::cout << "Failed to subscribe to topic: " << topic << std::endl;
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            std::cout << "Operation error: " << error->GetMessage().value() <<
std::endl;
        } else {
            std::cout << "RPC error: " << response.GetRpcError() << std::endl;
        }
        exit(-1);
    }

    // Keep the main thread alive, or the process will exit.
    while (true) {
        std::this_thread::sleep_for(std::chrono::seconds(10));
    }

    operation->Close();
    return 0;
```

```
}
```

發布/訂閱MQTT 訊 AWS IoT Core 息

AWS IoT Core MQTT 訊息 IPC 服務可讓您傳送和接收 MQTT 訊息。 AWS IoT Core元件可以將訊息發佈至 AWS IoT Core 並訂閱主題，以處理來自其他來源的 MQTT 訊息。如需 MQTT AWS IoT Core 實作的詳細資訊，請參閱開發人員指南中的 MQTT。AWS IoT Core

> **Note**
>
> 此 MQTT 訊息 IPC 服務可讓您與之交換訊息。 AWS IoT Core如需如何在元件之間交換郵件的相關資訊，請參閱發佈/訂閱本地訊息。

主題

最低 SDK 版本

下表列出您必須用來發行和訂閱 MQTT 訊息的最低版本。 AWS IoT Device SDK AWS IoT Core

SDK	最低版本	
AWS IoT Device SDK 對於 Java V2	v1.2.10	
AWS IoT Device SDK 對於 Python V2	V1.5.3	
AWS IoT Device SDK 對於 C++ 第 2	v1.17.0	

SDK	最低版本	
AWS IoT Device SDK 對於 JavaScript V2	v1.12.0	

授權

若要在自訂元件中使用 AWS IoT Core MQTT 訊息，您必須定義授權原則，以允許元件傳送和接收有關主題的訊息。如需定義授權原則的資訊，請參閱授權元件執行 IPC 作業。

AWS IoT Core MQTT 訊息的授權原則具有下列屬性。

IPC 服務識別碼：`aws.greengrass.ipc.mqttproxy`

作業	描述	資源
`aws.greengrass#PublishToIoTCore`	允許元件將訊息發佈到 AWS IoT Core 您指定的 MQTT 主題上。	主題字串，例如`test/topic`，或`*`允許存取所有主題。您可以使用 MQTT 主題萬用字元 (#和+) 來比對多個資源。
`aws.greengrass#SubscribeToIoTCore`	允許元件訂閱您指 AWS IoT Core 定主題的訊息。	主題字串，例如`test/topic`，或`*`允許存取所有主題。您可以使用 MQTT 主題萬用字元 (#和+) 來比對多個資源。
*	允許元件針對您指定的主題發佈和訂閱 AWS IoT Core MQTT 訊息。	主題字串，例如`test/topic`，或`*`允許存取所有主題。您可以使用 MQTT 主題萬用字元 (#和+) 來比對多個資源。

MQTT 授權政策中 AWS IoT Core 的 MQTT 萬用字元

您可以在 MQTT IPC 授權原則中使用 AWS IoT Core MQTT 萬用字元。元件可以發佈和訂閱符合授權原則中允許的主題篩選器的主題。例如，如果元件的授權原則授與存取權`test/topic/#`，則該元件可以訂閱`test/topic/#`，而且可以發佈和訂閱`test/topic/filter`。

AWS IoT Core MQTT 授權政策中的配方變數

如果您使用 v2.6.0 或更新版本的 Greengrass 核心，您可以在授權政策中使用 `{iot:thingName}` recipe 變數。此功能可讓您為一組核心裝置設定單一授權原則，其中每個核心裝置只能存取包含自己名稱的主題。例如，您可以允許元件存取下列主題資源。

```
devices/{iot:thingName}/messages
```

如需詳細資訊，請參閱 配方變數 及 在合併更新中使用配方變數。

授權政策範例

您可以參考下列授權原則範例，協助您設定元件的授權原則。

Example 無限制存取權限的授權原則範例

下列範例授權原則可讓元件發佈和訂閱所有主題。

JSON

```
{
  "accessControl": {
    "aws.greengrass.ipc.mqttproxy": {
      "com.example.MyIoTCorePubSubComponent:mqttproxy:1": {
        "policyDescription": "Allows access to publish/subscribe to all topics.",
        "operations": [
          "aws.greengrass#PublishToIoTCore",
          "aws.greengrass#SubscribeToIoTCore"
        ],
        "resources": [
          "*"
        ]
      }
    }
  }
}
```

YAML

```
---
accessControl:
  aws.greengrass.ipc.mqttproxy:
    com.example.MyIoTCorePubSubComponent:mqttproxy:1:
```

```
      policyDescription: Allows access to publish/subscribe to all topics.
      operations:
        - aws.greengrass#PublishToIoTCore
        - aws.greengrass#SubscribeToIoTCore
      resources:
        - "*"
```

Example 限制存取權限的授權原則範例

下列範例授權原則可讓元件發行和訂閱兩個名為factory/1/events和的主題factory/1/actions。

JSON

```
{
  "accessControl": {
    "aws.greengrass.ipc.mqttproxy": {
      "com.example.MyIoTCorePubSubComponent:mqttproxy:1": {
        "policyDescription": "Allows access to publish/subscribe to factory 1
 topics.",
        "operations": [
          "aws.greengrass#PublishToIoTCore",
          "aws.greengrass#SubscribeToIoTCore"
        ],
        "resources": [
          "factory/1/actions",
          "factory/1/events"
        ]
      }
    }
  }
}
```

YAML

```
---
accessControl:
  aws.greengrass.ipc.mqttproxy:
    "com.example.MyIoTCorePubSubComponent:mqttproxy:1":
      policyDescription: Allows access to publish/subscribe to factory 1 topics.
      operations:
        - aws.greengrass#PublishToIoTCore
```

```
        - aws.greengrass#SubscribeToIoTCore
      resources:
        - factory/1/actions
        - factory/1/events
```

Example 一組核心裝置的授權原則範例

> **Important**
>
> 此範例使用的功能可用於 v2.6.0 及更新版本的 Greeng rass 核元件。Greengrass 核 v2.6.0 增加了對大多數配方變量的支持，例如，在組件配置。{iot:thingName}

下列範例授權原則可讓元件發佈並訂閱包含執行元件之核心裝置名稱的主題。

JSON

```
{
  "accessControl": {
    "aws.greengrass.ipc.mqttproxy": {
      "com.example.MyIoTCorePubSubComponent:mqttproxy:1": {
        "policyDescription": "Allows access to publish/subscribe to all topics.",
        "operations": [
          "aws.greengrass#PublishToIoTCore",
          "aws.greengrass#SubscribeToIoTCore"
        ],
        "resources": [
          "factory/1/devices/{iot:thingName}/controls"
        ]
      }
    }
  }
}
```

YAML

```
---
accessControl:
  aws.greengrass.ipc.mqttproxy:
    "com.example.MyIoTCorePubSubComponent:mqttproxy:1":
      policyDescription: Allows access to publish/subscribe to all topics.
```

```
    operations:
      - aws.greengrass#PublishToIoTCore
      - aws.greengrass#SubscribeToIoTCore
    resources:
      - factory/1/devices/{iot:thingName}/controls
```

PublishToIoTCore

針 AWS IoT Core 對主題將 MQTT 訊息發佈至。

當您將 MQTT 訊息發佈到時 AWS IoT Core，每秒的交易配額為 100 個。如果您超出此配額，訊息就會排入佇列，以便在 Greengrass 裝置上進行處理。此外，每秒還有 512 Kb 的資料配額，以及每秒發佈 20,000 個帳戶的配額 (部分 AWS 區域為 2,000 個)。如需中 MQTT 訊息代理程式限制的相關資訊 AWS IoT Core，請參閱訊AWS IoT Core 息代理程式和通訊協定限制與配額。

如果您超過這些配額，Greengrass 裝置會限制將訊息發佈到。 AWS IoT Core消息存儲在內存中的後台處理程序中。根據預設，分配給多工緩衝處理程式的記憶體為 2.5 Mb。如果多工緩衝處理程式填滿，則會拒絕新郵件。您可以增加多工緩衝處理程式的大小。如需詳細資訊，請參閱 Greengrass 核文件中的 組態。為了避免填充多工緩衝處理程序並需要增加分配的內存，請將發布請求限制為每秒不超過 100 個請求。

當您的應用程式需要以較高的速率或較大的訊息傳送訊息時，請考慮使用將訊息傳送串流管理員至 Kinesis Data Streams。串流管理員元件旨在將大容量資料傳輸到 AWS 雲端. 如需詳細資訊，請參閱管理核心裝 Greengrass 資料串流。

請求

此操作的請求具有以下參數：

`topicName`(Python:`topic_name`)

要發佈郵件的目標主題。

`qos`

要使用的 MQoS TT 品質指標。此枚舉具有以下值：`QOS`

- `AT_MOST_ONCE`— QoS 0. MQTT 訊息最多只會傳送一次。
- `AT_LEAST_ONCE`— QoS 1. MQTT 訊息至少會傳送一次。

`payload`

(選擇性) 訊息承載為 blob。

使用 MQTT 5 時，v2.10.0 及更新版本可以Greengrass 核使用下列功能。當您使用 MQTT 3.1.1 時，會忽略這些功能。下表列出存取這些功能所必須使用的 AWS IoT 裝置 SDK 最低版本。

SDK	最低版本
適用於 Python 的 AWS IoT Device SDK v2	v1.15.0
適用於 JAVA 的 AWS IoT Device SDK v2	v1.13.0
適用於 C++ 的 AWS IoT Device SDK v2	V1.24.0
適用於 JavaScript 的 AWS IoT Device SDK v2	v1.13.0

`payloadFormat`

(選擇性) 訊息承載的格式。如果您未設定`payloadFormat`，則會假設類型為BYTES。枚舉具有以下值：

- BYTES— 有效負載的內容是二進位 blob。
- UTF8— 有效負載的內容是 UTF8 字元字串。

`retain`

(選擇性) 指出發佈時是否將 MQTT 保留選項設`true`定為。

`userProperties`

(選擇性) 要傳送的應用程式特定`UserProperty`物件清單。物`UserProperty`件定義如下：

```
UserProperty:
  key: string
  value: string
```

`messageExpiryIntervalSeconds`

(選擇性) 郵件到期前的秒數，並由伺服器刪除。如果未設定此值，則訊息不會過期。

`correlationData`

(選擇性) 新增至要求的資訊，可用來建立要求與回應的關聯。

`responseTopic`

(選擇性) 應該用於回應訊息的主題。

`contentType`

(選擇性) 訊息內容類型的應用程式特定識別碼。

回應

此操作在其響應中不提供任何信息。

範例

下列範例示範如何在自訂元件程式碼中呼叫此作業。

Java (IPC client V2)

Example 範例：發佈訊息

```
package com.aws.greengrass.docs.samples.ipc;

import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClientV2;
import software.amazon.awssdk.aws.greengrass.model.PublishToIoTCoreRequest;
import software.amazon.awssdk.aws.greengrass.model.QOS;
import java.nio.charset.StandardCharsets;

public class PublishToIoTCore {

    public static void main(String[] args) {
        String topic = args[0];
        String message = args[1];
        QOS qos = QOS.get(args[2]);

        try (GreengrassCoreIPCClientV2 ipcClientV2 =
 GreengrassCoreIPCClientV2.builder().build()) {
            ipcClientV2.publishToIoTCore(new PublishToIoTCoreRequest()
                    .withTopicName(topic)
                    .withPayload(message.getBytes(StandardCharsets.UTF_8))
                    .withQos(qos));
            System.out.println("Successfully published to topic: " + topic);
        } catch (Exception e) {
            System.err.println("Exception occurred.");
            e.printStackTrace();
            System.exit(1);
        }
    }
```

```
}
```

Python (IPC client V2)

Example 範例：發佈訊息

> **Note**
>
> 這個範例假設您使用的是適用 AWS IoT Device SDK 於 Python v2 的 1.5.4 版或更新版本。

```
import awsiot.greengrasscoreipc.clientv2 as clientV2

topic = 'my/topic'
qos = '1'
payload = 'Hello, World'

ipc_client = clientV2.GreengrassCoreIPCClientV2()
resp = ipc_client.publish_to_iot_core(topic_name=topic, qos=qos, payload=payload)
ipc_client.close()
```

Java (IPC client V1)

Example 範例：發佈訊息

> **Note**
>
> 此範例會使用`IPCUtils`類別來建立與 AWS IoT Greengrass Core IPC 服務的連線。如需詳細資訊，請參閱 Connect 到AWS IoT Greengrass酷睿 IPC 服務。

```
package com.aws.greengrass.docs.samples.ipc;

import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.PublishToIoTCoreResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.PublishToIoTCoreRequest;
import software.amazon.awssdk.aws.greengrass.model.PublishToIoTCoreResponse;
import software.amazon.awssdk.aws.greengrass.model.QOS;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;
```

```
import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class PublishToIoTCore {

    public static final int TIMEOUT_SECONDS = 10;

    public static void main(String[] args) {
        String topic = args[0];
        String message = args[1];
        QOS qos = QOS.get(args[2]);
        try (EventStreamRPCConnection eventStreamRPCConnection =
                     IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient =
                    new GreengrassCoreIPCClient(eventStreamRPCConnection);
            PublishToIoTCoreResponseHandler responseHandler =
                    PublishToIoTCore.publishBinaryMessageToTopic(ipcClient, topic,
 message, qos);
            CompletableFuture<PublishToIoTCoreResponse> futureResponse =
                    responseHandler.getResponse();
            try {
                futureResponse.get(TIMEOUT_SECONDS, TimeUnit.SECONDS);
                System.out.println("Successfully published to topic: " + topic);
            } catch (TimeoutException e) {
                System.err.println("Timeout occurred while publishing to topic: " +
 topic);
            } catch (ExecutionException e) {
                if (e.getCause() instanceof UnauthorizedError) {
                    System.err.println("Unauthorized error while publishing to
 topic: " + topic);
                } else {
                    throw e;
                }
            }
        } catch (InterruptedException e) {
            System.out.println("IPC interrupted.");
        } catch (ExecutionException e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
```

```
        }
    }

    public static PublishToIoTCoreResponseHandler
 publishBinaryMessageToTopic(GreengrassCoreIPCClient greengrassCoreIPCClient, String
 topic, String message, QOS qos) {
        PublishToIoTCoreRequest publishToIoTCoreRequest = new
 PublishToIoTCoreRequest();
        publishToIoTCoreRequest.setTopicName(topic);

 publishToIoTCoreRequest.setPayload(message.getBytes(StandardCharsets.UTF_8));
        publishToIoTCoreRequest.setQos(qos);
        return greengrassCoreIPCClient.publishToIoTCore(publishToIoTCoreRequest,
 Optional.empty());
    }
}
```

Python (IPC client V1)

Example 範例：發佈訊息

> **Note**
>
> 這個範例假設您使用的是適用 AWS IoT Device SDK 於 Python v2 的 1.5.4 版或更新版本。

```
import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import (
    QOS,
    PublishToIoTCoreRequest
)

TIMEOUT = 10

ipc_client = awsiot.greengrasscoreipc.connect()

topic = "my/topic"
message = "Hello, World"
qos = QOS.AT_LEAST_ONCE

request = PublishToIoTCoreRequest()
```

```
request.topic_name = topic
request.payload = bytes(message, "utf-8")
request.qos = qos
operation = ipc_client.new_publish_to_iot_core()
operation.activate(request)
future_response = operation.get_response()
future_response.result(TIMEOUT)
```

C++

Example 範例：發佈訊息

```
#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        // Handle connection to IPC service.
    }

    void OnDisconnectCallback(RpcError error) override {
        // Handle disconnection from IPC service.
    }

    bool OnErrorCallback(RpcError error) override {
        // Handle IPC service connection error.
        return true;
    }
};

int main() {
    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
```

```
        std::cerr << "Failed to establish IPC connection: " <<
 connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    String message("Hello, World!");
    String topic("my/topic");
    QOS qos = QOS_AT_MOST_ONCE;
    int timeout = 10;

    PublishToIoTCoreRequest request;
    Vector<uint8_t> messageData({message.begin(), message.end()});
    request.SetTopicName(topic);
    request.SetPayload(messageData);
    request.SetQos(qos);

    auto operation = ipcClient.NewPublishToIoTCore();
    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
 std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from Greengrass
 Core." << std::endl;
        exit(-1);
    }

    auto response = responseFuture.get();
    if (!response) {
        // Handle error.
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            (void)error;
            // Handle operation error.
        } else {
            // Handle RPC error.
        }
    }

    return 0;
}
```

JavaScript

Example 範例：發佈訊息

```
import * as greengrasscoreipc from "aws-iot-device-sdk-v2/dist/greengrasscoreipc";
import {QOS, PublishToIoTCoreRequest} from "aws-iot-device-sdk-v2/dist/
greengrasscoreipc/model";

class PublishToIoTCore {
    private ipcClient: greengrasscoreipc.Client
    private readonly topic: string;

    constructor() {
        // define your own constructor, e.g.
        this.topic = "<define_your_topic>";
        this.publishToIoTCore().then(r => console.log("Started workflow"));
    }

    private async publishToIoTCore() {
        try {
            const request: PublishToIoTCoreRequest = {
                topicName: this.topic,
                qos: QOS.AT_LEAST_ONCE, // you can change this depending on your use
 case
            }

            this.ipcClient = await getIpcClient();

            await this.ipcClient.publishToIoTCore(request);
        } catch (e) {
            // parse the error depending on your use cases
            throw e
        }
    }
}


export async function getIpcClient(){
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
```

```
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

// starting point
const publishToIoTCore = new PublishToIoTCore();
```

SubscribeToIoTCore

訂閱來自主題或主題篩選器 AWS IoT Core 的 MQTT 訊息。 AWS IoT Greengrass 核心軟體會在元件到達其生命週期結束時移除訂閱。

此作業是訂閱作業，您可以在其中訂閱事件訊息串流。若要使用此作業，請定義具有處理事件訊息、錯誤和資料流結束之函數的串流回應處理常式。如需詳細資訊，請參閱 訂閱 IPC 事件串流。

事件訊息類型：`IoTCoreMessage`

請求

此操作的請求具有以下參數：

`topicName`(Python:`topic_name`)

要訂閱的主題。您可以使用 MQTT 主題萬用字元 (#和+) 來訂閱多個主題。

`qos`

要使用的 MQoS TT 品質指標。此枚舉具有以下值：`QOS`

- `AT_MOST_ONCE`— QoS 0. MQTT 訊息最多只會傳送一次。
- `AT_LEAST_ONCE`— QoS 1. MQTT 訊息至少會傳送一次。

回應

此作業的回應包含下列資訊：

`messages`

MQTT 訊息的串流。此物件`IoTCoreMessage`包含下列資訊：

`message`

MQTT 訊息。此物件`MQTTMessage`包含下列資訊：

`topicName`(Python:`topic_name`)

郵件發行目標的主題。

`payload`

(選擇性) 訊息承載為 blob。

使用 MQTT 5 時，v2.10.0 及更新版本可以Greengrass 核使用下列功能。當您使用 MQTT 3.1.1 時，會忽略這些功能。下表列出存取這些功能所必須使用的 AWS IoT 裝置 SDK 最低版本。

SDK	最低版本
適用於 Python 的 AWS IoT Device SDK v2	v1.15.0
適用於 JAVA 的 AWS IoT Device SDK v2	v1.13.0
適用於 C++ 的 AWS IoT Device SDK v2	V1.24.0
適用於 JavaScript 的 AWS IoT Device SDK v2	v1.13.0

`payloadFormat`

(選擇性) 訊息承載的格式。如果您未設定`payloadFormat`，則會假設類型為`BYTES`。枚舉具有以下值：

- BYTES— 有效負載的內容是二進位 blob。
- UTF8— 有效負載的內容是 UTF8 字元字串。

`retain`

(選擇性) 指出發佈時是否將 MQTT 保留選項設`true`定為。

`userProperties`

(選擇性) 要傳送的應用程式特定`UserProperty`物件清單。物`UserProperty`件定義如下：

```
UserProperty:
  key: string
  value: string
```

`messageExpiryIntervalSeconds`

(選擇性) 郵件到期前的秒數，並由伺服器刪除。如果未設定此值，則訊息不會過期。

`correlationData`

(選擇性) 新增至要求的資訊，可用來建立要求與回應的關聯。

`responseTopic`

(選擇性) 應該用於回應訊息的主題。

`contentType`

(選擇性) 訊息內容類型的應用程式特定識別碼。

範例

下列範例示範如何在自訂元件程式碼中呼叫此作業。

Java (IPC client V2)

Example 範例：訂閱訊息

```
package com.aws.greengrass.docs.samples.ipc;

import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClientV2;
import software.amazon.awssdk.aws.greengrass.SubscribeToIoTCoreResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.QOS;
import software.amazon.awssdk.aws.greengrass.model.IoTCoreMessage;
import software.amazon.awssdk.aws.greengrass.model.SubscribeToIoTCoreRequest;
import software.amazon.awssdk.aws.greengrass.model.SubscribeToIoTCoreResponse;

import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.function.Consumer;
import java.util.function.Function;


public class SubscribeToIoTCore {
```

```
    public static void main(String[] args) {
        String topic = args[0];
        QOS qos = QOS.get(args[1]);

        Consumer<IoTCoreMessage> onStreamEvent = ioTCoreMessage ->
                System.out.printf("Received new message on topic %s: %s%n",
                        ioTCoreMessage.getMessage().getTopicName(),
                        new String(ioTCoreMessage.getMessage().getPayload(),
StandardCharsets.UTF_8));

        Optional<Function<Throwable, Boolean>> onStreamError =
                Optional.of(e -> {
                    System.err.println("Received a stream error.");
                    e.printStackTrace();
                    return false;
                });

        Optional<Runnable> onStreamClosed = Optional.of(() ->
                System.out.println("Subscribe to IoT Core stream closed."));

        try (GreengrassCoreIPCClientV2 ipcClientV2 =
GreengrassCoreIPCClientV2.builder().build()) {
            SubscribeToIoTCoreRequest request = new SubscribeToIoTCoreRequest()
                    .withTopicName(topic)
                    .withQos(qos);

            GreengrassCoreIPCClientV2.StreamingResponse<SubscribeToIoTCoreResponse,
SubscribeToIoTCoreResponseHandler>
                    streamingResponse = ipcClientV2.subscribeToIoTCore(request,
onStreamEvent, onStreamError, onStreamClosed);

            streamingResponse.getResponse();
            System.out.println("Successfully subscribed to topic: " + topic);

            // Keep the main thread alive, or the process will exit.
            while (true) {
                Thread.sleep(10000);
            }

            // To stop subscribing, close the stream.
            streamingResponse.getHandler().closeStream();
        } catch (InterruptedException e) {
            System.out.println("Subscribe interrupted.");
        } catch (Exception e) {
```

```
            System.err.println("Exception occurred.");
            e.printStackTrace();
            System.exit(1);
        }
    }
}
```

Python (IPC client V2)

Example 範例：訂閱訊息

> **Note**
>
> 這個範例假設您使用的是適用 AWS IoT Device SDK 於 Python v2 的 1.5.4 版或更新版本。

```
import threading
import traceback

import awsiot.greengrasscoreipc.clientv2 as clientV2

topic = 'my/topic'
qos = '1'

def on_stream_event(event):
    try:
        topic_name = event.message.topic_name
        message = str(event.message.payload, 'utf-8')
        print(f'Received new message on topic {topic_name}:  {message}')
    except:
        traceback.print_exc()

def on_stream_error(error):
    # Return True to close stream, False to keep stream open.
    return True

def on_stream_closed():
    pass

ipc_client = clientV2.GreengrassCoreIPCClientV2()
resp, operation = ipc_client.subscribe_to_iot_core(
    topic_name=topic,
```

```
    qos=qos,
    on_stream_event=on_stream_event,
    on_stream_error=on_stream_error,
    on_stream_closed=on_stream_closed
)

# Keep the main thread alive, or the process will exit.
event = threading.Event()
event.wait()

# To stop subscribing, close the operation stream.
operation.close()
ipc_client.close()
```

Java (IPC client V1)

Example 範例：訂閱訊息

> **Note**
>
> 此範例會使用`IPCUtils`類別來建立與 AWS IoT Greengrass Core IPC 服務的連線。如需詳細資訊，請參閱 Connect 到AWS IoT Greengrass酷睿 IPC 服務。

```
package com.aws.greengrass.docs.samples.ipc;

import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.SubscribeToIoTCoreResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.*;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;
import software.amazon.awssdk.eventstreamrpc.StreamResponseHandler;

import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class SubscribeToIoTCore {

    public static final int TIMEOUT_SECONDS = 10;
```

```
    public static void main(String[] args) {
        String topic = args[0];
        QOS qos = QOS.get(args[1]);
        try (EventStreamRPCConnection eventStreamRPCConnection =
                     IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient =
                    new GreengrassCoreIPCClient(eventStreamRPCConnection);
            StreamResponseHandler<IoTCoreMessage> streamResponseHandler =
                    new SubscriptionResponseHandler();
            SubscribeToIoTCoreResponseHandler responseHandler =
                    SubscribeToIoTCore.subscribeToIoTCore(ipcClient, topic, qos,
                            streamResponseHandler);
            CompletableFuture<SubscribeToIoTCoreResponse> futureResponse =
                    responseHandler.getResponse();
            try {
                futureResponse.get(TIMEOUT_SECONDS, TimeUnit.SECONDS);
                System.out.println("Successfully subscribed to topic: " + topic);
            } catch (TimeoutException e) {
                System.err.println("Timeout occurred while subscribing to topic: " +
topic);
            } catch (ExecutionException e) {
                if (e.getCause() instanceof UnauthorizedError) {
                    System.err.println("Unauthorized error while subscribing to
topic: " + topic);
                } else {
                    throw e;
                }
            }

            // Keep the main thread alive, or the process will exit.
            try {
                while (true) {
                    Thread.sleep(10000);
                }
            } catch (InterruptedException e) {
                System.out.println("Subscribe interrupted.");
            }

            // To stop subscribing, close the stream.
            responseHandler.closeStream();
        } catch (InterruptedException e) {
            System.out.println("IPC interrupted.");
        } catch (ExecutionException e) {
```

```
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static SubscribeToIoTCoreResponseHandler
subscribeToIoTCore(GreengrassCoreIPCClient greengrassCoreIPCClient, String topic,
QOS qos, StreamResponseHandler<IoTCoreMessage> streamResponseHandler) {
        SubscribeToIoTCoreRequest subscribeToIoTCoreRequest = new
SubscribeToIoTCoreRequest();
        subscribeToIoTCoreRequest.setTopicName(topic);
        subscribeToIoTCoreRequest.setQos(qos);
        return
greengrassCoreIPCClient.subscribeToIoTCore(subscribeToIoTCoreRequest,
                Optional.of(streamResponseHandler));
    }

    public static class SubscriptionResponseHandler implements
StreamResponseHandler<IoTCoreMessage> {

        @Override
        public void onStreamEvent(IoTCoreMessage ioTCoreMessage) {
            try {
                String topic = ioTCoreMessage.getMessage().getTopicName();
                String message = new
String(ioTCoreMessage.getMessage().getPayload(),
                        StandardCharsets.UTF_8);
                System.out.printf("Received new message on topic %s: %s%n", topic,
message);
            } catch (Exception e) {
                System.err.println("Exception occurred while processing subscription
response " +
                        "message.");
                e.printStackTrace();
            }
        }

        @Override
        public boolean onStreamError(Throwable error) {
            System.err.println("Received a stream error.");
            error.printStackTrace();
            return false;
        }
```

```
        @Override
        public void onStreamClosed() {
            System.out.println("Subscribe to IoT Core stream closed.");
        }
    }
}
```

Python (IPC client V1)

Example 範例：訂閱訊息

> **Note**
>
> 這個範例假設您使用的是適用 AWS IoT Device SDK 於 Python v2 的 1.5.4 版或更新版本。

```
import time
import traceback

import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import (
    IoTCoreMessage,
    QOS,
    SubscribeToIoTCoreRequest
)

TIMEOUT = 10

ipc_client = awsiot.greengrasscoreipc.connect()

class StreamHandler(client.SubscribeToIoTCoreStreamHandler):
    def __init__(self):
        super().__init__()

    def on_stream_event(self, event: IoTCoreMessage) -> None:
        try:
            message = str(event.message.payload, "utf-8")
            topic_name = event.message.topic_name
            # Handle message.
        except:
            traceback.print_exc()
```

```
    def on_stream_error(self, error: Exception) -> bool:
        # Handle error.
        return True  # Return True to close stream, False to keep stream open.

    def on_stream_closed(self) -> None:
        # Handle close.
        pass


topic = "my/topic"
qos = QOS.AT_MOST_ONCE

request = SubscribeToIoTCoreRequest()
request.topic_name = topic
request.qos = qos
handler = StreamHandler()
operation = ipc_client.new_subscribe_to_iot_core(handler)
operation.activate(request)
future_response = operation.get_response()
future_response.result(TIMEOUT)

# Keep the main thread alive, or the process will exit.
while True:
    time.sleep(10)

# To stop subscribing, close the operation stream.
operation.close()
```

C++

Example 範例：訂閱訊息

```
#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class IoTCoreResponseHandler : public SubscribeToIoTCoreStreamHandler {

    public:
```

```
        virtual ~IoTCoreResponseHandler() {}

    private:
        void OnStreamEvent(IoTCoreMessage *response) override {
            auto message = response->GetMessage();
            if (message.has_value() && message.value().GetPayload().has_value()) {
                auto messageBytes = message.value().GetPayload().value();
                std::string messageString(messageBytes.begin(), messageBytes.end());
                std::string topicName =
 message.value().GetTopicName().value().c_str();
                // Handle message.
            }
        }

        bool OnStreamError(OperationError *error) override {
            // Handle error.
            return false; // Return true to close stream, false to keep stream open.
        }

        void OnStreamClosed() override {
            // Handle close.
        }
};

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        // Handle connection to IPC service.
    }

    void OnDisconnectCallback(RpcError error) override {
        // Handle disconnection from IPC service.
    }

    bool OnErrorCallback(RpcError error) override {
        // Handle IPC service connection error.
        return true;
    }
};

int main() {
    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
```

```
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    String topic("my/topic");
    QOS qos = QOS_AT_MOST_ONCE;
    int timeout = 10;

    SubscribeToIoTCoreRequest request;
    request.SetTopicName(topic);
    request.SetQos(qos);
    auto streamHandler = MakeShared<IoTCoreResponseHandler>(DefaultAllocator());
    auto operation = ipcClient.NewSubscribeToIoTCore(streamHandler);
    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from Greengrass
Core." << std::endl;
        exit(-1);
    }

    auto response = responseFuture.get();
    if (!response) {
        // Handle error.
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            (void)error;
            // Handle operation error.
        } else {
            // Handle RPC error.
        }
        exit(-1);
    }

    // Keep the main thread alive, or the process will exit.
```

```
    while (true) {
        std::this_thread::sleep_for(std::chrono::seconds(10));
    }

    operation->Close();
    return 0;
}
```

JavaScript

Example 範例：訂閱訊息

```
import * as greengrasscoreipc from "aws-iot-device-sdk-v2/dist/greengrasscoreipc";
import {IoTCoreMessage, QOS, SubscribeToIoTCoreRequest} from "aws-iot-device-sdk-v2/
dist/greengrasscoreipc/model";
import {RpcError} from "aws-iot-device-sdk-v2/dist/eventstream_rpc";

class SubscribeToIoTCore {
    private ipcClient: greengrasscoreipc.Client
    private readonly topic: string;

    constructor() {
        // define your own constructor, e.g.
        this.topic = "<define_your_topic>";
        this.subscribeToIoTCore().then(r => console.log("Started workflow"));
    }

    private async subscribeToIoTCore() {
        try {
            const request: SubscribeToIoTCoreRequest = {
                topicName: this.topic,
                qos: QOS.AT_LEAST_ONCE, // you can change this depending on your use
 case
            }

            this.ipcClient = await getIpcClient();

            const streamingOperation = this.ipcClient.subscribeToIoTCore(request);

            streamingOperation.on('message', (message: IoTCoreMessage) => {
                // parse the message depending on your use cases, e.g.
                if (message.message && message.message.payload) {
                    const receivedMessage = message.message.payload.toString();
```

```
                }
            });

            streamingOperation.on('streamError', (error : RpcError) => {
                // define your own error handling logic
            });

            streamingOperation.on('ended', () => {
                // define your own logic
            });

            await streamingOperation.activate();

            // Keep the main thread alive, or the process will exit.
            await new Promise((resolve) => setTimeout(resolve, 10000))
        } catch (e) {
            // parse the error depending on your use cases
            throw e
        }
    }
}

export async function getIpcClient(){
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

// starting point
const subscribeToIoTCore = new SubscribeToIoTCore();
```

範例

請參閱下列範例，瞭解如何在元件中使用 AWS IoT Core MQTT IPC 服務。

AWS IoT Core MQTT 發行者範例 (C++)

下列範例方案可讓元件發佈至所有主題。

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.IoTCorePublisherCpp",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that publishes MQTT messages to IoT Core.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.mqttproxy": {
          "com.example.IoTCorePublisherCpp:mqttproxy:1": {
            "policyDescription": "Allows access to publish to all topics.",
            "operations": [
              "aws.greengrass#PublishToIoTCore"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Lifecycle": {
        "run": "{artifacts:path}/greengrassv2_iotcore_publisher"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.IoTCorePublisherCpp/1.0.0/greengrassv2_iotcore_publisher",
          "Permission": {
            "Execute": "OWNER"
```

```
          }
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.IoTCorePublisherCpp
ComponentVersion: 1.0.0
ComponentDescription: A component that publishes MQTT messages to IoT Core.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.mqttproxy:
        com.example.IoTCorePublisherCpp:mqttproxy:1:
          policyDescription: Allows access to publish to all topics.
          operations:
            - aws.greengrass#PublishToIoTCore
          resources:
            - "*"
Manifests:
  - Lifecycle:
      run: "{artifacts:path}/greengrassv2_iotcore_publisher"
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.IoTCorePublisherCpp/1.0.0/greengrassv2_iotcore_publisher
        Permission:
          Execute: OWNER
```

下列範例 C++ 應用程式示範如何使用 AWS IoT Core MQTT IPC 服務將訊息發佈至。 AWS IoT Core

```
#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
```

```
using namespace Aws::Greengrass;

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        std::cout << "OnConnectCallback" << std::endl;
    }

    void OnDisconnectCallback(RpcError error) override {
        std::cout << "OnDisconnectCallback: " << error.StatusToString() << std::endl;
        exit(-1);
    }

    bool OnErrorCallback(RpcError error) override {
        std::cout << "OnErrorCallback: " << error.StatusToString() << std::endl;
        return true;
    }
};

int main() {
    String message("Hello from the Greengrass IPC MQTT publisher (C++).");
    String topic("test/topic/cpp");
    QOS qos = QOS_AT_LEAST_ONCE;
    int timeout = 10;

    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
 connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    while (true) {
        PublishToIoTCoreRequest request;
        Vector<uint8_t> messageData({message.begin(), message.end()});
        request.SetTopicName(topic);
        request.SetPayload(messageData);
        request.SetQos(qos);
```

```
        auto operation = ipcClient.NewPublishToIoTCore();
        auto activate = operation->Activate(request, nullptr);
        activate.wait();

        auto responseFuture = operation->GetResult();
        if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
 std::future_status::timeout) {
            std::cerr << "Operation timed out while waiting for response from
 Greengrass Core." << std::endl;
            exit(-1);
        }

        auto response = responseFuture.get();
        if (response) {
            std::cout << "Successfully published to topic: " << topic << std::endl;
        } else {
            // An error occurred.
            std::cout << "Failed to publish to topic: " << topic << std::endl;
            auto errorType = response.GetResultType();
            if (errorType == OPERATION_ERROR) {
                auto *error = response.GetOperationError();
                std::cout << "Operation error: " << error->GetMessage().value() <<
 std::endl;
            } else {
                std::cout << "RPC error: " << response.GetRpcError() << std::endl;
            }
            exit(-1);
        }

        std::this_thread::sleep_for(std::chrono::seconds(5));
    }

    return 0;
}
```

示例 AWS IoT Core MQTT 用戶（C ++）

下列範例方案可讓元件訂閱所有主題。

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.IoTCoreSubscriberCpp",
```

```
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that subscribes to MQTT messages from IoT
 Core.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.mqttproxy": {
          "com.example.IoTCoreSubscriberCpp:mqttproxy:1": {
            "policyDescription": "Allows access to subscribe to all topics.",
            "operations": [
              "aws.greengrass#SubscribeToIoTCore"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Lifecycle": {
        "run": "{artifacts:path}/greengrassv2_iotcore_subscriber"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.IoTCoreSubscriberCpp/1.0.0/greengrassv2_iotcore_subscriber",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
```

```
ComponentName: com.example.IoTCoreSubscriberCpp
ComponentVersion: 1.0.0
ComponentDescription: A component that subscribes to MQTT messages from IoT Core.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.mqttproxy:
        com.example.IoTCoreSubscriberCpp:mqttproxy:1:
          policyDescription: Allows access to subscribe to all topics.
          operations:
            - aws.greengrass#SubscribeToIoTCore
          resources:
            - "*"
Manifests:
  - Lifecycle:
      run: "{artifacts:path}/greengrassv2_iotcore_subscriber"
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.IoTCoreSubscriberCpp/1.0.0/greengrassv2_iotcore_subscriber
        Permission:
          Execute: OWNER
```

下列範例 C++ 應用程式示範如何使用 AWS IoT Core MQTT IPC 服務來訂閱訊息。 AWS IoT Core

```
#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class IoTCoreResponseHandler : public SubscribeToIoTCoreStreamHandler {

    public:
        virtual ~IoTCoreResponseHandler() {}

    private:

        void OnStreamEvent(IoTCoreMessage *response) override {
            auto message = response->GetMessage();
```

```
            if (message.has_value() && message.value().GetPayload().has_value()) {
                auto messageBytes = message.value().GetPayload().value();
                std::string messageString(messageBytes.begin(), messageBytes.end());
                std::string messageTopic =
 message.value().GetTopicName().value().c_str();
                std::cout << "Received new message on topic: " << messageTopic <<
 std::endl;
                std::cout << "Message: " << messageString << std::endl;
            }
        }

        bool OnStreamError(OperationError *error) override {
            std::cout << "Received an operation error: ";
            if (error->GetMessage().has_value()) {
                std::cout << error->GetMessage().value();
            }
            std::cout << std::endl;
            return false; // Return true to close stream, false to keep stream open.
        }

        void OnStreamClosed() override {
            std::cout << "Subscribe to IoT Core stream closed." << std::endl;
        }
};

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        std::cout << "OnConnectCallback" << std::endl;
    }

    void OnDisconnectCallback(RpcError error) override {
        std::cout << "OnDisconnectCallback: " << error.StatusToString() << std::endl;
        exit(-1);
    }

    bool OnErrorCallback(RpcError error) override {
        std::cout << "OnErrorCallback: " << error.StatusToString() << std::endl;
        return true;
    }
};

int main() {
    String topic("test/topic/cpp");
    QOS qos = QOS_AT_LEAST_ONCE;
```

```
    int timeout = 10;

    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    SubscribeToIoTCoreRequest request;
    request.SetTopicName(topic);
    request.SetQos(qos);
    auto streamHandler = MakeShared<IoTCoreResponseHandler>(DefaultAllocator());
    auto operation = ipcClient.NewSubscribeToIoTCore(streamHandler);
    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from Greengrass
Core." << std::endl;
        exit(-1);
    }

    auto response = responseFuture.get();
    if (response) {
        std::cout << "Successfully subscribed to topic: " << topic << std::endl;
    } else {
        // An error occurred.
        std::cout << "Failed to subscribe to topic: " << topic << std::endl;
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            std::cout << "Operation error: " << error->GetMessage().value() <<
std::endl;
        } else {
            std::cout << "RPC error: " << response.GetRpcError() << std::endl;
```

```
        }
        exit(-1);
    }

    // Keep the main thread alive, or the process will exit.
    while (true) {
        std::this_thread::sleep_for(std::chrono::seconds(10));
    }

    operation->Close();
    return 0;
}
```

與組件生命週期互動

使用元件生命週期 IPC 服務來：

- 更新核心裝置上的元件狀態。
- 訂閱元件狀態更新。
- 防止核心在部署期間停止元件以套用更新。
- 暫停並繼續元件程序。

主題

最低 SDK 版本

下表列出與元件生命週期互動AWS IoT Device SDK所必須使用的最低版本。

SDK	最低版本	
AWS IoT Device SDK對於爪哇 V2	v1.2.10	
AWS IoT Device SDK對於 Python V2	V1.5.3	
AWS IoT Device SDK對於 C ++ 第 2	v1.17.0	
AWS IoT Device SDK對於 JavaScript V2	v1.12.0	

授權

若要從自訂元件暫停或恢復其他元件，您必須定義授權原則，以允許您的元件管理其他元件。如需有關定義授權原則的資訊，請參閱授權元件執行 IPC 作業。

元件生命週期管理的授權原則具有下列屬性。

IPC 服務識別碼：`aws.greengrass.ipc.lifecycle`

操作	描述	資源
`aws.greengrass#PauseComponent`	允許元件暫停您指定的元件。	元件名稱，或*允許存取所有元件。
`aws.greengrass#ResumeComponent`	允許元件恢復您指定的元件。	元件名稱，或*允許存取所有元件。
*	允許元件暫停和恢復您指定的元件。	元件名稱，或*允許存取所有元件。

授權政策範例

您可以參考下列授權原則範例，協助您設定元件的授權原則。

Example 授權政策範例

下列範例授權原則可讓元件暫停和繼續所有元件。

```
{
  "accessControl": {
    "aws.greengrass.ipc.lifecycle": {
      "com.example.MyLocalLifecycleComponent:lifecycle:1": {
        "policyDescription": "Allows access to pause/resume all components.",
        "operations": [
          "aws.greengrass#PauseComponent",
          "aws.greengrass#ResumeComponent"
        ],
        "resources": [
          "*"
        ]
      }
    }
  }
}
```

UpdateState

更新核心裝置上元件的狀態。

請求

此操作的請求具有以下參數：

`state`

要設定的狀態。此枚舉具有以下值：`LifecycleState`

- `RUNNING`
- `ERRORED`

回應

此操作在其響應中不提供任何信息。

SubscribeToComponentUpdates

訂閱以在 AWS IoT Greengrass Core 軟體更新元件之前接收通知。通知會指定核心是否會在更新過程中重新啟動。

僅當部署的元件更新原則指定通知元件時，核心才會傳送更新通知。預設行為是通知元件。如需詳細資訊，請參閱建立部署和呼叫CreateDeployment作業時可提供的DeploymentComponentUpdatePolicy物件。

> **Important**
>
> 本機部署不會在更新之前通知元件。

此作業是訂閱作業，您可以在其中訂閱事件訊息串流。若要使用此作業，請定義具有處理事件訊息、錯誤和資料流結束之函數的串流回應處理常式。如需詳細資訊，請參閱 訂閱 IPC 事件串流。

事件訊息類型：`ComponentUpdatePolicyEvents`

> **Tip**
>
> 您可以按照自學課程學習如何開發有條件地延遲元件更新的元件。如需詳細資訊，請參閱 教學課程：開發延遲元件更新的 Greengrass 元件。

請求

此操作的請求沒有任何參數。

回應

此作業的回應包含下列資訊：

`messages`

通知訊息的資料流。此物件`ComponentUpdatePolicyEvents`包含下列資訊：

`preUpdateEvent`(Python:`pre_update_event`)

(選擇性) 指出核心想要更新元件的事件。您可以回應DeferComponentUpdate作業以確認或延遲更新，直到元件準備好重新啟動為止。此物件`PreComponentUpdateEvent`包含下列資訊：

`deploymentId`(Python:`deployment_id`)

更新元件的AWS IoT Greengrass部署識別碼。

`isGgcRestarting`(Python:`is_ggc_restarting`)

核心是否需要重新啟動才能套用更新。

`postUpdateEvent`(Python:`post_update_event`)

(選擇性) 表示原子核已更新元件的事件。此物件`PostComponentUpdateEvent`包含下列資訊：

`deploymentId`(Python:`deployment_id`)

更新元件的AWS IoT Greengrass部署識別碼。

> **Note**
>
> 此功能需要 v2.7.0 或更高版 Greengrass 核成分。

DeferComponentUpdate

確認或延遲您發現的SubscribeToComponentUpdates元件更新。您可以指定在核心再次檢查元件是否準備好讓元件進行更新之前等待的時間長度。您也可以使用此作業告知核心您的元件已準備好進行更新。

如果元件未回應元件更新通知，則核心會等待您在部署的元件更新原則中指定的時間量。在該逾時之後，核心會繼續進行部署。預設元件更新逾時為 60 秒。如需詳細資訊，請參閱建立部署和呼叫CreateDeployment作業時可提供的DeploymentComponentUpdatePolicy物件。

> **Tip**
>
> 您可以按照自學課程學習如何開發有條件地延遲元件更新的元件。如需詳細資訊，請參閱 教學課程：開發延遲元件更新的 Greengrass 元件。

請求

此操作的請求具有以下參數：

`deploymentId`(Python:`deployment_id`)

要延遲的部AWS IoT Greengrass署識別碼。

`message`

(選擇性) 要延期更新的元件名稱。

默認為發出請求的組件的名稱。

`recheckAfterMs`(Python:`recheck_after_ms`)

延遲更新的時間 (以毫秒為單位)。細胞核等待這段時間，然後發送另一個`PreComponentUpdateEvent`您可以發現的時間。SubscribeToComponentUpdates

指定`0`以確認更新。這會告訴核心您的元件已準備好進行更新。

默認為零毫秒，這意味著要確認更新。

回應

此操作在其響應中不提供任何信息。

PauseComponent

此功能適用於 v2.4.0 和更高版 Greeng rass 核組件。 AWS IoT Greengrass目前在 Windows 核心裝置上不支援此功能。

在核心裝置上暫停元件的處理序。若要恢復元件，請使用此ResumeComponent作業。

您只能暫停類屬元件。如果您嘗試暫停任何其他類型的元件，此作業會擲回`InvalidRequestError`.

> **Note**
>
> 此作業無法暫停容器化處理序，例如 Docker 容器。若要暫停和繼續 docker 容器，您可以使用 docker 暫停和泊塢視窗取消暫停命令。

此作業不會暫停依賴於您暫停之元件的元件相依性或元件。當您暫停屬於另一個元件相依性的元件時，請考慮這個行為，因為相依元件在暫停其相依性時可能會遇到問題。

當您重新啟動或關閉暫停的元件 (例如透過部署) 時，Greengrass 核心會恢復元件並執行其關閉生命週期。如需重新啟動元件的詳細資訊，請參閱RestartComponent。

> **Important**
>
> 若要使用此作業，您必須定義授與使用此作業之權限的授權原則。如需詳細資訊，請參閱 授權。

最低 SDK 版本

下表列出暫停和恢復元件所AWS IoT Device SDK必須使用的最低版本。

SDK	最低版本	
AWS IoT Device SDK對於爪哇 V2	v1.4.3	
AWS IoT Device SDK對於 Python V2	v1.6.2	
AWS IoT Device SDK對於 C ++ 第 2	V1.13.1	
AWS IoT Device SDK對於 JavaScript V2	v1.12.0	

請求

此操作的請求具有以下參數：

`componentName`(Python:`component_name`)

要暫停的元件名稱，必須是類屬元件。如需詳細資訊，請參閱 元件類型。

回應

此操作在其響應中不提供任何信息。

ResumeComponent

此功能適用於 v2.4.0 和更高版 Greeng rass 核組件。 AWS IoT Greengrass目前在 Windows 核心裝置上不支援此功能。

在核心裝置上繼續執行元件的程序。若要暫停元件，請使用此PauseComponent作業。

您只能繼續暫停的元件。如果您嘗試繼續未暫停的元件，此作業會擲`InvalidRequestError`回.

> **Important**
>
> 若要使用此作業，您必須定義授與權限的授權原則。如需詳細資訊，請參閱 授權。

最低 SDK 版本

下表列出暫停和恢復元件所AWS IoT Device SDK必須使用的最低版本。

SDK	最低版本	
AWS IoT Device SDK對於爪哇 V2	v1.4.3	
AWS IoT Device SDK對於 Python V2	v1.6.2	
AWS IoT Device SDK對於 C ++ 第 2	V1.13.1	
AWS IoT Device SDK對於 JavaScript V2	v1.12.0	

請求

此操作的請求具有以下參數：

`componentName`(Python:`component_name`)

要恢復的元件名稱。

回應

此操作在其響應中不提供任何信息。

與組件配置互動

元件組態 IPC 服務可讓您執行下列作業：

- 取得並設定元件組態參數。
- 訂閱元件組態更新。
- 在核心套用元件之前，先驗證零組件模型組態更新。

主題

最低 SDK 版本

下表列出與元件組態互動AWS IoT Device SDK所必須使用的最低版本。

SDK	最低版本	
AWS IoT Device SDK對於 Java V2	v1.2.10	
AWS IoT Device SDK對於 Python V2	V1.5.3	
AWS IoT Device SDK對於 C++ 2	v1.17.0	

SDK	最低版本	
AWS IoT Device SDK對於 JavaScript V2	v1.12.0	

GetConfiguration

取得核心裝置上元件的組態值。您可以指定要取得組態值的金鑰路徑。

請求

此操作的請求具有以下參數：

componentName(Python:component_name)

(選擇性) 元件的名稱。

默認為發出請求的組件的名稱。

keyPath(Python:key_path)

組態值的關鍵路徑。指定清單，其中每個項目都是組態物件中單一層級的索引鍵。例如，指定`["mqtt", "port"]`以取得下列組態`port`中的值。

```
{
  "mqtt": {
    "port": 443
  }
}
```

若要取得元件的完整組態，請指定空白清單。

回應

此作業的回應包含下列資訊：

componentName(Python:component_name)

元件的名稱。

`value`

請求的配置作為對象。

UpdateConfiguration

更新核心裝置上此元件的組態值。

請求

此操作的請求具有以下參數：

`keyPath`(Python:`key_path`)

(選擇性) 要更新之容器節點 (物件) 的金鑰路徑。指定清單，其中每個項目都是組態物件中單一層級的索引鍵。例如，指定要`port`在下列組態中設`{ "port": 443 }`定值的金鑰路徑`["mqtt"]`和合併值。

```
{
  "mqtt": {
    "port": 443
  }
}
```

索引鍵路徑必須在組態中指定容器節點 (物件)。如果該節點不存在於組件的配置中，則此操作會創建該節點並將其值設置為中的對象`valueToMerge`。

預設為配置物件的根目錄。

`timestamp`

目前 Unix 紀元時間 (以毫秒為單位)。此作業會使用此時間戳記來解析金鑰的並行更新。如果元件組態中的索引鍵的時間戳記大於要求中的時間戳記，則要求會失敗。

`valueToMerge`(Python:`value_to_merge`)

要在中指定的位置合併的配置物件`keyPath`。如需詳細資訊，請參閱 更新零組件組態。

回應

此操作在其響應中不提供任何信息。

SubscribeToConfigurationUpdate

訂閱以在元件的組態更新時接收通知。當您訂閱金鑰時，您會在該金鑰的任何子系更新時收到通知。

此作業是訂閱作業，您可以在其中訂閱事件訊息串流。若要使用此作業，請定義具有處理事件訊息、錯誤和資料流結束之函數的串流回應處理常式。如需詳細資訊，請參閱 訂閱 IPC 事件串流。

事件訊息類型：`ConfigurationUpdateEvents`

請求

此操作的請求具有以下參數：

`componentName`(Python:`component_name`)

(選擇性) 元件的名稱。

默認為發出請求的組件的名稱。

`keyPath`(Python:`key_path`)

要訂閱之組態值的金鑰路徑。指定清單，其中每個項目都是組態物件中單一層級的索引鍵。例如，指定`["mqtt", "port"]`以取得下列組態`port`中的值。

```
{
  "mqtt": {
    "port": 443
  }
}
```

若要訂閱元件組態中所有值的更新，請指定空白清單。

回應

此作業的回應包含下列資訊：

`messages`

通知訊息串流。此物件`ConfigurationUpdateEvents`包含下列資訊：

`configurationUpdateEvent`(Python:`configuration_update_event`)

組態更新事件。此物件`ConfigurationUpdateEvent`包含下列資訊：

componentName(Python:component_name)

元件的名稱。

keyPath(Python:key_path)

更新之組態值的索引鍵路徑。

SubscribeToValidateConfigurationUpdates

訂閱以在此元件的組態更新之前接收通知。這可讓元件驗證自己組態的更新。使用此SendConfigurationValidityReport作業告訴核心組態是否有效。

> **Important**
>
> 本機部署不會通知更新元件。

此作業是訂閱作業，您可以在其中訂閱事件訊息串流。若要使用此作業，請定義具有處理事件訊息、錯誤和資料流結束之函數的串流回應處理常式。如需詳細資訊，請參閱 訂閱 IPC 事件串流。

事件訊息類型：ValidateConfigurationUpdateEvents

請求

此操作的請求沒有任何參數。

回應

此作業的回應包含下列資訊：

messages

通知訊息串流。此物件ValidateConfigurationUpdateEvents包含下列資訊：

validateConfigurationUpdateEvent(Python:validate_configuration_update_event)

組態更新事件。此物件ValidateConfigurationUpdateEvent包含下列資訊：

deploymentId(Python:deployment_id)

更新元件的AWS IoT Greengrass部署識別碼。

`configuration`

包含新組態的物件。

SendConfigurationValidityReport

告訴核心對此元件的組態更新是否有效。如果您告訴核心新設定無效，則部署會失敗。使用此SubscribeToValidateConfigurationUpdates作業來訂閱以驗證組態更新。

如果元件未回應驗證組態更新通知，則核心會等待您在部署的組態驗證原則中指定的時間量。在該逾時之後，核心會繼續進行部署。預設元件驗證逾時為 20 秒。如需詳細資訊，請參閱建立部署和呼叫CreateDeployment作業時可提供的DeploymentConfigurationValidationPolicy物件。

請求

此操作的請求具有以下參數：

`configurationValidityReport`(Python:`configuration_validity_report`)

告知核心組態更新是否有效的報告。此物件`ConfigurationValidityReport`包含下列資訊：

`status`

有效性狀態。此枚舉具有以下值：`ConfigurationValidityStatus`

- `ACCEPTED`— 配置有效，細胞核可以將其應用於此組件。
- `REJECTED`— 設定無效且部署失敗。

`deploymentId`(Python:`deployment_id`)

要求組態更新的AWS IoT Greengrass部署識別碼。

`message`

(選擇性) 報告設定為何無效的訊息。

回應

此操作在其響應中不提供任何信息。

擷取秘密值

使用密碼管理員 IPC 服務從核心裝置上的密碼擷取密碼值。您可以使用 Secret Manager 元件將加密的密碼部署到核心裝置。然後，您可以使用 IPC 作業解密，並在自訂元件中使用其值。

主題

最低 SDK 版本

下表列出從核心裝置上AWS IoT Device SDK的密碼擷取秘密值所必須使用的最低版本。

SDK	最低版本	
AWS IoT Device SDK對於爪哇 V2	v1.2.10	
AWS IoT Device SDK對於 Python V2	V1.5.3	
AWS IoT Device SDK對於 C + + 第 2	v1.17.0	
AWS IoT Device SDK對於 JavaScript V2	v1.12.0	

授權

若要在自訂元件中使用密碼管理員，您必須定義授權原則，讓元件取得儲存在核心裝置上的密碼值。如需有關定義授權原則的資訊，請參閱授權元件執行 IPC 作業。

密碼管理員的授權原則具有下列屬性。

IPC 服務識別碼：`aws.greengrass.SecretManager`

操作	描述	資源
`aws.greengrass#GetSecretValue` 或 *	允許元件取得核心裝置上加密的密碼值。	Secrets Manager 秘密 ARN，或*允許訪問所有秘密。

授權政策範例

您可以參考下列授權原則範例，協助您設定元件的授權原則。

Example 授權政策範例

下列範例授權原則可讓元件取得核心裝置上任何密碼的值。

> **Note**
>
> 建議您在實際執行環境中減少授權原則的範圍，以便元件只擷取其使用的密碼。您可以在部署元件時，將*萬用字元變更為秘密 ARN 清單。

```
{
  "accessControl": {
    "aws.greengrass.SecretManager": {
      "com.example.MySecretComponent:secrets:1": {
        "policyDescription": "Allows access to a secret.",
        "operations": [
          "aws.greengrass#GetSecretValue"
        ],
        "resources": [
          "*"
        ]
      }
    }
  }
}
```

GetSecretValue

取得您儲存在核心裝置上的密碼值。

此作業類似於您可以用來取得中密碼值的 Secret Manager 作業AWS 雲端。如需詳細資訊，請參閱 AWS Secrets Manager API 參考中的 GetSecretValue。

請求

此操作的請求具有以下參數：

`secretId`(Python:`secret_id`)

要獲得的秘密的名稱。您可以指定 Amazon 資源名稱 (ARN) 或密碼的易記名稱。

`versionId`(Python:`version_id`)

(選擇性) 要取得的版本 ID。

您可指定為 `versionId` 或 `versionStage`。

如果未指定`versionId`或`versionStage`，則此作業預設為帶有`AWSCURRENT`標籤的版本。

`versionStage`(Python:`version_stage`)

(選擇性) 要取得之版本的暫存標籤。

您可指定為 `versionId` 或 `versionStage`。

如果未指定`versionId`或`versionStage`，則此作業預設為帶有`AWSCURRENT`標籤的版本。

回應

此作業的回應包含下列資訊：

`secretId`(Python:`secret_id`)

密碼的識別碼。

`versionId`(Python:`version_id`)

此密碼版本的識別碼。

`versionStage`(Python:`version_stage`)

附加至此密碼版本的暫存標籤清單。

`secretValue`(Python:`secret_value`)

這個版本的密碼的值。此物件`SecretValue`包含下列資訊。

`secretString`(Python:`secret_string`)

您以字串形式提供給 Secret Secrets Manager 之受保護機密資訊的解密部分。

`secretBinary`(Python:`secret_binary`)

(選擇性) 受保護機密資訊的解密部分，您以位元組陣列的二進位資料形式提供給 Secrets Manager。此內容包含以 base64 編碼字串的二進位資料。

如果您在秘密管理員主控台中建立密碼，則不會使用此屬性。

範例

下列範例示範如何在自訂元件程式碼中呼叫此作業。

Java (IPC client V1)

Example 範例：取得秘密值

> **Note**
>
> 此範例會使用`IPCUtils`類別來建立與 AWS IoT Greengrass Core IPC 服務的連線。如需詳細資訊，請參閱 Connect 到AWS IoT Greengrass酷睿 IPC 服務。

```
package com.aws.greengrass.docs.samples.ipc;

import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
import software.amazon.awssdk.aws.greengrass.GetSecretValueResponseHandler;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.model.GetSecretValueRequest;
import software.amazon.awssdk.aws.greengrass.model.GetSecretValueResponse;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;

import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class GetSecretValue {
```

```
   public static final int TIMEOUT_SECONDS = 10;

   public static void main(String[] args) {
       String secretArn = args[0];
       String versionStage = args[1];
       try (EventStreamRPCConnection eventStreamRPCConnection =
                    IPCUtils.getEventStreamRpcConnection()) {
           GreengrassCoreIPCClient ipcClient =
                   new GreengrassCoreIPCClient(eventStreamRPCConnection);
           GetSecretValueResponseHandler responseHandler =
                   GetSecretValue.getSecretValue(ipcClient, secretArn,
versionStage);
           CompletableFuture<GetSecretValueResponse> futureResponse =
                   responseHandler.getResponse();
           try {
               GetSecretValueResponse response =
futureResponse.get(TIMEOUT_SECONDS, TimeUnit.SECONDS);
               response.getSecretValue().postFromJson();
               String secretString = response.getSecretValue().getSecretString();
               System.out.println("Successfully retrieved secret value: " +
secretString);
           } catch (TimeoutException e) {
               System.err.println("Timeout occurred while retrieving secret: " +
secretArn);
           } catch (ExecutionException e) {
               if (e.getCause() instanceof UnauthorizedError) {
                   System.err.println("Unauthorized error while retrieving secret:
" + secretArn);
               } else {
                   throw e;
               }
           }
       } catch (InterruptedException e) {
           System.out.println("IPC interrupted.");
       } catch (ExecutionException e) {
           System.err.println("Exception occurred when using IPC.");
           e.printStackTrace();
           System.exit(1);
       }
   }

   public static GetSecretValueResponseHandler
getSecretValue(GreengrassCoreIPCClient greengrassCoreIPCClient, String secretArn,
String versionStage) {
```

```
        GetSecretValueRequest getSecretValueRequest = new GetSecretValueRequest();
        getSecretValueRequest.setSecretId(secretArn);
        getSecretValueRequest.setVersionStage(versionStage);
        return greengrassCoreIPCClient.getSecretValue(getSecretValueRequest,
 Optional.empty());
    }
}
```

Python (IPC client V1)

Example 範例：取得秘密值

> **Note**
>
> 這個範例假設您使用的是適用AWS IoT Device SDK於 Python v2 的 1.5.4 版或更新版本。

```
import json

import awsiot.greengrasscoreipc
from awsiot.greengrasscoreipc.model import (
    GetSecretValueRequest,
    GetSecretValueResponse,
    UnauthorizedError
)

secret_id = 'arn:aws:secretsmanager:us-
west-2:123456789012:secret:MyGreengrassSecret-abcdef'
TIMEOUT = 10

ipc_client = awsiot.greengrasscoreipc.connect()

request = GetSecretValueRequest()
request.secret_id = secret_id
request.version_stage = 'AWSCURRENT'
operation = ipc_client.new_get_secret_value()
operation.activate(request)
future_response = operation.get_response()
response = future_response.result(TIMEOUT)
secret_json = json.loads(response.secret_value.secret_string)
# Handle secret value.
```

JavaScript

Example 範例：取得秘密值

```
import {
    GetSecretValueRequest,
} from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc/model';
import * as greengrasscoreipc from "aws-iot-device-sdk-v2/dist/greengrasscoreipc";

class GetSecretValue {
    private readonly secretId : string;
    private readonly versionStage : string;
    private ipcClient : greengrasscoreipc.Client

    constructor() {
        this.secretId = "<define_your_own_secretId>"
        this.versionStage = "<define_your_own_versionStage>"

        this.getSecretValue().then(r => console.log("Started workflow"));
    }

    private async getSecretValue() {
        try {
            this.ipcClient = await getIpcClient();

            const getSecretValueRequest : GetSecretValueRequest = {
                secretId: this.secretId,
                versionStage: this.versionStage,
            };

            const result = await
 this.ipcClient.getSecretValue(getSecretValueRequest);
            const secretString = result.secretValue.secretString;
            console.log("Successfully retrieved secret value: " + secretString)
        } catch (e) {
            // parse the error depending on your use cases
            throw e
        }
    }
}

export async function getIpcClient(){
    try {
```

```
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

const getSecretValue = new GetSecretValue();
```

範例

使用下列範例來瞭解如何在元件中使用密碼管理員 IPC 服務。

範例：列印密碼 (Python、IPC 用戶端 V1)

此範例元件會列印您部署至核心裝置的密碼值。

> **Important**
>
> 這個範例元件會列印密碼的值，因此只能用於儲存測試資料的密碼。不要使用此組件來打印存儲重要信息的密鑰的值。

主題

- Recipe
- 成品
- 用量

Recipe

下列範例方案定義秘密 ARN 組態參數，並允許元件取得核心裝置上任何密碼的值。

> **Note**
>
> 建議您在實際執行環境中減少授權原則的範圍，以便元件只擷取其使用的密碼。您可以在部署元件時，將*萬用字元變更為秘密 ARN 清單。

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PrintSecret",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Prints the value of an AWS Secrets Manager secret.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.SecretManager": {
      "VersionRequirement": "^2.0.0",
      "DependencyType": "HARD"
    }
  },
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "SecretArn": "",
      "accessControl": {
        "aws.greengrass.SecretManager": {
          "com.example.PrintSecret:secrets:1": {
            "policyDescription": "Allows access to a secret.",
            "operations": [
              "aws.greengrass#GetSecretValue"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
```

```
      "Lifecycle": {
        "install": "python3 -m pip install --user awsiotsdk",
        "run": "python3 -u {artifacts:path}/print_secret.py \"{configuration:/
SecretArn}\""
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "py -3 -m pip install --user awsiotsdk",
        "run": "py -3 -u {artifacts:path}/print_secret.py \"{configuration:/
SecretArn}\""
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PrintSecret
ComponentVersion: 1.0.0
ComponentDescription: Prints the value of a Secrets Manager secret.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.SecretManager:
    VersionRequirement: "^2.0.0"
    DependencyType: HARD
ComponentConfiguration:
  DefaultConfiguration:
    SecretArn: ''
    accessControl:
      aws.greengrass.SecretManager:
        com.example.PrintSecret:secrets:1:
          policyDescription: Allows access to a secret.
          operations:
            - aws.greengrass#GetSecretValue
          resources:
            - "*"
Manifests:
```

```
  - Platform:
      os: linux
    Lifecycle:
      install: python3 -m pip install --user awsiotsdk
      run: python3 -u {artifacts:path}/print_secret.py "{configuration:/SecretArn}"
  - Platform:
      os: windows
    Lifecycle:
      install: py -3 -m pip install --user awsiotsdk
      run: py -3 -u {artifacts:path}/print_secret.py "{configuration:/SecretArn}"
```

成品

下面的示例 Python 應用程序演示了如何使用秘密管理器 IPC 服務來獲取核心設備上的密鑰的值。

```
import concurrent.futures
import json
import sys
import traceback

import awsiot.greengrasscoreipc
from awsiot.greengrasscoreipc.model import (
    GetSecretValueRequest,
    GetSecretValueResponse,
    UnauthorizedError
)

TIMEOUT = 10

if len(sys.argv) == 1:
    print('Provide SecretArn in the component configuration.', file=sys.stdout)
    exit(1)

secret_id = sys.argv[1]

try:
    ipc_client = awsiot.greengrasscoreipc.connect()

    request = GetSecretValueRequest()
    request.secret_id = secret_id
    operation = ipc_client.new_get_secret_value()
    operation.activate(request)
    future_response = operation.get_response()
```

```
    try:
        response = future_response.result(TIMEOUT)
        secret_json = json.loads(response.secret_value.secret_string)
        print('Successfully got secret: ' + secret_id)
        print('Secret value: ' + str(secret_json))
    except concurrent.futures.TimeoutError:
        print('Timeout occurred while getting secret: ' + secret_id, file=sys.stderr)
    except UnauthorizedError as e:
        print('Unauthorized error while getting secret: ' + secret_id,
 file=sys.stderr)
        raise e
    except Exception as e:
        print('Exception while getting secret: ' + secret_id, file=sys.stderr)
        raise e
except Exception:
    print('Exception occurred when using IPC.', file=sys.stderr)
    traceback.print_exc()
    exit(1)
```

用量

您可以將這個範例元件與 secret Manager 元件搭配使用，在核心裝置上部署和列印密碼的值。

若要建立、部署和列印測試密碼

1. 使用測試資料建立密碼管理員密碼。

   Linux or Unix

   ```
   aws secretsmanager create-secret \
     --name MyTestGreengrassSecret \
     --secret-string '{"my-secret-key": "my-secret-value"}'
   ```

   Windows Command Prompt (CMD)

   ```
   aws secretsmanager create-secret ^
     --name MyTestGreengrassSecret ^
     --secret-string '{"my-secret-key": "my-secret-value"}'
   ```

PowerShell

```
aws secretsmanager create-secret `
  --name MyTestGreengrassSecret `
  --secret-string '{"my-secret-key": "my-secret-value"}'
```

儲存密碼的 ARN，以便在下列步驟中使用。

若要取得更多資訊，請參閱《AWS Secrets Manager使用指南》中的〈建立密碼〉。

2. 使用下列組態合併更新部署秘密管理員元件 (aws.greengrass.SecretManager)。指定您先前建立之密碼的 ARN。

```
{
  "cloudSecrets": [
    {
      "arn": "arn:aws:secretsmanager:us-
west-2:123456789012:secret:MyTestGreengrassSecret-abcdef"
    }
  ]
}
```

如需詳細資訊，請參閱將AWS IoT Greengrass元件部署到裝置或 Greengrass CLI 部署命令。

3. 使用下列組態合併更新，在本節中建立及部署範例元件。指定您先前建立之密碼的 ARN。

```
{
  "SecretArn": "arn:aws:secretsmanager:us-
west-2:123456789012:secret:MyTestGreengrassSecret",
  "accessControl": {
    "aws.greengrass.SecretManager": {
      "com.example.PrintSecret:secrets:1": {
        "policyDescription": "Allows access to a secret.",
        "operations": [
          "aws.greengrass#GetSecretValue"
        ],
        "resources": [
          "arn:aws:secretsmanager:us-
west-2:123456789012:secret:MyTestGreengrassSecret-abcdef"
        ]
      }
```

```
    }
  }
}
```

如需更多資訊，請參閱 建立 AWS IoT Greengrass 元件

4. 檢視AWS IoT Greengrass核心軟體記錄檔以確認部署是否成功，並檢視`com.example.PrintSecret`元件記錄以查看列印的密碼值。如需更多詳細資訊，請參閱 監控AWS IoT Greengrass日誌。

與局部陰影互動

使用陰影 IPC 服務與設備上的局部陰影互動。您選擇與之互動的裝置可以是您的核心裝置或連線的用戶端裝置。

若要使用這些 IPC 作業，請在自訂元件中加入陰影管理員元件做為相依性。然後，您可以在自訂元件中使用 IPC 作業，透過陰影管理員與裝置上的局部陰影互動。若要啟用自訂元件回應本機陰影狀態中的變更，您也可以使用發佈/訂閱 IPC 服務來訂閱陰影事件。如需有關使用發佈/訂閱服務的詳細資訊，請參閱。發佈/訂閱本地訊息

> **Note**
>
> 若要讓核心裝置與用戶端裝置陰影互動，您還必須設定和部署 MQTT 橋接器元件。如需詳細資訊，請參閱啟用陰影管理員與用戶端裝置通訊。

主題

- 最低 SDK 版本
- 授權
- GetThingShadow
- UpdateThingShadow
- DeleteThingShadow
- ListNamedShadowsForThing

最低 SDK 版本

下表列出與局部陰影互動AWS IoT Device SDK所必須使用的最低版本。

SDK	最低版本	
AWS IoT Device SDK對於 Java V2	V1.4.0	
AWS IoT Device SDK對於 Python V2	v1.6.0	
AWS IoT Device SDK對於 C ++ 2	v1.17.0	
AWS IoT Device SDK對於 JavaScript V2	v1.12.0	

授權

若要在自訂元件中使用 shadow IPC 服務，您必須定義允許元件與陰影互動的授權原則。如需有關定義授權原則的資訊，請參閱授權元件執行 IPC 作業。

陰影互動的授權原則具有下列屬性。

IPC 服務識別碼：`aws.greengrass.ShadowManager`

操作	描述	資源
`aws.greengrass#GetThingShadow`	允許元件擷取物件的陰影。	下列其中一個字串： • `$aws/things/`*`thingName`*`/shadow/`，以允許存取傳統裝置陰影。 • `$aws/things/`*`thingName`*`/shadow/name/`*`shadowName`*，以允許存取已命名的陰影。 • `*`以允許存取所有陰影。

操作	描述	資源
aws.greengrass#UpdateThingShadow	允許元件更新物件的陰影。	下列其中一個字串： • $aws/things/ *thingName* /shadow/，以允許存取傳統裝置陰影。 • $aws/things/ *thingName* /shadow/name/ *shadowName* ，以允許存取已命名的陰影。 • *以允許存取所有陰影。
aws.greengrass#DeleteThingShadow	允許元件刪除物件的陰影。	下列其中一個字串： • $aws/things/ *thingName* /shadow/，以允許存取傳統裝置陰影 • $aws/things/ *thingName* /shadow/name/ *shadowName* ，以允許存取已命名的陰影 • *，以允許存取所有陰影。
aws.greengrass#ListNamedShadowsForThing	允許元件擷取物件的已命名陰影清單。	物件名稱字串，允許存取物件以列出其陰影。 用*於允許存取所有項目。

IPC 服務識別碼：aws.greengrass.ipc.pubsub

操作	描述	資源
aws.greengrass#SubscribeToTopic	允許元件訂閱您指定主題的訊息。	下列其中一個主題字串： • *shadowTopicPrefix* /get/accepted • *shadowTopicPrefix* /get/rejected • *shadowTopicPrefix* /delete/accepted • *shadowTopicPrefix* /delete/rejected • *shadowTopicPrefix* /update/accepted • *shadowTopicPrefix* /update/delta • *shadowTopicPrefix* /update/rejected 主題字首的值*shadowTopicPrefix* 取決於陰影的類型： • 經典陰影：$aws/things/ *thingName* /shadow • 命名陰影：$aws/things/ *thingName* /shadow/name/ *shadowName* 用*於允許存取所有主題。 在 Greengrass 核心 v2.6.0 及更新版本中，您可以訂閱包含 MQTT 主題萬用字元 (和) 的主

操作	描述	資源
		題。# +此主題字串支援 MQTT 主題萬用字元做為常值字元。例如，如果元件的授權原則授與存取權`test/topic/#` ，則該元件可以訂閱`test/topic/#` ，但無法訂閱`test/topic/filter` 。

本機陰影授權原則中的配方變數

如果您使用 v2.6.0 或更新版本的 Greengrass 核心，並將 `true` Greengrass 核心的interpolateComponentConfiguration組態選項設定為，您可以在授權原則中使用 recipe 變數。`{iot:thingName}`此功能可讓您為一組核心裝置設定單一授權原則，其中每個核心裝置只能存取自己的陰影。例如，您可以允許元件存取下列資源，以進行陰影 IPC 作業。

```
$aws/things/{iot:thingName}/shadow/
```

授權政策範例

您可以參考下列授權原則範例，協助您設定元件的授權原則。

Example 範例：允許一組核心裝置與局部陰影互動

> **Important**
>
> 此範例使用的功能可用於 v2.6.0 及更新版本的 Greeng rass 核元件。Greengrass 核 v2.6.0 增加了對大多數配方變量的支持，例如，在組件配置中。`{iot:thingName}`若要啟用此功能，請將 Greengrass 核心的組態選項設定為。interpolateComponentConfiguration`true`如需適用於所有 Greengrass 核心版本的範例，請參閱單一核心裝置的授權原則範例。

下列範例授權原則可`com.example.MyShadowInteractionComponent`讓元件與執行元件之核心裝置的傳統裝置陰影和具名陰影`myNamedShadow`互動。此原則也允許此元件接收有關這些陰影之本機主題的訊息。

JSON

```
{
  "accessControl": {
    "aws.greengrass.ShadowManager": {
      "com.example.MyShadowInteractionComponent:shadow:1": {
        "policyDescription": "Allows access to shadows",
        "operations": [
          "aws.greengrass#GetThingShadow",
          "aws.greengrass#UpdateThingShadow",
          "aws.greengrass#DeleteThingShadow"
        ],
        "resources": [
          "$aws/things/{iot:thingName}/shadow",
          "$aws/things/{iot:thingName}/shadow/name/myNamedShadow"
        ]
      },
      "com.example.MyShadowInteractionComponent:shadow:2": {
        "policyDescription": "Allows access to things with shadows",
        "operations": [
          "aws.greengrass#ListNamedShadowsForThing"
        ],
        "resources": [
          "{iot:thingName}"
        ]
      }
    },
    "aws.greengrass.ipc.pubsub": {
      "com.example.MyShadowInteractionComponent:pubsub:1": {
        "policyDescription": "Allows access to shadow pubsub topics",
        "operations": [
          "aws.greengrass#SubscribeToTopic"
        ],
        "resources": [
          "$aws/things/{iot:thingName}/shadow/get/accepted",
          "$aws/things/{iot:thingName}/shadow/name/myNamedShadow/get/accepted"
        ]
      }
    }
  }
}
```

YAML

```
accessControl:
  aws.greengrass.ShadowManager:
    'com.example.MyShadowInteractionComponent:shadow:1':
      policyDescription: 'Allows access to shadows'
      operations:
        - 'aws.greengrass#GetThingShadow'
        - 'aws.greengrass#UpdateThingShadow'
        - 'aws.greengrass#DeleteThingShadow'
      resources:
        - $aws/things/{iot:thingName}/shadow
        - $aws/things/{iot:thingName}/shadow/name/myNamedShadow
    'com.example.MyShadowInteractionComponent:shadow:2':
      policyDescription: 'Allows access to things with shadows'
      operations:
        - 'aws.greengrass#ListNamedShadowsForThing'
      resources:
        - '{iot:thingName}'
  aws.greengrass.ipc.pubsub:
    'com.example.MyShadowInteractionComponent:pubsub:1':
      policyDescription: 'Allows access to shadow pubsub topics'
      operations:
        - 'aws.greengrass#SubscribeToTopic'
      resources:
        - $aws/things/{iot:thingName}/shadow/get/accepted
        - $aws/things/{iot:thingName}/shadow/name/myNamedShadow/get/accepted
```

Example 範例：允許一組核心裝置與用戶端裝置陰影互動

> **Important**
>
> 此功能需要 Greengrass 2.6.0 或更新版本、陰影管理器 v2.2.0 或更新版本，以及 MQTT 橋接器 v2.2.0 或更新版本。您必須設定 MQTT 橋接器，才能讓陰影管理員與用戶端裝置通訊。

下列範例授權原則可讓元件`com.example.MyShadowInteractionComponent`與名稱開頭為之用戶端裝置的所有裝置陰影互動`MyClientDevice`。

> **Note**
>
> 若要讓核心裝置與用戶端裝置陰影互動，您還必須設定和部署 MQTT 橋接器元件。如需詳細資訊，請參閱啟用陰影管理員與用戶端裝置通訊。

JSON

```
{
  "accessControl": {
    "aws.greengrass.ShadowManager": {
      "com.example.MyShadowInteractionComponent:shadow:1": {
        "policyDescription": "Allows access to shadows",
        "operations": [
          "aws.greengrass#GetThingShadow",
          "aws.greengrass#UpdateThingShadow",
          "aws.greengrass#DeleteThingShadow"
        ],
        "resources": [
          "$aws/things/MyClientDevice*/shadow",
          "$aws/things/MyClientDevice*/shadow/name/*"
        ]
      },
      "com.example.MyShadowInteractionComponent:shadow:2": {
        "policyDescription": "Allows access to things with shadows",
        "operations": [
          "aws.greengrass#ListNamedShadowsForThing"
        ],
        "resources": [
          "MyClientDevice*"
        ]
      }
    }
  }
}
```

YAML

```
accessControl:
  aws.greengrass.ShadowManager:
    'com.example.MyShadowInteractionComponent:shadow:1':
      policyDescription: 'Allows access to shadows'
```

```
      operations:
        - 'aws.greengrass#GetThingShadow'
        - 'aws.greengrass#UpdateThingShadow'
        - 'aws.greengrass#DeleteThingShadow'
      resources:
        - $aws/things/MyClientDevice*/shadow
        - $aws/things/MyClientDevice*/shadow/name/*
    'com.example.MyShadowInteractionComponent:shadow:2':
      policyDescription: 'Allows access to things with shadows'
      operations:
        - 'aws.greengrass#ListNamedShadowsForThing'
      resources:
        - MyClientDevice*
```

Example 範例：允許單核心裝置與局部陰影互動

下列範例授權原則可`com.example.MyShadowInteractionComponent`讓元件與裝置的傳統裝置陰影和具名陰影`myNamedShadow`互動`MyThingName`。此原則也允許此元件接收有關這些陰影之本機主題的訊息。

JSON

```
{
  "accessControl": {
    "aws.greengrass.ShadowManager": {
      "com.example.MyShadowInteractionComponent:shadow:1": {
        "policyDescription": "Allows access to shadows",
        "operations": [
          "aws.greengrass#GetThingShadow",
          "aws.greengrass#UpdateThingShadow",
          "aws.greengrass#DeleteThingShadow"
        ],
        "resources": [
          "$aws/things/MyThingName/shadow",
          "$aws/things/MyThingName/shadow/name/myNamedShadow"
        ]
      },
      "com.example.MyShadowInteractionComponent:shadow:2": {
        "policyDescription": "Allows access to things with shadows",
        "operations": [
          "aws.greengrass#ListNamedShadowsForThing"
        ],
```

```
        "resources": [
          "MyThingName"
        ]
      }
    },
    "aws.greengrass.ipc.pubsub": {
      "com.example.MyShadowInteractionComponent:pubsub:1": {
        "policyDescription": "Allows access to shadow pubsub topics",
        "operations": [
          "aws.greengrass#SubscribeToTopic"
        ],
        "resources": [
          "$aws/things/MyThingName/shadow/get/accepted",
          "$aws/things/MyThingName/shadow/name/myNamedShadow/get/accepted"
        ]
      }
    }
  }
}
```

YAML

```
accessControl:
  aws.greengrass.ShadowManager:
    'com.example.MyShadowInteractionComponent:shadow:1':
      policyDescription: 'Allows access to shadows'
      operations:
        - 'aws.greengrass#GetThingShadow'
        - 'aws.greengrass#UpdateThingShadow'
        - 'aws.greengrass#DeleteThingShadow'
      resources:
        - $aws/things/MyThingName/shadow
        - $aws/things/MyThingName/shadow/name/myNamedShadow
    'com.example.MyShadowInteractionComponent:shadow:2':
      policyDescription: 'Allows access to things with shadows'
      operations:
        - 'aws.greengrass#ListNamedShadowsForThing'
      resources:
        - MyThingName
  aws.greengrass.ipc.pubsub:
    'com.example.MyShadowInteractionComponent:pubsub:1':
      policyDescription: 'Allows access to shadow pubsub topics'
      operations:
```

```
        - 'aws.greengrass#SubscribeToTopic'
      resources:
        - $aws/things/MyThingName/shadow/get/accepted
        - $aws/things/MyThingName/shadow/name/myNamedShadow/get/accepted
```

Example 範例：允許一組核心裝置回應本機陰影狀態變更

> **Important**
>
> 此範例使用的功能可用於 v2.6.0 及更新版本的 Greeng rass 核元件。Greengrass 核 v2.6.0 增加了對大多數配方變量的支持，例如，在組件配置中。`{iot:thingName}`若要啟用此功能，請將 Greengrass 核心的組態選項設定為。interpolateComponentConfiguration`true`如需適用於所有 Greengrass 核心版本的範例，請參閱單一核心裝置的授權原則範例。

下列範例存取控制原則允許自訂`com.example.MyShadowReactiveComponent`接收有關傳統裝置陰影和執行元件之每個核心裝置`myNamedShadow`上具名陰影`/update/delta`主題的訊息。

JSON

```
{
  "accessControl": {
    "aws.greengrass.ipc.pubsub": {
      "com.example.MyShadowReactiveComponent:pubsub:1": {
        "policyDescription": "Allows access to shadow pubsub topics",
        "operations": [
          "aws.greengrass#SubscribeToTopic"
        ],
        "resources": [
          "$aws/things/{iot:thingName}/shadow/update/delta",
          "$aws/things/{iot:thingName}/shadow/name/myNamedShadow/update/delta"
        ]
      }
    }
  }
}
```

YAML

```
accessControl:
```

```
  aws.greengrass.ipc.pubsub:
    "com.example.MyShadowReactiveComponent:pubsub:1":
      policyDescription: Allows access to shadow pubsub topics
      operations:
        - 'aws.greengrass#SubscribeToTopic'
      resources:
        - $aws/things/{iot:thingName}/shadow/update/delta
        - $aws/things/{iot:thingName}/shadow/name/myNamedShadow/update/delta
```

Example 範例：允許單一核心裝置回應本機陰影狀態變更

下列範例存取控制原則允許自訂`com.example.MyShadowReactiveComponent`接收有關傳統裝置陰影和裝置具名陰影`/update/delta`主題`myNamedShadow`的訊息`MyThingName`。

JSON

```
{
  "accessControl": {
    "aws.greengrass.ipc.pubsub": {
      "com.example.MyShadowReactiveComponent:pubsub:1": {
        "policyDescription": "Allows access to shadow pubsub topics",
        "operations": [
          "aws.greengrass#SubscribeToTopic"
        ],
        "resources": [
          "$aws/things/MyThingName/shadow/update/delta",
          "$aws/things/MyThingName/shadow/name/myNamedShadow/update/delta"
        ]
      }
    }
  }
}
```

YAML

```
accessControl:
  aws.greengrass.ipc.pubsub:
    "com.example.MyShadowReactiveComponent:pubsub:1":
      policyDescription: Allows access to shadow pubsub topics
      operations:
        - 'aws.greengrass#SubscribeToTopic'
```

```
      resources:
        - $aws/things/MyThingName/shadow/update/delta
        - $aws/things/MyThingName/shadow/name/myNamedShadow/update/delta
```

GetThingShadow

取得指定物件的陰影。

請求

此操作的請求具有以下參數：

`thingName`(Python:`thing_name`)

物件名稱。

類型：`string`

`shadowName`(Python:`shadow_name`)

影子的名稱。若要指定物件的經典陰影，請將此參數設定為空字串 ("")。

> **Warning**
>
> 此AWS IoT Greengrass服務會使用`AWSManagedGreengrassV2Deployment`具名的陰影來管理以個別核心裝置為目標的部署。此具名陰影會保留供AWS IoT Greengrass服務使用。請勿更新或刪除此命名陰影。

類型：`string`

回應

此作業的回應包含下列資訊：

`payload`

回應狀態文件為 blob。

類型：`object`包含下列資訊：

state

狀態資訊。

此物件包含下列資訊。

desired

要求在設備中更新的狀態屬性和值。

類型：map鍵值對

reported

設備報告的狀態屬性和值。

類型：map鍵值對

delta

所需和報告的狀態屬性和值之間的差異。只有在desired和reported狀態不同時，才會顯示此屬性。

類型：map鍵值對

metadata

desired和reported區段中每個屬性的時間戳記，以便您判斷狀態的更新時間。

類型：string

timestamp

產生回應的紀元日期和時間。

類型：integer

clientToken(Python:clientToken)

用於匹配請求和相應響應的令牌

類型：string

version

本機陰影文件的版本。

類型：integer

錯誤

此作業可能會傳回下列錯誤。

`InvalidArgumentsError`

本機陰影服務無法驗證要求參數。如果要求包含格式錯誤或不支援的字元，就會發生這種情況。

`ResourceNotFoundError`

找不到要求的本機陰影文件。

`ServiceError`

發生內部服務錯誤，或 IPC 服務的要求數目超過陰影管理員元件`maxLocalRequestsPerSecondPerThing`和`maxTotalLocalRequestsRate`組態參數中指定的限制。

`UnauthorizedError`

元件的授權原則不包含此作業所需的權限。

範例

下列範例示範如何在自訂元件程式碼中呼叫此作業。

Java (IPC client V1)

Example 範例：取得物件陰影

> **Note**
>
> 此範例會使用`IPCUtils`類別來建立與 AWS IoT Greengrass Core IPC 服務的連線。如需詳細資訊，請參閱 Connect 到AWS IoT Greengrass酷睿 IPC 服務。

```
package com.aws.greengrass.docs.samples.ipc;

import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
import software.amazon.awssdk.aws.greengrass.GetThingShadowResponseHandler;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.model.GetThingShadowRequest;
import software.amazon.awssdk.aws.greengrass.model.GetThingShadowResponse;
import software.amazon.awssdk.aws.greengrass.model.ResourceNotFoundError;
```

```
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;

import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class GetThingShadow {

    public static final int TIMEOUT_SECONDS = 10;

    public static void main(String[] args) {
        // Use the current core device's name if thing name isn't set.
        String thingName = args[0].isEmpty() ? System.getenv("AWS_IOT_THING_NAME") :
 args[0];
        String shadowName = args[1];
        try (EventStreamRPCConnection eventStreamRPCConnection =
                     IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient =
                    new GreengrassCoreIPCClient(eventStreamRPCConnection);
            GetThingShadowResponseHandler responseHandler =
                    GetThingShadow.getThingShadow(ipcClient, thingName,
 shadowName);
            CompletableFuture<GetThingShadowResponse> futureResponse =
                    responseHandler.getResponse();
            try {
                GetThingShadowResponse response =
 futureResponse.get(TIMEOUT_SECONDS,
                        TimeUnit.SECONDS);
                String shadowPayload = new String(response.getPayload(),
 StandardCharsets.UTF_8);
                System.out.printf("Successfully got shadow %s/%s: %s%n", thingName,
 shadowName,
                        shadowPayload);
            } catch (TimeoutException e) {
                System.err.printf("Timeout occurred while getting shadow: %s/%s%n",
 thingName,
                        shadowName);
            } catch (ExecutionException e) {
                if (e.getCause() instanceof UnauthorizedError) {
```

```
                    System.err.printf("Unauthorized error while getting shadow: %s/
%s%n",
                            thingName, shadowName);
                } else if (e.getCause() instanceof ResourceNotFoundError) {
                    System.err.printf("Unable to find shadow to get: %s/%s%n",
 thingName,
                            shadowName);
                } else {
                    throw e;
                }
            }
        } catch (InterruptedException e) {
            System.out.println("IPC interrupted.");
        } catch (ExecutionException e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static GetThingShadowResponseHandler
 getThingShadow(GreengrassCoreIPCClient greengrassCoreIPCClient, String thingName,
 String shadowName) {
        GetThingShadowRequest getThingShadowRequest = new GetThingShadowRequest();
        getThingShadowRequest.setThingName(thingName);
        getThingShadowRequest.setShadowName(shadowName);
        return greengrassCoreIPCClient.getThingShadow(getThingShadowRequest,
 Optional.empty());
    }
}
```

Python (IPC client V1)

Example 範例：取得物件陰影

```
import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import GetThingShadowRequest

TIMEOUT = 10

def sample_get_thing_shadow_request(thingName, shadowName):
    try:
        # set up IPC client to connect to the IPC server
```

```
        ipc_client = awsiot.greengrasscoreipc.connect()

        # create the GetThingShadow request
        get_thing_shadow_request = GetThingShadowRequest()
        get_thing_shadow_request.thing_name = thingName
        get_thing_shadow_request.shadow_name = shadowName

        # retrieve the GetThingShadow response after sending the request to the IPC
 server
        op = ipc_client.new_get_thing_shadow()
        op.activate(get_thing_shadow_request)
        fut = op.get_response()

        result = fut.result(TIMEOUT)
        return result.payload

    except InvalidArgumentsError as e:
        # add error handling
        ...
    # except ResourceNotFoundError | UnauthorizedError | ServiceError
```

JavaScript

Example 範例：取得物件陰影

```
import {
    GetThingShadowRequest
} from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc/model';
import * as greengrasscoreipc from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc';

class GetThingShadow {
    private ipcClient: greengrasscoreipc.Client;
    private thingName: string;
    private shadowName: string;

    constructor() {
        // Define args parameters here
        this.thingName = "<define_your_own_thingName>";
        this.shadowName = "<define_your_own_shadowName>";
        this.bootstrap();
    }

    async bootstrap() {
        try {
```

```
            this.ipcClient = await getIpcClient();
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }

        try {
            await this.handleGetThingShadowOperation(this.thingName,
                this.shadowName);
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }
    }

    async handleGetThingShadowOperation(
        thingName: string,
        shadowName: string
    ) {
        const request: GetThingShadowRequest = {
            thingName: thingName,
            shadowName: shadowName
        };
        const response = await this.ipcClient.getThingShadow(request);
    }
}

export async function getIpcClient() {
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use caseså
        throw err
    }
}

const startScript = new GetThingShadow();
```

UpdateThingShadow

更新指定物件的陰影。如果陰影不存在，則會建立一個陰影。

請求

此操作的請求具有以下參數：

`thingName`(Python:`thing_name`)

物件名稱。

類型：`string`

`shadowName`(Python:`shadow_name`)

影子的名稱。若要指定物件的經典陰影，請將此參數設定為空字串 ("")。

> **Warning**
>
> 此AWS IoT Greengrass服務會使用`AWSManagedGreengrassV2Deployment`具名的陰影來管理以個別核心裝置為目標的部署。此具名陰影會保留供AWS IoT Greengrass服務使用。請勿更新或刪除此命名陰影。

類型：`string`

`payload`

請求狀態文檔作為一個 blob。

類型：`object`包含下列資訊：

`state`

要更新的狀態資訊。此 IPC 操作僅影響指定的字段。

此物件包含下列資訊。一般而言，您會在同一個要求中使用`reported`屬性或屬性，但不能同時使用這兩者。`desired`

`desired`

要求在設備中更新的狀態屬性和值。

類型：`map`鍵值對

reported

設備報告的狀態屬性和值。

類型：map鍵值對

clientToken(Python:client_token)

(選擇性) 用來比對要求和用戶端權杖對應回應的權杖。

類型：string

version

(選擇性) 要更新的本機陰影文件版本。只有當指定的版本與其擁有的最新版本相符時，陰影服務才會處理更新。

類型：integer

回應

此作業的回應包含下列資訊：

payload

回應狀態文件為 blob。

類型：object包含下列資訊：

state

狀態資訊。

此物件包含下列資訊。

desired

要求在設備中更新的狀態屬性和值。

類型：map鍵值對

reported

設備報告的狀態屬性和值。

類型：map鍵值對

delta

設備報告的狀態屬性和值。

類型：map鍵值對

metadata

desired和reported區段中每個屬性的時間戳記，以便您判斷狀態的更新時間。

類型：string

timestamp

產生回應的紀元日期和時間。

類型：integer

clientToken(Python:client_token)

用於匹配請求和相應響應的令牌。

類型：string

version

更新完成後的本機陰影文件版本。

類型：integer

錯誤

此作業可能會傳回下列錯誤。

ConflictError

本機陰影服務在更新作業期間遇到版本衝突。當請求裝載中的版本與最新可用的本機陰影文件中的版本不符時，就會發生這種情況。

InvalidArgumentsError

本機陰影服務無法驗證要求參數。如果要求包含格式錯誤或不支援的字元，就會發生這種情況。

一個有效的payload具有以下屬性：

- state節點存在，且是包含desired或reported狀態資訊的物件。

- desired和reported節點可以是物件或空值。這些物件中至少有一個必須包含有效的狀態資訊。
- desired和reported物件的深度不能超過八個節點。
- clientToken值的長度不能超過 64 個字元。
- 該version值必須是1或更高。

ServiceError

發生內部服務錯誤，或 IPC 服務的要求數目超過陰影管理員元件maxLocalRequestsPerSecondPerThing和maxTotalLocalRequestsRate組態參數中指定的限制。

UnauthorizedError

元件的授權原則不包含此作業所需的權限。

範例

下列範例示範如何在自訂元件程式碼中呼叫此作業。

Java (IPC client V1)

Example 範例：更新物件陰影

> **Note**
>
> 此範例會使用IPCUtils類別來建立與 AWS IoT Greengrass Core IPC 服務的連線。如需詳細資訊，請參閱 Connect 到AWS IoT Greengrass酷睿 IPC 服務。

```
package com.aws.greengrass.docs.samples.ipc;

import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.UpdateThingShadowResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.aws.greengrass.model.UpdateThingShadowRequest;
import software.amazon.awssdk.aws.greengrass.model.UpdateThingShadowResponse;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;

import java.nio.charset.StandardCharsets;
```

```
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class UpdateThingShadow {

    public static final int TIMEOUT_SECONDS = 10;

    public static void main(String[] args) {
        // Use the current core device's name if thing name isn't set.
        String thingName = args[0].isEmpty() ? System.getenv("AWS_IOT_THING_NAME") :
 args[0];
        String shadowName = args[1];
        byte[] shadowPayload = args[2].getBytes(StandardCharsets.UTF_8);
        try (EventStreamRPCConnection eventStreamRPCConnection =
                     IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient =
                    new GreengrassCoreIPCClient(eventStreamRPCConnection);
            UpdateThingShadowResponseHandler responseHandler =
                    UpdateThingShadow.updateThingShadow(ipcClient, thingName,
 shadowName,
                            shadowPayload);
            CompletableFuture<UpdateThingShadowResponse> futureResponse =
                    responseHandler.getResponse();
            try {
                futureResponse.get(TIMEOUT_SECONDS, TimeUnit.SECONDS);
                System.out.printf("Successfully updated shadow: %s/%s%n", thingName,
 shadowName);
            } catch (TimeoutException e) {
                System.err.printf("Timeout occurred while updating shadow: %s/%s%n",
 thingName,
                        shadowName);
            } catch (ExecutionException e) {
                if (e.getCause() instanceof UnauthorizedError) {
                    System.err.printf("Unauthorized error while updating shadow: %s/
%s%n",
                            thingName, shadowName);
                } else {
                    throw e;
                }
            }
        } catch (InterruptedException e) {
```

```
            System.out.println("IPC interrupted.");
        } catch (ExecutionException e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static UpdateThingShadowResponseHandler
 updateThingShadow(GreengrassCoreIPCClient greengrassCoreIPCClient, String
 thingName, String shadowName, byte[] shadowPayload) {
        UpdateThingShadowRequest updateThingShadowRequest = new
 UpdateThingShadowRequest();
        updateThingShadowRequest.setThingName(thingName);
        updateThingShadowRequest.setShadowName(shadowName);
        updateThingShadowRequest.setPayload(shadowPayload);
        return greengrassCoreIPCClient.updateThingShadow(updateThingShadowRequest,
                Optional.empty());
    }
}
```

Python (IPC client V1)

Example 範例：更新物件陰影

```
import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import UpdateThingShadowRequest

TIMEOUT = 10

def sample_update_thing_shadow_request(thingName, shadowName, payload):
    try:
        # set up IPC client to connect to the IPC server
        ipc_client = awsiot.greengrasscoreipc.connect()

        # create the UpdateThingShadow request
        update_thing_shadow_request = UpdateThingShadowRequest()
        update_thing_shadow_request.thing_name = thingName
        update_thing_shadow_request.shadow_name = shadowName
        update_thing_shadow_request.payload = payload

        # retrieve the UpdateThingShadow response after sending the request to the
 IPC server
```

```
        op = ipc_client.new_update_thing_shadow()
        op.activate(update_thing_shadow_request)
        fut = op.get_response()

        result = fut.result(TIMEOUT)
        return result.payload

    except InvalidArgumentsError as e:
        # add error handling
    ...
    # except ConflictError | UnauthorizedError | ServiceError
```

JavaScript

Example 範例：更新物件陰影

```
import {
    UpdateThingShadowRequest
} from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc/model';
import * as greengrasscoreipc from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc';

class UpdateThingShadow {
    private ipcClient: greengrasscoreipc.Client;
    private thingName: string;
    private shadowName: string;
    private shadowDocumentStr: string;

    constructor() {
        // Define args parameters here

        this.thingName = "<define_your_own_thingName>";
        this.shadowName = "<define_your_own_shadowName>";
        this.shadowDocumentStr = "<define_your_own_payload>";

        this.bootstrap();
    }

    async bootstrap() {
        try {
            this.ipcClient = await getIpcClient();
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }
```

```
        try {
            await this.handleUpdateThingShadowOperation(
                this.thingName,
                this.shadowName,
                this.shadowDocumentStr);
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }
    }

    async handleUpdateThingShadowOperation(
        thingName: string,
        shadowName: string,
        payloadStr: string
    ) {
        const request: UpdateThingShadowRequest = {
            thingName: thingName,
            shadowName: shadowName,
            payload: payloadStr
        }
        // make the UpdateThingShadow request
        const response = await this.ipcClient.updateThingShadow(request);
    }
}

export async function getIpcClient() {
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

const startScript = new UpdateThingShadow();
```

DeleteThingShadow

刪除特定物件的影子。

從陰影管理器 v2.0.4 開始，刪除陰影會增加版本號。例如，當您刪除版本 1 的陰影`MyThingShadow`時，已刪除陰影的版本為 2。如果您接著使用名稱重新建立陰影`MyThingShadow`，則該陰影的版本為 3。

請求

此操作的請求具有以下參數：

`thingName`(Python:`thing_name`)

物件名稱。

類型：`string`

`shadowName`(Python:`shadow_name`)

影子的名稱。若要指定物件的經典陰影，請將此參數設定為空字串 ("")。

> **Warning**
>
> 此AWS IoT Greengrass服務會使用`AWSManagedGreengrassV2Deployment`具名的陰影來管理以個別核心裝置為目標的部署。此具名陰影會保留供AWS IoT Greengrass服務使用。請勿更新或刪除此命名陰影。

類型：`string`

回應

此作業的回應包含下列資訊：

`payload`

一個空的響應狀態文檔。

錯誤

此作業可能會傳回下列錯誤。

`InvalidArgumentsError`

本機陰影服務無法驗證要求參數。如果要求包含格式錯誤或不支援的字元，就會發生這種情況。

`ResourceNotFoundError`

找不到要求的本機陰影文件。

`ServiceError`

發生內部服務錯誤，或 IPC 服務的要求數目超過陰影管理員元件`maxLocalRequestsPerSecondPerThing`和`maxTotalLocalRequestsRate`組態參數中指定的限制。

`UnauthorizedError`

元件的授權原則不包含此作業所需的權限。

範例

下列範例示範如何在自訂元件程式碼中呼叫此作業。

Java (IPC client V1)

Example 範例：刪除物件陰影

> **Note**
>
> 此範例會使用`IPCUtils`類別來建立與 AWS IoT Greengrass Core IPC 服務的連線。如需詳細資訊，請參閱 Connect 到AWS IoT Greengrass酷睿 IPC 服務。

```
package com.aws.greengrass.docs.samples.ipc;

import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
import software.amazon.awssdk.aws.greengrass.DeleteThingShadowResponseHandler;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.model.DeleteThingShadowRequest;
import software.amazon.awssdk.aws.greengrass.model.DeleteThingShadowResponse;
import software.amazon.awssdk.aws.greengrass.model.ResourceNotFoundError;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;

import java.util.Optional;
```

```
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class DeleteThingShadow {

    public static final int TIMEOUT_SECONDS = 10;

    public static void main(String[] args) {
        // Use the current core device's name if thing name isn't set.
        String thingName = args[0].isEmpty() ? System.getenv("AWS_IOT_THING_NAME") :
 args[0];
        String shadowName = args[1];
        try (EventStreamRPCConnection eventStreamRPCConnection =
                     IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient =
                    new GreengrassCoreIPCClient(eventStreamRPCConnection);
            DeleteThingShadowResponseHandler responseHandler =
                    DeleteThingShadow.deleteThingShadow(ipcClient, thingName,
 shadowName);
            CompletableFuture<DeleteThingShadowResponse> futureResponse =
                    responseHandler.getResponse();
            try {
                futureResponse.get(TIMEOUT_SECONDS, TimeUnit.SECONDS);
                System.out.printf("Successfully deleted shadow: %s/%s%n", thingName,
 shadowName);
            } catch (TimeoutException e) {
                System.err.printf("Timeout occurred while deleting shadow: %s/%s%n",
 thingName,
                        shadowName);
            } catch (ExecutionException e) {
                if (e.getCause() instanceof UnauthorizedError) {
                    System.err.printf("Unauthorized error while deleting shadow: %s/
%s%n",
                            thingName, shadowName);
                } else if (e.getCause() instanceof ResourceNotFoundError) {
                    System.err.printf("Unable to find shadow to delete: %s/%s%n",
 thingName,
                            shadowName);
                } else {
                    throw e;
                }
            }
```

```
        } catch (InterruptedException e) {
            System.out.println("IPC interrupted.");
        } catch (ExecutionException e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static DeleteThingShadowResponseHandler
 deleteThingShadow(GreengrassCoreIPCClient greengrassCoreIPCClient, String
 thingName, String shadowName) {
        DeleteThingShadowRequest deleteThingShadowRequest = new
 DeleteThingShadowRequest();
        deleteThingShadowRequest.setThingName(thingName);
        deleteThingShadowRequest.setShadowName(shadowName);
        return greengrassCoreIPCClient.deleteThingShadow(deleteThingShadowRequest,
                Optional.empty());
    }
}
```

Python (IPC client V1)

Example 範例：刪除物件陰影

```
import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import DeleteThingShadowRequest

TIMEOUT = 10

def sample_delete_thing_shadow_request(thingName, shadowName):
    try:
        # set up IPC client to connect to the IPC server
        ipc_client = awsiot.greengrasscoreipc.connect()

        # create the DeleteThingShadow request
        delete_thing_shadow_request = DeleteThingShadowRequest()
        delete_thing_shadow_request.thing_name = thingName
        delete_thing_shadow_request.shadow_name = shadowName

        # retrieve the DeleteThingShadow response after sending the request to the
 IPC server
        op = ipc_client.new_delete_thing_shadow()
```

```
        op.activate(delete_thing_shadow_request)
        fut = op.get_response()

        result = fut.result(TIMEOUT)
        return result.payload

    except InvalidArgumentsError as e:
        # add error handling
    ...
    # except ResourceNotFoundError | UnauthorizedError | ServiceError
```

JavaScript

Example 範例：刪除物件陰影

```
import {
    DeleteThingShadowRequest
} from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc/model';
import * as greengrasscoreipc from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc';

class DeleteThingShadow {
    private ipcClient: greengrasscoreipc.Client;
    private thingName: string;
    private shadowName: string;

    constructor() {
        // Define args parameters here
        this.thingName = "<define_your_own_thingName>";
        this.shadowName = "<define_your_own_shadowName>";
        this.bootstrap();
    }

    async bootstrap() {
        try {
            this.ipcClient = await getIpcClient();
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }

        try {
            await this.handleDeleteThingShadowOperation(this.thingName,
 this.shadowName)
        } catch (err) {
```

```
            // parse the error depending on your use cases
            throw err
        }
    }

    async handleDeleteThingShadowOperation(thingName: string, shadowName: string) {
        const request: DeleteThingShadowRequest = {
            thingName: thingName,
            shadowName: shadowName
        }
        // make the DeleteThingShadow request
        const response = await this.ipcClient.deleteThingShadow(request);
    }
}

export async function getIpcClient() {
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

const startScript = new DeleteThingShadow();
```

ListNamedShadowsForThing

列出指定物件的命名陰影。

請求

此操作的請求具有以下參數：

`thingName`(Python:`thing_name`)

物件名稱。

類型：`string`

`pageSize`(Python:`page_size`)

(選擇性) 每次呼叫中要傳回的陰影名稱數目。

類型：`integer`

預設：25

上限：100

`nextToken`(Python:`next_token`)

(選擇性) 用來擷取下一組結果的權杖。此值會在分頁結果上傳回，並在傳回下一頁的呼叫中使用。

類型：`string`

回應

此作業的回應包含下列資訊：

`results`

陰影名稱的清單。

類型：`array`

`timestamp`

(選擇性) 產生回應的日期和時間。

類型：`integer`

`nextToken`(Python:`next_token`)

(選擇性) 在分頁要求中用來擷取序列中下一頁的權杖值。當沒有更多的陰影名稱返回時，此令牌不存在。

類型：`string`

> **Note**
>
> 如果請求的頁面大小與響應中的陰影名稱的數量完全匹配，則存在此令牌; 但是，使用時，它返回一個空列表。

錯誤

此作業可能會傳回下列錯誤。

`InvalidArgumentsError`

本機陰影服務無法驗證要求參數。如果要求包含格式錯誤或不支援的字元，就會發生這種情況。

`ResourceNotFoundError`

找不到要求的本機陰影文件。

`ServiceError`

發生內部服務錯誤，或 IPC 服務的要求數目超過陰影管理員元件`maxLocalRequestsPerSecondPerThing`和`maxTotalLocalRequestsRate`組態參數中指定的限制。

`UnauthorizedError`

元件的授權原則不包含此作業所需的權限。

範例

下列範例示範如何在自訂元件程式碼中呼叫此作業。

Java (IPC client V1)

Example 範例：列出物件的命名陰影

> **Note**
>
> 此範例會使用`IPCUtils`類別來建立與 AWS IoT Greengrass Core IPC 服務的連線。如需詳細資訊，請參閱 Connect 到AWS IoT Greengrass酷睿 IPC 服務。

```
package com.aws.greengrass.docs.samples.ipc;

import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import
 software.amazon.awssdk.aws.greengrass.ListNamedShadowsForThingResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.ListNamedShadowsForThingRequest;
```

```
import
 software.amazon.awssdk.aws.greengrass.model.ListNamedShadowsForThingResponse;
import software.amazon.awssdk.aws.greengrass.model.ResourceNotFoundError;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;

import java.util.ArrayList;
import java.util.List;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class ListNamedShadowsForThing {

    public static final int TIMEOUT_SECONDS = 10;

    public static void main(String[] args) {
        // Use the current core device's name if thing name isn't set.
        String thingName = args[0].isEmpty() ? System.getenv("AWS_IOT_THING_NAME") :
 args[0];
        try (EventStreamRPCConnection eventStreamRPCConnection =
                     IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient =
                    new GreengrassCoreIPCClient(eventStreamRPCConnection);
            List<String> namedShadows = new ArrayList<>();
            String nextToken = null;
            try {
                // Send additional requests until there's no pagination token in the
 response.
                do {
                    ListNamedShadowsForThingResponseHandler responseHandler =

 ListNamedShadowsForThing.listNamedShadowsForThing(ipcClient, thingName,
                                    nextToken, 25);
                    CompletableFuture<ListNamedShadowsForThingResponse>
 futureResponse =
                            responseHandler.getResponse();
                    ListNamedShadowsForThingResponse response =
                            futureResponse.get(TIMEOUT_SECONDS, TimeUnit.SECONDS);
                    List<String> responseNamedShadows = response.getResults();
                    namedShadows.addAll(responseNamedShadows);
                    nextToken = response.getNextToken();
```

```
                } while (nextToken != null);
                System.out.printf("Successfully got named shadows for thing %s: %s
%n", thingName,
                        String.join(",", namedShadows));
            } catch (TimeoutException e) {
                System.err.println("Timeout occurred while listing named shadows for
 thing: " + thingName);
            } catch (ExecutionException e) {
                if (e.getCause() instanceof UnauthorizedError) {
                    System.err.println("Unauthorized error while listing named
 shadows for " +
                            "thing: " + thingName);
                } else if (e.getCause() instanceof ResourceNotFoundError) {
                    System.err.println("Unable to find thing to list named shadows:
" + thingName);
                } else {
                    throw e;
                }
            }
        } catch (InterruptedException e) {
            System.out.println("IPC interrupted.");
        } catch (ExecutionException e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static ListNamedShadowsForThingResponseHandler
 listNamedShadowsForThing(GreengrassCoreIPCClient greengrassCoreIPCClient, String
 thingName, String nextToken, int pageSize) {
        ListNamedShadowsForThingRequest listNamedShadowsForThingRequest =
                new ListNamedShadowsForThingRequest();
        listNamedShadowsForThingRequest.setThingName(thingName);
        listNamedShadowsForThingRequest.setNextToken(nextToken);
        listNamedShadowsForThingRequest.setPageSize(pageSize);
        return
 greengrassCoreIPCClient.listNamedShadowsForThing(listNamedShadowsForThingRequest,
                Optional.empty());
    }
}
```

Python (IPC client V1)

Example 範例：列出物件的命名陰影

```
import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import ListNamedShadowsForThingRequest

TIMEOUT = 10

def sample_list_named_shadows_for_thing_request(thingName, nextToken, pageSize):
    try:
        # set up IPC client to connect to the IPC server
        ipc_client = awsiot.greengrasscoreipc.connect()

        # create the ListNamedShadowsForThingRequest request
        list_named_shadows_for_thing_request = ListNamedShadowsForThingRequest()
        list_named_shadows_for_thing_request.thing_name = thingName
        list_named_shadows_for_thing_request.next_token = nextToken
        list_named_shadows_for_thing_request.page_size = pageSize

        # retrieve the ListNamedShadowsForThingRequest response after sending the
 request to the IPC server
        op = ipc_client.new_list_named_shadows_for_thing()
        op.activate(list_named_shadows_for_thing_request)
        fut = op.get_response()

        list_result = fut.result(TIMEOUT)

        # additional returned fields
        timestamp = list_result.timestamp
        next_token = result.next_token
        named_shadow_list = list_result.results

        return named_shadow_list, next_token, timestamp

    except InvalidArgumentsError as e:
        # add error handling
    ...
    # except ResourceNotFoundError | UnauthorizedError | ServiceError
```

JavaScript

Example 範例：列出物件的命名陰影

```
import {
    ListNamedShadowsForThingRequest
} from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc/model';
import * as greengrasscoreipc from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc';

class listNamedShadowsForThing {
    private ipcClient: greengrasscoreipc.Client;
    private thingName: string;
    private pageSizeStr: string;
    private nextToken: string;

    constructor() {
        // Define args parameters here
        this.thingName = "<define_your_own_thingName>";
        this.pageSizeStr = "<define_your_own_pageSize>";
        this.nextToken = "<define_your_own_token>";
        this.bootstrap();
    }

    async bootstrap() {
        try {
            this.ipcClient = await getIpcClient();
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }

        try {
            await this.handleListNamedShadowsForThingOperation(this.thingName,
                this.nextToken, this.pageSizeStr);
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }
    }

    async handleListNamedShadowsForThingOperation(
        thingName: string,
        nextToken: string,
        pageSizeStr: string
```

```
    ) {
        let request: ListNamedShadowsForThingRequest = {
            thingName: thingName,
            nextToken: nextToken,
        };
        if (pageSizeStr) {
            request.pageSize = parseInt(pageSizeStr);
        }
        // make the ListNamedShadowsForThing request
        const response = await this.ipcClient.listNamedShadowsForThing(request);
        const shadowNames = response.results;
    }
}

export async function getIpcClient(){
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

const startScript = new listNamedShadowsForThing();
```

管理本機部署和元件

> **Note**
>
> 此功能適用於 v2.6.0 及更高版 Greeng rass 核組件。

使用 Greengrass CLI IPC 服務來管理核心裝置上的本機部署和 Greengrass 元件。

若要使用這些 IPC 作業，請將 Greengrass CLI 元件的 2.6.0 版或更新版本納入為自訂元件中的相依性。然後，您可以在自訂元件中使用 IPC 操作來執行以下操作：

- 建立本機部署以修改和設定核心裝置上的 Greengrass 元件。
- 重新啟動並停止核心裝置上的 Greengrass 元件。
- 產生可用來登入本機除錯主控台的密碼。

主題

最低 SDK 版本

下表列出與 Greengrass CLI IPC 服務互動時必須使用的最低版本。AWS IoT Device SDK

SDK	最低版本	
AWS IoT Device SDK對於爪哇 V2	v1.2.10	
AWS IoT Device SDK對於 Python V2	V1.5.3	
AWS IoT Device SDK對於 C++ 第 2	v1.17.0	

SDK	最低版本	
AWS IoT Device SDK對於 JavaScript V2	v1.12.0	

授權

若要在自訂元件中使用 Greengrass CLI IPC 服務，您必須定義允許元件管理本機部署和元件的授權原則。如需有關定義授權原則的資訊，請參閱授權元件執行 IPC 作業。

對於 Greengrass CLI 的授權策略具有以下屬性。

IPC 服務識別碼：`aws.greengrass.Cli`

操作	描述	資源
`aws.greengrass#CreateLocalDeployment`	允許元件在核心裝置上建立本機部署。	*
`aws.greengrass#ListLocalDeployments`	允許元件列出核心裝置上的本機部署。	*
`aws.greengrass#GetLocalDeploymentStatus`	允許元件取得核心裝置上本機部署的狀態。	本機部署 ID，或允*許存取所有本機部署。
`aws.greengrass#ListComponents`	允許元件列出核心裝置上的元件。	*
`aws.greengrass#GetComponentDetails`	允許元件取得核心裝置上元件的詳細資料。	允許存取所有元件的元件名稱`com.example.HelloWorld` ，例如或*。
`aws.greengrass#RestartComponent`	允許元件重新啟動核心裝置上的元件。	允許存取所有元件的元件名稱`com.example.HelloWorld` ，例如或*。

操作	描述	資源
aws.greengrass#StopComponent	允許元件停止核心裝置上的元件。	允許存取所有元件的元件名稱com.example.HelloWorld ，例如或*。
aws.greengrass#CreateDebugPassword	允許元件產生用來登入本機除錯主控台元件的密碼。	*

Example 授權政策範例

下列範例授權原則可讓元件建立本機部署、檢視所有本機部署和元件，以及重新啟動和停止名為的元件com.example.HelloWorld。

```
{
  "accessControl": {
    "aws.greengrass.Cli": {
      "com.example.MyLocalManagerComponent:cli:1": {
        "policyDescription": "Allows access to create local deployments and view
 deployments and components.",
        "operations": [
          "aws.greengrass#CreateLocalDeployment",
          "aws.greengrass#ListLocalDeployments",
          "aws.greengrass#GetLocalDeploymentStatus",
          "aws.greengrass#ListComponents",
          "aws.greengrass#GetComponentDetails"
        ],
        "resources": [
          "*"
        ]
      }
    },
    "aws.greengrass.Cli": {
      "com.example.MyLocalManagerComponent:cli:2": {
        "policyDescription": "Allows access to restart and stop the Hello World
 component.",
        "operations": [
          "aws.greengrass#RestartComponent",
          "aws.greengrass#StopComponent"
        ],
        "resources": [
```

```
          "com.example.HelloWorld"
        ]
      }
    }
  }
}
```

CreateLocalDeployment

使用指定的元件配方、成品和執行階段引數來建立或更新本機部署。

此作業提供與 Greengrass CLI 中的部署建立命令相同的功能。

請求

此操作的請求具有以下參數：

`recipeDirectoryPath`(Python:`recipe_directory_path`)

(選擇性) 包含元件 recipe 檔案之資料夾的絕對路徑。

`artifactDirectoryPath`(Python:`artifact_directory_path`)

(選擇性) 包含要包含在部署中之人工因素檔案之資料夾的絕對路徑。人工因素資料夾必須包含下列資料夾結構：

```
/path/to/artifact/folder/component-name/component-version/artifacts
```

`rootComponentVersionsToAdd`(Python:`root_component_versions_to_add`)

(選擇性) 要安裝在核心裝置上的元件版本。這個物件是包含下列索引鍵值配對的對映：`ComponentToVersionMap`

`key`

元件的名稱。

`value`

元件的版本。

`rootComponentsToRemove`(Python:`root_components_to_remove`)

(選擇性) 要從核心裝置解除安裝的元件。指定每個項目都是元件名稱的清單。

componentToConfiguration(Python:component_to_configuration)

(選擇性) 部署中每個元件的組態會更新。這個物件是包含下列索引鍵值配對的對映：ComponentToConfiguration

key

元件的名稱。

value

組態會更新元件的 JSON 物件。JSON 物件必須具有下列格式。

```
{
  "MERGE": {
    "config-key": "config-value"
  },
  "RESET": [
    "path/to/reset/"
  ]
}
```

如需有關組態更新的詳細資訊，請參閱更新零組件組態。

componentToRunWithInfo(Python:component_to_run_with_info)

(選擇性) 部署中每個元件的執行階段組態。此組態包括擁有每個元件處理程序的系統使用者，以及套用至每個元件的系統限制。這個物件是包含下列索引鍵值配對的對映：ComponentToRunWithInfo

key

元件的名稱。

value

元件的執行階段組態。如果您省略執行階段組態參數，AWS IoT Greengrass核心軟體會使用您在 Greengr ass 核心上設定的預設值。此物件RunWithInfo包含下列資訊：

posixUser(Python:posix_user)

(選擇性) 要用來在 Linux 核心裝置上執行此元件的 POSIX 系統使用者和 (選擇性) 群組。使用者和群組 (若有指定) 必須存在於每個 Linux 核心裝置上。以下列格式指定使用者和群組，並以冒號 (:) 分隔：user:group。群組為選用項目。如果您未指定群組，AWS IoT

GreengrassCore 軟體會使用使用者的主要群組。如需詳細資訊，請參閱設定執行元件的使用者。

`windowsUser`(Python:`windows_user`)

(選擇性) 用來在 Windows 核心裝置上執行此元件的 Windows 使用者。使用者必須存在於每個 Windows 核心裝置上，且其名稱和密碼必須儲存在 LocalSystem 帳戶的認證管理員執行個體中。如需詳細資訊，請參閱設定執行元件的使用者。

`systemResourceLimits`(Python:`system_resource_limits`)

(選擇性) 套用至此元件處理序的系統資源限制。您可以將系統資源限制套用至一般和非容器化 Lambda 元件。如需詳細資訊，請參閱設定元件的系統資源限制。

AWS IoT Greengrass目前在 Windows 核心裝置上不支援此功能。

此物件`SystemResourceLimits`包含下列資訊：

`cpus`

(選擇性) 此元件的處理序可在核心裝置上使用的 CPU 時間上限。核心裝置的 CPU 總時間等於裝置的 CPU 核心數。例如，在具有 4 個 CPU 核心的核心裝置上，您可以將此值設定為2將此元件的處理序限制為每個 CPU 核心的 50% 使用率。在具有 1 個 CPU 核心的裝置上，您可以將此值設定`0.25`為將此元件的處理序限制為 CPU 使用率 25%。如果您將此值設定為大於 CPU 核心數目的數字，AWS IoT Greengrass核心軟體就不會限制元件的 CPU 使用率。

`memory`

(選擇性) 此元件的處理序可在核心裝置上使用的最大 RAM 容量 (以 KB 為單位)。

`groupName`(Python:`group_name`)

(選擇性) 要使用此部署作為目標的物群組名稱。

回應

此作業的回應包含下列資訊：

`deploymentId`(Python:`deployment_id`)

要求所建立之本機部署的識別碼。

ListLocalDeployments

取得最近 10 個本機部署的狀態。

此作業提供與 Greengrass CLI 中的部署清單命令相同的功能。

請求

此操作的請求沒有任何參數。

回應

此作業的回應包含下列資訊：

`localDeployments`(Python:`local_deployments`)

本機部署的清單。此清單中的每個物件都是一個`LocalDeployment`物件，其中包含下列資訊：

`deploymentId`(Python:`deployment_id`)

本機部署的識別碼。

`status`

本機部署的狀態。此枚舉具有以下值：`DeploymentStatus`

- `QUEUED`
- `IN_PROGRESS`
- `SUCCEEDED`
- `FAILED`

GetLocalDeploymentStatus

取得本機部署的狀態。

此作業提供與 Greengrass CLI 中的部署狀態命令相同的功能。

請求

此操作的請求具有以下參數：

deploymentId(Python:deployment_id)

要取得之本機部署的識別碼。

回應

此作業的回應包含下列資訊：

deployment

本機部署。此物件LocalDeployment包含下列資訊：

deploymentId(Python:deployment_id)

本機部署的識別碼。

status

本機部署的狀態。此枚舉具有以下值：DeploymentStatus

- QUEUED
- IN_PROGRESS
- SUCCEEDED
- FAILED

ListComponents

取得核心裝置上每個根元件的名稱、版本、狀態和組態。根元件是您在部署中指定的元件。此回應不包含安裝為其他元件相依性的元件。

此作業提供與 Greengrass CLI 中的元件清單指令相同的功能。

請求

此操作的請求沒有任何參數。

回應

此作業的回應包含下列資訊：

components

核心裝置上的根元件清單。此清單中的每個物件都是一個ComponentDetails物件，其中包含下列資訊：

componentName(Python:component_name)

元件的名稱。

version

元件的版本。

state

元件的狀態。此狀態可以是下列其中一種：

- BROKEN
- ERRORED
- FINISHED
- INSTALLED
- NEW
- RUNNING
- STARTING
- STOPPING

configuration

該組件的配置為 JSON 對象。

GetComponentDetails

取得核心裝置上元件的版本、狀態和組態。

此作業提供與 Greengrass CLI 中的元件詳細資料指令相同的功能。

請求

此操作的請求具有以下參數：

componentName(Python:component_name)

要取得的元件名稱。

回應

此作業的回應包含下列資訊：

componentDetails(Python:component_details)

組件的詳細信息。此物件ComponentDetails包含下列資訊：

componentName(Python:component_name)

元件的名稱。

version

元件的版本。

state

元件的狀態。此狀態可以是下列其中一種：

- BROKEN
- ERRORED
- FINISHED
- INSTALLED
- NEW
- RUNNING
- STARTING
- STOPPING

configuration

該組件的配置為 JSON 對象。

RestartComponent

重新啟動核心裝置上的元件。

> **Note**
>
> 雖然您可以重新啟動任何元件，但建議您只重新啟動一般元件。

此作業提供與 Greengrass CLI 中的元件重新啟動指令相同的功能。

請求

此操作的請求具有以下參數：

`componentName`(Python:`component_name`)

元件的名稱。

回應

此作業的回應包含下列資訊：

`restartStatus`(Python:`restart_status`)

重新啟動要求的狀態。請求狀態可以是下列其中一種：

- `SUCCEEDED`
- `FAILED`

`message`

關於為什麼元件無法重新啟動 (如果要求失敗) 的訊息。

StopComponent

在核心裝置上停止元件的程序。

> **Note**
>
> 雖然您可以停止任何元件，但建議您僅停止類屬元件。

此操作提供了相同的功能，作為 Greengrass CLI 中的組件停止命令。

請求

此操作的請求具有以下參數：

`componentName`(Python:`component_name`)

元件的名稱。

回應

此作業的回應包含下列資訊：

`stopStatus`(Python:`stop_status`)

停止要求的狀態。請求狀態可以是下列其中一種：

- `SUCCEEDED`
- `FAILED`

`message`

有關為什麼組件無法停止，如果請求失敗的消息。

CreateDebugPassword

產生可用來登入本機除錯主控台元件的隨機密碼。密碼會在產生 8 小時後過期。

此作業提供與 Greengrass CLI 中的get-debug-password 命令相同的功能。

請求

此操作的請求沒有任何參數。

回應

此作業的回應包含下列資訊：

`username`

用於登入的使用者名稱。

`password`

用於登入的密碼。

`passwordExpiration`(Python:`password_expiration`)

密碼到期的時間。

`certificateSHA256Hash`(Python:`certificate_sha256_hash`)

啟用 HTTPS 時，本機偵錯主控台使用的自我簽署憑證的 SHA-256 指紋。當您開啟本機偵錯主控台時，請使用此指紋來驗證憑證是否合法且連線是安全的。

`certificateSHA1Hash`(Python:`certificate_sha1_hash`)

啟用 HTTPS 時，本機偵錯主控台使用的自我簽署憑證的 SHA-1 指紋。當您開啟本機偵錯主控台時，請使用此指紋來驗證憑證是否合法且連線是安全的。

驗證和授權用戶端裝置

> **Note**
>
> 此功能適用於 v2.6.0 及更高版 Greeng rass 核組件。

使用用戶端裝置驗證 IPC 服務來開發自訂本機代理程式元件，讓本機 IoT 裝置 (例如用戶端裝置) 可以連線。

要使用這些 IPC 操作，請將客戶端設備身份驗證組件的版本 2.2.0 或更高版本作為自定義組件的依賴項。然後，您可以在自訂元件中使用 IPC 操作來執行以下操作：

- 驗證連線至核心裝置的用戶端裝置身分。
- 建立用戶端裝置連線至核心裝置的工作階段。
- 確認用戶端裝置是否具有執行處理行動的權限。
- 在核心裝置的伺服器憑證旋轉時收到通知。

主題

- VerifyClientDeviceIdentity
- GetClientDeviceAuthToken
- AuthorizeClientDeviceAction
- SubscribeToCertificateUpdates

最低 SDK 版本

下表列出與用戶端裝置驗證 IPC 服務互動時必須使用的最低版本。AWS IoT Device SDK

SDK	最低版本	
AWS IoT Device SDK對於 Java V2	v1.9.3	
AWS IoT Device SDK對於 Python V2	v1.11.3	
AWS IoT Device SDK對於 C ++ 2	v1.18.3	
AWS IoT Device SDK對於 JavaScript V2	v1.12.0	

授權

若要在自訂元件中使用用戶端裝置驗證 IPC 服務，您必須定義允許元件執行這些作業的授權原則。如需定義授權原則的資訊，請參閱授權元件執行 IPC 作業。

用戶端裝置驗證和授權的授權原則具有下列屬性。

IPC 服務識別碼：`aws.greengrass.clientdevices.Auth`

操作	描述	資源
`aws.greengrass#VerifyClientDeviceIdentity`	允許元件驗證用戶端裝置的身分。	*

操作	描述	資源
aws.greengrass#GetClientDeviceAuthToken	允許元件驗證用戶端裝置的認證，並為該用戶端裝置建立工作階段。	*
aws.greengrass#AuthorizeClientDeviceAction	允許元件驗證用戶端裝置是否有執行動作的權限。	*
aws.greengrass#SubscribeToCertificateUpdates	允許元件在核心裝置的伺服器憑證旋轉時接收通知。	*
*	允許元件執行所有用戶端裝置驗證 IPC 服務作業。	*

授權政策範例

您可以參考下列授權原則範例，協助您設定元件的授權原則。

Example 授權政策範例

下列範例授權原則可讓元件執行所有用戶端裝置驗證 IPC 作業。

```
{
  "accessControl": {
    "aws.greengrass.clientdevices.Auth": {
      "com.example.MyLocalBrokerComponent:clientdevices:1": {
        "policyDescription": "Allows access to authenticate and authorize client
 devices.",
        "operations": [
          "aws.greengrass#VerifyClientDeviceIdentity",
          "aws.greengrass#GetClientDeviceAuthToken",
          "aws.greengrass#AuthorizeClientDeviceAction",
          "aws.greengrass#SubscribeToCertificateUpdates"
        ],
        "resources": [
          "*"
        ]
```

```
      }
    }
  }
}
```

VerifyClientDeviceIdentity

驗證用戶端裝置的身分。此作業會驗證用戶端裝置是否為有效AWS IoT物件。

請求

此操作的請求具有以下參數：

`credential`

用戶端裝置的認證。此物件`ClientDeviceCredential`包含下列資訊：

`clientDeviceCertificate`(Python:`client_device_certificate`)

用戶端裝置的 X.509 裝置憑證。

回應

此作業的回應包含下列資訊：

`isValidClientDevice`(Python:`is_valid_client_device`)

用戶端裝置的身分識別是否有效。

GetClientDeviceAuthToken

驗證用戶端裝置的認證，並為用戶端裝置建立工作階段。此作業會傳回工作階段 Token，您可以在後續要求中使用它來授權用戶端裝置動作。

若要成功連接用戶端裝置，用戶端裝置驗證元件必須授予用戶端裝置所使用之用戶端 ID 的`mqtt:connect`權限。

請求

此操作的請求具有以下參數：

`credential`

用戶端裝置的認證。此物件`CredentialDocument`包含下列資訊：

`mqttCredential`(Python:`mqtt_credential`)

用戶端裝置的 MQTT 認證。指定用戶端裝置用來連線的用戶端 ID 和憑證。此物件`MQTTCredential`包含下列資訊：

`clientId`(Python:`client_id`)

用於連線的用戶端 ID。

`certificatePem`(Python:`certificate_pem`)

用於連線的 X.509 裝置憑證。

`username`

> **Note**
>
> 此屬性目前未使用。

`password`

> **Note**
>
> 此屬性目前未使用。

回應

此作業的回應包含下列資訊：

`clientDeviceAuthToken`(Python:`client_device_auth_token`)

用戶端裝置的工作階段權杖。您可以在後續請求中使用此會話令牌來授權此客戶端設備的操作。

AuthorizeClientDeviceAction

確認用戶端裝置是否具有對資源執行動作的權限。用戶端裝置授權原則會指定用戶端裝置在連線至核心裝置時可執行的權限。您可以在設定用戶端裝置驗證元件時定義用戶端裝置授權原則。

請求

此操作的請求具有以下參數：

`clientDeviceAuthToken`(Python:`client_device_auth_token`)

用戶端裝置的工作階段權杖。

`operation`

要授權的作業。

`resource`

用戶端裝置執行作業的資源。

回應

此作業的回應包含下列資訊：

`isAuthorized`(Python:`is_authorized`)

用戶端裝置是否獲得對資源執行作業的授權。

SubscribeToCertificateUpdates

訂閱以在每次核心裝置循環時接收核心裝置的新伺服器憑證。當伺服器憑證輪換時，代理程式必須使用新的伺服器憑證重新載入。

根據預設，用戶端裝置驗證元件會每 7 天輪換一次伺服器憑證。您可以將輪換間隔設定為 2 到 10 天之間。

此作業是訂閱作業，您可以在其中訂閱事件訊息串流。若要使用此作業，請定義具有處理事件訊息、錯誤和資料流結束之函數的串流回應處理常式。如需詳細資訊，請參閱 訂閱 IPC 事件串流。

事件訊息類型：`CertificateUpdateEvent`

請求

此操作的請求具有以下參數：

certificateOptions(Python:certificate_options)

要訂閱的憑證更新類型。此物件CertificateOptions包含下列資訊：

certificateType(Python:certificate_type)

要訂閱的憑證更新類型。選擇下列選項：

- SERVER

回應

此作業的回應包含下列資訊：

messages

訊息串流。此物件CertificateUpdateEvent包含下列資訊：

certificateUpdate(Python:certificate_update)

有關新憑證的資訊。此物件CertificateUpdate包含下列資訊：

certificate

憑證。

privateKey(Python:private_key)

憑證的私密金鑰。

publicKey(Python:public_key)

憑證的公開金鑰。

caCertificates(Python:ca_certificates)

憑證之 CA 憑證鏈結中的憑證授權單位 (CA) 憑證清單。

與本機 IoT 裝置互動

用戶端裝置是本機 IoT 裝置，可透過 MQTT 連線至 Greengrass 核心裝置並與之通訊。您可以將用戶端裝置連線到核心裝置，以執行下列動作：

- 與綠色組件中的 MQTT 消息進行交互。
- 在用戶端裝置與之間轉送訊息和資料AWS IoT Core。
- 與 Greengrass 組件中的客戶端設備陰影進行交互。
- 同步用戶端裝置的陰影AWS IoT Core。

若要連線至核心裝置，用戶端裝置可以使用雲端探索。用戶端裝置會連線至AWS IoT Greengrass雲端服務，以擷取可連線之核心裝置的相關資訊。然後，他們可以連接到核心設備以處理其消息並將其數據與AWS IoT Core雲服務同步。

您可以按照教學課程逐步介紹如何設定核心裝置以連接物件並與AWS IoT物件通訊。本教學課程也探討如何開發與用戶端裝置互動的自訂 Greengrass 元件。如需更多詳細資訊，請參閱 教學課程：透過 MQTT 與本地 IoT 裝置互動。

主題

AWS-提供的客戶端設備組件

AWS IoT Greengrass提供下列可部署至核心裝置的公用元件。這些元件可讓用戶端裝置連線並與核心裝置進行通訊。

> **Note**
>
> 幾個AWS提供的組件取決於 Greengrass 核的特定次要版本。由於這種依賴關係，您需要在將 Greengrass 核更新為新的次要版本時更新這些組件。如需每個元件所依賴之特定原子核

版本的詳細資訊，請參閱對應的元件主題。如需更新核心的詳細資訊，請參閱更新AWS IoT Greengrass核心軟件（OTA）。

當組件具有泛型和 Lambda 的組件類型時，該組件的當前版本是泛型類型，而先前版本的組件是 Lambda 類型。

元件	描述	元件類型	支援的作業系統	開放原始碼
用戶端裝置驗證	讓本機 IoT 裝置 (稱為用戶端裝置) 連線至核心裝置。	外掛程式	Linux、Windows	是
IP 偵測器	向其報告 MQTT 代理程式連線資訊AWS IoT Greengrass，以便用戶端裝置可以探索如何連線。	外掛程式	Linux、Windows	是
MQTT 大橋	在用戶端裝置、本機AWS IoT Greengrass發佈/訂閱和之間轉送 MQTT 訊息。AWS IoT Core	外掛程式	Linux、Windows	是
MQTT 3.1.1 經紀商 (平均)	執行 MQTT 3.1.1 代理程式，可處理用戶端裝置與核	外掛程式	Linux、Windows	是

元件	描述	元件類型	支援的作業系統	開放原始碼
	心裝置之間的訊息。			
MQTT 5 經紀商	執行 MQTT 5 代理程式，以處理用戶端裝置與核心裝置之間的訊息。	一般	Linux、Windows	否
陰影管理	啟用與核心裝置上的陰影互動。它會管理陰影文件儲存，以及本機陰影狀態與 AWS IoT Device Shadow 服務的同步處理。	外掛程式	Linux、Windows	是

將用戶端裝置連線至核心裝置

您可以將雲端探索設定為將用戶端裝置連線到核心裝置。設定雲端探索時，用戶端裝置可以連線到 AWS IoT Greengrass雲端服務，以擷取可連線的核心裝置相關資訊。然後，用戶端裝置可以嘗試連線到每個核心裝置，直到它們成功連線為止。

若要使用雲端探索，您必須執行下列動作：

- 將用戶端裝置與可連線的核心裝置建立關聯。
- 指定用戶端裝置可連線至每個核心裝置的 MQTT 代理程式端點。
- 將元件部署到可支援用戶端裝置的核心裝置。

  您也可以部署選用元件來執行下列作業：

  - 在AWS IoT Core用戶端裝置、Greengrass 元件和雲端服務之間轉送訊息。

- 自動為您管理核心裝置 MQTT 代理程式端點。
- 管理本機用戶端裝置陰影，並與AWS IoT Core雲端服務同步陰影。

您也必須檢閱並更新核心裝置的AWS IoT原則，以確認其具有連線用戶端裝置所需的權限。如需詳細資訊，請參閱 需求。

設定雲端探索之後，您可以測試用戶端裝置與核心裝置之間的通訊。如需詳細資訊，請參閱 測試用戶端裝置通訊。

主題

需求

若要將用戶端裝置連線到核心裝置，您必須具備下列項目：

- 核心設備必須運 Greengrass v2.2.0 或更高版本。
- 與您在核心裝置運作所在AWS地區相AWS IoT Greengrass關聯AWS 帳戶的 Greengrass 服務角色。如需詳細資訊，請參閱 設定綠色服務角色。
- 核心裝置的AWS IoT策略必須允許以下權限：
  - `greengrass:PutCertificateAuthorities`
  - `greengrass:VerifyClientDeviceIdentity`
  - `greengrass:VerifyClientDeviceIoTCertificateAssociation`

- `greengrass:GetConnectivityInfo`
- `greengrass:UpdateConnectivityInfo`— (選用) IP 偵測器元件需要此權限，IP 偵測器元件會向AWS IoT Greengrass雲端服務報告核心裝置的網路連線資訊。
- `iot:GetThingShadow`、`iot:UpdateThingShadow`、和 `iot:DeleteThingShadow` — (選擇性) 若要使用陰影管理員元件與用戶端裝置陰影同步處理，需要這些權限AWS IoT Core。此功能需要 Greengrass 2.6.0 或更新版本、陰影管理器 v2.2.0 或更新版本，以及 MQTT 橋接器 v2.2.0 或更新版本。

如需詳細資訊，請參閱 設定物AWS IoT件原則。

> **Note**
>
> 如果您在安裝 AWS IoT Greengrass Core 軟體時使用預設AWS IoT原則，核心裝置會有允許存取所有AWS IoT Greengrass動作的AWS IoT原則 (`greengrass:*`)。

- AWS IoT您可以作為客戶端設備連接的東西。如需詳細資訊，請參閱AWS IoT Core開發人員指南中的建立AWS IoT資源。
- 用戶端裝置必須使用用戶端 ID 進行連線。用戶端 ID 是物件名稱。不接受其他用戶端 ID。
- 每個用戶端裝置的AWS IoT策略都必須允許`greengrass:Discover`權限。如需詳細資訊，請參閱用戶端裝置的最低AWS IoT原則。

主題

- 設定綠色服務角色
- 設定物AWS IoT件原則

設定綠色服務角色

Greengrass 服務角色是一種 AWS Identity and Access Management (IAM) 服務角色，可授權代表 AWS IoT Greengrass您從服AWS務存取資源。此角色可讓AWS IoT Greengrass您驗證用戶端裝置的身分識別，並管理核心裝置連線資訊。

如果您先前尚未在此區域中設定 Greengrass 服務角色，則必須將 Greengrass 服務角色與AWS IoT Greengrass您在此區域中的服務角色建立關聯。AWS 帳戶

使用AWS IoT Greengrass主控台中的 [設定核心裝置探索] 頁面時，請為您AWS IoT Greengrass設定 Greengrass 服務角色。否則，您可以使用AWS IoT控制台或 AWS IoT Greengrass API 手動設置它。

在本節中，您會檢查 Greengrass 服務角色是否已設定。如果未設定，您可以建立新的 Greengrass 服務角色，以便與AWS IoT Greengrass此區AWS 帳戶域中的您建立關聯。

設定綠色服務角色 (主控台)

1. 檢查 Greengrass 服務角色是否與您在此AWS 帳戶區域中AWS IoT Greengrass的相關聯。請執行下列操作：

   a. 導覽至 AWS IoT主控台。

   b. 在導覽窗格中，選擇設定。

   c. 在 Greengrass 服務角色區段中，尋找目前的服務角色，以查看 Greengrass 服務角色是否已關聯。

      如果您有相關聯的 Greengrass 服務角色，則您符合使用 IP 偵測器元件的此需求。跳至 設定物AWS IoT件原則。

2. 如果 Greengrass 服務角色與您AWS 帳戶在此區域中沒有AWS IoT Greengrass關聯，請建立 Greengrass 服務角色並將其關聯。請執行下列操作：

   a. 導覽至 IAM 主控台。

   b. 選擇角色。

   c. 選擇建立角色。

   d. 在 [建立角色] 頁面上，執行下列動作：

      i. 在 [信任的實體類型] 下，選擇AWS 服務。

      ii. 在使用案例，其他用例下AWS 服務，選擇 Greengrass，選擇 Greengrass。此選項會指定新增AWS IoT Greengrass為可擔任此角色的受信任實體。

      iii. 選擇下一步。

      iv. 在 [權限原則] 下，選取AWSGreengrassResourceAccessRolePolicy要附加至角色的。

      v. 選擇下一步。

      vi. 在角色名稱中，輸入角色的名稱，例如**`Greengrass_ServiceRole`**。

      vii. 選擇建立角色。

   e. 導覽至 AWS IoT主控台。

   f. 在導覽窗格中，選擇設定。

   g. 在 [Greengrass 服務角色] 區段中，選擇 [附加角色]。

   h. 在 [更新 Greengrass 服務角色模式] 中，選取您建立的 IAM 角色，然後選擇 [連接角色]。

設定綠色服務角色 () AWS CLI

1. 檢查 Greengrass 服務角色是否與您在此AWS 帳戶區域中AWS IoT Greengrass的相關聯。

   ```
   aws greengrassv2 get-service-role-for-account
   ```

   如果 Greengrass 服務角色相關聯，則作業會傳回包含角色相關資訊的回應。

   如果您有相關聯的 Greengrass 服務角色，則您符合使用 IP 偵測器元件的此需求。跳至 設定物AWS IoT件原則。

2. 如果 Greengrass 服務角色與您AWS 帳戶在此區域中沒有AWS IoT Greengrass關聯，請建立 Greengrass 服務角色並將其關聯。請執行下列操作：

   a. 使用允許 AWS IoT Greengrass 擔任角色之信任政策的角色。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原則中加入`aws:SourceArn`和`aws:SourceAccount`全域條件內容金鑰，以協助避免混淆的副安全性問題。條件內容索引鍵會限制存取權，只允許來自指定帳戶和 Greengrass 工作區的要求。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 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": {
              "ArnLike": {
                "aws:SourceArn": "arn:aws:greengrass:region:account-id:*"
              },
              "StringEquals": {
                "aws:SourceAccount": "account-id"
              }
            }
          }
        ]
      ```

```
}'
```

Windows Command Prompt (CMD)

```
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\\"}}}]}"
```

PowerShell

```
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"
        }
      }
    }
  ]
}'
```

b. 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。

c. 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
```

d. 將 Greengrass 服務角色與AWS IoT Greengrass您的. AWS 帳戶 將*## arn ###*服務角色的 ARN。

```
aws greengrassv2 associate-service-role-to-account --role-arn role-arn
```

如果成功，作業會傳回下列回應。

```
{
  "associatedAt": "timestamp"
}
```

設定物AWS IoT件原則

核心裝置使用 X.509 裝置憑證來授權連線。AWS您可以將AWS IoT原則附加至裝置憑證，以定義核心裝置的權限。如需詳細資訊，請參閱 資料平面操作的AWS IoT 政策 及 支援用戶端裝置的最低AWS IoT原則。

若要將用戶端裝置連線到核心裝置，核心裝置的AWS IoT策略必須允許下列權限：

- `greengrass:PutCertificateAuthorities`
- `greengrass:VerifyClientDeviceIdentity`
- `greengrass:VerifyClientDeviceIoTCertificateAssociation`
- `greengrass:GetConnectivityInfo`
- `greengrass:UpdateConnectivityInfo`— (選用) IP 偵測器元件需要此權限，IP 偵測器元件會向AWS IoT Greengrass雲端服務報告核心裝置的網路連線資訊。
- `iot:GetThingShadow`、`iot:UpdateThingShadow`、和 `iot:DeleteThingShadow` — (選擇性) 若要使用陰影管理員元件與用戶端裝置陰影同步處理，需要這些權限AWS IoT Core。此功能需要 Greengrass 2.6.0 或更新版本、陰影管理器 v2.2.0 或更新版本，以及 MQTT 橋接器 v2.2.0 或更新版本。

在本節中，您可以檢閱核心裝置的AWS IoT原則，並新增任何遺失的必要權限。如果您使用 AWS IoT GreengrassCore 軟體安裝程式佈建資源，您的核心裝置會有允許存取所有AWS IoT Greengrass動作的AWS IoT原則 (`greengrass:*`)。在此情況下，只有當您計劃部署陰影管理員元件以同步處理裝置陰影時，才必須更新AWS IoT原則AWS IoT Core。否則，您可以跳過此部分。

設定物AWS IoT件原則 (主控台)

1. 在AWS IoT Greengrass主控台瀏覽功能表中，選擇 [核心裝置]。
2. 在 [核心裝置] 頁面上，選擇要更新的核心裝置。
3. 在核心裝置詳細資料頁面上，選擇核心裝置物件的連結。此連結會在AWS IoT主控台中開啟物件詳細資訊頁面。
4. 在物件詳細資訊頁面上，選擇 [憑證]。
5. 在「憑證」索引標籤中，選擇物件的使用中憑證。
6. 在憑證詳細資料頁面上，選擇 [原則]。
7. 在「策略」索引標籤中，選擇要檢閱和更新的AWS IoT策略。您可以將必要的權限新增至任何附加至核心裝置作用中憑證的原則。

   > **Note**
   >
   > 如果您使用 AWS IoT GreengrassCore 軟體安裝程式佈建資源，則有兩個AWS IoT原則。建議您選擇名為GreengrassV2IoTThingPolicy的策略 (如果存在的話)。依預設，您使用快速安裝程式建立的核心裝置會使用此原則名稱。如果您將權限新增至此原則，也會將這些權限授與使用此原則的其他核心裝置。

8. 在策略概觀中，選擇編輯作用中的版本。
9. 檢閱所需權限的原則，並新增任何遺失的必要權限。

   - `greengrass:PutCertificateAuthorities`
   - `greengrass:VerifyClientDeviceIdentity`
   - `greengrass:VerifyClientDeviceIoTCertificateAssociation`
   - `greengrass:GetConnectivityInfo`
   - `greengrass:UpdateConnectivityInfo`— (選用) IP 偵測器元件需要此權限，IP 偵測器元件會向AWS IoT Greengrass雲端服務報告核心裝置的網路連線資訊。
   - `iot:GetThingShadow`、`iot:UpdateThingShadow`、和 `iot:DeleteThingShadow` — (選擇性) 若要使用陰影管理員元件與用戶端裝置陰影同步處理，需要這些權限AWS IoT Core。此功能需要 Greengrass 2.6.0 或更新版本、陰影管理器 v2.2.0 或更新版本，以及 MQTT 橋接器 v2.2.0 或更新版本。

10. (選擇性) 若要允許核心裝置與陰影同步處理AWS IoT Core，請將下列陳述式新增至原則。如果您打算與用戶端裝置陰影互動，但不與其同步AWS IoT Core，請略過此步驟。將*##*和*## ID* 替換為您使用的地區和您的AWS 帳戶號碼。

- 此範例陳述式允許存取所有物件的裝置陰影。若要遵循最佳安全性做法，您可以限制只有核心裝置和連線至核心裝置的用戶端裝置的存取權。如需詳細資訊，請參閱 支援用戶端裝置的最低 AWS IoT原則。

```
{
  "Effect": "Allow",
  "Action": [
    "iot:GetThingShadow",
    "iot:UpdateThingShadow",
    "iot:DeleteThingShadow"
  ],
  "Resource": [
    "arn:aws:iot:region:account-id:thing/*"
  ]
}
```

在您新增這個陳述式之後，原則文件看起來可能會類似下列範例。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Connect",
        "iot:Publish",
        "iot:Subscribe",
        "iot:Receive",
        "greengrass:*"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:GetThingShadow",
        "iot:UpdateThingShadow",
        "iot:DeleteThingShadow"
      ],
      "Resource": [
        "arn:aws:iot:region:account-id:thing/*"
```

```
      ]
    }
  ]
}
```

11. 若要將新的原則版本設定為作用中版本，請在 [原則版本狀態] 下選取 [將編輯的版本設定為此原則的作用中版本]。
12. 選擇「另存為新版本」。

設定AWS IoT物件原則 (AWS CLI)

1. 列出核心裝置物AWS IoT件的主體。物件主參與者可以是 X.509 裝置憑證或其他識別。執行下列命令，並*MyGreengrassCore*以核心裝置的名稱取代。

```
aws iot list-thing-principals --thing-name MyGreengrassCore
```

作業會傳回列出核心裝置物件主參與者的回應。

```
{
    "principals": [
        "arn:aws:iot:us-west-2:123456789012:cert/certificateId"
    ]
}
```

2. 識別核心裝置的作用中憑證。執行下列命令，並將 certific *ateId* 取代為上一個步驟中每個憑證的 ID，直到找到作用中的憑證為止。憑證 ID 是位於憑證 ARN 結尾的十六進位字串。引--query數指定僅輸出憑證的狀態。

```
aws iot describe-certificate --certificate-id certificateId --query
 'certificateDescription.status'
```

作業會以字串形式傳回憑證狀態。例如，如果憑證處於作用中狀態，則此作業會輸出"ACTIVE"。

3. 列出附加至憑證的AWS IoT原則。執行下列命令，並以憑證的 ARN 取代憑證 ARN。

```
aws iot list-principal-policies --principal arn:aws:iot:us-
west-2:123456789012:cert/certificateId
```

作業會傳回回應，列出附加至憑證的AWS IoT原則。

```
{
    "policies": [
        {
            "policyName":
 "GreengrassTESCertificatePolicyMyGreengrassCoreTokenExchangeRoleAlias",
            "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassTESCertificatePolicyMyGreengrassCoreTokenExchangeRoleAlias"
        },
        {
            "policyName": "GreengrassV2IoTThingPolicy",
            "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy"
        }
    ]
}
```

4. 選擇要檢視和更新的原則。

> **Note**
>
> 如果您使用 AWS IoT GreengrassCore 軟體安裝程式佈建資源，則有兩個AWS IoT原則。建議您選擇名為GreengrassV2IoTThingPolicy的策略 (如果存在的話)。依預設，您使用快速安裝程式建立的核心裝置會使用此原則名稱。如果您將權限新增至此原則，也會將這些權限授與使用此原則的其他核心裝置。

5. 取得政策的文件。執行下列命令，並以原則的名稱取代 *GreenGrassv2IoT ThingPolicy*。

```
aws iot get-policy --policy-name GreengrassV2IoTThingPolicy
```

作業會傳回回應，其中包含原則的文件和其他有關原則的資訊。政策文件是序列化為字串的 JSON 物件。

```
{
    "policyName": "GreengrassV2IoTThingPolicy",
    "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy",
    "policyDocument": "{\
  \\"Version\\": \\"2012-10-17\\",\
  \\"Statement\\": [\
    {\
```

```
      \\"Effect\\": \\"Allow\\",\
      \\"Action\\": [\
                \\"iot:Connect\\",\
                \\"iot:Publish\\",\
                \\"iot:Subscribe\\",\
                \\"iot:Receive\\",\
                \\"greengrass:*\\"\
],\
      \\"Resource\\": \\"*\\"\
    }\
  ]\
}",
    "defaultVersionId": "1",
    "creationDate": "2021-02-05T16:03:14.098000-08:00",
    "lastModifiedDate": "2021-02-05T16:03:14.098000-08:00",
    "generationId":
 "f19144b798534f52c619d44f771a354f1b957dfa2b850625d9f1d0fde530e75f"
}
```

6. 使用線上轉換器或其他工具將原則文件字串轉換為 JSON 物件，然後將其儲存至名為的檔案`iot-policy.json`。

   例如，如果您已安裝 jq 工具，您可以執行下列命令來取得原則文件、將其轉換為 JSON 物件，並將原則文件儲存為 JSON 物件。

   ```
   aws iot get-policy --policy-name GreengrassV2IoTThingPolicy --query
    'policyDocument' | jq fromjson >> iot-policy.json
   ```

7. 檢閱所需權限的原則，並新增任何遺失的必要權限。

   例如，在 Linux 系統上，您可以執行下列命令來使用 GNU nano 來開啟檔案。

   ```
   nano iot-policy.json
   ```

   - `greengrass:PutCertificateAuthorities`
   - `greengrass:VerifyClientDeviceIdentity`
   - `greengrass:VerifyClientDeviceIoTCertificateAssociation`
   - `greengrass:GetConnectivityInfo`
   - `greengrass:UpdateConnectivityInfo`— (選用) IP 偵測器元件需要此權限，IP 偵測器元件會向AWS IoT Greengrass雲端服務報告核心裝置的網路連線資訊。

- `iot:GetThingShadow`、`iot:UpdateThingShadow`、和 `iot:DeleteThingShadow` — (選擇性) 若要使用陰影管理員元件與用戶端裝置陰影同步處理，需要這些權限AWS IoT Core。此功能需要 Greengrass 2.6.0 或更新版本、陰影管理器 v2.2.0 或更新版本，以及 MQTT 橋接器 v2.2.0 或更新版本。

8. 將變更儲存為策略的新版本。執行下列命令，並以原則的名稱取代 *GreenGrassv2IoTThingPolicy*。

```
aws iot create-policy-version --policy-name GreengrassV2IoTThingPolicy --policy-
document file://iot-policy.json --set-as-default
```

如果成功，作業會傳回類似下列範例的回應。

```
{
    "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy",
    "policyDocument": "{\
  \\"Version\\": \\"2012-10-17\\",\
  \\"Statement\\": [\
    {\
      \\"Effect\\": \\"Allow\\",\
      \\"Action\\": [\
\\t\\t\\"iot:Connect\\",\
\\t\\t\\"iot:Publish\\",\
\\t\\t\\"iot:Subscribe\\",\
\\t\\t\\"iot:Receive\\",\
\\t\\t\\"greengrass:*\\"\
      ],\
      \\"Resource\\": \\"*\\"\
    }\
  ]\
}",
    "policyVersionId": "2",
    "isDefaultVersion": true
}
```

用於客戶端設備支持的 Greengrass 組件

> **Important**
>
> 核心裝置必須執行 Greengrass 核心 v2.2.0 或更新版本，才能支援用戶端裝置。

若要讓用戶端裝置能夠連線並與核心裝置通訊，請將下列 Greengrass 元件部署到核心裝置：

- 用戶端裝置驗證 (`aws.greengrass.clientdevices.Auth`)

  部署用戶端裝置驗證元件以驗證用戶端裝置並授權用戶端裝置動作。這個組件允許你的AWS IoT東西連接到一個核心設備。

  此組件需要一些配置才能使用它。您必須指定用戶端裝置群組，以及每個群組獲授權執行的作業，例如透過 MQTT 進行連線和通訊。如需詳細資訊，請參閱用戶端裝置驗證元件組態。

- MQTT 3.1.1 經紀商 (平均) (`aws.greengrass.clientdevices.mqtt.Moquette`)

  部署 Moquette MQTT 代理程式元件以執行輕量型 MQTT 代理程式。Moquette MQTT 代理程式符合 MQTT 3.1.1 規範，並包含對 QoS 0、QoS 1、QoS 2、保留訊息、最後將訊息和持續訂閱的本機支援。

  您不需要配置此組件即可使用它。不過，您可以設定此元件操作 MQTT 代理程式的連接埠。依預設，它會使用連接埠 8883。

- MQTT 5 經紀商 (`aws.greengrass.clientdevices.mqtt.EMQX`)

  > **Note**
  >
  > 若要使用 EMQX MQTT 5 代理程式，您必須使用 Greengrass 核心 v2.6.0 或更新版本，以及用戶端裝置驗證 v2.2.0 或更新版本。

  部署 EMQX MQTT 代理程式元件，以便在用戶端裝置與核心裝置之間的通訊中使用 MQTT 5.0 功能。EMQX MQTT 代理程式符合 MQTT 5.0 規範，並包含工作階段和訊息到期間隔、使用者屬性、共用訂閱、主題別名等支援。

  您不需要配置此組件即可使用它。不過，您可以設定此元件操作 MQTT 代理程式的連接埠。依預設，它會使用連接埠 8883。

- MQTT 大橋 (aws.greengrass.clientdevices.mqtt.Bridge)

  (選擇性) 部署 MQTT 橋接器元件，以在用戶端裝置 (本機 MQTT)、本機發佈/訂閱和 MQTT 之間轉送訊息。AWS IoT Core設定此元件，以便與 Greengrass 元件的用戶端裝置同步處理用戶端裝置，AWS IoT Core並與其互動。

  此元件需要組態才能使用。您必須指定此元件轉送訊息的主題對映。如需詳細資訊，請參閱 MQTT 橋接器元件組態。

- IP 偵測器 (aws.greengrass.clientdevices.IPDetector)

  (選擇性) 部署 IP 偵測器元件，以自動向AWS IoT Greengrass雲端服務報告核心裝置的 MQTT 代理程式端點。如果您有複雜的網路設定，例如路由器將 MQTT 代理程式連接埠轉送至核心裝置的網路設定，則無法使用此元件。

  您不需要配置此組件即可使用它。

- 陰影管理 (aws.greengrass.ShadowManager)

  > **Note**
  >
  > 若要管理用戶端裝置陰影，您必須使用 Greengrass 核 v2.6.0 或更新版本、陰影管理員 v2.2.0 或更新版本，以及 MQTT 橋接器 v2.2.0 或更新版本。

  (選擇性) 部署陰影管理員元件，以管理核心裝置上的用戶端裝置陰影。Greengrass 組件可以獲取，更新和刪除客戶端設備陰影以與客戶端設備進行交互。您也可以設定陰影管理員元件，以同步處理用戶端裝置陰影與AWS IoT Core雲端服務。

  若要將此元件與用戶端裝置陰影搭配使用，您必須設定 MQTT 橋接器元件，以便在用戶端裝置和陰影管理員 (使用本機發佈/訂閱) 之間轉送訊息。否則，此組件不需要配置即可使用，但確實需要配置才能同步設備陰影。

> **Note**
>
> 我們建議您只部署一個 MQTT 代理程式元件。MQTT 橋接器和 IP 偵測器元件一次只能搭配一個 MQTT 代理程式元件使用。如果您部署多個 MQTT 代理程式元件，則必須將它們設定為使用不同的連接埠。

設定雲端探索 (主控台)

您可以使用AWS IoT Greengrass主控台來關聯用戶端裝置、管理核心裝置端點，以及部署元件以啟用用戶端裝置支援。如需詳細資訊，請參閱 步驟 2：啟用用戶端裝置支援。

設定雲端探索 (AWS CLI)

您可以使用 AWS Command Line Interface (AWS CLI) 來關聯用戶端裝置、管理核心裝置端點，以及部署元件以啟用用戶端裝置支援。如需詳細資訊，請參閱下列內容：

- 管理用戶端裝置關聯 (AWS CLI)
- 管理核心裝置端點
- AWS-提供的客戶端設備組件
- 建立部署

關聯用戶端裝置

若要使用雲端探索，請將用戶端裝置與核心裝置建立關聯，以便他們可以探索核心裝置。然後，他們可以使用 Greengrass 探索 API 擷取關聯核心裝置的連線資訊和憑證。

同樣地，請取消用戶端裝置與核心裝置的關聯，以防止它們探索核心裝置。

主題

- 管理用戶端裝置關聯 (主控台)
- 管理用戶端裝置關聯 (AWS CLI)
- 管理用戶端裝置關聯 (API)

管理用戶端裝置關聯 (主控台)

您可以使用主AWS IoT Greengrass控台來檢視、新增和刪除用戶端裝置關聯。

檢視核心裝置 (主控台) 的用戶端裝置關聯

1. 導覽至 AWS IoT Greengrass主控台。
2. 選擇核心裝置。
3. 選擇要管理的核心裝置。
4. 在核心裝置的詳細資料頁面上，選擇 [用戶端裝置] 索引標籤。

5. 在 [關聯的用戶端裝置] 區段中，您可以查看與核心裝置相關聯的用戶端裝置 (AWS IoT物件)。

建立用戶端裝置與核心裝置 (主控台) 的關聯

1. 導覽至 AWS IoT Greengrass主控台。
2. 選擇核心裝置。
3. 選擇要管理的核心裝置。
4. 在核心裝置的詳細資料頁面上，選擇 [用戶端裝置] 索引標籤。
5. 在關聯的用戶端裝置區段中，選擇關聯用戶端裝置。
6. 在「將用戶端裝置與核心裝置建立關聯」模式中，針對要關聯的每個用戶端裝置執行下列動作

   a. 輸入AWS IoT要關聯為用戶端裝置的物件名稱。

   b. 選擇新增。
7. 選擇 Associate (關聯)。

   您關聯的用戶端裝置現在可以使用 Greengrass 探索 API 來探索此核心裝置。

取消用戶端裝置與核心裝置 (主控台) 的關聯

1. 導覽至 AWS IoT Greengrass主控台。
2. 選擇核心裝置。
3. 選擇要管理的核心裝置。
4. 在核心裝置的詳細資料頁面上，選擇 [用戶端裝置] 索引標籤。
5. 在「關聯的用戶端裝置」區段中，選取要取消關聯的每個用戶端裝置。
6. 選擇 Disassociate (取消關聯)。
7. 在確認模式中，選擇「取消關聯」。

   您取消關聯的用戶端裝置無法再使用 Greengrass 探索 API 來探索此核心裝置。

管理用戶端裝置關聯 (AWS CLI)

您可以使用 AWS Command Line Interface (AWS CLI) 來管理核心裝置的用戶端裝置關聯。

檢視核心裝置的用戶端裝置關聯 (AWS CLI)

- 使用以下命令：list-client-devices-associated-with-core-device。

建立用戶端裝置與核心裝置的關聯 (AWS CLI)

- 使用以下命令：batch-associate-client-device-with-core-device。

取消用戶端裝置與核心裝置的關聯 () AWS CLI

- 使用以下命令：batch-disassociate-client-device-from-core-device。

管理用戶端裝置關聯 (API)

您可以使用 AWS API 來管理核心裝置的用戶端裝置關聯。

檢視核心裝置 (AWSAPI) 的用戶端裝置關聯

- 使用以下操作：ListClientDevicesAssociatedWithCoreDevice。

建立用戶端裝置與核心裝置 (AWSAPI) 的關聯

- 使用以下操作：BatchAssociateClientDeviceWithCoreDevice。

取消用戶端裝置與核心裝置 (AWSAPI) 的關聯

- 使用以下操作：BatchDisassociateClientDeviceFromCoreDevice。

離線時驗證用戶端

使用離線驗證，您可以配置 AWS IoT Greengrass Core 設備，以便客戶端設備可以連接到核心設備，即使核心設備未連接到雲端也是如此。當您使用離線驗證時，您的 Greengrass 裝置可以在部分離線的環境中繼續運作。

若要針對已連線至雲端的用戶端裝置使用離線驗證，您需要下列項目：

- 已部署用戶端裝置驗證元件的 AWS IoT Greengrass 核心裝置。您必須使用 2.3.0 或更高版本進行離線驗證。
- 用戶端裝置初始連線期間核心裝置的雲端連線。

儲存用戶端認證

當用戶端裝置首次連線至核心裝置時，核心裝置會呼叫 AWS IoT Greengrass 服務。在呼叫時，Greengrass 會驗證用戶端裝置的註冊為物件。 AWS IoT 它也會驗證裝置是否具有有效的憑證。核心裝置接著會將此資訊儲存在本機。

下次裝置連線時，Greengrass 核心裝置會嘗試使用服務驗證用戶端裝置。 AWS IoT Greengrass 如果無法連線到 AWS IoT Greengrass，核心裝置會使用其本機儲存的裝置資訊來驗證用戶端裝置。

您可以設定 Greengrass 核心裝置儲存認證的時間長度。[您可以在用戶端裝置驗證元件組態中設定`clientDeviceTrustDurationMinutes`組態選項，將逾時時間從一分鐘設定為 2,147,483,647 分鐘。](#) 預設值為一分鐘，可有效關閉離線驗證。當您設定此逾時時間時，我們建議您考慮安全性需求。您還應該考慮在與雲中斷連接時，核心設備可以運行多長時間。

核心裝置會更新其憑證儲存裝置三次：

1. 當設備首次連接到核心設備時。
2. 如果核心裝置已連線至雲端，則當用戶端裝置重新連線至核心裝置時。
3. 如果核心裝置已連線至雲端，則每天一次重新整理整個憑證存放區。

當 Greengrass 核心裝置重新整理其認證存放區時，它會使用該作業。[ListClientDevicesAssociatedWithCoreDevice](#)Greengrass 只會重新整理此作業傳回的裝置。若要將用戶端裝置與核心裝置建立關聯，請參閱[關聯用戶端裝置](#)。

若要使用該`ListClientDevicesAssociatedWithCoreDevice`作業，您必須將作業的權限新增至與執行的相關聯 AWS 帳戶 的 AWS Identity and Access Management (IAM) 角色 AWS IoT Greengrass。如需更多詳細資訊，請參閱 [授權核心設備與AWS服務](#)。

管理核心裝置端點

當您使用雲端探索時，您會將核心裝置的 MQTT 代理程式端點儲存在AWS IoT Greengrass雲端服務中。用戶端裝置會連線AWS IoT Greengrass以擷取這些端點和其關聯核心裝置的其他資訊。

對於每個核心裝置，您可以自動或手動管理端點。

- 使用 IP 偵測器自動管理端點

  如果您的網路設定非複雜，例如用戶端裝置與核心裝置位於相同網路上的位置，則可以部署 [IP 偵測器元件](#)以自動為您管理核心裝置端點。例如，如果核心設備位於將 MQTT 代理端口轉發到核心設備的路由器後面，則無法使用 IP 檢測器組件。

如果您部署到物件群組，IP 偵測器元件也很有用，因為它會管理物件群組中所有核心裝置的端點。如需詳細資訊，請參閱 使用 IP 偵測器自動管理端點。

- 手動管理端點

  如果您無法使用 IP 偵測器元件，則必須手動管理核心裝置端點。您可以使用主控台或 API 更新這些端點。如需詳細資訊，請參閱 手動管理端點。

主題

使用 IP 偵測器自動管理端點

如果您有簡單的網路設定 (例如與核心裝置位於相同網路上的用戶端裝置)，則可以部署 IP 偵測器元件以執行下列作業：

- 監控 Greengrass 核心裝置的區域網路連線資訊。此資訊包括核心裝置的網路端點，以及 MQTT 代理程式運作所在的連接埠。
- 向AWS IoT Greengrass雲服務報告核心設備的連接信息。

IP 偵測器元件會覆寫您手動設定的端點。

> **Important**
>
> 核心裝置的AWS IoT原則必須允`greengrass:UpdateConnectivityInfo`許使用 IP 偵測器元件的權限。如需詳細資訊，請參閱 資料平面操作的AWS IoT 政策 及 設定物AWS IoT件原則。

您可以執行下列其中一項作業來部署 IP 偵測器元件：

- 使用主控台中的 [設定探查] 頁面。如需詳細資訊，請參閱 設定雲端探索 (主控台)。
- 建立和修訂部署以包含 IP 偵測器。您可以使用主控AWS CLI台或 AWS API 來管理部署。如需詳細資訊，請參閱 建立部署。

部署 IP 偵測器元件 (主控台)

1. 在AWS IoT Greengrass主控台瀏覽功能表中，選擇 [元件]。
2. 在 [元件] 頁面上，選擇 [公用元件] 索引標籤，然後選擇aws.greengrass.clientdevices.IPDetector。
3. 在 aws.greengrass.clientdevices.IPDetector頁面中，選擇部署。
4. 從 [新增至部署] 中，選擇要修訂的現有部署，或選擇建立新部署，然後選擇 [下一步]。
5. 如果您選擇建立新部署，請為部署選擇目標核心裝置或物件群組。在 [指定目標] 頁面的 [部署目標] 下，選擇核心裝置或物件群組，然後選擇 [下一步]。
6. 在 [選取元件] 頁面上，確認已選取aws.greengrass.clientdevices.IPDetector元件，然後選擇 [下一步]。
7. 在 [設定元件] 頁面上，選取 aws.greengrass.clientdevices.IPDetector，然後執行下列動作：

   a. 選擇 設定元件。

   b. 在 [設定aws.greengrass.clientdevices.IPDetector強制回應] 的 [組態更新] 下的 [要合併的組態] 中，您可以輸入組態更新以設定 IP 偵測器元件。您可以指定下列任何一個組態選項：

      - `defaultPort`— (選用) 此元件偵測到 IP 位址時要報告的 MQTT 代理程式連接埠。如果您將 MQTT 代理程式設定為使用與預設連接埠 8883 不同的連接埠，則必須指定此參數。
      - `includeIPv4LoopbackAddrs`— (選用) 您可以啟用此選項來偵測和報告 IPv4 回送位址。這些是 IP 位址，例如`localhost`裝置可以與自身通訊的位置。在核心裝置和用戶端裝置在相同系統上執行的測試環境中使用此選項。
      - `includeIPv4LinkLocalAddrs`— (選用) 您可以啟用此選項來偵測和報告 IPv4 連結本機位址。如果核心裝置的網路沒有動態主機設定通訊協定 (DHCP) 或靜態指派的 IP 位址，請使用此選項。

      組態更新看起來可能類似下列範例。

      ```
      {
        "defaultPort": "8883",
        "includeIPv4LoopbackAddrs": false,
        "includeIPv4LinkLocalAddrs": false
      }
      ```

   c. 選擇 [確認] 關閉強制回應，然後選擇 [下一步]。

8. 在設定進階設定頁面上，保留預設組態設定，然後選擇 下一步。

9. 在 Review (檢閱) 頁面，選擇 Deploy (部署)。

   部署最多可能需要一分鐘的時間才能完成。

部署 IP 偵測器元件 (AWS CLI)

若要部署 IP 偵測器元件，請建立包含在`components`物件`aws.greengrass.clientdevices.IPDetector`中的部署文件，並指定元件的組態更新。遵循中建立部署的指示建立新部署或修訂既有部署。

建立部署文件時，您可以指定下列任一選項來設定 IP 偵測器元件：

- `defaultPort`— (選用) 此元件偵測到 IP 位址時要報告的 MQTT 代理程式連接埠。如果您將 MQTT 代理程式設定為使用與預設連接埠 8883 不同的連接埠，則必須指定此參數。
- `includeIPv4LoopbackAddrs`— (選用) 您可以啟用此選項來偵測和報告 IPv4 回送位址。這些是 IP 位址，例如`localhost`裝置可以與自身通訊的位置。在核心裝置和用戶端裝置在相同系統上執行的測試環境中使用此選項。
- `includeIPv4LinkLocalAddrs`— (選用) 您可以啟用此選項來偵測和報告 IPv4 連結本機位址。如果核心裝置的網路沒有動態主機設定通訊協定 (DHCP) 或靜態指派的 IP 位址，請使用此選項。

下列範例部分部署文件指定將連接埠 8883 報告為 MQTT 代理人連接埠。

```
{
  ...,
  "components": {
    ...,
    "aws.greengrass.clientdevices.IPDetector": {
      "componentVersion": "2.1.1",
      "configurationUpdate": {
        "merge": "{\"defaultPort\":\"8883\",}"
      }
    }
  }
}
```

手動管理端點

您可以手動管理核心裝置的 MQTT 代理程式端點。

每個 MQTT 代理程式端點都有下列資訊：

端點 (HostAddress)

用戶端裝置可連線至核心裝置上的 MQTT 代理程式的 IP 位址或 DNS 位址。

連接埠 (PortNumber)

MQTT 代理程式在核心裝置上運作的連接埠。

您可以在預設使用連接埠 8883 的 Moquette MQTT 代理程式元件上設定此連接埠。

中繼資料 (Metadata)

提供給連線至此端點的用戶端裝置的其他中繼資料。

主題

- 管理端點 (主控台)
- 管理端點 (AWS CLI)
- 管理端點 (API)

管理端點 (主控台)

您可以使用AWS IoT Greengrass主控台來檢視、更新和移除核心裝置的端點。

管理核心裝置 (主控台) 的端點

1. 導覽至 AWS IoT Greengrass主控台。
2. 選擇核心裝置。
3. 選擇要管理的核心裝置。
4. 在核心裝置的詳細資料頁面上，選擇 [用戶端裝置] 索引標籤。
5. 在 MQTT 代理程式端點區段中，您可以查看核心裝置的 MQTT 代理程式端點。選擇「管理端點」。
6. 在「管理端點」模式中，新增或移除核心裝置的 MQTT 代理程式端點。
7. 選擇更新。

管理端點 (AWS CLI)

您可以使用 AWS Command Line Interface (AWS CLI) 管理核心裝置的端點。

> **Note**
>
> 由於中的用戶端裝置支援與回溯相容AWS IoT Greengrass V1，因AWS IoT Greengrass V2此您可以使用AWS IoT Greengrass V2或 AWS IoT Greengrass V1 API 作業來管理核心裝置端點。

取得核心裝置的端點 (AWS CLI)

- 請使用下列其中一個指令：
  - 格雷格拉斯 2： get-connectivity-info
  - 格雷格拉斯： get-connectivity-info

更新核心裝置的端點 (AWS CLI)

- 請使用下列其中一個指令：
  - 格雷格拉斯 2： update-connectivity-info
  - 格雷格拉斯： update-connectivity-info

管理端點 (API)

您可以使用 AWS API 管理核心裝置的端點。

> **Note**
>
> 由於中的用戶端裝置支援與回溯相容AWS IoT Greengrass V1，因AWS IoT Greengrass V2此您可以使用AWS IoT Greengrass V2或 AWS IoT Greengrass V1 API 作業來管理核心裝置端點。

取得核心裝置 (AWSAPI) 的端點

- 請使用下列其中一項作業：
  - 第二版： GetConnectivityInfo
  - V1： GetConnectivityInfo

更新核心裝置 (AWSAPI) 的端點

- 請使用下列其中一項作業：
  - 第二版： UpdateConnectivityInfo
  - V1： UpdateConnectivityInfo

選擇一個 MQTT 經紀商

AWS IoT Greengrass提供選項供您選擇要在核心裝置上執行的本機 MQTT 代理程式。用戶端裝置會連線至核心裝置上執行的 MQTT 代理程式，因此請選擇與您要連線之用戶端裝置相容的 MQTT 代理程式。

> **Note**
>
> 建議您只部署一個 MQTT 代理程式元件。MQTT 橋接器和 IP 偵測器元件一次只能搭配一個 MQTT 代理程式元件使用。如果您部署多個 MQTT 代理程式元件，則必須將它們設定為使用不同的連接埠。

您可以從以下 MQTT 經紀人中進行選擇：

- MQTT 3.1.1 經紀商 (平均) — `aws.greengrass.clientdevices.mqtt.Moquette`

  針對符合 MQTT 3.1.1 標準的輕量型 MQTT 代理程式，請選擇此選項。AWS IoT CoreMQTT 代理程式AWS IoT Device SDK也符合 MQTT 3.1.1 標準，因此您可以使用這些功能建立在您的裝置和. AWS 雲端

- MQTT 5 經紀商 — `aws.greengrass.clientdevices.mqtt.EMQX`

  選擇此選項可在核心裝置與用戶端裝置之間的通訊中使用 MQTT 5 功能。這個組件使用比 Moquette MQTT 3.1.1 代理更多的資源，並在 Linux 核心設備上，它需要碼頭。

  MQTT 5 與 MQTT 3.1.1 向後相容，因此您可以將使用 MQTT 3.1.1 的用戶端裝置連接到此代理程式。如果您運行 Moquette MQTT 3.1.1 代理商，則可以將其替換為 EMQX MQTT 5 代理商，並且客戶端設備可以繼續連接並照常運行。

- 實施自定義代理

選擇此選項可建立用於與用戶端裝置通訊的自訂本機代理程式元件。您可以建立使用 MQTT 以外的通訊協定的自訂本機代理程式。 AWS IoT Greengrass提供可用來驗證和授權用戶端裝置的元件 SDK。如需更多詳細資訊，請參閱 使AWS IoT Device SDK用與 Greengrass 核、其他元件和通訊 AWS IoT Core 及 驗證和授權用戶端裝置。

使用 MQTT 代理程式將用戶端裝置連接至AWS IoT Greengrass核心裝置

當您在AWS IoT Greengrass Core 裝置上使用 MQTT 代理程式時，裝置會使用裝置唯一的核心裝置憑證授權單位 (CA)，向代理程式核發憑證，以便與用戶端建立相互 TLS 連線。

AWS IoT Greengrass將核心裝置 CA 自有核心裝置 CA 建議視為自有核心裝置憑證機構憑證機構 核心裝置 CA 會在連接用戶端裝置驗證元件AWS IoT Greengrass時註冊。自動產生的核心裝置 CA 是永久性的，只要設定用戶端裝置驗證元件，裝置就會繼續使用相同的 CA。

當 MQTT 代理程式啟動時，它會要求憑證。用戶端裝置驗證元件使用核心裝置 CA 發行 X.509 憑證。當 Broker 啟動、憑證到期或連線資訊 (例如 IP 位址) 變更時，會輪替憑證。如需詳細資訊，請參閱本機 MQTT 代理程式上的憑證輪替。

若要將用戶端連接到 MQTT 代理商，您需要下列資訊：

- 用戶端裝置必須具有AWS IoT Greengrass核心裝置 CA。您可以透過雲端探索或手動提供 CA 來取得此 CA。如需詳細資訊，請參閱使用自己的憑證授權單位。
- 核心裝置的完整網域名稱 (FQDN) 或 IP 位址必須存在於核心裝置 CA 核發的代理人憑證中。您可以使用該IP 偵測器組件或手動配置 IP 地址確保這一點。如需詳細資訊，請參閱管理核心裝置端點。
- 客戶端設備身份驗證組件必須授予客戶端設備連接到 Greengrass 核心設備的權限。如需詳細資訊，請參閱用戶端裝置驗證。

使用自己的憑證授權單位

如果您的用戶端裝置無法存取雲端來探索核心裝置，您可以提供核心裝置憑證授權單位 (CA)。您的 Greengrass 核心裝置會使用核心裝置 CA 為您的 MQTT 代理程式發行憑證。一旦您設定核心裝置並使用其 CA 佈建用戶端裝置，您的用戶端裝置就可以連線到端點，並使用核心裝置 CA (自己提供的 CA 或自動產生) 驗證 TLS 交握。

若要將用戶端裝置驗證元件設定為使用核心裝置 CA，請在部署元件時設定`certificateAuthority`組態參數。您必須在組態期間提供下列詳細資訊：

- 核心裝置 CA 憑證的位置。
- 核心裝置 CA 憑證的私密金鑰。
- (選擇性) 如果核心裝置 CA 是中繼 CA，則根憑證的憑證鏈結。

如果您提供核心裝置 CA，請在雲端AWS IoT Greengrass註冊 CA。

您可以將憑證儲存在硬體安全模組或檔案系統中。下列範例顯示使用 HSM/TPM 儲存之中繼 CA 的`certificateAuthority`組態。請注意，憑證鏈結只能儲存在磁碟上。

```
"certificateAuthority": {
    "certificateUri": "pkcs11:object=CustomerIntermediateCA;type=cert",
    "privateKeyUri": "pkcs11:object=CustomerIntermediateCA;type=private"
    "certificateChainUri": "file:///home/ec2-user/creds/certificateChain.pem",
  }
```

在此範例中，`certificateAuthority`組態參數會將用戶端裝置驗證元件設定為使用來自檔案系統的中繼 CA：

```
"certificateAuthority": {
    "certificateUri": "file:///home/ec2-user/creds/intermediateCA.pem",
    "privateKeyUri": "file:///home/ec2-user/creds/intermediateCA.privateKey.pem",
    "certificateChainUri": "file:///home/ec2-user/creds/certificateChain.pem",
  }
```

若要將裝置連線到AWS IoT Greengrass Core 裝置，請執行下列動作：

1. 使用組織的根憑證機構憑證建 Greengrass 憑證授權單位 (CA)。我們將中繼憑證建議視為安全最佳實務。
2. 將中繼 CA 憑證、私密金鑰以及根 CA 的憑證鏈結提供給 Greengrass 核心裝置。如需詳細資訊，請參閱用戶端裝置驗證。中繼 CA 會成為 Greengrass 核心裝置的核心裝置 CA，而且裝置會將 CA 註冊到AWS IoT Greengrass。
3. 將用戶端裝置註冊為AWS IoT物件。如需詳細資訊，請參閱AWS IoT Core開發人員指南中的建立物件物件。將私有金鑰、公有金鑰、裝置憑證，以及根憑證複製到用戶端裝置憑證中。新增資訊的方式取決於您的裝置和軟體。

設定裝置後，您可以使用憑證和公開金鑰鏈來連線到 Greengrass 核心裝置。您的軟體負責尋找核心裝置端點。您可以為核心裝置手動設定端點。如需詳細資訊，請參閱 手動管理端點。

測試用戶端裝置通訊

用戶端裝置可以使用AWS IoT Device SDK來探索、連線和與核心裝置進行通訊。您可以使用中的 Greengrass 探索用戶端AWS IoT Device SDK來使用 Greengrass 探索 API，該 API 會傳回用戶端裝置可連線之核心裝置的相關資訊。API 回應包括用於連線的 MQTT 代理程式端點，以及用來驗證每個核心裝置身分識別的憑證。然後，用戶端裝置可以嘗試每個端點，直到它成功連線到核心裝置為止。

用戶端裝置只能探索與它們建立關聯的核心裝置。在測試用戶端裝置與核心裝置之間的通訊之前，您必須將用戶端裝置與核心裝置建立關聯。如需詳細資訊，請參閱 關聯用戶端裝置。

Greengrass 探索 API 會傳回您指定的核心裝置 MQTT 代理程式端點。您可以使用 IP 偵測器元件為您管理這些端點，也可以為每個核心裝置手動管理這些端點。如需詳細資訊，請參閱 管理核心裝置端點。

> **Note**
>
> 若要使用 Greengrass 探索 API，用戶端裝置必須具有權限。`greengrass:Discover`如需詳細資訊，請參閱 用戶端裝置的最低AWS IoT原則。

提供多種程式設計語言版本AWS IoT Device SDK。如需詳細資訊，請參閱AWS IoT Core開發人員指南中的AWS IoT裝置 SDK。

主題

- 測試通訊
- 測試通訊
- 測試通訊 (JavaScript)
- 測試通訊

測試通訊

在本節中，您可以使用適用於 Python 的 AWS IoT Device SDK v2 中的 Greengrass 探索範例來測試用戶端裝置與核心裝置之間的通訊。

> **Important**
>
> 若要將 AWS IoT Device SDK V2 用於 Python，裝置必須執行 Python 3.6 或更新版本。

要測試通信（對於 Python 的 AWS IoT Device SDK v2）

1. 下載並安裝適用於 Python 的 AWS IoT Device SDK v2 到作為客戶端設備連接的AWS IoT東西。

   在用戶端裝置上，執行下列動作：

   a. 克隆 AWS IoT Device SDK v2 的 Python 存儲庫下載它。

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

   b. 安裝適用於 Python 的 AWS IoT Device SDK V2。

   ```
   python3 -m pip install --user ./aws-iot-device-sdk-python-v2
   ```

2. 切換到示例文件夾在 AWS IoT Device SDK v2 中的 Python。

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

3. 執行範例 Greengrass 探索應用程式。此應用程式需要引數，這些引數會指定用戶端裝置物件名稱、要使用的 MQTT 主題和訊息，以及驗證和保護連線的憑證。下列範例會將 Hello World 訊息傳送至clients/*MyClientDevice1*/hello/world主題。

   - 將 *MyClientDevice1* 取代為用戶端裝置的物件名稱。
   - 將 *~/##/ AmazonRoot CA1.pem* 取代為用戶端裝置上 Amazon 根 CA 憑證的路徑。
   - 將*#/##/## .pem.crt* 替換為客戶端設備上的設備證書的路徑。
   - 將*#/##/## .pem.key ############*件的路徑。
   - 將 *us-east-1* 取代為用戶端裝置和核心裝置運作的AWS區域。

   ```
   python3 basic_discovery.py \\
     --thing_name MyClientDevice1 \\
     --topic 'clients/MyClientDevice1/hello/world' \\
     --message 'Hello World!' \\
     --ca_file ~/certs/AmazonRootCA1.pem \\
     --cert ~/certs/device.pem.crt \\
     --key ~/certs/private.pem.key \\
     --region us-east-1 \\
     --verbosity Warn
   ```

探索範例應用程式會傳送訊息 10 次並中斷連線。它也會訂閱發佈訊息的相同主題。如果輸出指出應用程式收到有關該主題的 MQTT 訊息，則用戶端裝置可以成功與核心裝置通訊。

```
Performing greengrass discovery...
awsiot.greengrass_discovery.DiscoverResponse(gg_groups=[awsiot.greengrass_discovery.GGGroup
coreDevice-MyGreengrassCore',
 cores=[awsiot.greengrass_discovery.GGCore(thing_arn='arn:aws:iot:us-
east-1:123456789012:thing/MyGreengrassCore',
 connectivity=[awsiot.greengrass_discovery.ConnectivityInfo(id='203.0.113.0',
 host_address='203.0.113.0', metadata='', port=8883)])],
 certificate_authorities=['-----BEGIN CERTIFICATE-----\
MIICiT...EXAMPLE=\
-----END CERTIFICATE-----\
'])])
Trying core arn:aws:iot:us-east-1:123456789012:thing/MyGreengrassCore at host
 203.0.113.0 port 8883
Connected!
Published topic clients/MyClientDevice1/hello/world: {"message": "Hello World!",
 "sequence": 0}

Publish received on topic clients/MyClientDevice1/hello/world
b'{"message": "Hello World!", "sequence": 0}'
Published topic clients/MyClientDevice1/hello/world: {"message": "Hello World!",
 "sequence": 1}

Publish received on topic clients/MyClientDevice1/hello/world
b'{"message": "Hello World!", "sequence": 1}'

...

Published topic clients/MyClientDevice1/hello/world: {"message": "Hello World!",
 "sequence": 9}

Publish received on topic clients/MyClientDevice1/hello/world
b'{"message": "Hello World!", "sequence": 9}'
```

如果應用程式輸出錯誤，請參閱 Greengrass 探索問題疑難排解。

您也可以在核心裝置上檢視 Greengrass 記錄檔，以驗證用戶端裝置是否成功連線並傳送訊息。如需詳細資訊，請參閱 監控AWS IoT Greengrass日誌。

測試通訊

在本節中，您可以在 AWS IoT Device SDKv2 中針對 C++ 使用 Greengrass 探索範例來測試用戶端裝置與核心裝置之間的通訊。

要為 C ++ 構建 AWS IoT Device SDK v2，設備必須具有以下工具：

- C ++ 11 或更高版本
- 製造 3.1 或更高版本
- 下列其中一個編譯器：
  - 海灣合作委員會 4.8 或
  - 鐺 3.9 或更高版本
  - 無線電視廣告 2015 或以後

若要測試通訊 (C++ 為 AWS IoT Device SDK v2)

1. 下載並構建 C ++ 的 AWS IoT Device SDK v2 以作為客戶端設備連接的AWS IoT東西。

   在用戶端裝置上，執行下列動作：

   a. 為 C ++ 工作區 AWS IoT Device SDK v2 創建一個文件夾，並對其進行更改。

   ```
   cd
   mkdir iot-device-sdk-cpp
   cd iot-device-sdk-cpp
   ```

   b. 克隆 AWS IoT Device SDK v2 用於 C ++ 存儲庫以下載它。該`--recursive`標誌指定下載子模塊。

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

   c. 為 C ++ 構建輸出的 AWS IoT Device SDK v2 創建一個文件夾，然後更改為它。

   ```
   mkdir aws-iot-device-sdk-cpp-v2-build
   cd aws-iot-device-sdk-cpp-v2-build
   ```

   d. 為 C ++ 構建 AWS IoT Device SDK V2。

   ```
   cmake -DCMAKE_INSTALL_PREFIX="~/iot-device-sdk-cpp" -
   DCMAKE_BUILD_TYPE="Release" ../aws-iot-device-sdk-cpp-v2
   ```

```
cmake --build . --target install
```

2. 在適用於 C++ 的 AWS IoT Device SDK v2 中建置 Greengrass 探索範例應用程式。請執行下列操作：

   a. 變更為適用於 C++ 的 AWS IoT Device SDK v2 中的 Greengrass 探索範例資料夾。

   ```
   cd ../aws-iot-device-sdk-cpp-v2/samples/greengrass/basic_discovery
   ```

   b. 為 Greengrass 探索範例建置輸出建立資料夾，並加以變更。

   ```
   mkdir build
   cd build
   ```

   c. 建置 Greengrass 探索範例應用程式。

   ```
   cmake -DCMAKE_PREFIX_PATH="~/iot-device-sdk-cpp" -
   DCMAKE_BUILD_TYPE="Release" ..
   cmake --build . --config "Release"
   ```

3. 執行範例 Greengrass 探索應用程式。此應用程式需要指定用戶端裝置物件名稱、要使用的 MQTT 主題，以及驗證和保護連線的憑證的引數。下列範例會訂閱`clients/`*`MyClientDevice1`*`/hello/world`主題，並將您在指令行輸入的訊息發佈至相同主題。

   - 將 *`MyClientDevice1`* 取代為用戶端裝置的物件名稱。
   - 將 *`~/##/ AmazonRoot CA1.pem`* 取代為用戶端裝置上 Amazon 根 CA 憑證的路徑。
   - 將*`#/##/## .pem.crt`* 替換為客戶端設備上的設備證書的路徑。
   - 將*`#/##/## .pem.key ############`*件的路徑。
   - 將 *`us-east-1`* 取代為用戶端裝置和核心裝置運作的AWS區域。

   ```
   ./basic-discovery \
     --thing_name MyClientDevice1 \
     --topic 'clients/MyClientDevice1/hello/world' \
     --ca_file ~/certs/AmazonRootCA1.pem \
     --cert ~/certs/device.pem.crt \
     --key ~/certs/private.pem.key \
     --region us-east-1
   ```

   探索範例應用程式會訂閱主題，並提示您輸入要發佈的訊息。

```
Connecting to group greengrassV2-coreDevice-MyGreengrassCore with thing arn
 arn:aws:iot:us-east-1:123456789012:thing/MyGreengrassCore, using endpoint
 203.0.113.0:8883
Connected to group greengrassV2-coreDevice-MyGreengrassCore, using connection to
 203.0.113.0:8883
Successfully subscribed to clients/MyClientDevice1/hello/world
Enter the message you want to publish to topic clients/MyClientDevice1/hello/world
 and press enter. Enter 'exit' to exit this program.
```

如果應用程式輸出錯誤，請參閱 Greengrass 探索問題疑難排解。

4. 輸入訊息，例如**`Hello World!`**。

```
Enter the message you want to publish to topic clients/MyClientDevice1/hello/world
 and press enter. Enter 'exit' to exit this program.
Hello World!
```

如果輸出指出應用程式收到有關該主題的 MQTT 訊息，則用戶端裝置可以成功與核心裝置通訊。

```
Operation on packetId 2 Succeeded
Publish received on topic clients/MyClientDevice1/hello/world
Message:
Hello World!
```

您也可以在核心裝置上檢視 Greengrass 記錄檔，以驗證用戶端裝置是否成功連線並傳送訊息。如需詳細資訊，請參閱 監控AWS IoT Greengrass日誌。

測試通訊 (JavaScript)

在本節中，您可以使用 AWS IoT Device SDKv2 中的 Greengrass 探索範例 JavaScript來測試用戶端裝置與核心裝置之間的通訊。

> **Important**
>
> 若要將 AWS IoT Device SDK v2 用於 JavaScript，裝置必須執行節點 v10.0 或更新版本。

若要測試通訊 (AWS IoT Device SDKv2 用於 JavaScript)

1. 下載並安裝 AWS IoT Device SDKv2 JavaScript 以作為客戶端設備連接的AWS IoT物件。

   在用戶端裝置上，執行下列動作：

   a. 克隆 AWS IoT Device SDK v2 以供 JavaScript 存儲庫下載它。

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

   b. 安裝適用於的 AWS IoT Device SDK v2 JavaScript。

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

2. 變更到 v2 中的 Greengrass 探索範例資料夾。AWS IoT Device SDK JavaScript

   ```
   cd samples/node/basic_discovery
   ```

3. 安裝 Greengrass 探索範例應用程式。

   ```
   npm install
   ```

4. 執行範例 Greengrass 探索應用程式。此應用程式需要引數，這些引數會指定用戶端裝置物件名稱、要使用的 MQTT 主題和訊息，以及驗證和保護連線的憑證。下列範例會將 Hello World 訊息傳送至clients/*MyClientDevice1*/hello/world主題。

   - 將 *MyClientDevice1* 取代為用戶端裝置的物件名稱。
   - 將 *~/##/ AmazonRoot CA1.pem* 取代為用戶端裝置上 Amazon 根 CA 憑證的路徑。
   - 將*#/##/## .pem.crt* 替換為客戶端設備上的設備證書的路徑。
   - 將*#/##/## .pem.key ############*件的路徑。
   - 將 *us-east-1* 取代為用戶端裝置和核心裝置運作的AWS區域。

   ```
   node dist/index.js \
     --thing_name MyClientDevice1 \
     --topic 'clients/MyClientDevice1/hello/world' \
     --message 'Hello World!' \
     --ca_file ~/certs/AmazonRootCA1.pem \
     --cert ~/certs/device.pem.crt \
   ```

```
  --key ~/certs/private.pem.key \
  --region us-east-1 \
  --verbose warn
```

探索範例應用程式會傳送訊息 10 次並中斷連線。它也會訂閱發佈訊息的相同主題。如果輸出指出應用程式收到有關該主題的 MQTT 訊息，則用戶端裝置可以成功與核心裝置通訊。

```
Discovery Response:
{"gg_groups":[{"gg_group_id":"greengrassV2-coreDevice-
MyGreengrassCore","cores":[{"thing_arn":"arn:aws:iot:us-
east-1:123456789012:thing/MyGreengrassCore","connectivity":
[{"id":"203.0.113.0","host_address":"203.0.113.0","port":8883,"metadata":""}]}],"certificat
["-----BEGIN CERTIFICATE-----\nMIICiT...EXAMPLE=\n-----END CERTIFICATE-----\n"]}]}
Trying
 endpoint={"id":"203.0.113.0","host_address":"203.0.113.0","port":8883,"metadata":""}
[WARN] [2021-06-12T00:46:45Z] [00007f90c0e8d700] [socket] - id=0x7f90b8018710
 fd=26: setsockopt() for NO_SIGNAL failed with errno 92. If you are having SIGPIPE
 signals thrown, you may want to install a signal trap in your application layer.
Connected to
 endpoint={"id":"203.0.113.0","host_address":"203.0.113.0","port":8883,"metadata":""}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":1}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":2}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":3}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":4}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":5}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":6}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":7}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
```

```
{"message":"Hello World!","sequence":8}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":9}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":10}
Complete!
```

如果應用程式輸出錯誤，請參閱 Greengrass 探索問題疑難排解。

您也可以在核心裝置上檢視 Greengrass 記錄檔，以驗證用戶端裝置是否成功連線並傳送訊息。如需詳細資訊，請參閱 監控AWS IoT Greengrass日誌。

測試通訊

在本節中，您可以使用適用於 Java 的 AWS IoT Device SDK v2 中的 Greengrass 探索範例來測試用戶端裝置與核心裝置之間的通訊。

> **Important**
>
> 要為 Java 構建 AWS IoT Device SDK v2，設備必須具有以下工具：
>
> - Java 8 或更高版本，並`JAVA_HOME`指向 Java 文件夾。
> - Apache Maven

若要測試通訊 (Java 為 AWS IoT Device SDK v2)

1. 下載並構建適用於 Java 的 AWS IoT Device SDK v2 以作為客戶端設備連接的AWS IoT東西。

   在用戶端裝置上，執行下列動作：

   a. 克隆 AWS IoT Device SDK V2 的 Java 存儲庫下載它。

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

   b. 更改為 AWS IoT Device SDK V2 的 Java 文件夾。

   c. 建置適用於 Java 的 AWS IoT Device SDK V2。

```
cd aws-iot-device-sdk-java-v2
mvn versions:use-latest-versions -Dincludes="software.amazon.awssdk.crt*"
mvn clean install
```

2. 執行範例 Greengrass 探索應用程式。此應用程式需要指定用戶端裝置物件名稱、要使用的 MQTT 主題，以及驗證和保護連線的憑證的引數。下列範例會訂閱clients/*MyClientDevice1*/hello/world主題，並將您在指令行輸入的訊息發佈至相同主題。

   - 將 *MyClientDevice1* 的兩個執行個體取代為用戶端裝置的物件名稱。
   - 以用戶端裝置上 Amazon 根 CA *####### $ ##/##/AmazonRootCA1.PEM*。
   - 將 *$##/##/## .pem.crt* 取代為用戶端裝置上裝置憑證的路徑。
   - 將 *$##/##/##.pem.key ############*檔案的路徑。
   - 將 *us-east-1* 替換為客戶端設備和核心設備的運行AWS 區域位置。

   ```
   DISCOVERY_SAMPLE_ARGS="--thing_name MyClientDevice1 \
     --topic 'clients/MyClientDevice1/hello/world' \
     --ca_file $HOME/certs/AmazonRootCA1.pem \
     --cert $HOME/certs/device.pem.crt \
     --key $HOME/certs/private.pem.key \
     --region us-east-1"

   mvn exec:java -pl samples/Greengrass \
     -Dexec.mainClass=greengrass.BasicDiscovery \
     -Dexec.args="$DISCOVERY_SAMPLE_ARGS"
   ```

   探索範例應用程式會訂閱主題，並提示您輸入要發佈的訊息。

   ```
   Connecting to group ID greengrassV2-coreDevice-MyGreengrassCore, with thing
    arn arn:aws:iot:us-east-1:123456789012:thing/MyGreengrassCore, using endpoint
    203.0.113.0:8883
   Started a clean session
   Enter the message you want to publish to topic clients/MyClientDevice1/hello/world
    and press Enter. Type 'exit' or 'quit' to exit this program:
   ```

   如果應用程式輸出錯誤，請參閱 Greengrass 探索問題疑難排解。

3. 輸入訊息，例如**Hello World!**。

```
Enter the message you want to publish to topic clients/MyClientDevice1/hello/world
 and press Enter. Type 'exit' or 'quit' to exit this program:
Hello World!
```

如果輸出指出應用程式收到有關該主題的 MQTT 訊息，則用戶端裝置可以成功與核心裝置通訊。

```
Message received on topic clients/MyClientDevice1/hello/world: Hello World!
```

您也可以在核心裝置上檢視 Greengrass 記錄檔，以驗證用戶端裝置是否成功連線並傳送訊息。如需更多詳細資訊，請參閱 監控AWS IoT Greengrass日誌。

Greengrass 發現寧靜 API

AWS IoT Greengrass提供用戶端裝置可用來識別 Greengrass 核心裝置在其中連線的 `Discover` API 作業。用戶端裝置會使用此資料平面作業擷取連線到 Greengrass 核心裝置所需的資訊，以便將它們與 API 作業相關聯。BatchAssociateClientDeviceWithCoreDevice當用戶端裝置上線時，它可以連線到 AWS IoT Greengrass雲端服務，並使用探索 API 來尋找：

- 每個關聯的 Greengrass 核心裝置的 IP 位址和連接埠。
- 核心裝置 CA 憑證，用戶端裝置可用來驗證 Greengrass 核心裝置。

> **Note**
>
> 用戶端裝置也可以使用中的探索用戶端AWS IoT Device SDK來探索 Greengrass 核心裝置的連線資訊。探索用戶端會使用探索 API。如需詳細資訊，請參閱下列內容：
>
> - 測試用戶端裝置通訊
> - 開發人員指南中的格林格拉斯發現 REST 風格 API。AWS IoT Greengrass Version 1

若要使用此 API 作業，請將 HTTP 要求傳送至 Greengrass 資料平面端點上的探索 API。此 API 端點具有以下格式。

```
https://greengrass-ats.iot.region.amazonaws.com:port/greengrass/discover/thing/thing-name
```

如需AWS IoT Greengrass探查 API 的支援AWS 區域和AWS IoT Greengrass V2端點清單，請參閱 AWS 一般參考. 此 API 作業只能在 Greengrass 資料平面端點上使用。用於管理元件和部署的控制平面端點與資料平面端點不同。

> **Note**
>
> 和的探索 API 是相同AWS IoT Greengrass V1的AWS IoT Greengrass V2。如果您有連接到 AWS IoT Greengrass V1核心的用戶端裝置，您可以將它們連接到AWS IoT Greengrass V2核心裝置，而無需變更用戶端裝置上的程式碼。如需詳細資訊，請參閱開發人員指南中的《綠色探索 RESTful API》。AWS IoT Greengrass Version 1

主題

探索驗證和授權

若要使用探索 API 擷取連線資訊，用戶端裝置必須使用 TLS 相互驗證與 X.509 用戶端憑證來進行驗證。如需詳細資訊，請參閱AWS IoT Core開發人員指南中的 X.509 用戶端憑證。

用戶端裝置也必須具有執行處理行`greengrass:Discover`動的權限。下列範例AWS IoT原則允許名`MyClientDevice1`為的AWS IoT物件`Discover`為自己執行。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "greengrass:Discover",
      "Resource": [
        "arn:aws:iot:us-west-2:123456789012:thing/MyClientDevice1"
      ]
    }
  ]
}
```

> **Important**
>
> 核心裝置或 Greengrass 資料平面作業的原AWS IoT則中不支援物件原則變數 (`iot:Connection.Thing.*`)。相反地，您可以使用萬用字元來比對具有相似名稱的多個裝置。例如，您可以指定`MyGreengrassDevice*`要相符`MyGreengrassDevice1`、`MyGreengrassDevice2`、等等。

如需詳細資訊，請參閱AWS IoT Core開發人員指南中的AWS IoT Core政策。

請求

要求包含標準 HTTP 標頭，並會傳送至 Greengrass 探索端點，如下列範例所示。

連接埠號碼取決於核心裝置是設定為透過連接埠 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”。

> **Note**
>
> 這些範例使用 Amazon 信任服務 (ATS) 端點，該端點可與建議的 ATS 根 CA 憑證搭配使用。端點必須符合根 CA 憑證類型。

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
```

> **Note**
>
> 在連接埠 443 上連線的用戶端必須實作應用程式層通訊協定交涉 (ALPN) TLS 延伸模組，並按`x-amzn-http-ca`照`ProtocolName`中的. `ProtocolNameList` 如需詳細資訊，請參閱AWS IoT開發人員指南中的通訊協定。

回應

成功之後，回應標頭會包含 HTTP 200 狀態碼，而回應主體則包含探索回應文件。

> **Note**
>
> 因為AWS IoT Greengrass V2使用與相同的探索 APIAWS IoT Greengrass V1，因此回應會根據AWS IoT Greengrass V1概念 (例如 Greengrass 群組) 來組織資訊。響應包 Greengrass 色組的列表。在中AWS IoT Greengrass V2，每個核心裝置都位於自己的群組中，其中該群組僅包含該核心裝置及其連線資訊。

範例 Discover 回應文件

下列文件顯示與一個 Greengrass 核心裝置相關聯的用戶端裝置的回應。核心裝置具有一個端點和一個 CA 憑證。

```
{
  "GGGroups": [
    {
      "GGGroupId": "greengrassV2-coreDevice-core-device-01-thing-name",
      "Cores": [
        {
          "thingArn": "core-device-01-thing-arn",
          "Connectivity": [
            {
              "id": "core-device-01-connection-id",
              "hostAddress": "core-device-01-address",
              "portNumber": core-device-01-port,
              "metadata": "core-device-01-description"
            }
          ]
        }
      ],
      "CAs": [
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----"
      ]
    }
  ]
}
```

下列文件顯示與兩個核心裝置相關聯的用戶端裝置的回應。核心裝置具有多個端點和多個群組 CA 憑證。

```
{
  "GGGroups": [
    {
      "GGGroupId": "greengrassV2-coreDevice-core-device-01-thing-name",
      "Cores": [
        {
          "thingArn": "core-device-01-thing-arn",
          "Connectivity": [
            {
              "id": "core-device-01-connection-id",
              "hostAddress": "core-device-01-address",
              "portNumber": core-device-01-port,
              "metadata": "core-device-01-connection-1-description"
            },
            {
              "id": "core-device-01-connection-id-2",
              "hostAddress": "core-device-01-address-2",
              "portNumber": core-device-01-port-2,
              "metadata": "core-device-01-connection-2-description"
            }
          ]
        }
      ],
      "CAs": [
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----"
      ]
    },
    {
      "GGGroupId": "greengrassV2-coreDevice-core-device-02-thing-name",
      "Cores": [
        {
          "thingArn":"core-device-02-thing-arn",
          "Connectivity" : [
            {
              "id": "core-device-02-connection-id",
              "hostAddress": "core-device-02-address",
              "portNumber": core-device-02-port,
              "metadata": "core-device-02-connection-1-description"
```

```
            }
          ]
        }
      ],
      "CAs": [
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----"
      ]
    }
  ]
}
```

使用 cURL 測試探索 API

如果您已cURL安裝，則可以測試探索 API。下列範例會指定用戶端裝置的憑證，以驗證對 Greengrass 探索 API 端點的要求。

```
curl -i \
  --cert 1a23bc4d56.cert.pem \
  --key 1a23bc4d56.private.key \
  https://greengrass-ats.iot.us-west-2.amazonaws.com:8443/greengrass/discover/
thing/MyClientDevice1
```

> **Note**
>
> 引-i數指定要輸出 HTTP 回應標頭。您可以使用此選項來協助識別錯誤。

如果要求成功，此命令會輸出類似下列範例的回應。

```
{
  "GGGroups": [
    {
      "GGGroupId": "greengrassV2-coreDevice-MyGreengrassCore",
      "Cores": [
        {
          "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
          "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"
      ]
    }
  ]
}
```

如果命令輸出錯誤，請參閱 Greengrass 探索問題疑難排解。

在用戶端裝置之間轉送 MQTT 訊息，AWS IoT Core

您可以在用戶端裝置與之間轉送 MQTT 訊息和其他資料。AWS IoT Core用戶端裝置會連線至核心裝置上執行的 MQTT 代理程式元件。根據預設，核心裝置不會在用戶端裝置和. AWS IoT Core 根據預設，用戶端裝置只能透過 MQTT 互相通訊。

若要在用戶端裝置之間轉送 MQTT 訊息AWS IoT Core，請將 MQTT 橋接器元件設定為執行下列動作：

- 將訊息從用戶端裝置轉送至AWS IoT Core。
- 將訊息從轉送AWS IoT Core至用戶端裝置。

> **Note**
>
> 即使用戶端裝置使用 QoS 0 發佈和訂閱本機 MQTT 代理程式AWS IoT Core，MQTT 橋接器也會使用 QoS 1 來發佈和訂閱。因此，當您將 MQTT 訊息從本機 MQTT 代理程式上的用戶端裝置轉送至時，可能會發現額外的延遲。AWS IoT Core如需核心裝置上 MQTT 組態的詳細資訊，請參閱。設定 MQTT 逾時和快取設定

主題

- 設定和部署 MQTT 橋接器元件
- 轉送 MQTT 訊息

設定和部署 MQTT 橋接器元件

MQTT 橋接器元件會使用主題對應清單，每個主題對應都會指定訊息來源和訊息目的地。若要在用戶端裝置之間轉送訊息AWS IoT Core，請部署 MQTT 橋接器元件，並在元件組態中指定每個來源和目的地主題。

若要將 MQTT 橋接元件部署到核心裝置或核心裝置群組，請建立包含元件的部署。`aws.greengrass.clientdevices.mqtt.Bridge`在部署的 MQTT 橋接器元件組態中指定主題對應。`mqttTopicMapping`

下列範例會定義將 MQTT 橋接元件設定為將符合主題篩選器的主題訊息從用戶端裝置轉送至的`clients/+/hello/world`部署。AWS IoT Core組`merge`態更新需要序列化的 JSON 物件。如需詳細資訊，請參閱 更新零組件組態。

Console

```
{
  "mqttTopicMapping": {
    "HelloWorldIotCore": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

AWS CLI

```
{
  "components": {
    "aws.greengrass.clientdevices.mqtt.Bridge": {
      "version": "2.0.0",
      "configurationUpdate": {
        "merge": "{\"mqttTopicMapping\":{\"HelloWorldIotCore\":{\"topic\":\"clients/
+/hello/world\",\"source\":\"LocalMqtt\",\"target\":\"IotCore\"}}}"
      }
    }
    ...
  }
}
```

轉送 MQTT 訊息

若要在用戶端裝置之間轉送 MQTT 訊息AWS IoT Core，請設定和部署 MQTT Bridge 元件，並指定要轉送的主題。

Example 範例：將主題的訊息從用戶端裝置轉送至 AWS IoT Core

下列 MQTT 橋接器元件組態會針對符合從用戶端裝置的主題篩選器的主`clients/+/hello/world/event`題，指定轉送郵件。AWS IoT Core

```
{
  "mqttTopicMapping": {
    "HelloWorldEvent": {
      "topic": "clients/+/hello/world/event",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

Example 範例：將主題的訊息從用戶端裝置轉送AWS IoT Core至用戶端裝置

下列 MQTT 橋接器元件組態會指定將符合主題篩選器的主題訊息轉送AWS IoT Core至用戶端裝置。`clients/+/hello/world/event/response`

```
{
  "mqttTopicMapping": {
    "HelloWorldEventConfirmation": {
      "topic": "clients/+/hello/world/event/response",
      "source": "IotCore",
      "target": "LocalMqtt"
    }
  }
}
```

與元件中的用戶端裝置互動

您可以開發與連接至核心裝置的用戶端裝置互動的自訂 Greengrass 元件。例如，您可以開發執行下列作業的元件：

- 處理來自用戶端裝置的 MQTT 訊息，並將資料傳送至AWS 雲端目的地。

- 將 MQTT 訊息傳送至用戶端裝置以啟動處理行動。

用戶端裝置會透過核心裝置上執行的 MQTT Broker 元件連線至核心裝置並與核心裝置進行通訊。根據預設，用戶端裝置只能透過 MQTT 互相通訊，而 Greengrass 元件無法接收這些 MQTT 訊息或將訊息傳送至用戶端裝置。

Greengrass 組件使用本地發布/訂閱接口在核心設備上進行通信。若要與 Greengrass 元件中的用戶端裝置通訊，請設定 MQTT 橋接器元件以執行下列動作：

- 將 MQTT 訊息從用戶端裝置轉送至本機發佈/訂閱。
- 轉送來自本機發佈/訂閱用戶端裝置的 MQTT 訊息。

您還可以與 Greengrass 組件中的客戶端設備陰影進行交互。如需詳細資訊，請參閱 與用戶端裝置陰影互動並同步。

主題

設定和部署 MQTT 橋接器元件

MQTT 橋接器元件會使用主題對應清單，每個主題對應都會指定訊息來源和訊息目的地。若要與用戶端裝置通訊，請部署 MQTT 橋接器元件，並在元件組態中指定每個來源和目的地主題。

若要將 MQTT 橋接元件部署到核心裝置或核心裝置群組，請建立包含元件的部署。`aws.greengrass.clientdevices.mqtt.Bridge`在部署的 MQTT 橋接器元件組態中指定主題對應。`mqttTopicMapping`

下列範例會定義將 MQTT 橋接器元件設定為將`clients/MyClientDevice1/hello/world`主題從用戶端裝置轉送至本機發佈/訂閱代理程式的部署。組`merge`態更新需要序列化的 JSON 物件。如需詳細資訊，請參閱 更新零組件組態。

Console

```
{
  "mqttTopicMapping": {
```

```
    "HelloWorldPubsub": {
      "topic": "clients/MyClientDevice1/hello/world",
      "source": "LocalMqtt",
      "target": "Pubsub"
    }
  }
}
```

AWS CLI

```
{
  "components": {
    "aws.greengrass.clientdevices.mqtt.Bridge": {
      "version": "2.0.0",
      "configurationUpdate": {
        "merge": "\"mqttTopicMapping\":{\"HelloWorldPubsub\":{\"topic\":\"clients/
MyClientDevice1/hello/world\",\"source\":\"LocalMqtt\",\"target\":\"Pubsub\"}}}"
      }
    }
    ...
  }
}
```

您可以使用 MQTT 主題萬用字元來轉送符合主題篩選器之主題的郵件。如果您使用 MQTT 橋接器 v2.2.0 或更新版本，則當來源代理程式為本機發佈/訂閱時，您可以在主題篩選器中使用 MQTT 主題萬用字元。如需詳細資訊，請參閱 MQTT 橋接器元件組態。

從用戶端裝置接收 MQTT 訊息

您可以訂閱您為 MQTT 橋接元件設定的本機發佈/訂閱主題，以接收來自用戶端裝置的訊息。

在自訂元件中接收來自用戶端裝置的 MQTT 訊息

1. 設定和部署 MQTT 橋接器元件，以轉送來自 MQTT 主題的訊息，其中用戶端裝置會發佈至本機發佈/訂閱主題。
2. 使用本機發佈/訂閱IPC 介面訂閱 MQTT 橋接器轉送訊息的主題。如需詳細資訊，請參閱 發佈/訂閱本地訊息 及 SubscribeToTopic。

[Connect 並測試用戶端裝置] 教學課程包含一個區段，您可以在其中開發可從用戶端裝置訂閱訊息的元件。如需詳細資訊，請參閱 步驟 4：開發與用戶端裝置通訊的元件。

將 MQTT 訊息傳送至用戶端裝置

您可以發佈到您為 MQTT 橋接元件設定的本機發佈/訂閱主題，以便將訊息傳送至用戶端裝置。

若要將 MQTT 訊息發佈到自訂元件中的用戶端裝置

1. 設定並部署 MQTT 橋接器元件，將來自本機發佈/訂閱主題的訊息轉送至用戶端裝置訂閱的 MQTT 主題。
2. 使用本機發佈/訂閱IPC 介面，發佈至 MQTT 橋接器轉送郵件的主題。如需更多詳細資訊，請參閱發佈/訂閱本地訊息 及 PublishToTopic。

與用戶端裝置陰影互動並同步

您可以使用陰影管理員元件來管理本機陰影，包括用戶端裝置陰影。您可以使用陰影管理員執行下列作業：

- 與 Greengrass 組件中的客戶端設備陰影進行交互。
- 與用戶端裝置陰影同步AWS IoT Core。

> **Note**
>
> 根AWS IoT Core據預設，陰影管理員組件不會與陰影同步。您必須設定陰影管理員元件，以指定要同步的用戶端裝置陰影。

主題

必要條件

若要與用戶端裝置陰影互動並將用戶端裝置陰影與之同步AWS IoT Core，核心裝置必須符合下列需求：

- 除了用戶端裝置支援的 Greengrass 元件之外，核心裝置還必須執行下列元件：
  - Greengrass 核 v2.6.0 或更高版本
  - 陰影管理器 v2.2.0 或更新版本
  - MQTT 橋接器 v2.2.0 或更新版本
- 必須將用戶端裝置驗證元件設定為允許用戶端裝置在裝置陰影主題上進行通訊。

啟用陰影管理員與用戶端裝置通訊

根據預設，陰影管理員元件不會管理用戶端裝置的陰影。若要啟用此功能，您必須在用戶端裝置和陰影管理員元件之間轉送 MQTT 訊息。用戶端裝置會使用 MQTT 訊息來接收和傳送裝置陰影更新。陰影管理員元件會訂閱本機 Greengrass 發佈/訂閱介面，因此您可以將 MQTT 橋接元件設定為轉送裝置陰影主題上的 MQTT 訊息。

MQTT 橋接器元件會使用主題對應清單，每個主題對應都會指定訊息來源和訊息目的地。若要讓陰影管理員元件管理用戶端裝置陰影，請部署 MQTT 橋接器元件，並指定用戶端裝置陰影的陰影主題。您必須將橋接器設定為在本機 MQTT 與本機發佈/訂閱之間雙向轉送訊息。

若要將 MQTT 橋接元件部署到核心裝置或核心裝置群組，請建立包含元件的部署。`aws.greengrass.clientdevices.mqtt.Bridge`在部署的 MQTT 橋接器元件組態中指定主題對應。`mqttTopicMapping`

使用下列範例來設定 MQTT 橋接器元件，以啟用用戶端裝置與陰影管理員元件之間的通訊。

> **Note**
>
> 您可以在AWS IoT Greengrass主控台中使用這些設定範例。如果您使用 AWS IoT Greengrass API，`merge`組態更新需要序列化的 JSON 物件，因此您必須將下列 JSON 物件序列化為字串。如需詳細資訊，請參閱 更新零組件組態。

Example 範例：管理所有用戶端裝置陰影

下列 MQTT 橋接器設定範例可讓陰影管理員管理所有用戶端裝置的所有陰影。

```
{
  "mqttTopicMapping": {
    "ShadowsLocalMqttToPubsub": {
      "topic": "$aws/things/+/shadow/#",
```

```
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ShadowsPubsubToLocalMqtt": {
      "topic": "$aws/things/+/shadow/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    }
  }
}
```

Example 範例：管理用戶端裝置的陰影

下列 MQTT 橋接器設定範例可讓陰影管理員管理名為的用戶端裝置的所有陰影。MyClientDevice

```
{
  "mqttTopicMapping": {
    "ShadowsLocalMqttToPubsub": {
      "topic": "$aws/things/MyClientDevice/shadow/#",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ShadowsPubsubToLocalMqtt": {
      "topic": "$aws/things/MyClientDevice/shadow/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    }
  }
}
```

Example 範例：管理所有用戶端裝置的已命名陰影

下列 MQTT 橋接器組態範例可讓陰影管理員管理DeviceConfiguration為所有用戶端裝置命名的陰影。

```
{
  "mqttTopicMapping": {
    "ShadowsLocalMqttToPubsub": {
      "topic": "$aws/things/+/shadow/name/DeviceConfiguration/#",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ShadowsPubsubToLocalMqtt": {
```

```
      "topic": "$aws/things/+/shadow/name/DeviceConfiguration/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    }
  }
}
```

Example 範例：管理所有用戶端裝置的未命名陰影

下列 MQTT 橋接器組態範例可讓陰影管理員管理所有用戶端裝置的未命名陰影，但未命名陰影。

```
{
  "mqttTopicMapping": {
    "DeleteShadowLocalMqttToPubsub": {
      "topic": "$aws/things/+/shadow/delete",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "DeleteShadowPubsubToLocalMqtt": {
      "topic": "$aws/things/+/shadow/delete/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    },
    "GetShadowLocalMqttToPubsub": {
      "topic": "$aws/things/+/shadow/get",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "GetShadowPubsubToLocalMqtt": {
      "topic": "$aws/things/+/shadow/get/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    },
    "UpdateShadowLocalMqttToPubsub": {
      "topic": "$aws/things/+/shadow/update",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "UpdateShadowPubsubToLocalMqtt": {
      "topic": "$aws/things/+/shadow/update/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    }
  }
```

```
}
```

與元件中的用戶端裝置陰影互動

您可以開發使用本機陰影服務來讀取和修改用戶端裝置的本機陰影文件的自訂元件。如需詳細資訊，請參閱 與組件中的陰影互動。

同步用戶端裝置陰影 AWS IoT Core

您可以將陰影管理員元件設定為與同步處理本機用戶端裝置陰影狀態AWS IoT Core。如需更多詳細資訊，請參閱 同步本地設備陰影 AWS IoT Core。

疑難排解用戶端

使用本節中的疑難排解資訊和解決方案，協助解決 Greengrass 用戶端裝置和用戶端裝置元件的問題。

主題

- Greengrass 发现问题
- MQTT 連線問題

Greengrass 发现问题

請使用下列資訊來疑難排解 Greengrass 探索的問題。當用戶端裝置使用 Greengrass 探索 API 來識別可以連線的 Greengrass 核心裝置時，可能會發生這些問題。

主題

- Greengrass 發現問題 (HTTP API)
- Greengrass 發現問題（Python 的 AWS IoT Device SDK v2）
- Greengrass 發現問題（C ++ 的 AWS IoT Device SDK v2）
- Greengrass 發現問題（v2 用於）AWS IoT Device SDK JavaScript
- Greengrass 發現問題（Java 的 AWS IoT Device SDK v2）

Greengrass 發現問題 (HTTP API)

請使用下列資訊來疑難排解 Greengrass 探索的問題。如果您使用 cURL 測試探索 API，可能會看到這些錯誤。

主題

curl: (52) Empty reply from server

如果您在要求中指定非作用中的AWS IoT憑證，可能會看到這個錯誤。

檢查用戶端裝置是否有附加的憑證，以及憑證是否處於作用中狀態。如需詳細資訊，請參閱「將物件或原則附加至用戶端憑證」和「AWS IoT Core開發人員指南」中的「啟用或停用用戶端憑證」。

HTTP 403: {"message":null,"traceId":"a1b2c3d4-5678-90ab-cdef-11111EXAMPLE"}

如果用戶端裝置沒有自行呼叫`greengrass:Discover`的權限，您可能會看到此錯誤。

檢查用戶端裝置的憑證是否具有允許的策略`greengrass:Discover`。您無法針對此權限使用`Resource`區段中的物件原則變數 (`iot:Connection.Thing.*`)。如需詳細資訊，請參閱 探索驗證和授權。

HTTP 404: {"errorMessage":"The thing provided for discovery was not found"}

在下列情況下，您可能會看到此錯誤：

- 用戶端裝置未與任何 Greengrass 核心裝置或群組相關聯。AWS IoT Greengrass V1
- 用戶端裝置關聯的 Greengrass 核心裝置或AWS IoT Greengrass V1群組都沒有 MQTT 代理程式端點。
- 客戶端設備關聯的 Greengrass 核心設備都不會運行客戶端設備身份驗證組件。

檢查用戶端裝置是否與您要連線的核心裝置相關聯。然後，檢查核心設備是否運行客戶端設備身份驗證組件，並且至少具有一個 MQTT 代理端點。如需詳細資訊，請參閱下列內容：

- 關聯用戶端裝置
- 管理核心裝置端點
- 設定雲端探索 (主控台)

Greengrass 發現問題（Python 的 AWS IoT Device SDK v2）

請使用下列資訊，針對 Python 的 AWS IoT Device SDKv2 中的 Greengrass 探索進行疑難排解。

主題

- awscrt.exceptions.AwsCrtError: AWS_ERROR_HTTP_CONNECTION_CLOSED: The connection has closed or is closing.
- awsiot.greengrass_discovery.DiscoveryException: ('Error during discover call: response_code=403', 403)
- awsiot.greengrass_discovery.DiscoveryException: ('Error during discover call: response_code=404', 404)

awscrt.exceptions.AwsCrtError: AWS_ERROR_HTTP_CONNECTION_CLOSED: The connection has closed or is closing.

如果您在要求中指定非作用中的AWS IoT憑證，可能會看到這個錯誤。

檢查用戶端裝置是否有附加的憑證，以及憑證是否處於作用中狀態。如需詳細資訊，請參閱「將物件或原則附加至用戶端憑證」和「AWS IoT Core開發人員指南」中的「啟用或停用用戶端憑證」。

awsiot.greengrass_discovery.DiscoveryException: ('Error during discover call: response_code=403', 403)

如果用戶端裝置沒有自行呼叫`greengrass:Discover`的權限，您可能會看到此錯誤。

檢查用戶端裝置的憑證是否具有允許的策略`greengrass:Discover`。您無法針對此權限使用`Resource`區段中的物件原則變數 (`iot:Connection.Thing.*`)。如需詳細資訊，請參閱 探索驗證和授權。

awsiot.greengrass_discovery.DiscoveryException: ('Error during discover call: response_code=404', 404)

在下列情況下，您可能會看到此錯誤：

- 用戶端裝置未與任何 Greengrass 核心裝置或群組相關聯。AWS IoT Greengrass V1
- 用戶端裝置關聯的 Greengrass 核心裝置或AWS IoT Greengrass V1群組都沒有 MQTT 代理程式端點。
- 客戶端設備關聯的 Greengrass 核心設備都不會運行客戶端設備身份驗證組件。

檢查用戶端裝置是否與您要連線的核心裝置相關聯。然後，檢查核心設備是否運行客戶端設備身份驗證組件，並且至少具有一個 MQTT 代理端點。如需詳細資訊，請參閱下列內容：

- 關聯用戶端裝置
- 管理核心裝置端點
- 設定雲端探索 (主控台)

Greengrass 發現問題（C ++ 的 AWS IoT Device SDK v2）

請使用下列資訊來疑難排解 AWS IoT Device SDKv2 中適用於 C++ 的 Greengrass 探索問題。

主題

- aws-c-http: AWS_ERROR_HTTP_CONNECTION_CLOSED, The connection has closed or is closing.
- aws-c-common: AWS_ERROR_UNKNOWN, Unknown error. (HTTP 403)
- aws-c-common: AWS_ERROR_UNKNOWN, Unknown error. (HTTP 404)

aws-c-http: AWS_ERROR_HTTP_CONNECTION_CLOSED, The connection has closed or is closing.

如果您在要求中指定非作用中的AWS IoT憑證，可能會看到這個錯誤。

檢查用戶端裝置是否有附加的憑證，以及憑證是否處於作用中狀態。如需詳細資訊，請參閱「將物件或原則附加至用戶端憑證」和「AWS IoT Core開發人員指南」中的「啟用或停用用戶端憑證」。

aws-c-common: AWS_ERROR_UNKNOWN, Unknown error. (HTTP 403)

如果用戶端裝置沒有自行呼叫`greengrass:Discover`的權限，您可能會看到此錯誤。

檢查用戶端裝置的憑證是否具有允許的策略`greengrass:Discover`。您無法針對此權限使用`Resource`區段中的物件原則變數 (`iot:Connection.Thing.*`)。如需詳細資訊，請參閱 探索驗證和授權。

aws-c-common: AWS_ERROR_UNKNOWN, Unknown error. (HTTP 404)

在下列情況下，您可能會看到此錯誤：

- 用戶端裝置未與任何 Greengrass 核心裝置或群組相關聯。AWS IoT Greengrass V1
- 用戶端裝置關聯的 Greengrass 核心裝置或AWS IoT Greengrass V1群組都沒有 MQTT 代理程式端點。

- 客戶端設備關聯的 Greengrass 核心設備都不會運行客戶端設備身份驗證組件。

檢查用戶端裝置是否與您要連線的核心裝置相關聯。然後，檢查核心設備是否運行客戶端設備身份驗證組件，並且至少具有一個 MQTT 代理端點。如需詳細資訊，請參閱下列內容：

- 關聯用戶端裝置
- 管理核心裝置端點
- 設定雲端探索 (主控台)

Greengrass 發現問題（v2 用於）AWS IoT Device SDK JavaScript

使用下列資訊來疑難排解 v2 中 Greengrass 探索的AWS IoT Device SDK問題。 JavaScript

主題

- Error: aws-c-http: AWS_ERROR_HTTP_CONNECTION_CLOSED, The connection has closed or is closing.
- Error: Discovery failed (headers: [object Object]) { response_code: 403 }
- Error: Discovery failed (headers: [object Object]) { response_code: 404 }
- Error: Discovery failed (headers: [object Object])

Error: aws-c-http: AWS_ERROR_HTTP_CONNECTION_CLOSED, The connection has closed or is closing.

如果您在要求中指定非作用中的AWS IoT憑證，可能會看到這個錯誤。

檢查用戶端裝置是否有附加的憑證，以及憑證是否處於作用中狀態。如需詳細資訊，請參閱「將物件或原則附加至用戶端憑證」和「AWS IoT Core開發人員指南」中的「啟用或停用用戶端憑證」。

Error: Discovery failed (headers: [object Object]) { response_code: 403 }

如果用戶端裝置沒有自行呼叫`greengrass:Discover`的權限，您可能會看到此錯誤。

檢查用戶端裝置的憑證是否具有允許的策略`greengrass:Discover`。您無法針對此權限使用`Resource`區段中的物件原則變數 (`iot:Connection.Thing.*`)。如需詳細資訊，請參閱 探索驗證和授權。

Error: Discovery failed (headers: [object Object]) { response_code: 404 }

在下列情況下，您可能會看到此錯誤：

- 用戶端裝置未與任何 Greengrass 核心裝置或群組相關聯。AWS IoT Greengrass V1
- 用戶端裝置關聯的 Greengrass 核心裝置或AWS IoT Greengrass V1群組都沒有 MQTT 代理程式端點。
- 客戶端設備關聯的 Greengrass 核心設備都不會運行客戶端設備身份驗證組件。

檢查用戶端裝置是否與您要連線的核心裝置相關聯。然後，檢查核心設備是否運行客戶端設備身份驗證組件，並且至少具有一個 MQTT 代理端點。如需詳細資訊，請參閱下列內容：

- 關聯用戶端裝置
- 管理核心裝置端點
- 設定雲端探索 (主控台)

Error: Discovery failed (headers: [object Object])

當您執行 Greengrass 探索範例時，您可能會看到這個錯誤 (沒有 HTTP 回應碼)。發生此錯誤的原因有多種。

- 如果用戶端裝置沒有自行呼叫`greengrass:Discover`的權限，您可能會看到此錯誤。

  檢查用戶端裝置的憑證是否具有允許的策略`greengrass:Discover`。您無法針對此權限使用`Resource`區段中的物件原則變數 (`iot:Connection.Thing.*`)。如需詳細資訊，請參閱 探索驗證和授權。
- 在下列情況下，您可能會看到此錯誤：
  - 用戶端裝置未與任何 Greengrass 核心裝置或群組相關聯。AWS IoT Greengrass V1
  - 用戶端裝置關聯的 Greengrass 核心裝置或AWS IoT Greengrass V1群組都沒有 MQTT 代理程式端點。
  - 客戶端設備關聯的 Greengrass 核心設備都不會運行客戶端設備身份驗證組件。

  檢查用戶端裝置是否與您要連線的核心裝置相關聯。然後，檢查核心設備是否運行客戶端設備身份驗證組件，並且至少具有一個 MQTT 代理端點。如需詳細資訊，請參閱下列內容：
  - 關聯用戶端裝置
  - 管理核心裝置端點

- 設定雲端探索 (主控台)

Greengrass 發現問題（Java 的 AWS IoT Device SDK v2）

請使用下列資訊，針對 Java AWS IoT Device SDKv2 中的 Greengrass 探索進行疑難排解。

主題

- software.amazon.awssdk.crt.CrtRuntimeException: Error Getting Response Status Code from HttpStream. (aws_last_error: AWS_ERROR_HTTP_DATA_NOT_AVAILABLE(2062), This data is not yet available.)
- java.lang.RuntimeException: Error x-amzn-ErrorType(403)
- java.lang.RuntimeException: Error x-amzn-ErrorType(404)

software.amazon.awssdk.crt.CrtRuntimeException: Error Getting Response Status Code from HttpStream. (aws_last_error: AWS_ERROR_HTTP_DATA_NOT_AVAILABLE(2062), This data is not yet available.)

如果您在要求中指定非作用中的AWS IoT憑證，可能會看到這個錯誤。

檢查用戶端裝置是否有附加的憑證，以及憑證是否處於作用中狀態。如需詳細資訊，請參閱「將物件或原則附加至用戶端憑證」和「AWS IoT Core開發人員指南」中的「啟用或停用用戶端憑證」。

java.lang.RuntimeException: Error x-amzn-ErrorType(403)

如果用戶端裝置沒有自行呼叫`greengrass:Discover`的權限，您可能會看到此錯誤。

檢查用戶端裝置的憑證是否具有允許的策略`greengrass:Discover`。您無法針對此權限使用`Resource`區段中的物件原則變數 (`iot:Connection.Thing.*`)。如需詳細資訊，請參閱 探索驗證和授權。

java.lang.RuntimeException: Error x-amzn-ErrorType(404)

在下列情況下，您可能會看到此錯誤：

- 用戶端裝置未與任何 Greengrass 核心裝置或群組相關聯。AWS IoT Greengrass V1
- 用戶端裝置關聯的 Greengrass 核心裝置或AWS IoT Greengrass V1群組都沒有 MQTT 代理程式端點。

- 客戶端設備關聯的 Greengrass 核心設備都不會運行客戶端設備身份驗證組件。

檢查用戶端裝置是否與您要連線的核心裝置相關聯。然後，檢查核心設備是否運行客戶端設備身份驗證組件，並且至少具有一個 MQTT 代理端點。如需詳細資訊，請參閱下列內容：

- 關聯用戶端裝置
- 管理核心裝置端點
- 設定雲端探索 (主控台)

MQTT 連線問題

請使用下列資訊疑難排解用戶端裝置 MQTT 連線的問題。當用戶端裝置嘗試透過 MQTT 連線至核心裝置時，可能會發生這些問題。

主題

- io.moquette.broker.Authorizator: Client does not have read permissions on the topic
- MQTT 連線問題 (Python)
- MQTT 連線問題 (C++)
- MQTT 連線問題 (Java)
- MQTT 連線問題 () JavaScript

io.moquette.broker.Authorizator: Client does not have read permissions on the topic

當用戶端裝置嘗試訂閱沒有權限的 MQTT 主題時，您可能會在 Greengrass 記錄檔中看到此錯誤。錯誤訊息包含主題。

檢查客戶端設備身份驗證組件的配置是否包括以下內容：

- 符合用戶端裝置的裝置群組。
- 該裝置群組的用戶端裝置授權原則，可授與該主題的`mqtt:subscribe`權限。

如需有關如何部署和設定用戶端裝置驗證元件的詳細資訊，請參閱下列內容：

- 設定雲端探索 (主控台)
- 用戶端裝置驗證

- 建立部署

MQTT 連線問題 (Python)

當您將 AWS IoT Device SDKv2 用於 Python 時，請使用下列資訊來疑難排解用戶端裝置 MQTT 連線的問題。

主題

- AWS_ERROR_MQTT_PROTOCOL_ERROR: Protocol error occurred
- AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

AWS_ERROR_MQTT_PROTOCOL_ERROR: Protocol error occurred

如果用戶端裝置驗證元件未定義授與用戶端裝置連線權限的用戶端裝置授權原則，您可能會看到此錯誤。

檢查客戶端設備身份驗證組件的配置是否包括以下內容：

- 符合用戶端裝置的裝置群組。
- 該裝置群組的用戶端裝置授權原則，`mqtt:connect`可授與用戶端裝置的權限。

如需有關如何部署和設定用戶端裝置驗證元件的詳細資訊，請參閱下列內容：

- 設定雲端探索 (主控台)
- 用戶端裝置驗證
- 建立部署

AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

如果用戶端裝置驗證元件未定義授與用戶端裝置連線權限的用戶端裝置授權原則，您可能會看到此錯誤。

檢查客戶端設備身份驗證組件的配置是否包括以下內容：

- 符合用戶端裝置的裝置群組。
- 該裝置群組的用戶端裝置授權原則，`mqtt:connect`可授與用戶端裝置的權限。

如需有關如何部署和設定用戶端裝置驗證元件的詳細資訊，請參閱下列內容：

- 設定雲端探索 (主控台)
- 用戶端裝置驗證
- 建立部署

MQTT 連線問題 (C++)

使用 AWS IoT Device SDKv2 進行 C++ 時，請使用下列資訊疑難排解用戶端裝置 MQTT 連線的問題。

主題

- AWS_ERROR_MQTT_PROTOCOL_ERROR: Protocol error occurred
- AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

AWS_ERROR_MQTT_PROTOCOL_ERROR: Protocol error occurred

如果用戶端裝置驗證元件未定義授與用戶端裝置連線權限的用戶端裝置授權原則，您可能會看到此錯誤。

檢查客戶端設備身份驗證組件的配置是否包括以下內容：

- 符合用戶端裝置的裝置群組。
- 該裝置群組的用戶端裝置授權原則，`mqtt:connect`可授與用戶端裝置的權限。

如需有關如何部署和設定用戶端裝置驗證元件的詳細資訊，請參閱下列內容：

- 設定雲端探索 (主控台)
- 用戶端裝置驗證
- 建立部署

AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

如果用戶端裝置驗證元件未定義授與用戶端裝置連線權限的用戶端裝置授權原則，您可能會看到此錯誤。

檢查客戶端設備身份驗證組件的配置是否包括以下內容：

- 符合用戶端裝置的裝置群組。
- 該裝置群組的用戶端裝置授權原則，`mqtt:connect`可授與用戶端裝置的權限。

如需有關如何部署和設定用戶端裝置驗證元件的詳細資訊，請參閱下列內容：

- 設定雲端探索 (主控台)
- 用戶端裝置驗證
- 建立部署

MQTT 連線問題 (Java)

當您將 AWS IoT Device SDKv2 用於 Java 時，請使用下列資訊來疑難排解用戶端裝置 MQTT 連線的問題。

主題

- software.amazon.awssdk.crt.mqtt.MqttException: Protocol error occurred
- AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

software.amazon.awssdk.crt.mqtt.MqttException: Protocol error occurred

如果用戶端裝置驗證元件未定義授與用戶端裝置連線權限的用戶端裝置授權原則，您可能會看到此錯誤。

檢查客戶端設備身份驗證組件的配置是否包括以下內容：

- 符合用戶端裝置的裝置群組。
- 該裝置群組的用戶端裝置授權原則，`mqtt:connect`可授與用戶端裝置的權限。

如需有關如何部署和設定用戶端裝置驗證元件的詳細資訊，請參閱下列內容：

- 設定雲端探索 (主控台)
- 用戶端裝置驗證
- 建立部署

AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

如果用戶端裝置驗證元件未定義授與用戶端裝置連線權限的用戶端裝置授權原則，您可能會看到此錯誤。

檢查客戶端設備身份驗證組件的配置是否包括以下內容：

- 符合用戶端裝置的裝置群組。
- 該裝置群組的用戶端裝置授權原則，`mqtt:connect`可授與用戶端裝置的權限。

如需有關如何部署和設定用戶端裝置驗證元件的詳細資訊，請參閱下列內容：

- 設定雲端探索 (主控台)
- 用戶端裝置驗證
- 建立部署

MQTT 連線問題 () JavaScript

使用下列資訊來疑難排解使用 AWS IoT Device SDKv2 時，用戶端裝置 MQTT 連線的問題。JavaScript

主題

- AWS_ERROR_MQTT_PROTOCOL_ERROR: Protocol error occurred
- AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

AWS_ERROR_MQTT_PROTOCOL_ERROR: Protocol error occurred

如果用戶端裝置驗證元件未定義授與用戶端裝置連線權限的用戶端裝置授權原則，您可能會看到此錯誤。

檢查客戶端設備身份驗證組件的配置是否包括以下內容：

- 符合用戶端裝置的裝置群組。
- 該裝置群組的用戶端裝置授權原則，`mqtt:connect`可授與用戶端裝置的權限。

如需有關如何部署和設定用戶端裝置驗證元件的詳細資訊，請參閱下列內容：

- 設定雲端探索 (主控台)

- 用戶端裝置驗證
- 建立部署

AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

如果用戶端裝置驗證元件未定義授與用戶端裝置連線權限的用戶端裝置授權原則，您可能會看到此錯誤。

檢查客戶端設備身份驗證組件的配置是否包括以下內容：

- 符合用戶端裝置的裝置群組。
- 該裝置群組的用戶端裝置授權原則，`mqtt:connect`可授與用戶端裝置的權限。

如需有關如何部署和設定用戶端裝置驗證元件的詳細資訊，請參閱下列內容：

- 設定雲端探索 (主控台)
- 用戶端裝置驗證
- 建立部署

與裝置陰影互動

Greengrass 核心設備可以使用組件與設AWS IoT備陰影進行交互。陰影是儲存物件目前或所需狀態資訊的 JSON 文AWS IoT件。陰影可以使設備的狀態可用於其他AWS IoT Greengrass組件，無論設備是AWS IoT否連接。每個AWS IoT設備都有自己的經典，未命名的陰影。您也可以為每個裝置建立多個具名陰影。

裝置和服務可以建立、更新和刪除雲端陰影使用 MQTT 和保留的 MQTT 陰影主題、使用 Device Shadow REST API 的 HTTP，以及. AWS CLIAWS IoT

陰影管理員元件可讓您的 Greengrass 元件使用本機陰影服務和本機發行/訂閱陰影主題來建立、更新和刪除本機陰影。shadow Manager 也會管理核心裝置上這些本機陰影文件的儲存空間，並處理陰影狀態資訊與雲端陰影的同步處理。

您也可以使用陰影管理員元件來管理連線至核心裝置的用戶端裝置的本機陰影。若要讓陰影管理員管理用戶端裝置陰影，您可以將 MQTT 橋接器元件設定為在本機 MQTT 代理程式和本機發佈/訂閱服務之間轉送訊息。如需詳細資訊，請參閱 與用戶端裝置陰影互動並同步。

如需有關 AWS IoT Device Shadow 概念的詳細資訊，請參閱AWS IoT開發人員指南中的AWS IoT裝置陰影服務。

主題

- 與組件中的陰影互動
- 同步本地設備陰影 AWS IoT Core

與組件中的陰影互動

您可以開發使用本機陰影服務讀取和修改本機陰影文件和用戶端裝置陰影文件的自訂元件 (包括 Lambda 函數元件)。

自訂元件使用中的 AWS IoT Greengrass Core IPC 程式庫與本機陰影服務互動AWS IoT Device SDK。陰影管理員元件會在核心裝置上啟用本機陰影服務。

若要將陰影管理員元件部署到 Greengrass 核心裝置，請建立包含該元件的部署。`aws.greengrass.ShadowManager`

> **Note**
>
> 根據預設，部署陰影管理員元件只會啟用本機陰影作業。若AWS IoT Greengrass要啟用將核心裝置陰影或用戶端裝置的任何陰影的陰影狀態資訊同步至中對應的雲端陰影文件AWS IoT Core，您必須為包含`synchronize`參數的陰影管理員元件建立組態更新。如需詳細資訊，請參閱 同步本地設備陰影 AWS IoT Core。

主題

擷取和修改陰影狀態

陰影 IPC 作業會擷取並更新本機陰影文件中的狀態資訊。陰影管理員元件會處理核心裝置上這些陰影文件的儲存。

修改局部陰影狀態

1. 將授權原則新增至自訂元件的方案，以允許元件接收有關本機陰影主題的訊息。

   如需授權原則範例，請參閱本機陰影 IPC 授權原則範例。
2. 使用陰影 IPC 作業擷取和修改陰影狀態資訊。如需在元件程式碼中使用陰影 IPC 作業的更多資訊，請參閱〈與局部陰影互動。

> **Note**
>
> 若要讓核心裝置與用戶端裝置陰影互動，您還必須設定和部署 MQTT 橋接器元件。如需詳細資訊，請參閱啟用陰影管理員與用戶端裝置通訊。

對陰影狀態變化做出反應

Greengrass 組件使用本地發布/訂閱接口在核心設備上進行通信。若要讓自訂元件回應陰影狀態變更，您可以訂閱本機發佈/訂閱主題。這可讓元件接收有關本機陰影主題的訊息，然後對這些訊息採取行動。

本機陰影主題使用與AWS IoT裝置陰影 MQTT 主題相同的格式。如需陰影主題的詳細資訊，請參閱AWS IoT開發人員指南中的 Device Shadow MQTT 主題。

對局部陰影狀態變化做出反應

1. 將存取控制原則新增至自訂元件的方案，以允許元件接收有關本機陰影主題的訊息。

   如需授權原則範例，請參閱本機陰影 IPC 授權原則範例。
2. 若要在元件中啟動自訂動作，請使用 `SubscribeToTopic` IPC 作業來訂閱您要接收訊息的陰影主題。如需有關在元件程式碼中使用本機發佈/訂閱 IPC 作業的詳細資訊，請參閱。發佈/訂閱本地訊息
3. 若要叫用 Lambda 函數，請使用事件來源組態提供陰影主題的名稱，並指定它是本機發佈/訂閱主題。如需建立 Lambda 函數元件的相關資訊，請參閱執行AWS Lambda函數。

> **Note**
>
> 若要讓核心裝置與用戶端裝置陰影互動，您還必須設定和部署 MQTT 橋接器元件。如需詳細資訊，請參閱啟用陰影管理員與用戶端裝置通訊。

同步本地設備陰影 AWS IoT Core

陰影管理員元件可AWS IoT Greengrass讓您同步本機裝置陰影狀態與AWS IoT Core。您必須修改陰影管理員元件的組態以包含`synchronization`組態參數，並指定裝置的AWS IoT物件名稱，以及要同步的陰影。

當您設定陰影管理員同步陰影時，無論變更是發生在本機陰影文件還是雲陰影文件中，都會同步指定陰影的所有狀態變更。

您也可以指定陰影管理員元件是即時或定期同步陰影。根據預設，陰影管理員元件會即時同步陰影，因此核心裝置會在每次更新AWS IoT Core時傳送和接收陰影更新。您可以設定定期間隔，以減少頻寬使用量和費用。

主題

- 必要條件
- 設定陰影管理員元件
- 同步局部陰影

- 陰影合併衝突行為

必要條件

若要同步本機陰影AWS IoT Core，您必須將 Greengrass 核心裝置的原則設定為允許下列AWS IoT Core陰影AWS IoT原則動作。

- `iot:GetThingShadow`
- `iot:UpdateThingShadow`
- `iot:DeleteThingShadow`

如需詳細資訊，請參閱下列內容：

- AWS IoT CoreAWS IoT開發人員指南中的政策動作
- AWS IoT Greengrass V2核心裝置的最低AWS IoT原則
- 更新核心裝置的AWS IoT政策

設定陰影管理員元件

陰影管理員需要陰影名稱對映清單，才能將本機陰影文件中的陰影狀態資訊同步到中的雲端陰影文件AWS IoT Core。

若要同步陰影狀態，請建立包含`aws.greengrass.ShadowManager`元件的部署，然後在部署的陰影管理員`synchronize`組態中的組態參數中指定要同步的陰影。

> **Note**
>
> 若要讓核心裝置與用戶端裝置陰影互動，您還必須設定和部署 MQTT 橋接器元件。如需詳細資訊，請參閱啟用陰影管理員與用戶端裝置通訊。

下列範例組態更新會指示陰影管理員元件與AWS IoT Core下列陰影同步：

- 核心裝置的經典陰影
- `MyCoreShadow`為核心裝置命名
- IoT 物件的經典陰影 `MyDevice2`

- 命名的陰影MyShadowA和名MyShadowB為 IoT 的東西 MyDevice1

此組態更新指定要即時同步陰影。AWS IoT Core如果您使用陰影管理員 v2.1.0 或更新版本，您可以設定陰影管理員元件以定期間隔同步陰影。若要設定此功能，請將同步策略變更為periodic，並以秒為單delay位指定間隔。如需詳細資訊，請參閱陰影管理員元件的策略組態參數。

此配置更新指定在和核心設備之間AWS IoT Core雙向同步陰影。如果您使用陰影管理員 v2.2.0 或更新版本，您可以將陰影管理員元件設定為僅在一個方向同步陰影。若要設定此功能，請將同步變更direction為deviceToCloud或cloudToDevice。如需詳細資訊，請參閱陰影管理員元件的方向組態參數。

```
{
  "strategy": {
    "type": "realTime"
  },
  "synchronize": {
    "coreThing": {
      "classic": true,
      "namedShadows": [
        "MyCoreShadow"
      ]
    },
    "shadowDocuments": [
      {
        "thingName": "MyDevice1",
        "classic": false,
        "namedShadows": [
          "MyShadowA",
          "MyShadowB"
        ]
      },
      {
        "thingName": "MyDevice2",
        "classic": true,
        "namedShadows": [ ]
      }
    ],
    "direction": "betweenDeviceAndCloud"
  }
}
```

同步局部陰影

當 Greengrass 核心裝置連線到AWS IoT雲端時，陰影管理員會針對您在元件組態中指定的陰影執行下列工作。行為取決於您指定的陰影同步方向組態選項。根據預設，陰影管理員會使用`betweenDeviceAndCloud`此選項來同步兩個方向的陰影。如果您使用陰影管理器 v2.2.0 或更新版本，則可以將核心設備配置為僅在一個方向上同步陰影，可以是`cloudToDevice`或。`deviceToCloud`

- 如果陰影同步方向設定為`betweenDeviceAndCloud`或`cloudToDevice`，陰影管理員會從中的雲陰影文件擷取報告的狀態資訊AWS IoT Core。然後，它會更新本機儲存的陰影文件，以同步處理裝置狀態。
- 如果陰影同步方向設定為`betweenDeviceAndCloud`或`deviceToCloud`，陰影管理員會將裝置的目前狀態發佈至雲端陰影文件。

陰影合併衝突行為

在某些情況下，例如核心裝置與網際網路中斷連線時，陰影管理員在陰影管理員同步處理變更之前，本機陰影服務和AWS IoT雲端中的陰影可能會變更。因此，本機陰影服務和AWS IoT雲端之間所需和報告的狀態不同

當陰影管理員同步處理陰影時，會根據下列行為合併變更：

- 如果您使用 v2.2.0 之前的陰影管理員版本，或指定`betweenDeviceAndCloud`陰影同步方向時，會套用下列行為：
  - 當陰影的所需狀態下發生合併衝突時，陰影管理員會以AWS IoT雲端的值覆寫本機陰影文件的衝突區段。
  - 當陰影的報告狀態中發生合併衝突時，陰影管理員會以本機陰影文件的值覆寫AWS IoT雲端中衝突的陰影區段。
- 當您指定`deviceToCloud`陰影同步方向時，陰影管理員會以本機陰影文件的值覆寫AWS IoT雲端中衝突的陰影區段。
- 當您指定`cloudToDevice`陰影同步方向時，陰影管理員會以AWS IoT雲端的值覆寫本機陰影文件的衝突區段。

管理核心裝 Greengrass 資料串流

AWS IoT Greengrass串流管理員可讓您更有效率且可靠地將大量 IoT 資料傳輸到AWS 雲端. 流管理器在AWS IoT Greengrass核心上處理數據流，然後再將其導出到AWS 雲端. 串流管理員與常見的邊緣案例整合，例如機器學習 (ML) 推論，AWS IoT GreengrassCore 裝置會在將資料匯出至AWS 雲端或本機儲存目的地之前處理和分析資料。

串流管理員提供通用介面，可簡化自訂元件開發，因此您不需要建置自訂串流管理功能。您的元件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原則。您可以為每個串流定義儲存區類型、大小和資料保留的原則，以控制串流管理員處理和匯出資料的方式。

流管理器在具有間歇性或有限連接的環境中工作。您可以定義頻寬使用、逾時行為，以及 AWS IoT Greengrass Core 在連線或中斷連線時如何處理串流資料。您也可以設定優先順序，以控制 AWS IoT Greengrass Core 將串流匯出至AWS 雲端. 這使您可以比其他數據更快地處理關鍵數據。

您可以設定串流管理員自動將資料匯AWS 雲端出至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，以協助制定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？。
- 亞馬遜 Kinesis 資料串流中的串流。您可以使用 Kinesis Data Streams 來彙總大量資料，並將其載入資料倉儲或 MapReduce 叢集。如需詳細資訊，請參閱《Amazon Kinesis Data Streams 開發人員指南》中的什麼是 Amazon Kinesis Data Streams？。
- 中的資產性質AWS IoT SiteWise。 AWS IoT SiteWise可讓您大規模收集、組織和分析來自工業設備的資料。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 簡單存儲服務 Amazon S3 中的對象。您可以使用 Amazon S3 存放和擷取大量資料。如需詳細資訊，請參閱什麼是 Amazon S3？ 在 Amazon 簡單存儲服務開發人員指南中。

串流管理工作流程

您的 IoT 應用程式會透過串流管理員 SDK 與串流管理員互動。

在簡單的工作流程中，AWS IoT Greengrass核心上的元件會消耗 IoT 資料，例如時間序列溫度和壓力指標。元件可能會篩選或壓縮資料，然後呼叫串流管理員 SDK，將資料寫入串流管理員中的串流。串

流管理員可以根據您為串流定義的原則，AWS 雲端自動將串流匯出至該串流。組件還可以將數據直接發送到本地數據庫或存儲庫。

您的 IoT 應用程式可以包含多個讀取或寫入串流的自訂元件。這些元件可以讀取和寫入串流，以篩選、彙總和分析AWS IoT Greengrass核心裝置上的資料。這使得可以快速響應本地事件，並在將數據從核心傳輸到AWS 雲端或本地目的地之前提取有價值的信息。

若要開始使用，請將串流管理員元件部署到您的AWS IoT Greengrass核心裝置。在部署中，設定串流管理員元件參數，以定義適用於 Greengrass 核心裝置上所有串流的設定。使用這些參數可根據您的業務需求和環境限制來控制串流管理員儲存、處理和匯出串流的方式。

設定串流管理員之後，您可以建立和部署 IoT 應用程式。這些通常是在串流管理員 SDK `StreamManagerClient` 中用來建立串流並與其互動的自訂元件。建立串流時，您可以定義每個串流原則，例如匯出目的地、優先順序和持續性。

要求

下列需求適用於使用串流管理員：

- 串流管理員除了AWS IoT Greengrass核心軟體之外，至少需要 70 MB RAM。您的總記憶體需求視您的工作負載而定。
- AWS IoT Greengrass組件必須使用串流管理員 SDK 與串流管理員互動。串流管理員 SDK 提供下列語言版本：
  - 適用於 Java 的流管理器 SDK（v1.1.0 或更高版本）
  - 適用於 Node.js 的串流管理員 SDK (v1.1.0 或更新版本)
  - 適用於 Python 的流管理器開發套件（1.1.0 或更高版本）
- AWS IoT Greengrass組件必須將流管理器組件（`aws.greengrass.StreamManager`）指定為其使用流管理器的配方中的依賴項。

  > **Note**
  >
  > 如果您使用串流管理員將資料匯出至雲端，則無法將串流管理員元件的 2.0.7 版升級為 v2.0.8 和 v2.0.11 之間的版本。如果您是第一次部署串流管理員，強烈建議您部署最新版本的串流管理員元件。

- 如果您為串流定義AWS 雲端匯出目的地，則必須建立匯出目標並授與 Greengrass 裝置角色中的存取權限。視目的地而定，也可能適用其他需求。如需詳細資訊，請參閱：

- the section called “AWS IoT Analytics 頻道”
- the section called “Amazon Kinesis 數據流”
- the section called “AWS IoT SiteWise資產性質”
- the section called “Amazon S3 對象”

您有責任維護這些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不會加密核心裝置上本機元件之間靜態或傳輸中的串流資料。

- 靜態資料。串流資料以本機之方式儲存在儲存目錄中。為了確保資料安全性，AWS IoT Greengrass必須依賴檔案權限和全磁碟加密 (如果已啟用)。您可以使用選用的 STREAM_MANAGER_STORE_ROOT_DIR 參數來指定儲存目錄。如果您稍後變更此參數以使用不同的儲存目錄，則 AWS IoT Greengrass 不會刪除之前的儲存目錄或其內容。
- 本機傳輸中的資料。 AWS IoT Greengrass在資料來源、AWS IoT Greengrass元件、串流管理員 SDK 和串流管理員之間的本機傳輸中，不會加密串流資料。
- 傳輸中的資料AWS 雲端。由串流管理員匯出至AWS 雲端使用標準AWS服務用戶端加密與傳輸層安全性 (TLS) 的資料流。

用戶端身分驗證

串流管理員用戶端使用串流管理員 SDK 與串流管理員進行通訊。啟用客戶端身份驗證時，只有 Greengrass 組件可以與流管理器中的流進行交互。停用用戶端驗證時，Greengrass 核心裝置上執行的任何程序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。當您將串流管理員元件部署到核心裝置時，可以設定此參數。

	已啟用	已停用
參數值	`true` (預設和建議)	`false`

	已啟用	已停用
允許的用戶端	核心設備上的 Greengrass 組件	核心設備上的 Greengrass 組件 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 “用 StreamManagerClient 於使用串流”
- the section called “匯出支援雲端目的地的組態”

建立使用串流管理員的自訂元件

在自訂 Greengrass 元件中使用串流管理員來儲存、處理和匯出 IoT 裝置資料。使用本節中的程序和範例，建立可搭配串流管理員使用的元件方法、成品和應用程式。如需如何開發和測試元件的詳細資訊，請參閱建立 AWS IoT Greengrass 元件。

主題

- 定義使用流管理器的組件配方
- 在應用程式程式碼中 Connect 至串流管

定義使用流管理器的組件配方

若要在自訂元件中使用串流管理員，您必須將`aws.greengrass.StreamManager`元件定義為相依性。您也必須提供串流管理員 SDK。完成下列工作，以您選擇的語言下載並使用串流管理員 SDK。

使用適用於 Java 的串流管理員 SDK

適用於 Java 的串流管理員 SDK 可做為 JAR 檔案使用，供您用來編譯元件。然後，您可以建立包含串流管理員 SDK 的應用程式 JAR、將應用程式 JAR 定義為元件加工品，並在元件生命週期中執行應用程式 JAR。

若要使用適用於 Java 的串流管理員 SDK

1. 下載適用於 Java JAR 檔案的串流管理員 SDK。
2. 執行下列其中一項動作，從 Java 應用程式和串流管理員 SDK JAR 檔案建立元件成品：
   - 將應用程式建置為包含串流管理員 SDK JAR 的 JAR 檔案，並在元件方案中執行此 JAR 檔案。
   - 將串流管理員 SDK JAR 定義為元件加工品。當您在元件方案中執行應用程式時，將該成品新增至類別路徑。

   您的組件配方可能如以下範例所示。此元件會執行 StreamManagerS3.java 範例的修改版本，其中`StreamManagerS3.jar`包含串流管理員 SDK JAR。

   JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.StreamManagerS3Java",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Uses stream manager to upload a file to an S3
 bucket.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.StreamManager": {
      "VersionRequirement": "^2.0.0"
    }
  },
  "Manifests": [
    {
      "Lifecycle": {
        "run": "java -jar {artifacts:path}/StreamManagerS3.jar"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Java/1.0.0/StreamManagerS3.jar"
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.StreamManagerS3Java
ComponentVersion: 1.0.0
ComponentDescription: Uses stream manager to upload a file to an S3 bucket.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.StreamManager:
    VersionRequirement: "^2.0.0"
Manifests:
  - Lifecycle:
      run: java -jar {artifacts:path}/StreamManagerS3.jar
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Java/1.0.0/StreamManagerS3.jar
```

如需如何開發和測試元件的詳細資訊，請參閱建立 AWS IoT Greengrass 元件。

使用適用於 Python 的串流管理員 SDK

適用於 Python 的串流管理員 SDK 可作為原始程式碼使用，您可以將其包含在元件中。建立串流管理員 SDK 的 ZIP 檔案、將 ZIP 檔案定義為元件加工品，然後在元件生命週期中安裝 SDK 的需求。

若要使用適用於 Python 的串流管理員 SDK

1. 克隆或下載 aws-greengrass-stream-manager-sdk- 蟒蛇存儲庫。

   ```
   git clone git@github.com:aws-greengrass/aws-greengrass-stream-manager-sdk-
   python.git
   ```

2. 建立包含`stream_manager`資料夾的 ZIP 檔案，其中包含適用於 Python 的串流管理員 SDK 的原始程式碼。您可以提供此 ZIP 檔案做為AWS IoT Greengrass核心軟體在安裝元件時解壓縮的元件加工品。請執行下列動作：

   a. 開啟包含您在上一個步驟中下載的存放庫的資料。

      ```
      cd aws-greengrass-stream-manager-sdk-python
      ```

b. 將stream_manager資料夾壓縮到名為的 ZIP 檔案中stream_manager_sdk.zip。

Linux or Unix

```
zip -rv stream_manager_sdk.zip stream_manager
```

Windows Command Prompt (CMD)

```
tar -acvf stream_manager_sdk.zip stream_manager
```

PowerShell

```
Compress-Archive stream_manager stream_manager_sdk.zip
```

c. 確認stream_manager_sdk.zip檔案包含資stream_manager料夾及其內容。執行下列命令，列出 ZIP 檔案的內容。

Linux or Unix

```
unzip -l stream_manager_sdk.zip
```

Windows Command Prompt (CMD)

```
tar -tf stream_manager_sdk.zip
```

輸出應看起來如下列內容。

```
Archive:  aws-greengrass-stream-manager-sdk-python/stream_manager.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
        0  02-24-2021 20:45   stream_manager/
      913  02-24-2021 20:45   stream_manager/__init__.py
     9719  02-24-2021 20:45   stream_manager/utilinternal.py
     1412  02-24-2021 20:45   stream_manager/exceptions.py
     1004  02-24-2021 20:45   stream_manager/util.py
        0  02-24-2021 20:45   stream_manager/data/
   254463  02-24-2021 20:45   stream_manager/data/__init__.py
    26515  02-24-2021 20:45   stream_manager/streammanagerclient.py
---------                     -------
```

```
    294026                    8 files
```

3. 將串流管理員 SDK 成品複製到元件的成品資料夾。除了串流管理員 SDK ZIP 檔案之外，您的元件還會使用 SDK 的`requirements.txt`檔案來安裝串流管理員 SDK 的相依性。將 *~/greengrass ###############*夾的路徑。

   Linux or Unix

   ```
   cp {stream_manager_sdk.zip,requirements.txt} ~/greengrass-components/artifacts/
   com.example.StreamManagerS3Python/1.0.0/
   ```

   Windows Command Prompt (CMD)

   ```
   robocopy . %USERPROFILE%\greengrass-components\artifacts
   \com.example.StreamManagerS3Python\1.0.0 stream_manager_sdk.zip
   robocopy . %USERPROFILE%\greengrass-components\artifacts
   \com.example.StreamManagerS3Python\1.0.0 requirements.txt
   ```

   PowerShell

   ```
   cp .\stream_manager_sdk.zip,.\requirements.txt ~\greengrass-components\artifacts
   \com.example.StreamManagerS3Python\1.0.0\
   ```

4. 建立您的元件配方。在配方中，請執行下列操作：

   a. 定義`stream_manager_sdk.zip`並`requirements.txt`作為人工因素。

   b. 將您的 Python 應用程式定義為成品。

   c. 在安裝生命週期中，從中安裝串流管理員 SDK 需求`requirements.txt`。

   d. 在執行生命週期中，將串流管理員 SDK 附加至`PYTHONPATH`，然後執行您的 Python 應用程式。

   您的組件配方可能如以下範例所示。此元件會執行 stream_manager_s3.py 範例。

   JSON

   ```
   {
     "RecipeFormatVersion": "2020-01-25",
     "ComponentName": "com.example.StreamManagerS3Python",
     "ComponentVersion": "1.0.0",
   ```

```
  "ComponentDescription": "Uses stream manager to upload a file to an S3
 bucket.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.StreamManager": {
      "VersionRequirement": "^2.0.0"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "pip3 install --user -r {artifacts:path}/requirements.txt",
        "run": "export PYTHONPATH=$PYTHONPATH:{artifacts:decompressedPath}/
stream_manager_sdk; python3 {artifacts:path}/stream_manager_s3.py"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_sdk.zip",
          "Unarchive": "ZIP"
        },
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_s3.py"
        },
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/requirements.txt"
        }
      ]
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "pip3 install --user -r {artifacts:path}/requirements.txt",
        "run": "set \"PYTHONPATH=%PYTHONPATH%;{artifacts:decompressedPath}/
stream_manager_sdk\" & py -3 {artifacts:path}/stream_manager_s3.py"
      },
      "Artifacts": [
```

```
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_sdk.zip",
          "Unarchive": "ZIP"
        },
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_s3.py"
        },
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/requirements.txt"
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.StreamManagerS3Python
ComponentVersion: 1.0.0
ComponentDescription: Uses stream manager to upload a file to an S3 bucket.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.StreamManager:
    VersionRequirement: "^2.0.0"
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: pip3 install --user -r {artifacts:path}/requirements.txt
      run: |
        export PYTHONPATH=$PYTHONPATH:{artifacts:decompressedPath}/
stream_manager_sdk
        python3 {artifacts:path}/stream_manager_s3.py
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_sdk.zip
        Unarchive: ZIP
```

```
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_s3.py
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/requirements.txt
  - Platform:
      os: windows
    Lifecycle:
      install: pip3 install --user -r {artifacts:path}/requirements.txt
      run: |
        set "PYTHONPATH=%PYTHONPATH%;{artifacts:decompressedPath}/
stream_manager_sdk"
        py -3 {artifacts:path}/stream_manager_s3.py
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_sdk.zip
        Unarchive: ZIP
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_s3.py
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/requirements.txt
```

如需如何開發和測試元件的詳細資訊，請參閱建立 AWS IoT Greengrass 元件。

使用串流管理員 SDK JavaScript

的串流管理員 SDK 可做為原始程式碼使用，您可以將其包含在元件中。JavaScript建立串流管理員 SDK 的 ZIP 檔案、將 ZIP 檔案定義為元件加工品，然後在元件生命週期中安裝 SDK。

若要使用串流管理員 SDK JavaScript

1. 克隆或下載 aws-greengrass-stream-manager-sdk- js 存儲庫。

   ```
   git clone git@github.com:aws-greengrass/aws-greengrass-stream-manager-sdk-js.git
   ```

2. 建立包含`aws-greengrass-stream-manager-sdk`資料夾的 ZIP 檔案，其中包含的串流管理員 SDK 的原始程式碼JavaScript。您可以提供此 ZIP 檔案做為AWS IoT Greengrass核心軟體在安裝元件時解壓縮的元件加工品。請執行下列動作：

   a. 開啟包含您在上一個步驟中下載的存放庫的資料。

```
cd aws-greengrass-stream-manager-sdk-js
```

b. 將aws-greengrass-stream-manager-sdk資料夾壓縮到名為的 ZIP 檔案中stream-manager-sdk.zip。

Linux or Unix

```
zip -rv stream-manager-sdk.zip aws-greengrass-stream-manager-sdk
```

Windows Command Prompt (CMD)

```
tar -acvf stream-manager-sdk.zip aws-greengrass-stream-manager-sdk
```

PowerShell

```
Compress-Archive aws-greengrass-stream-manager-sdk stream-manager-sdk.zip
```

c. 確認stream-manager-sdk.zip檔案包含資aws-greengrass-stream-manager-sdk料夾及其內容。執行下列命令，列出 ZIP 檔案的內容。

Linux or Unix

```
unzip -l stream-manager-sdk.zip
```

Windows Command Prompt (CMD)

```
tar -tf stream-manager-sdk.zip
```

輸出應看起來如下列內容。

```
Archive:  stream-manager-sdk.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
        0  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/
      369  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/package.json
     1017  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/util.js
     8374  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/utilInternal.js
     1937  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/exceptions.js
```

```
         0  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/data/
    353343  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/data/index.js
     22599  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/client.js
       216  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/index.js
---------                      -------
    387855                     9 files
```

3. 將串流管理員 SDK 成品複製到元件的成品資料夾。將 *~/greengrass ###############*夾的路徑。

   Linux or Unix

   ```
   cp stream-manager-sdk.zip ~/greengrass-components/artifacts/
   com.example.StreamManagerS3JS/1.0.0/
   ```

   Windows Command Prompt (CMD)

   ```
   robocopy . %USERPROFILE%\greengrass-components\artifacts
   \com.example.StreamManagerS3JS\1.0.0 stream-manager-sdk.zip
   ```

   PowerShell

   ```
   cp .\stream-manager-sdk.zip ~\greengrass-components\artifacts
   \com.example.StreamManagerS3JS\1.0.0\
   ```

4. 建立您的元件配方。在配方中，請執行下列操作：

   a. 定義`stream-manager-sdk.zip`為人工因素。

   b. 將您的JavaScript應用程式定義為成品。

   c. 在安裝生命週期中，從`stream-manager-sdk.zip`成品安裝串流管理員 SDK。這個`npm install`命令會建立包`node_modules`含串流管理員 SDK 及其相依性的資料夾。

   d. 在執行生命週期中，將`node_modules`資料夾附加至`NODE_PATH`，然後執行您的JavaScript應用程式。

   您的組件配方可能如以下範例所示。此元件會執行 StreamManagerS3 範例。

   JSON

   ```
   {
   ```

```
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.StreamManagerS3JS",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Uses stream manager to upload a file to an S3
 bucket.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.StreamManager": {
      "VersionRequirement": "^2.0.0"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "npm install {artifacts:decompressedPath}/stream-manager-sdk/
aws-greengrass-stream-manager-sdk",
        "run": "export NODE_PATH=$NODE_PATH:{work:path}/node_modules; node
 {artifacts:path}/index.js"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/stream-manager-sdk.zip",
          "Unarchive": "ZIP"
        },
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/index.js"
        }
      ]
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "npm install {artifacts:decompressedPath}/stream-manager-sdk/
aws-greengrass-stream-manager-sdk",
        "run": "set \"NODE_PATH=%NODE_PATH%;{work:path}/node_modules\" & node
 {artifacts:path}/index.js"
      },
```

```
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/stream-manager-sdk.zip",
          "Unarchive": "ZIP"
        },
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/index.js"
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.StreamManagerS3JS
ComponentVersion: 1.0.0
ComponentDescription: Uses stream manager to upload a file to an S3 bucket.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.StreamManager:
    VersionRequirement: "^2.0.0"
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: npm install {artifacts:decompressedPath}/stream-manager-sdk/aws-
greengrass-stream-manager-sdk
      run: |
        export NODE_PATH=$NODE_PATH:{work:path}/node_modules
        node {artifacts:path}/index.js
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/stream-manager-sdk.zip
        Unarchive: ZIP
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/index.js
  - Platform:
      os: windows
```

```
    Lifecycle:
      install: npm install {artifacts:decompressedPath}/stream-manager-sdk/aws-
greengrass-stream-manager-sdk
      run: |
        set "NODE_PATH=%NODE_PATH%;{work:path}/node_modules"
        node {artifacts:path}/index.js
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/stream-manager-sdk.zip
        Unarchive: ZIP
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/index.js
```

如需如何開發和測試元件的詳細資訊，請參閱建立 AWS IoT Greengrass 元件。

在應用程式程式碼中 Connect 至串流管

若要連線至應用程式中的串流管理員，請`StreamManagerClient`從串流管理員 SDK 建立的執行個體。此用戶端會連線至串流管理員元件的預設連接埠 8088 或您指定的連接埠。如需如何在建立執行個體`StreamManagerClient`之後使用的詳細資訊，請參閱用 StreamManagerClient 於使用串流。

Example 範例：使用預設 Connect 埠連線至串流管理員

Java

```
import com.amazonaws.greengrass.streammanager.client.StreamManagerClient;

public class MyStreamManagerComponent {

    void connectToStreamManagerWithDefaultPort() {
        StreamManagerClient client = StreamManagerClientFactory.standard().build();

        // Use the client.
    }
}
```

Python

```
from stream_manager import (
    StreamManagerClient
)
```

```
def connect_to_stream_manager_with_default_port():
    client = StreamManagerClient()

    # Use the client.
```

JavaScript

```
const {
    StreamManagerClient
} = require('aws-greengrass-stream-manager-sdk');

function connectToStreamManagerWithDefaultPort() {
    const client = new StreamManagerClient();

    // Use the client.
}
```

Example 範例：使用非預設 Connect 埠連線至串流管理員

如果您使用預設值以外的連接埠設定串流管理員，則必須使用處理序間通訊，從元件組態擷取連接埠。

> **Note**
>
> 組port態參數包含您在部署串流管理員STREAM_MANAGER_SERVER_PORT時所指定的值。

Java

```
void connectToStreamManagerWithCustomPort() {
    EventStreamRPCConnection eventStreamRpcConnection =
 IPCUtils.getEventStreamRpcConnection();
    GreengrassCoreIPCClient greengrassCoreIPCClient = new
 GreengrassCoreIPCClient(eventStreamRpcConnection);
    List<String> keyPath = new ArrayList<>();
    keyPath.add("port");

    GetConfigurationRequest request = new GetConfigurationRequest();
    request.setComponentName("aws.greengrass.StreamManager");
    request.setKeyPath(keyPath);
    GetConfigurationResponse response =
```

```
            greengrassCoreIPCClient.getConfiguration(request,
 Optional.empty()).getResponse().get();
    String port = response.getValue().get("port").toString();
    System.out.print("Stream Manager is running on port: " + port);

    final StreamManagerClientConfig config = StreamManagerClientConfig.builder()

 .serverInfo(StreamManagerServerInfo.builder().port(Integer.parseInt(port)).build()).build()

    StreamManagerClient client =
 StreamManagerClientFactory.standard().withClientConfig(config).build();

    // Use the client.
}
```

Python

```
import awsiot.greengrasscoreipc
from awsiot.greengrasscoreipc.model import (
    GetConfigurationRequest
)
from stream_manager import (
    StreamManagerClient
)

TIMEOUT = 10

def connect_to_stream_manager_with_custom_port():
    # Use IPC to get the port from the stream manager component configuration.
    ipc_client = awsiot.greengrasscoreipc.connect()
    request = GetConfigurationRequest()
    request.component_name = "aws.greengrass.StreamManager"
    request.key_path = ["port"]
    operation = ipc_client.new_get_configuration()
    operation.activate(request)
    future_response = operation.get_response()
    response = future_response.result(TIMEOUT)
    stream_manager_port = str(response.value["port"])

    # Use port to create a stream manager client.
    stream_client = StreamManagerClient(port=stream_manager_port)

    # Use the client.
```

用 StreamManagerClient 於使用串流

在 Greengrass 核心裝置上執行的使用者定義 Greengrass 元件可以使用串流管理員 SDK 中的`StreamManagerClient`物件在串流管理員中建立串流，然後與串流互動。元件建立串流時，會定義串流的AWS 雲端目的地、優先順序以及其他匯出和資料保留原則。要將數據發送到流管理器，組件將數據附加到流。如果為串流定義了匯出目的地，串流管理員會自動匯出串流。

> **Note**
>
> 通常情況下，流管理器的客戶端是用戶定義的 Greengrass 組件。如果您的業務案例需要它，您還可以允許在 Greengrass 核心（例如，Docker 容器）上運行的非組件進程與流管理器進行交互。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說明用戶端如何呼叫`StreamManagerClient`方法來處理串流。如需有關方法及其引數的實作詳細資訊，請使用每個程式碼片段後面列出的 SDK 參考連結。

如果您在 Lambda 函數中使用串流管理員，則 Lambda 函數應該在函數處理常式`StreamManagerClient`之外實例化。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

要創建一個流，用戶定義的 Greengrass 組件調用創建方法，並在一個對象傳遞。`MessageStreamDefinition`此對象指定流的唯一名稱，並定義當達到最大流大小時，流管理器應該如何處理新的數據。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：

- 用於自動匯出的目標AWS IoT AnalyticsAWS IoT SiteWise、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。
- Kinesis Data Streams 和AWS IoT SiteWise目標的AWS IoT Analytics最大批次大小和批次間隔。符合任一條件時，串流管理員會匯出訊息。
- Time-to-live （ 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。
- 起始序列號。指定要在匯出時用作開始郵件的郵件序號。

如需詳細資訊`MessageStreamDefinition`，請參閱目標語言的 SDK 參考資料：

- MessageStreamDefinition在 Java 開發套件中
- MessageStreamDefinition在 Node.js 軟體開發套件中
- MessageStreamDefinition在 Python 開發套件

> **Note**
>
> `StreamManagerClient`也提供可用來將串流匯出至 HTTP 伺服器的目標目的地。此目標僅供測試之用。它不穩定或不支持在生產環境中使用。

建立串流之後，您的 Greengrass 元件可以將訊息附加至串流以傳送資料以供匯出，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。一種策略是為 Kinesis 資料串

流中AWS IoT Analytics的每個目標通道建立串流，不過您可以為串流定義多個目標。串流的生命週期相當耐久。

要求

此操作具有以下要求：

- 最低流管理器 SDK 版本:Python: 1.1.0 Node.js

範例

以下程式碼片段會建立名為 StreamName 的串流。它定義了MessageStreamDefinition和下屬數據類型流屬性。

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 發套件參考資料:建立訊息串流 MessageStreamDefinition

Java

```
try (final StreamManagerClient client =
 StreamManagerClientFactory.standard().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)
                                    .withS3(null)
                    )

    );
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Java 開發套件參考資料：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)
            .withS3(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 開發套件參考資料：createMessageStream| MessageStreamDefinition

如需設定匯出目的地的詳細資訊，請參閱the section called “匯出支援雲端目的地的組態”。

附加訊息

要將數據發送到流管理器進行導出，您的 Greengrass 組件將數據附加到目標流。匯出目的地決定要傳遞給此方法的資料類型。

要求

此操作具有以下要求：

- 最低流管理器 SDK 版本:Python: 1.1.0 Node.js

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 Greengrass 元件會附加一組資料。

此代碼片段具有以下要求：

- 最低流管理器 SDK 版本:Python: 1.1.0 Node.js

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：附加消息

Java

```
try (final StreamManagerClient client =
 StreamManagerClientFactory.standard().build()) {
    long sequenceNumber = client.appendMessage("StreamName", "Arbitrary byte
 array".getBytes());
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Java SDK 參考資料：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 軟件開發套件參考：appendMessage

AWS IoT SiteWise出口目的地

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 Greengrass 組件附加一個序列化對象。PutAssetPropertyValueEntry如需詳細資訊，請參閱 the section called “匯出至 AWS IoT SiteWise”。

> **Note**
>
> 當您將資料傳送至時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。

此代碼片段具有以下要求：

- 最低流管理器 SDK 版本:Python: 1.1.0 Node.js

Python

```
client = StreamManagerClient()

try:
    # SiteWise requires unique timestamps in all messages and also needs timestamps
 not earlier
    # than 10 minutes in the past. 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",
 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：附加消息 | 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 and also needs
 timestamps not earlier
    // than 10 minutes in the past. 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 SDK 參考資料：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 軟件開發套件參考：appendMessage | PutAssetPropertyValueEntry

Amazon S3 出口目的地

下列程式碼片段會將匯出工作附加至名為StreamName的資料流。對於 Amazon S3 目的地，您的 Greengrass 元件會附加序列化S3ExportTaskDefinition物件，其中包含有關來源輸入檔案和目標 Amazon S3 物件的資訊。如果指定的物件不存在，「串流管理員」會為您建立該物件。如需詳細資訊，請參閱 the section called “匯出到 Amazon S3”。

此代碼片段具有以下要求：

- 最低流管理器 SDK 版本:Python: 1.1.0 Node.js

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",
 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 發套件參考：附加訊息 ExportTaskDefinition

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 開發套件參考資料：appendMessage ExportTaskDefinition

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 開發套件參考資料：appendMessage ExportTaskDefinition

讀取訊息

讀取串流中的訊息。

要求

此操作具有以下要求：

- 最低流管理器 SDK 版本:Python: 1.1.0 Node.js

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_messsage_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 參考資料:讀取訊息 | ReadMessagesOptions

Java

```
try (final StreamManagerClient client =
 StreamManagerClientFactory.standard().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_messsage_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 開發套件參考資料：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 開發套件參考資料:readMessages | ReadMessagesOptions

列出串流

在流管理器中獲取流列表。

要求

此操作具有以下要求：

- 最低流管理器 SDK 版本:Python: 1.1.0 Node.js

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 發套件參考:清單流

Java

```
try (final StreamManagerClient client =
 StreamManagerClientFactory.standard().build()) {
    List<String> streamNames = client.listStreams();
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

開發套件參考資料：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 開發套件參考資料: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

此操作具有以下要求：

- 最低流管理器 SDK 版本:Python: 1.1.0 Node.js

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 參考資料:描述訊息串流

Java

```
try (final StreamManagerClient client =
 StreamManagerClientFactory.standard().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.
}
```

開發套件參考資料：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 開發套件參考資料：describeMessageStream

更新訊息串流

更新現有流的屬性。如果串流建立後您的需求有所變更，您可能會想要更新串流。例如：

- 為目的AWS 雲端地新增匯出組態。

- 增加串流的大小上限，以變更資料匯出或保留的方式。例如，串流大小與您的完整設定策略相結合，可能會導致資料遭到刪除或拒絕，串流管理員才能處理資料。
- 暫停並繼續匯出；例如，如果匯出工作長時間執行，而您想要對上傳資料進行配量。

您的 Greengrass 組件遵循以下高級過程來更新流：

1. 取得串流的說明。
2. 更新對應物件`MessageStreamDefinition`和從屬物件上的目標性質。
3. 通過在更新的`MessageStreamDefinition`。請務必包含更新串流的完整物件定義。未定義的屬性還原為預設值。

   您可以指定要在匯出作業中用作開始郵件的郵件序號。

要求

此操作具有以下要求：

- 最低流管理器 SDK 版本:Python: 1.1.0 Node.js

範例

下列程式碼片段會更新名為的串流`StreamName`。它會更新匯出至 Kinesis 資料串流的串流的多個屬性。

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.
```

開 Python 套件參考資料: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 SDK 參考資料：更新訊息串流 | 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 軟體開發套件參考資料：updateMessageStreamMessageStreamDefinition

更新串流的限制

更新串流時會套用下列限制條件。除非在下列清單中註明，否則更新會立即生效。

- 您無法更新資料流的持續性。若要變更此行為，請刪除串流並建立定義新持續性原則的串流。
- 您只能在下列情況下更新串流的大小上限：
  - 最大大小必須大於或等於流的當前大小。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。
  - 最大大小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。更新後的設定會套用至新區段。
- 存留時間 (TTL) 屬性的更新會套用至新的附加作業。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。
- 完整屬性的策略更新會套用至新的附加作業。如果您將策略設定為覆寫最舊的資料，串流管理員也可能會根據新設定覆寫現有區段。
- 寫入時清除屬性的更新會套用至新郵件。
- 匯出組態的更新會套用至新的匯出。更新要求必須包含您要支援的所有匯出組態。否則，流管理器將刪除它們。
  - 更新匯出組態時，請指定目標匯出組態的識別碼。
  - 若要新增匯出組態，請為新匯出組態指定唯一識別碼。
  - 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

此操作具有以下要求：

- 最低流管理器 SDK 版本:Python: 1.1.0 Node.js

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.
```

開 Python 套件參考資料：deleteMessageStream

Java

```
try (final StreamManagerClient client =
 StreamManagerClientFactory.standard().build()) {
    client.deleteMessageStream("StreamName");
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Java SDK 參考資料：刪除訊息串流

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.
});
```

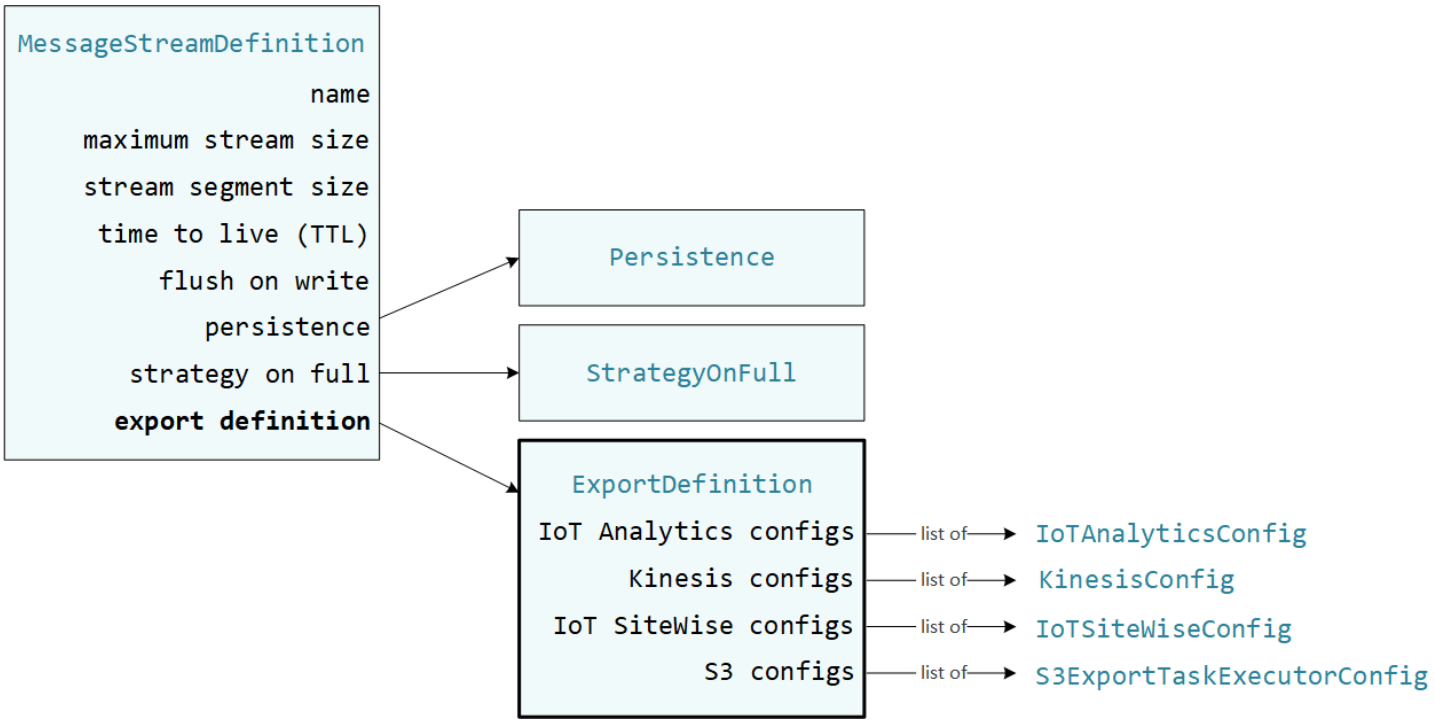
Node.js 開發套件參考資料：deleteMessageStream

另請參閱

- 管理核心裝 Greengrass 資料串流
- 設定 AWS IoT Greengrass 串流管理員
- 匯出支援AWS 雲端目的地的組態
- `StreamManagerClient`在流管理器 SDK 參考中：
  - Python
  - Java
  - Node.js

匯出支援AWS 雲端目的地的組態

使用者定義 Greengrass 元件會在串流管理員 SDK `StreamManagerClient` 中使用，與串流管理員互動。當組件建立串流或更新串流時，它會傳遞代表資料流屬性的`MessageStreamDefinition`物件，包括匯出定義。該`ExportDefinition`對象包含為流定義的導出配置。串流管理員會使用這些匯出設定來決定匯出串流的位置和方式。



您可以在串流上定義零個或多個匯出設定，包括單一目標類型的多個匯出設定。例如，您可以將串流匯出到兩個AWS IoT Analytics通道和一個 Kinesis 資料串流。

對於失敗的匯出嘗試，串流管理員會持續重試將資料匯出至最多五分鐘的間隔。AWS 雲端重試次數沒有上限。

> **Note**
>
> `StreamManagerClient`也提供可用來將串流匯出至 HTTP 伺服器的目標目的地。此目標僅供測試之用。它不穩定或不支持在生產環境中使用。

支援AWS 雲端目的地

- AWS IoT Analytics 頻道
- Amazon Kinesis 數據流
- AWS IoT SiteWise資產性質
- Amazon S3 對象

您有責任維護這些AWS 雲端資源。

AWS IoT Analytics 頻道

流管理器支持自動導出到AWS IoT Analytics. AWS IoT Analytics可讓您對資料執行進階分析，以協助制定商業決策並改善機器學習模型。如需詳細資訊，請參閱什麼是AWS IoT Analytics？ 在《AWS IoT Analytics使用者指南》中。

在串流管理員 SDK 中，您的 Greengrass 元件會使用`IoTAnalyticsConfig`來定義此目標類型的匯出設定。如需詳細資訊，請參閱您目標語言的 SDK 參考資料：

- 開發套件AnalyticsConfig中的 IoT
- Java SDK AnalyticsConfig 中的 IoT
- Node.js 開發套件AnalyticsConfig中的 IoT

要求

此匯出目的地具有下列需求：

- 中的目標通道AWS IoT Analytics必須AWS 帳戶與 AWS 區域 Greengrass 核心裝置位於相同的通道中。

- 授權核心設備與AWS服務必須允許以通道為目標的`iotanalytics:BatchPutMessage`權限。例如：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iotanalytics:BatchPutMessage"
      ],
      "Resource": [
        "arn:aws:iotanalytics:region:account-id:channel/channel_1_name",
        "arn:aws:iotanalytics:region:account-id:channel/channel_2_name"
      ]
    }
  ]
}
```

  您可以授與資源的細微或條件式存取權，例如使用萬用字元*命名配置。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 Greengrass 元件會建立包含一或多個物件的匯出定義串流。`IoTAnalyticsConfig`此物件會定義匯出設定，例如目標通道、批次大小、批次間隔和優先順序。

當 Greengrass 組件從設備接收數據時，它們會將包含數據 Blob 的消息附加到目標流中。

然後，串流管理員會根據串流的匯出設定中定義的批次設定和優先順序匯出資料。

Amazon Kinesis 數據流

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 MapReduce 叢集。如需詳細資訊，請參閱什麼是 Amazon Kinesis Data Streams？ 在 Amazon Kinesis 開發人員指南中。

在串流管理員 SDK 中，您的 Greengrass 元件會使用`KinesisConfig`來定義此目標類型的匯出設定。如需詳細資訊，請參閱您目標語言的 SDK 參考資料：

- KinesisConfig在 Python 開發套件
- KinesisConfig在 Java 開發套件中
- KinesisConfig在 Node.js 軟體開發套件中

要求

此匯出目的地具有下列需求：

- Kinesis 資料串流中的目標串流必須AWS 帳戶與 AWS 區域 Greengrass 核心裝置相同。
- 授權核心設備與AWS服務必須允許以資料串流為目標的`kinesis:PutRecords`權限。例如：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kinesis:PutRecords"
      ],
      "Resource": [
        "arn:aws:kinesis:region:account-id:stream/stream_1_name",
        "arn:aws:kinesis:region:account-id:stream/stream_2_name"
      ]
    }
  ]
}
```

  您可以授與資源的細微或條件式存取權，例如使用萬用字元*命名配置。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，您的 Greengrass 元件會建立包含一或多個物件的匯出定義串流。`KinesisConfig`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 Greengrass 組件從設備接收數據時，它們會將包含數據 Blob 的消息附加到目標流中。然後，串流管理員會根據串流的匯出設定中定義的批次設定和優先順序匯出資料。

串流管理員會針對上傳到 Amazon Kinesis 的每筆記錄，產生一個唯一的隨機 UUID 做為分區金鑰。

AWS IoT SiteWise資產性質

流管理器支持自動導出到AWS IoT SiteWise. AWS IoT SiteWise可讓您大規模收集、組織和分析來自工業設備的資料。如需詳細資訊，請參閱什麼是AWS IoT SiteWise？ 在《AWS IoT SiteWise使用者指南》中。

在串流管理員 SDK 中，您的 Greengrass 元件會使用`IoTSiteWiseConfig`來定義此目標類型的匯出設定。如需詳細資訊，請參閱您目標語言的 SDK 參考資料：

- 開發套件SiteWiseConfig中的 IoT
- Java SDK SiteWiseConfig 中的 IoT
- Node.js 開發套件SiteWiseConfig中的 IoT

> **Note**
>
> AWS也提供AWS IoT SiteWise元件，這些元件提供預先建置的解決方案，可用來串流來自 OPC-UA 來源的資料。如需詳細資訊，請參閱 IoT 集 SiteWise 電極。

需求

此匯出目的地具有下列需求：

- 中的目標資產屬性AWS IoT SiteWise必須AWS 帳戶與 AWS 區域 Greengrass 核心裝置位於相同的內容。

  > **Note**
  >
  > 如需AWS IoT SiteWise支援的AWS 區域清單，請參閱AWS一般參考中的AWS IoT SiteWise端點和配額。

- 授權核心設備與AWS服務必須允許以資產屬性為目標的`iotsitewise:BatchPutAssetPropertyValue`權限。下列範例策略使用`iotsitewise:assetHierarchyPath`條件索引鍵來授與目標根資產及其子系的存取權。您可以`Condition`從政策中移除，以允許存取所有資AWS IoT SiteWise產或指定個別資產的 ARN。

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

```
  "Statement": [
    {
       "Effect": "Allow",
       "Action": "iotsitewise:BatchPutAssetPropertyValue",
       "Resource": "*",
       "Condition": {
         "StringLike": {
           "iotsitewise:assetHierarchyPath": [
             "/root node asset ID",
             "/root node asset ID/*"
           ]
         }
       }
    }
  ]
}
```

您可以授與資源的細微或條件式存取權，例如使用萬用字元*命名配置。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 Greengrass 元件會建立包含一或多個物件的匯出定義串流。`IoTSiteWiseConfig`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 Greengrass 組件從設備接收資產屬性數據時，它們會將包含數據的消息附加到目標流中。消息是 JSON 序列化`PutAssetPropertyValueEntry`對象，其中包含一個或多個資產屬性的屬性值。如需詳細資訊，請參閱為AWS IoT SiteWise匯出目的地附加郵件。

> **Note**
>
> 當您將資料傳送至時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的匯出設定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 Greengrass 元件邏輯來設計匯出策略。例如：

- 對於近乎即時的匯出，請設定較低的批次大小和間隔設定，並在收到串流時將資料附加至串流。
- 為了最佳化批次處理、減輕頻寬限制或將成本降至最低，Greengrass 元件可以在將資料附加至串流之前，為單一資產屬性集區收到的 timestamp-quality-value (TQV) 資料點。其中一種策略是在一則訊息中批次處理最多 10 個不同屬性資產組合或屬性別名的項目，而不是針對相同屬性傳送一個以上的項目。這有助於流管理器保持在AWS IoT SiteWise配額內。

Amazon S3 對象

串流管理員支援自動匯出至 Amazon S3。您可以使用 Amazon S3 存放和擷取大量資料。如需詳細資訊，請參閱什麼是 Amazon S3？ 在 Amazon 簡單存儲服務開發人員指南中。

在串流管理員 SDK 中，您的 Greengrass 元件會使用`S3ExportTaskExecutorConfig`來定義此目標類型的匯出設定。如需詳細資訊，請參閱您目標語言的 SDK 參考資料：

- 開發套件ExportTaskExecutorConfig中 Python S3
- Java 開發套件ExportTaskExecutorConfig中的 S3
- Node.js 開發套件ExportTaskExecutorConfig中的 S3

要求

此匯出目的地具有下列需求：

- 目標 Amazon S3 儲存貯體必須與 Greengrass 核心裝置位於AWS 帳戶相同的位置。
- 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，您必須將目錄掛載為具有寫入權限的容器中的磁碟區。這樣可確保檔案會寫入根檔案系統，並且可以看到在容器外部執行的串流管理員元件。
- 如果 Docker 容器組件將輸入文件寫入輸入文件目錄，則必須將該目錄掛載為具有寫入權限的容器中的卷。這樣可確保檔案會寫入根檔案系統，並且可以看到在容器外部執行的串流管理員元件。
- 授權核心設備與AWS服務必須允許目標值區的下列權限。例如：

```
{
  "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/*"
      ]
    }
  ]
}
```

您可以授與資源的細微或條件式存取權，例如使用萬用字元*命名配置。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。

匯出到 Amazon S3

若要建立匯出至 Amazon S3 的串流，您的 Greengrass 元件會使用該`S3ExportTaskExecutorConfig`物件來設定匯出政策。此原則會定義匯出設定，例如分段上傳臨界值和優先順序。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 Greengrass 元件會將匯出工作附加至目標資料流。匯出任務包含輸入檔案和目標 Amazon S3 物件的相關資訊。流管理器按照它們被附加到流的順序運行任務。

> **Note**
>
> 目標值區必須已存在於您的AWS 帳戶. 如果指定索引鍵的物件不存在，串流管理員會為您建立物件。

流管理器使用多部分上傳閾值屬性，最小部分大小設置和輸入文件的大小來確定如何上傳數據。多部分上傳臨界值必須大於或等於最小零件大小。如果您想要 parallel 上傳資料，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。您可以使用這些時間戳記預留位置，根據輸入檔案資料上傳的時間在 Amazon S3 中分割資料。例如，下列索引鍵名稱會解析為一個值，例如`my-key/2020/12/31/data.txt`。

```
my-key/!{timestamp:YYYY}/!{timestamp:MM}/!{timestamp:dd}/data.txt
```

> **Note**
>
> 如果要監視串流的匯出狀態，請先建立狀態串流，然後設定匯出串流以使用它。如需詳細資訊，請參閱 the section called “監控匯出工作”。

管理輸入資料

您可以撰寫 IoT 應用程式用來管理輸入資料生命週期的程式碼。下列範例工作流程顯示如何使用 Greengrass 元件來管理此資料。

1. 本機處理程序會從裝置或周邊設備接收資料，然後將資料寫入核心裝置上目錄中的檔案。這些都是流管理器的輸入文件。
2. Greengrass 組件掃描目錄，並在創建新文件時將導出任務附加到目標流。工作是 JSON 序列化`S3ExportTaskDefinition`物件，用於指定輸入檔案的 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，並按照附加任務的順序將資料匯出至 Amazon S3。目標值區必須已存在於您的AWS 帳戶. 如果指定索引鍵的物件不存在，串流管理員會為您建立物件。
4. Greengrass 元件會從狀態串流讀取訊息，以監視匯出狀態。匯出工作完成後，Greengrass 元件可以刪除對應的輸入檔案。如需詳細資訊，請參閱 the section called “監控匯出工作”。

監控匯出工作

您可以撰寫 IoT 應用程式用來監控 Amazon S3 匯出狀態的程式碼。您的 Greengrass 元件必須建立狀態串流，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，建立用作狀態串流的串流。您可以設定串流的大小和保留原則，以控制狀態訊息的壽命。例如：

- `Memory`如果您不想儲存狀態訊息，請設`Persistence`定為。
- 設定`StrategyOnFull`為可`OverwriteOldestData`讓新的狀態訊息不會遺失。

然後，建立或更新匯出串流以使用狀態串流。具體來說，設置流的`S3ExportTaskExecutorConfig`導出配置的 status 配置屬性。此設置告訴流管理器寫入有關導出任務到狀態流的狀態消息。在`StatusConfig`物件中，指定狀態串流的名稱和詳細程度層級。下列支援的值範圍從最小冗長 (`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 Greengrass 元件如何使用狀態串流監視匯出狀態。

1. 如先前的工作流程所述，Greengrass 元件會將匯出工作附加至設定為寫入有關匯出工作至狀態串流的狀態訊息的串流。附加作業會傳回代表工作 ID 的序號。
2. Greengrass 組件從狀態流中按順序讀取消息，然後根據流名稱和任務 ID 或基於從消息上下文的導出任務屬性過濾消息。例如，Greengrass 組件可以依匯出工作的輸入檔案 URL 進行篩選，此 URL 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完成狀態：

   - `Success`。 上傳已成功完成。
   - `Failure`。 流管理器遇到錯誤，例如，指定的存儲桶不存在。解決問題後，您可以再次將匯出工作附加至串流。
   - `Canceled`。 因為已刪除串流或匯出定義，或任務的 time-to-live (TTL) 期間到期，因此作業已停止。

   > **Note**
   >
   > 工作的狀態也可能為`InProgress`或`Warning`。當事件傳回不會影響工作執行的錯誤時，串流管理員會發出警告。例如，如果清除部分上傳失敗，就會傳回警告。

3. 匯出工作完成後，Greengrass 元件可以刪除對應的輸入檔案。

下列範例顯示 Greengrass 元件可能如何讀取和處理狀態訊息。

Python

```
import time
from stream_manager import (
    ReadMessagesOptions,
    Status,
    StatusConfig,
```

```
    StatusLevel,
    StatusMessage,
    StreamManagerClient,
)
from 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 參考資料:讀取訊息 | StatusMessage

Java

```
import com.amazonaws.greengrass.streammanager.client.StreamManagerClient;
import com.amazonaws.greengrass.streammanager.client.StreamManagerClientFactory;
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 =
 StreamManagerClientFactory.standard().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 開發套件參考資料：readMessages | StatusMessage

Node.js

```
const {
    StreamManagerClient, ReadMessagesOptions,
    Status, StatusConfig, StatusLevel, StatusMessage,
    util,
} = require(*'aws-greengrass-stream-manager-sdk'*);

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 開發套件參考資料:readMessages | StatusMessage

設定 AWS IoT Greengrass 串流管理員

在 Greengrass 核心裝置上，串流管理員可以儲存、處理和匯出 IoT 裝置資料。串流管理員會提供您用來設定執行階段設定的參數。這些設定會套用至 Greengrass 核心裝置上的所有串流。部署元件時，您可以使用AWS IoT Greengrass主控台或 API 來設定串流管理員設定。變更會在部署完成後生效。

串流管理員參數

串流管理員提供下列參數，您可以在將元件部署到核心裝置時設定這些參數。所有參數都是選用的。

儲存目錄

參數名稱：STREAM_MANAGER_STORE_ROOT_DIR

用來儲存串流之本機資料夾的絕對路徑。此值必須以正斜線開頭 (例如 /data)。

您必須指定現有的資料夾，而且執行串流管理員元件的系統使用者必須擁有讀取和寫入此資料夾的權限。例如，您可以執行下列命令來建立和設定資料夾/var/greengrass/streams，您可以將其指定為串流管理員根資料夾。這些指令允許預設系統使用者讀取和寫入此資料夾。ggc_user

```
sudo mkdir /var/greengrass/streams
sudo chown ggc_user /var/greengrass/streams
sudo chmod 700 /var/greengrass/streams
```

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”。

預設：*/greengrass/v2*/work/aws.greengrass.StreamManager

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。

您可以指定0使用隨機可用的連接埠。

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。客戶端和流管理器之間的所有交互都由流管理器 SDK 控制。這個參數決定哪些用戶端可以呼叫串流管理員 SDK 來處理串流。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`。 只允許 Greengrass 組件作為客戶端。元件會使用內部AWS IoT Greengrass核心通訊協定，透過串流管理員 SDK 進行驗證。
- `false`。 允許在AWS IoT Greengrass核心上運行的任何進程是一個客戶端。`false`除非您的商業案例需要，否則請勿將值設定為。例如，`false`只有在核心裝置上的非元件程序必須與串流管理員直接通訊時才使用。

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。

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。每個串流都會耗用硬體資源，因此請嘗試限制受限裝置上的匯出串流數量。

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。

Logging level (記錄層級)

參數名稱：`LOG_LEVEL`

元件的記錄層級。從下列記錄層級中進行選擇，依照層級順序列出：

- TRACE
- DEBUG
- INFO
- WARN
- ERROR

預設：INFO

多部分上傳的最小尺寸

參數名
稱：STREAM_MANAGER_EXPORTER_S3_DESTINATION_MULTIPART_UPLOAD_MIN_PART_SIZE_BYTE

多部分上傳到 Amazon S3 中，零件的大小下限 (以位元組為單位)。流管理器使用此設置和輸入文件的大小來確定如何批處理多部分 PUT 請求中的數據。預設值和最小值為5242880位元組 (5 MB)。

> **Note**
>
> 串流管理員使用串流的sizeThresholdForMultipartUploadBytes屬性來判斷是以單一或多部分上傳方式匯出到 Amazon S3。使用者定義的 Greengrass 元件在建立匯出至 Amazon S3 的串流時，會設定此閾值。預設臨界值為 5 MB。

另請參閱

- 管理核心裝 Greengrass 資料串流
- 用 StreamManagerClient 於使用串流
- 匯出支援AWS 雲端目的地的組態

執行機器學習推論

透過AWS IoT Greengrass，您可以使用雲端訓練的模型，在本機產生的資料上，在邊緣裝置上執行機器學習 (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。

AWS IoT Greengrass使執行推論所需的步驟更有效率。您可以在任何地方訓練推論模型，並在本機部署為機器學習元件。例如，您可以在 Amazon 中建立和訓練深度學習模型， SageMaker或在 Amazon 瞭望視覺中建置和訓練電腦視覺模型。然後，您可以將這些模型存放在 Amazon S3 儲存貯體中，以便將這些模型用作元件中的成品，在核心裝置上執行推論。

主題

AWS IoT Greengrass 機器學習推論如何運作

AWS提供機器學習元件，您可以使用這些元件建立單步驟部署，以便在裝置上執行機器學習推論。您也可以使用這些元件做為範本來建立自訂元件，以符合您的特定需求。

AWS提供下列類別的機器學習元件：

- 模型元件 — 包含做為 Greengrass 人工因素的機器學習模型。
- 執行階段元件 — 包含在 Greengrass 核心裝置上安裝機器學習架構及其相依性的指令碼。
- 推論元件：包含推論程式碼並包含元件相依性，可安裝機器學習架構和下載預先訓練的機器學習模型。

您為執行機器學習推論而建立的每個部署都包含至少一個執行推論應用程式的元件、安裝機器學習架構，以及下載您的機器學習模型。若要使用AWS提供的元件執行範例推論，您可以將推論元件部署到核心裝置，而核心裝置會自動包含對應的模型和執行階段元件做為相依性。若要自訂部署，您可以使用自訂模型元件插入或替換範例模型元件，或者您可以使用AWS提供之元件的元件配方做為範本，以建立您自己的自訂推論、模型和執行階段元件。

若要使用自訂元件執行機器學習推論：

1. 建立模型元件。此元件包含您要用來執行推論的機器學習模型。 AWS提供樣品預先訓練的 DLR 和 TensorFlow 精簡版模型。若要使用自訂模型，請建立您自己的模型元件。
2. 建立執行階段元件。此元件包含為您的模型安裝機器學習執行階段所需的指令碼。 AWS提供深度學習執行階段 (DLR) 和 TensorFlow Lite 的範例執行階段元件。若要將其他執行階段與自訂模型和推論程式碼搭配使用，請建立您自己的執行階段元件。
3. 建立推論元件。此元件包含您的推論程式碼，並包含您的模型和執行階段元件做為相依性。 AWS提供使用 DLR 和 Lite 進行影像分類和物件偵測的範例推論元件。 TensorFlow 若要執行其他類型的推論，或使用自訂模型和執行階段，請建立您自己的推論元件。
4. 部署推論元件。當您部署此元件時，AWS IoT Greengrass也會自動部署模型和執行階段元件相依性。

若要開始使用提AWS供的元件，請參閱the section called “執行範例影像分類推論”。

如需有關建立自訂機器學習元件的資訊，請參閱自訂您的機器學習元件。

AWS IoT Greengrass版本 2 有什麼不同？

AWS IoT Greengrass將機器學習的功能單元 (例如模型、執行階段和推論程式碼) 整合到元件中，讓您使用單一步驟程序來安裝機器學習執行階段、下載訓練過的模型，以及在裝置上執行推論。

透過使用AWS提供的機器學習元件，您可以彈性地開始使用範例推論程式碼和預先訓練的模型來執行機器學習推論。您可以插入自訂模型元件，將您自己的自訂訓練模型與提供的推論和執行階段元件搭配使用AWS。對於完全自訂的機器學習解決方案，您可以使用公用元件做為範本來建立自訂元件，並使用任何您想要的執行階段、模型或推論類型。

要求

若要建立和使用機器學習元件，您必須具備下列項目：

- 一個 Greengrass 核心設備。如果您沒有帳戶，請參閱 教學課程：AWS IoT Greengrass V2 入門。
- 至少 500 MB 本機儲存空間，可使用AWS提供的範例機器學習元件。

支援的模型來源

AWS IoT Greengrass支援使用存放在 Amazon S3 中的自訂訓練機器學習模型。您也可以使用 Amazon SageMaker Edge 封裝任務，為 SageMaker 新編譯的模型直接建立模型元件。如需搭配使用 SageMaker 邊管理員的資訊AWS IoT Greengrass，請參閱在核心設備上使用 Amazon SageMaker 邊緣管理器。您也可以使用 Amazon Lookout for Vision 模型封裝任務，為您的 Lookout for Vision 模型建立模型元件。如需使用瞭望視覺搭配使用的更多資訊AWS IoT Greengrass，請參閱Greengrass out out out out out out out out out out on Vision。

包含您模型的 S3 儲存貯體必須符合下列需求：

- 它們不得使用 SSE-C 進行加密。 對於使用伺服器端加密的儲存貯體，AWS IoT Greengrass機器學習推論目前僅支援 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 簡單儲存服務使用者指南中的使用伺服器端加密保護資料。
- 它們的名稱不得包含句點 (.)。如需詳細資訊，請參閱 Amazon 簡單儲存體服務使用者指南中的儲存貯體命名規則中有關將虛擬託管樣式值區與 SSL 搭配使用的規則。
- 存放模型來源的 S3 儲存貯體必須AWS 帳戶與AWS 區域您的機器學習元件位於相同的儲存貯體中。
- AWS IoT Greengrass必須具有模型來源的`read`權限。若AWS IoT Greengrass要啟用存取 S3 儲存貯體，Greengrass 裝置角色必須允許此動作。`s3:GetObject`如需有關裝置角色的詳細資訊，請參閱授權核心設備與AWS服務。

支援的機器學習執行階段

AWS IoT Greengrass可讓您建立自訂元件，以使用您選擇的任何機器學習執行階段，透過自訂訓練的模型執行機器學習推論。如需有關建立自訂機器學習元件的資訊，請參閱自訂您的機器學習元件。

為了讓開始使用機器學習的程序更有效率，請AWS IoT Greengrass提供使用下列機器學習執行階段的範例推論、模型和執行階段元件：

- 深度學習執行階段 (DLR)
- TensorFlow 精簡版

AWS-提供機器學習元件

下表列出用於機器學習的 AWS-提供的元件。

> **Note**
>
> 幾個AWS提供的組件取決於 Greengrass 核的特定次要版本。由於這種依賴關係，您需要在將 Greengrass 核更新為新的次要版本時更新這些組件。如需每個元件所依賴之特定原子核版本的詳細資訊，請參閱對應的元件主題。如需更新核心的詳細資訊，請參閱更新AWS IoT Greengrass核心軟件（OTA）。

元件	描述	元件類型	支援的作業系統	開放原始碼
Lookout for Vision 邊緣代理	在 Greengrass 核心裝置上部署適用於視覺的 Amazon Lookout 執行階段，因此您可以使用電腦視覺來尋找工業產品中的瑕疵。	通用型	Linux	否
SageMaker 邊緣管理員	在 Greengrass 核心裝置上部署 Amazon SageMaker 邊緣管理器代理程式。	通用型	Linux、Windows	否
DLR 影像分類	使用 DLR 影像分類模型存放區和 DLR 執行階段元件做為	通用型	Linux、Windows	否

元件	描述	元件類型	支援的作業系統	開放原始碼
	相依性的推論元件，以安裝 DLR、下載範例影像分類模型，以及在支援的裝置上執行影像分類推論。			
DLR 物體偵測	使用 DLR 物件偵測模型存放區和 DLR 執行階段元件做為相依性的推論元件，以安裝 DLR、下載範例物件偵測模型，以及在支援的裝置上執行物件偵測推論。	通用型	Linux、Windows	否
DLR 圖像分類模型商店	包含示例 ResNet -50 圖像分類模型作為 Greengrass 工件的模型組件。	通用型	Linux、Windows	否
DLR 物件偵測模型商店	包含樣本 Yolov3 對象檢測模型作為綠色工件的模型組件。	通用型	Linux、Windows	否

元件	描述	元件類型	支援的作業系統	開放原始碼
DLR 執行階段	包含用於在 Greengrass 核心裝置上安裝 DLR 及其相依性的安裝指令碼的執行階段元件。	通用型	Linux、Windows	否
TensorFlow 精簡版圖片分類	使用 TensorFlow Lite 影像分類模型存放區和 Lite 執行階段元件做為相依性的推論元件，以安裝 TensorFlow Lite、下載範例影像分類模型，以及在支援的裝置上執行影像分類推論。TensorFlow	通用型	Linux、Windows	否

元件	描述	元件類型	支援的作業系統	開放原始碼
TensorFlow 精簡型物體偵測	使用 TensorFlow Lite 物件偵測模型存放區和 Lite 執行階段元件做為相依性的推論元件，以安裝 TensorFlow Lite、下載範例物件偵測模型，以及在支援的裝置上執行物件偵測推論。TensorFlow	通用型	Linux、Windows	否
TensorFlow 精簡版圖片分類模型店	包含做為 Greengrass 人工因素的 MobileNet v1 模型範例的模型元件。	通用型	Linux、Windows	否
TensorFlow 精簡型物件偵測模型商店	包含做為 Greengrass 假影的範例單次發射偵測 (SSD) MobileNet 模型的模型元件。	通用型	Linux、Windows	否

元件	描述	元件類型	支援的作業系統	開放原始碼
TensorFlow 精簡版運行	包含用於在 Greengrass 核心裝置上安裝 TensorFlow Lite 及其相依性的安裝指令碼的執行階段元件。	通用型	Linux、Windows	否

在核心設備上使用 Amazon SageMaker 邊緣管理器

> **Important**
>
> SageMaker 邊緣管理員將於 2024 年 4 月 26 日停止使用。如需有關繼續將模型部署到邊緣裝置的詳細資訊，請參閱SageMaker 邊緣管理員生命週期結束。

Amazon SageMaker 邊緣管理器是在邊緣裝置上執行的軟體代理程式。 SageMaker Edge Manager 提供邊緣裝置的模型管理功能，讓您可以直接在 Greengrass 核心裝置上封裝和使用 Amazon SageMaker 新編譯模型。透過使用 SageMaker Edge Manager，您也可以從核心裝置取樣模型輸入和輸出資料，並將該資料傳送至以進AWS 雲端行監視和分析。由於 SageMaker Edge Manager 使用 SageMaker Neo 針對目標硬體最佳化模型，因此您不需要直接在裝置上安裝 DLR 執行階段。在 Greengrass 裝置上， SageMaker 邊緣管理員不會載入本機AWS IoT憑證或直接呼叫AWS IoT認證提供者端點。相反地， SageMaker Edge 管理員會使用權杖交換服務從 TES 端點擷取暫時認證。

本節說明 SageMaker 邊緣管理員如何在 Greengrass 核心裝置上運作。

SageMaker 邊緣管理器如何在 Greengrass 設備上工作

若要將 SageMaker Edge Manager 代理程式部署到核心裝置，請建立包含該`aws.greengrass.SageMakerEdgeManager`元件的部署。 AWS IoT Greengrass管理您裝置上 Edge Manager 代理程式的安裝和生命週期。當有新版的代理程式二進位檔可用時，請部

署aws.greengrass.SageMakerEdgeManager元件的更新版本，以升級裝置上安裝的代理程式版本。

搭配使用 SageMaker Edge Manager 時AWS IoT Greengrass，您的工作流程包括下列高階步驟：

1. 使用 SageMaker Neo 編譯模型。
2. 使用 SageMaker 邊緣 P SageMaker ackage 裝工作來封裝您的新編譯模型。當您為模型執行邊緣封裝工作時，您可以選擇建立包含封裝模型的模型元件，做為可部署到 Greengrass 核心裝置的成品。
3. 建立自訂推論元件。您可以使用此推論元件與 Edge Manager 代理程式互動，以便在核心裝置上執行推論。這些作業包括載入模型、叫用預測要求以執行推論，以及在元件關閉時卸載模型。
4. 部署 SageMaker Edge Manager 元件、封裝的模型元件和推論元件，以便在裝置上的推 SageMaker 論引擎 (Edge Manager 代理程式) 上執行您的模型。

如需建立與 Edge Manager 搭配使用的邊 SageMaker 緣封裝任務和推論元件的詳細資訊，請參閱 Amazon SageMaker 開發人員指南AWS IoT Greengrass中的部署模型 Package 件和 Edge Manager 代理程式。

本教學課程：開始使用 SageMaker 邊緣管理員教學課程說明如何使用可用來建立範例推論和模型AWS元件的範例程式碼，在現有 Greengrass 核心裝置上設定和使用 SageMaker Edge Manager 代理程式。

當您在 Greengrass 核心裝置上使用 SageMaker 邊緣管理員時，您也可以使用擷取資料功能將範例資料上傳至. AWS 雲端 擷取資料是一 SageMaker 項功能，可用來將推論輸入、推論結果和其他推論資料上傳至 S3 儲存貯體或本機目錄，以供 future 分析使用。如需有關透過 SageMaker Edge Manager 使用擷取資料的詳細資訊，請參閱 Amazon SageMaker 開發人員指南中的管理模型。

要求

您必須符合下列需求，才能在 Greengrass 核心裝置上使用 SageMaker 邊緣管理員代理程式。

- 在 Amazon Linux 2，基於 Debian 的 Linux 平台（x86_64 或 Armv8）或視窗（x86_64）上運行的 Greengrass 核心設備。如果您沒有帳戶，請參閱 教學課程：AWS IoT Greengrass V2 入門。
- Python 3.6 或更高版本，包括您pip的 Python 版本，安裝在您的核心設備上。
- 設定了以下內容的 Greengrass 裝置角色：
  - 允許credentials.iot.amazonaws.com和擔任角色sagemaker.amazonaws.com的信任關係，如以下 IAM 政策範例所示。

```
{
```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
       },
      "Action": "sts:AssumeRole"
    },
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "sagemaker.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

- AmazonSageMakerEdgeDeviceFleetPolicyIAM 受管政策。
- 動`s3:PutObject`作，如以下 IAM 政策範例所示。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:PutObject"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

- 在AWS 帳戶與AWS 區域您的 Greengrass 核心裝置相同建立的 Amazon S3 儲存貯體。 SageMaker Edge Manager 需要 S3 儲存貯體來建立邊緣裝置叢集，並存放在裝置上執行推論時所產生的範例資料。如需建立 S3 儲存貯體的相關資訊，請參閱開始使用 Amazon S3。
- SageMaker 邊緣裝置叢集使用與 Greengrass 核心裝置相同的AWS IoT角色別名。如需詳細資訊，請參閱 建立邊緣裝置叢集。

- 您的 Greengrass 核心裝置已註冊為 Edge 裝置群組中的邊 SageMaker 緣裝置。Edge 裝置名稱必須與核心裝置的AWS IoT物件名稱相符。如需詳細資訊，請參閱 註冊 Greengrass 設備。

開始使用 SageMaker 邊緣管理員

您可以完成教學課程以開始使用「 SageMaker 邊緣管理員」。本教學課程說明如何開始使用 SageMaker Edge Manager，搭配現有核心裝置上AWS提供的範例元件。這些範例元件會使用 SageMaker Edge Manager 元件做為相依性來部署 Edge Manager 代理程式，並使用使用 Neo 編譯的預先訓練模型執行推論。 SageMaker 如需更多詳細資訊，請參閱 教學課程：開始使用 SageMaker 邊緣管理員。

Greengrass out out out out out out out out out out on Vision

> **Note**
>
> AWS IoT Greengrass目前在 Windows 核心裝置上不支援此功能。

亞馬遜視 Lookout for Vision 察AWS 服務是一種您可以使用它來查找工業產品中的視覺缺陷。它使用電腦視覺來識別工業產品中遺失的元件、車輛或結構的損壞、生產線上的不規則性、印刷電路板上的電容器遺失，以及矽晶圓或任何其他品質至關重要的實體項目的缺陷。如需詳細資訊，請參閱什麼是 Amazon Lookout for Vision 在亞馬遜 Lookout for Vision 開發人員指南。

您可以建立 Greengrass 應用程式，使用瞭望視覺推論來尋找 Greengrass 核心裝置上的視覺缺陷。在您將 Lookout for Vision 工作流程部署到 Greengrass 核心裝置之後，您可以執行電腦視覺，而不需要連線至中 Lookout for Vision 服務AWS 雲端。若要建立使用 Lookout for Vision 的 Greengrass 應用程式，您可以設定並部署下列 Greengrass 元件：

- Lookout for Vision 模型元件 — 包含 Lookout for Vision 機器學習模型做為 Greengrass 工件。您可以使用 Lookout for Vision 主控台和 API 來產生模型元件，以封裝預先訓練的機器學習模型。這些 Greengrass 您的AWS 帳戶. 如需詳細資訊，請參閱 Amazon Lookout for Vision 開發人員指南中的建立視覺監視模型和封裝 Lookout for Vision 視模型。
- Lookout for Vision Edge 代理程式元件 — 提供本機 Lookout for Vision 執行階段伺服器，該伺服器使用電腦視覺，使用您提供的機器學習模型偵測異常。這個組件是一個AWS-提供的組件。如需詳細資訊，請參閱檢 Lo okout for Vision 邊緣代理程式元件。
- Lookout for Vision 戶端應用程式元件 — 與 Lookout for Vision 邊緣代理程式元件互動，以處理異常的影像。您可以開發自訂的用戶端應用程式元件，將影像和視訊串流傳送至本機 Lookout for Vision

Edge 代理程式，並報告機器學習模型偵測到的任何異常。如需詳細資訊，請參閱 Amazon Lookout 視覺開發人員指南中的撰寫用戶端應用程式元件和 Lookout for Vision 邊緣代理程式 API 參考。

如需有關如何建立、設定和使用這些元件的詳細資訊，請參閱 Amazon Lookout for Vision 開發人員指南中的在邊緣裝置上使 Lookout for Vision 檢視模型。

自訂您的機器學習元件

在中AWS IoT Greengrass，您可以設定範例機器學習元件，以自訂在裝置上執行機器學習推論的方式，並將推論、模型和執行階段元件做為建置區塊。 AWS IoT Greengrass此外，您還可以靈活地使用範例元件做為範本，並視需要建立自己的自訂元件。您可以混搭這種模組化方法，以下列方式自訂機器學習推論元件：

使用範例推論元件

- 在部署推論元件時修改它們的組態。
- 將範例模型存放區元件取代為自訂模型元件，以使用範例推論元件的自訂模型。您的自訂模型必須使用與範例模型相同的執行階段進行訓練。

使用自訂推論元件

- 透過新增公用模型元件和執行階段元件做為自訂推論元件的相依性，將自訂推論程式碼與範例模型和執行階段搭配使用。
- 建立並新增自訂模型元件或執行階段元件，做為自訂推論元件的相依性。如果您想要使用自訂推論程式碼或AWS IoT Greengrass未提供範例元件的執行階段，則必須使用自訂組件。

主題

修改公用推論元件的組態

在AWS IoT Greengrass主控台中，元件頁面會顯示該元件的預設組態。例如， TensorFlow Lite 影像分類元件的預設設定如下所示：

```
{
```

```
  "accessControl": {
    "aws.greengrass.ipc.mqttproxy": {
      "aws.greengrass.TensorFlowLiteImageClassification:mqttproxy:1": {
        "policyDescription": "Allows access to publish via topic ml/tflite/image-
classification.",
        "operations": [
          "aws.greengrass#PublishToIoTCore"
        ],
        "resources": [
          "ml/tflite/image-classification"
        ]
      }
    }
  },
  "PublishResultsOnTopic": "ml/tflite/image-classification",
  "ImageName": "cat.jpeg",
  "InferenceInterval": 3600,
  "ModelResourceKey": {
    "model": "TensorFlowLite-Mobilenet"
  }
}
```

部署公用推論元件時，您可以修改預設組態以自訂部署。如需有關每個公用推論元件可用組態參數的資訊，請參閱中AWS-提供機器學習元件的元件主題。

本節說明如何從AWS IoT Greengrass主控台部署已修改的元件。如需有關使用部署元件的資訊AWS CLI，請參閱建立部署。

若要部署修改過的公用推論元件 (主控台)

1. 登入 AWS IoT Greengrass 主控台。
2. 在導覽功能表中，選擇 [元件]。
3. 在 [元件] 頁面的 [公用元件] 索引標籤上，選擇您要建置的元件。
4. 在 [元件] 頁面上，選擇 [部署]。
5. 從新增至部署中，選擇下列其中一項：

    a. 若要將此元件合併至目標裝置上的現有部署，請選擇 [新增至現有部署]，然後選取要修訂的部署。

    b. 若要在目標裝置上建立新部署，請選擇 [建立新部署]。如果您的設備上有現有的部署，則選擇此步驟將取代現有部署。

6. 在指定目標頁面上，執行下列作業：

   a. 在部署資訊下，輸入或修改部署的易記名稱。

   b. 在部署目標下，選取部署的目標，然後選擇下一步。如果您要修訂既有部署，則無法變更部署目標。

7. 在 [選取元件] 頁面的 [公用元件] 下，確認已選取含有您修改之組態的推論元件，然後選擇下一步。

8. 在 [設定元件] 頁面上，執行下列動作：

   a. 選取推論元件，然後選擇設定元件。

   b. 在「組態更新」下，輸入您要更新的組態值。例如，在 [合併的組態] 方塊中輸入下列組態更新，將推論間隔變更為 15 秒，並指示元件尋找資料夾`custom.jpg`中指定的`/custom-ml-inference/images/`影像。

      ```
      {
        "InferenceInterval": "15",
        "ImageName": "custom.jpg",
        "ImageDirectory": "/custom-ml-inference/images/"
      }
      ```

      若要將零組件的整個組態重設為預設值，請在「重設路徑」(Reset paths) 方塊""中指定單一空白字串。

   c. 選擇確認，然後選擇下一步。

9. 在 [設定進階設定] 頁面上，保留預設組態設定，然後選擇 [下一步]。

10. 在「複查」頁面上，選擇「建置」

搭配範例推論元件使用自訂模型

如果您想要在AWS IoT Greengrass提供範例執行階段元件的執行階段中，將範例推論元件與您自己的機器學習模型搭配使用，則必須使用使用這些模型作為人工因素的元件來覆寫公用模型元件。在高階中，您可以完成下列步驟，以便搭配範例推論元件使用自訂模型：

1. 建立使用 S3 儲存貯體中的自訂模型作為成品的模型元件。您的自訂模型必須使用與您要取代的模型相同的執行階段進行訓練。
2. 修改推論元件中的`ModelResourceKey`組態參數以使用自訂模型。如需更新推論元件組態的相關資訊，請參閱 修改公用推論元件的組態

當您部署推論元件時，會AWS IoT Greengrass尋找其元件相依性的最新版本。如果元件的較新自訂版本存在於相同的AWS 帳戶和中，則會覆寫相依公用模型元件AWS 區域。

建立自訂模型元件 (主控台)

1. 將您的模型上傳到 S3 儲存貯體。如需將模型上傳到 S3 儲存貯體的相關資訊，請參閱 Amazon 簡單儲存服務使用者指南中的使用 Amazon S3 儲存貯體。

   > **Note**
   >
   > 您必須將成品存放在AWS 帳戶與元件相同的 S3 儲AWS 區域存貯體中。若 AWS IoT Greengrass要啟用存取這些成品，Greengrass 裝置角色必須允許動作。`s3:GetObject`如需有關裝置角色的詳細資訊，請參閱授權核心設備與AWS服務。

2. 在AWS IoT Greengrass主控台瀏覽功能表中，選擇 [元件]。
3. 擷取公用模型存放區元件的元件方案。

   a. 在 [元件] 頁面的 [公用元件] 索引標籤上，尋找並選擇您要為其建立新版本的公用模型元件。例如 `variant.DLR.ImageClassification.ModelStore`。

   b. 在元件頁面上，選擇 [檢視方案]，然後複製顯示的 JSON 方案。
4. 在 [元件] 頁面的 [我的元件] 索引標籤上，選擇 [建立元件]。
5. 在 [建立元件] 頁面的 [元件資訊] 下，選取 [將方案輸入為 JSON 做為元件來源]。
6. 在「配方」方塊中，貼上先前複製的元件配方。
7. 在方案中，更新下列值：

   - `ComponentVersion`：增加元件的次要版本。

     當您建立自訂元件來覆寫公用模型元件時，您必須只更新現有元件版本的次要版本。例如，如果公用元件版本是`2.1.0`，您可以使用版本建立自訂元件`2.1.1`。

   - `Manifests.Artifacts.Uri`：將每個 URI 值更新為您要使用的模型的 Amazon S3 URI。

   > **Note**
   >
   > 請勿變更元件的名稱。

8. 選擇 [建立元件]。

建立自訂模型元件 (AWS CLI)

1. 將您的模型上傳到 S3 儲存貯體。如需將模型上傳到 S3 儲存貯體的相關資訊，請參閱 Amazon 簡單儲存服務使用者指南中的使用 Amazon S3 儲存貯體。

   > **Note**
   >
   > 您必須將成品存放在AWS 帳戶與元件相同的 S3 儲AWS 區域存貯體中。若 AWS IoT Greengrass要啟用存取這些成品，Greengrass 裝置角色必須允許動作。`s3:GetObject`如需有關裝置角色的詳細資訊，請參閱授權核心設備與AWS服務。

2. 執行下列命令以擷取 public 元件的元件方案。此指令會將元件 recipe 寫入您在指令中提供的輸出檔案。視需要將擷取的 base64 編碼字串轉換為 JSON 或 YAML。

   Linux, macOS, or Unix

   ```
   aws greengrassv2 get-component \
       --arn <arn> \
       --recipe-output-format <recipe-format> \
       --query recipe \
       --output text | base64 --decode > <recipe-file>
   ```

   Windows Command Prompt (CMD)

   ```
   aws greengrassv2 get-component ^
       --arn <arn> ^
       --recipe-output-format <recipe-format> ^
       --query recipe ^
       --output text > <recipe-file>.base64

   certutil -decode <recipe-file>.base64 <recipe-file>
   ```

   PowerShell

   ```
   aws greengrassv2 get-component `
       --arn <arn> `
       --recipe-output-format <recipe-format> `
       --query recipe `
       --output text > <recipe-file>.base64
   ```

```
certutil -decode <recipe-file>.base64 <recipe-file>
```

3. 將 recipe 檔案的名稱更新為*<component-name>*-*<component-version>*，其中元件版本是新元件的目標版本。例如 `variant.DLR.ImageClassification.ModelStore-2.1.1.yaml`。
4. 在方案中，更新下列值：

   - `ComponentVersion`：增加元件的次要版本。

     當您建立自訂元件來覆寫公用模型元件時，您必須只更新現有元件版本的次要版本。例如，如果公用元件版本是`2.1.0`，您可以使用版本建立自訂元件`2.1.1`。
   - `Manifests.Artifacts.Uri`：將每個 URI 值更新為您要使用的模型的 Amazon S3 URI。

   > **Note**
   >
   > 請勿變更元件的名稱。

5. 執行下列指令，以使用您擷取和修改的方案建立新元件。

```
aws greengrassv2 create-component-version \
    --inline-recipe fileb://path/to/component/recipe
```

> **Note**
>
> 此步驟會在中的AWS IoT Greengrass服務中建立元件AWS 雲端。在將元件上傳到雲端之前，您可以使用 Greengrass CLI 在本機開發、測試和部署元件。如需詳細資訊，請參閱開發AWS IoT Greengrass元件。

如需建立元件的更多資訊，請參閱開發AWS IoT Greengrass元件。

建立自訂機器學習元件

如果您想要使用自訂推論程式碼或AWS IoT Greengrass未提供範例元件的執行階段，則必須建立自訂組件。您可以將自訂推論程式碼與AWS提供的範例機器學習模型和執行階段搭配使用，也可以使用自己的模型和執行階段來開發完全自訂的機器學習推論解決方案。如果您的模型使用AWS IoT Greengrass提供範例執行階段元件的執行階段，則您可以使用該執行階段元件，而且您只需要為您的推論程式碼和您想要使用的模型建立自訂組件。

主題

檢索公共組件的配方

您可以使用現有公用機器學習元件的方案做為範本來建立自訂元件。若要檢視最新版公用元件的元件方案，請使用主控台或AWS CLI如下所示：

- 使用主控台

  1. 在 [元件] 頁面的 [公用元件] 索引標籤上，尋找並選擇公用元件。
  2. 在元件頁面上，選擇 [檢視方案]。

- 使用 AWS CLI

  執行下列命令以擷取公用變體元件的元件方案。此命令會將元件方案寫入您在命令中提供的 JSON 或 YAML 方案檔案。

  Linux, macOS, or Unix

  ```
  aws greengrassv2 get-component \
      --arn <arn> \
      --recipe-output-format <recipe-format> \
      --query recipe \
      --output text | base64 --decode > <recipe-file>
  ```

  Windows Command Prompt (CMD)

  ```
  aws greengrassv2 get-component ^
      --arn <arn> ^
      --recipe-output-format <recipe-format> ^
      --query recipe ^
      --output text > <recipe-file>.base64

  certutil -decode <recipe-file>.base64 <recipe-file>
  ```

PowerShell

```
aws greengrassv2 get-component `
    --arn <arn> `
    --recipe-output-format <recipe-format> `
    --query recipe `
    --output text > <recipe-file>.base64

certutil -decode <recipe-file>.base64 <recipe-file>
```

取代指令中的值，如下所示：

- *<arn>*。 公共組件的 Amazon 資源名稱（ARN）。
- *<recipe-format>*。 您要用來建立 recipe 檔案的格式。支援的值為 JSON 和 YAML。
- *<recipe-file>*。 格式中的配方名稱*<component-name>*-*<component-version>*。

擷取範例元件加工品

您可以使用公用機器學習元件所使用的成品做為範本，來建立自訂元件成品，例如推論程式碼或執行階段安裝指令碼。

若要檢視公用機器學習元件中包含的範例成品，請部署公用推論元件，然後在*/greengrass/v2/*packages/artifacts-unarchived/*component-name*/*component-version*/資料夾中檢視裝置上的成品。

將元件成品上傳到 S3 儲存貯體

在建立自訂元件之前，您必須將元件成品上傳到 S3 儲存貯體，並在元件方案中使用 S3 URI。例如，若要在推論元件中使用自訂推論程式碼，請將程式碼上傳到 S3 儲存貯體。然後，您可以使用推論程式碼的 Amazon S3 URI 做為元件中的成品。

如需將內容上傳到 S3 儲存貯體的相關資訊，請參閱 Amazon 簡單儲存服務使用者指南中的使用 Amazon S3 儲存貯體。

> **Note**
>
> 您必須將成品存放在AWS 帳戶與元件相同的 S3 儲AWS 區域存貯體中。若AWS IoT Greengrass要啟用存取這些成品，Greengrass 裝置角色必須允許動作。`s3:GetObject`如需有關裝置角色的詳細資訊，請參閱授權核心設備與AWS服務。

建立自訂元件

您可以使用擷取的成品和配方來建立自訂機器學習元件。如需範例，請參閱 建立自訂推論元件。

如需建立和部署元件至 Greengrass 裝置的詳細資訊，請參閱和。開發AWS IoT Greengrass元件 將AWS IoT Greengrass元件部署到裝置

建立自訂推論元件

本節說明如何使用 DLR 影像分類元件做為範本來建立自訂推論元件。

主題

- 將您的推論程式碼上傳到 Amazon S3 儲存貯體
- 為您的推論元件建立配方
- 建立推論元件

將您的推論程式碼上傳到 Amazon S3 儲存貯體

建立您的推論程式碼，然後將其上傳到 S3 儲存貯體。如需將內容上傳到 S3 儲存貯體的相關資訊，請參閱 Amazon 簡單儲存服務使用者指南中的使用 Amazon S3 儲存貯體。

> **Note**
>
> 您必須將成品存放在AWS 帳戶與元件相同的 S3 儲AWS 區域存貯體中。若AWS IoT Greengrass要啟用存取這些成品，Greengrass 裝置角色必須允許動作。`s3:GetObject`如需有關裝置角色的詳細資訊，請參閱授權核心設備與AWS服務。

為您的推論元件建立配方

1. 執行下列命令以擷取 DLR 影像分類元件的元件方案。此命令會將元件方案寫入您在命令中提供的 JSON 或 YAML 方案檔案。

   Linux, macOS, or Unix

   ```
   aws greengrassv2 get-component \
       --arn
    arn:aws:greengrass:region:aws:components:aws.greengrass.DLRImageClassification:versions
    \
   ```

```
    --recipe-output-format JSON | YAML \
    --query recipe \
    --output text | base64 --decode > <recipe-file>
```

Windows Command Prompt (CMD)

```
aws greengrassv2 get-component ^
    --arn
 arn:aws:greengrass:region:aws:components:aws.greengrass.DLRImageClassification:versions
 ^
    --recipe-output-format JSON | YAML ^
    --query recipe ^
    --output text > <recipe-file>.base64

certutil -decode <recipe-file>.base64 <recipe-file>
```

PowerShell

```
aws greengrassv2 get-component `
    --arn
 arn:aws:greengrass:region:aws:components:aws.greengrass.DLRImageClassification:versions
 `
    --recipe-output-format JSON | YAML `
    --query recipe `
    --output text > <recipe-file>.base64

certutil -decode <recipe-file>.base64 <recipe-file>
```

<recipe-file>替換為格式的配方名稱*<component-name>*-*<component-version>*。

2. 在方案中的ComponentDependencies物件中，根據您要使用的模型和執行階段元件，執行下列一或多個動作：

   - 如果要使用 DLR 編譯的模型，請保留 DLR 組件的依賴關係。您也可以使用自訂執行階段元件的相依性來取代它，如下列範例所示。

     運行時組件

     JSON

     ```
     {
         "<runtime-component>": {
     ```

```
        "VersionRequirement": "<version>",
        "DependencyType": "HARD"
    }
}
```

YAML

```
<runtime-component>:
    VersionRequirement: "<version>"
    DependencyType: HARD
```

- 保留 DLR 映像分類模型存放區相依性，以使用預先訓練的 ResNet -50 模型，AWS提供或修改它以使用自訂模型元件。當您包含公用模型元件的相依性時，如果該元件的較新自訂版本存在於相同元件中 AWS 帳戶AWS 區域，則推論元件會使用該自訂元件。指定模型元件從屬關係，如下列範例所示。

  公共模型組件

  JSON

  ```
  {
      "variant.DLR.ImageClassification.ModelStore": {
          "VersionRequirement": "<version>",
          "DependencyType": "HARD"
      }
  }
  ```

  YAML

  ```
  variant.DLR.ImageClassification.ModelStore:
      VersionRequirement: "<version>"
      DependencyType: HARD
  ```

  自訂模型元件

  JSON

  ```
  {
      "<custom-model-component>": {
          "VersionRequirement": "<version>",
          "DependencyType": "HARD"
      }
  ```

```
}
```

YAML

```
<custom-model-component>:
    VersionRequirement: "<version>"
    DependencyType: HARD
```

3. 在ComponentConfiguration物件中，新增此元件的預設組態。您可以稍後在部署元件時修改此組態。下列摘錄顯示 DLR 影像分類元件的元件組態。

   例如，如果您使用自訂模型元件作為自訂推論元件的相依性，請修改ModelResourceKey以提供您正在使用之模型的名稱。

   JSON

```
{
  "accessControl": {
    "aws.greengrass.ipc.mqttproxy": {
      "aws.greengrass.ImageClassification:mqttproxy:1": {
        "policyDescription": "Allows access to publish via topic ml/dlr/image-
classification.",
        "operations": [
          "aws.greengrass#PublishToIoTCore"
        ],
        "resources": [
          "ml/dlr/image-classification"
        ]
      }
    }
  },
  "PublishResultsOnTopic": "ml/dlr/image-classification",
  "ImageName": "cat.jpeg",
  "InferenceInterval": 3600,
  "ModelResourceKey": {
    "armv7l": "DLR-resnet50-armv7l-cpu-ImageClassification",
    "x86_64": "DLR-resnet50-x86_64-cpu-ImageClassification",
    "aarch64": "DLR-resnet50-aarch64-cpu-ImageClassification"
  }
}
```

YAML

```
accessControl:
    aws.greengrass.ipc.mqttproxy:
        'aws.greengrass.ImageClassification:mqttproxy:1':
            policyDescription: 'Allows access to publish via topic ml/dlr/image-
classification.'
            operations:
                - 'aws.greengrass#PublishToIoTCore'
            resources:
                - ml/dlr/image-classification
PublishResultsOnTopic: ml/dlr/image-classification
ImageName: cat.jpeg
InferenceInterval: 3600
ModelResourceKey:
    armv7l: "DLR-resnet50-armv7l-cpu-ImageClassification"
    x86_64: "DLR-resnet50-x86_64-cpu-ImageClassification"
    aarch64: "DLR-resnet50-aarch64-cpu-ImageClassification"
```

4. 在Manifests物件中，提供元件建置到不同平台時所使用的人工因素和此元件組態的相關資訊，以及順利執行元件所需的任何其他資訊。下列摘錄顯示 DLR 影像分類元Manifests件中 Linux 平台之物件的組態。

   JSON

```
{
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
        "architecture": "arm"
      },
      "Name": "32-bit armv7l - Linux (raspberry pi)",
      "Artifacts": [
        {
          "URI": "s3://SAMPLE-BUCKET/sample-artifacts-directory/
image_classification.zip",
          "Unarchive": "ZIP"
        }
      ],
      "Lifecycle": {
        "Setenv": {
```

```
          "DLR_IC_MODEL_DIR":
 "{variant.DLR.ImageClassification.ModelStore:artifacts:decompressedPath}/
{configuration:/ModelResourceKey/armv7l}",
          "DEFAULT_DLR_IC_IMAGE_DIR": "{artifacts:decompressedPath}/
image_classification/sample_images/"
        },
        "run": {
          "RequiresPrivilege": true,
          "script": ". {variant.DLR:configuration:/MLRootPath}/
greengrass_ml_dlr_venv/bin/activate\npython3 {artifacts:decompressedPath}/
image_classification/inference.py"
        }
      }
    }
  ]
}
```

YAML

```
Manifests:
  - Platform:
      os: linux
      architecture: arm
    Name: 32-bit armv7l - Linux (raspberry pi)
    Artifacts:
      - URI: s3://SAMPLE-BUCKET/sample-artifacts-directory/
image_classification.zip
        Unarchive: ZIP
    Lifecycle:
      Setenv:
        DLR_IC_MODEL_DIR:
 "{variant.DLR.ImageClassification.ModelStore:artifacts:decompressedPath}/
{configuration:/ModelResourceKey/armv7l}"
        DEFAULT_DLR_IC_IMAGE_DIR: "{artifacts:decompressedPath}/
image_classification/sample_images/"
      run:
        RequiresPrivilege: true
        script: |-
          . {variant.DLR:configuration:/MLRootPath}/greengrass_ml_dlr_venv/bin/
activate
          python3 {artifacts:decompressedPath}/image_classification/inference.py
```

如需建立元件配方的詳細資訊，請參閱AWS IoT Greengrass元件配方參考。

建立推論元件

使用AWS IoT Greengrass主控台或AWS CLI使用您剛定義的方案建立元件。建立元件之後，您可以將其部署在裝置上執行推論。如需如何部署推論元件的範例，請參閱教學課程：使 TensorFlow 用 Lite 執行範例影像分類推論。

建立自訂推論元件 (主控台)

1. 登入 AWS IoT Greengrass 主控台。
2. 在導覽功能表中，選擇 [元件]。
3. 在 [元件] 頁面的 [我的元件] 索引標籤上，選擇 [建立元件]。
4. 在 [建立元件] 頁面的 [元件資訊] 下，選取 [以 JSON 格式輸入方案] 或 [將方案輸入為 YAML] 做為元件來源。
5. 在「配方」方塊中，輸入您建立的自訂配方。
6. 按一下建立元件。

建立自訂推論元件 () AWS CLI

執行下列命令，以使用您建立的方案建立新的自訂元件。

```
aws greengrassv2 create-component-version \
    --inline-recipe fileb://path/to/recipe/file
```

> **Note**
>
> 此步驟會在中的AWS IoT Greengrass服務中建立元件AWS 雲端。在將元件上傳到雲端之前，您可以使用 Greengrass CLI 在本機開發、測試和部署元件。如需更多詳細資訊，請參閱 開發AWS IoT Greengrass元件。

機器學習推論疑難排解

使用本節中的疑難排解資訊和解決方案，協助解決機器學習元件的問題。如需公用機器學習推論元件，請參閱下列元件記錄檔中的錯誤訊息：

Linux or Unix

- */greengrass/v2*/logs/aws.greengrass.DLRImageClassification.log
- */greengrass/v2*/logs/aws.greengrass.DLRObjectDetection.log
- */greengrass/v2*/logs/aws.greengrass.TensorFlowLiteImageClassification.log
- */greengrass/v2*/logs/aws.greengrass.TensorFlowLiteObjectDetection.log

Windows

- *C:\greengrass\v2*\logs\aws.greengrass.DLRImageClassification.log
- *C:\greengrass\v2*\logs\aws.greengrass.DLRObjectDetection.log
- *C:\greengrass\v2*\logs\aws.greengrass.TensorFlowLiteImageClassification.log
- *C:\greengrass\v2*\logs\aws.greengrass.TensorFlowLiteObjectDetection.log

如果元件已正確安裝，則元件記錄檔會包含其用於推論之程式庫的位置。

問題

無法擷取程式庫

當安裝程式指令碼無法在 Raspberry Pi 裝置上部署期間下載所需的程式庫時，會發生下列錯誤。

```
Err:2 http://raspbian.raspberrypi.org/raspbian buster/main armhf python3.7-dev armhf
 3.7.3-2+deb10u1
404 Not Found [IP: 93.93.128.193 80]
E: Failed to fetch http://raspbian.raspberrypi.org/raspbian/pool/main/p/python3.7/
libpython3.7-dev_3.7.3-2+deb10u1_armhf.deb 404 Not Found [IP: 93.93.128.193 80]
```

再次執行`sudo apt-get update`並部署您的元件。

Cannot open shared object file

當安裝程式指令碼在 Raspberry Pi 裝置上部署`opencv-python`期間無法下載必要的相依性時，您可能會看到類似下列的錯誤。

```
ImportError: libopenjp2.so.7: cannot open shared object file: No such file or directory
```

執行下列命令以手動安裝的相依性`opencv-python`：

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
 libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

Error: ModuleNotFoundError: No module named '<library>'

當 ML 執行階段程式庫`variant.DLR.log`或其相依性未正確安裝時，您可能會在 ML 執行階段元件記錄檔 (或`variant.TensorFlowLite.log`) 中看到這個錯誤。在下列情況下，可能會發生此錯誤：

- 如果您使用預設為啟用的`UseInstaller`選項，則此錯誤表示 ML 執行階段元件無法安裝執行階段或其相依性。請執行下列操作：
  1. 設定 ML 執行階段元件以停用`UseInstaller`此選項。
  2. 安裝 ML 執行階段及其相依性，並讓執行 ML 元件的系統使用者可以使用它們。如需詳細資訊，請參閱下列內容：
     - DLR 執行階段 UseInstaller 選項
     - TensorFlow精簡運行時 UseInstaller 選項
- 如果您未使用此選`UseInstaller`項，則此錯誤表示 ML 執行階段或其相依性並未針對執行 ML 元件的系統使用者安裝。請執行下列操作：
  1. 檢查是否已為執行 ML 元件的系統使用者安裝程式庫。將 *`ggc_user`* 取代為系統使用者的名稱，並以要檢查的程式庫名稱取代 *`tflite_runtime`*。

Linux or Unix

```
sudo -H -u ggc_user bash -c "python3 -c 'import tflite_runtime'"
```

Windows

```
runas /user:ggc_user "py -3 -c \"import tflite_runtime\""
```

2. 如果未安裝程式庫，請為該使用者安裝它。將 *ggc_user* 取代為系統使用者的名稱，並以程式庫的名稱取代 *tflite_runtime*。

   Linux or Unix

   ```
   sudo -H -u ggc_user bash -c "python3 -m pip install --user tflite_runtime"
   ```

   Windows

   ```
   runas /user:ggc_user "py -3 -m pip install --user tflite_runtime"
   ```

   如需有關每個 ML 執行階段之相依性的詳細資訊，請參閱下列內容：

   - DLR 執行階段 UseInstaller 選項
   - TensorFlow精簡運行時 UseInstaller 選項

3. 如果問題仍然存在，請為其他使用者安裝程式庫，以確認此裝置是否可以安裝程式庫。使用者可以是例如，您的使用者、root 使用者或管理員使用者。如果您無法為任何使用者成功安裝程式庫，您的裝置可能不支援該程式庫。請參閱程式庫的文件，以檢閱需求並疑難排解安裝問題。

未偵測到具有 CUDA 功能的裝置

使用 GPU 加速時，您可能會看到下列錯誤。執行下列命令以啟用 Greengrass 使用者的 GPU 存取權。

```
sudo usermod -a -G video ggc_user
```

沒有這樣的文件或目錄

下列錯誤表示執行階段元件無法正確設定虛擬環境：

- *MLRootPath*/greengrass_ml_dlr_conda/bin/conda: No such file or directory

- *MLRootPath*/greengrass_ml_dlr_venv/bin/activate: No such file or directory
- *MLRootPath*/greengrass_ml_tflite_conda/bin/conda: No such file or directory
- *MLRootPath*/greengrass_ml_tflite_venv/bin/activate: No such file or directory

檢查記錄檔，確定所有執行階段相依性都已正確安裝。如需安裝程式指令碼所安裝之程式庫的詳細資訊，請參閱下列主題：

- DLR 執行階段
- TensorFlow 精簡版運行

默認情況下 *ML RootPath* 設置為*/greengrass/v2*/work/*component-name*/greengrass_ml。若要變更此位置，請直接在部署中包含DLR 執行階段或TensorFlow 精簡版運行執行階段元件，並在組態合併更新中指定MLRootPath參數的修改值。如需有關配置元件的更多資訊，請參閱更新零組件組態。

> **Note**
>
> 對於 DLR 組件 v1.3.x，您可以在推論元件的組態中設定MLRootPath參數，預設值為。$HOME/greengrass_ml

RuntimeError: module compiled against API version 0xf but this version of NumPy is <version>

您可能會看到下列錯誤，當您運行機器學習推論上樹莓派作業系統靶心樹莓派.

```
RuntimeError: module compiled against API version 0xf but this version of numpy is 0xd
ImportError: numpy.core.multiarray failed to import
```

發生此錯誤是因為樹莓派 OS 靶心包含 NumPy比 OpenCV 需要的版本的早期版本。若要修正此問題，請執行下列命令以升級 NumPy 至最新版本。

```
pip3 install --upgrade numpy
```

picamera.exc.PiCameraError: Camera is not enabled

您可能會看到下列錯誤，當您執行機器學習推論樹莓派上運行樹莓派作業系統靶心.

```
picamera.exc.PiCameraError: Camera is not enabled. Try running 'sudo raspi-config' and
 ensure that the camera has been enabled.
```

之所以發生此錯誤，是因為樹莓派 OS 靶心包含與 ML 元件不相容的新相機堆疊。若要修正此問題，請啟用舊式攝影機堆疊。

啟用舊式相機堆疊

1. 運行以下命令以打開樹莓派配置工具。

   ```
   sudo raspi-config
   ```

2. 選取介面選項。
3. 選取「舊式相機」以啟用舊式相機堆疊。
4. 重新啟動 Raspberry Pi。

記憶體錯誤

當裝置沒有足夠的記憶體且元件程序中斷時，通常會發生下列錯誤。

- `stderr. Killed.`
- `exitCode=137`

我們建議您至少使用 500 MB 的記憶體來部署公用機器學習推論元件。

磁碟空間錯誤

該`no space left on device`錯誤通常發生在設備沒有足夠的存儲空間時。請確定裝置上有足夠的可用磁碟空間，然後再次部署元件。我們建議至少有 500 MB 的可用磁碟空間來部署公用機器學習推論元件。

逾時錯誤

公用機器學習元件會下載大於 200 MB 的大型機器學習模型檔案。如果下載在部署期間逾時，請檢查網際網路連線速度，然後重試部署。

使用 Greengrass 管理核心裝置 AWS Systems Manager

> **Note**
>
> AWS IoT Greengrass目前在 Windows 核心裝置上不支援此功能。

Systems Manager 是可用來檢視和控制基礎設施的AWS服務AWS，包括 Amazon EC2 執行個體、現場部署伺服器和虛擬機器 (VM) 和邊緣裝置。Systems Manager 可讓您檢視作業資料、將作業工作自動化，以及維護安全性與合規性。當您向 Systems Manager 註冊機器時，它稱為受管理節點。如需詳細資訊，請參閱《AWS Systems Manager 使用者指南》中的什麼是 AWS Systems Manager？。

AWS Systems Manager代理程式 (Systems Manager 代理程式) 是您可以在裝置上安裝的軟體，讓 Systems Manager 更新、管理及設定這些裝置。若要在 Greengrass 核心裝置上安裝系統管理員代理程式，請部署系統管理員代理程式元件。當您第一次部署系統管理員代理程式時，它會將核心裝置註冊為系統管理的節點。系統管理員代理程式會在裝置上執行，以啟用與中的 Systems Manager 服務通訊AWS 雲端。如需如何安裝及設定系統管理員代理程式元件的相關資訊，請參閱安裝 AWS Systems Manager 代理程式。

Systems Manager 的工具和功能稱為功能。Greengrass 核心設備支持所有 Systems Manager 功能。如需有關這些功能以及如何使用 Systems Manager 來管理核心裝置的詳細資訊，請參閱使用AWS Systems Manager者指南中的 Systems Manager 功能。

AWS Systems Manager為系統管理員受管節點提供標準執行個體層級和進階執行個體層。如果您是第一次使用 Systems Manager，請從標準執行個體層開始。在標準執行個體層上，您可以AWS 區域在. AWS 帳戶 如果您需要在單一帳戶和區域中註冊 1,000 個以上的受管節點，或者您需要使用工作階段管理員功能，請使用進階執行個體層。如需詳細資訊，請參閱AWS Systems Manager使用指南中的設定執行個體層級。

主題

- 安裝 AWS Systems Manager 代理程式
- 解除安裝 AWS Systems Manager 代理程式

安裝 AWS Systems Manager 代理程式

AWS Systems Manager代理程式 (Systems Manager 代理程式) 是您安裝的 Amazon 軟體，可讓 Systems Manager 更新、管理和設定 Greengrass 核心裝置、Amazon EC2 執行個體和其他資源。代

理程式會處理和執行中 Systems Manager 服務的要求AWS 雲端。然後，代理程式會將狀態和執行階段資訊傳送回系 Systems Manager 服務。如需詳細資訊，請參閱AWS Systems Manager使用指南中的關於系統管理員代理程式。

AWS提供 Systems Manager 代理程式做為 Greengrass 元件，您可以將其部署到 Greengrass 核心裝置，以便使用系統管理員來管理這些元件。系統管理員代理程式元件會安裝系統管理員代理程式軟體，並將核心裝置註冊為系統管理員中的受管理節點。請依照此頁面中的步驟完成必要條件，並將 Systems Manager Agent 元件部署至核心裝置或核心裝置群組。

主題

步驟 1：完成一般 Systems Manager 設定步驟

如果您尚未這麼做，請完成的一般設定步驟AWS Systems Manager。如需詳細資訊，請參閱AWS Systems Manager使用者指南中的完整一般 Systems Manager 設定步驟。

步驟 2：為 Systems Manager 建立 IAM 服務角色

系統管理員代理程式會使用 AWS Identity and Access Management (IAM) 服務角色與之通訊AWS Systems Manager。Systems Manager 擔任此角色，以便在每個核心裝置上啟用 Systems Manager 功能。當您部署元件時，Systems Manager 代理程式元件也會使用此角色，將核心裝置註冊為系統管理員管理的節點。如果您尚未這麼做，請建立要使用的 Systems Manager 代理程式元件的 Systems Manager 服務角色。如需詳細資訊，請參閱使AWS Systems Manager用指南中的為邊緣裝置建立 IAM 服務角色。

步驟 3：將權限新增至權杖交換角色

Greengrass 核心裝置使用 IAM 服務角色 (稱為權杖交換角色) 與服務互動。AWS每個核心裝置都有您在安裝 AWS IoT Greengrass Core 軟體時所建立的權杖交換角色。許多 Greengrass 元件 (例如系統管理員代理程式) 都需要此角色的額外權限。系統管理員代理程式元件需要下列權限，其中包括使用您在中建立之角色的權限步驟 2：為 Systems Manager 建立 IAM 服務角色。

```
{
```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "iam:PassRole"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:iam::account-id:role/SSMServiceRole"
      ]
    },
    {
      "Action": [
        "ssm:AddTagsToResource",
        "ssm:RegisterManagedInstance"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

如果您尚未這麼做，請將這些權限新增至核心裝置的權杖交換角色，以允許 Systems Manager 代理程式運作。您可以將新政策添加到令牌交換角色以授予此權限。

若要將權限新增至權杖交換角色 (主控台)

1. 在 IAM 主控台導覽功能表中，選擇 [角色]。
2. 選擇安裝 AWS IoT Greengrass Core 軟體時設定為權杖交換角色的 IAM 角色。如果您在安裝 AWS IoT Greengrass Core 軟體時未指定權杖交換角色的名稱，則會建立名為的角色`GreengrassV2TokenExchangeRole`。
3. 在 [權限] 下，選擇 [新增權限]，然後選擇 [附加原則]。
4. 選擇建立政策。[建立原則] 頁面會在新的瀏覽器索引標籤中開啟。
5. 在 Create policy (建立政策) 頁面上，執行下列動作：

   a. 選擇 JSON 以開啟 JSON 編輯器。

   b. 將以下 政策貼到 JSON 編輯器。以您在步驟 2：為 Systems Manager 建立 IAM 服務角色中建立的服務角色名稱取代 *SSM ServiceRole*。

      ```
      {
      ```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "iam:PassRole"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:iam::account-id:role/SSMServiceRole"
      ]
    },
    {
      "Action": [
        "ssm:AddTagsToResource",
        "ssm:RegisterManagedInstance"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

c. 選擇下一步：標籤。

d. 選擇下一步：檢閱。

e. 為政策輸入 Name (名稱)，例如 **GreengrassSSMAgentComponentPolicy**。

f. 選擇建立政策。

g. 切換到上一個瀏覽器標籤，您可以在其中打開令牌交換角色。

6. 在 [新增權限] 頁面上，選擇 [重新整理] 按鈕，然後選取您在上一個步驟中建立的 Greengrass 系統管理員代理程式原則。

7. 選擇連接政策。

使用此權杖交換角色的核心裝置現在具有與 Systems Manager 服務互動的權限。

若要將權限新增至權杖交換角色 (AWS CLI)

若要新增授與使用 Systems Manager 之權限的原則

1. 創建一個名為的文件，`ssm-agent-component-policy.json`並將以下 JSON 複製到該文件中。以您在步驟 2：為 Systems Manager 建立 IAM 服務角色中建立的服務角色名稱取代 *SSM ServiceRole*。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "iam:PassRole"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:iam::account-id:role/SSMServiceRole"
      ]
    },
    {
      "Action": [
        "ssm:AddTagsToResource",
        "ssm:RegisterManagedInstance"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

2. 執行下列命令，從中的策略文件建立策略ssm-agent-component-policy.json。

Linux or Unix

```
aws iam create-policy \
  --policy-name GreengrassSSMAgentComponentPolicy \
  --policy-document file://ssm-agent-component-policy.json
```

Windows Command Prompt (CMD)

```
aws iam create-policy ^
  --policy-name GreengrassSSMAgentComponentPolicy ^
  --policy-document file://ssm-agent-component-policy.json
```

PowerShell

```
aws iam create-policy `
  --policy-name GreengrassSSMAgentComponentPolicy `
```

```
  --policy-document file://ssm-agent-component-policy.json
```

從輸出中的政策中繼資料複製政策 Amazon 資源名稱 (ARN)。您可以使用此 ARN 將此原則附加至下一個步驟中的核心裝置角色。

3. 執行下列命令，將原則附加至權杖交換角色。

   - 將 *GreenRassv2 TokenExchangeRole* 取代為您在安裝核心軟體時指定的權杖交換角色名稱。AWS IoT Greengrass如果您在安裝 AWS IoT Greengrass Core 軟體時未指定權杖交換角色的名稱，則會建立名為的角色`GreengrassV2TokenExchangeRole`。
   - 將原則 ARN 取代為上一個步驟的 ARN。

   Linux or Unix

   ```
   aws iam attach-role-policy \
     --role-name GreengrassV2TokenExchangeRole \
     --policy-arn
    arn:aws:iam::123456789012:policy/GreengrassSSMAgentComponentPolicy
   ```

   Windows Command Prompt (CMD)

   ```
   aws iam attach-role-policy ^
     --role-name GreengrassV2TokenExchangeRole ^
     --policy-arn
    arn:aws:iam::123456789012:policy/GreengrassSSMAgentComponentPolicy
   ```

   PowerShell

   ```
   aws iam attach-role-policy `
     --role-name GreengrassV2TokenExchangeRole `
     --policy-arn
    arn:aws:iam::123456789012:policy/GreengrassSSMAgentComponentPolicy
   ```

   如果命令沒有輸出，則成功。使用此權杖交換角色的核心裝置現在具有與 Systems Manager 服務互動的權限。

步驟 4：部署系統管理員代理程式元件

完成下列步驟，以部署和設定系統管理員代理程式元件。您可以將元件部署到單一核心裝置或核心裝置群組。

若要部署系統管理員代理程式元件 (主控台)

1. 在AWS IoT Greengrass主控台瀏覽功能表中，選擇 [元件]。
2. 在 [元件] 頁面上，選擇 [公用元件] 索引標籤，然後選擇aws.greengrass.SystemsManagerAgent。
3. 在 aws.greengrass.SystemsManagerAgent頁面中，選擇部署。
4. 從 [新增至部署] 中，選擇要修訂的現有部署，或選擇建立新部署，然後選擇 [下一步]。
5. 如果您選擇建立新部署，請為部署選擇目標核心裝置或物件群組。在 [指定目標] 頁面的 [部署目標] 下，選擇核心裝置或物件群組，然後選擇 [下一步]。
6. 在 [選取元件] 頁面上，確認已選取aws.greengrass.SystemsManagerAgent元件，然後選擇 [下一步]。
7. 在 [設定元件] 頁面上，選取 aws.greengrass.SystemsManagerAgent，然後執行下列動作：

   a. 選擇 設定元件。

   b. 在設定aws.greengrass.SystemsManagerAgent強制回應的組態更新下，在要合併的組態中，輸入下列組態更新。以您在步驟 2：為 Systems Manager 建立 IAM 服務角色中建立的服務角色名稱取代 *SSM ServiceRole*。

```
{
  "SSMRegistrationRole": "SSMServiceRole",
  "SSMOverrideExistingRegistration": false
}
```

> **Note**
>
> 如果核心裝置已執行以混合式啟動方式註冊的系統管理員代理程式，請變更SSMOverrideExistingRegistration為true。此參數指定當系統管理員代理程式已在具有混合式啟動的裝置上執行時，系統管理員代理程式元件是否註冊核心裝置。
>
> 您也可以指定標籤 (SSMResourceTags)，以新增至系統管理員管理的節點，此節點是系統管理員代理程式元件為核心裝置建立的。如需詳細資訊，請參閱系統管理員代理程式元件組態

c. 選擇「確認」以關閉強制回應，然後選擇「下一步」。

8. 在設定進階設定頁面上，保留預設組態設定，然後選擇 下一步。

9. 在 Review (檢閱) 頁面，選擇 Deploy (部署)。

部署最多可能需要一分鐘的時間才能完成。

若要部署系統管理員代理程式元件 (AWS CLI)

若要部署 Systems Manager 代理程式元件，請建立包含在`components`物件`aws.greengrass.SystemsManagerAgent`中的部署文件，並指定元件的組態更新。遵循中建立部署的指示建立新部署或修訂既有部署。

下列範例部分部署文件會指定使用名為的服務角色`SSMServiceRole`。以您在步驟 2：為 Systems Manager 建立 IAM 服務角色中建立的服務角色名稱取代 *`SSM ServiceRole`*。

```
{
  ...,
  "components": {
    ...,
    "aws.greengrass.SystemsManagerAgent": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "merge": "{\"SSMRegistrationRole\":\"SSMServiceRole\",
\"SSMOverrideExistingRegistration\":false}"
      }
    }
  }
}
```

> **Note**
>
> 如果核心裝置已執行以混合式啟動方式註冊的系統管理員代理程式，請變更`SSMOverrideExistingRegistration`為`true`。此參數指定當系統管理員代理程式已在具有混合式啟動的裝置上執行時，系統管理員代理程式元件是否註冊核心裝置。
> 您也可以指定標籤 (`SSMResourceTags`)，以新增至系統管理員管理的節點，此節點是系統管理員代理程式元件為核心裝置建立的。如需詳細資訊，請參閱系統管理員代理程式元件組態

可能需要幾分鐘才能完成部署。您可以使用此AWS IoT Greengrass服務來檢查部署的狀態，也可以檢查AWS IoT Greengrass核心軟體記錄檔和系統管理員代理程式元件記錄檔，以確認系統管理員代理程式是否成功執行。如需詳細資訊，請參閱下列內容：

- 檢查部署狀態
- 監控AWS IoT Greengrass日誌
- 檢視AWS Systems Manager使用指南中的系統管理員代理程式記錄

如果部署失敗或系統管理員代理程式未執行，您可以疑難排解每個核心裝置上的部署問題。如需詳細資訊，請參閱下列內容：

- 疑難排 AWS IoT Greengrass V2
- 疑難排解AWS Systems Manager使用指南中的系統管理員代理

步驟 5：透過 Systems Manager 驗證核心裝置註冊

系統管理員代理程式元件執行時，會將核心裝置註冊為系統管理員中的受管理節點。您可以使用AWS IoT Greengrass主控台、Systems Manager 主控台和 Systems Manager API 來驗證核心裝置是否已註冊為受管理節點。受管節點在AWS主控台和 API 的一部分中也稱為執行個體。

驗證核心裝置註冊 (AWS IoT Greengrass主控台)

1. 在AWS IoT Greengrass主控台瀏覽功能表中，選擇 [核心裝置]。
2. 選擇要驗證的核心裝置。
3. 在核心裝置的詳細資料頁面上，找到AWS Systems Manager執行個體屬性。如果存在此內容並顯示 Systems Manager 主控台的連結，則核心裝置會註冊為受管理節點。

   您也可以找到 AWS Systems Managerping 狀態屬性，以檢查核心裝置上系統管理員代理程式的狀態。當狀態為 [線上] 時，您可以使用系統管理員來管理核心裝置。

驗證核心裝置註冊 (Systems Manager 主控台)

1. 在 [Systems Manager] 主控台瀏覽功能表中，選擇 [叢集管理
2. 在受管節點下，執行下列動作：

   a. 新增「來源」類型為的篩選器AWS::IoT::Thing。
   b. 添加一個過濾器，其中源 ID 是要驗證的核心設備的名稱。

3. 在「受管理的節點」表格中找到核心裝置。如果核心裝置位於表格中，則會註冊為受管理節點。

   您也可以找到系統管理員代理程式偵測狀態屬性，以檢查核心裝置上系統管理員代理程式的狀態。當狀態為 [線上] 時，您可以使用系統管理員來管理核心裝置。

驗證核心裝置註冊 (AWS CLI)

- 使用此DescribeInstanceInformation作業可取得符合您指定之篩選器的受管理節點清單。執行下列命令以驗證核心裝置是否已註冊為受管理節點。以核心裝置的名稱取*MyGreengrassCore*代以進行驗證。

  ```
  aws ssm describe-instance-information --filter
   Key=SourceIds,Values=MyGreengrassCore Key=SourceTypes,Values=AWS::IoT::Thing
  ```

  回應包含符合篩選器的受管節點清單。如果清單包含受管理節點，則核心裝置會註冊為受管理節點。您也可以在回應中找到有關核心裝置受管理節點的其他資訊。如果`PingStatus`屬性是`Online`，您可以使用系統管理員來管理核心裝置。

在您確認核心裝置已在 Systems Manager 中註冊為受管理節點之後，您就可以使用 Systems Manager 主控台和 API 來管理該核心裝置。如需有關可用來管理 Greengrass 核心裝置之系統管理員功能的詳細資訊，請參閱使用者指南中的 Systems Manager 功能。AWS Systems Manager

解除安裝 AWS Systems Manager 代理程式

如果您不想再使用管理 Greengrass 核心裝置AWS Systems Manager，您可以從 Systems Manager 取消註冊核心裝置，然後從裝置解除安裝AWS Systems Manager代理程式 (Systems Manager 代理程式)。

您可以隨時重新註冊核心裝置。若要這麼做，請再次部署系統管理員代理程式元件，以便在安裝時向 Systems Manager 註冊核心裝置。Systems Manager 會存放已取消註冊之核心裝置的命令歷史記錄 30 天。

主題

- 步驟 1：從 Systems Manager 中取消註冊核心裝置
- 步驟 2：解除安裝系統管理員代理程式元件
- 步驟 3：解除安裝 Systems Manager Agent 軟體

步驟 1：從 Systems Manager 中取消註冊核心裝置

您可以使用 Systems Manager 主控台或 API 取消註冊核心裝置。如需詳細資訊，請參閱AWS Systems Manager使用指南中的取消註冊受管理節點。

步驟 2：解除安裝系統管理員代理程式元件

取消註冊核心裝置之後，請從裝置解除安裝系統管理員代理程式元件。若要從 Greengrass 核心裝置移除元件，請修訂安裝該元件的部署，然後從部署中移除元件。當AWS IoT Greengrass核心裝置的部署沒有指定該元件時，Core 軟體會解除安裝元件。如需詳細資訊，請參閱將AWS IoT Greengrass元件部署到裝置。

解除安裝系統管理員代理程式元件 (主控台)

1. 在AWS IoT Greengrass主控台瀏覽功能表中，選擇 [核心裝置]。
2. 選擇您要解除安裝系統管理員代理程式元件的核心裝置。
3. 在核心裝置詳細資訊頁面上，選擇部署索引標籤。
4. 選擇將系統管理員代理程式元件部署到核心裝置的部署。
5. 在部署詳細資料頁面上，選擇「修訂」。
6. 在「修訂部署」模式中，選擇「修訂部署」。
7. 在步驟 1：指定目標中，選擇下一步。
8. 在步驟 2：選取元件中，清除aws.greengrass.SystemsManagerAgent元件的選取，然後選擇下一步。
9. 在步驟 3：設定元件中，選擇下一步。
10. 在 [步驟 4：設定進階設定] 中，選擇 [下一步]。
11. 在 [步驟 5：複查] 中，選擇 [部署]。

解除安裝系統管理員代理程式元件 (CLI)

若要解除安裝系統管理員代理程式元件，請修訂部署該元件的部署，然後將其從部署中移除。如需詳細資訊，請參閱修訂部署。

部署需要幾分鐘的時間來完成。您可以使用此AWS IoT Greengrass服務檢查部署的狀態。如需詳細資訊，請參閱檢查部署狀態。

步驟 3：解除安裝 Systems Manager Agent 軟體

移除系統管理員代理程式元件之後，系統管理員代理程式軟體會繼續在核心裝置上執行。若要移除系統管理員代理程式軟體，您可以在核心裝置上執行命令。如需詳細資訊，請參閱AWS Systems Manager 使用指南中的從 Linux 執行個體解除安裝系統管理員代理程式

AWS IoT Greengrass 中的安全性

雲端安全是 AWS 最重視的一環。身為 AWS 客戶的您，將能從資料中心和網路架構的建置中獲益，以滿足組織最為敏感的安全要求。

安全是 AWS 與您共同肩負的責任。共同的責任模式將其稱為雲端的安全性和雲端中的安全性：

- 雲端本身的安全 – AWS 負責保護執行 AWS 雲端 內 AWS 服務的基礎設施。AWS 提供的服務，也可讓您安全使用。第三方稽核人員會定期測試和驗證我們安全性的有效性，作為 AWS 合規計畫的一部分。若要了解適用於 AWS IoT Greengrass 的合規計畫，請參閱合規計畫的 AWS 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、您公司的要求和適用法律和法規。

使用 AWS IoT Greengrass 時，您也必須負責保護裝置、本機網路連線和私有金鑰的安全。

本文件有助於您了解如何在使用 AWS IoT Greengrass 時套用共同責任模型。下列主題說明如何將 AWS IoT Greengrass 設定為達到您的安全及法規遵循目標。您也會了解如何使用其他 AWS 服務來協助監控並保護 AWS IoT Greengrass 資源。

主題

AWS IoT Greengrass 中的資料保護

AWS 共同的責任模型適用於 AWS IoT Greengrass 中的資料保護。如此模型所述，AWS 負責保護執行所有 AWS 雲端 的全球基礎設施。您負責維護在此基礎設施上託管內容的控制權。您也必須負責您所使用 AWS 服務 的安全組態和管理任務。如需有關資料隱私權的更多相關資訊，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，建議您使用 AWS IAM Identity Center 或 AWS Identity and Access Management (IAM) 保護 AWS 帳戶 憑證，並設定個人使用者。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 AWS CloudTrail 設定 API 和使用者活動日誌記錄。
- 使用 AWS 加密解決方案，以及 AWS 服務 內的所有預設安全控制項。
- 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。
- 如果您在透過命令列介面或 API 存取 AWS 時，需要 FIPS 140-2 驗證的加密模組，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) 140-2 概觀。

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 Name(名稱) 欄位。這包括當您使用 AWS IoT Greengrass 或使用主控台、API、AWS CLI 或 AWS 開發套件的其他 AWS 服務。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。

如需有關保護機密資訊的更多資訊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在傳輸中 (透過網際網路或區域網路) 和靜態 (儲存在) 時，會使用加密來保護資料。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，以進一步處理。如需有關其他 AWS 服務上資料加密的詳細資訊，請參閱該服務的安全性文件。

主題

傳輸中加密

AWS IoT Greengrass有兩種傳輸資料的通訊模式：

- the section called “透過網際網路傳輸資料”。 Greengrass 核心與之間的通訊AWS IoT Greengrass通訊會加密。
- the section called “核心裝置上的資料”。 Greengrass 核心裝置上元件之間的通訊並不會加密。

透過網際網路傳輸資料

AWS IoT Greengrass 使用 Transport Layer Security (TLS) 加密透過網際網路的所有通訊。傳送至 AWS 雲端是通過使用 MQTT 或 HTTPS 通訊協定的 TLS 連線傳送的，因此在默認情況下是安全的。AWS IoT Greengrass使用AWS IoT傳輸安全模型。如需詳細資訊，請參閱「」傳輸安全性中的AWS IoT Core開發人員指南。

核心裝置上的資料

AWS IoT Greengrass 不會加密 Greengrass 核心裝置本機上交換的資料，因為資料不會離開裝置。這包括用户定義的組件之間的通信，AWS IoT設備 SDK 和公共組件（如流管理器）。

靜態加密

AWS IoT Greengrass 存放您的資料：

- the section called “磁盤區內的靜態資料AWS 雲端”。 此資料已加密。

- the section called “Greengrass 核心上的靜態資料”。 這些資料不會加密 (您的秘密的本機副本除外)。

磁盤區內的靜態資料AWS 雲端

AWS IoT Greengrass會加密存放在AWS 雲端。此資料會使用由AWS IoT Greengrass 管理的 AWS KMS 金鑰進行保護。

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 (若啟用) 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 AWS Secrets Manager 擷取之秘密的本機副本。如需詳細資訊，請參閱 。Secret Manager元件。

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在以下情況下使用公鑰和私鑰：

- 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。

Greengrass 核心裝置支援使用檔案系統許可的私有金鑰儲存體或硬體安全模組。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。

硬體安全整合

> **Note**
>
> 此功能適用於 v2.5.3 及更高版 Greeng rass 核組件。 AWS IoT Greengrass目前在 Windows 核心裝置上不支援此功能。

您可以透過 PKCS #11 介面將AWS IoT Greengrass核心軟體設定為使用硬體安全性模組 (HSM)。此功能使您可以安全地存儲設備的私鑰和證書，以便它們不會在軟件中暴露或複製。您可以將私密金鑰和憑證儲存在硬體模組 (例如 HSM 或信任平台模組 (TPM) 上。

AWS IoT Greengrass核心軟體使用私密金鑰和 X.509 憑證來驗證AWS IoT與和AWS IoT Greengrass服務的連線。密碼管理員元件使用此私密金鑰來安全地加密和解密您部署到 Greengrass 核心裝置的密碼。當您將核心裝置設定為使用 HSM 時，這些元件會使用您儲存在 HSM 中的私密金鑰和憑證。

Moquette MQTT 代理程式元件也會儲存其本機 MQTT 伺服器憑證的私密金鑰。此元件會將私密金鑰儲存在元件的工作資料夾中的裝置的檔案系統上。目前AWS IoT Greengrass不支援在 HSM 中儲存此私密金鑰或憑證。

> **Tip**
>
> 在AWS合作夥伴裝置目錄中搜尋支援此功能的裝置。

主題

要求

您必須符合下列需求才能在 Greengrass 核心裝置上使用 HSM：

- 核心設備上安裝的 Greengrass 核 v2.5.3 或更高版本。在核心裝置上安裝 AWS IoT Greengrass Core 軟體時，您可以選擇相容的版本。
- 安裝在核心裝置上的 PKCS #11 提供者元件。當您在核心裝置上安裝 AWS IoT Greengrass Core 軟體時，您可以下載並安裝此元件。
- 一種硬體安全性模組，支援 PKCS #1 v1.5 簽章配置，以及具有 RSA-2048 金鑰大小 (或更大) 或 ECC 金鑰的 RSA 金鑰。

> **Note**
>
> 若要搭配 ECC 金鑰使用硬體安全性模組，您必須使用 Greengrass 核心 v2.5.6 或更新版本。
> 要使用硬件安全模塊和密碼管理器，您必須使用帶有 RSA 密鑰的硬件安全模塊。

- AWS IoT Greengrass核心軟體可以在執行階段 (使用 libdl) 載入的 PKCS #11 提供者程式庫，以呼叫 PKCS #11 函數。PKCS #11 提供者程式庫必須實作下列 PKCS #11 API 作業：
  - `C_Initialize`
  - `C_Finalize`
  - `C_GetSlotList`
  - `C_GetSlotInfo`
  - `C_GetTokenInfo`
  - `C_OpenSession`
  - `C_GetSessionInfo`
  - `C_CloseSession`
  - `C_Login`
  - `C_Logout`
  - `C_GetAttributeValue`
  - `C_FindObjectsInit`
  - `C_FindObjects`
  - `C_FindObjectsFinal`
  - `C_DecryptInit`
  - `C_Decrypt`
  - `C_DecryptUpdate`
  - `C_DecryptFinal`
  - `C_SignInit`
  - `C_Sign`
  - `C_SignUpdate`
  - `C_SignFinal`
  - `C_GetMechanismList`

  - `C_GetMechanismInfo`
  - `C_GetInfo`
  - `C_GetFunctionList`
- 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。
- 您必須將私密金鑰和憑證儲存在相同的插槽中，而且如果 HSM 支援物件 ID，則必須使用相同的物件標籤和物件 ID。
- 憑證和私密金鑰必須可透過物件標籤來解析。
- 私密金鑰必須具有下列權限：
  - `sign`
  - `decrypt`
- (選擇性) 若要使用密碼管理員元件，您必須使用 2.1.0 或更新版本，且私密金鑰必須具有下列權限：
  - `unwrap`
  - `wrap`

硬體安全性最佳做法

在 Greengrass 核心裝置上設定硬體安全性時，請考慮下列最佳作法。

- 使用內部硬體隨機號碼產生器，直接在 HSM 產生私有金鑰。這種方法比匯入您在其他地方產生的私密金鑰更安全，因為私密金鑰會保留在 HSM 中。
- 將私鑰配置為不可變並禁止導出。
- 使用 HSM 硬體廠商建議的佈建工具，使用受硬體保護的私密金鑰產生憑證簽署要求 (CSR)，然後使用AWS IoT主控台或 API 產生用戶端憑證。

> **Note**
>
> 在 HSM 上產生私密金鑰時，不適用輪換金鑰的安全性最佳做法。

安裝具有硬件安全性的AWS IoT Greengrass核心軟件

安裝 AWS IoT Greengrass Core 軟體時，您可以將其設定為使用在 HSM 中產生的私密金鑰。此方法遵循安全性最佳實務，在 HSM 中產生私密金鑰，因此私密金鑰會保留在 HSM 中。

若要安裝具有硬體安全性的 AWS IoT Greengrass Core 軟體，請執行下列動作：

1. 在 HSM 中產生私密金鑰。
2. 從私密金鑰建立憑證簽署要求 (CSR)。
3. 從 CSR 建立憑證。您可以建立由AWS IoT其他根憑證授權單位 (CA) 簽署的憑證。如需有關如何使用其他根 CA 的詳細資訊，請參閱AWS IoT Core開發人員指南中的建立您自己的用戶端憑證。
4. 下載AWS IoT憑證並將其匯入 HSM。
5. 從指定要在 HSM 中使用 PKCS #11 提供者元件以及私密金鑰和憑證的組態檔安裝 AWS IoT Greengrass Core 軟體。

您可以選擇下列其中一個安裝選項來安裝具有硬體安全性的 AWS IoT Greengrass Core 軟體：

- 手動安裝

  選擇此選項可手動建立必要的AWS資源並設定硬體安全性。如需詳細資訊，請參閱 透過手動佈建資源安裝 AWS IoT Greengrass 核心軟體。
- 以自訂佈建進行安裝

  選擇此選項可開發自訂 Java 應用程式，該應用程式會自動建立必要的AWS資源並設定硬體安全性。如需詳細資訊，請參閱 使用自訂資源佈建安裝 AWS IoT Greengrass 核心軟體。

目前，當您使用自動資源佈建或AWS IoT叢集佈建進行安裝時，AWS IoT Greengrass不支援使用硬體安全性安裝 AWS IoT Greengrass Core 軟體。

在現有核心裝置上設定硬體安全性

您可以將核心裝置的私密金鑰和憑證匯入 HSM，以設定硬體安全性。

> **考量事項**
>
> - 您必須擁有核心裝置檔案系統的 root 存取權。
> - 在此程序中，您會關閉 AWS IoT Greengrass Core 軟體，以便在您設定硬體安全性時，核心裝置處於離線狀態且無法使用。

若要在現有核心裝置上設定硬體安全性，請執行下列動作：

1. 初始化 HSM。
2. 將 PKCS #11 提供者元件部署至核心裝置。
3. 停止AWS IoT Greengrass核心軟體。
4. 將核心裝置的私密金鑰和憑證匯入 HSM。
5. 更新 AWS IoT Greengrass Core 軟體的組態檔案，以使用 HSM 中的私密金鑰和憑證。
6. 啟動AWS IoT Greengrass核心軟體。

步驟 1：初始化硬體安全模組

完成下列步驟，在核心裝置上初始化 HSM。

初始化硬體安全模組

- 在 HSM 中初始化 PKCS #11 權杖，並儲存該權杖的插槽識別碼和使用者 PIN 碼。請查看 HSM 的文件以了解如何初始化權杖。稍後部署和設定 PKCS #11 提供者元件時，您可以使用插槽識別碼和使用者 PIN 碼。

步驟 2：部署 PKCS #11 提供者元件

完成下列步驟，以部署並設定 PKCS #11 提供者元件。您可以將元件部署到一或多個核心裝置。

部署 PKCS #11 提供者元件 (主控台)

1. 在AWS IoT Greengrass主控台瀏覽功能表中，選擇 [元件]。
2. 在 [元件] 頁面上，選擇 [公用元件] 索引標籤，然後選擇aws.greengrass.crypto.Pkcs11Provider。
3. 在 aws.greengrass.crypto.Pkcs11Provider頁面中，選擇部署。
4. 從 [新增至部署] 中，選擇要修訂的現有部署，或選擇建立新部署，然後選擇 [下一步]。
5. 如果您選擇建立新部署，請為部署選擇目標核心裝置或物件群組。在 [指定目標] 頁面的 [部署目標] 下，選擇核心裝置或物件群組，然後選擇 [下一步]。
6. 在 [選取元件] 頁面的 [公用元件] 底下，選取 aws.greengrass.crypto.Pkcs11Provider，然後選擇 [下一步]。
7. 在 [設定元件] 頁面上，選取 aws.greengrass.crypto.Pkcs11Provider，然後執行下列動作：

   a. 選擇 設定元件。

b. 在設定aws.greengrass.crypto.Pkcs11Provider強制回應的組態更新下，在要合併的組態中，輸入下列組態更新。使用目標核心裝置的值更新下列組態參數。指定您先前初始化 PKCS #11 權杖的插槽識別碼和使用者 PIN 碼。稍後您將私密金鑰和憑證匯入 HSM 中的這個插槽。

`name`

PKCS #11 組態的名稱。

`library`

PKCS #11 實作程式庫的絕對檔案路徑，AWS IoT Greengrass核心軟體可以使用 libdl 載入。

`slot`

包含私密金鑰和裝置憑證的插槽識別碼。此值與槽索引或槽標籤不同。

`userPin`

用來存取插槽的使用者 PIN 碼。

```
{
  "name": "softhsm_pkcs11",
  "library": "/usr/lib/softhsm/libsofthsm2.so",
  "slot": 1,
  "userPin": "1234"
}
```

c. 選擇 [確認] 關閉強制回應，然後選擇 [下一步]。

8. 在設定進階設定頁面上，保留預設組態設定，然後選擇 下一步。
9. 在 Review (檢閱) 頁面，選擇 Deploy (部署)。

部署最多可能需要一分鐘的時間才能完成。

若要部署 PKCS #11 提供者元件 () AWS CLI

若要部署 PKCS #11 提供者元件，請建立包含`aws.greengrass.crypto.Pkcs11Provider`在物件中的部署文`components`件，並指定元件的組態更新。遵循中建立部署的指示建立新部署或修訂既有部署。

下列範例部分部署文件指定要部署及設定 PKCS #11 提供者元件。使用目標核心裝置的值更新下列組態參數。儲存插槽 ID 和使用者 PIN 碼，以便稍後在將私密金鑰和憑證匯入 HSM 時使用。

`name`

PKCS #11 組態的名稱。

`library`

PKCS #11 實作程式庫的絕對檔案路徑，AWS IoT Greengrass核心軟體可以使用 libdl 載入。

`slot`

包含私密金鑰和裝置憑證的插槽識別碼。此值與槽索引或槽標籤不同。

`userPin`

用來存取插槽的使用者 PIN 碼。

```
{
  "name": "softhsm_pkcs11",
  "library": "/usr/lib/softhsm/libsofthsm2.so",
  "slot": 1,
  "userPin": "1234"
}
```

```
{
  ...,
  "components": {
    ...,
    "aws.greengrass.crypto.Pkcs11Provider": {
      "componentVersion": "2.0.0",
      "configurationUpdate": {
        "merge": "{\"name\":\"softhsm_pkcs11\",\"library\":\"/usr/lib/softhsm/
libsofthsm2.so\",\"slot\":1,\"userPin\":\"1234\"}"
      }
    }
  }
}
```

可能需要幾分鐘才能完成部署。您可以使用該AWS IoT Greengrass服務來檢查部署的狀態。您可以檢查AWS IoT Greengrass核心軟體記錄檔，以確認 PKCS #11 提供者元件是否已成功部署。如需詳細資訊，請參閱下列內容：

- 檢查部署狀態
- 監控AWS IoT Greengrass日誌

如果部署失敗，您可以疑難排解每個核心裝置上的部署問題。如需詳細資訊，請參閱 疑難排 AWS IoT Greengrass V2。

步驟 3：更新核心設備上的配置

AWS IoT Greengrass核心軟體使用組態檔案來指定裝置的運作方式。此組態檔案包括在何處尋找私密金鑰和裝置用來連線到的憑證AWS 雲端。完成下列步驟，將核心裝置的私密金鑰和憑證匯入 HSM，並更新組態檔以使用 HSM。

更新核心裝置上的組態以使用硬體安全性

1. 停止AWS IoT Greengrass核心軟體。如果您使用 systemd 將 AWS IoT GreengrassCore 軟體設定為系統服務，則可以執行下列命令來停止軟體。

   ```
   sudo systemctl stop greengrass.service
   ```

2. 尋找核心裝置的私密金鑰和憑證檔案。

   - 如果您使用自動佈建或叢集佈建來安裝 AWS IoT Greengrass Core 軟體，則私密金鑰會存在於*/greengrass/v2*/privKey.key，且憑證存在於*/greengrass/v2*/thingCert.crt。
   - 如果您以手動佈建方式安裝 AWS IoT Greengrass Core 軟體，則預設會存在*/greengrass/v2*/private.pem.key私密金鑰，且憑證*/greengrass/v2*/device.pem.crt預設存在。

   您也可以入庫納管system.privateKeyPath和system.certificateFilePath性質，*/greengrass/v2*/config/effectiveConfig.yaml以尋找這些檔案的位置。

3. 將私密金鑰和憑證匯入 HSM。請查看 HSM 的文件，了解如何將私密金鑰和憑證匯入其中。使用先前初始化 PKCS #11 權杖的插槽識別碼和使用者 PIN 碼匯入私密金鑰和憑證。私密金鑰和憑證必須使用相同的物件標籤和物件 ID。儲存匯入每個檔案時指定的物件標籤。稍後當您更新AWS IoT Greengrass核心軟體組態以使用 HSM 中的私密金鑰和憑證時，您可以使用此標籤。

4. 更新AWS IoT Greengrass核心組態以使用 HSM 中的私密金鑰和憑證。若要更新組態，請修改 AWS IoT Greengrass Core 組態檔案，並使用更新的組態檔執行 AWS IoT Greengrass Core 軟體，以套用新的組態。

   請執行下列操作：

a. 創建AWS IoT Greengrass核心配置文件的備份。如果您在設定硬體安全性時遇到問題，可以使用此備份來還原核心裝置。

```
sudo cp /greengrass/v2/config/effectiveConfig.yaml ~/ggc-config-backup.yaml
```

b. 在文字編輯器中開啟 AWS IoT Greengrass Core 組態檔案。例如，您可以執行下列命令來使用 GNU nano 來編輯檔案。取代*/greengrass/v2*為 Greengrass 根資料夾的路徑。

```
sudo nano /greengrass/v2/config/effectiveConfig.yaml
```

c. 將的值取代為 `system.privateKeyPath` HSM 中私密金鑰的 PKCS #11 URI。將 *iotdevicekey* 取代為您先前匯入私密金鑰和憑證的物件標籤。

```
pkcs11:object=iotdevicekey;type=private
```

d. 將的值取代為 HSM 中憑證的 PKCS #11 URI。`system.certificateFilePath`將 *iotdevicekey* 取代為您先前匯入私密金鑰和憑證的物件標籤。

```
pkcs11:object=iotdevicekey;type=cert
```

完成這些步驟之後，AWS IoT GreengrassCore 配置檔案中的`system`屬性看起來應該類似於下列範例。

```
system:
  certificateFilePath: "pkcs11:object=iotdevicekey;type=cert"
  privateKeyPath: "pkcs11:object=iotdevicekey;type=private"
  rootCaPath: "/greengrass/v2/rootCA.pem"
  rootpath: "/greengrass/v2"
  thingName: "MyGreengrassCore"
```

5. 在更新的檔`effectiveConfig.yaml`案中套用組態。`Greengrass.jar`使用要在中套用組態的`--init-config`參數執行`effectiveConfig.yaml`。取代*/greengrass/v2*為 Greengrass 根資料夾的路徑。

```
sudo java -Droot="/greengrass/v2" \
  -jar /greengrass/v2/alts/current/distro/lib/Greengrass.jar \
  --start false \
  --init-config /greengrass/v2/config/effectiveConfig.yaml
```

6. 啟動AWS IoT Greengrass核心軟體。如果您使用 systemd 將 AWS IoT GreengrassCore 軟體設定為系統服務，則可以執行下列命令來啟動軟體。

   ```
   sudo systemctl start greengrass.service
   ```

   如需詳細資訊，請參閱 執行AWS IoT Greengrass核心軟體。

7. 檢查AWS IoT Greengrass核心軟體記錄檔，確認軟體是否啟動並連線至AWS 雲端. AWS IoT Greengrass核心軟體會使用私密金鑰和憑證來連線到AWS IoT和AWS IoT Greengrass服務。

   ```
   sudo tail -f /greengrass/v2/logs/greengrass.log
   ```

   下列資訊層級記錄訊息指出 AWS IoT Greengrass Core 軟體已成功連線到AWS IoT和AWS IoT Greengrass服務。

   ```
   2021-12-06T22:47:53.702Z [INFO] (Thread-3)
    com.aws.greengrass.mqttclient.AwsIotMqttClient: Successfully connected to AWS IoT
    Core. {clientId=MyGreengrassCore5, sessionPresent=false}
   ```

8. (選擇性) 確認 AWS IoT Greengrass Core 軟體可與 HSM 中的私密金鑰和憑證搭配使用後，請從裝置的檔案系統中刪除私密金鑰和憑證檔案。執行下列命令，並以私密金鑰和憑證檔案的路徑取代檔案路徑。

   ```
   sudo rm /greengrass/v2/privKey.key
   sudo rm /greengrass/v2/thingCert.crt
   ```

使用不支援 PKCS #11 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。但是，如果您的硬體沒有對應的 PKCS #11 程式庫實作，或者如果您想要撰寫自訂 PKCS #11 提供者，請聯絡您的 Amazon Web Services 企業 Support 代表，詢問與整合相關的問題。

另請參閱

- PKCS #11 密碼編譯權杖介面使用指南 2.4.0 版
- RFC 7512
- PKCS #1: RSA Encryption Version 1.5

AWS IoT Greengrass 的裝置身分驗證和授權

AWS IoT Greengrass 環境中的裝置會使用 X.509 憑證進行身分驗證以及 AWS IoT 政策進行授權。憑證和原則允許裝置彼此、與 AWS IoT Core 和 AWS IoT Greengrass 安全地連線。

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 政策會定義允許 AWS IoT 裝置的操作集。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 訊息和擷取裝置陰影。

所有裝置在 AWS IoT Core 登錄中都需要有個項目，以及一個已連接 AWS IoT 政策且已啟動的 X.509 憑證。裝置可分為兩類：

- Greengrass 核心器件

  Greengrass 核心裝置會使用憑證和AWS IoT原則來連線到和。AWS IoT Core AWS IoT Greengrass 憑證和原則也允許AWS IoT Greengrass將元件和組態部署到核心裝置。
- 用戶端裝置

  MQTT 用戶端裝置使用憑證和原則來連線AWS IoT Core和服務。AWS IoT Greengrass這可讓用戶端裝置使用AWS IoT Greengrass雲端探索來尋找 Greengrass 核心裝置並連線。用戶端裝置使用相同的憑證來連線至AWS IoT Core雲端服務和核心裝置。用戶端裝置也會使用探索資訊與核心裝置進行相互驗證。如需詳細資訊，請參閱 與本機 IoT 裝置互動。

X.509 憑證

核心裝置與用戶端裝置之間以及裝置與AWS IoT Core或之間的通訊AWS IoT Greengrass必須經過驗證。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，裝置會針對下列 Transport Layer Security (TLS) 連線使用具有公有和私有金鑰的憑證：

- Greengrass 核心裝置上連線AWS IoT Core並AWS IoT Greengrass透過網際網路連線的AWS IoT用戶端元件。
- AWS IoT Greengrass透過網際網路連線以探索核心裝置的用戶端裝置。
- Greengrass 核心上的 MQTT 代理程式元件，透過區域網路連接至群組中的 Greengrass 裝置。

AWS IoT Greengrass核心裝置會將憑證儲存在 Greengrass 根資料夾中。

憑證授權機構 (CA) 憑證

Greengrass 核心裝置和用戶端裝置會下載用於與服務驗證的根 CA 憑證。AWS IoT Core AWS IoT Greengrass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開發人員指南中的用於伺服器驗證的 CA 憑證。

用戶端裝置也會下載 Greengrass 核心裝置 CA 憑證。他們使用此憑證在相互驗證期間驗證核心裝置上的 MQTT 伺服器憑證。

本機 MQTT 代理程式上的憑證輪替

當您啟用用戶端裝置支援時，Greengrass 核心裝置會產生用於相互驗證的本機 MQTT 伺服器憑證。此憑證由核心裝置 CA 憑證簽署，核心裝置會儲存在AWS IoT Greengrass雲端。用戶端裝置在探索核心裝置時擷取核心裝置 CA 憑證。當核心裝置連線至核心裝置時，他們會使用核心裝置 CA 憑證來驗證核心裝置的 MQTT 伺服器憑證。核心裝置 CA 憑證會在 5 年後到期。

根據預設，MQTT 伺服器憑證每 7 天到期一次，您可以將此持續時間設定為 2 到 10 天。此有限期間是依據安全最佳實務。此輪換有助於降低攻擊者竊取 MQTT 伺服器憑證和私密金鑰來模擬 Greengrass 核心裝置的威脅。

Greengrass 核心裝置會在到期前 24 小時旋轉 MQTT 伺服器憑證。Greengrass 核心裝置會產生新的憑證，並重新啟動本機 MQTT 代理程式。發生這種情況時，連接到 Greengrass 核心裝置的所有用戶端裝置都會中斷連線。用戶端裝置可以在短時間後重新連線到 Greengrass 核心裝置。

資料平面操作的AWS IoT 政策

使用AWS IoT政策授權存取AWS IoT Core和AWS IoT Greengrass資料平面。資AWS IoT Core料平面可為裝置、使用者和應用程式提供作業。這些操作包括連接AWS IoT Core和訂閱主題的功能。資AWS IoT Greengrass料平面提供 Greengrass 裝置的作業。如需詳細資訊，請參閱 AWS IoT Greengrass V2 政策動作。這些作業包括解決元件相依性和下載公用元件加工品的能力。

AWS IoT政策是類似於 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`。 策略允許或拒絕的處理行動清單。
- `Resource`。 允許或拒絕動作的資源清單。

AWS IoT原則支援*為萬用字元，並將 MQTT 萬用字元 (+和#) 視為常值字串。有關*萬用字元的詳細資訊，請參閱《使用指南》中的〈在資源 ARN 中使用萬用字元〉。AWS Identity and Access Management

如需詳細資訊，請參閱AWS IoT Core開發人員指南中的AWS IoT政AWS IoT策和政策動作。

> **Important**
>
> 核心裝置或 Greengrass 資料平面作業的原AWS IoT則中不支援物件原則變數 (`iot:Connection.Thing.*`)。相反地，您可以使用萬用字元來比對具有相似名稱的多個裝置。例如，您可以指定`MyGreengrassDevice*`要相符`MyGreengrassDevice1`、`MyGreengrassDevice2`、等等。

> **Note**
>
> AWS IoT Core可讓您將AWS IoT原則附加至物件群組，以定義裝置群組的權限。物件群組原則不允許存取 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，請將許可新增至您附加至實物憑證的 AWS IoT 政策。

AWS IoT Greengrass V2 政策動作

AWS IoT Greengrass V2定義 Greengrass 核心裝置和用戶端裝置可在策略中使用的下列策略處理行動。AWS IoT若要指定政策動作的資源，請使用資源的 Amazon 資源名稱 (ARN)。

核心裝置動作

`greengrass:GetComponentVersionArtifact`

授予取得預先簽署 URL 的權限，以下載公用元件成品或 Lambda 元件成品。

當核心裝置收到指定公用元件或含有成品的 Lambda 的部署時，就會評估此權限。如果核心設備已經具有成品，則不會再次下載成品。

資源類型：`componentVersion`

資源檔案格式:`arn:aws:greengrass:`*`region`*`:`*`account-id`*`:components:`*`component-name`*`:versions:`*`component-version`*

`greengrass:ResolveComponentCandidates`

授與識別符合部署元件、版本和平台需求之元件清單的權限。如果需求衝突，或沒有符合需求的元件，則此作業會傳回錯誤，且設備上的部署失敗。

當核心裝置收到指定元件的部署時，會評估此權限。

資源類型:無

資源檔案格式:*

`greengrass:GetDeploymentConfiguration`

授予取得預先簽署 URL 以下載大型部署文件的權限。

當核心裝置收到指定部署文件大於 7 KB (如果部署以物件為目標) 或 31 KB (如果部署以物件為目標) 的部署時，就會評估此權限。部署文件包括元件組態、部署原則和部署中繼資料。如需詳細資訊，請參閱 將AWS IoT Greengrass元件部署到裝置。

此功能適用於 v2.3.0 及更高版 Greeng rass 核組件。

資源類型:無

資源檔案格式:*

`greengrass:ListThingGroupsForCoreDevice`

授予取得核心裝置物件群組階層的權限。

核心裝置從中接收部署時，會檢查此權限AWS IoT Greengrass。核心裝置會使用此動作來識別自上次部署之後是否已將其從物件群組中移除。如果核心裝置已從物件群組中移除，而該物件群組是核心裝置部署的目標，則核心裝置會移除該部署所安裝的元件。

此功能由 v2.5.0 和更高版本的 Greeng rass 件使用。

資源類型:`thing`(核心裝置)

資源檔案格式:`arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`core-device-thing-name`*

`greengrass:VerifyClientDeviceIdentity`

授予驗證連線至核心裝置之用戶端裝置身分識別的權限。

當核心裝置執行用戶端裝置驗證元件並從用戶端裝置接收 MQTT 連線時，會評估此權限。用戶端裝置會顯示其AWS IoT裝置憑證。然後，核心裝置會將裝置憑證傳送至AWS IoT Greengrass雲端服務，以驗證用戶端裝置的身分識別。如需詳細資訊，請參閱 與本機 IoT 裝置互動。

資源類型:無

資源檔案格式:*

`greengrass:VerifyClientDeviceIoTCertificateAssociation`

授與驗證用戶端裝置是否與AWS IoT憑證相關聯的權限。

當核心裝置執行用戶端裝置驗證元件並授權用戶端裝置透過 MQTT 連線時，會評估此權限。如需詳細資訊，請參閱 與本機 IoT 裝置互動。

> **Note**
>
> 若要讓核心裝置使用此作業，Greengrass 服務角色必須與您的關聯AWS 帳戶並允許權限。`iot:DescribeCertificate`

資源類型:`thing`(用戶端裝置)

資源檔案格式:`arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`client-device-thing-name`*

`greengrass:PutCertificateAuthorities`

授與上傳憑證授權單位 (CA) 憑證的權限，供用戶端裝置下載以驗證核心裝置。

當核心設備安裝並運行客戶端設備身份驗證組件時，將評估此權限。此元件會建立本機憑證授權單位，並使用此作業上傳其 CA 憑證。用戶端裝置會在使用「探索」作業尋找可連線的核心裝置時，下載這些 CA 憑證。當用戶端裝置連線至核心裝置上的 MQTT 代理程式時，會使用這些 CA 憑證來驗證核心裝置的身分識別。如需詳細資訊，請參閱 與本機 IoT 裝置互動。

資源類型:無

ARN 格式:*

`greengrass:GetConnectivityInfo`

授予取得核心裝置連線資訊的權限。此資訊說明用戶端裝置如何連線至核心裝置。

當核心設備安裝並運行客戶端設備身份驗證組件時，將評估此權限。此元件會使用連線資訊來產生有效的 CA 憑證，以便隨PutCertificateAuthories作業一起上傳至AWS IoT Greengrass雲端服務。用戶端裝置會使用這些 CA 憑證來驗證核心裝置的身分。如需詳細資訊，請參閱 與本機 IoT 裝置互動。

您也可以在AWS IoT Greengrass控制平面上使用此操作來檢視核心裝置的連線資訊。如需詳細資訊，請參閱 AWS IoT Greengrass V1 API 參考中的 GetConnectivityInfo。

資源類型:thing(核心裝置)

資源檔案格式:arn:aws:iot:*region*:*account-id*:thing/*core-device-thing-name*

greengrass:UpdateConnectivityInfo

授予更新核心裝置連線資訊的權限。此資訊說明用戶端裝置如何連線至核心裝置。

核心裝置執行 IP 偵測器元件時，會評估此權限。此元件可識別用戶端裝置連線至區域網路上的核心裝置所需的資訊。然後，此元件會使用此作業將連線資訊上傳至AWS IoT Greengrass雲端服務，以便用戶端裝置可以透過探索作業擷取此資訊。如需詳細資訊，請參閱 與本機 IoT 裝置互動。

您也可以在AWS IoT Greengrass控制平面上使用此作業，手動更新核心裝置的連線資訊。如需詳細資訊，請參閱 AWS IoT Greengrass V1 API 參考中的 UpdateConnectivityInfo。

資源類型:thing(核心裝置)

資源檔案格式:arn:aws:iot:*region*:*account-id*:thing/*core-device-thing-name*

用戶端裝置動作

greengrass:Discover

授予探索用戶端裝置可連線之核心裝置連線資訊的權限。此資訊說明用戶端裝置如何連線至核心裝置。用戶端裝置只能使用此BatchAssociateClientDeviceWithCoreDevice作業探索與其關聯的核心裝置。如需詳細資訊，請參閱 與本機 IoT 裝置互動。

資源類型:thing(用戶端裝置)

資源檔案格式:arn:aws:iot:*region*:*account-id*:thing/*client-device-thing-name*

更新核心裝置的AWS IoT政策

您可以使用AWS IoT Greengrass和AWS IoT主控台或 AWS IoT API 來檢視和更新核心裝置的AWS IoT政策。

> **Note**
>
> 如果您使用 AWS IoT GreengrassCore 軟體安裝程式佈建資源，您的核心裝置會有允許存取所有AWS IoT Greengrass動作的AWS IoT原則 (greengrass:*)。您可以依照下列步驟，限制只存取核心裝置所使用的動作。

檢閱並更新核心裝置的政AWS IoT策 (主控台)

1. 在AWS IoT Greengrass主控台瀏覽功能表中，選擇 [核心裝置]。
2. 在 [核心裝置] 頁面上，選擇要更新的核心裝置。
3. 在核心裝置詳細資料頁面上，選擇核心裝置物件的連結。此連結會在AWS IoT主控台中開啟物件詳細資訊頁面。
4. 在物件詳細資訊頁面上，選擇憑證。
5. 在「憑證」索引標籤中，選擇物件的使用中憑證。
6. 在憑證詳細資料頁面上，選擇 [原則]。
7. 在「策略」索引標籤中，選擇要檢閱和更新的AWS IoT策略。您可以將必要的權限新增至任何附加至核心裝置作用中憑證的原則。

   > **Note**
   >
   > 如果您使用 AWS IoT GreengrassCore 軟體安裝程式佈建資源，則有兩個AWS IoT原則。建議您選擇名為GreengrassV2IoTThingPolicy的策略 (如果存在的話)。依預設，您使用快速安裝程式建立的核心裝置會使用此原則名稱。如果您將權限新增至此原則，也會將這些權限授與使用此原則的其他核心裝置。

8. 在策略概觀中，選擇編輯作用中的版本。
9. 檢閱原則，並視需要新增、移除或編輯權限。
10. 若要將新的原則版本設定為作用中版本，請在 [原則版本狀態] 下選取 [將編輯的版本設定為此原則的作用中版本]。
11. 選擇「另存為新版本」。

檢閱並更新核心裝置的AWS IoT政策 (AWS CLI)

1. 列出核心裝置物AWS IoT件的主體。物件主體可以是 X.509 裝置憑證或其他識別。執行下列命令，並*MyGreengrassCore*以核心裝置的名稱取代。

   ```
   aws iot list-thing-principals --thing-name MyGreengrassCore
   ```

   作業會傳回列出核心裝置物件主參與者的回應。

   ```
   {
       "principals": [
   ```

```
        "arn:aws:iot:us-west-2:123456789012:cert/certificateId"
    ]
}
```

2. 識別核心裝置的作用中憑證。執行下列命令，並將 certific *ateId* 取代為上一個步驟中每個憑證的 ID，直到找到作用中的憑證為止。憑證 ID 是位於憑證 ARN 結尾的十六進位字串。引--query數指定僅輸出憑證的狀態。

```
aws iot describe-certificate --certificate-id certificateId --query
 'certificateDescription.status'
```

作業會以字串形式傳回憑證狀態。例如，如果憑證處於作用中狀態，則此作業會輸出"ACTIVE"。

3. 列出附加至憑證的AWS IoT原則。執行下列命令，並以憑證的 ARN 取代憑證 ARN。

```
aws iot list-principal-policies --principal arn:aws:iot:us-
west-2:123456789012:cert/certificateId
```

作業會傳回回應，列出附加至憑證的AWS IoT原則。

```
{
    "policies": [
        {
            "policyName":
 "GreengrassTESCertificatePolicyMyGreengrassCoreTokenExchangeRoleAlias",
            "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassTESCertificatePolicyMyGreengrassCoreTokenExchangeRoleAlias"
        },
        {
            "policyName": "GreengrassV2IoTThingPolicy",
            "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy"
        }
    ]
}
```

4. 選擇要檢視和更新的原則。

> **Note**
>
> 如果您使用 AWS IoT GreengrassCore 軟體安裝程式佈建資源，則有兩個AWS IoT原則。建議您選擇名為GreengrassV2IoTThingPolicy的策略 (如果存在的話)。依預設，您使用快

速安裝程式建立的核心裝置會使用此原則名稱。如果您將權限新增至此原則，也會將這些權限授與使用此原則的其他核心裝置。

5. 取得政策的文件。執行下列命令，並以原則的名稱取代 *GreenGrassv2IoT ThingPolicy*。

```
aws iot get-policy --policy-name GreengrassV2IoTThingPolicy
```

作業會傳回回應，其中包含原則的文件及其他有關原則的資訊。政策文件是序列化為字串的 JSON 物件。

```
{
    "policyName": "GreengrassV2IoTThingPolicy",
    "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy",
    "policyDocument": "{\
  \\"Version\\": \\"2012-10-17\\",\
  \\"Statement\\": [\
    {\
      \\"Effect\\": \\"Allow\\",\
      \\"Action\\": [\
                \\"iot:Connect\\",\
                \\"iot:Publish\\",\
                \\"iot:Subscribe\\",\
                \\"iot:Receive\\",\
                \\"greengrass:*\\"\
],\
      \\"Resource\\": \\"*\\"\
    }\
  ]\
}",
    "defaultVersionId": "1",
    "creationDate": "2021-02-05T16:03:14.098000-08:00",
    "lastModifiedDate": "2021-02-05T16:03:14.098000-08:00",
    "generationId":
 "f19144b798534f52c619d44f771a354f1b957dfa2b850625d9f1d0fde530e75f"
}
```

6. 使用線上轉換器或其他工具將原則文件字串轉換為 JSON 物件，然後將其儲存到名為的檔案中`iot-policy.json`。

例如，如果您已安裝 jq 工具，您可以執行下列命令來取得原則文件、將其轉換為 JSON 物件，並將原則文件儲存為 JSON 物件。

```
aws iot get-policy --policy-name GreengrassV2IoTThingPolicy --query
 'policyDocument' | jq fromjson >> iot-policy.json
```

7. 檢閱原則文件，並視需要新增、移除或編輯權限。

   例如，在 Linux 系統上，您可以執行下列命令來使用 GNU nano 來開啟檔案。

   ```
   nano iot-policy.json
   ```

   完成後，原則文件看起來可能類似於核心裝置的最小AWS IoT原則。

8. 將變更儲存為策略的新版本。執行下列命令，並以原則的名稱取代 *GreenGrassv2IoTThingPolicy*。

   ```
   aws iot create-policy-version --policy-name GreengrassV2IoTThingPolicy --policy-
   document file://iot-policy.json --set-as-default
   ```

   如果成功，作業會傳回類似下列範例的回應。

   ```
   {
       "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
   GreengrassV2IoTThingPolicy",
       "policyDocument": "{\
     \\"Version\\": \\"2012-10-17\\",\
     \\"Statement\\": [\
       {\
         \\"Effect\\": \\"Allow\\",\
         \\"Action\\": [\
   \\t\\t\\"iot:Connect\\",\
   \\t\\t\\"iot:Publish\\",\
   \\t\\t\\"iot:Subscribe\\",\
   \\t\\t\\"iot:Receive\\",\
   \\t\\t\\"greengrass:*\\"\
         ],\
         \\"Resource\\": \\"*\\"\
       }\
     ]\
   }",
       "policyVersionId": "2",
       "isDefaultVersion": true
   }
   ```

AWS IoT Greengrass V2核心裝置的最低AWS IoT原則

> **Important**
>
> 較新版本的 Greengrass 核心元件需要最小原則的額外權限。AWS IoT您可能需要更新核心裝置的政AWS IoT策以授予其他權限。
>
> - 執行 Greengrass 核心核心 v2.5.0 及更新版本的核心裝置會在您從物件群組移除核心裝置時，使用`greengrass:ListThingGroupsForCoreDevice`權限解除安裝元件。
> - 執行 Greengrass 核心 v2.3.0 及更新版本的核心裝置會使用此`greengrass:GetDeploymentConfiguration`權限來支援大型部署設定文件。

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。

- 此`Connect`原則會在核心裝置物件名稱之後包含`*`萬用字元 (例如，*core-device-thing-name*`*`)。核心裝置使用相同的裝置憑證來建立多個並行訂閱AWS IoT Core，但連線中的用戶端 ID 可能與核心裝置物件名稱不完全相符。在前 50 個訂閱之後，核心裝置會使用*core-device-thing-name*`#`*number*做為用戶端 ID，其中每*number*增加 50 個訂閱就會增加一次。例如，當名為的核心裝置`MyCoreDevice`建立 150 個並行訂閱時，它會使用下列用戶端 ID：
  - 訂閱項目 1 至 50：`MyCoreDevice`
  - 訂閱項目 51 到 100：`MyCoreDevice#2`
  - 訂閱項目 101 到 150：`MyCoreDevice#3`

  萬用字元可讓核心裝置在使用這些具有尾碼的用戶端 ID 時進行連線。
- 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 元件和用戶端裝置之間AWS IoT Core的訊息交換，請指定您要允許的主題和主題篩選器。如需詳細資訊，請參閱開發人員指南中的AWS IoT Core發佈/訂閱政策範例。
- 此原則會授與發佈至下列遙測資料主題的權限。

  ```
  $aws/things/core-device-thing-name/greengrass/health/json
  ```

  您可以針對停用遙測的核心裝置移除此權限。如需詳細資訊，請參閱 從AWS IoT Greengrass核心裝置收集系統健康狀態遙測資料。

- 該政策授予透過角色別名假設 IAM 角AWS IoT色的權限。核心裝置會使用這個角色 (稱為 Token Exchange 角色) 來取得可用來驗證AWS要求的AWS認證。如需詳細資訊，請參閱 [授權核心設備與AWS服務](#)。

  當您安裝 AWS IoT Greengrass Core 軟體時，您會建立並附加僅包含此權限的第二個AWS IoT原則。如果您在核心裝置的主要AWS IoT原則中包含此權限，則可以卸離並刪除其他AWS IoT原則。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "arn:aws:iot:region:account-id:client/core-device-thing-name*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive",
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:topic/$aws/things/core-device-thing-name/greengrass/health/json",
                "arn:aws:iot:region:account-id:topic/$aws/things/core-device-thing-name/greengrassv2/health/json",
                "arn:aws:iot:region:account-id:topic/$aws/things/core-device-thing-name/jobs/*",
                "arn:aws:iot:region:account-id:topic/$aws/things/core-device-thing-name/shadow/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:topicfilter/$aws/things/core-device-thing-name/jobs/*",
```

```
                "arn:aws:iot:region:account-id:topicfilter/$aws/things/core-device-
thing-name/shadow/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iot:AssumeRoleWithCertificate",
            "Resource": "arn:aws:iot:region:account-id:rolealias/token-exchange-role-
alias-name"
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:GetComponentVersionArtifact",
                "greengrass:ResolveComponentCandidates",
                "greengrass:GetDeploymentConfiguration",
                "greengrass:ListThingGroupsForCoreDevice"
            ],
            "Resource": "*"
        }
    ]
}
```

支援用戶端裝置的最低AWS IoT原則

下列範例原則包含支援核心裝置上用戶端裝置互動所需的最低處理行動集。若要支援用戶端裝置，核心裝置除了基本作業的最小AWS IoT原則之外，還必須具有此AWS IoT原則中的權限。

- 此原則可讓核心裝置更新自己的連線資訊。僅當您將 IP 偵測器元件部署到核心裝置時，才需要此權限 (greengrass:UpdateConnectivityInfo)。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:topic/$aws/things/core-device-thing-
name-gci/shadow/get"
```

```
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:topicfilter/$aws/things/core-device-
thing-name-gci/shadow/update/delta",
                "arn:aws:iot:region:account-id:topicfilter/$aws/things/core-device-
thing-name-gci/shadow/get/accepted"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:topic/$aws/things/core-device-thing-
name-gci/shadow/update/delta",
                "arn:aws:iot:region:account-id:topic/$aws/things/core-device-thing-
name-gci/shadow/get/accepted"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:PutCertificateAuthorities",
                "greengrass:VerifyClientDeviceIdentity"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:VerifyClientDeviceIoTCertificateAssociation"
            ],
            "Resource": "arn:aws:iot:region:account-id:thing/*"
        },
        {
            "Effect": "Allow",
            "Action": [
```

```
                "greengrass:GetConnectivityInfo",
                "greengrass:UpdateConnectivityInfo"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:thing/core-device-thing-name"
            ]
        }
    ]
}
```

用戶端裝置的最低AWS IoT原則

下列範例原則包含用戶端裝置探索核心裝置所需的最低處理行動集，以便透過 MQTT 進行連線和通訊。用戶端裝置的AWS IoT策略必須包含該`greengrass:Discover`處理行動，以允許裝置探索其關聯 Greengrass 核心裝置的連線資訊。在此`Resource`區段中，指定用戶端裝置的 Amazon 資源名稱 (ARN)，而不是 Greengrass 核心裝置的 ARN。

- 此原則允許所有 MQTT 主題進行通訊。若要遵循最佳安全性做法，請將`iot:Publish`、`iot:Subscribe`、和`iot:Receive`權限限制為用戶端裝置對您的使用案例所需的最小主題集合。
- 該策略允許事物發現所有AWS IoT事情的核心設備。若要遵循最佳安全性做法，請限制對用戶端裝置物AWS IoT件的`greengrass:Discover`權限，或限制符合一組項目的萬用字元AWS IoT的權限。

  > **Important**
  >
  > 核心裝置或 Greengrass 資料平面作業的原AWS IoT則中不支援物件原則變數 (`iot:Connection.Thing.*`)。相反地，您可以使用萬用字元來比對具有相似名稱的多個裝置。例如，您可以指定`MyGreengrassDevice*`要相符`MyGreengrassDevice1`、`MyGreengrassDevice2`、等等。

- 用戶端裝置的AWS IoT原則通常不需要、或`iot:DeleteThingShadow`動作的權限`iot:GetThingShadow`、`iot:UpdateThingShadow`，因為 Greengrass 核心裝置會處理用戶端裝置的陰影同步作業。若要讓核心裝置處理用戶端裝置陰影，請檢查核心裝置的AWS IoT政策是否允許執行這些處理行動，以及`Resource`區段是否包含用戶端裝置的 ARN。

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

```
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:topic/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:topicfilter/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:topic/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:Discover"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:thing/*"
            ]
        }
    ]
```

```
}
```

適用於 AWS IoT Greengrass 的 Identity and Access Management

AWS Identity and Access Management (IAM) 是一種 AWS 服務，讓管理員能夠安全控制對 AWS 資源的存取權限。IAM 管理員可以控制身分身分驗證 (已登入) 和授權 (具有許可) 以使用 AWS IoT Greengrass 資源。IAM 是一種您可以免費使用的 AWS 服務。

> **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 搭配 IAM 的運作方式”。

物件

AWS Identity and Access Management (IAM) 的使用方式會不同，需視您在 AWS IoT Greengrass 中所執行的工作而定。

服務使用者：如果使用 AWS IoT Greengrass 執行任務，管理員會為您提供所需的憑證和許可。隨著您為了執行作業而使用的 AWS IoT Greengrass 功能數量變多，您可能會需要額外的許可。了解存取的管理方式可協助您向管理員請求正確的許可。若您無法存取 AWS IoT Greengrass 中的某項功能，請參閱 針對 AWS IoT Greengrass 的識別和存取問題進行故障診斷。

服務管理員：如果您負責公司內的 AWS IoT Greengrass 資源，您可能具備 AWS IoT Greengrass 的完整存取權限。您的任務是判斷服務使用者應存取的 AWS IoT Greengrass 功能及資源。接著，您必須將請求提交給您的 IAM 管理員，來變更您服務使用者的許可。檢閱此頁面上的資訊，了解 IAM 的基本概念。若要進一步了解貴公司可搭配 AWS IoT Greengrass 使用 IAM 的方式，請參閱 AWS IoT Greengrass 搭配 IAM 的運作方式。

IAM 管理員：如果您是 IAM 管理員，建議您掌握如何撰寫政策以管理 AWS IoT Greengrass 存取權的詳細資訊。若要檢視您可以在 IAM 中使用的範例 AWS IoT Greengrass 身分型政策，請參閱 AWS IoT Greengrass 的身分型政策範例。

使用身分驗證

身分驗證是使用身分憑證登入 AWS 的方式。您必須以 AWS 帳戶根使用者、IAM 使用者身分，或擔任 IAM 角色進行 驗證 (登入至 AWS)。

您可以使用透過身分來源 AWS IAM Identity Center 提供的憑證，以聯合身分登入 AWS。(IAM Identity Center) 使用者、貴公司的單一登入身分驗證和您的 Google 或 Facebook 憑證都是聯合身分的範例。您以聯合身分登入時，您的管理員先前已設定使用 IAM 角色的聯合身分。您 AWS 藉由使用聯合進行存取時，您會間接擔任角色。

根據您的使用者類型，您可以登入 AWS Management Console 或 AWS 存取入口網站。如需有關登入至 AWS 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。

如果您是以程式設計的方式存取 AWS，AWS 提供軟體開發套件 (SDK) 和命令列介面 (CLI)，以便使用您的憑證透過密碼編譯方式簽署您的請求。如果您不使用 AWS 工具，您必須自行簽署請求。如需使用建議的方法自行簽署請求的詳細資訊，請參閱《IAM 使用者指南》中的簽署 AWS API 請求。

無論您使用何種身分驗證方法，您可能都需要提供額外的安全性資訊。例如，AWS 建議您使用多重要素驗證 (MFA) 以提高帳戶的安全。如需更多資訊，請參閱《AWS IAM Identity Center 使用者指南》中的多重要素驗證和《IAM 使用者指南》中的在 AWS 中使用多重要素驗證 (MFA)。

AWS 帳戶 根使用者

如果是建立 AWS 帳戶，您會先有一個登入身分，可以完整存取帳戶中所有 AWS 服務 與資源。此身分稱為 AWS 帳戶 根使用者，使用建立帳戶時所使用的電子郵件地址和密碼即可登入並存取。強烈建議您不要以根使用者處理日常作業。保護您的根使用者憑證，並將其用來執行只能由根使用者執行的任務。如需這些任務的完整清單，了解需以根使用者登入的任務，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 使用者是您 AWS 帳戶 中的一種身分，具備單一人員或應用程式的特定許可。建議您盡可能依賴暫時憑證，而不是擁有建立長期憑證 (例如密碼和存取金鑰) 的 IAM 使用者。但是如果特定使用案例需要擁有長期憑證的 IAM 使用者，建議您輪換存取金鑰。如需詳細資訊，請參閱《IAM 使用者指南》https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html#rotate-credentials中的為需要長期憑證的使用案例定期輪換存取金鑰。

IAM 群組是一種指定 IAM 使用者集合的身分。您無法以群組身分登入。您可以使用群組來一次為多名使用者指定許可。群組可讓管理大量使用者許可的過程變得更為容易。例如，您可以擁有一個名為 IAMAdmins 的群組，並給予該群組管理 IAM 資源的許可。

使用者與角色不同。使用者只會與單一人員或應用程式建立關聯，但角色的目的是在由任何需要它的人員取得。使用者擁有永久的長期憑證，但角色僅提供暫時憑證。如需進一步了解，請參閱《IAM 使用者指南》中的建立 IAM 使用者 (而非角色) 的時機。

IAM 角色

IAM 角色是您 AWS 帳戶 中的一種身分，具備特定許可。它類似 IAM 使用者，但不與特定的人員相關聯。您可以在 AWS Management Console 中透過切換角色來暫時取得 IAM 角色。您可以透過呼叫 AWS CLI 或 AWS API 操作，或是使用自訂 URL 來取得角色。如需使用角色的方法詳細資訊，請參閱《IAM 使用者指南》中的使用 IAM 角色。

使用暫時憑證的 IAM 角色在下列情況中非常有用：

- 聯合身分使用者存取 — 如需向聯合身分指派許可，請建立角色，並為角色定義許可。當聯合身分進行身分驗證時，該身分會與角色建立關聯，並取得由角色定義的許可。如需有關聯合角色的詳細資訊，請參閱《IAM 使用者指南》https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_create_for-idp.html中的為第三方身分供應商建立角色。如果您使用 IAM Identity Center，則需要設定許可集。為控制身分驗證後可以存取的內容，IAM Identity Center 將許可集與 IAM 中的角色相關聯。如需有關許可集的資訊，請參閱《AWS IAM Identity Center 使用者指南》中的許可集。
- 暫時 IAM 使用者許可 – IAM 使用者或角色可以擔任 IAM 角色來暫時針對特定任務採用不同的許可。
- 跨帳户存取權 – 您可以使用 IAM 角色，允許不同帳戶中的某人 (信任的委託人) 存取您帳戶中的資源。角色是授予跨帳户存取權的主要方式。但是，針對某些 AWS 服務，您可以將政策直接連接到資源 (而非使用角色作為代理)。如需了解使用角色和資源型政策進行跨帳戶存取之間的差異，請參閱《IAM 使用者指南》中的 IAM 角色與資源類型政策的差異。
- 跨服務存取 – 有些 AWS 服務 會使用其他 AWS 服務 中的功能。例如，當您在服務中進行呼叫時，該服務通常會在 Amazon EC2 中執行應用程式或將物件儲存在 Amazon Simple Storage Service (Amazon S3) 中。服務可能會使用呼叫主體的許可、使用服務角色或使用服務連結角色來執行此作業。
  - 轉發存取工作階段 (FAS)：當您使用 IAM 使用者或角色在 AWS 中執行動作時，系統會將您視為主體。當您使用某些服務時，您可能會執行一個動作，而該動作之後會在不同的服務中啟動另一個動作。FAS 使用主體的許可呼叫 AWS 服務，搭配請求 AWS 服務 以向下游服務發出請求。只有在服務收到需要與其他 AWS 服務 或資源互動才能完成的請求之後，才會提出 FAS 請求。在此情況下，您必須具有執行這兩個動作的許可。如需提出 FAS 請求時的政策詳細資訊，請參閱《轉發存取工作階段》。
  - 服務角色：服務角色是服務擔任的 IAM 角色，可代表您執行動作。IAM 管理員可以從 IAM 內建立、修改和刪除服務角色。如需詳細資訊，請參閱《IAM 使用者指南》中的建立角色以委派許可給 AWS 服務 服務。

- 服務連結角色 – 服務連結角色是一種連結到 AWS 服務 的服務角色類型。服務可以擔任代表您執行動作的角色。服務連結角色會顯示在您的 AWS 帳戶 中，並由該服務所擁有。IAM 管理員可以檢視，但不能編輯服務連結角色的許可。
- 在 Amazon EC2 上執行的應用程式 – 針對在 EC2 執行個體上執行並提出 AWS CLI 和 AWS API 請求的應用程式，您可以使用 IAM 角色來管理暫時憑證。這是在 EC2 執行個體內儲存存取金鑰的較好方式。如需指派 AWS 角色給 EC2 執行個體並提供其所有應用程式使用，您可以建立連接到執行個體的執行個體設定檔。執行個體設定檔包含該角色，並且可讓 EC2 執行個體上執行的程式取得暫時憑證。如需詳細資訊，請參閱《IAM 使用者指南》中的利用 IAM 角色來授予許可給 Amazon EC2 執行個體上執行的應用程式。

如需了解是否要使用 IAM 角色或 IAM 使用者，請參閱《IAM 使用者指南》中的建立 IAM 角色 (而非使用者) 的時機。

使用政策管理存取權

您可以透過建立政策並將其附加到 AWS 身分或資源，在 AWS 中控制存取。政策是 AWS 中的一個物件，當其和身分或資源建立關聯時，便可定義其許可。AWS 會在主體 (使用者、根使用者或角色工作階段) 發出請求時評估這些政策。政策中的許可，決定是否允許或拒絕請求。大部分政策以 JSON 文件形式儲存在 AWS 中。如需 JSON 政策文件結構和內容的詳細資訊，請參閱《IAM 使用者指南》中的 JSON 政策概觀。

管理員可以使用 AWS JSON 政策來指定誰可以存取哪些內容。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。

預設情況下，使用者和角色沒有許可。若要授與使用者對其所需資源執行動作的許可，IAM 管理員可以建立 IAM 政策。然後，管理員可以將 IAM 政策新增至角色，使用者便能擔任這些角色。

IAM 政策定義該動作的許可，無論您使用何種方法來執行操作。例如，假設您有一個允許 `iam:GetRole` 動作的政策。具備該政策的使用者便可以從 AWS Management Console、AWS CLI 或 AWS API 取得角色資訊。

身分型政策

身分型政策是可以附加到身分 (例如 IAM 使用者、使用者群組或角色) 的 JSON 許可政策文件。這些政策可控制身分在何種條件下能對哪些資源執行哪些動作。若要了解如何建立身分類型政策，請參閱《IAM 使用者指南》中的建立 IAM 政策。

身分型政策可進一步分類成內嵌政策或受管政策。內嵌政策會直接內嵌到單一使用者、群組或角色。受管政策則是獨立的政策，您可以將這些政策附加到 AWS 帳戶 中的多個使用者、群組和角色。受管政

策包含 AWS 管理政策和客戶管理政策。如需瞭解如何在受管政策及內嵌政策間選擇，請參閱 IAM 使用者指南中的在受管政策和內嵌政策間選擇。

資源型政策

資源型政策是連接到資源的 JSON 政策文件。資源型政策的最常見範例是 IAM 角色信任政策和 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。對於附加政策的資源，政策會定義指定的主體可以對該資源執行的動作以及在何種條件下執行的動作。您必須在資源型政策中指定主體。主體可以包括帳戶、使用者、角色、聯合身分使用者或 AWS 服務。

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) 可控制哪些委託人 (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 Simple Storage Service (Amazon S3)、AWS WAF 和 Amazon VPC 是支援 ACL 的服務範例。若要進一步了解 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 支援其他較少見的政策類型。這些政策類型可設定較常見政策類型授與您的最大許可。

- 許可界限 – 許可範圍是一種進階功能，可供您設定身分型政策能授予 IAM 實體 (IAM 使用者或角色)的最大許可。您可以為實體設定許可界限。所產生的許可會是實體的身分型政策和其許可界限的交集。會在 `Principal` 欄位中指定使用者或角色的資源型政策則不會受到許可界限限制。所有這類政策中的明確拒絕都會覆寫該允許。如需許可範圍的更多相關資訊，請參閱《IAM 使用者指南》中的 IAM 實體許可範圍。
- 服務控制政策 (SCP) – SCP 是 JSON 政策，可指定 AWS Organizations 中組織或組織單位 (OU) 的最大許可。AWS Organizations 服務可用來分組和集中管理您企業所擁有的多個 AWS 帳戶。若您啟用組織中的所有功能，您可以將服務控制政策 (SCP) 套用到任何或所有帳戶。SCP 會限制成員帳戶中實體的許可，包括每個 AWS 帳戶根使用者。如需組織和 SCP 的更多相關資訊，請參閱《AWS Organizations 使用者指南》中的 SCP 運作方式。
- 工作階段政策 – 工作階段政策是一種進階政策，您可以在透過編寫程式的方式建立角色或聯合使用者的暫時工作階段時，作為參數傳遞。所產生工作階段的許可會是使用者或角色的身分型政策和工作

階段政策的交集。許可也可以來自資源型政策。所有這類政策中的明確拒絕都會覆寫該允許。如需更多資訊，請參閱《IAM 使用者指南》中的工作階段政策。

多種政策類型

將多種政策類型套用到請求時，其結果形成的許可會更為複雜、更加難以理解。如需瞭解 AWS 在涉及多種政策類型時如何判斷是否允許一項請求，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 搭配 IAM 的運作方式”
- 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 搭配 IAM 的運作方式

在您使用存取權管理的存取權前AWS IoT Greengrass，您應該先了解可與搭配使用的可搭配使用的可搭配使用的服務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	是
暫時性憑證	是
服務連結角色	否
服務角色	是

如要取得其他AWS服務如何與使用者搭配運作的高階檢視，請參閱《IAM 使用者指南》中的可搭配運作的AWS服務。

適用於 AWS IoT Greengrass 的身分型政策

使用 IAM 身分型政策，您可以指定允許或拒絕的動作和資源，以及在何種條件下允許或拒絕動作。AWS IoT Greengrass 支援特定動作、資源及條件金鑰。如要了解您在政策中使用的所有元素，請參閱《IAM 使用者指南》中的有關使用者指南中的所有元素。

動作

管理員可以使用 AWS JSON 政策來指定誰可以存取哪些內容。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。政策動作的名稱通常會和相關聯的 AWS API 操作相同。有一些例外狀況，例如沒有相符的 API 作業的僅限許可動作。也有一些作業需要政策中的多個動作。這些額外的動作稱為相依動作。

政策會使用動作來授予執行相關聯操作的許可。

AWS IoT Greengrass 的政策動作會在動作之前使用 `greengrass:` 字首。例如，若要允許某人使用`ListCoreDevices` API 操作列出其中的核心裝置AWS 帳戶，請在其政策中包含`greengrass:ListCoreDevices`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。

如要在單一陳述式中指定多個動作，請在方括號 (`[]`) 中列出動作，並以逗號進行分隔，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》AWS IoT Greengrass中的「定義動作」。

資源

管理員可以使用 AWS JSON 政策來指定誰可以存取哪些內容。也就是說，哪個主體在什麼条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。陳述式必須包含 `Resource` 或 `NotResource` 元素。最佳實務是使用其 Amazon Resource Name (ARN) 來指定資源。您可以針對支援特定資源類型的動作 (稱為資源層級許可) 來這麼做。

對於不支援資源層級許可的動作 (例如列出作業)，請使用萬用字元 (*) 來表示陳述式適用於所有資源。

```
"Resource": "*"
```

下表包含可用於政策陳述式的 `Resource` 元素中的 AWS IoT Greengrass 資源 ARN。如需AWS IoT Greengrass動作支援的資源層級許可對應，請參閱《IAM 使用者指南》AWS IoT Greengrass中的「定義動作」。

某些 AWS IoT Greengrass 動作 (例如，某些列示操作) 無法在特定資源上執行。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 ARN，請將它們列在括號 ([]) 之間，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 資源名稱 (ARN) 和AWS服務命名空間Amazon Web Services 一般參考。

條件金鑰

管理員可以使用 AWS JSON 政策來指定誰可以存取哪些內容。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 (或 Condition 區塊) 可讓您指定使陳述式生效的條件。Condition 元素是選用項目。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。

若您在陳述式中指定多個 Condition 元素，或是在單一 Condition 元素中指定多個金鑰，AWS 會使用邏輯 AND 操作評估他們。若您為單一條件索引鍵指定多個值，AWS 會使用邏輯 OR 操作評估條件。必須符合所有條件，才會授予陳述式的許可。

您也可以在指定條件時使用預留位置變數。例如，您可以只在使用者使用其 IAM 使用者名稱標記時，將存取資源的許可授予該 IAM 使用者。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM 政策元素：變數和標籤。

AWS 支援全域條件索引鍵和服務特定的條件索引鍵。若要查看 AWS 全域條件索引鍵，請參閱《IAM 使用者指南》中的 AWS 全域條件內容金鑰。

範例

若要檢視 AWS IoT Greengrass 身分型政策範例，請參閱 the section called “身分型政策範例”。

AWS IoT Greengrass 的資源型政策

AWS IoT Greengrass 不支援資源型政策。

存取控制清單 (ACL)

AWS IoT Greengrass 不支援 ACL。

以 AWS IoT Greengrass 標籤為基礎的授權

您可以將標籤連接到支援的 AWS IoT Greengrass 資源，或是在請求中將標籤傳遞給 AWS IoT Greengrass。若要根據標籤控制存取，請使用、或aws:TagKeys條件金鑰aws:ResourceTag/${TagKey}，aws:RequestTag/${TagKey}在政策的條件元素中提供標籤資訊。如需詳細資訊，請參閱標記您的 資源。

適用於的 IAM 角色AWS IoT Greengrass

IAM 角色是您 AWS 帳戶 中具備特定許可的實體。

將臨時憑證與 AWS IoT Greengrass 搭配使用

暫時登入資料可用來登入聯合、擔任有關角色，或是取得跨帳戶角色。您取得暫時安全憑證的方式是透過呼叫AWS STS API 操作，例如AssumeRole或GetFederationToken。

在 Greengrass 核心上，裝置角色的臨時認證可供 Greengrass 元件使用。如果您的組件使用AWS SDK，則不需要添加邏輯來獲取憑據，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核心裝置使用服務角色來允許 Greengrass 元件和 Lambda 函數代表您存取部分AWS資源。如需詳細資訊，請參閱the section called “授權核心設備與AWS服務”。

AWS IoT Greengrass 會使用服務角色代表您存取部分 AWS 資源。如需詳細資訊，請參閱 Greengrass 服務角色。

AWS IoT Greengrass 的身分型政策範例

根據預設，IAM 使用者和角色不具備建立或修改 AWS IoT Greengrass 資源的許可。他們也無法使用 AWS Management Console、AWS CLI 或 AWS API 執行任務。IAM 管理員必須建立 IAM 政策，授予使用者和角色在指定資源上執行特定 API 操作的所需許可。管理員接著必須將這些政策連接至需要這些許可的 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 受管政策並朝向最低權限許可的目標邁進 – 若要開始授予許可給使用者和工作負載，請使用 AWS 受管政策，這些政策會授予許可給許多常用案例。它們可在您的 AWS 帳戶 中使用。我們建議您定義特定於使用案例的 AWS 客戶管理政策，以便進一步減少許可。如需詳細資訊，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 套用許可的詳細資訊，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。您也可以使用條件來授予對服務動

作的存取權，前提是透過特定 AWS 服務 (例如 AWS CloudFormation) 使用條件。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，可協助您編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 IAM Access Analyzer 政策驗證。
- 需要多重要素驗證 (MFA) – 如果存在 AWS 帳戶 中需要 IAM 使用者或根使用者的情況，請開啟 MFA 提供額外的安全性。若要在呼叫 API 操作時要求 MFA，請將 MFA 條件新增至您的政策。如需詳細資訊，請參閱 IAM 使用者指南中的設定 MFA 保護的 API 存取。

有關 IAM 中最佳實務的詳細資訊，請參閱 IAM 使用者指南中的 IAM 安全最佳實務。

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 使用者檢視連接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上，或是使用 AWS CLI 或 AWS API 透過編寫程式的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": "*"
        }
    ]
}
```

授權核心設備與AWS服務

AWS IoT Greengrass核心裝置使用AWS IoT Core證書提供程序授權調用AWS服務。所以此AWS IoT Core憑據提供程序允許設備使用其 X.509 證書作為唯一設備標識進行身份驗證AWS請求。這讓您不需要將AWS存取金鑰 ID 和私密存取金鑰AWS IoT Greengrass核心裝置。如需詳細資訊，請參閱「」授權直接呼叫AWS服務中的AWS IoT Core開發人員指南。

當您運行AWS IoT Greengrass核心軟件，您可以選擇配置AWS核心設備所需的資源。其中包括AWS Identity and Access Management(IAM) 角色，您的核心設備通過AWS IoT Core登入資料提供者。使用`--provision true`參數來配置允許核心設備獲取臨時AWS登入資料。此參數還配置一個AWS IoT指向此 IAM 角色的角色別名。您可以指定 IAM 角色的名稱，AWS IoT角色別名。如果指定`--provision true`沒有這些其他名稱參數，Greengrass 核心設備將創建並使用以下默認資源：

- IAM 角色：`GreengrassV2TokenExchangeRole`

  此角色具有名為`GreengrassV2TokenExchangeRoleAccess`並建立信任關係`credentials.iot.amazonaws.com`來擔任此角色。該策略包括核心設備的最低權限。

> **Important**
>
> 此策略不包括對 S3 存儲桶中文件的訪問權限。您必須向角色添加許可，以允許核心設備從 S3 儲存貯體檢索組件。如需詳細資訊，請參閱 允許存取 S3 儲存貯體的存取。

- AWS IoT角色別名：`GreengrassV2TokenExchangeRoleAlias`

  此角色別名指的是 IAM 角色。

如需詳細資訊，請參閱 步驟 3：安裝AWS IoT Greengrass核心軟體。

您還可以為現有核心設備設置角色別名。若要執行此作業，請將`iotRoleAlias`配置參數的Greengrass 核組成。

您可以獲得臨時AWSIAM 角色若要執行的登入資料AWS操作。如需詳細資訊，請參閱 與AWS服務互動。

主題

- 核心裝置的服務角色許可
- 允許存取 S3 儲存貯體的存取

核心裝置的服務角色許可

此角色允許下列服務擔任此角色：

- `credentials.iot.amazonaws.com`

如果您使用AWS IoT Greengrass核心軟件創建此角色時，它使用以下權限策略允許核心設備連接並將日誌發送到AWS。策略的名稱默認為 IAM 角色的名稱，以`Access`。例如，如果您使用默認 IAM 角色名稱，則此策略的名稱為`GreengrassV2TokenExchangeRoleAccess`。

Greengrass nucleus v2.5.0 and later

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

```
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    }
  ]
}
```

v2.4.x

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:DescribeCertificate",
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    }
  ]
}
```

Earlier than v2.4.0

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:DescribeCertificate",
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
```

```
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "iot:Connect",
        "iot:Publish",
        "iot:Subscribe",
        "iot:Receive",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    }
  ]
}
```

允許存取 S3 儲存貯體的存取

默認核心設備角色不允許核心設備訪問 S3 存儲桶。要部署 S3 存儲桶中包含工件的組件，您必須將`s3:GetObject`權限以允許核心設備下載組件工件。您可以向核心設備角色添加新策略以授予此權限。

添加允許訪問 Amazon S3 中組件項目的策略

1. 建立名為`component-artifact-policy.json`並將下列 JSON 複製到檔案。此政策允許存取 S3 儲存貯體中的所有檔案。Replace*#######*，其中包括 S3 儲存貯體的名稱，以允許核心設備存取。

   ```
   {
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Action": [
           "s3:GetObject"
         ],
         "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*"
       }
     ]
   }
   ```

2. 執行下列命令，從`component-artifact-policy.json`。

Linux or Unix

```
aws iam create-policy \
  --policy-name MyGreengrassV2ComponentArtifactPolicy \
  --policy-document file://component-artifact-policy.json
```

Windows Command Prompt (CMD)

```
aws iam create-policy ^
  --policy-name MyGreengrassV2ComponentArtifactPolicy ^
  --policy-document file://component-artifact-policy.json
```

PowerShell

```
aws iam create-policy `
  --policy-name MyGreengrassV2ComponentArtifactPolicy `
  --policy-document file://component-artifact-policy.json
```

從輸出中的政策元數據複製政策 Amazon 資源名稱 (ARN)。在下一步驟中，您可以使用此 ARN 將此政策連接至核心設備角色。

3. 執行下列命令將政策連接至核心裝置角色。Replace#### 2-####中指定的角色名稱，其中包括您運行AWS IoT Greengrass核心軟體。然後，將政策 ARN 取代成先前步驟中的 ARN。

Linux or Unix

```
aws iam attach-role-policy \
  --role-name GreengrassV2TokenExchangeRole \
  --policy-arn
 arn:aws:iam::123456789012:policy/MyGreengrassV2ComponentArtifactPolicy
```

Windows Command Prompt (CMD)

```
aws iam attach-role-policy ^
  --role-name GreengrassV2TokenExchangeRole ^
  --policy-arn
 arn:aws:iam::123456789012:policy/MyGreengrassV2ComponentArtifactPolicy
```

PowerShell

```
aws iam attach-role-policy `
  --role-name GreengrassV2TokenExchangeRole `
  --policy-arn
 arn:aws:iam::123456789012:policy/MyGreengrassV2ComponentArtifactPolicy
```

如果命令沒有輸出，則會成功，並且您的核心設備可以訪問您上傳到此 S3 存儲桶的工件。

安裝程式佈建資源的最低 IAM 政策

安裝 AWS IoT Greengrass Core 軟體時，您可以佈建必要的 AWS 資源，例如裝置的 AWS IoT 物件和 IAM 角色。您也可以將本機開發工具部署到裝置。安裝程式需要 AWS 認證，以便它可以在您的 AWS 帳戶. 如需詳細資訊，請參閱 安裝 AWS IoT Greengrass 核心軟體。

下列範例原則包含安裝程式佈建這些資源所需的最小動作集。如果您指定安裝程式的`--provision`引數，則需要這些權限。以您的*#######*AWS 帳戶 識別碼，並將 *GreengrassV2TokenExchangeRole* 取代為您以安裝程式引數指定的權杖交換角色名稱。`--tes-role-name`

> **Note**
>
> 只有在您為安裝程式指定`--deploy-dev-tools`引數時，才需要`DeployDevTools`原則陳述式。

Greengrass nucleus v2.5.0 and later

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CreateTokenExchangeRole",
            "Effect": "Allow",
            "Action": [
                "iam:AttachRolePolicy",
                "iam:CreatePolicy",
                "iam:CreateRole",
                "iam:GetPolicy",
                "iam:GetRole",
```

```
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::account-id:role/GreengrassV2TokenExchangeRole",
                "arn:aws:iam::account-
id:policy/GreengrassV2TokenExchangeRoleAccess",
                "arn:aws:iam::aws:policy/GreengrassV2TokenExchangeRoleAccess"
            ]
        },
        {
            "Sid": "CreateIoTResources",
            "Effect": "Allow",
            "Action": [
                "iot:AddThingToThingGroup",
                "iot:AttachPolicy",
                "iot:AttachThingPrincipal",
                "iot:CreateKeysAndCertificate",
                "iot:CreatePolicy",
                "iot:CreateRoleAlias",
                "iot:CreateThing",
                "iot:CreateThingGroup",
                "iot:DescribeEndpoint",
                "iot:DescribeRoleAlias",
                "iot:DescribeThingGroup",
                "iot:GetPolicy"
            ],
            "Resource": "*"
        },
        {
            "Sid": "DeployDevTools",
            "Effect": "Allow",
            "Action": [
                "greengrass:CreateDeployment",
                "iot:CancelJob",
                "iot:CreateJob",
                "iot:DeleteThingShadow",
                "iot:DescribeJob",
                "iot:DescribeThing",
                "iot:DescribeThingGroup",
                "iot:GetThingShadow",
                "iot:UpdateJob",
                "iot:UpdateThingShadow"
            ],
            "Resource": "*"
```

```
        }
    ]
}
```

Earlier than v2.5.0

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CreateTokenExchangeRole",
            "Effect": "Allow",
            "Action": [
                "iam:AttachRolePolicy",
                "iam:CreatePolicy",
                "iam:CreateRole",
                "iam:GetPolicy",
                "iam:GetRole",
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::account-id:role/GreengrassV2TokenExchangeRole",
                "arn:aws:iam::account-
id:policy/GreengrassV2TokenExchangeRoleAccess",
                "arn:aws:iam::aws:policy/GreengrassV2TokenExchangeRoleAccess"
            ]
        },
        {
            "Sid": "CreateIoTResources",
            "Effect": "Allow",
            "Action": [
                "iot:AddThingToThingGroup",
                "iot:AttachPolicy",
                "iot:AttachThingPrincipal",
                "iot:CreateKeysAndCertificate",
                "iot:CreatePolicy",
                "iot:CreateRoleAlias",
                "iot:CreateThing",
                "iot:CreateThingGroup",
                "iot:DescribeEndpoint",
                "iot:DescribeRoleAlias",
                "iot:DescribeThingGroup",
                "iot:GetPolicy"
```

```
            ],
            "Resource": "*"
        },
        {
            "Sid": "DeployDevTools",
            "Effect": "Allow",
            "Action": [
                "greengrass:CreateDeployment",
                "iot:CancelJob",
                "iot:CreateJob",
                "iot:DeleteThingShadow",
                "iot:DescribeJob",
                "iot:DescribeThing",
                "iot:DescribeThingGroup",
                "iot:GetThingShadow",
                "iot:UpdateJob",
                "iot:UpdateThingShadow"
            ],
            "Resource": "*"
        }
    ]
}
```

Greengrass 服務角色

Greengrass 服務角色是一種 AWS Identity and Access Management (IAM) 服務角色，可授權代表 AWS IoT Greengrass您從服AWS務存取資源。此角色可讓AWS IoT Greengrass您驗證用戶端裝置的身分識別，並管理核心裝置連線資訊。

> **Note**
>
> AWS IoT Greengrass V1也會使用此角色來執行基本工作。如需詳細資訊，請參閱開發人員指 AWS IoT Greengrass V1南中的 Greengrass 服務角色。

若AWS IoT Greengrass要允許存取您的資源，Greengrass 服務角色必須與您的關聯，AWS 帳戶並指定AWS IoT Greengrass為受信任的實體。角色必須包含AWSGreengrassResourceAccessRolePolicy受管理的原則或自訂原則，該原則可針對您使用的AWS IoT Greengrass功能定義對等權限。 AWS維護此策略，該策略定義了AWS

IoT Greengrass用於存取AWS資源的一組權限。如需詳細資訊，請參閱 AWS 受管政策：AWSGreengrassResourceAccessRolePolicy。

您可以在各處重複使用相同的 Greengrass 服務角色AWS 區域，但您必須在每個使用位置將AWS 區域其與您的帳戶建立關聯。AWS IoT Greengrass如果未在目前設定服務角色AWS 區域，核心裝置將無法驗證用戶端裝置，而且無法更新連線資訊。

下列各節說明如何使用或建立及管理 Greengrass 服務角色。AWS Management Console AWS CLI

主題

- 管理綠色服務角色 (主控台)
- 管理綠色服務角色 (CLI)
- 另請參閱

> **Note**
>
> 除了授權服務層級存取的服務角色之外，您還可以將權杖交換角色指派給 Greengrass 核心裝置。權杖交換角色是個別的 IAM 角色，可控制核心裝置上 Greengrass 元件和 Lambda 函數存取服務的方式。AWS如需詳細資訊，請參閱 授權核心設備與AWS服務。

管理綠色服務角色 (主控台)

您可以透過 AWS IoT 主控台，輕鬆管理 Greengrass 服務角色。例如，當您為核心裝置設定用戶端裝置探索時，主控台會檢查您AWS 帳戶是否已連接至目前的 Greengrass 服務角色。AWS 區域如果未連接，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”。

您可以使用主控台執行下列角色管理工作：

主題

- 尋找您的 Greengrass 服務角色 (主控台)
- 建立 Greengrass 服務角色 (主控台)
- 變更 Greengrass 服務角色 (主控台)
- 分離 Greengrass 服務角色 (主控台)

> **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)

使用下列步驟尋找目前AWS IoT Greengrass使用的服務角色AWS 區域。

1. 導覽至 AWS IoT主控台。
2. 在導覽窗格中，選擇設定。
3. 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。

   如果您看不到服務角色，主控台可以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	值
名稱	`Greengrass_ServiceRole`
信任實體	`AWS service: greengrass`
政策	AWSGreengrassResourceAccessRolePolicy

> **Note**
>
> 如果您使用AWS IoT Greengrass V1裝置設定指令碼建立此角色，則角色名稱為`GreengrassServiceRole_`*`random-string`*。

當您為核心裝置設定用戶端裝置探索時，主控台會檢查 Greengrass 服務角色是否與目前裝置AWS 帳戶相關聯。AWS 區域如果沒有，主控台會提示您AWS IoT Greengrass允許代表您讀取和寫入AWS服務。

如果您授與權限，控制台會檢查您的AWS 帳戶. `Greengrass_ServiceRole`

- 如果角色存在，則主控台會將服務角色附加到目前AWS 帳戶的服務角色AWS 區域。
- 如果角色不存在，控制台會創建一個默認的 Greengrass 服務角色，並將其附加到當前的AWS 帳戶。AWS 區域

> **Note**
>
> 如果要使用自訂角色政策建立服務角色，請使用 IAM 主控台建立或修改角色。如需詳細資訊，請參閱《IAM 使用者指南》中的建立角色以委派許可給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原則中加入`aws:SourceArn`和`aws:SourceAccount`全域條件內容金鑰，以協助避免混淆的副安全性問題。條件內容索引鍵會限制存取權，只允許來自指定帳戶和 Greengrass 工作區的要求。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，請返回主AWS IoT控台並將角色附加到您的AWS 帳戶. 您可以在「設置」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)

使用下列程序來選擇不同的 Greengrass 服務角色，以連接至主控台AWS 帳戶中AWS 區域目前選取的您的角色。

1. 導覽至 AWS IoT主控台。
2. 在導覽窗格中，選擇設定。
3. 在「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，並顯示您在定義AWS IoT Greengrass為受信任實體AWS 帳戶的 IAM 角色。
4. 選擇要附加的 Greengrass 色服務角色。
5. 選擇 [附加角色]。

分離 Greengrass 服務角色 (主控台)

使用下列程序，將 Greengrass 服務角色從目前AWS帳戶中卸離。AWS 區域這會撤銷存AWS IoT Greengrass取目前AWSAWS 區域服務的權限。

> **Important**
>
> 分離服務角色可能會中斷作用中的操作。

1. 導覽至 AWS IoT主控台。
2. 在導覽窗格中，選擇設定。
3. 在 [Greengrass 服務角色] 下，選擇 [卸離角色]。
4. 在確認對話方塊中，選擇 Detach (分離)。

> **Note**
>
> 如果您不再需要該角色，可以在 IAM 主控台中將其刪除。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。
> 其他角色可能會允許 AWS IoT Greengrass 存取您的資源。若要尋找允許AWS IoT Greengrass代表您採用權限的所有角色，請在 IAM 主控台的 [角色] 頁面上，在 [受信任的實體] 欄中尋找包含AWS服務:greengrass 的角色。

管理綠色服務角色 (CLI)

在下列程序中，我們假設AWS Command Line Interface已安裝並設定為使用您的AWS 帳戶. 如需詳細資訊，請參閱《AWS Command Line Interface使用指南》AWS CLI中的〈安裝、更新AWS CLI和解除安裝〉和〈設定〉。

您可以使用 AWS CLI 進行下列角色管理任務：

主題

- 取得 Greengrass 服務角色 (CLI)
- 建立 Greengrass 服務角色 (CLI)
- 移除 Greengrass 服務角色 (CLI)

取得 Greengrass 服務角色 (CLI)

請使用下列程序來瞭解 Greengrass 服務角色是否與中的AWS 帳戶. AWS 區域

- 取得服務角色。將##取代為您的 AWS 區域 (例如，`us-west-2`)。

```
aws greengrassv2 get-service-role-for-account --region region
```

如果 Greengrass 服務角色已與您的帳戶相關聯，則要求會傳回下列角色中繼資料。

```
{
  "associatedAt": "timestamp",
  "roleArn": "arn:aws:iam::account-id:role/path/role-name"
}
```

如果請求未傳回角色中繼資料，則您必須建立服務角色 (如果該角色不存在)，並將其與AWS 區域.

建立 Greengrass 服務角色 (CLI)

請使用下列步驟來建立角色，並將其與您的角色相關聯AWS 帳戶。

使用 IAM 建立服務角色

1. 使用允許 AWS IoT Greengrass 擔任角色之信任政策的角色。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原則中加入`aws:SourceArn`和`aws:SourceAccount`全域條件內容金鑰，以協助避免混淆的副安全性問題。條件內容索引鍵會限制存取權，只允許來自指定帳戶和 Greengrass 工作區的要求。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 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": {
           "ArnLike": {
             "aws:SourceArn": "arn:aws:greengrass:region:account-id:*"
           },
           "StringEquals": {
   ```

```
          "aws:SourceAccount": "account-id"
        }
      }
    }
  ]
}'
```

Windows Command Prompt (CMD)

```
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\"}}}]}"
```

PowerShell

```
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
```

若要將服務角色與您的 AWS 帳戶

- 將角色與您的帳戶相關聯。將*## arn* 替換為服務角色 ARN 和*##*為您的AWS 區域（例如，）。us-west-2

```
aws greengrassv2 associate-service-role-to-account --role-arn role-arn --
region region
```

如果成功，請求會傳回下列回應。

```
{
  "associatedAt": "timestamp"
}
```

移除 Greengrass 服務角色 (CLI)

請使用下列步驟取消 Greengrass 服務角色與您的. 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 greengrassv2 disassociate-service-role-from-account --region region
```

如果成功，會傳回下列回應。

```
{
  "disassociatedAt": "timestamp"
}
```

> **Note**
>
> 如果您沒有在任何服務角色中使用，則應刪除該服務角色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

- 在 IAM 使用者指南中建立角色以將權限委派給AWS服務
- 修改 IAM 使用者指南中的角色
- 在 IAM 使用者指南中刪除角色或執行個體設定檔
- AWS IoT Greengrass《指令參考》中的指AWS CLI令
  - associate-service-role-to-帳戶
  - disassociate-service-role-from-帳戶
  - get-service-role-for-帳戶
- 命令參考中的 IAM AWS CLI 命令
  - attach-role-policy
  - create-role
  - delete-role
  - delete-role-policy

AWS IoT Greengrass 的 AWS 受管政策

一個AWS受管理的策略是由建立和管理的獨立策略AWS。AWS受管理的策略旨在為許多常見使用案例提供權限，以便您可以開始將權限指派給使用者、群組和角色。

請記住，AWS受管理的政策可能不會為您的特定使用案例授與最低權限，因為這些權限適用於所有使用案例AWS客戶使用。建議您透過定義進一步減少權限客戶管理的政策特定於您的使用案例。

您無法更改 AWS 受管政策中定義的許可。如果AWS更新中定義的權限AWS受管理的原則，更新會影響所附加原則的所有主體識別 (使用者、群組和角色)。AWS最有可能更新AWS管理策略，當一個新的 AWS 服務已啟動或新的 API 操作可用於現有服務。

如需詳細資訊，請參閱《IAM 使用者指南》https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies_managed-vs-inline.html#aws-managed-policies中的 AWS 受管政策。

主題

- AWS 受管政策：AWSGreengrassFullAccess

- AWS 受管政策：AWSGreengrassReadOnlyAccess
- AWS 受管政策：AWSGreengrassResourceAccessRolePolicy
- AWS 受管政策的 AWS IoT Greengrass 更新項目

AWS 受管政策：AWSGreengrassFullAccess

您可將 AWSGreengrassFullAccess 政策連接到 IAM 身分。

此原則會授與允許主體完整存取所有權限的管理權限AWS IoT Greengrass動作。

許可詳細資訊

此政策包含以下許可：

- greengrass— 允許主參與者完全存取全部AWS IoT Greengrass動作。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:*"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS 受管政策：AWSGreengrassReadOnlyAccess

您可將 AWSGreengrassReadOnlyAccess 政策連接到 IAM 身分。

此原則會授與唯讀權限，這些權限允許主參與者檢視 (但不能修改) 中的資訊AWS IoT Greengrass。例如，具有這些權限的主體可以檢視部署到 Greengrass 核心裝置的元件清單，但無法建立部署來變更在該裝置上執行的元件。

許可詳細資訊

此政策包含以下許可：

- greengrass— 允許主參與者執行傳回項目清單或項目詳細資訊的動作。這包括開頭的 API 操作List或者Get。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:List*",
                "greengrass:Get*"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS 受管政策：AWSGreengrassResourceAccessRolePolicy

您可以附加AWSGreengrassResourceAccessRolePolicyIAM 實體的政策。AWS IoT Greengrass也會將此原則附加至允許的服務角色AWS IoT Greengrass代表您執行動作。如需詳細資訊，請參閱Greengrass 服務角色。

此原則會授與允許的管理權限AWS IoT Greengrass執行基本工作，例如擷取 Lambda 函數、管理AWS IoT設備陰影，並驗證 Greengrass 客戶端設備。

許可詳細資訊

此政策包含以下許可。

- greengrass— 管理格林格拉斯資源.
- iot(*Shadow) — 管理AWS IoT陰影名稱中有下列特殊識別碼。這些權限是必需的，以便AWS IoT Greengrass可以與核心設備進行通信。
  - *-gci—AWS IoT Greengrass使用此陰影來儲存核心裝置連線資訊，以便用戶端裝置可以探索並連線至核心裝置。
  - *-gcm—AWS IoT Greengrass V1使用此陰影通知核心裝置 Greengrass 群組的憑證授權單位 (CA) 憑證已輪替。
  - *-gda—AWS IoT Greengrass V1使用此陰影通知核心裝置部署。
  - GG_*— 未使用。

- `iot(DescribeThing和DescribeCertificate)` — 擷取有關的資訊AWS IoT東西和證書。這些權限是必需的，以便AWS IoT Greengrass可以驗證連接到核心設備的客戶端設備。如需詳細資訊，請參閱與本機 IoT 裝置互動。
- `lambda`— 檢索有關的信息AWS Lambda函數。此權限是必需的，以便AWS IoT Greengrass V1可以將 Lambda 函數部署到綠色核心。如需詳細資訊，請參閱在上執行 Lambda 函數AWS IoT Greengrass核心在AWS IoT Greengrass V1開發者指南。
- `secretsmanager`— 擷取的值AWS Secrets Manager名字開頭的秘密`greengrass-`。此權限是必需的，以便AWS IoT Greengrass V1可以將秘密管理員密碼部署到 Greengrass 核心。如需詳細資訊，請參閱將密碼部署到AWS IoT Greengrass核心在AWS IoT Greengrass V1開發者指南。
- `s3`— 從名稱包含的 S3 儲存貯體擷取檔案物件`greengrass`或者`sagemaker`。這些權限是必需的，以便AWS IoT Greengrass V1可以部署存放在 S3 儲存貯體中的機器學習資源。如需詳細資訊，請參閱機器學習資源在AWS IoT Greengrass V1開發者指南。
- `sagemaker`— 檢索有關亞馬遜的信息SageMaker機器學習推論模型。此權限是必需的，以便AWS IoT Greengrass V1可以將 ML 模型部署到綠核心。如需詳細資訊，請參閱執行機器學習推論在AWS IoT Greengrass V1開發者指南。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowGreengrassAccessToShadows",
            "Action": [
                "iot:DeleteThingShadow",
                "iot:GetThingShadow",
                "iot:UpdateThingShadow"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:iot:*:*:thing/GG_*",
                "arn:aws:iot:*:*:thing/*-gcm",
                "arn:aws:iot:*:*:thing/*-gda",
                "arn:aws:iot:*:*:thing/*-gci"
            ]
        },
        {
            "Sid": "AllowGreengrassToDescribeThings",
            "Action": [
                "iot:DescribeThing"
            ],
```

```
            "Effect": "Allow",
            "Resource": "arn:aws:iot:*:*:thing/*"
        },
        {
            "Sid": "AllowGreengrassToDescribeCertificates",
            "Action": [
                "iot:DescribeCertificate"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:iot:*:*:cert/*"
        },
        {
            "Sid": "AllowGreengrassToCallGreengrassServices",
            "Action": [
                "greengrass:*"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Sid": "AllowGreengrassToGetLambdaFunctions",
            "Action": [
                "lambda:GetFunction",
                "lambda:GetFunctionConfiguration"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Sid": "AllowGreengrassToGetGreengrassSecrets",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:secretsmanager:*:*:secret:greengrass-*"
        },
        {
            "Sid": "AllowGreengrassAccessToS3Objects",
            "Action": [
                "s3:GetObject"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:s3:::*Greengrass*",
```

```
                "arn:aws:s3:::*GreenGrass*",
                "arn:aws:s3:::*greengrass*",
                "arn:aws:s3:::*Sagemaker*",
                "arn:aws:s3:::*SageMaker*",
                "arn:aws:s3:::*sagemaker*"
            ]
        },
        {
            "Sid": "AllowGreengrassAccessToS3BucketLocation",
            "Action": [
                "s3:GetBucketLocation"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Sid": "AllowGreengrassAccessToSageMakerTrainingJobs",
            "Action": [
                "sagemaker:DescribeTrainingJob"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:sagemaker:*:*:training-job/*"
            ]
        }
    ]
}
```

AWS 受管政策的 AWS IoT Greengrass 更新項目

您可以檢視更新的詳細資訊AWS受管理的政策AWS IoT Greengrass從這項服務開始追蹤這些變更的時間。如需有關此頁面變更的自動警示，請訂閱AWS IoT Greengrass V2文件歷史記錄頁。

變更	描述	日期
AWS IoT Greengrass 已開始追蹤變更	AWS IoT Greengrass 已開始追蹤其 AWS 受管政策的變更。	2021 年 7 月 2 日

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。

若要限制 AWS IoT Greengrass 為資源提供另一項服務的許可，我們建議在資源政策中使用 `aws:SourceArn` 和 `aws:SourceAccount` 全域條件內容索引鍵。如果同時使用全域條件內容索引鍵，則在相同政策陳述式中使用`aws:SourceAccount` 值和 `aws:SourceArn` 值中的帳戶時，必須使用相同的帳戶 ID。

值`aws:SourceArn`Greengrass 是與`sts:AssumeRole`請求。

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::account-id:*`。

有關使用`aws:SourceArn`和`aws:SourceAccount`全域條件內容金鑰，請參閱建立 Greengrass 服務角色。

針對 AWS IoT Greengrass 的識別和存取問題進行故障診斷

請使用以下資訊來協助您診斷和修復使用 AWS IoT Greengrass 和 IAM 時發生的常見問題。

問題

- 我未獲授權在 AWS IoT Greengrass 中執行動作
- 我未獲得執行 iam:PassRole
- 我是管理員，並且想要允許其他人存取 AWS IoT Greengrass
- 我想要允許 AWS 帳戶 外的人員存取我的 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 AWS IoT Greengrass 中執行動作

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。

下面的例子發生錯誤時mateojacksonIAM 使用者嘗試檢視核心裝置的詳細資料，但沒有greengrass:GetCoreDevice許可。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to
 perform: greengrass:GetCoreDevice on resource: arn:aws:greengrass:us-
west-2:123456789012:coreDevices/MyGreengrassCore
```

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:coreDevices/MyGreengrassCore 動作存取 greengrass:GetCoreDevice 資源。

使用時可能遇到的 IAM 問題AWS IoT Greengrass。

我未獲得執行 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 允許您傳遞現有的角色至該服務，而無須建立新的服務角色或服務連結角色。若要執行此作業，您必須擁有將角色傳遞至該服務的許可。

當名為 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 沒有將角色傳遞至該服務的許可。

```
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。

如需任何協助，請聯絡您的 AWS 管理員。您的管理員提供您的登入憑證。

我是管理員，並且想要允許其他人存取 AWS IoT Greengrass

若要允許其他人存取 AWS IoT Greengrass，您必須針對需要存取的人員或應用程式建立 IAM 實體 (使用者或角色)。他們將使用該實體的憑證來存取 AWS。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。

若要立即開始使用，請參閱《IAM 使用者指南》中的建立您的第一個 IAM 委派使用者及群組。

我想要允許 AWS 帳戶 外的人員存取我的 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使用它來存取您的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，請參閱在另一個中提供 IAM 使用者存取權 AWS 帳戶您擁有的和提供隊服務的存取AWS 帳戶由第三方擁有在IAM User Guide。

AWS IoT Greengrass 不支援以資源為基礎的政策或存取控制清單 (ACL) 為基礎的跨帳戶存取。

允許裝置流量透過 Proxy 或防火牆

Greengrass 核心裝置和 Greengrass 元件會執行對服務和其他網站的輸出要求。AWS基於安全性考量，您可能會將輸出流量限制在少量端點和通訊埠範圍內。您可以使用下列有關端點和連接埠的資訊，限制透過 Proxy、防火牆或 Amazon VPC 安全群組的裝置流量。如需如何將核心裝置設定為使用 Proxy 的詳細資訊，請參閱連線至連接埠 443 或透過網路代理。

主題

- 基本操作的端點
- 具有自動佈建功能的安裝端點
- AWS所提供元件的端點

基本操作的端點

Greengrass 核心設備使用以下端點和端口進行基本操作。

擷取AWS IoT端點

取得您的AWS IoT端點AWS 帳戶，並儲存以供稍後使用。您的裝置會使用這些端點連線至AWS IoT。請執行下列操作：

1. 取得適AWS IoT用於您的AWS 帳戶.

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

   如果要求成功，回應看起來類似下列範例。

   ```
   {
     "endpointAddress": "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
   }
   ```

2. 取AWS IoT得您的AWS 帳戶.

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

如果要求成功，回應看起來類似下列範例。

```
{
  "endpointAddress": "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
}
```

端點	連線埠	必要	描述
greengrass-ats.iot. *region*.amazonaws.com	8443 或 443	是	用於資料平面作業，例如安裝部署和使用用戶端裝置。
device-data-prefix -ats.iot. *region*.amazonaws.com	MQTT：8883 或 443 HTTPS：8443 或 443	是	用於設備管理的數據平面操作，例如 MQTT 通信和陰影同步。AWS IoT Core
device-credentials-prefix .credentials.iot. *region*.amazonaws.com	443	是	用於取得AWS登入資料，核心裝置會使用這些登入資料從 Amazon S3 下載元件成品並

端點	連線埠	必要	描述
			執行其他操作。如需詳細資訊，請參閱 授權核心設備與AWS服務。
*.s3.amazonaws.com *.s3.*region*.amazonaws.com	443	是	用於部署。此格式包含*字元，因為端點前置字元是由內部控制的，而且可能隨時變更。

端點	連線埠	必要	描述
data.iot. *region*.amazonaws.com	443	否	如果核心裝置執行的 Greengrass 核心版本早於 v2.4.0，且設定為使用網路代理，則需要此選項。核心設備在代理後面AWS IoT Core時使用此端點進行 MQTT 通信。如需詳細資訊，請參閱 設定網路代理。

具有自動佈建功能的安裝端點

當您安裝具有自動資源佈建功能的 Core 軟體時，Greengrass AWS IoT Greengrass 核心裝置會使用下列端點和連接埠。

端點	連線埠	必要	描述
iot.*region*.amazonaws.com	443	是	用於創建AWS IoT資源和檢索有關現有AWS IoT資源的信息。

端點	連線埠	必要	描述
iam.amazonaws.com	443	是	用於建立 IAM 資源並擷取有關現有 IAM 資源的資訊。
sts.*region*.amazonaws.com	443	是	用來取得您的AWS 帳戶.
greengrass. *region*.amazonaws.com	443	否	如果您使用--deploy-dev-tools 引數將 Greengrass CLI 元件部署到核心裝置，則需要此參數。

AWS所提供元件的端點

Greengrass 核心裝置會根據其執行的軟體元件，使用其他端點。您可以在此開發人員指南中每個元件頁面的 [需求] 區段中找到每個AWS提供的元件所需的端點。如需更多詳細資訊，請參閱 AWS-提供的組件。

AWS IoT Greengrass 的合規驗證

要了解 AWS 服務 是否在特定法規遵循方案範圍內，請參閱法規遵循方案範圍內的 AWS 服務，並選擇您感興趣的法規遵循方案。如需一般資訊，請參閱 AWS 法規遵循方案。

您可使用 AWS Artifact 下載第三方稽核報告。如需詳細資訊，請參閱 AWS Artifact 中的下載報告。

您使用 AWS 服務 時的法規遵循責任取決於資料的敏感度、您的公司的合規目標，以及適用的法律和法規。AWS 提供以下資源協助您處理法規遵循事宜：

- 安全與合規快速入門指南 – 這些部署指南討論在 AWS 上部署以安全及合規為重心的基準環境的架構考量和步驟。
- Amazon Web Services 的 HIPAA 安全與法規遵循架構：本白皮書說明公司可如何運用 AWS 來建立符合 HIPAA 規定的應用程式。

> **Note**
>
> 並非全部的 AWS 服務 都符合 HIPAA 資格。如需詳細資訊，請參閱 HIPAA 資格服務參照。

- AWS 合規資源：這組手冊和指南可能適用於您的產業和位置。
- AWS 客戶合規指南：透過合規的角度了解共同的責任模式。這份指南橫跨多個架構 (包含國家標準技術研究所 (NIST)、支付卡產業安全標準委員會 (PCI) 和國際標準組織 (ISO))，總結保護 AWS 服務的最佳實務並將指導方針對應至安全控制。
- AWS Config 開發人員指南中的使用規則評估資源：AWS Config 服務可評估您的資源組態對於內部實務、業界準則和法規的合規狀態。
- AWS Security Hub – 此 AWS 服務 可供您全面檢視 AWS 中的安全狀態。Security Hub 使用安全控制，可評估您的 AWS 資源並檢查您的法規遵循是否符合安全業界標準和最佳實務。如需支援的服務和控制清單，請參閱 Security Hub controls reference。
- AWS Audit Manager – 此 AWS 服務 可協助您持續稽核 AWS 使用情況，以簡化管理風險與法規與業界標準的法規遵循方式。

AWS IoT Greengrass 中的恢復能力

所以此AWS全球基礎設施是以 Amazon Web Services 區域與可用區域為中心建置。EASEAWS 區域提供多個分開且隔離的實際可用區域，它們以低延遲、高輸送量和高度備援聯網功能相互連結。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、容錯能力和擴充能力，均較單一或多個資料中心的傳統基礎設施還高。

如需詳細資訊，請參閱 AWS 全球基礎設施。

除了 AWS 全球基礎設施，AWS IoT Greengrass 還提供數種功能，可協助支援資料的彈性和備份需求。

- 您可以將 Greengrass 核心裝置設備設定為將記錄寫入本機檔案系統和 CloudWatch 日誌。如果核心裝置中斷連線，裝置可以繼續記錄檔案系統中的消息。重新連接時，它會將日誌消息寫入 CloudWatch 日誌。如需詳細資訊，請參閱 監控AWS IoT Greengrass日誌。
- 如果核心設備在部署過程中斷電源，則在AWS IoT Greengrass核心軟件再次啟動。
- 如果核心裝置中斷網際網路連線，Greengrass 裝置可以繼續透過區域網路進行通訊。
- 你可以創作 Greengrass 組件，這些組件讀取串流管理員流並將數據發送到本地存儲目標。

AWS IoT Greengrass 中的基礎設施安全

AWS IoT Greengrass 為受管服務，受到 Amazon Web Services：安全程序概觀白皮書所述的 AWS 全球網路安全程序所保護。

您可使用 AWS 發佈的 API 呼叫，透過網路存取 AWS IoT Greengrass。用戶端必須支援 Transport Layer Security (TLS) 1.2 或更新版本。建議使用 TLS 1.3 或更新版本。用戶端也必須支援具備完美轉送私密 (PFS) 的密碼套件，例如臨時 Diffie-Hellman (DHE) 或橢圓曲線臨時 Diffie-Hellman (ECDHE)。現代系統 (如 Java 7 和更新版本) 大多會支援這些模式。

請求必須使用存取金鑰 ID 和與 IAM 委託人相關聯的私密存取金鑰來簽署。或者，您可以使用 AWS Security Token Service (AWS STS) 產生臨時安全憑證來簽署請求。

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線並進行驗證。AWS 雲端如需詳細資訊，請參閱 the section called “裝置身分驗證和授權”。

AWS IoT Greengrass 中的組態與漏洞分析

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 藉由提供識別安全性問題和偏離最佳實務的工具，來因應這些挑戰。您可以使用 AWS IoT Device Defender 來分析、稽核和監控連線的裝置，以偵測異常行為，並降低安全風險。AWS IoT Device Defender 可稽核裝置，以確保裝置遵守安全最佳實務，並偵測裝置上的異常行為。它可讓您的裝置實施一致的安全政策，並在裝置受到入侵時快速回應。如需詳細資訊，請參下列主題：

- 所以此Device Defender 組件

- AWS IoT Device Defender (在 AWS IoT Core 開發人員指南中)。

在 AWS IoT Greengrass 環境中，您應該注意下列考慮事項：

- 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不會為用户定義的 Greengrass 組件強制實施網絡隔離，無論它們是否在 Greengrass 容器中運行。因此，Greengrass ice 組件可以與系統內或網路外部執行的任何其他處理序進行通訊。

代碼完整性AWS IoT Greengrass V2

AWS IoT Greengrass部署軟件組件從AWS 雲端設備上運行AWS IoT Greengrass核心軟體。這些軟體組件包括AWS提供的組件和自定義組件上傳到您的AWS 帳戶。每個組件都由一個配方組成。配方定義組件的元數據和任意數量的工件，它們是組件二進制文件，如編譯的代碼和靜態資源。Amazon S3 中存放物件。

當您開發和部署 Greengrass 組件時，請按照以下基本步驟來處理AWS 帳戶和您的設備上：

1. 創建對象並將其上傳到 S3 存儲桶。
2. 從配方和工件中創建組件AWS IoT Greengrass服務，該服務計算密碼編譯散列每個神器。
3. 將組件部署到 Greengrass 核心設備，以下載並驗證每個工件的完整性。

AWS負責在將工件上傳到 S3 存儲桶後維護工件的完整性，包括將組件部署到 Greengrass 核心設備時。在將工件上傳到 S3 存儲桶之前，您負責保護軟件項目。您還負責保護對AWS 帳戶，包括您在其中上傳組件工件的 S3 存儲桶。

> **Note**
>
> Amazon S3 提供了一個名為 S3 對象鎖定的功能，您可以使用該功能防止 S3 存儲桶中的組件工件發生更改AWS 帳戶。您可以使用 S3 物件鎖定，讓物件遭到刪除或覆寫。如需詳細資訊，請參閱「」使用 S3 物件鎖定中的Amazon Simple Storage Service 用户指南。

時機AWS發佈公共組件，當您上傳自定義組件時，AWS IoT Greengrass計算每個組件工件的加密摘要。AWS IoT Greengrass更新組件配方以包含每個工件的摘要和用於計算該摘要的哈希算法。此摘要保證了工件的完整性，因為如果工件在AWS 雲端或在下載過程中，其文件摘要將不匹配AWS IoT Greengrass存儲在組件配方中。如需詳細資訊，請參閱「」組件處方參考中的對象。

將組件部署到核心設備時，AWS IoT Greengrass核心軟件下載組件配方和配方定義的每個組件工件。所以此AWS IoT Greengrass核心軟件計算每個下載的工件文件的摘要，並將其與配方中該工件的摘要進行比較。如果摘要不相符，則會失敗，AWS IoT Greengrass核心軟件會從設備的文件系統中刪除下載的工件。如需核心設備與AWS IoT Greengrass已安全，請參閱傳輸中加密。

您負責保護核心設備文件系統上的組件工件文件。所以此AWS IoT Greengrass核心軟件將工件保存到packages文件夾 Greengrass 的文件夾。您可以使用AWS IoT Device Defender來分析、審核和監控核心設備。如需詳細資訊，請參閱AWS IoT Greengrass 中的組態與漏洞分析。

AWS IoT Greengrass 和介面 VPC 端點 (AWS PrivateLink)

您可以建立介面 VPC 端點，在 VPC 和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服務中的元件、部署和核心裝置。介面端點採用這項技術 AWS PrivateLink，可讓您在沒有網際網路閘道、NAT 裝置、VPN 連線或AWS直 Connect 連線的情況下私密存取 AWS IoT Greengrass API。VPC 中的執行個體不需要公有 IP 地址，即能與 AWS IoT Greengrass API 通訊。 您的 VPC 與 AWS IoT Greengrass 之間的網路流量都會在 Amazon 網路的範圍內。

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
- 建立用於AWS IoT Greengrass控制平面作業的介面 VPC 端點
- 為 AWS IoT Greengrass 建立 VPC 端點政策
- 在 VPC 中操作AWS IoT Greengrass核心設備

AWS IoT Greengrass VPC 端點的考量事項

在為其設定介面 VPC 端點之前AWS IoT Greengrass，請先參閱 Amazon VPC 使用者指南中的界面端點屬性和限制。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。控制平面包括諸如CreateDeployment和之類的操作ListEffectiveDeployments。控制平面不包括操作，如ResolveComponentCandidates和發現，這是數據平面操作。
- AWS中國地區目前AWS IoT Greengrass不支援的 VPC 端點。

建立用於AWS IoT Greengrass控制平面作業的介面 VPC 端點

您可以使用 Amazon VPC 主AWS IoT Greengrass控台或 AWS Command Line Interface (AWS CLI) 為控制平面建立 VPC 端點。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的建立介面端點。

使用下列服務名稱建立 AWS IoT Greengrass 的 VPC 端點：

- com.amazonaws.*region*.greengrass

如果您為該端點啟用私有 DNS，您可以使用其區域的預設 DNS 名稱 (例如 `greengrass.us-east-1.amazonaws.com`)，向 AWS IoT Greengrass 發出 API 請求。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 AWS IoT Greengrass 建立 VPC 端點政策

您可以將端點策略附加到 VPC 端點，以AWS IoT Greengrass控制對控制平面操作的存取。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。
- 主參與者可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 AWS IoT Greengrass 端點政策的範例。附加至端點後，此政策會針對所有資源上的所有主體，授予列出的 AWS IoT Greengrass 動作的存取權限。

```
{
    "Statement": [
        {
            "Principal": "*",
            "Effect": "Allow",
            "Action": [
                "greengrass:CreateDeployment",
                "greengrass:ListEffectiveDeployments"
            ],
            "Resource": "*"
```

```
        }
    ]
}
```

在 VPC 中操作AWS IoT Greengrass核心設備

您可以操作 Greengrass 核心設備並在沒有公共互聯網訪問的情況下在 VPC 中執行部署。您至少必須使用對應的 DNS 別名來設定下列 VPC 端點。有關如何建立和使用 VPC 端點的詳細資訊，請參閱 Amazon VPC 使用者指南中的建立 VPC 端點。

> **Note**
>
> 自動建立 DNS 記錄的 VPC 功能已停用AWS IoT data和AWS IoT認證。若要連線這些端點，您必須手動建立私人 DNS 記錄。如需詳細資訊，請參閱介面端點的私有 DNS。如需 AWS IoT Core VPC 限制的詳細資訊，請參閱 VPC 端點的限制。

必要條件

- 您必須使用手動佈建步驟來安裝 AWS IoT Greengrass Core 軟體。如需詳細資訊，請參閱 透過手動佈建資源安裝 AWS IoT Greengrass 核心軟體。

限制

- 在 VPC 中操作 Greengrass 核心設備在中國地區和. AWS GovCloud (US) Regions
- 如需有關AWS IoT認證提供者 VPC 端點限制的AWS IoT data詳細資訊，請參閱限制。

設置您的核心設備以在 VPC 中運行

1. 取得您的AWS IoT端點AWS 帳戶，並儲存以供稍後使用。您的裝置會使用這些端點連線至AWS IoT。請執行下列操作：

   a. 取得適AWS IoT用於您的AWS 帳戶.

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

   如果要求成功，回應看起來類似下列範例。

```
{
  "endpointAddress": "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
}
```

b. 取AWS IoT得您的AWS 帳戶.

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

如果要求成功，回應看起來類似下列範例。

```
{
  "endpointAddress": "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
}
```

2. 為AWS IoT data和AWS IoT登入資料端點建立 Amazon VPC 界面：

a. 瀏覽至 VPC Endpoint 主控台，在左側功能表的「虛擬私有雲」下，選擇「端點」，然後選擇「建立端點」。

b. 在「建立端點」頁面中，指定下列資訊。

- 選擇服務類別的 AWS 服務。
- 針對 Service Name (服務名稱)，輸入關鍵字 `iot` 進行搜尋。在顯示的`iot`服務清單中，選擇端點。

  如果您為AWS IoT Core資料平面建立 VPC 端點，請為您的區域選擇AWS IoT Core資料平面 API 端點。端點的格式為 `com.amazonaws.region.iot.data`。

  如果您為AWS IoT Core認證提供者建立 VPC 端點，請為您的區域選擇AWS IoT Core認證提供者端點。端點的格式為 `com.amazonaws.region.iot.credentials`。

  > **Note**
  >
  > 中國地區資AWS IoT Core料平面的服務名稱格式為`cn.com.amazonaws.region.iot.data`。中國地區不支援為AWS IoT Core憑證提供者建立 VPC 端點。

- 針對 VPC 和 Subnets (子網路)，選擇要在其中建立端點的 VPC，以及要在其中建立端點網路的可用區域 (AZ)。

- 針對 Enable DNS name (啟用 DNS 名稱)，確定未選取 Enable for this endpoint (為此端點啟用)。AWS IoT Core資料平面和AWS IoT Core憑證提供者都不支援私有 DNS 名稱。
- 針對 Security group (安全群組)，選擇要與端點網路介面建立關聯的安全群組。
- 您可以選擇性地新增或移除標籤。標籤是您用來與端點建立關聯的名稱值對。

c. 若要建立 VPC 端點，請選擇 Create endpoint (建立端點)。

3. 建立端點後，在AWS PrivateLink端點的 [詳細資料] 索引標籤中，您會看到 DNS 名稱清單。您可以使用您在本節中建立的其中一個 DNS 名稱來設定您的私有託管區域。
4. 創建一個 Amazon S3 端點。如需詳細資訊，請參閱為 Amazon S3 建立 VPC 人雲端端點。
5. 如果您使用的是AWS提供的 Greengrass 元件，則可能需要其他端點和組態。若要檢視端點需求，請從AWS提供的元件清單中選取元件，然後查看「需求」區段。例如，記錄管理員元件需求建議此元件必須能夠對端點執行輸出要求`logs.region.amazonaws.com`。

   如果您使用自己的元件，您可能需要檢閱相依性並執行其他測試，以判斷是否需要任何其他端點。
6. 在 Greengrass 核配置中，`greengrassDataPlaneEndpoint`必須設置為。**`iotdata`**如需詳細資訊，請參閱 Greengrass 核組態。
7. 如果您在`us-east-1`區域中，請在 Greengrass 核組態**`REGIONAL`**中`s3EndpointType`將組態參數設定為。此功能適用於 Greengrass 2.11.3 或更新版本。

Example 範例：零組件組態

```
{
"aws.greengrass.Nucleus": {
   "configuration": {
      "awsRegion": "us-east-1",
      "iotCredEndpoint": "xxxxxx.credentials.iot.region.amazonaws.com",
      "iotDataEndpoint": "xxxxxx-ats.iot.region.amazonaws.com",
      "greengrassDataPlaneEndpoint": "iotdata",
      "s3EndpointType": "REGIONAL"
      ...
     }
   }
}
```

下表提供對應自訂私人 DNS 別名的相關資訊。

服務	VPC 端點服務名稱	VPC 端點類型	自訂私人 DNS 別名	備註
AWS IoT data	com.amazonaws. *region*.iot.d	界面	*prefix*-ats.iot. *region*.amazonaws.com	私人 DNS 記錄應與您帳戶的AWS IoT data端點相符：aws iot describe-endpoint --endpoint-type iot:Data-ATS 。
AWS IoT 憑證	com.amazonaws. *region*.iot.credentials	界面	*prefix*.credentials.iot.amazonaws.com	私人 DNS 記錄應與您的帳戶AWS IoT認證端點相符：aws iot describe-endpoint --endpoint-type iot:CredentialProvider 。

服務	VPC 端點服務名稱	VPC 端點類型	自訂私人 DNS 別名	備註
Amazon S3	com.amazonaws. *region*.s3	界面		系統會自動建立 DNS 記錄。

AWS IoT Greengrass 的安全最佳實務

本主題包含 AWS IoT Greengrass 的安全最佳實務。

盡可能授予最低的許可

請遵循元件的最低權限原則，方法是以無權限的使用者身分執行這些元件。除非絕對必要，否則組件不應以 root 身份運行。

在 IAM 角色中使用最低一組許可。限制使用*萬用字元Action和ResourceIAM 政策中的屬性。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”。

最小特權的最佳作法也適用於AWS IoT您附加到格林格拉斯核心的策略。

不要在 Greengrass 組件中對憑據進行硬編碼

不要在用戶定義的 Greengrass 組件中對憑據進行硬編碼。為了更妥善地保護您的登入資料：

- 若要與之互動AWS服務, 定義特定動作和資源的權限格林格拉斯核心裝置服務角色。
- 使用秘密管理器組件儲存您的認證。或者，如果函數使用AWSSDK，使用來自預設認證提供者鏈結的認證。

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)。我們建議您實作下列保護措施，即使存取核心裝置上的本機記錄檔需要 root 權限和存取權 CloudWatch 記錄檔需要 IAM 許可。

- 請勿在 MQTT 主題路徑中使用敏感資訊。

- 請勿在 AWS IoT Core 登錄中的裝置 (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 Greengrass 元件或 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：
  - 核心裝置
  - 元件
  - 部署
  - Loggers

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，請參閱讓裝置的時鐘保持同步最佳做法AWS IoT Core開發者指南。

加密套件建議

格雷格拉斯默認選擇設備上可用的最新 TLS 加密套件。請考慮停用裝置上舊版加密套件的使用。例如，CBC 加密套件。

如需詳細資訊，請參閱Java 密碼編譯組態。

另請參閱

- 安全性最佳做法AWS IoT Core在AWS IoT開發者指南
- 工業物聯網解決方案十大安全黃金法則在物聯網AWS官方部落格

AWS IoT Device Tester對於 AWS IoT Greengrass V2 使用

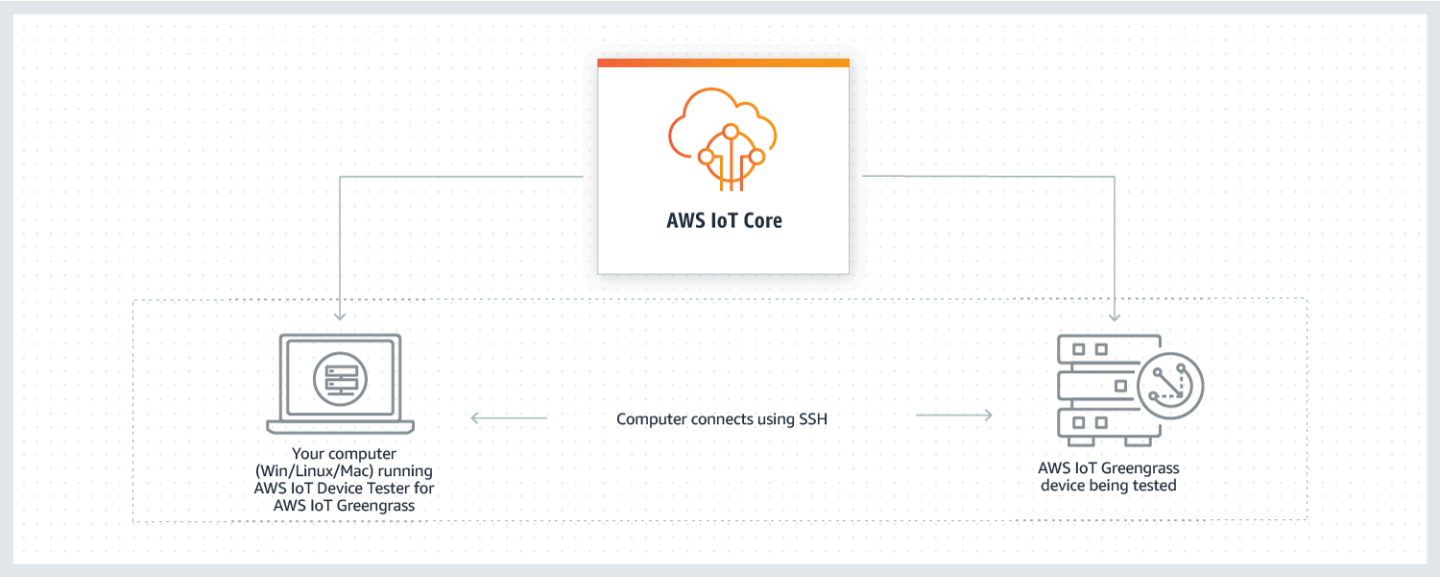
AWS IoT裝置測試器 (IDT) 是可下載的測試架構，可讓您驗證 IoT 裝置。您可以使用 IDT AWS IoT Greengrass 來運行AWS IoT Greengrass資格套件，以及為您的設備創建和運行自定義測試套件。

IDT for AWS IoT Greengrass 在連接到要測試之裝置的主機電腦上執行 (Windows、macOS 或 Linux)。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資格套房

使AWS IoT Device Tester用 AWS IoT Greengrass V2 來驗證 AWS IoT Greengrass Core 軟體是否在您的硬體上執行，並且可以與AWS 雲端. 它還執行 end-to-end 測試與AWS IoT Core. 例如，它會驗證您的裝置是否可以部署元件並對其進行升級。

如果要將硬體新增至AWS Partner裝置目錄，請執行AWS IoT Greengrass資格套件以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。



IDT in AWS IoT Greengrass V2 使用測試套件和測試組的概念組織測試。

- 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。
- 測試群組是與特定功能 (例如元件部署) 相關的一組個別測試。

如需詳細資訊，請參閱 使用 IDT 執行 AWS IoT Greengrass 資格套件。

自訂測試套件

從 IDT v4.0.1 開始，適用於 AWS IoT Greengrass V2 的 IDT 將標準化的組態設定和結果格式與測試套件環境結合在一起，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添加自定義測試，也可以將其提供給客戶進行設備驗證。

測試編寫者如何配置自定義測試套件確定運行自定義測試套件所需的設置配置。如需更多詳細資訊，請參閱 使用 IDT 開發和運行自己的測試套件。

AWS IoT Greengrass V2 的 AWS IoT Device Tester 支援版本

本主題列出 AWS IoT Greengrass V2 支援的 IDT 版本。最佳作法是，建議您使用支援目標 AWS IoT Greengrass V2 版本的最新 IDT 版本 AWS IoT Greengrass 。新版本的 AWS IoT Greengrass 可能會要求您下載適用於 AWS IoT Greengrass V2 的新版 IDT。如果 AWS IoT Greengrass V2 的 IDT 與您正在使用的 AWS IoT Greengrass 版本不兼容，則在啟動測試運行時收到通知。

下載軟體即表示您同意AWS IoT Device Tester 授權合約。

> **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。

適用於 V2 的最新 IDT 版本 AWS IoT Greengrass

您可以將此版本的 IDT 用於 AWS IoT Greengrass V2，並搭配此處列出的 AWS IoT Greengrass 版本。

印尼盾第 4.9.3 版，適用於 AWS IoT Greengrass

支援的 AWS IoT Greengrass 版本：

- Greengrass 核, v2.11.0, V2.10.0, 和第 2.9.5 版

IDT 軟件下載：

- IDT v4.9.3 含測試套件 (適用於 Linux)
- IDT v4.9.3 與測試套件 GV2Q_2.5.3 適用於 macOS 果系統
- 帶有測試套件的 IDT V4.9.3，適用於視窗

版本備註：

- 修復了從 Windows 主機測試 Linux 設備時，組件測試中的問題，反之亦然。
- 從`localcomponent`測試組中刪除測`component`試用例。此測試用例不再需要資格。

其他注意事項：

- 如果您的裝置使用 HSM 且您使用的是核心 2.10.x，請移轉至 Greengrass 核心 2.11.0 版或更新版本。

測試套件版本：

`GGV2Q_2.5.3`

- 二零二零二四年四月五日發表

AWS IoT Device Tester 適用於 AWS IoT Greengrass V2 的不支援版本

本主題列出 AWS IoT Greengrass V2 不受支援的 IDT 版本。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 “的 Support AWS IoT Device Tester 政策 AWS IoT Greengrass”。

印尼盾第 4.9.2 版，適用於 AWS IoT Greengrass

版本備註：

- 修正了 Lambda 測試套件因 Java 8 被棄用而失敗的問題。

測試套件版本：

`GGV2Q_2.5.2`

- 二零二四年三月一十八日發表

印尼盾第 4.9.1 版，適用於 AWS IoT Greengrass

版本備註：

- 可讓您驗證並限定執行 AWS IoT Greengrass 核心軟體版本 2.12.0、2.10.0 和 2.9.5 版的裝置。
- 次要錯誤修正。

測試套件版本：

`GGV2Q_2.5.1`

- 二零二零三年十月二十日

IDT 版本 4.7.0，適用於 AWS IoT Greengrass

支援的 AWS IoT Greengrass 版本：

- Greengrass 核 V2.11.0，第 10.0 版和第 2.9.5 版

版本備註：

- 可讓您驗證並限定執行 AWS IoT Greengrass 核心軟體版本 2.11.0、2.10.0 和 2.9.5 的裝置。
- 添加了在參數存儲中存儲 IDT 用戶 AWS Systems Manager 數據值並使用佔位符語法將其獲取到配置中的支持。
- 次要錯誤修正。

測試套件版本：

`GGV2Q_2.5.0`

- 二零二零二年十二月一三

印尼盾第 4.5.11 版，適用於 AWS IoT Greengrass

版本備註：

- 可讓您驗證並限定執行 AWS IoT Greengrass 核心軟體版本 2.9.1、2.9.0、2.8.1、2.8.0、2.7.0 和 2.6.0 版的裝置。
- 添加對在核心設備上測試 PreInstalled Greengrass 的支持。
- 次要錯誤修正。

測試套件版本：

`GGV2Q_2.4.1`

- 二零二零二年十月一三日

IDT 第 4.5.8 版，適用於 AWS IoT Greengrass

版本備註：

- 可讓您驗證並限定執行 AWS IoT Greengrass 核心軟體版本 2.7.0、2.6.0 和 2.5.6 的裝置。
- 可讓您在核心裝置上使用 PreInstalled Greengrass 進行測試。
- 次要錯誤修正。

測試套件版本：

`GGV2Q_2.4.0`

- 二零二零二年八月十二日發表

IDT 版本 4.5.3，適用於 AWS IoT Greengrass

版本備註：

- 可讓您驗證並限定執行 AWS IoT Greengrass 核心軟體版本 2.7.0、2.6.0、2.5.6、2.5.5、2.5.4 和 2.5.3 版的裝置。
- 更新 DockerApplicationManager 測試以使用基於 ECR 的 docker 圖像。
- 次要錯誤修正。

測試套件版本：

`GGV2Q_2.3.1`

- 二零二零零四年四月二十五日

IDT 版本 4.5.1，適用於 AWS IoT Greengrass

版本備註：

- 可讓您驗證和限定執行 AWS IoT Greengrass 核心軟體 v2.5.3 的裝置。
- 添加對使用硬體安全模組 (HSM) 儲存核心軟體使用的私密金鑰和憑證的 Linux 型裝置的驗證和限定支援。 AWS IoT Greengrass
- 實作新的 IDT 測試協調器，以設定自訂測試套件。如需詳細資訊，請參閱 配置 IDT 測試編排器。
- 其他小錯誤修正。

測試套件版本：

`GGV2Q_2.3.0`

- 二零二零零一年一月一日發佈

印尼盾第 4.4.1 版，適用於 AWS IoT Greengrass

版本備註：

- 可讓您驗證和限定執行 AWS IoT Greengrass 核心軟體 v2.5.2 的裝置。
- 添加對使用用戶定義的 IAM 角色作為被測設備假定與 AWS 資源交互的令牌交換角色的支持。

  您可以在`userdata.json`檔案中指定 IAM 角色。如果您指定自訂角色，IDT 會在測試執行期間使用該角色，而不是建立預設的 Token 交換角色。
- 其他小錯誤修正。

測試套件版本：

`GGV2Q_2.2.1`

- 二零二一二年十二月發表

IDT 版本 4.4.0，適用於 AWS IoT Greengrass

版本備註：

- 可讓您驗證和限定執行 AWS IoT Greengrass 核心軟體 v2.5.0 的裝置。
- 添加對在 Windows 上運行 AWS IoT Greengrass 核心軟件的驗證和限定設備的支持。
- 支援使用公開金鑰驗證進行安全殼層 (SSH) 裝置連線。
- 使用安全性最佳實務改善 IDT 許可 IAM 政策。
- 其他小錯誤修正。

測試套件版本：

`GGV2Q_2.1.0`

- 二零二一九年十一月二十

IDT 版本 4.2.0，適用於 AWS IoT Greengrass

版本備註：

- 在執行 AWS IoT Greengrass Core 軟體 v2.2.0 及更新版本的裝置上，支援下列功能的認證：
  - 碼頭 — 驗證裝置是否可以從亞馬遜彈性容器登錄 (Amazon ECR) 下載碼頭容器映像。
  - 機器學習：驗證裝置是否可以使用深度學習執行階段或 Lite ML 架構執行機器學習 (ML) 推論。TensorFlow
  - 串流管理員 — 驗證裝置是否可以下載、安裝和執行串流管理員。 AWS IoT Greengrass
- 可讓您驗證和限定執行 AWS IoT Greengrass 核心軟體 v2.4.0、v2.3.0、v2.2.0 和 2.1.0 版的裝置。
- 將每個測試用例的測試日誌分組在*<device-tester-extract-location>*/results/*<execution-id>*/logs/*<test-group-id>*目錄中單獨的 *< test-case-id >* 文件夾中。
- 其他小錯誤修正。

測試套件版本：

`GGV2Q_2.0.1`

- 二零二一年八月三十一日發佈

IDT 版本 4.1.0，適用於 AWS IoT Greengrass

版本備註：

- 可讓您驗證和限定執行 AWS IoT Greengrass 核心軟體 v2.3.0、v2.2.0、v2.1.0 和 2.0.5 版的裝置。

- 透過移除指定`GreengrassNucleusVersion`和`GreengrassCLIVersion`屬性的需求來改善`userdata.json`組態。
- 包括 AWS IoT Greengrass 核心軟體 v2.1.0 及更新版本的 Lambda 和 MQTT 功能認證的支援。您現在可以使用 IDT for AWS IoT Greengrass V2 來驗證您的核心裝置是否可以執行 Lambda 函數，以及裝置是否可以發佈和訂閱 AWS IoT Core MQTT 主題。
- 改善記錄功能。
- 其他小錯誤修正。

測試套件版本：

`GGV2Q_1.1.1`

- 二零一八年六月二十八日

IDT 版本 4.0.2，適用於 AWS IoT Greengrass

版本備註：

- 可讓您驗證和限定執行 AWS IoT Greengrass 核心軟體 v2.1.0 的裝置。
- 為 AWS IoT Greengrass 核心軟體 v2.1.0 及更新版本新增 Lambda 和 MQTT 功能認證的支援。您現在可以使用 IDT for AWS IoT Greengrass V2 來驗證您的核心裝置是否可以執行 Lambda 函數，以及裝置是否可以發佈和訂閱 AWS IoT Core MQTT 主題。
- 改善記錄功能。
- 其他小錯誤修正。

測試套件版本：

`GGV2Q_1.1.1`

- 5 月 5 日發佈

IDT 版本 4.0.1，適用於 AWS IoT Greengrass

版本備註：

- 可讓您驗證和限定執行第 2 AWS IoT Greengrass 版軟體的裝置。
- 可讓您使用 in 開發和執行自訂測試套 AWS IoT Device Tester 件 AWS IoT Greengrass。如需詳細資訊，請參閱 使用 IDT 開發和運行自己的測試套件。
- 提供適用於 macOS 和視窗的程式碼簽章 IDT 應用程式。在 macOS 上，您可能需要為 IDT 授予安全例外狀況。如需詳細資訊，請參閱 macOS 上的安全性例外。

測試套件版本：

`GGV2Q_1.0.0`

- 二零二零二年十二月發佈

- 測試套件僅運行所需的限定測試，除非您將`features`數組`value`中的對應測試設置為`yes`。

下載適 AWS IoT Greengrass 用於 V2 的 IDT

本主題說明 AWS IoT Device Tester 為 AWS IoT Greengrass V2 下載的選項。您可以使用下列其中一個軟體下載連結，也可以按照指示以程式設計方式下載 IDT。

主題

下載軟體即表示您同意AWS IoT Device Tester 授權合約。

> **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。

手動下載 IDT

本主題列出 AWS IoT Greengrass V2 支援的 IDT 版本。最佳作法是，建議您使用支援目標 AWS IoT Greengrass V2 版本的最新 IDT 版本 AWS IoT Greengrass 。新版本的 AWS IoT Greengrass 可能會要求您下載適用於 AWS IoT Greengrass V2 的新版 IDT。如果 AWS IoT Greengrass V2 的 IDT 與您正在使用的 AWS IoT Greengrass 版本不兼容，則在啟動測試運行時收到通知。

印尼盾第 4.9.3 版，適用於 AWS IoT Greengrass

支援的 AWS IoT Greengrass 版本：

- Greengrass 核, v2.11.0, V2.10.0, 和

IDT 軟件下載：

- IDT v4.9.3 含測試套件 (適用於 Linux)
- 帶有測試套件的 IDT V4.9.3，適用於 macOS 果系統
- 帶有測試套件的 IDT V4.9.3，適用於視窗

版本備註：

- 修復了從 Windows 主機測試 Linux 設備時，組件測試中的問題，反之亦然。
- 從`localcomponent`測試組中刪除測`component`試用例。此測試用例不再需要資格。

其他注意事項：

- 如果您的裝置使用 HSM 且您使用的是核心 2.10.x，請移轉至 Greengrass 核心 2.11.0 版或更新版本。

測試套件版本：

`GGV2Q_2.5.3`

- 二零二零二四年四月五日發表

以編程方式下載 IDT

IDT 提供了一個 API 操作，您可以使用它來檢索 URL，您可以在其中以編程方式下載 IDT。您也可以使用此 API 操作來檢查您是否擁有最新版本的 IDT。此 API 操作具有以下端點。

```
https://download.devicetester.iotdevicesecosystem.amazonaws.com/latestidt
```

若要呼叫此 API 作業，您必須擁有執行**`iot-device-tester:LatestIdt`**動作的權限。包括您的 AWS 簽名並用`iot-device-tester`作服務名稱。

API 要求

HostOs — 主機的作業系統。您可以從以下選項中選擇：

- `mac`
- `linux`
- `windows`

TestSuiteType — 測試套件的類型。選擇下列選項：

`GGV2`— 對 AWS IoT Greengrass 於 V2 的 IDT

ProductVersion

（可選）Greengrass 核的版本。此服務會傳回該版本 Greengrass 核心的最新相容版本的 IDT。如果未指定此選項，服務會傳回最新版本的 IDT。

API 回應

API 回應具有下列格式。包DownloadURL含一個壓縮檔案。

```
{
    "Success": True or False,
    "Message": Message,
    "LatestBk": {
        "Version": The version of the IDT binary,
        "TestSuiteVersion": The version of the test suite,
        "DownloadURL": The URL to download the IDT Bundle, valid for one hour
    }
 }
```

範例

您可以參考下列範例，以程式設計方式下載 IDT。這些範例使用您儲存在AWS_ACCESS_KEY_ID和AWS_SECRET_ACCESS_KEY環境變數中的認證。若要遵循最佳安全性做法，請勿將憑證儲存在程式碼中。

Example 示例：使用 cURL 7.75.0 或更高版本（Mac 和 Linux）下載

如果您有 cURL 版本 7.75.0 或更高版本，則可以使用該aws-sigv4標誌來簽署 API 請求。這個例子使用 jq 來解析響應中的下載 URL。

> **Warning**
>
> 該aws-sigv4標誌要求 curl GET 請求的查詢參數按HostOs/ProductVersion/TestSuiteType或的順序排列HostOs/TestSuiteType。不符合的訂單將導致從 API Gateway 取得標準字串不符合的簽章時發生錯誤。
> 如果包含選ProductVersion用參數，您必須使用支援的產品版本，如 AWS IoT Greengrass V2 的 AWS IoT Device Tester 支援版本中所述。

- 替換 *us-west-2* 與您的. AWS 區域如需區域代碼清單，請參閱區域端點。
- 將 *linux* 取代為主機的作業系統。
- 用您的 AWS IoT Greengrass 核子核版本替換 *2.5.3*。

```
url=$(curl --request GET "https://
download.devicetester.iotdevicesecosystem.amazonaws.com/latestidt?
HostOs=linux&ProductVersion=2.5.3&TestSuiteType=GGV2" \
--user $AWS_ACCESS_KEY_ID:$AWS_SECRET_ACCESS_KEY \
--aws-sigv4 "aws:amz:us-west-2:iot-device-tester" \
| jq -r '.LatestBk["DownloadURL"]')

curl $url --output devicetester.zip
```

Example 範例：使用較早版本的 cURL (Mac 和 Linux) 下載

您可以使用下面的 cURL 命令與您 AWS 簽名和計算的簽名。如需如何簽署和計算簽 AWS 名的詳細資訊，請參閱簽署 AWS API 要求。

- 將 *linux* 取代為主機的作業系統。
- 將時間*#*替換為日期和時間，例如**20220210T004606Z**。
- 將*##*替換為日期，例如**20220210**。
- 替換*AWSRegion*為您的 AWS 區域. 如需區域代碼清單，請參閱區域端點。
- 以您產生的AWS 簽名取*AWSSignature*代。

```
curl --location --request GET 'https://
download.devicetester.iotdevicesecosystem.amazonaws.com/latestidt?
HostOs=linux&TestSuiteType=GGV2' \
--header 'X-Amz-Date: Timestamp \
--header 'Authorization: AWS4-HMAC-SHA256 Credential=$AWS_ACCESS_KEY_ID/Date/AWSRegion/
iot-device-tester/aws4_request, SignedHeaders=host;x-amz-date, Signature=AWSSignature'
```

Example 範例：使用 Python 指令碼下載

這個例子使用了 Python 請求庫。這個例子是從 Python 示例改編為在AWS 一般參考中簽署 AWS API 請求。

- 將 *us-west-2* 取代為您的區域。如需區域代碼清單，請參閱區域端點。
- 將 *linux* 取代為主機的作業系統。

```
# Copyright 2010-2022 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#
# This file is licensed under the Apache License, Version 2.0 (the "License").
```

```
# You may not use this file except in compliance with the License. A copy of the
#License is located at
#
# http://aws.amazon.com/apache2.0/
#
# This file is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS
# OF ANY KIND, either express or implied. See the License for the specific
# language governing permissions and limitations under the License.

# See: http://docs.aws.amazon.com/general/latest/gr/sigv4_signing.html
# This version makes a GET request and passes the signature
# in the Authorization header.
import sys, os, base64, datetime, hashlib, hmac
import requests # pip install requests
# ************* REQUEST VALUES *************
method = 'GET'
service = 'iot-device-tester'
host = 'download.devicetester.iotdevicesecosystem.amazonaws.com'
region = 'us-west-2'
endpoint = 'https://download.devicetester.iotdevicesecosystem.amazonaws.com/latestidt'
request_parameters = 'HostOs=linux&TestSuiteType=GGV2'

# Key derivation functions. See:
# http://docs.aws.amazon.com/general/latest/gr/signature-v4-examples.html#signature-v4-
examples-python
def sign(key, msg):
    return hmac.new(key, msg.encode('utf-8'), hashlib.sha256).digest()

def getSignatureKey(key, dateStamp, regionName, serviceName):
    kDate = sign(('AWS4' + key).encode('utf-8'), dateStamp)
    kRegion = sign(kDate, regionName)
    kService = sign(kRegion, serviceName)
    kSigning = sign(kService, 'aws4_request')
    return kSigning

# Read AWS access key from env. variables or configuration file. Best practice is NOT
# to embed credentials in code.
access_key = os.environ.get('AWS_ACCESS_KEY_ID')
secret_key = os.environ.get('AWS_SECRET_ACCESS_KEY')
if access_key is None or secret_key is None:
    print('No access key is available.')
    sys.exit()

# Create a date for headers and the credential string
```

```
t = datetime.datetime.utcnow()
amzdate = t.strftime('%Y%m%dT%H%M%SZ')
datestamp = t.strftime('%Y%m%d') # Date w/o time, used in credential scope

# ************* TASK 1: CREATE A CANONICAL REQUEST *************
# http://docs.aws.amazon.com/general/latest/gr/sigv4-create-canonical-request.html
# Step 1 is to define the verb (GET, POST, etc.)--already done.
# Step 2: Create canonical URI--the part of the URI from domain to query
# string (use '/' if no path)
canonical_uri = '/latestidt'
# Step 3: Create the canonical query string. In this example (a GET request),
# request parameters are in the query string. Query string values must
# be URL-encoded (space=%20). The parameters must be sorted by name.
# For this example, the query string is pre-formatted in the request_parameters
 variable.
canonical_querystring = request_parameters
# Step 4: Create the canonical headers and signed headers. Header names
# must be trimmed and lowercase, and sorted in code point order from
# low to high. Note that there is a trailing \n.
canonical_headers = 'host:' + host + '\n' + 'x-amz-date:' + amzdate + '\n'
# Step 5: Create the list of signed headers. This lists the headers
# in the canonical_headers list, delimited with ";" and in alpha order.
# Note: The request can include any headers; canonical_headers and
# signed_headers lists those that you want to be included in the
# hash of the request. "Host" and "x-amz-date" are always required.
signed_headers = 'host;x-amz-date'
# Step 6: Create payload hash (hash of the request body content). For GET
# requests, the payload is an empty string ("").
payload_hash = hashlib.sha256(('').encode('utf-8')).hexdigest()
# Step 7: Combine elements to create canonical request
canonical_request = method + '\n' + canonical_uri + '\n' + canonical_querystring + '\n'
 + canonical_headers + '\n' + signed_headers + '\n' + payload_hash

# ************* TASK 2: CREATE THE STRING TO SIGN*************
# Match the algorithm to the hashing algorithm you use, either SHA-1 or
# SHA-256 (recommended)
algorithm = 'AWS4-HMAC-SHA256'
credential_scope = datestamp + '/' + region + '/' + service + '/' + 'aws4_request'
string_to_sign = algorithm + '\n' +  amzdate + '\n' +  credential_scope + '\n' +
 hashlib.sha256(canonical_request.encode('utf-8')).hexdigest()
# ************* TASK 3: CALCULATE THE SIGNATURE *************
# Create the signing key using the function defined above.
signing_key = getSignatureKey(secret_key, datestamp, region, service)
# Sign the string_to_sign using the signing_key
```

```
signature = hmac.new(signing_key, (string_to_sign).encode('utf-8'),
 hashlib.sha256).hexdigest()

# ************* TASK 4: ADD SIGNING INFORMATION TO THE REQUEST *************
# The signing information can be either in a query string value or in
# a header named Authorization. This code shows how to use a header.
# Create authorization header and add to request headers
authorization_header = algorithm + ' ' + 'Credential=' + access_key + '/' +
 credential_scope + ', ' +  'SignedHeaders=' + signed_headers + ', ' + 'Signature=' +
 signature
# The request can include any headers, but MUST include "host", "x-amz-date",
# and (for this scenario) "Authorization". "host" and "x-amz-date" must
# be included in the canonical_headers and signed_headers, as noted
# earlier. Order here is not significant.
# Python note: The 'host' header is added automatically by the Python 'requests'
 library.
headers = {'x-amz-date':amzdate, 'Authorization':authorization_header}

# ************* SEND THE REQUEST *************
request_url = endpoint + '?' + canonical_querystring
print('\nBEGIN REQUEST++++++++++++++++++++++++++++++++++++')
print('Request URL = ' + request_url)
response = requests.get(request_url, headers=headers)
print('\nRESPONSE++++++++++++++++++++++++++++++++++++')
print('Response code: %d\n' % response.status_code)
print(response.text)

download_url = response.json()["LatestBk"]["DownloadURL"]
r = requests.get(download_url)
open('devicetester.zip', 'wb').write(r.content)
```

使用 IDT 執行 AWS IoT Greengrass 資格套件

您可以使 AWS IoT Device Tester 用 AWS IoT Greengrass V2 來驗證 AWS IoT Greengrass Core 軟體是否在您的硬體上執行，並且可以與 AWS 雲端. 它還執行 end-to-end 測試與 AWS IoT Core. 例如，它會驗證您的裝置是否可以部署元件並對其進行升級。

除了測試裝置之外，IDT for AWS IoT Greengrass V2 還會在您的中建立資源 (例如 AWS IoT 物件、群組等)， AWS 帳戶 以促進資格程序。

若要建立這些資源，IDT for AWS IoT Greengrass V2 會使用`config.json`檔案中設定的 AWS 認證代表您進行 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 V2 執行 AWS IoT Greengrass 資格套件時，它會執行下列步驟：

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. 產生測試報告以指出主機板是否通過符合資格所需的測試。

測試套件版本

IDT for AWS IoT Greengrass V2 將測試組織到測試套件和測試組中。

- 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。
- 測試群組是與特定功能 (例如元件部署) 相關的一組個別測試。

例如`GGV2Q_1.0.0`，測試套件使用*`major`*`.`*`minor`*`.`*`patch`*格式進行版本化。當您下載 IDT 時，該軟件包包括最新的 Greengrass 資格套件版本。

> **Important**
>
> 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。如需詳細資訊，請參閱 the section called “的 Support AWS IoT Device Tester 政策 AWS IoT Greengrass”。
>
> 您可以運行`list-supported-products`以列出當前版本的 IDT 支持的版本 AWS IoT Greengrass 和測試套件。

測試群組描述

核心資格的必要測試群組

需要這些測試群組才能使您的 AWS IoT Greengrass V2 裝置符合裝 AWS Partner 置目錄的資格。

核心相依性

驗證裝置是否符合 AWS IoT Greengrass Core 軟體的所有軟體和硬體需求。該測試組包括以下測試用例：

Java 版本

檢查所需的 Java 版本是否已安裝在被測設備上。 AWS IoT Greengrass 需要 Java 8 或更新版本。

PreTest 驗證

檢查裝置是否符合執行測試的軟體需求。

- 對於基於 Linux 的設備，此測試會檢查設備是否可以運行以下 Linux 命令：

  `chmod`, `cp`, `echo`, `grep`, `kill`, `ln`, `mkinfo`, `ps`, `rm`, `sh`, `uname`

- 對於以 Windows 為基礎的裝置，此測試會檢查裝置是否已安裝下列 Microsoft 軟體：

  升級版 5.1 或更高版本，.NET v4.6.1 或更高版本，視覺 C ++ 2017 或更高版本，實用程序 PsExec

版本檢查器

檢查 AWS IoT Greengrass 提供的版本是否與您正在使用的 AWS IoT 設備測試器版本兼容。

元件

驗證裝置是否可以部署元件並對其進行升級。該測試組包括以下測試：

雲端元件

驗證雲端元件的裝置功能。

本機元件

驗證本機元件的裝置功能。

Lambda

此測試不適用於以 Windows 為基礎的裝置。

驗證裝置是否可以部署使用 Java 執行階段的 Lambda 函數元件，以及 Lambda 函數可以使用 AWS IoT Core MQTT 主題做為工作訊息的事件來源。

MQTT

驗證裝置是否可以訂閱並發佈至 AWS IoT Core MQTT 主題。

選用測試群組

> **Note**
>
> 這些測試群組是選擇性的，且僅適用於符合 Linux 資格的 Greengrass 核心裝置。如果您選擇符合選擇性測試的資格，您的裝置會在裝 AWS Partner 置目錄中列出其他功能。

碼頭依賴關係

驗證裝置是否符合使用 AWS提供的 Docker 應用程式管理員 () `aws.greengrass.DockerApplicationManager` 元件所需的所有技術相依性。

碼頭應用程序管理器資格

驗證裝置是否可以從 Amazon ECR 下載碼頭容器映像檔。

Machine Learning 相依性

驗證裝置是否符合使用 AWS提供的機器學習 (ML) 元件所需的所有技術相依性。

Machine Learning 推論測試

驗證裝置是否可以使用深度學習執行階段和 TensorFlow Lite ML 架構執行 ML 推論。

流管理器依賴項

驗證裝置是否可以下載、安裝和執行AWS IoT Greengrass 串流管理員。

硬體安全整合 (HSI)

> **Note**
>
> 此測試僅適用於以 Linux 為基礎的裝置，IDT v4.5.1 及更新版本。 AWS IoT Greengrass 目前不支援 Windows 裝置的硬體安全性整合。

驗證裝置是否可以使用儲存在硬體安全性模組 (HSM) 中的私密金鑰和憑證來驗證與和 AWS IoT Greengrass 服務的連線。 AWS IoT 此測試也會驗證提供的 PKCS #11 AWS提供者元件是否可以使用廠商提供的 PKCS #11 程式庫與 HSM 連接。如需更多詳細資訊，請參閱 硬體安全整合。

執行資AWS IoT Greengrass格套件的先決條件

本節說明使用 AWS IoT Device Tester (IDT) 的先決條件。AWS IoT Greengrass

下載最新版本AWS IoT Device Tester的 AWS IoT Greengrass

下載最新版本的 IDT，並將軟體解壓縮至檔案系統上具有讀取/寫入權限的位置 (*< device-tester-extract-location >*)。

> **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。

下載AWS IoT Greengrass軟體

適用於 AWS IoT Greengrass V2 的 IDT 會測試您的裝置是否與特定版本的AWS IoT Greengrass. 執行下列命令，將 AWS IoT Greengrass Core 軟體下載至名為的檔案`aws.greengrass.nucleus.zip`。將*##*取代為您 IDT 版本支援的核心元件版本。

Linux or Unix

```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip >
 aws.greengrass.nucleus.zip
```

Windows Command Prompt (CMD)

```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip >
 aws.greengrass.nucleus.zip
```

PowerShell

```
iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip -
OutFile aws.greengrass.nucleus.zip
```

將下載的`aws.greengrass.nucleus.zip`檔案放在`<device-tester-extract-location>/products/`資料夾中。

> **Note**
>
> 請勿將相同作業系統和架構的多個檔案放在這個目錄中。

建立和設定 AWS 帳戶

您必須先執行下列步驟，才能使用 AWS IoT Device Tester AWS IoT Greengrass V2：

1. 設定AWS 帳戶. 如果您已有AWS 帳戶，請跳至步驟 2。
2. 設定 IDT 的權限。

這些帳戶權限允許 IDT 代表您存取AWS服務並建立AWS資源，例如AWS IoT物AWS IoT Greengrass件和元件。

若要建立這些資源，IDT for AWS IoT Greengrass V2 會使用`config.json`檔案中設定的AWS認證代表您進行 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AWS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：設定 AWS 帳戶

在此步驟中，建立並設定AWS 帳戶. 如果您已擁有 AWS 帳戶，請跳到 the section called “步驟 2：設定 IDT 的許可”。

如果您還沒有 AWS 帳戶，請完成以下步驟建立新帳戶。

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，並在電話鍵盤輸入驗證碼。

註冊 AWS 帳戶時，會建立 AWS 帳戶根使用者。根使用者有權存取該帳戶中的所有 AWS 服務和資源。作為最佳安全實務，將管理存取權指派給管理使用者，並且僅使用根使用者來執行需要根使用者存取權的任務。

若要建立管理員使用者，請選擇下列其中一個選項。

選擇一種管理管理員的方式	到	By	您也可以
在 IAM Identity Center (建議)	使用短期憑證存取 AWS。 這與安全性最佳實務一致。有關最佳實務的資訊，請參閱 IAM 使用者指南中的 IAM 安全最佳實務。	請遵循 AWS IAM Identity Center 使用者指南的入門中的說明。	請參閱 AWS Command Line Interface 使用者指南中的設定 AWS CLI 以使用 AWS IAM Identity Center 設定程式設計存取。
在 IAM 中 (不建議使用)	使用長期憑證存取 AWS。	請遵循 IAM 使用者指南中建立您的第一個 IAM 管理員使用者和使用者群組的說明。	請參閱 IAM 使用者指南中的管理 IAM 使用者的存取金鑰，設定程式設計存取。

步驟 2：設定 IDT 的許可

在此步驟中，設定 IDT for AWS IoT Greengrass V2 用來執行測試和收集 IDT 使用量資料的權限。您可以使用AWS Management Console或 AWS Command Line Interface(AWS CLI) 建立 IDT 的 IAM 政策和測試使用者，然後將政策附加到使用者。如果您已經建立 IDT 的測試使用者，請跳至配置您的設備以運行 IDT 測試。

設定 IDT (主控台) 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 Policies (政策)，然後選擇 Create policy (建立政策)。

b. 如果您未使用 PreInstalled，請在 JSON 索引標籤上，以下列原則取代預留位置內容。如果您正在使用 PreInstalled，請繼續執行以下步驟。

```
{
    "Version":"2012-10-17",
    "Statement":[
    {
      "Sid":"passRoleForResources",
      "Effect":"Allow",
      "Action":"iam:PassRole",
      "Resource":"arn:aws:iam::*:role/idt-*",
      "Condition":{
        "StringEquals":{
          "iam:PassedToService":[
            "iot.amazonaws.com",
            "lambda.amazonaws.com",
            "greengrass.amazonaws.com"
          ]
        }
      }
    },
    {
      "Sid":"lambdaResources",
      "Effect":"Allow",
      "Action":[
        "lambda:CreateFunction",
        "lambda:PublishVersion",
        "lambda:DeleteFunction",
        "lambda:GetFunction"
      ],
      "Resource":[
        "arn:aws:lambda:*:*:function:idt-*"
      ]
    },
    {
      "Sid":"iotResources",
      "Effect":"Allow",
      "Action":[
        "iot:CreateThing",
        "iot:DeleteThing",
        "iot:DescribeThing",
```

```
        "iot:CreateThingGroup",
        "iot:DeleteThingGroup",
        "iot:DescribeThingGroup",
        "iot:AddThingToThingGroup",
        "iot:RemoveThingFromThingGroup",
        "iot:AttachThingPrincipal",
        "iot:DetachThingPrincipal",
        "iot:UpdateCertificate",
        "iot:DeleteCertificate",
        "iot:CreatePolicy",
        "iot:AttachPolicy",
        "iot:DetachPolicy",
        "iot:DeletePolicy",
        "iot:GetPolicy",
        "iot:Publish",
        "iot:TagResource",
        "iot:ListThingPrincipals",
        "iot:ListAttachedPolicies",
        "iot:ListTargetsForPolicy",
        "iot:ListThingGroupsForThing",
        "iot:ListThingsInThingGroup",
        "iot:CreateJob",
        "iot:DescribeJob",
        "iot:DescribeJobExecution",
        "iot:CancelJob"
      ],
      "Resource":[
        "arn:aws:iot:*:*:thing/idt-*",
        "arn:aws:iot:*:*:thinggroup/idt-*",
        "arn:aws:iot:*:*:policy/idt-*",
        "arn:aws:iot:*:*:cert/*",
        "arn:aws:iot:*:*:topic/idt-*",
        "arn:aws:iot:*:*:job/*"
      ]
    },
    {
      "Sid":"s3Resources",
      "Effect":"Allow",
      "Action":[
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObjectVersion",
        "s3:DeleteObject",
        "s3:CreateBucket",
```

```
        "s3:ListBucket",
        "s3:ListBucketVersions",
        "s3:DeleteBucket",
        "s3:PutObjectTagging",
        "s3:PutBucketTagging"
      ],
      "Resource":"arn:aws:s3::*:idt-*"
    },
    {
      "Sid":"roleAliasResources",
      "Effect":"Allow",
      "Action":[
        "iot:CreateRoleAlias",
        "iot:DescribeRoleAlias",
        "iot:DeleteRoleAlias",
        "iot:TagResource",
        "iam:GetRole"
      ],
      "Resource":[
        "arn:aws:iot:*:*:rolealias/idt-*",
        "arn:aws:iam::*:role/idt-*"
      ]
    },
    {
      "Sid":"idtExecuteAndCollectMetrics",
      "Effect":"Allow",
      "Action":[
        "iot-device-tester:SendMetrics",
        "iot-device-tester:SupportedVersion",
        "iot-device-tester:LatestIdt",
        "iot-device-tester:CheckVersion",
        "iot-device-tester:DownloadTestSuite"
      ],
      "Resource":"*"
    },
    {
      "Sid":"genericResources",
      "Effect":"Allow",
      "Action":[
        "greengrass:*",
        "iot:GetThingShadow",
        "iot:UpdateThingShadow",
        "iot:ListThings",
        "iot:DescribeEndpoint",
```

```
        "iot:CreateKeysAndCertificate"
      ],
      "Resource":"*"
    },
    {
      "Sid":"iamResourcesUpdate",
      "Effect":"Allow",
      "Action":[
        "iam:CreateRole",
        "iam:DeleteRole",
        "iam:CreatePolicy",
        "iam:DeletePolicy",
        "iam:AttachRolePolicy",
        "iam:DetachRolePolicy",
        "iam:TagRole",
        "iam:TagPolicy",
        "iam:GetPolicy",
        "iam:ListAttachedRolePolicies",
        "iam:ListEntitiesForPolicy"
      ],
      "Resource":[
        "arn:aws:iam::*:role/idt-*",
        "arn:aws:iam::*:policy/idt-*"
      ]
    }
  ]
}
```

c. 如果您使用 PreInstalled，請在 JSON 索引標籤上，以下列原則取代預留位置內容。請確定您：

- 將`iotResources`陳述式中的 *`Thing Name # ThingGroup`* 取代為 Greengrass 安裝期間所建立的 thingName 與物件群組，以新增權限。
- 使用在 DUT 上安裝 Greengrass 期間建立的`roleAliasResources`角色，取代`passRoleForResources`陳述式和陳述式中的 *`PassRole ### Alias`*。

```
{
    "Version":"2012-10-17",
    "Statement":[
    {
      "Sid":"passRoleForResources",
      "Effect":"Allow",
```

```
      "Action":"iam:PassRole",
      "Resource":"arn:aws:iam::*:role/passRole",
      "Condition":{
        "StringEquals":{
          "iam:PassedToService":[
            "iot.amazonaws.com",
            "lambda.amazonaws.com",
            "greengrass.amazonaws.com"
          ]
        }
      }
    },
    {
      "Sid":"lambdaResources",
      "Effect":"Allow",
      "Action":[
        "lambda:CreateFunction",
        "lambda:PublishVersion",
        "lambda:DeleteFunction",
        "lambda:GetFunction"
      ],
      "Resource":[
        "arn:aws:lambda:*:*:function:idt-*"
      ]
    },
    {
      "Sid":"iotResources",
      "Effect":"Allow",
      "Action":[
        "iot:CreateThing",
        "iot:DeleteThing",
        "iot:DescribeThing",
        "iot:CreateThingGroup",
        "iot:DeleteThingGroup",
        "iot:DescribeThingGroup",
        "iot:AddThingToThingGroup",
        "iot:RemoveThingFromThingGroup",
        "iot:AttachThingPrincipal",
        "iot:DetachThingPrincipal",
        "iot:UpdateCertificate",
        "iot:DeleteCertificate",
        "iot:CreatePolicy",
        "iot:AttachPolicy",
        "iot:DetachPolicy",
```

```
        "iot:DeletePolicy",
        "iot:GetPolicy",
        "iot:Publish",
        "iot:TagResource",
        "iot:ListThingPrincipals",
        "iot:ListAttachedPolicies",
        "iot:ListTargetsForPolicy",
        "iot:ListThingGroupsForThing",
        "iot:ListThingsInThingGroup",
        "iot:CreateJob",
        "iot:DescribeJob",
        "iot:DescribeJobExecution",
        "iot:CancelJob"
      ],
      "Resource":[
        "arn:aws:iot:*:*:thing/thingName",
        "arn:aws:iot:*:*:thinggroup/thingGroup",
        "arn:aws:iot:*:*:policy/idt-*",
        "arn:aws:iot:*:*:cert/*",
        "arn:aws:iot:*:*:topic/idt-*",
        "arn:aws:iot:*:*:job/*"
      ]
    },
    {
      "Sid":"s3Resources",
      "Effect":"Allow",
      "Action":[
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObjectVersion",
        "s3:DeleteObject",
        "s3:CreateBucket",
        "s3:ListBucket",
        "s3:ListBucketVersions",
        "s3:DeleteBucket",
        "s3:PutObjectTagging",
        "s3:PutBucketTagging"
      ],
      "Resource":"arn:aws:s3::*:idt-*"
    },
    {
      "Sid":"roleAliasResources",
      "Effect":"Allow",
      "Action":[
```

```
        "iot:CreateRoleAlias",
        "iot:DescribeRoleAlias",
        "iot:DeleteRoleAlias",
        "iot:TagResource",
        "iam:GetRole"
      ],
      "Resource":[
        "arn:aws:iot:*:*:rolealias/roleAlias",
        "arn:aws:iam::*:role/idt-*"
      ]
    },
    {
      "Sid":"idtExecuteAndCollectMetrics",
      "Effect":"Allow",
      "Action":[
        "iot-device-tester:SendMetrics",
        "iot-device-tester:SupportedVersion",
        "iot-device-tester:LatestIdt",
        "iot-device-tester:CheckVersion",
        "iot-device-tester:DownloadTestSuite"
      ],
      "Resource":"*"
    },
    {
      "Sid":"genericResources",
      "Effect":"Allow",
      "Action":[
        "greengrass:*",
        "iot:GetThingShadow",
        "iot:UpdateThingShadow",
        "iot:ListThings",
        "iot:DescribeEndpoint",
        "iot:CreateKeysAndCertificate"
      ],
      "Resource":"*"
    },
    {
      "Sid":"iamResourcesUpdate",
      "Effect":"Allow",
      "Action":[
        "iam:CreateRole",
        "iam:DeleteRole",
        "iam:CreatePolicy",
        "iam:DeletePolicy",
```

```
        "iam:AttachRolePolicy",
        "iam:DetachRolePolicy",
        "iam:TagRole",
        "iam:TagPolicy",
        "iam:GetPolicy",
        "iam:ListAttachedRolePolicies",
        "iam:ListEntitiesForPolicy"
      ],
      "Resource":[
        "arn:aws:iam::*:role/idt-*",
        "arn:aws:iam::*:policy/idt-*"
      ]
    }
  ]
}
```

> **Note**
>
> 如果您想要使用自訂 IAM 角色做為待測裝置的權杖交換角色，請務必更新政策中的`roleAliasResources`陳述式和`passRoleForResources`陳述式，以允許自訂 IAM 角色資源。

d. 選擇檢閱政策。

e. 針對名稱，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 使用者並附加 IDT 所需的許可。AWS IoT Greengrass

   a. 建立 IAM 使用者。遵循 IAM 使用者指南中建立 IAM 使用者 (主控台) 中的步驟 1 到 5。

   b. 將許可附加到您的 IAM 使用者：

      i. 在 Set permissions (設定許可) 頁面上，選擇 Attach existing policies directly (直接連接現有的政策)。

      ii. 搜尋您在上一個步驟中建立的 IDTGreengrassIAMPermissions 政策。選取核取方塊。

   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。稍後您可以使用此資訊來設定AWS認證檔案。

4. 下一步：設定您的實體裝置。

步驟 2：設定 IDT (AWS CLI) 的許可

1. 如果尚未安裝 AWS CLI，請在電腦上安裝並設定。請按照《AWS Command Line Interface使用者指南》中的〈安裝〉AWS CLI中的步驟進行。

> **Note**
>
> 這AWS CLI是一個開放原始碼工具，您可以用來從命令列殼層與AWS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。

a. 如果您未使用 PreInstalled，請開啟文字編輯器，並將下列原則內容儲存在 JSON 檔案中。如果您正在使用 PreInstalled，請繼續執行以下步驟。

```
{
    "Version":"2012-10-17",
    "Statement":[
    {
      "Sid":"passRoleForResources",
      "Effect":"Allow",
      "Action":"iam:PassRole",
      "Resource":"arn:aws:iam::*:role/idt-*",
      "Condition":{
        "StringEquals":{
          "iam:PassedToService":[
            "iot.amazonaws.com",
            "lambda.amazonaws.com",
            "greengrass.amazonaws.com"
          ]
        }
      }
    },
    {
      "Sid":"lambdaResources",
      "Effect":"Allow",
      "Action":[
```

```
        "lambda:CreateFunction",
        "lambda:PublishVersion",
        "lambda:DeleteFunction",
        "lambda:GetFunction"
      ],
      "Resource":[
        "arn:aws:lambda:*:*:function:idt-*"
      ]
    },
    {
      "Sid":"iotResources",
      "Effect":"Allow",
      "Action":[
        "iot:CreateThing",
        "iot:DeleteThing",
        "iot:DescribeThing",
        "iot:CreateThingGroup",
        "iot:DeleteThingGroup",
        "iot:DescribeThingGroup",
        "iot:AddThingToThingGroup",
        "iot:RemoveThingFromThingGroup",
        "iot:AttachThingPrincipal",
        "iot:DetachThingPrincipal",
        "iot:UpdateCertificate",
        "iot:DeleteCertificate",
        "iot:CreatePolicy",
        "iot:AttachPolicy",
        "iot:DetachPolicy",
        "iot:DeletePolicy",
        "iot:GetPolicy",
        "iot:Publish",
        "iot:TagResource",
        "iot:ListThingPrincipals",
        "iot:ListAttachedPolicies",
        "iot:ListTargetsForPolicy",
        "iot:ListThingGroupsForThing",
        "iot:ListThingsInThingGroup",
        "iot:CreateJob",
        "iot:DescribeJob",
        "iot:DescribeJobExecution",
        "iot:CancelJob"
      ],
      "Resource":[
        "arn:aws:iot:*:*:thing/idt-*",
```

```
        "arn:aws:iot:*:*:thinggroup/idt-*",
        "arn:aws:iot:*:*:policy/idt-*",
        "arn:aws:iot:*:*:cert/*",
        "arn:aws:iot:*:*:topic/idt-*",
        "arn:aws:iot:*:*:job/*"
      ]
    },
    {
      "Sid":"s3Resources",
      "Effect":"Allow",
      "Action":[
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObjectVersion",
        "s3:DeleteObject",
        "s3:CreateBucket",
        "s3:ListBucket",
        "s3:ListBucketVersions",
        "s3:DeleteBucket",
        "s3:PutObjectTagging",
        "s3:PutBucketTagging"
      ],
      "Resource":"arn:aws:s3::*:idt-*"
    },
    {
      "Sid":"roleAliasResources",
      "Effect":"Allow",
      "Action":[
        "iot:CreateRoleAlias",
        "iot:DescribeRoleAlias",
        "iot:DeleteRoleAlias",
        "iot:TagResource",
        "iam:GetRole"
      ],
      "Resource":[
        "arn:aws:iot:*:*:rolealias/idt-*",
        "arn:aws:iam::*:role/idt-*"
      ]
    },
    {
      "Sid":"idtExecuteAndCollectMetrics",
      "Effect":"Allow",
      "Action":[
        "iot-device-tester:SendMetrics",
```

```
        "iot-device-tester:SupportedVersion",
        "iot-device-tester:LatestIdt",
        "iot-device-tester:CheckVersion",
        "iot-device-tester:DownloadTestSuite"
      ],
      "Resource":"*"
    },
    {
      "Sid":"genericResources",
      "Effect":"Allow",
      "Action":[
        "greengrass:*",
        "iot:GetThingShadow",
        "iot:UpdateThingShadow",
        "iot:ListThings",
        "iot:DescribeEndpoint",
        "iot:CreateKeysAndCertificate"
      ],
      "Resource":"*"
    },
    {
      "Sid":"iamResourcesUpdate",
      "Effect":"Allow",
      "Action":[
        "iam:CreateRole",
        "iam:DeleteRole",
        "iam:CreatePolicy",
        "iam:DeletePolicy",
        "iam:AttachRolePolicy",
        "iam:DetachRolePolicy",
        "iam:TagRole",
        "iam:TagPolicy",
        "iam:GetPolicy",
        "iam:ListAttachedRolePolicies",
        "iam:ListEntitiesForPolicy"
      ],
      "Resource":[
        "arn:aws:iam::*:role/idt-*",
        "arn:aws:iam::*:policy/idt-*"
      ]
    }
  ]
}
```

b. 如果您正在使用 PreInstalled，請開啟文字編輯器，並將下列原則內容儲存在 JSON 檔案中。請確定您：

- 取代在受測裝置 (DUT) 上安裝 Greengrass 期間建立的`iotResources`陳述式中的「*thingName#######*」，以新增權限。
- 使用在 DUT 上安裝 Greengrass 期間建立的`roleAliasResources`角色，取代`passRoleForResources`陳述式和陳述式中的 *PassRole ### Alias*。

```
{
    "Version":"2012-10-17",
    "Statement":[
    {
      "Sid":"passRoleForResources",
      "Effect":"Allow",
      "Action":"iam:PassRole",
      "Resource":"arn:aws:iam::*:role/passRole",
      "Condition":{
        "StringEquals":{
          "iam:PassedToService":[
            "iot.amazonaws.com",
            "lambda.amazonaws.com",
            "greengrass.amazonaws.com"
          ]
        }
      }
    },
    {
      "Sid":"lambdaResources",
      "Effect":"Allow",
      "Action":[
        "lambda:CreateFunction",
        "lambda:PublishVersion",
        "lambda:DeleteFunction",
        "lambda:GetFunction"
      ],
      "Resource":[
        "arn:aws:lambda:*:*:function:idt-*"
      ]
    },
    {
      "Sid":"iotResources",
```

```
      "Effect":"Allow",
      "Action":[
        "iot:CreateThing",
        "iot:DeleteThing",
        "iot:DescribeThing",
        "iot:CreateThingGroup",
        "iot:DeleteThingGroup",
        "iot:DescribeThingGroup",
        "iot:AddThingToThingGroup",
        "iot:RemoveThingFromThingGroup",
        "iot:AttachThingPrincipal",
        "iot:DetachThingPrincipal",
        "iot:UpdateCertificate",
        "iot:DeleteCertificate",
        "iot:CreatePolicy",
        "iot:AttachPolicy",
        "iot:DetachPolicy",
        "iot:DeletePolicy",
        "iot:GetPolicy",
        "iot:Publish",
        "iot:TagResource",
        "iot:ListThingPrincipals",
        "iot:ListAttachedPolicies",
        "iot:ListTargetsForPolicy",
        "iot:ListThingGroupsForThing",
        "iot:ListThingsInThingGroup",
        "iot:CreateJob",
        "iot:DescribeJob",
        "iot:DescribeJobExecution",
        "iot:CancelJob"
      ],
      "Resource":[
        "arn:aws:iot:*:*:thing/thingName",
        "arn:aws:iot:*:*:thinggroup/thingGroup",
        "arn:aws:iot:*:*:policy/idt-*",
        "arn:aws:iot:*:*:cert/*",
        "arn:aws:iot:*:*:topic/idt-*",
        "arn:aws:iot:*:*:job/*"
      ]
    },
    {
      "Sid":"s3Resources",
      "Effect":"Allow",
      "Action":[
```

```
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObjectVersion",
        "s3:DeleteObject",
        "s3:CreateBucket",
        "s3:ListBucket",
        "s3:ListBucketVersions",
        "s3:DeleteBucket",
        "s3:PutObjectTagging",
        "s3:PutBucketTagging"
      ],
      "Resource":"arn:aws:s3::*:idt-*"
    },
    {
      "Sid":"roleAliasResources",
      "Effect":"Allow",
      "Action":[
        "iot:CreateRoleAlias",
        "iot:DescribeRoleAlias",
        "iot:DeleteRoleAlias",
        "iot:TagResource",
        "iam:GetRole"
      ],
      "Resource":[
        "arn:aws:iot:*:*:rolealias/roleAlias",
        "arn:aws:iam::*:role/idt-*"
      ]
    },
    {
      "Sid":"idtExecuteAndCollectMetrics",
      "Effect":"Allow",
      "Action":[
        "iot-device-tester:SendMetrics",
        "iot-device-tester:SupportedVersion",
        "iot-device-tester:LatestIdt",
        "iot-device-tester:CheckVersion",
        "iot-device-tester:DownloadTestSuite"
      ],
      "Resource":"*"
    },
    {
      "Sid":"genericResources",
      "Effect":"Allow",
      "Action":[
```

```
        "greengrass:*",
        "iot:GetThingShadow",
        "iot:UpdateThingShadow",
        "iot:ListThings",
        "iot:DescribeEndpoint",
        "iot:CreateKeysAndCertificate"
      ],
      "Resource":"*"
    },
    {
      "Sid":"iamResourcesUpdate",
      "Effect":"Allow",
      "Action":[
        "iam:CreateRole",
        "iam:DeleteRole",
        "iam:CreatePolicy",
        "iam:DeletePolicy",
        "iam:AttachRolePolicy",
        "iam:DetachRolePolicy",
        "iam:TagRole",
        "iam:TagPolicy",
        "iam:GetPolicy",
        "iam:ListAttachedRolePolicies",
        "iam:ListEntitiesForPolicy"
      ],
      "Resource":[
        "arn:aws:iam::*:role/idt-*",
        "arn:aws:iam::*:policy/idt-*"
      ]
    }
  ]
}
```

> **Note**
>
> 如果您想要使用自訂 IAM 角色做為待測裝置的權杖交換角色，請務必更新政策中的`roleAliasResources`陳述式和`passRoleForResources`陳述式，以允許自訂 IAM 角色資源。

c. 執行下列命令以建立名為的客戶管理策略`IDTGreengrassIAMPermissions`。取代*`policy.json`*為您在上一個步驟中建立的 JSON 檔案的完整路徑。

```
aws iam create-policy --policy-name IDTGreengrassIAMPermissions --policy-
document file://policy.json
```

3. 建立 IAM 使用者並附加 IDT 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 中建立的IDTGreengrassIAMPermissions政策附加到 IAM 使用者。<account-id>在命令中以您的AWS 帳戶.

   ```
   aws iam attach-user-policy --user-name IDTGreengrassUser --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 IDTGreengrassUser
   ```

   將輸出儲存在安全的位置。稍後您可以使用此資訊來設定AWS認證檔案。

5. 下一步：設定您的實體裝置。

AWS IoT Device Tester 許可

下列原則說明AWS IoT Device Tester權限。

AWS IoT Device Tester版本檢查和自動更新功能需要這些權限。

- iot-device-tester:SupportedVersion

  授AWS IoT Device Tester予獲取支持產品，測試套件和 IDT 版本列表的權限。

- iot-device-tester:LatestIdt

  授AWS IoT Device Tester予獲取可供下載的最新 IDT 版本的權限。

- iot-device-tester:CheckVersion

  授AWS IoT Device Tester予檢查 IDT、測試套件和產品版本相容性的權限。

- iot-device-tester:DownloadTestSuite

授AWS IoT Device Tester予下載測試套件更新的權限。

AWS IoT Device Tester也會針對選擇性量度報告使用下列權限：

- `iot-device-tester:SendMetrics`

  授予收集有關AWS IoT Device Tester內部使用量度之指標的權限。AWS如果省略此權限，將不會收集這些測量結果。

配置您的設備以運行 IDT 測試

若要讓 IDT 執行裝置資格測試，您必須將主機電腦設定為存取裝置，並在裝置上設定使用者權限。

在主機電腦上安裝 Java

從 IDT v4.2.0 開始，AWS IoT Greengrass需要 Java 才能執行的選擇性資格測試。

您可以使用 Java 版本 8 或更高版本。我們建議您使用 Amazon Corretto 或 OpenJDK 長期支持版本。需要版本 8 或更高版本。

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>***

   用*`remote-ssh-user`*於登錄被測設備的用戶名在哪裡，並且*`remote-device-ip`*是要對其進行測試的設備的 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 伺服器和用戶端。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`。

設定 Windows 裝置的使用者身份證明

若要符合 Windows 裝置的資格，您必須在受測裝置的LocalSystem 帳戶中為下列使用者設定使用者認證：

- 默認使用者 ()。`ggc_user`
- 您用來連接到被測設備的用戶。您可以在`device.json`檔案中設定此使用者。

您必須在被測裝置上的 LocalSystem 帳戶中建立每個使用者，然後將該使用者的使用者名稱和密碼儲存在該LocalSystem 帳戶的憑證管理員執行個體中。

在 Windows 裝置上設定使用者

1. 以系統管理員身分開啟 Windows 命令提示字元 (cmd.exe)。
2. 在 Windows 裝置上的 LocalSystem 帳戶中建立使用者。針對您要建立的每個使用者執行下列命令。*##### Greengrass ###############* ggc_user以安全*##*取代密碼。

   ```
   net user /add user-name password
   ```

3. 在設備上從 Microsoft 下載並安裝該PsExec實用程序。
4. 使用此 PsExec 公用程式將預設使用者的使用者名稱和密碼儲存在 LocalSystem 帳戶的認證管理員執行個體中。

   針對您要在認證管理員中設定的每個使用者執行下列命令。*##### Greengrass ########## ####* ggc_user*################*

   ```
   psexec -s cmd /c cmdkey /generic:user-name /user:user-name /pass:password
   ```

   如果打PsExec License Agreement開，請選Accept擇同意許可證並運行命令。

   > **Note**
   >
   > 在 Windows 裝置上， LocalSystem 帳戶會執行 Greengrass 核心，而且您必須使用公用 PsExec 程式將使用者資訊儲存在帳戶中。 LocalSystem使用認證管理員應用程式會將此資訊儲存在目前登入使用者的 Windows 帳戶中，而非 LocalSystem帳戶中。

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)。若要自動執行這些作業，AWS IoT GreengrassV2 的 IDT 必須能夠在不提示輸入密碼的情況下使用 sudo 執行命令。

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
   ```

設定自訂權杖交換角色

您可以選擇使用自定義 IAM 角色作為被測設備假定與AWS資源進行交互的令牌交換角色。如需建立 IAM 角色的相關資訊，請參閱 IAM 使用者指南中的建立 IAM 角色。

您必須符合以下要求，才能允許 IDT 使用您的自訂 IAM 角色。強烈建議您只將必要的最低原則動作新增至此角色。

- 必須更新使用者資料 .json 組態檔案，才能將參數設定`GreengrassV2TokenExchangeRole`為。`true`
- 自訂 IAM 角色必須設定下列最低信任政策：

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
            "Service":[
               "credentials.iot.amazonaws.com",
               "lambda.amazonaws.com",
               "sagemaker.amazonaws.com"
            ]
         },
         "Action":"sts:AssumeRole"
      }
   ]
}
```

- 自訂 IAM 角色必須設定下列最低許可政策：

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "iot:DescribeCertificate",
            "logs:CreateLogGroup",
            "logs:CreateLogStream",
            "logs:PutLogEvents",
            "logs:DescribeLogStreams",
            "iot:Connect",
            "iot:Publish",
            "iot:Subscribe",
            "iot:Receive",
            "iot:ListThingPrincipals",
            "iot:GetThingShadow",
            "iot:UpdateThingShadow",
            "s3:GetBucketLocation",
            "s3:GetObject",
            "s3:PutObject",
            "s3:AbortMultipartUpload",
            "s3:ListMultipartUploadParts"
         ],
         "Resource":"*"
      }
   ]
}
```

- 自訂 IAM 角色的名稱必須與您在測試使用者的 IAM 許可中指定的 IAM 角色資源相符。根據預設，測試使用者政策允許存取角色名稱中具有`idt-`前置詞的 IAM 角色。如果您的 IAM 角色名稱不使用此前綴，請將`arn:aws:iam::*:role/`*`custom-iam-role-name`*資源添加到`roleAliasResources`語句和測試用戶策略中的`passRoleForResources`語句中，如以下示例所示：

  Example **`passRoleForResources`** 陳述式

```
{
   "Sid":"passRoleForResources",
   "Effect":"Allow",
   "Action":"iam:PassRole",
```

```
   "Resource":"arn:aws:iam::*:role/custom-iam-role-name",
   "Condition":{
      "StringEquals":{
         "iam:PassedToService":[
            "iot.amazonaws.com",
            "lambda.amazonaws.com",
            "greengrass.amazonaws.com"
         ]
      }
   }
}
```

Example **`roleAliasResources`** 陳述式

```
{
   "Sid":"roleAliasResources",
   "Effect":"Allow",
   "Action":[
      "iot:CreateRoleAlias",
      "iot:DescribeRoleAlias",
      "iot:DeleteRoleAlias",
      "iot:TagResource",
      "iam:GetRole"
   ],
   "Resource":[
      "arn:aws:iot:*:*:rolealias/idt-*",
      "arn:aws:iam::*:role/custom-iam-role-name"
   ]
}
```

設定您的裝置以測試選用功能

本節說明針對選用的 Docker 和機器學習 (ML) 功能執行 IDT 測試的裝置需求。只有在您要測試這些功能時，才必須確保您的裝置符合這些需求。否則，請繼續進行the section called “設定 IDT 設定”。

主題

- 泊塢工人資格要求
- ML 資格要求
- HSM 資格要求

泊塢工人資格要求

IDT for AWS IoT Greengrass V2 提供 Docker 資格測試，以驗證您的設備是否可以使用提AWS供的 Docker 應用程序管理器組件來下載可以使用自定義 Docker 容器組件運行的 Docker 容器映像。如需建立自訂 Docker 元件的詳細資訊，請參閱運行碼頭容器。

要運行 Docker 資格測試，被測設備必須滿足以下要求才能部署 Docker 應用程序管理器組件。

- 碼頭引擎 1.9.1 或更高版本安裝在 Greengrass 核心設備上。版本 20.10 是經過驗證可與AWS IoT Greengrass核心軟件配合使用的最新版本。在部署執行 Docker 容器的元件之前，您必須直接在核心裝置上安裝 Docker。
- 在您部署此元件之前，Docker 精靈會在核心裝置上啟動並執行。
- 執行 Docker 容器元件的系統使用者必須具有根或系統管理員權限，或者您必須將 Docker 設定為以非 root 使用者或非管理員使用者的身分執行。
  - 在 Linux 裝置上，您可以將使用者新增至`docker`群組，以便在不使用的情況下呼叫`docker`指令`sudo`。
  - 在 Windows 裝置上，您可以將使用者新增至`docker-users`群組，以便在沒有系統管理員權限的情況下呼叫`docker`命令。

  Linux or Unix

  若要將`ggc_user`您用來執行 Docker 容器元件的非 root 使用者新增至`docker`群組，請執行下列命令。

  ```
  sudo usermod -aG docker ggc_user
  ```

  如需詳細資訊，請參閱以非 root 使用者身分管理 Docker。

  Windows Command Prompt (CMD)

  若要將`ggc_user`或您用來執行 Docker 容器元件的使用者新增至`docker-users`群組，請以系統管理員身分執行下列命令。

  ```
  net localgroup docker-users ggc_user /add
  ```

  Windows PowerShell

  若要將`ggc_user`或您用來執行 Docker 容器元件的使用者新增至`docker-users`群組，請以系統管理員身分執行下列命令。

```
Add-LocalGroupMember -Group docker-users -Member ggc_user
```

ML 資格要求

IDT for AWS IoT Greengrass V2 提供 ML 資格測試，以驗證您的裝置是否可以使用提AWS供的機器學習元件，使用深度學習執行階段或 TensorFlow Lite ML 架構在本機執行 ML 推論。如需有關在 Greengrass 裝置上執行 ML 推論的詳細資訊，請參閱。執行機器學習推論

若要執行 ML 資格測試，受測裝置必須符合下列需求，才能部署機器學習元件。

- 在運行 Amazon Linux 2 或 Ubuntu 18.04 的 Greengrass 核心設備上，GNU C 庫（glibc）2.27 版或更高版本安裝在設備上。
- 在 ARMV7L 設備上，如樹莓派，對於設備上安裝了 OpenCV-Python 的依賴關係。執行下列命令以安裝相依性。

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
 libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- 樹莓派運行樹莓派 OS 靶心設備必須滿足以下要求:
  - NumPy 1.22.4 或更新版本已安裝在裝置上。樹莓派 OS 靶心包括早期版本的 NumPy，因此您可以在設備上運行以下命令進 NumPy 行升級。

    ```
    pip3 install --upgrade numpy
    ```

  - 裝置上啟用的舊式攝影機堆疊。樹莓派 OS Bullseye 包括一個新的相機堆疊，預設情況下啟用且不相容，因此您必須啟用傳統的相機堆疊。

    啟用舊式相機堆疊

    1. 運行以下命令以打開樹莓派配置工具。

       ```
       sudo raspi-config
       ```

    2. 選取介面選項。
    3. 選取「舊式相機」以啟用舊式相機堆疊。
    4. 重新啟動 Raspberry Pi。

HSM 資格要求

AWS IoT Greengrass提供 PKCS #11 提供者元件，以與裝置上的 PKCS 硬體安全模組 (HSM) 整合。HSM 設定取決於您的裝置和您選擇的 HSM 模組。只要提供預期的 HSM 組態 (如 IDT 組態設定中所述)，IDT 就會擁有執行此選用功能限定測試所需的資訊。

設定 IDT 設定以執行AWS IoT Greengrass資格套件

在執行測試之前，您必須設定主機電腦上的AWS認證和裝置的設定。

在設定. AWS json 中設定認證

您必須在檔案中設定 IAM 使用者登入*<device_tester_extract_location>*/configs/config.json資料。使用中建立之 AWS IoT Greengrass V2 使用者之 IDT 的證明資料。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：

- 在認證檔案中
- 作為環境變量

使用AWS認證檔設定認證

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認證新增至`credentials`檔案：

```
[default]
aws_access_key_id = <your_access_key_id>
aws_secret_access_key = <your_secret_access_key>
```

若要設定 AWS IoT Greengrass V2 的 IDT 使用`credentials`檔案中的AWS認證，請依照下列步驟編輯`config.json`檔案：

```
{
```

```
  "awsRegion": "region",
  "auth": {
    "method": "file",
    "credentials": {
      "profile": "default"
    }
  }
}
```

> **Note**
>
> 如果您不使用`default`AWS設定檔，請務必變更檔案中的設定`config.json`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認證

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 V2 可以使用`AWS_ACCESS_KEY_ID`和`AWS_SECRET_ACCESS_KEY`環境變數來儲存您的AWS認證。

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": "region",
  "auth": {
    "method": "environment"
  }
}
```

設定 device.json

除了AWS認證之外，AWS IoT GreengrassV2 的 IDT 還需要有關在其上運行測試的設備的信息。範例資訊包括 IP 位址、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：

```
[
  {
    "id": "<pool-id>",
    "sku": "<sku>",
    "features": [
      {
        "name": "arch",
        "value": "x86_64 | armv6l | armv7l | aarch64"
      },
      {
        "name": "ml",
        "value": "dlr | tensorflowlite | dlr,tensorflowlite | no"
      },
      {
        "name": "docker",
        "value": "yes | no"
      },
      {
        "name": "streamManagement",
        "value": "yes | no"
      },
      {
        "name": "hsi",
        "value": "hsm | no"
      }
    ],
    "devices": [
      {
        "id": "<device-id>",
        "operatingSystem": "Linux | Windows",
        "connectivity": {
          "protocol": "ssh",
          "ip": "<ip-address>",
          "port": 22,
          "publicKeyPath": "<public-key-path>",
```

```
          "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`

包含值的所有屬性都是必要的，如下所述：

`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**
>
> 如果您要在裝置目錄中刊登您的AWS Partner裝置，您在此處指定的 SKU 必須與您在刊登程序中使用的 SKU 相符。

`features`

包含裝置支援功能的陣列。所有功能都是必需的。

`arch`

測試運行驗證的支持操作系統架構。有效的 值如下：

- `x86_64`
- `armv6l`
- `armv7l`
- `aarch64`

`ml`

驗證裝置是否符合使用AWS提供的機器學習 (ML) 元件所需的所有技術相依性。

啟用此功能也會驗證裝置是否可以使用深度學習執行階段和 TensorFlow Lite ML 架構執行 ML 推論。

有效值是`dlr`與`tensorflowlite`、或的任意組合`no`。

`docker`

驗證裝置是否符合使用AWS提供的 Docker 應用程式管理員 () `aws.greengrass.DockerApplicationManager` 元件所需的所有技術相依性。

啟用此功能也會驗證裝置是否可以從 Amazon ECR 下載 Docker 容器映像。

有效值是`yes`或的任意組合`no`。

`streamManagement`

驗證裝置是否可以下載、安裝和執行AWS IoT Greengrass串流管理員。

有效值是`yes`或的任意組合`no`。

`hsi`

驗證裝置可以使用儲存在硬體安全性模組 (HSM) 中的私密金鑰和憑證來驗證與和AWS IoT Greengrass服務的連線。AWS IoT此測試也會驗證提供的 PKCS #11 AWS 提供者元件是否可以使用廠商提供的 PKCS #11 程式庫與 HSM 連接。如需詳細資訊，請參閱 硬體安全整合。

有效值為 `hsm` 或 `no`。

> **Note**
>
> IDT v4.2.0 及更新版本支援測試`ml`、`docker`、和功能。`streamManagement`如果您不想測試這些功能，請將對應的值設定為`no`。

> **Note**
>
> 測試僅hsi適用於 IDT v4.5.1 及更高版本。

`devices.id`

使用者定義的唯一識別符，用於識別要測試的裝置。

`devices.operatingSystem`

裝置作業系統。支援的值為 `Linux` 和 `Windows`。

`connectivity.protocol`

用來與此裝置通訊的通訊協定。目前，唯一支援的值ssh適用於實體裝置。

`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.publicKeyPath`

選用。用於驗證與被測設備的連接的公鑰的完整路徑。

當您指定時publicKeyPath，IDT 會在設備與被測設備建立 SSH 連接時驗證設備的公鑰。如果未指定此值，IDT 會建立 SSH 連線，但不會驗證裝置的公開金鑰。

強烈建議您指定公開金鑰的路徑，並使用安全方法來擷取此公開金鑰。對於基於命令行的標準 SSH 客戶端，公鑰在known_hosts文件中提供。如果您指定個別的公開金鑰檔案，則此檔案必須使用與known_hosts檔案相同的格式，也就是 *`ip-address key-type public-key`*。如果有多個項目具有相同 IP 位址，IDT 使用的金鑰類型項目必須在檔案中的其他項目之前。

`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

用於驗證的燈入資料。

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。

配置用戶數據

適用於 AWS IoT Greengrass V2 的 IDT 還需要有關測試成品和AWS IoT Greengrass軟件位置的其他信息。

您必須使用位於 *<device_tester_extract_location>*/configs/userdata.json 中的 userdata.json 範本提供此資訊：

```
{
    "TempResourcesDirOnDevice": "/path/to/temp/folder",
    "InstallationDirRootOnDevice": "/path/to/installation/folder",
    "GreengrassNucleusZip": "/path/to/aws.greengrass.nucleus.zip",
```

```
    "PreInstalled": "yes/no",
    "GreengrassV2TokenExchangeRole": "custom-iam-role-name",
 "hsm": {
        "greengrassPkcsPluginJar": "/path/to/aws.greengrass.crypto.Pkcs11Provider-
latest.jar",
        "pkcs11ProviderLibrary": "/path/to/pkcs11-vendor-library",
        "slotId": "slot-id",
        "slotLabel": "slot-label",
        "slotUserPin": "slot-pin",
        "keyLabel": "key-label",
        "preloadedCertificateArn": "certificate-arn"
        "rootCA": "path/to/root-ca"
    }
}
```

包含值的所有屬性都是必需的，如下所述：

`TempResourcesDirOnDevice`

待測設備上存儲測試工件的臨時文件夾的完整路徑。請確定不需要 sudo 權限即可寫入此目錄。

> **Note**
>
> IDT 會在完成測試執行時刪除此資料夾的內容。

`InstallationDirRootOnDevice`

要安裝之裝置上資料夾的完整路徑AWS IoT Greengrass。對於 PreInstalled Greengrass，這是安裝目錄的路徑。

您必須為此資料夾設定必要的檔案權限。針對安裝路徑中的每個資料夾執行下列命令。

```
sudo chmod 755 folder-name
```

`GreengrassNucleusZip`

主機電腦上 Greengrass 核 ZIP (`greengrass-nucleus-latest.zip`) 檔案的完整路徑。使用 PreInstalled Greengrass 進行測試時不需要此欄位。

> **Note**
>
> 如需 IDT 的 Greengrass 核受支援版本的資訊，請參閱。AWS IoT Greengrass 適用於 V2 的最新 IDT 版本 AWS IoT Greengrass若要下載最新的 Greengrass 軟體，請參閱下載軟體。AWS IoT Greengrass

`PreInstalled`

此功能僅適用於 Linux 裝置上的 IDT v4.5.8 及更新版本。

(選擇性) 當值為 *`yes`* 時，IDT 會假設中提到的路徑`InstallationDirRootOnDevice`為安裝 Greengrass 的目錄。

如需有關如何在裝置上安裝 Greengrass 的詳細資訊，請參閱。使用自動資源佈建安裝 AWS IoT Greengrass核心軟體如果使用手動佈建進行安裝，請在手動建立AWS IoT物件時包含「將物件新增至新的或現有的AWS IoT物件群組」步驟。IDT 假設物件與物件群組是在安裝設定期間建立的。請確定這些值反映在`effectiveConfig.yaml`檔案中。IDT 會檢查`effectiveConfig.yaml`下`<InstallationDirRootOnDevice>/config/effectiveConfig.yaml`的檔案。

若要使用 HSM 執行測試，請確定已在中`effectiveConfig.yaml`更新`aws.greengrass.crypto.Pkcs11Provider`欄位。

`GreengrassV2TokenExchangeRole`

（可選）要用作令牌交換角色的自定義 IAM 角色，該角色被測設備假定與AWS資源進行交互。

> **Note**
>
> IDT 使用此自定義 IAM 角色，而不是在測試運行期間創建默認令牌交換角色。如果您使用自訂角色，則可以更新測試使用者的 IAM 許可，以排除允許使用者建立和刪除 IAM 角色和政策的`iamResourcesUpdate`陳述式。

如需建立自訂 IAM 角色做為權杖交換角色的詳細資訊，請參閱設定自訂權杖交換角色。

`hsm`

此功能適用於 IDT v4.5.1 及更新版本。

(選擇性) 使用AWS IoT Greengrass硬體安全性模組 (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。

> **Warning**
>
> 如果硬體安全模組在 IDT 與其他系統之間共用，則 HSM 組態可能會被視為敏感資料。在此情況下，您可以避免將這些組態值儲存在參數存放區AWS SecureString 參數中，並將 IDT 設定為在測試執行期間擷取它們，以避免以純文字形式保護這些組態值。如需更多資訊，請參閱 ???

`hsm.greengrassPkcsPluginJar`

您下載至 IDT 主機的 PKCS #11 提供者元件的完整路徑。 AWS IoT Greengrass將此元件提供為 JAR 檔案，您可以下載該檔案，以便在安裝期間指定為佈建外掛程式。您可以透過下列網址下載最新版本的元件 JAR 檔案：https://d2s8p88vqu9w66.cloudfront.net/releases/Pkcs11Provider/aws.greengrass.crypto.Pkcs11Provider-latest.jar。

`hsm.pkcs11ProviderLibrary`

PKCS #11 程式庫的完整路徑，由硬體安全性模組 (HSM) 廠商所提供，可與 HSM 互動。

`hsm.slotId`

用於識別您將金鑰和憑證載入的 HSM 插槽的插槽識別碼。

`hsm.slotLabel`

用於識別您將金鑰和憑證載入的 HSM 插槽的插槽標籤。

`hsm.slotUserPin`

IDT 用來向 HSM 驗證AWS IoT Greengrass核心軟體的使用者 PIN 碼。

> **Note**
>
> 安全性最佳做法是，請勿在生產裝置上使用相同的使用者 PIN 碼。

`hsm.keyLabel`

用於識別硬體模組中之金鑰的標籤。金鑰和憑證都必須使用相同的金鑰標籤。

`hsm.preloadedCertificateArn`

AWS IoT雲端中已上傳裝置憑證的 Amazon 資源名稱 (ARN)。

您必須先前使用 HSM 中的金鑰產生此憑證，並將其匯入您的 HSM，並將其上傳到雲端AWS IoT。如需產生及匯入憑證的相關資訊，請參閱 HSM 的說明文件。

您必須將憑證上傳到您在 config.json 中提供的相同帳戶和區域。 。如需將憑證上傳至的詳細資訊AWS IoT，請參閱AWS IoT開發人員指南中的手動註冊用戶端憑證。

`hsm.rootCAPath`

(選擇性) IDT 主機上到簽署憑證之根憑證授權單位 (CA) 的完整路徑。如果在您建立的 HSM 中的憑證未由 Amazon 根 CA 簽署，則需要這個動作。

從AWS參數存放區擷取組態

AWS IoT裝置測試器 (IDT) 包含選用功能，可從「AWSSystems Manager 參數存放區」擷取組態值。AWS參數存放區允許安全且加密的組態儲存。設定完成後，IDT 可以從AWS參數存放區擷取參數，以取代以純文字形式將參數儲存在檔案內。`userdata.json`這對於應加密存儲的任何敏感數據非常有用，例如：密碼，PIN 和其他密碼。

1. 若要使用此功能，您必須更新建立 IDT 使用者時所使用的權限，以允許針對已配置 IDT 使用的參數 GetParameter 執行動作。以下是可新增至 IDT 使用者的權限陳述式範例。如需詳細資訊，請參閱AWS Systems Manager使用者指南。

   ```
   {
          "Sid":"parameterStoreResources",
          "Effect": "Allow",
           "Action": [
               "ssm:GetParameter"
           ],
           "Resource": "arn:aws:ssm:*:*:parameter/IDT*"
   }
   ```

   上述權限設定為允許使用萬用字*元擷取名稱開IDT頭為的所有參數。您應該根據需要對其進行自定義，以便 IDT 可以根據您正在使用的參數的命名獲取任何配置的參數。

2. 您需要在AWS參數商店中存儲配置值。這可以從AWS控制台或從 AWS CLI 完成。 AWS參數存放區可讓您選擇加密或未加密的儲存。若要儲存機密、密碼和 pin 碼等敏感值，您應該使用的是參數類型的加密選項 SecureString。若要使用 AWS CLI 上傳參數，您可以使用下列命令：

```
aws ssm put-parameter --name IDT-example-name --value IDT-example-value --type
 SecureString
```

您可以使用下列指令驗證是否已儲存參數。(選擇性) 使用--with-decryption旗標擷取已解密的SecureString 參數。

```
aws ssm get-parameter --name IDT-example-name
```

使用 AWS CLI 將在目前 CLI 使用者的AWS區域中上傳參數，IDT 會從中config.json設定的區域擷取參數。若要從 AWS CLI 檢查您的區域，請使用下列指令：

```
aws configure get region
```

3. 一旦您在AWS雲中有一個配置值，您可以更新 IDT 配置中的任何值以從AWS雲中獲取。若要這麼做，您可以在表單{{AWS.Parameter.parameter_name}}的 IDT 組態中使用預留位置，從參數存放區依該名稱擷取AWS參數。

   例如，假設您想要在 HSM 組態中使用步驟 2 中的IDT-example-name參數作為 HSM 金鑰標籤。要做到這一點，你可以更新你的userdata.json如下：

```
"hsm": {
        "keyLabel": "{{AWS.Parameter.IDT-example-name}}",
        [...]
    }
```

   IDT 會在執行階段擷取IDT-example-value在步驟 2 中設定為的此參數值。此設定與設定類似，"keyLabel": "IDT-example-value"但是該值會以加密方式儲存在AWS雲端中。

執行資AWS IoT Greengrass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。

使用下面的run-suite命令來運行一套測試。

```
devicetester_[linux | mac | win]_x86-64 run-suite  \\
    --suite-id suite-id  \\
    --group-id group-id  \\
```

```
    --pool-id your-device-pool \\
    --test-id test-id  \\
    --update-idt y|n  \\
    --userdata userdata.json
```

所有選項都是可選的。例如，pool-id如果您只有一個裝置集區，也就是device.json檔案中定義的一組相同裝置，則可以省略。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 “測試套件版本”。

執行限定套件的範例命令

下列命令列範例說明如何針對裝置集區執行資格測試。如需 run-suite 和其他 IDT 命令的詳細資訊，請參閱 the section called “IDT 命令”。

使用以下命令在指定的測試套件中運行所有測試組。該list-suites命令列出了tests文件夾中的測試套件。

```
devicetester_[linux | mac | win]_x86-64 run-suite \
    --suite-id GGV2Q_1.0.0 \
    --pool-id <pool-id> \
    --userdata userdata.json
```

使用以下命令在測試套件中運行特定的測試組。該list-groups命令列出了測試套件中的測試組。

```
devicetester_[linux | mac | win]_x86-64 run-suite \
    --suite-id GGV2Q_1.0.0 \
    --group-id <group-id> \
    --pool-id <pool-id> \
    --userdata userdata.json
```

使用以下命令在測試組中運行特定的測試用例。

```
devicetester_[linux | mac | win]_x86-64 run-suite \
```

```
    --group-id <group-id> \
    --test-id <test-id> \
    --userdata userdata.json
```

使用以下命令在測試組中運行多個測試用例。

```
devicetester_[linux | mac | win]_x86-64 run-suite \
    --group-id <group-id> \
    --test-id <test-id1>,<test-id2>
    --userdata userdata.json
```

使用以下命令列出測試組中的所有測試用例。

```
devicetester_[linux | mac | win]_x86-64 list-test-cases --group-id <group-id>
```

我們建議您執行完整的限定測試套件，該套件會以正確的順序執行測試群組相依性。如果您選擇執行特定的測試群組，建議您先執行相依性檢查程式測試群組，以確保在執行相關測試群組之前已安裝所有 Greengrass 相依性。例如：

- 在執行核心資格測試組之前執行 `coredependencies`。

適用於 AWS IoT Greengrass V2 指令的 IDT

IDT 命令位於 *<device-tester-extract-location>*/bin 目錄中。若要執行測試套件，請以下列格式提供命令：

`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 版本支援的產品 (即本例中的 AWS IoT Greengrass 版本) 和測試套件版本。

`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`。 要運行的測試套件版本。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，以逗號分隔的清單形式。如果未指定，IDT 會根據中設定的設定，在測試套件中執行所有適當的測試群組。`device.json`IDT 不會根據您配置的設置運行設備不支持的任何測試組，即使在`group-id`列表中指定了這些測試組也是如此。
- `test-id`。 要運行的測試用例，以逗號分隔的列表。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。
- `stop-on-first-failure`。 將 IDT 設定為在第一次失敗時停止執行。`group-id`當您要偵錯指定的測試群組時，請使用此選項。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。如果 IDT 檢測到有更新的版本，`Y`響應將停止測試執行。`N`回應會繼續執行測試。
- `userdata`。 包含測試人工因素路徑相關資訊之`userdata.json`檔案的完整路徑。此選項是`run-suite`指令所需的。*`#userdata.json############## /devicetester_ ggv2_ [WIN| ## | Linux] /##/ ####`*

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。

若要疑難排解錯誤，請參閱[疑難排解 IDTAWS IoT GreengrassV2](#)。

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。這些報告位於中*`<device-tester-extract-location>`*/results/*`<execution-id>`*/。這兩個報告都會擷取執行資格測試套件的結果。

這awsiotdevicetester_report.xml是您提交的資格測試報告，以AWS便在設備目錄中列出您的AWS Partner設備。該報告包含下列元素：

- IDT 版本。
- 經過測試的 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。
- 根據設備功能（例如本地資源訪問，陰影和 MQTT）進行測試的庫進行測試的測試結果明細。

GGV2Q_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。
- 依測試的 AWS IoT Greengrass 功能將測試結果分類。

解譯AWS IoT Device Tester結果

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>包含測試運行的摘要。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

執行的測試數目。

`failures`

已執行但未通過的測試次數。

`errors`

IDT 無法運行的測試數量。

`disabled`

忽略此屬性。不會使用。

`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。

```
<name="aws-iot-greengrass-v2-core" value="supported" type="required"></feature>
```

如果所需功能沒有測試失敗或錯誤，您的裝置即符合執行 AWS IoT Greengrass 的技術要求，可與 AWS IoT 服務相互運作。如果您想要在裝置目錄中列出AWS Partner裝置，您可以使用此報告作為資格證據。

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<testcase>籤中，針對測試群組執行的每個測試都有標籤。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 從中的測試運行生成日誌*<devicetester-extract-location>*/results/*<execution-id>*/logs。該工具會產生兩組日誌：

test_manager.log

從的測試管理員元件產生的記錄檔AWS IoT Device Tester (例如，與組態、測試順序和報告產生相關的記錄檔)。

<test-case-id>.log (for example, lambdaDeploymentTest.log)

測試組中測試用例的日誌，包括來自被測設備的日誌。從 IDT v4.2.0 開始，IDT 將每個測試用例的測試日誌分組在*<devicetester-extract-location>*/results/*<execution-id>*/logs/*<test-group-id>/*目錄中的單獨 *<test-case-id >* 文件夾中。

使用 IDT 開發和運行自己的測試套件

從 IDT v4.0.1 開始，適用於 AWS IoT Greengrass V2 的 IDT 將標準化的組態設定和結果格式與測試套件環境結合在一起，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添加自定義測試，也可以將其提供給客戶進行設備驗證。

使用 IDT 開發和執行自訂測試套件，如下所示：

若要開發自訂測試套件

- 為您要測試的 Greengrass 設備創建具有自定義測試邏輯的測試套件。
- 為 IDT 提供您的自定義測試套件以測試運行者。包括有關測試套件的特定設置配置的信息。

執行自訂測試套件

- 設定您要測試的裝置。
- 根據要使用的測試套件的要求實施設置配置。
- 使用 IDT 運行您的自定義測試套件。
- 檢視 IDT 執行之測試的測試結果和執行記錄。

下載最新版本AWS IoT Device Tester的 AWS IoT Greengrass

下載最新版本的 IDT，並將軟體解壓縮至檔案系統上具有讀取/寫入權限的位置 (*< device-tester-extract-location >*)。

> **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

測試套件由三種類型的文件組成：

- 為 IDT 提供有關如何運行測試套件的信息的配置文件。
- 測試 IDT 用於運行測試用例的可執行文件。

- 運行測試所需的其他文件。

完成下列基本步驟以建立自訂 IDT 測試：

1. 為您的測試套件創建配置文件。
2. 創建包含測試套件測試邏輯的測試用例可執行文件。
3. 驗證並記錄測試運行程序運行測試套件所需的配置信息。
4. 驗證 IDT 可以按預期運行測試套件並產生測試結果。

若要快速建置範例自訂套件並執行它，請遵循中的指示教學課程：建置並執行範例 IDT 測試套件。

要開始在 Python 中創建自定義測試套件，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

此下AWS IoT Device Tester載包含範例測試套件的原始程式碼。您可以完成本教程來構建和運行示例測試套件，以了解如何使用 IDT AWS IoT Greengrass 來運行自定義測試套件。

在本教學課程中，您將完成以下步驟：

1. 建置範例測試套件
2. 使用 IDT 運行示例測試套件

必要條件

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 或更高版本

    若要檢查電腦上安裝的 Python 版本，請執行下列命令：

    ```
    python3 --version
    ```

在 Windows 上，如果使用此命令返回錯誤，則使用`python --version`代替。如果傳回的版本號碼為 3.7 或更高版本，請在 PowerShell 終端機中執行下列命令，以設定`python3`為`python`命令的別名。

```
Set-Alias -Name "python3" -Value "python"
```

如果沒有傳回任何版本資訊，或是版本號碼小於 3.7，請依照下載 Python 中的指示安裝 Python 3.7+。如需詳細資訊，請參閱 Python 文件。

- urllib3

  若要確認`urllib3`是否已正確安裝，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 作業系統以及與主機電腦相同網路的網路連線的裝置。

    我們建議您使用樹莓派與樹莓派操作系統. 確保您在樹莓派上設置了 SSH 以遠程連接到它。

設定 IDT 的裝置資訊

配置 IDT 的設備信息以運行測試。您必須使用下列資訊更新位於資*<device-tester-extract-location>*/configs料夾中的`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碼。

`connectivity.ip`

您裝置的 IP 位址。

`connectivity.port`

選用。用於連線至裝置的 SSH 連線的連接埠號碼。

`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`

用於驗證的燈入資料。

`connectivity.auth.credentials.user`

用於登入裝置的使用者名稱。

`connectivity.auth.credentials.privKeyPath`

用於登入裝置之私密金鑰的完整路徑。

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

該文件*<device-tester-extract-location>*/samples/python夾包含示例配置文件，源代碼和 IDT Client SDK，您可以使用提供的構建腳本將其合併到測試套件中。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
```

若要建立測試套件，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
```

這會在資料夾內的IDTSampleSuitePython_1.0.0資料夾中建立範例測試套*<device-tester-extract-location>*/tests件。檢閱IDTSampleSuitePython_1.0.0資料夾中的檔案，瞭解範例測試套件的結構方式，並查看測試案例可執行檔和測試設定 JSON 檔案的各種範例。

> **Note**
>
> 示例測試套件包括 python 源代碼。請勿在測試套件代碼中包含敏感信息。

下一步：使用 IDT 運行您創建的示例測試套件。

使用 IDT 運行示例測試套件

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 運行示例測試套件並將結果流式傳輸到控制台。測試執行完成後，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
```

故障診斷

使用下列資訊可協助解決完成自學課程時的任何問題。

測試用例未成功運行

如果測試未成功運行，IDT 將錯誤日誌流式傳輸到控制台，以幫助您對測試運行進行故障排除。請確定您符合本教學課程的所有先決條件。

無法連接到被測設備

請確認下列內容：

- 您的`device.json`檔案包含正確的 IP 位址、連接埠和驗證資訊。
- 您可以通過 SSH 從主機連接到設備。

教學課程：開發簡單的 IDT 測試套件

測試套件結合了以下內容：

- 測試包含測試邏輯的可執行文件
- 描述測試套件的配置文件

本教程向您展示如何使用 IDT AWS IoT Greengrass 來開發包含單個測試用例的 Python 測試套件。在本教學課程中，您將完成以下步驟：

1. 創建測試套件目錄
2. 建立組態檔

3. 創建測試用例可執行文件
4. 運行測試套件

必要條件

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 或更高版本

    若要檢查電腦上安裝的 Python 版本，請執行下列命令：

    ```
    python3 --version
    ```

    在 Windows 上，如果使用此命令返回錯誤，則使用`python --version`代替。如果傳回的版本號碼為 3.7 或更高版本，請在 PowerShell 終端機中執行下列命令，以設定`python3`為`python`命令的別名。

    ```
    Set-Alias -Name "python3" -Value "python"
    ```

    如果沒有傳回任何版本資訊，或是版本號碼小於 3.7，請依照下載 Python 中的指示安裝 Python 3.7+。如需詳細資訊，請參閱 Python 文件。
  - urllib3

    若要確認`urllib3`是否已正確安裝，請執行下列命令：

    ```
    python3 -c 'import urllib3'
    ```

    如果`urllib3`未安裝，請執行下列命令進行安裝：

    ```
    python3 -m pip install urllib3
    ```
- 裝置要求
  - 具有 Linux 作業系統以及與主機電腦相同網路的網路連線的裝置。

    我們建議您使用樹莓派與樹莓派操作系統. 確保您在樹莓派上設置了 SSH 以遠程連接到它。

創建測試套件目錄

IDT 邏輯上將測試用例分離為每個測試套件中的測試組。每個測試用例必須位於測試組內。在本教學課程中，請建立名為的資料夾，MyTestSuite_1.0.0並在此資料夾中建立下列目錄樹：

```
MyTestSuite_1.0.0
### suite
    ### myTestGroup
        ### myTestCase
```

建立組態檔

您的測試套件必須包含以下必要的配置文件：

必要的組態檔

suite.json

包含測試套件的相關資訊。請參閱 設定套件。

group.json

包含測試群組的相關資訊。您必須為測試套group.json件中的每個測試組創建一個文件。請參閱設定群組。

test.json

包含有關測試用例的信息。您必須為測試套test.json件中的每個測試用例創建一個文件。請參閱配置測試。

1. 在MyTestSuite_1.0.0/suite資料夾中，建立具有下列結構的suite.json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，建立具有下列結構的group.json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，建立具有下列結構的test.json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"
            ]
        }
    }
}
```

MyTestSuite_1.0.0資料夾的目錄樹現在應該如下所示：

```
MyTestSuite_1.0.0
### suite
    ### suite.json
    ### myTestGroup
```

```
        ### group.json
        ### myTestCase
            ### test.json
```

取得 IDT 用戶端開發套件

您可以使用 IDT 用戶端 SDK 來啟用 IDT 與待測裝置互動，並報告測試結果。在本教程中，您將使用 Python 版本的 SDK。

從*<device-tester-extract-location>*/sdks/python/資料夾中，將資idt_client料夾複製到您的MyTestSuite_1.0.0/suite/myTestGroup/myTestCase資料夾。

若要驗證 SDK 是否已成功複製，請執行下列命令。

```
cd MyTestSuite_1.0.0/suite/myTestGroup/myTestCase
python3 -c 'import idt_client'
```

創建測試用例可執行文件

測試用例可執行文件包含要運行的測試邏輯。一個測試套件可以包含多個測試用例可執行文件。在本教程中，您將只創建一個測試用例可執行文件。

1. 建立測試套件檔案。

   在MyTestSuite_1.0.0/suite/myTestGroup/myTestCase資料夾中，建立包含下列內容的myTestCase.py檔案：

   ```
   from idt_client import *

   def main():
       # Use the client SDK to communicate with IDT
       client = Client()

   if __name__ == "__main__":
       main()
   ```

2. 使用客戶端 SDK 函數將以下測試邏輯添加到您的myTestCase.py文件中：

   a. 在被測設備上運行 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. 將測試結果發送給 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

配置 IDT 的設備信息以運行測試。您必須使用下列資訊更新位於資*<device-tester-extract-location>*/configs料夾中的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碼。

`connectivity.ip`

您裝置的 IP 位址。

`connectivity.port`

選用。用於連線至裝置的 SSH 連線的連接埠號碼。

`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`

用於驗證的燈入資料。

`connectivity.auth.credentials.user`

用於登入裝置的使用者名稱。

`connectivity.auth.credentials.privKeyPath`

用於登入裝置之私密金鑰的完整路徑。

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`

運行測試套件

創建測試套件後，要確保它按預期運行。完成以下步驟以使用現有設備池運行測試套件以執行此操作。

1. 將您的資MyTestSuite_1.0.0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 運行您的測試套件並將結果流式傳輸到控制台。測試執行完成後，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
```

故障診斷

使用下列資訊可協助解決完成自學課程時的任何問題。

測試用例未成功運行

如果測試未成功運行，IDT 將錯誤日誌流式傳輸到控制台，以幫助您對測試運行進行故障排除。檢查錯誤記錄檔之前，請先確認下列事項：

- 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。

無法連接到被測設備

請確認下列內容：

- 您的device.json檔案包含正確的 IP 位址、連接埠和驗證資訊。
- 您可以通過 SSH 從主機連接到設備。

建立 IDT 測試套件設定檔

本節說明您在撰寫自訂測試套件時所包含的組態檔案所使用的格式。

必要的組態檔案

`suite.json`

包含測試套件的相關資訊。請參閱 設定套件。

`group.json`

包含測試群組的相關資訊。您必須為測試套group.json件中的每個測試組創建一個文件。請參閱設定群組。

`test.json`

包含有關測試用例的信息。您必須為測試套test.json件中的每個測試用例創建一個文件。請參閱配置測試。

可選配置文件

`test_orchestrator.yaml` 或 `state_machine.json`

定義 IDT 執行測試套件時如何執行測試。上交所設定測試協調工具。

> **Note**
>
> 從 IDT v4.5.1 開始，您可以使用test_orchestrator.yaml檔案來定義測試工作流程。在舊版 IDT 中，您可以使用該state_machine.json檔案。如需狀態機的相關資訊，請參閱配置 IDT 狀態機。

userdata_schema.json

定義測試運行程序可以包含在其設置配置中的userdata.json文件的模式。該userdata.json文件用於運行測試所需的任何其他配置信息，但不存在於device.json文件中。請參閱 配置用戶數據模式。

配置文件放置在您的*<custom-test-suite-folder>*，如下所示。

```
<custom-test-suite-folder>
### suite
    ### suite.json
    ### test_orchestrator.yaml
    ### userdata_schema.json
    ### <test-group-folder>
        ### group.json
        ### <test-case-folder>
            ### test.json
```

設定套件

該suite.json文件設置環境變量，並確定是否需要用戶數據來運行測試套件。使用下列範本來設定*<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`必須與檔案所在的測試套`suite.json`件資料夾名稱相符。套件名稱和套件版本也必須符合下列需求：

- *`<suite-name>`*不能包含底線。
- *`<suite-version>`*被表示為*`x.x.x`*，其中`x`是一個數字。

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。測試執行程式的名稱會顯示在 IDT CLI 中。

`details`

測試套件用途的簡短描述。

`userDataRequired`

定義測試執行程式是否需要在`userdata.json`檔案中包含自訂資訊。如果將此值設定為`true`，則還必須在測試套[userdata_schema.json件資料夾中包含該檔案](#)。

`environmentVariables`

選用。要為此測試套件設置的環境變量數組。

`environmentVariables.key`

環境變數的名稱。

`environmentVariables.value`

環境變數的值。

設定群組

該group.json文件定義測試組是必需的還是可選的。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檔案所在之測試群組資料夾的名稱，且不能包含底線 (_)。ID 會用於 IDT 產生的測試報告中。

title

測試群組的描述性名稱。測試執行程式的名稱會顯示在 IDT CLI 中。

details

測試群組用途的簡短描述。

optional

選用。設定true為可在 IDT 完成執行必要測試後，將此測試群組顯示為選用群組。預設值為false。

配置測試

該test.json文件確定測試用例可執行文件和由測試用例使用的環境變量。如需建立測試案例可執行檔的詳細資訊，請參閱創建 IDT 測試用例可執行文件。

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`

測試用例的唯一用戶定義 ID。的值`id`必須與檔案所在之測試`test.json`案例資料夾的名稱相符，且不能包含底線 (_)。ID 會用於 IDT 產生的測試報告中。

`title`

測試用例的描述性名稱。測試執行程式的名稱會顯示在 IDT CLI 中。

`details`

測試用例目的的簡短描述。

`requireDUT`

選用。設定為是`true`否需要裝置才能執行此測試，否則將設定為`false`。預設值為 `true`。測試運行者將配置他們將用於在其`device.json`文件中運行測試的設備。

`requiredResources`

選用。陣列，提供執行此測試所需資源裝置的相關資訊。

`requiredResources.name`

此測試運行時為資源設備提供的唯一名稱。

`requiredResources.features`

使用者定義的資源裝置功能陣列。

`requiredResources.features.name`

特徵的名稱。您要使用此裝置的裝置功能。此名稱與`resource.json`文件中測試運行器提供的功能名稱匹配。

`requiredResources.features.version`

選用。功能的版本。該值與`resource.json`文件中的測試運行程序提供的功能版本匹配。如果未提供版本，則不檢查該功能。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次數。預設值為 1。如果您希望 IDT 針對個別功能使用不同的裝置，建議您將`1`此值設定為。

`execution.timeout`

IDT 等待測試完成執行的時間 (以毫秒為單位)。若要取得有關設定此值的更多資訊，請參閱[創建 IDT 測試用例可執行文件](#)。

`execution.os`

根據執行 IDT 之主機電腦的作業系統，要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。要提供執行測試案例可執行檔的引數。

`environmentVariables`

選用。為此測試用例設置的環境變量數組。

`environmentVariables.key`

環境變數的名稱。

`environmentVariables.value`

環境變數的值。

> **Note**
>
> 如果您在`test.json`檔案和檔案中指定相同的環境變數，則`test.json`檔案中的值優先。`suite.json`

設定測試協調工具

測試協調器是一種控制測試套件執行流程的構造。它會判斷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，並繼續透過這些狀態轉換，直到達到結束狀態為止。

如果您的測試套件不包含用戶定義的測試協調器，IDT 將為您生成測試協調器。

默認測試協調器執行以下功能：

- 為測試運行者提供選擇和運行特定測試組的功能，而不是整個測試套件。
- 如果未選擇特定的測試組，請以隨機順序運行測試套件中的每個測試組。
- 生成報告並打印顯示每個測試組和測試用例的測試結果的控制台摘要。

如需 IDT 測試協調器如何運作的詳細資訊，請參閱。配置 IDT 測試編排器

配置用戶數據模式

該userdata_schema.json文件確定測試運行程序提供用戶數據的模式。如果您的測試套件需要device.json文件中不存在的信息，則需要用戶數據。例如，您的測試可能需要使用者必須提供的 Wi-Fi 網路認證、特定開放連接埠或憑證。此資訊可以作為使用者在其資*<device-tester-extract-location>*/config料夾中建立的名為userdatauserdata.json檔案的輸入參數提供給 IDT。該userdata.json文件的格式基於您包含在測試套userdata_schema.json件中的文件。

為了表示測試運行者必須提供一個userdata.json文件：

1. 在suite.json檔案中，設定userDataRequired為true。
2. 在您的*<custom-test-suite-folder>*中建立userdata_schema.json檔案。
3. 編輯該userdata_schema.json文件以創建一個有效的 IETF 草案 v4 JSON 模式。

當 IDT 運行測試套件時，它會自動讀取模式並使用它來驗證測試運行器提供的userdata.json文件。如果有效，則userdata.json檔案內容可在 IDT 前後關聯和測試協調器內容中使用。

配置 IDT 測試編排器

從 IDT 版本 4.5.1 開始，IDT 包含一個新的測試協調流程元件。測試編排器是一個 IDT 組件，用於控制測試套件執行流程，並在 IDT 完成運行所有測試後生成測試報告。測試編排器確定測試選擇以及基於用户定義的規則運行測試的順序。

如果測試套件不包含用户定義的測試編排器，IDT 將為您生成測試編排器。

默認測試編排器執行以下功能：

- 為測試運行者提供選擇和運行特定測試組的能力，而不是整個測試套件。
- 如果未選擇特定測試組，則以隨機順序運行測試套件中的每個測試組。
- 生成報告並打印顯示每個測試組和測試用例的測試結果的控制台摘要。

測試編排器替換 IDT 測試編排器。強烈建議您使用測試編排器來開發測試套件，而不是 IDT 測試編排器。測試編排器提供了以下改進的功能：

- 使用聲明格式與 IDT 狀態機使用的命令式格式進行比較。這可讓您指定要運行的測試和何時你想運行它們。

- 管理特定的組處理、報告生成、錯誤處理和結果跟蹤，以便您不需要手動管理這些操作。
- 使用 YAML 格式，默認情況下支持註釋。
- 需要百分之八十比測試編排器更少的磁盤空間來定義相同的工作流。
- 添加測試前驗證以驗證工作流定義不包含不正確的測試 ID 或循環依賴關係。

測試編排器格式

您可以使用以下模板配置自己的*<custom-test-suite-folder>*/suite/test_orchestrator.yaml文件:

```
Aliases:
  string: context-expression

ConditionalTests:
  - Condition: context-expression
    Tests:
      - test-descriptor

Order:
  - - group-descriptor
    - group-descriptor

Features:
  - Name: feature-name
    Value: support-description
    Condition: context-expression
    Tests:
        - test-descriptor
    OneOfTests:
        - test-descriptor
    IsRequired: boolean
```

如下所述，包含值的所有欄位皆為必要：

`Aliases`

選用。映射到上下文表達式的用户定義字符串。別名允許您生成友好名稱以標識測試編排器配置中的上下文表達式。如果您正在創建複雜的內容表達式或在多個地方使用的表達式，此功能特別有用。

您可以使用上下文表達式來存儲允許您訪問其他 IDT 配置中的數據的上下文查詢。如需詳細資訊，請參閱 訪問上下文中的數據。

Example 範例

```
Aliases:
    FizzChosen: "'{{$pool.features[?(@.name == 'Fizz')].value[0]}}' == 'yes'"
    BuzzChosen: "'{{$pool.features[?(@.name == 'Buzz')].value[0]}}' == 'yes'"
    FizzBuzzChosen: "'{{$aliases.FizzChosen}}' && '{{$aliases.BuzzChosen}}'"
```

ConditionalTests

選用。條件列表以及滿足每個條件時運行的相應測試用例。每個條件都可以有多個測試用例；但是，您只能將給定的測試用例分配給一個條件。

默認情況下，IDT 運行未分配給此列表中條件的任何測試用例。如果您未指定此部分，IDT 會執行測試套件中的所有測試組。

中的每個項目ConditionalTests包含下列參數：

Condition

判斷值為布林值值。如果計算值為 true，IDT 將運行在Tests參數。

Tests

測試描述符的列表。

每個測試描述符使用測試組 ID 和一個或多個測試用例 ID 來標識要從特定測試組運行的單個測試。測試描述符使用下列格式：

```
GroupId: group-id
CaseIds: [test-id, test-id] # optional
```

Example 範例

以下示例使用通用上下文表達式，您可以將其定義為Aliases。

```
ConditionalTests:
    - Condition: "{{$aliases.Condition1}}"
      Tests:
          - GroupId: A
          - GroupId: B
```

```
    - Condition: "{{$aliases.Condition2}}"
      Tests:
          - GroupId: D
    - Condition: "{{$aliases.Condition1}} || {{$aliases.Condition2}}"
      Tests:
          - GroupId: C
```

IDT 根據定義的條件選擇測試組，如下所示：

- 如果`Condition1`為 true，則 IDT 在測試組 A、B 和 C 中運行測試。
- 如果`Condition2`為 true，則 IDT 在測試組 C 和 D 中運行測試。

`Order`

選用。測試的執行順序。您可以在測試組級別指定測試順序。如果未指定此部分，IDT 將以隨機順序運行所有適用的測試組。的值`Order`是組描述符列表的列表。未列出的任何測試組`Order`，可以與任何其他列出的測試組 parallel 運行。

每個組描述符列表都包含其中一個組描述符，並標識運行在每個描述符中指定的組的順序。您可以使用下列格式來定義各個組描述符：

- *`group-id`*-現有測試組的組 ID。
- [*`group-id`*, *`group-id`*]-可以以相對於彼此的任意順序運行的測試組列表。
- `"*"`-通配符。這等效於尚未在當前組描述符列表中指定的所有測試組的列表。

的值`Order`還必須符合下列需求：

- 在組描述符中指定的測試組 ID 必須存在於測試套件中。
- 每個組描述符列表必須至少包含一個測試組。
- 每個組描述符列表必須包含唯一的組 ID。不能在單個組描述符中重複測試組 ID。
- 一個組描述符列表最多只能包含一個通配符組描述符。通配符組描述符必須是列表中的第一個或最後一個項。

Example 範例

對於包含測試組 A、B、C、D 和 E 的測試套件，以下示例列表顯示了指定 IDT 應先運行測試組 A，然後運行測試組 B，然後按任意順序運行測試組 C、D 和 E 的不同方法。

-
```
Order:
    - - A
      - B
      - [C, D, E]
```

- ```
  Order:
      - - A
        - B
        - "*"
  ```
- ```
  Order:
      - - A
        - B

      - - B
        - C

      - - B
        - D

      - - B
        - E
  ```

Features

選用。您希望 IDT 添加到awsiotdevicetester_report.xmlfile. 如果您未指定此部分，IDT 將不會向報告中添加任何商品功能。

產品功能是用户定義的有關設備可能滿足的特定條件的信息。例如，MQTT 產品功能可以指定設備正確發佈 MQTT 消息。Inawsiotdevicetester_report.xml，則產品功能設置為supported、not-supported或自定義的用户定義值，具體取決於是否通過指定的測試。

中的每個項目Features包含下列參數：

Name

功能的名稱。

Value

選用。您想在報表中使用的自定義值，而不是supported。如果未指定此值，則基於 IDT 的要素值設置為supported或者not-supported基於測試結果。如果使用不同條件測試同一要素，則可以在Features列表中，IDT 將支持條件的要素值串聯起來。如需詳細資訊，請參閱「」

Condition

判斷值為布林值值。如果評估值為 true，則 IDT 在完成測試套件運行後將該功能添加到測試報告中。如果評估值為 false，則該檢驗不包括在報告中。

`Tests`

選用。測試描述符的列表。必須通過此列表中指定的所有測試，才能支持該功能。

此列表中的每個測試描述符都使用測試組 ID 和一個或多個測試用例 ID 來標識要從特定測試組運行的單個測試。測試描述符使用下列格式：

```
GroupId: group-id
CaseIds: [test-id, test-id] # optional
```

您必須擇一指定`Tests`或者`OneOfTests`中的每個要素`Features`清單。

`OneOfTests`

選用。測試描述符的列表。必須至少通過此列表中指定的測試之一，才能支持該功能。

此列表中的每個測試描述符都使用測試組 ID 和一個或多個測試用例 ID 來標識要從特定測試組運行的單個測試。測試描述符使用下列格式：

```
GroupId: group-id
CaseIds: [test-id, test-id] # optional
```

您必須擇一指定`Tests`或者`OneOfTests`中的每個要素`Features`清單。

`IsRequired`

用於定義測試報告中是否需要該功能的布爾值。預設值為 `false`。

Example

測試協調流程式內容

測試編排器上下文是一個只讀的 JSON 文檔，其中包含測試編排器在執行過程中可用的數據。測試編排器上下文僅可從測試編排器訪問，幷包含確定測試流程的信息。例如，您可以使用測試運行者在`userdata.json`文件來確定是否需要運行特定測試。

測試協調流程式內容使用下列格式：

```
{
    "pool": {
        <device-json-pool-element>
    },
```

```
    "userData": {
        <userdata-json-content>
    },
    "config": {
        <config-json-content>
    }
}
```

`pool`

有關為測試運行選擇的設備池的信息。對於選定的設備池，此信息將從`device.json`file.

`userData`

中的資訊`userdata.json`file.

`config`

中的資訊`config.json`file.

您可以使用 JSONPath 表示法查詢上下文。狀態定義中的 JSONPath 查詢的語法是{{*query*}}。當您從測試 Orchestrator 上下文訪問數據時，請確保每個值計算為字符串、數字或布林值。

如需使用 JSONPath 符號來訪問內容資料的詳細資訊，請參使用 IDT 上下文。

配置 IDT 狀態機

> **Important**
>
> 從 IDT v4.5.1 開始，此狀態機已棄用。強烈建議您使用新的測試編排器。如需詳細資訊，請參閱 配置 IDT 測試編排器。

狀態機是控制測試套件執行流程的構造。它確定測試套件的起始狀態，根據用户定義的規則管理狀態轉換，並繼續在這些狀態之間進行轉換，直到它達到結束狀態。

如果您的測試套件不包含用户定義的狀態機，IDT 將為您生成一個狀態機。默認狀態機執行以下功能：

- 為測試運行者提供選擇和運行特定測試組的能力，而不是整個測試套件。
- 如果未選擇特定測試組，則以隨機順序運行測試套件中的每個測試組。
- 生成報告並打印顯示每個測試組和測試用例的測試結果的控制台摘要。

IDT 測試套件的狀態機器必須滿足下列條件：

- 每個狀態對應於 IDT 要執行的操作，例如運行測試組或產品報告文件。
- 轉換到狀態將執行與狀態關聯的操作。
- 每個狀態都定義下一個狀態的過渡規則。
- 結束狀態必須為`Succeed`或者`Fail`。

狀態機器格式

您可以使用以下模板配置自己的*<custom-test-suite-folder>*/suite/state_machine.json文件:

```
{
  "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 開始運行測試套件的名稱。值`StartAt`必須設定為`States`物件。

`States`

將用户定義的狀態名稱映射到有效 IDT 狀態的對象。每個國家.###對象包含映射到###。

所以此`States`對象必須包含`Succeed`和`Fail`狀態。如需有效狀態的資訊，請參[有效的狀態和狀態定義](#)。

有效的狀態和狀態定義

本節介紹 IDT 狀態機中可以使用的所有有效狀態的狀態定義。以下某些狀態支持測試用例級別的配置。但是，除非絕對必要，否則我們建議您在測試組級別而不是在測試用例級別配置狀態轉換規則。

狀態定義

- [RunTask](#)
- [Choice](#)
- [平行](#)
- [添加產品功能](#)
- [報告](#)
- [日誌消息](#)
- [選擇組](#)
- [Fail](#)
- [Succeed](#)

RunTask

所以此`RunTask`狀態會從測試套件中定義的測試組中運行測試用例。

```
{
    "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`

在當前狀態下執行動作後，為狀態轉換至的名稱。

`TestGroup`

選用。要運行的測試組的 ID。如果未指定此值，則 IDT 運行測試運行者選擇的測試組。

`TestCases`

選用。測試用例 ID 的數組，來自`TestGroup`。基於`TestGroup`和`TestCases`，IDT 確定測試執行行為，如下所示：

- 當兩者`TestGroup`和`TestCases`時，IDT 將從測試組運行指定的測試用例。
- 時機`TestCases`的規則，但`TestGroup`，則 IDT 運行指定的測試用例。
- 時機`TestGroup`被指定，但`TestCases`，IDT 會執行指定測試組中的所有測試用例。
- 當兩者都沒有`TestGroup`或者`TestCases`時，IDT 運行測試運行者從 IDT CLI 中選擇的測試組中的所有測試用例。要為測試運行者啟用組選擇，必須同時包含`RunTask`和`Choice`狀態`state_machine.json`file. 如需此操作如何執行的範例，請參。範例狀態機器：運行用户選定的測試組。

  如需為測試運行者啟用 IDT CLI 命令的詳細資訊，請參the section called “啟用 IDT”。

`ResultVar`

要使用測試運行結果設置的上下文變量的名稱。如果未指定`TestGroup`。IDT 設置您在`ResultVar`至`true`或者`false`根據下列的規則：

- 如果變量名稱為*`text`*_*`text`*_passed，則該值將設置為第一個測試組中的所有測試是通過還是被跳過。
- 在所有其他情況下，該值都設置為所有測試組中的所有測試都通過還是跳過。

一般而言，您需要使用`RunTask`狀態指定測試組 ID，而不指定單個測試用例 ID，以便 IDT 將運行指定測試組中的所有測試用例。由此狀態運行的所有測試用例以隨機順序 parallel 運行。但是，如果所有測試用例都要求設備運行，並且只有一個設備可用，則測試用例將按順序運行。

錯誤處理

如果任何指定的測試組或測試用例 ID 無效，則此狀態會發出`RunTaskError`執行錯誤。如果狀態遇到執行錯誤，則它還會設置`hasExecutionError`變量設置為`true`。

Choice

所以此Choice狀態允許您根據用户定義的條件動態設置要轉換到的下一個狀態。

```
{
    "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`

如果未採用Choices可以評估為true。

`FallthroughOnError`

選用。指定狀態在計算表達式時遇到錯誤時的行為。設定為true如果您想在評估導致錯誤時跳過表達式。如果未匹配任何表達式，則狀態機器會轉換到Default狀態。如果FallthroughOnError值，則預設為false。

`Choices`

一個表達式和狀態數組，用於確定在當前狀態下執行操作後轉換到哪個狀態。

`Choices.Expression`

求值的表達式字符串。如果表達式的計算結果為true，則狀態機器會轉換到Choices.Next。表達式字符串從狀態機上下文中檢索值，然後對它們執行操作以獲得布爾值。有關訪問狀態機上下文的信息，請參閱狀態機器上下文。

`Choices.Next`

如果表達式在Choices.Expression評估結果為true。

錯誤處理

所以此Choice狀態可能會需要在下列情況下進行錯誤處理：

- 選擇表達式中的某些變量在狀態機上下文中不存在。
- 表達式的結果不是布爾值。
- JSON 查找的結果不是字符串、數字或布爾值。

您無法使用Catch塊來處理此狀態下的錯誤。如果要在遇到錯誤時停止執行狀態機，則必須將FallthroughOnError至false。不過，建議您設定FallthroughOnError至true，並根據您的用例，執行下列其中一項：

- 如果您正在訪問的變量在某些情況下預期不存在，則使用Default和額外Choices塊來指定下一個狀態。
- 如果您正在訪問的變量應該始終存在，則設置Default狀態Fail。

平行

所以此Parallel狀態允許您彼此並行定義和運 parallel 新狀態機。

```
{
    "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

在當前狀態下執行動作後，為狀態轉換至的名稱。

Branches

要運行的狀態機定義數組。每個狀態機器定義都必須包含其自己的StartAt、Succeed，以及Fail狀態。此數組中的狀態機定義不能引用其自身定義之外的狀態。

> **Note**
>
> 由於每個分支狀態機共享相同的狀態機上下文，所以在一個分支中設置變量，然後從另一個分支讀取這些變量可能會導致意外的行為。

所以此Parallel狀態只有在運行所有分支狀態計算機後才會移動到下一個狀態。需要設備的每個狀態都將等待運行，直到設備可用。如果有多個設備可用，則此狀態從多個組並行運 parallel 測試用例。如果沒有足夠的設備，則測試用例將按順序運行。由於測試用例在並行運行時以隨機順序運 parallel，因此可能會使用不同的設備從同一測試組運行測試。

錯誤處理

確保分支狀態機和父狀態機都轉換到Fail狀態來處理執行錯誤。

由於分支狀態計算機不會將執行錯誤傳輸到父狀態機，因此不能使用Catch塊來處理分支狀態計算機中的執行錯誤。而是使用hasExecutionErrors值在共享狀態機上下文中。如需此操作如何執行的範例，請參。範例狀態機器：parallel 行運行兩個測試組。

添加產品功能

所以此AddProductFeatures狀態允許您將產品功能添加到awsiotdevicetester_report.xml文件由 IDT 生成。

產品功能是用户定義的有關設備可能滿足的特定條件的信息。例如，MQTT產品功能可以指定設備正確發佈 MQTT 消息。在報告中，產品功能設置為supported、not-supported或自定義值，具體取決於是否通過指定的測試。

> **Note**
>
> 所以此AddProductFeatures狀態不會自行生成報告。此狀態必須轉換為Reportstate以生成報告。

```
{
    "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`

在當前狀態下執行動作後，為狀態轉換至的名稱。

`Features`

一組產品功能顯示在`awsiotdevicetester_report.xml`file.

`Feature`

功能的名稱

`FeatureValue`

選用。要在報表中使用的自定義值，而不是`supported`。如果未指定此值，則根據測試結果，將要素值設置為`supported`或者`not-supported`。

如果您將自定義值用於`FeatureValue`，則可以在不同條件下測試同一要素，並且 IDT 連接受支持條件的特徵值。例如，下列摘錄顯示`MyFeature`功能，具有兩個獨立的要素值：

```
...
{
    "Feature": "MyFeature",
    "FeatureValue": "first-feature-supported",
    "Groups": ["first-feature-group"]
},
{
    "Feature": "MyFeature",
    "FeatureValue": "second-feature-supported",
    "Groups": ["second-feature-group"]
```

```
},
...
```

如果兩個測試組均通過，則要素值設置為first-feature-supported, second-feature-supported。

Groups

選用。測試組 ID 陣列。每個指定測試組中的所有測試都必須通過，才能支持該功能。

OneOfGroups

選用。測試組 ID 陣列。至少一個指定測試組中的所有測試必須通過，才能支持該功能。

TestCases

選用。測試用例 ID 陣列。如果指定此值，則以下內容適用：

- 必須通過所有指定的測試用例，才能支持該功能。
- Groups必須僅包含一個測試組 ID。
- OneOfGroups必須指定。

IsRequired

選用。設定為false將此功能標記為報告中的可選功能。預設值為 true。

ExecutionMethods

選用。一個執行方法的數組，它們匹配protocol中指定的值。device.jsonfile. 如果指定了此值，則測試運行者必須指定protocol值，該值與此數組中的某個值匹配，以便在報告中包含該功能。如果未指定此值，則該功能將始終包含在報告中。

若要使用 AWS for WordPressAddProductFeatures狀態，則必須將ResultVar中的RunTask狀態設定為下列其中一個值：

- 如果您指定了單個測試用例 ID，則設置ResultVar至*group-id_test-id*_passed。
- 如果未指定單個測試用例 ID，則將ResultVar至*group-id*_passed。

所以此AddProductFeatures狀態會按以下方式檢查測試結果：

- 如果未指定任何測試用例 ID，則每個測試組的結果將根據*group-id*_passed變量在狀態機上下文中。

- 如果您確實指定了測試用例 ID，則每個測試的結果由*group-id_test-id*_passed變量在狀態機上下文中。

錯誤處理

如果在此狀態下提供的組 ID 不是有效的組 ID，則此狀態會導致AddProductFeaturesError執行錯誤。如果狀態遇到執行錯誤，則它還會設置hasExecutionErrors變量設置為true。

報告

所以此Report狀態會生成*suite-name*_Report.xml和awsiotdevicetester_report.xml檔案。此狀態還會將報告流式傳輸到控制台。

```
{
    "Type": "Report",
    "Next": "<state-name>"
}
```

如下所述，包含值的所有欄位皆為必要：

Next

在當前狀態下執行動作後，為狀態轉換至的名稱。

您應始終轉換到Report狀態，以便測試運行者可以查看測試結果。通常情況下，此狀態後的下一個狀態是Succeed。

錯誤處理

如果此狀態在生成報告時遇到問題，則會發出ReportError執行錯誤。

日誌消息

所以此LogMessage狀態會生成test_manager.log並將日誌訊息流式傳輸到主控台。

```
{
    "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`

在當前狀態下執行動作後，為狀態轉換至的名稱。

`Level`

創建日誌消息的錯誤級別。如果指定的級別無效，此狀態將生成一條錯誤消息並將其丟棄。

`Message`

要記錄的消息。

選擇組

所以此`SelectGroup`狀態會更新狀態機上下文以指示選擇哪些組。由此狀態設置的值由任何後續的`Choice`狀態。

```
{
    "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`

在當前狀態下執行動作後，為狀態轉換至的名稱。

`TestGroups`

將標記為選定的測試組的數組。對於此陣列中的每個測試組 ID，*group-id*_selected變量設置為`true`在情況下。請確保您提供有效的測試組 ID，因為 IDT 不會驗證指定的組是否存在。

Fail

所以此`Fail`狀態表示狀態機未正確執行。這是狀態機的結束狀態，每個狀態機定義都必須包含此狀態。

```
{
    "Type": "Fail"
}
```

Succeed

所以此Succeed狀態表示狀態機正確執行。這是狀態機的結束狀態，每個狀態機定義都必須包含此狀態。

```
{
    "Type": "Succeed"
}
```

狀態機器上下文

狀態機上下文是一個只讀的 JSON 文檔，其中包含在執行期間可供狀態機使用的數據。狀態機上下文只能從狀態機訪問，幷包含確定測試流的信息。例如，您可以使用測試運行者在userdata.json文件來確定是否需要運行特定測試。

狀態機器上下文使用下列格式：

```
{
    "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`

有關為測試運行選擇的設備池的信息。對於選定的設備池，此信息將從`device.json`file.

`userData`

中的資訊`userdata.json`file.

`config`

信息固定`config.json`file.

`suiteFailed`

值設定為`false`狀態機器啟動時。如果測試組在`RunTask`狀態，則此值將設置為`true`，用於狀態機器執行的剩餘持續時間。

`specificTestGroups`

如果測試運行者選擇要運行的特定測試組而不是整個測試套件，則會創建此項並包含特定測試組 ID 的列表。

`specificTestCases`

如果測試運行者選擇要運行的特定測試用例而不是整個測試套件，則會創建此密鑰並包含特定測試用例 ID 的列表。

`hasExecutionErrors`

狀態機啟動時不退出。如果任何狀態遇到執行錯誤，則會創建此變量並將其設置為`true`，用於狀態機器執行的剩餘持續時間。

您可以使用 JSONPath 表示法查詢上下文。狀態定義中的 JSONPath 查詢的語法是`{{$.query}}`。您可以在某些狀態中使用 JSONPath 查詢作為佔位符字符串。IDT 將佔位符字符串替換為上下文中評估的 JSONPath 查詢的值。您可以使用佔位符作為下列值：

- 所以此`TestCases`中的值`RunTask`狀態。
- 所以此`Expression`值`Choice`狀態。

當您從狀態機器上下文訪問資料時，請確保滿足下列條件：

- JSON 路徑必須以`$.`
- 每個值都必須計算為字符串、數字或布爾值。

如需使用 JSONPath 符號來訪問上下文中的資料的詳細資訊，請參。使用 IDT 上下文。

執行錯誤

執行錯誤是狀態機在執行狀態時遇到的狀態機定義中的錯誤。IDT 將有關每個錯誤的信息記錄在`test_manager.log`並將日誌訊息流式傳輸到主控台。

您可以使用下列方式來處理執行錯誤：

- 新增`Catch`塊在狀態定義中。
- 檢查`hasExecutionErrors`值在狀態機器上下文中。

捕獲

使用`Catch`中，將以下內容添加到狀態定義中：

```
"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 catch 的錯誤類型陣列。如果執行錯誤與指定的其中一個值匹配，則狀態機器會轉換到`Catch.Next`。有關其產生的錯誤類型的信息，請參閱每個狀態定義。

`Catch.Next`

如果當前狀態遇到與中指定的值之一匹配的執行錯誤，則轉換為的下一個狀態`Catch.ErrorEquals`。

Catch 塊按順序處理，直到一個匹配。如果沒有錯誤與 Catch 塊中列出的錯誤匹配，則狀態計算機將繼續執行。由於執行錯誤是由不正確的狀態定義導致的，因此建議您在狀態遇到執行錯誤時轉換為「失敗」狀態。

哈希執行錯誤

當某些狀態遇到執行錯誤時，除了發出錯誤之外，它們還會設置hasExecutionError值設定為true在狀態機器上下文中。您可以使用此值檢測錯誤何時發生，然後使用Choice狀態將狀態機轉換為Fail狀態。

此方法具有下列特性：

- 狀態機不以分配給hasExecutionError，並且在特定狀態設置之前，此值不可用。這表示您必須明確設定FallthroughOnError至false(針對)Choice狀態訪問此值，以防止狀態機在沒有發生執行錯誤時停止。
- 一旦它被設置為true、hasExecutionError永遠不會設置為 false 或從上下文中刪除。這意味着此值僅在第一次設置為true，並且對於所有後續狀態，它不提供有意義的值。
- 所以此hasExecutionError值與所有分支狀態計算機共享Parallel狀態，這可能會導致意外的結果，具體取決於訪問它的順序。

由於這些特性，如果您可以使用 Catch 塊，我們不建議您使用此方法。

範例狀態機器

本節將提供狀態機器配置範例。

範例

- 範例狀態機器：運行單個測試組
- 範例狀態機器：運行用户選定的測試組
- 範例狀態機器：運行具有產品功能的單個測試組
- 範例狀態機器：parallel 行運行兩個測試組

範例狀態機器：運行單個測試組

此狀態機器：

- 運行 ID 的測試組GroupA，它必須出現在group.jsonfile.
- 檢查執行錯誤和轉換到Fail（如果找到任何）。
- 生成報告並過渡到Succeed如果沒有錯誤，Fail否則為。

```
{
```

```
    "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"
        }
    }
}
```

範例狀態機器：運行用户選定的測試組

此狀態機器：

- 檢查測試運行者是否選擇了特定的測試組。狀態機不檢查特定測試用例，因為測試運行者不同時選擇測試組就無法選擇測試用例。

- 如果選定了測試組：
  - 在選定測試組中運行測試用例。為此，狀態機不會在RunTask狀態。
  - 運行所有測試並退出後生成報告。
- 如果未選擇測試組：
  - 在測試組中運行測試GroupA。
  - 生成報告並退出。

```
{
    "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"
        }
    }
}
```

範例狀態機器：運行具有產品功能的單個測試組

此狀態機器：

- 執行測試組GroupA。
- 檢查執行錯誤和轉換到Fail（如果找到任何）。
- 新增FeatureThatDependsOnGroupA功能的awsiotdevicetester_report.xml文件:
  - 如果GroupA通道，則要素會設定為supported。
  - 該功能在報告中未標記為可選。
- 生成報告並過渡到Succeed如果沒有錯誤，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"
        }
    }
}
```

範例狀態機器：parallel 行運行兩個測試組

此狀態機器：

- 執行GroupA和GroupB測試組並 parallel。所以此ResultVar變量存儲在上下文中的RunTask狀態可用於AddProductFeatures狀態。
- 檢查執行錯誤和轉換到Fail（如果找到任何）。此狀態機不使用Catch塊，因為該方法不檢測分支狀態計算機中的執行錯誤。
- 將要素添加到awsiotdevicetester_report.xml文件基於傳遞
  - 如果GroupA通道，則要素會設定為supported。
  - 該功能在報告中未標記為可選。
- 生成報告並過渡到Succeed如果沒有錯誤，Fail否則

如果在設備池中配置了兩個設備，則GroupA和GroupB可以同時執行。但是，如果GroupA或者GroupB有多個測試，那麼兩個設備都可以分配給這些測試。如果只配置了一個設備，測試組將按順序運行。

```
{
    "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"
        }
    }
}
```

創建 IDT 測試用例可執行文件

您可以通過以下方式在測試套件文件夾中創建並放置測試用例可執行文件：

- 對於使用文件中的參數或環境變量來確定要運行哪些測試的測試套`test.json`件，您可以為整個測試套件創建單個測試用例可執行文件，或為測試套件中的每個測試組創建一個測試可執行文件。
- 對於要根據指定命令運行特定測試的測試套件，您可以為測試套件中的每個測試用例創建一個可執行文件。

作為測試編寫者，您可以確定哪種方法適合您的用例，並相應地構建測試用例可執行文件。請確定您在每個`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是否正確執行。

當所有設備都準備好運行測試用例時，IDT 會讀取以下文件：

- 所選測試案例的決定要啟動的程序以及要設定的環境變數。`test.json`
- 測試套件的決定要設定的環境變數。`suite.json`

IDT 會根據`test.json`檔案中指定的命令和引數啟動所需的測試可執行程序，並將必要的環境變數傳遞至處理序。

使用 IDT 用戶端開發套件

IDT 客戶端 SDK 使您可以使用 API 命令簡化在測試可執行文件中編寫測試邏輯的方式，這些命令可以與 IDT 和受測設備進行交互。IDT 目前提供下列軟體開發套件：

- 適用於 Python
- 適用 SDK for Go

- SDK for Java T

這些 SDK 位於*<device-tester-extract-location>*/sdks資料夾中。當您創建新的測試用例可執行文件時，必須將要使用的 SDK 複製到包含測試用例可執行文件的文件夾中，並在代碼中引用 SDK。本節提供了可用於測試用例可執行文件中的可用 API 命令的簡要說明。

本節內容

裝置互動

以下命令使您可以與被測設備進行通信，而無需實現任何其他設備交互和連接管理功能。

`ExecuteOnDevice`

允許測試套件在支持 SSH 或 Docker 外殼連接的設備上運行 shell 命令。

`CopyToDevice`

允許測試套件將本地文件從運行 IDT 的主機複製到支持 SSH 或 Docker shell 連接的設備上的指定位置。

`ReadFromDevice`

允許測試套件從支持 UART 連接的設備的串行端口讀取。

> **Note**
>
> 由於 IDT 不管理與使用內容中的設備訪問信息進行的設備的直接連接，因此我們建議您在測試用例可執行文件中使用這些設備交互 API 命令。但是，如果這些命令不符合您的測試用例要求，則可以從 IDT 上下文中檢索設備訪問信息，並使用它從測試套件直接連接到設備。
> 若要建立直接連線，請分別擷取待測裝置的`device.connectivity`和`resource.devices.connectivity`欄位中的資訊，以及資源裝置的資訊。如需有關使用 IDT 前後關使用的詳細資訊，請參閱使用 IDT 上下文。

IDT 交互作用

以下命令使您的測試套件能夠與 IDT 進行通信。

`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 上下文中檢索值。如需詳細資訊，請參閱使用 IDT 上下文。

`SendResult`

允許測試套件向 IDT 報告測試用例結果。必須在測試套件中的每個測試用例結束時調用此命令。

主機互動

以下命令使您的測試套件能夠與主機進行通信。

`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 上下文中檢索值。如需詳細資訊，請參閱使用 IDT 上下文。

`ExecuteOnHost`

允許測試套件在本地計算機上運行命令，並允許 IDT 管理測試用例可執行生命週期。

啟用 IDT

命`run-suite`令 IDT CLI 提供了幾個選項，讓測試運行器自定義測試執行。為了允許測試運行程序使用這些選項來運行自定義測試套件，請實現對 IDT CLI 的支持。如果您沒有實現支持，測試運行程序仍然可以運行測試，但是某些 CLI 選項將無法正常工作。為了提供理想的客戶體驗，我們建議您在 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，此值將在執行測試時套用至所有逾時。

測試運行者可以使用此引數來增加他們想要運行的測試用例的超時時間。當測試運行程序在其run-suite命令中指定此引數時，IDT 會使用它來計算 IDT_TEST_TIMEOUT 環境變量的值，並在 IDT 上下文中設置該config.timeoutMultiplier字段。若要支援此引數，您必須執行下列動作：

- 而不是直接使用test.json檔案中的逾時值，而是讀取 IDT_TEST_TIMEOUT 環境變數以取得正確計算的逾時值。
- 從 IDT 內容擷取config.timeoutMultiplier值，並將其套用至長時間執行逾時。

如需因逾時事件而提前退出的詳細資訊，請參閱指定退出行為。

stop-on-first-failure

指定 IDT 遇到失敗時應停止執行所有測試。

當測試運行程序在其run-suite命令中指定此參數時，IDT 將在遇到故障時立即停止運行測試。但是，如果測試用例並行運 parallel，那麼這可能會導致意外的結果。要實現支持，請確保如果 IDT 遇到此事件，則測試邏輯會指示所有正在運行的測試用例停止，清理臨時資源並向 IDT 報告測試結果。如需有關使用失敗的詳細資訊，請參閱指定退出行為。

group-id 和 test-id

指定 IDT 應該只執行選取的測試群組或測試案例。

測試運行者可以將這些引數與其run-suite命令一起使用，以指定以下測試執行行為：

- 運行指定的測試組內的所有測試。
- 從指定的測試組中運行選擇的測試。

為了支持這些參數，測試套件的測試協調器必須在測試協調器中包含一組特定的RunTask和Choice狀態。如果您不使用自訂狀態機器，則預設的 IDT 測試協調器會為您包含必要的狀態，而且您不需要採取其他動作。但是，如果您使用自定義測試協調器，則將其用範例狀態機器：運行用户選定的測試組作示例，以在測試協調器中添加所需的狀態。

如需有關 IDT CLI 命令的詳細資訊，請參閱偵錯並執行自訂測試套件。

寫入事件記錄

在測試運行時，您將數據發送stderr到stdout並將事件日誌和錯誤消息寫入控制台。如需有關主控台訊息格式的詳細資訊，請參閱控制台訊息格式。

當 IDT 完成執行測試套件時，資*<devicetester-extract-location>*/results/*<execution-id>*/logs料夾中的test_manager.log檔案也會提供此資訊。

您可以配置每個測試用例以將日誌從其測試運行（包括來自被測設備的日誌）寫入*<group-id>*_*<test-id>*文件*<device-tester-extract-location>*/results/*execution-id*/logs夾中的文件。若要這麼做，請使用testData.logFilePath查詢從 IDT 內容擷取記錄檔的路徑，在該路徑上建立檔案，然後將您想要的內容寫入。IDT 會根據正在執行的測試案例自動更新路徑。如果您選擇不為測試用例創建日誌文件，則不會為該測試用例生成任何文件。

您也可以設定文字執行檔，視需要在*<device-tester-extract-location>*/logs資料夾中建立其他記錄檔。建議您為記錄檔名稱指定唯一的前置詞，這樣您的檔案就不會被覆寫。

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。這些報告檔案位在*<device-tester-extract-location>*/results/*<execution-id>*/。這兩個報告都會捕獲測試套件執行的結果。如需 IDT 用於這些報告之結構描述的詳細資訊，請參閱查看 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指令將測試結果報告給 IDT。如果 IDT 找不到測試結果，則會針對測試用例發出錯誤。以下 Python 摘錄顯示了將測試結果發送到 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。此資料夾的路徑儲存在 IDT 前後關聯中的testData.testArtifactsPath欄位中。在此文件夾中，IDT 使用它定位的第一個按字母順序排序的 XML 文件作為測試結果。

如果您的測試邏輯產生 JUnit XML 結果，則可以將測試結果寫入成品文件夾中的 XML 文件，以直接將結果提供給 IDT，而不是解析結果，然後使用 API 將其提交給 IDT。

如果您使用此方法，請確保您的測試邏輯準確地總結測試結果，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，但下列例外：

- IDT 會忽略testsuites標籤的所有性質。相反地，它會從其他報告的測試群組結果中計算標籤屬性。
- 中必須至少有一個testsuite標籤存在testsuites。

由於 IDT 對所有測試用例使用相同的工件文件夾，並且在測試運行之間不會刪除結果文件，因此如果 IDT 讀取不正確的文件，此方法也可能導致錯誤的報告。我們建議您在所有測試用例中為生成的 XML

結果文件使用相同的名稱，以覆蓋每個測試用例的結果，並確保 IDT 可以使用正確的結果。儘管您可以使用混合方法在測試套件中進行報告，也就是說，對某些測試用例使用 XML 結果文件，並通過 API 為其他測試用例提交結果，但我們不建議使用此方法。

指定退出行為

將您的文字可執行檔設定為永遠以 0 結束代碼結束，即使測試案例報告失敗或錯誤結果也是如此。僅使用非零退出代碼來指示測試用例未運行，或者測試用例可執行文件無法向 IDT 傳達任何結果。當 IDT 收到非零退出代碼時，它標誌著測試用例遇到了阻止其運行的錯誤。

IDT 可能會要求或預期測試用例在下列事件中完成之前停止執行。使用此信息配置測試用例可執行文件，以檢測測試用例中的每個事件：

Timeout (逾時)

當測試用例運行時超過`test.json`文件中指定的超時值時發生。如果測試執行程式使用`timeout-multiplier`引數來指定逾時乘數，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。當測試運行程序啟動測試時，IDT 將 IDT_TEST_TIMEOUT 環境變量的值設置為計算的超時值（以秒為單位），並將該變量傳遞給測試用例可執行文件。您可以讀取變數值來設定適當的計時器。

中斷

當測試運行器中斷 IDT 時發生。例如，按下Ctrl+C。

由於終端機將信號傳播到所有子進程，因此您只需在測試用例中配置信號處理程序即可檢測中斷信號。

或者，您可以定期輪詢 API 以檢查`PollForNotifications` API 回應中的`CancellationRequested`布林值。當 IDT 接收到中斷信號，它將`CancellationRequested`布爾值設置為`true`。

第一次失敗時停止

當與當前測試用例並行運 parallel 的測試用例失敗，並且測試運行程序使用`stop-on-first-failure`參數來指定 IDT 在遇到任何故障時應停止時發生。

若要偵測此事件，您可以定期輪詢 API，以檢查`PollForNotifications` API 回應中的`CancellationRequested`布林值。當 IDT 遇到失敗並配置為在第一次失敗時停止時，它將`CancellationRequested`布林值設置為`true`。

當發生任何這些事件時，IDT 會等待 5 分鐘，讓任何當前正在運行的測試用例完成運行。如果所有正在運行的測試用例都未在 5 分鐘內退出，則 IDT 會強制其每個進程停止。如果 IDT 在進程結束之前沒有收到測試結果，則會將測試用例標記為已超時。作為最佳實踐，您應該確保您的測試用例遇到其中一個事件時執行以下操作：

1. 停止運行正常測試邏輯。
2. 清理所有臨時資源，例如被測設備上的測試工件。
3. 向 IDT 報告測試結果，例如測試失敗或錯誤。
4. 退出。

使用 IDT 上下文

當 IDT 運行測試套件時，測試套件可以訪問一組數據，這些數據可用於確定每個測試的運行方式。此數據稱為 IDT 上下文。例如，測試運行者在`userdata.json`文件可用於在 IDT 上下文中測試套件。

IDT 上下文可以被視為只讀 JSON 文檔。測試套件可以使用標準 JSON 數據類型（如對象、數組、數字等）從上下文中檢索數據並將數據寫入上下文。

上下文架構

IDT 上下文使用下列格式：

```
{
    "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文件。所以此config字段還包含以下附加字段：

`config.timeoutMultiplier`

測試套件使用的任何超時值的乘數。此值由 IDT CLI 中的測試運行者指定。預設值為 1。

`device`

有關為測試運行選擇的設備的信息。此信息相當於devices數組元素device.json文件為選定的設備。

`devicePool`

有關為測試運行選擇的設備池的信息。此信息等效於頂級設備池陣列元素，在device.json文件中的所選設備池。

`resource`

有關資源設備的信息resource.jsonfile.

`resource.devices`

此信息相當於devices陣列中定義resource.jsonfile. EACHdevices元素包括下列附加欄位：

`resource.device.name`

資源裝置的名稱。此值設定為requiredResource.name中的值test.jsonfile.

`testData.awsCredentials`

所以此AWS憑據用於連接到AWS雲端。這些信息是從config.jsonfile.

`testData.logFilePath`

測試用例向其寫入日誌消息的日誌文件的路徑。如果此檔案不存在，則測試套件會創建此檔案。

`userData`

測試運行者在[userdata.json文件]。

訪問上下文中的數據

您可以使用 JSON 文件中的 JSONPath 符號來查詢上下文，也可以從文本可執行文件中使用`GetContextValue`和`GetContextString`API。用於訪問 IDT 上下文的 JSONPath 字符串的語法變化如下：

- 在`suite.json`和`test.json`，您使用`{{`*query*`}}`。也就是說，不要使用根元素`$.`開始表達式。
- 在`test_orchestrator.yaml`，您使用`{{`*query*`}}`。

  如果您使用已棄用的狀態機，則在`state_machine.json`，您使用`{{$.`*query*`}}`。
- 在 API 命令中，您可以使用*query*或者`{{$.`*query*`}}`，具體取決於命令。如需詳細資訊，請參閱軟體開發套件中的內聯文件。

下表描述典型 JSONPath 表達式中的運算符：

Operator	Description
`$`	The root element. Because the top-level context value for IDT is an object, you will typically use `$.` to start your queries.
`.##`	Accesses the child element with name `###` 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.AW ##` .

Operator	Description
`[##:##]`	Filters elements from an array, retrieving items beginning from the `start` index and going up to the ## index, both inclusive.
`[## 1### 2#...### N]`	Filters elements from an array, retrieving items from only the specified indices.
`[? (##)]`	Filters elements from an array using the `expr` expression. This expression must evaluate to a boolean value.

要創建篩選器表達式，請使用以下語法：

```
<jsonpath> | <value> operator <jsonpath> | <value>
```

在此語法中：

- `jsonpath`是一個使用標準 JSON 語法的 JSON 路徑。
- `value`是任何使用標準 JSON 語法的自定義值。
- `operator`是下列其中一項運算符：
  - < (小於)
  - <= (小於或等於)
  - == (等於)

    如果表達式中的 JSONPath 或值是數組、布爾值或對象值，則這是您可以使用的唯一受支持的二進制運算符。
  - >= (大於或等於)
  - > (大於)
  - =~ (正則表達式匹配)。要在過濾器表達式中使用此運算符，表達式左側的 JSONPath 或值必須計算為字符串，右側必須是一個跟隨RE2 語法。

您可以使用 {{*##*}} 作為佔位符字符串在`args`和`environmentVariables`欄位`test.json`文件和`environmentVariables`欄位`suite.json`檔案。IDT 執行上下文查找並使用查詢的評估值

填充字段。例如，在suite.json文件中，您可以使用佔位符字符串指定隨每個測試用例發生變化的環境變量值，IDT 將使用每個測試用例的正確值填充環境變量。但是，當您使用佔位符字符串test.json和suite.json檔案，下列考量適用於您的查詢：

- 您必須每次出現devicePool鍵在您的查詢中全部小寫。也就是說，使用devicepool要改。
- 對於數組，您只能使用字符串數組。此外，數組使用非標準item1, item2,...,itemN格式。如果數組只包含一個元素，那麼它被序列化為item，使其與字符串字段無法區分。
- 不能使用佔位符從上下文檢索對象。

由於這些考慮因素，我們建議儘可能使用 API 訪問測試邏輯中的上下文，而不是test.json和suite.json檔案。但是，在某些情況下，使用 JSONPath 佔位符來檢索要設置為環境變量的單個字符串可能會更方便。

設定測試執行程式的設定

要運行自定義測試套件，測試運行者必須根據要運行的測試套件配置其設置。設定是根據位於*<device-tester-extract-location>*/configs/資料夾中的組態檔案樣板來指定。如果需要，測試運行程序還必須設置 IDT 將用於連接到AWS雲的AWS憑據。

作為測試編寫者，您將需要配置這些文件來調試您的測試套件。您必須提供測試運行程序的說明，以便他們可以根據需要配置以下設置以運行測試套件。

設定 device.json

該device.json文件包含有關運行測試的設備的信息（例如，IP 地址，登錄信息，操作系統和 CPU 架構）。

測試運行者可以使用位於該文件*<device-tester-extract-location>*/configs/夾中的以下模板device.json文件提供此信息。

```
[
    {
        "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 用於追蹤合格的裝置。

> **Note**
>
> 如果您想要在 AWS Partner 裝置目錄中列出您的電路板，在此處指定的 SKU 必須符合您在列名程序中使用的 SKU。

`features`

選用。包含裝置支援功能的陣列。裝置功能是您在測試套件中設定的使用者定義值。您必須向測試執行者提供有關要包含在`device.json`檔案中的功能名稱和值的資訊。例如，如果您要測試裝置作為其他裝置的 MQTT 伺服器運作，您可以設定測試邏輯，以驗證名為的功能的特定支援層級。`MQTT_QOS`測試運行者提供此功能名稱，並將功能值設置為其設備支持的 QOS 級別。您可以使用查詢從 IDT 內容中擷取提供的資訊，或從`devicePool.features`查詢的測試協調器內容中擷取提供的資訊。`pool.features`

`features.name`

特徵的名稱。

`features.value`

支援的特徵值。

`features.configs`

功能的組態設定 (如果需要)。

`features.config.name`

組態設定的名稱。

`features.config.value`

支援的設定值。

devices

池中要測試的一系列設備。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。池中的每個設備都必須使用相同的協議。

目前，唯一支援的值是ssh實uart體裝置和 docker Docker 容器。

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

用於驗證的燈入資料。

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 ssh 時，才會套用此屬性。

connectivity.containerUser

選用。容器內的使用者對使用者的名稱。預設值是 Docker 檔案中提供的使用者。

預設值為 22。

只有當 connectivity.protocol 設為 ssh 時，才會套用此屬性。

> **Note**
>
> 要檢查測試運行程序是否為測試配置了不正確的設備連接，您可以pool.Devices[0].Connectivity.Protocol從測試 Orchestrator 上下文中檢索並將其與Choice狀態中的預期值進行比較。如果使用了不正確的協議，則使用LogMessage狀態和轉換到狀態打印消息。Fail
> 或者，您可以使用錯誤處理代碼來報告錯誤設備類型的測試失敗。

(選擇性) 設定使用者資料 .json

該userdata.json文件包含測試套件所需但未在device.json文件中指定的任何其他信息。這個文件的格式取決於在測試套userdata_scheme.json件中定義的文件。如果您是測試編寫者，請確保將此信息提供給將運行您編寫的測試套件的用戶。

(選擇性) 設定資源 .json

該resource.json文件包含有關將用作資源設備的任何設備的信息。資源設備是測試被測設備的某些功能所需的設備。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您的裝置是否能成功連線。資源設備是可選的，您可以根據需要任意數量的資源設備。作為測試編寫者，您可以使用 test.json 文件來定義測試所需的資源設備功能。然後，測試運行程序使用該resource.json文件提供具有所需功能的資源設備池。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。

測試運行者可以使用位於該文件*<device-tester-extract-location>*/configs/夾中的以下模板resource.json文件提供此信息。

```
[
    {
        "id": "<pool-id>",
        "features": [
            {
                "name": "<feature-name>",
                "version": "<feature-version>",
                "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。此字段中所需的信息在測試套件的 test.json 文件中定義，並確定要運行哪些測試以及如何運行這些測試。如果測試套件不需要任何功能，則不需要此字段。

`features.name`

特徵的名稱。

`features.version`

功能版本。

`features.jobSlots`

設定以指出可同時使用裝置的測試數目。預設值為 1。

`devices`

池中要測試的一系列設備。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。池中的每個設備都必須使用相同的協議。

目前，唯一支援的值是ssh實uart體裝置和 docker Docker 容器。

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

用於驗證的燈入資料。

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 `ssh` 時，才會套用此屬性。

`connectivity.containerUser`

選用。容器內的使用者對使用者的名稱。預設值是 Docker 檔案中提供的使用者。

預設值為 22。

只有當 `connectivity.protocol` 設為 `ssh` 時，才會套用此屬性。

（可選）配置

`config.json`檔案包含 IDT 的組態資訊。通常，測試運行者不需要修改此文件，除非為 IDT 提供AWS用戶憑據以及可選AWS地區提供用戶憑據。如果提供具有必要權限的AWS認證，則AWS IoT Device Tester會收集使用狀況測量結果並提交給。AWS這是一項選擇加入功能，用於改善 IDT 功能。如需詳細資訊，請參閱 IDT。

測試運行者可以通過以下方式之一配置其AWS憑據：

- 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 工作階段關閉後，無法使用在 SSH 工作階段期間定義的變數。IDT 可以使用`AWS_ACCESS_KEY_ID`和`AWS_SECRET_ACCESS_KEY`環境變數來儲存AWS認證

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>
```

要配置 IDT 的AWS憑據，測試運行程序編輯文件*<device-tester-extract-location>*/configs/夾中config.json文件中的auth部分。

```
{
    "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 >* 來定義的。

`log.location`

< device-tester-extract-location > 中記錄檔資料夾的路徑。

`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`

包含測試套件之資料夾的路徑。

`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。測試套件將使用的AWS區域。如果未設置，則測試套件將使用每個測試套件中指定的默認區域。

`auth.method`

IDT 用來擷取AWS認證的方法。支援的值是`file`從認證檔案擷取認證，`environment`以及使用環境變數擷取認證。

`auth.credentials.profile`

要從身份證明檔案使用的身份證明設定檔。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

設置所需的配置後，IDT 可以運行您的測試套件。完整測試套件的運行時取決於硬件和測試套件的組合。作為參考，大約需要 30 分鐘才能完成 Raspberry Pi 3B 上的完整AWS IoT Greengrass資格測試套件。

在編寫測試套件時，可以使用 IDT 在調試模式下運行測試套件，以在運行代碼之前檢查代碼或將其提供給測試運行程序。

在除錯模式下執行 IDT

由於測試套件依賴 IDT 與設備交互，提供上下文並接收結果，因此您無法在沒有任何 IDT 交互的情況下簡單地在 IDE 中調試測試套件。為此，IDT CLI 提供了可讓您在除錯模式下執行 IDT 的`debug-test-suite`命令。執行下列命令以檢視下列項目的可用選項`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 實際上並不會啟動測試套件或執行測試協調程式；相反，它會與 IDE 互動，以回應從 IDE 中執行的測試套件發出的要求，並將記錄檔列印到主控台。IDT 不會逾時並等待結束，直到手動中斷為止。在偵錯模式下，IDT 也不會執行測試協調程式，也不會產生任何報告檔案。要調試您的測試套件，您必須使用 IDE 來提供 IDT 通常從配置 JSON 文件中獲取的一些信息。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`suite.json`讀取此資訊。
- 用於選取資源裝置的引數。IDT 不會從`test.json`中讀取此信息。

若要偵錯測試套件，請完成以下步驟：

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`，請確保您根據需要配置所有這些。
2. 執行下列命令，將 IDT 置於偵錯模式，並選取執行測試所需的任何裝置。

   ```
   devicetester_[linux | mac | win_x86-64] debug-test-suite [options]
   ```

   運行此命令後，IDT 會等待來自測試套件的請求，然後對其進行響應。IDT 也會產生 IDT 用戶端 SDK 案例程序所需的環境變數。
3. 在您的 IDE 中，使用`run`或`debug`組態來執行下列動作：

   a. 設定 IDT 產生的環境變數的值。

   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。

   您可以根據需要多次調試和重新運行測試套件。IDT 在除錯模式下不會逾時。
5. 完成除錯之後，請中斷 IDT 以結束偵錯模式。

用於運行測試的 IDT CLI 命令

下一節將說明 IDT CLI 指令：

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`

列出您 IDT 版本的支援產品、AWS IoT Greengrass本例中的版本，以及適用於目前 IDT 版本的 AWS IoT Greengrass限定測試套件版本。

`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`。 要運行的測試套件版本。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，以逗號分隔的清單形式。如果未指定，IDT 會執行測試套件中的所有測試群組。
- `test-id`。 要運行的測試用例，以逗號分隔的列表。指定時，`group-id` 必須指定單一群組。
- `pool-id`。 要測試的裝置集區。如果測試運行程序在您的`device.json`文件中定義了多個設備池，則必須指定一個池。
- `timeout-multiplier`。 設定 IDT 以修改`test.json`檔案中指定的測試執行逾時，以使用使用者定義的乘數進行測試。

- `stop-on-first-failure`。 設定 IDT 以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`。 設置包含運行測試套件所需的用戶數據信息的文件。只有在測試套件`userdataRequired`的`suite.json`文件中設置為 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`

在調試模式下運行測試套件。如需更多詳細資訊，請參閱 在除錯模式下執行 IDT。

查看 IDT 測試結果和日誌

本節介紹 IDT 生成控制台日誌和測試報告的格式。

控制台訊息格式

AWS IoT Device Tester使用標準格式在控制台啟動測試套件時將消息打印到控制台。IDT 生成的控制台訊息如下列的摘錄所示。

```
time="2000-01-02T03:04:05-07:00" level=info msg=Using suite: MyTestSuite_1.0.0
executionId=9a52f362-1227-11eb-86c9-8c8590419f30
```

大多數控制台訊息包含下列字段：

`time`

記錄事件的完整 ISO 8601 時間戳。

`level`

記錄事件的消息級別。通常，記錄的消息級別是`info`、`warn`, 或`error`。IDT 發出`fatal`或者`panic`消息，如果遇到導致其提前退出的預期事件。

`msg`

記錄的消息。

`executionId`

當前 IDT 進程的唯一 ID 字符串。此 ID 用於區分單個 IDT 運行。

從測試套件生成的控制台消息提供了有關被測設備以及 IDT 運行的測試套件、測試組和測試用例的其他信息。以下摘錄顯示了從測試套件生成的控制台消息的示例。

```
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
```

控制台消息的測試套件特定部分包含以下字段：

`suiteId`

當前正在執行的測試套件的名稱。

`groupId`

當前運行的測試組的 ID。

`testCaseId`

當前正在執行的測試案例的 ID。

`deviceId`

當前測試用例正在使用的被測設備的 ID。

要在 IDT 完成測試運行時將測試摘要打印到控制台，您必須包含`Reportstate`在您的測試編輯器中。測試摘要包含有關測試套件的信息、所運行的每個組的測試結果以及生成的日誌和報告文件的位置。測試摘要訊息如下列範例所示。

```
========== 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報告架構

`awsiotdevicetester_report.xml`是包含下列資訊的簽名報告：

- IDT 版本。
- 測試套件版本。
- 報告簽名和用於簽名報告的密鑰。
- 指定的設備 SKU 和`device.json`file.
- 已測試的產品版本和設備功能。
- 測試結果的彙總摘要。此信息與*suite-name*`_report.xml`file.

```
<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-contents>
    </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。標示為required是測試套件驗證設備所必需的。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>
```

標示為optional不需要驗證。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>
```

測試套件報告架構

*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。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>
```

報表部分在awsiotdevicetester_report.xml或者*suite-name*_report.xml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="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`

運行測試套件所花費的時間（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。

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

如果您提供具有必要權限的AWS認證，請AWS IoT Device Tester收集使用狀況測量結果並將其提交給AWS。這是一項選擇加入功能，用於改善 IDT 功能。IDT 會收集以下資訊：

- 用來執行 IDT 的AWS 帳戶識別碼
- 用於運行測試的 IDTAWS CLI 命令
- 正在運行的測試套件
- *<device-tester-extract-location >* 資料夾中的測試套件
- 裝置集區中設定的裝置數目
- 測試用例名稱和運行時間
- 測試結果資訊，例如測試是否通過、失敗、遇到錯誤或被略過
- 產品功能測試
- IDT 退出行為，例如意外或提前退出

IDT 傳送的所有資訊也會記錄到資*<device-tester-extract-location>*/`results/`*<execution-id>*`/`料夾中的`metrics.log`檔案。您可以檢視記錄檔，以查看測試執行期間收集的資訊。只有當您選擇收集使用狀況測量結果時，才會產生此檔案。

若要停用測量結果收集，您不需要採取其他動作。只需不要存儲您的AWS憑據，如果您確實有存儲的AWS憑據，則不要配置`config.json`文件以訪問它們。

設定 AWS 憑證

如果還沒有AWS 帳戶，您必須建立一個。如果您已經擁有AWS 帳戶，則只需為您的帳戶設定必要的權限，以便 IDT 代表您傳送使用量指標。AWS

步驟 1：建立 AWS 帳戶

在此步驟中，建立並設定AWS 帳戶. 如果您已經擁有AWS 帳戶，請跳至the section called “步驟 2：設定 IDT 的許可”。

如果您還沒有 AWS 帳戶，請完成下列步驟建立新帳戶。

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，並在電話鍵盤輸入驗證碼。

   註冊 AWS 帳戶 時，會建立 AWS 帳戶根使用者。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，將管理存取權指派給管理使用者，並且僅使用根使用者來執行需要根使用者存取權的任務。

若要建立管理員使用者，請選擇下列其中一個選項。

選擇一種管理管理員的方式	若要	By	您也可以
在 IAM Identity Center (建議)	使用短期憑證存取 AWS。 這與安全性最佳實務一致。有關最佳實務的資訊，請參閱 IAM 使用者指南中的 IAM 安全最佳實務。	請遵循 AWS IAM Identity Center 使用者指南的入門中的說明。	請參閱 AWS Command Line Interface 使用者指南中的設定 AWS CLI 以使用 AWS IAM Identity Center 設定程式設計存取。
在 IAM 中 (不建議使用)	使用長期憑證存取 AWS。	請遵循 IAM 使用者指南中建立您的第一個 IAM 管理員使用者和使用者群組的說明。	請參閱 IAM 使用者指南中的管理 IAM 使用者的存取金鑰，設定程式設計存取。

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 使用情況資料的權限。您可以使用AWS Management Console或AWS Command Line Interface (AWS CLI) 建立 IDT 的 IAM 政策和使用者，然後將政策附加到使用者。

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 Policies (政策)，然後選擇 Create policy (建立政策)。

   b. 在 JSON 標籤上，用以下政策取代預留位置內容。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot-device-tester:SendMetrics"
            ],
            "Resource": "*"
        }
    ]
}
```

   c. 選擇 Review policy (檢閱政策)。

   d. 對於 Name (名稱)，輸入 **`IDTUsageMetricsIAMPermissions`**。在 Summary (摘要) 下，檢閱您的政策所授與的許可。

   e. 選擇 Create policy (建立政策)。

3. 建立 IAM 使用者。

   a. 建立 IAM 使用者。遵循 IAM 使用者指南中建立 IAM 使用者 (主控台) 中的步驟 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 Set permissions (設定許可) 頁面上，選擇 Attach existing policies directly (直接連接現有的政策)。

      ii. 搜尋您在上一個步驟中建立的 IDTUsageMetrics Iam許可權政策。選取核取方塊。

c. 選擇 Next: Tags (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。您稍後使用來設定您的AWS身分。

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 AWS IoT Greengrass 的許可。

1. 如果尚未安裝 AWS CLI，請在電腦上安裝並設定。請按照《AWS Command Line Interface使用者指南》中的〈安裝〉AWS CLI中的步驟進行。

> **Note**
>
> AWS CLI是開放原始碼工具，可讓您在命令列 shell 中使用來與AWS服務互動。

2. 建立下列客戶管理政策，以授與管理 IDT 和AWS IoT Greengrass角色的權限。

   Linux 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。

PowerShell

```
aws iam create-policy --policy-name IDTUsageMetricsIAMPermissions --policy-
document '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot-device-tester:SendMetrics"
            ],
            "Resource": "*"
        }
    ]
}'
```

3. 建立 IAM 使用者，並連接 IDT 所需的許可授權AWS IoT Greengrass。

   a. 建立 IAM 使用者。

   ```
   aws iam create-user --user-name user-name
   ```

   b. 將您建立的IDTUsageMetricsIAMPermissions政策附加到 IAM 使用者。<account-id> 將#####取代為您的 IAM 使用者名稱，並在命令中使用您的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。您稍後使用來設定您的AWS身分。

向 IDT 提供AWS憑證

若要允許 IDT 存取您的AWS認證並提交量度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登入資料檔案中：

   a. 若要使用環境變數，請執行下列命令。

   Linux or Unix

   ```
   export AWS_ACCESS_KEY_ID=access-key
   export AWS_SECRET_ACCESS_KEY=secret-access-key
   ```

   Windows Command Prompt (CMD)

   ```
   set AWS_ACCESS_KEY_ID=access-key
   set AWS_SECRET_ACCESS_KEY=secret-access-key
   ```

   PowerShell

   ```
   $env:AWS_ACCESS_KEY_ID="access-key"
   $env:AWS_SECRET_ACCESS_KEY="secret-access-key"
   ```

   b. 若要使用認證檔案，請將下列資訊新增至~/.aws/credentials檔案。

   ```
   [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 （可選）配置。

疑難排解 IDTAWS IoT GreengrassV2

IDT 適用於AWS IoT GreengrassV2 根據錯誤類型將錯誤寫入不同的位置。IDT 會將錯誤寫入主控台、記錄檔和測試報告。

在哪裡尋找錯誤

測試運行時，控制台上會顯示高級錯誤，並在所有測試完成時顯示失敗測試的摘要。`awsiotdevicetester_report.xml`包含導致測試失敗的所有錯誤的摘要。IDT 將每次測試運行的日誌文件存儲在具有用於測試執行的 UUID 的目錄中，該目錄在測試運行期間顯示在控制台上。

IDT 測試日誌目錄是*<device-tester-extract-location>*`/results/`*<execution-id>*`/logs/`。此目錄包含表格中顯示的下列檔案。這適用於除錯。

檔案	描述
`test_manager.log`	測試運行時寫入控制台的日誌。此檔案結尾的結果摘要包含測試失敗的清單。 這個檔案中的警告和錯誤日誌提供有關失敗的一些資訊。
test-group-id `/`*test-case-id* `/`*test-name* `.log`	測試組中特定測試的詳細日誌。對於部署 Greengrass 組件的測試，測試用例日誌文件被調用`greengrass-test-run.log` 。
test-group-id `/`*test-case-id* `/greengrass.log`	詳細記錄AWS IoT Greengrass核心軟體。IDT 運行安裝的測試時，從被測設備複製此文件AWS IoT Greengrass設備上的核心軟件。如需有關此記錄檔中訊息的更多資訊，請參閱疑難排 AWS IoT Greengrass V2。
test-group-id `/`*test-case-id*`/`*component-name* `.log`	測試運行期間部署的 Greengrass 組件的詳細日誌。IDT 執行部署特定元件的測試時，會從被測裝置複製元件記錄檔。每個元件記錄檔的名稱都會對應至已部署元件的名稱。如需有關此記錄檔中訊息的更多資訊，請參閱疑難排 AWS IoT Greengrass V2。

解決 IDT 的解決方案AWS IoT GreengrassV2 錯誤

在您執行 IDT 之前AWS IoT Greengrass」中，取得正確的組態檔案。如果您收到剖析和組態錯誤，您的第一個步驟是尋找並使用適合您環境的組態範本。

如果您仍然有問題，請參閱下列除錯程序。

主題

別名解析錯誤

當您執行自訂測試套件時，您可能會在主控台和`test_manager.log`。

```
Couldn't resolve placeholders: couldn't do a json lookup: index out of range
```

當 IDT 測試協調器中設定的別名無法正確解析，或者如果解析的值不存在於組態檔案中，就會發生此錯誤。要解決此錯誤，請確保您的`device.json`和`userdata.json` 包含測試套件所需的正確信息。有關所需配置的信息AWS IoT Greengrass資格，請參閱[設定 IDT 設定以執行AWS IoT Greengrass資格套件]。

衝突錯誤

執行時，您可能會看到下列錯誤AWS IoT Greengrass合格套件同時在多個設備上。

```
ConflictException: Component [com.example.IDTHelloWorld : 1.0.0] for account [account-id] already exists with state: [DEPLOYABLE] { RespMetadata: { StatusCode: 409,
 RequestID: “id” }, Message_: “Component [com.example.IDTHelloWorld : 1.0.0] for
 account [account-id] already exists with state: [DEPLOYABLE]” }
```

尚未支援並行測試執行AWS IoT Greengrass資格套房。為每個裝置順序執行合格套件。

無法開始測試錯誤

您可能會遇到指向測試嘗試啟動時發生的失敗的錯誤。有數種可能的原因，因此，請執行下列動作：

- 確保執行命令中的池名稱確實存在。IDT 直接從您的`device.json`文件。
- 確保您集區中的裝置都有正確的組態參數。

碼頭資格圖像存在錯誤

Docker 應用程式管理員資格測試使用`amazon/amazon-ec2-metadata-mock`Amazon ECR 中的容器映像檔，以符合被測裝置的資格。

如果圖像已經存在於待測設備上的 Docker 容器中，則可能會收到以下錯誤。

```
The Docker image amazon/amazon-ec2-metadata-mock:version already exists on the device.
```

如果您之前已下載此映像並執行`amazon/amazon-ec2-metadata-mock`設備上的容器，請確保在運行資格測試之前從被測設備中刪除此映像。

無法讀取認證

在測試 Windows 設備時，您可能會遇到`Failed to read credential`錯誤中的`greengrass.log`file，如果您用於連接到被測設備的用戶未在該設備的憑據管理器中設置。

要解決此錯誤，請在被測設備的憑據管理器中配置 IDT 用戶的用戶和密碼。

如需詳細資訊，請參閱設定 Windows 裝置的使用者身份證明。

基斯錯誤與 PreInstalled 格蘭格拉斯

在使用運行 IDT 時 PreInstalled 格蘭格拉斯，如果你遇到錯誤Guice或者ErrorInCustomProvider，檢查文件userdata.json具有InstalledDirRootOnDevice設置為綠色安裝文件夾。IDT 會檢查檔案effectiveConfig.yaml下<InstallationDirRootOnDevice>/config/effectiveConfig.yaml。

如需詳細資訊，請參閱設定 Windows 裝置的使用者身份證明。

無效的簽章例外

當您執行 Lambda 資格測試時，您可能會遇到invalidsignatureexception如果您的 IDT 主機遇到網路存取問題，則會發生錯誤。重置路由器並再次運行測試。

機器學習資格錯誤

當您執行機器學習 (ML) 資格測試時，如果您的裝置不符合要求以部署AWS-提供的 ML 組件。若要疑難排解 ML 限定錯誤，請執行下列動作：

- 在測試運行期間部署的組件的組件日誌中查找錯誤詳細信息。元件記錄檔位於*<device-tester-extract-location>*/results/*<execution-id>*/logs/*<test-group-id>*目錄。
- 添加-Dgg.persist=installed.software的參數test.json失敗測試用例的文件。該test.json檔案位於*<device-tester-extract-location>*/tests/GGV2Q_*version* directory.

開放式測試架構 (OTF) 部署失敗

如果 OTF 測試無法完成部署，可能的原因可能是為父資料夾設定的權限TempResourcesDirOnDevice和InstallationDirRootOnDevice。若要正確設定此資料夾的權限，請執行下列命令。取代*folder-name*與父文件夾的名稱。

```
sudo chmod 755 folder-name
```

剖析錯誤

JSON 組態中的錯別字可能會導致剖析錯誤。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。此資訊應足以協助您修正錯誤，但是如果您仍然找不到錯誤，您可以在 IDE、Atom 或 Sublime 等文字編輯器中手動執行驗證，或透過 JsonLint 等線上工具執行驗證。

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。

資格報告產生錯誤

IDT 支持最新的四個*major*.*minor*的版本AWS IoT GreengrassV2 資格套件 (GGV2Q) 可產生您可以提交的資格報告AWS Partner Network將您的裝置包含在AWS Partner裝置目錄。較早版本的資格套件不會產生資格報告。

如果您對支援政策有任何疑問，請聯絡AWS Support。

遺漏必要參數錯誤

當 IDT 添加新功能時，它可能會對配置文件進行更改。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 組態檔結構描述，以確認您使用的是最新的受支援版本。

macOS 上的安全性例外

當您在 macOS 主機上運行 IDT 時，它會阻止 IDT 運行。若要執行 IDT，請將安全性例外授與屬於 IDT 執行階段功能一部分的可執行檔。當您看到主機電腦上顯示的警告訊息時，請針對每個適用的可執行檔執行下列動作：

將安全例外授與 IDT 可執行檔

1. 在 MacOS 計算機上，在蘋果菜單上，打開系統偏好。
2. 選擇安全性與隱私權」，然後在「」將軍」標籤上，選擇鎖定圖示以變更安全性設定。
3. 在被阻止的情況下`devicetester_mac_x86-64`，尋找訊息`"devicetester_mac_x86-64" was blocked from use because it is not from an identified developer.`並選擇仍然允許。
4. 繼續 IDT 測試，直到您完成所有涉及的可執行文件。

SSH 連線錯誤

當 IDT 無法連接到被測設備時，它會記錄連接故障`/results/<execution-id>/logs/<test-case-id>.log`。SSH 消息顯示在此日誌文件的頂部，因為連接到被測設備是 IDT 執行的第一個操作之一。

大部分的 Windows 設定都會使用膩子終端機應用程式來連線到 Linux 主機。此應用程式會要求您將標準 PEM 私密金鑰檔案轉換為稱為 PPK 的專屬 Windows 格式。如果你在你的配置`SSHdevice.json`檔案中，請使用 PEM 檔案。如果您使用 PPK 檔案，IDT 將無法建立與AWS IoT Greengrass設備，無法運行測試。

從 IDT v4.4.0 開始，如果您尚未在被測設備上啟用 SFTP，則可能會在日誌文件中看到以下錯誤。

```
SSH connection failed with EOF
```

若要解決此錯誤，請在您的裝置上啟用 SFTP。

串流管理員資格錯誤

當您執行串流管理員資格測試時，您可能會在`com.aws.StreamManagerExport.log`文件。

```
Failed to upload data to S3
```

串流管理員使用AWS中的認證~/root/.aws/credentials您的設備上的文件，而不是使用 IDT 導出到被測設備的環境憑據。若要避免發生此問題，請刪除credentials在您的設備上文件，然後重新運行資格測試。

逾時錯誤

您可以通過指定應用於每個測試超時的默認值的超時倍數來增加每個測試的超時時間。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 GGV2Q_1.0.0 --pool-id DevicePool1 --timeout-
multiplier 2.5
```

如需詳細資訊，請執行 `run-suite --help`。

由於配置問題而無法完成 IDT 測試用例時，會發生一些超時錯誤。您無法藉由增加逾時倍數來解決這些錯誤。使用測試運行中的日誌來解決基礎配置問題。

- 如果 MQTT 或 Lambda 元件記錄包含Access denied錯誤，您的 Greengrass 安裝文件夾可能沒有正確的文件權限。針對您在中定義的安裝路徑中的每個資料夾執行下列命令userdata.json文件。

  ```
  sudo chmod 755 folder-name
  ```

- 如果 Greengrass 記錄檔指出 Greengrass CLI 部署尚未完成，請執行下列動作：
  - 驗證bash安裝在被測設備上。
  - 如果您的userdata.json檔案包括GreengrassCliVersion配置參數，將其刪除。此參數在 IDT v4.1.0 及更新版本中已淘汰。如需詳細資訊，請參閱配置用戶數據。
- 如果 Lambda 部署測試失敗並顯示「驗證 Lambda 發佈：逾時」的錯誤訊息，且您在測試記錄檔中收到錯誤訊息 (idt-gg2-lambda-function-idt-*<resource-id>*.log) 說Error: Could not find or load main class com.amazonaws.greengrass.runtime.LambdaRuntime.，執行下列動作：
  - 確認使用的資料夾InstallationDirRootOnDevice在userdata.json文件。
  - 確保您的設備上設置了正確的用戶權限。如需詳細資訊，請參閱在您的裝置上設定使用者權限。

版本檢查錯誤

IDT 發出以下錯誤時AWSIDT 使用者的使用者登入資料沒有必要的 IAM 許可。

```
Failed to check version compatibility
```

該AWS沒有必要 IAM 許可的使用者。

的 Support AWS IoT Device Tester 政策 AWS IoT Greengrass

AWS IoT Device Tester 為 AWS IoT Greengrass 是一種測試自動化工具，用於驗證和限定您的 AWS IoT Greengrass 設備以包含在AWS Partner 設備目錄中。我們建議您使用最新版本的 AWS IoT Greengrass 和 AWS IoT Device Tester 來測試或限定您的裝置。

每個受支援的版本 AWS IoT Device Tester 都至少有一個版本可用 AWS IoT Greengrass。如需支援的版本 AWS IoT Greengrass，請參閱 Greengrass 核心版本。如需支援的版本 AWS IoT Device Tester，請參閱AWS IoT Greengrass V2 的 AWS IoT Device Tester 支援版本。

您也可以使用任何支援的 AWS IoT Greengrass 和版本 AWS IoT Device Tester 來測試或限定您的裝置。雖然您可以繼續使用不受支援的版本 AWS IoT Device Tester，但這些版本不會收到錯誤修正或更新。如果您對支援政策有任何疑問，請聯絡AWS Support。

基於 Greengrass IoT 解決方案

歐洲科技公司的每個軟件 GreenEdge 是在預覽版本，AWS IoT Greengrass並可能會改變。這個解決方案不受 AWS 支援。對於此設備的任何問題，您必須聯繫歐洲科技公司。

AWS IoT Greengrass提供合作夥伴提供的解決方案，以優化您安裝 Greengrass 的體驗。以下是與 Eurotech 合作提供的解決方案。AWS此解決方案隨附AWS IoT Greengrass核心邊緣執行階段和預先安裝的其他功能

歐洲科技

AWS與 Eurotech 合作，為正在尋找預先安裝AWS IoT Greengrass核心軟件的設備的客戶提供 IoT 解決方案。歐洲科技的萬用軟件 GreenEdge 是預先配置和預審資格的 IoT 邊緣軟件。AWS該解決方案融合了 Greengrass 和歐洲技術萬用軟件框架（ESF）的能力，通過協議適配器為客戶提供廣泛的南向連接：Modbus，OPC-UA 客戶端/服務器，S7，TwinCAT，J1939，DNP3 主站/外站，等等。使用此解決方案，您還可以將資料傳送到所有北向AWS服務AWS 雲端並連接到所有北向服務 (例如AWS IoT CoreAWS IoT SiteWiseAWS IoT Analytics、、Amazon S3 和 Amazon Kinesis Video Streams)。與 Eurotech 的設備管理解決方案 Everyware Cloud 相結合，該解決方案引入了新穎的零接觸佈建服務，從而簡化了設備上架和大規模部署。

有關歐洲科技的更多信息，請參閱歐洲科技公司。

疑難排 AWS IoT Greengrass V2

使用本節中的疑難排解資訊和解決方案來協助解決與的問題 AWS IoT Greengrass Version 2。

主題

檢視 AWS IoT Greengrass 核心軟體和元件記錄

AWS IoT Greengrass Core 軟體會將記錄檔寫入本機檔案系統，您可以使用這些檔案系統來檢視核心裝置的即時資訊。您還可以將核心設備配置為將日誌寫入日 CloudWatch 誌，以便遠程對核心設備進行故障排除。這些記錄可協助您識別元件、部署和核心裝置的問題。如需詳細資訊，請參閱 監控AWS IoT Greengrass日誌。

AWS IoT Greengrass 核心軟體問題

疑難排解 AWS IoT Greengrass 核心軟體問題。

主題

- 記憶體不足錯誤
- 無法安 Greengrass CLI
- User root is not allowed to execute
- com.aws.greengrass.lifecyclemanager.GenericExternalService: Could not determine user/group to run with
- Failed to map segment from shared object: operation not permitted
- 無法設置視窗服務
- com.aws.greengrass.util.exceptions.TLSAuthException: Failed to get trust manager
- com.aws.greengrass.deployment.IotJobsHelper: No connection available during subscribing to Iot Jobs descriptions topic. Will retry in sometime
- software.amazon.awssdk.services.iam.model.IamException: The security token included in the request is invalid
- software.amazon.awssdk.services.iot.model.IotException: User: <user> is not authorized to perform: iot:GetPolicy
- Error: com.aws.greengrass.shadowmanager.sync.model.FullShadowSyncRequest: Could not execute cloud shadow get request
- Operation aws.greengrass#<operation> is not supported by Greengrass
- java.io.FileNotFoundException: <stream-manager-store-root-dir>/stream_manager_metadata_store (Permission denied)
- com.aws.greengrass.security.provider.pkcs11.PKCS11CryptoKeyService: Private key or certificate with label <label> does not exist
- software.amazon.awssdk.services.secretsmanager.model.SecretsManagerException: User: <user> is not authorized to perform: secretsmanager:GetSecretValue on resource: <arn>
- software.amazon.awssdk.services.secretsmanager.model.SecretsManagerException: Access to KMS is not allowed
- java.lang.NoClassDefFoundError: com/aws/greengrass/security/CryptoKeySpi
- com.aws.greengrass.security.provider.pkcs11.PKCS11CryptoKeyService: CKR_OPERATION_NOT_INITIALIZED
- Greengrass core device stuck on nucleus v2.12.3

無法設定核心裝置

如果 AWS IoT Greengrass 核心軟體安裝程式失敗，而您無法設定核心裝置，您可能需要解除安裝軟體，然後再試一次。如需詳細資訊，請參閱 解除安裝AWS IoT Greengrass核心軟體。

無法將 AWS IoT Greengrass 核心軟體作為系統服務啟動

如果 AWS IoT Greengrass Core 軟體無法啟動，請檢查系統服務記錄檔以識別問題。一個常見的問題是，在 PATH 環境變數 (Linux) 或 PATH 系統變數 (視窗) 上無法使用 Java。

無法將原子核設定為系統服務

當 AWS IoT Greengrass Core 軟體安裝程式無法設定 AWS IoT Greengrass 為系統服務時，您可能會看到此錯誤。在 Linux 裝置上，如果核心裝置沒有 systemd 初始化系統，通常會發生此錯誤。即使安裝程式無法設定系統服務，也可以成功設定 AWS IoT Greengrass Core 軟體。

執行以下任意一項：

- 將 AWS IoT Greengrass Core 軟體設定為系統服務並執行。您必須將軟體配置為系統服務，才能使用的所有功能 AWS IoT Greengrass。您可以安裝 systemd 或使用不同的初始化系統。如需詳細資訊，請參閱 將 Greengrass 核配置為系統服務。
- 在沒有系統服務的情況下運行 AWS IoT Greengrass Core 軟件。您可以使用安裝程式在 Greengrass 根資料夾中設定的載入程式指令碼來執行軟體。如需詳細資訊，請參閱 在沒有系統服務的情況下執行 AWS IoT Greengrass Core 軟體。

無法連線到 AWS IoT Core

例如，當 AWS IoT Greengrass Core 軟體無法連線至擷取部署工作時，您可能會看 AWS IoT Core 到此錯誤。請執行下列操作：

- 檢查您的核心設備是否可以連接到互聯網和 AWS IoT Core。如需裝置連線到的 AWS IoT Core 端點的詳細資訊，請參閱設定 AWS IoT Greengrass 核心軟體。
- 檢查核心裝置的 AWS IoT 物件是否使用允許`iot:Connect`、`iot:Publish`、`iot:Receive`、和`iot:Subscribe`權限的憑證。
- 如果您的核心裝置使用網路 Proxy，請檢查您的核心裝置是否具有裝置角色，且其角色是否允許`iot:Connect`、`iot:Publish`、`iot:Receive`、、和`iot:Subscribe`權限。

記憶體不足錯誤

如果您的裝置沒有足夠的記憶體無法在 Java 堆積中配置物件，通常會發生此錯誤。在記憶體有限的裝置上，您可能需要指定最大堆積大小來控制記憶體配置。如需詳細資訊，請參閱 使用 JVM 選項控制內存分配。

無法安 Greengrass CLI

當您在 AWS IoT Greengrass Core 的安裝命令中使用`--deploy-dev-tools`引數時，您可能會看到下列主控台訊息。

```
Thing group exists, it could have existing deployment and devices, hence NOT creating
 deployment for Greengrass first party dev tools, please manually create a deployment
 if you wish to
```

當未安裝 Greengrass CLI 元件時，就會發生這種情況，因為您的核心裝置是具有現有部署之物群組的成員。如果您看到此訊息，您可以手動將 Greengrass CLI 元件 (`aws.greengrass.Cli`) 部署到裝置以安裝 Greengrass CLI。如需詳細資訊，請參閱 安裝 Greengrass CLI。

User root is not allowed to execute

當執行 AWS IoT Greengrass Core 軟體的使用者 (通常`root`) 沒有`sudo`與任何使用者和任何群組一起執行的權限時，您可能會看到這個錯誤。對於預設的`ggc_user`系統使用者，此錯誤如下所示：

```
Sorry, user root is not allowed to execute <command> as ggc_user:ggc_group.
```

檢查您的`/etc/sudoers`文件是否授予用戶權限以其他組`sudo`身份運行。中的使用者權限`/etc/sudoers`應如下列範例所示。

```
root    ALL=(ALL:ALL) ALL
```

com.aws.greengrass.lifecyclemanager.GenericExternalService: Could not determine user/group to run with

當核心裝置嘗試執行元件，且 Greengrass 核心未指定用於執行元件的預設系統使用者時，您可能會看到此錯誤。

若要修正此問題，請設定 Greengrass 核心以指定執行元件的預設系統使用者。如需詳細資訊，請參閱 設定執行元件的使用者 及 設定預設元件使用者。

Failed to map segment from shared object: operation not permitted

當 AWS IoT Greengrass Core 軟體因為使用noexec權限掛載/tmp資料夾而無法啟動時，您可能會看到此錯誤。AWS 通用運行時（CRT）庫默認使用該/tmp文件夾。

執行以下任意一項：

- 執行下列命令以使用exec權限重新掛載/tmp資料夾，然後再試一次。

```
sudo mount -o remount,exec /tmp
```

- 如果您執行 Greengrass 核 v2.5.0 或更新版本，您可以設定 JVM 選項來變更 CRT 程式庫使用的資料夾。 AWS 您可以在部署或安裝核心軟體時，在 Greengrass 核心元件組態中指定jvmOptions參數。 AWS IoT Greengrass 將 */path/to ### AWS CRT #####*的文件夾的路徑。

```
{
  "jvmOptions": "-Daws.crt.lib.dir=\"/path/to/use\""
}
```

無法設置視窗服務

如果您在 Microsoft 視窗 2016 設備上安裝 AWS IoT Greengrass 核心軟件，則可能會看到此錯誤。AWS IoT Greengrass 核心軟體在 Windows 2016 上不受支援，如需支援的作業系統清單，請參閱支援平台。

如果您必須使用視窗 2016，您可以執行下列動作：

1. 解壓縮下載的 AWS IoT Greengrass 核心安裝歸檔
2. 在目Greengrass錄中打開bin/greengrass.xml.template文件。
3. 將標<autoRefresh>籤新增到標籤前面的檔案結</service>尾。

```
  </log>
  <autoRefresh>false</autoRefresh>
</service>
```

com.aws.greengrass.util.exceptions.TLSAuthException: Failed to get trust manager

當您在沒有根憑證授權單位 (CA) 檔案的情況下安裝 AWS IoT Greengrass Core 軟體時，可能會看到此錯誤。

```
2022-06-05T10:00:39.556Z [INFO] (main) com.aws.greengrass.lifecyclemanager.Kernel:
 service-loaded. {serviceName=DeploymentService}
2022-06-05T10:00:39.943Z [WARN] (main)
 com.aws.greengrass.componentmanager.ClientConfigurationUtils: configure-greengrass-
mutual-auth. Error during configure greengrass client mutual auth. {}
com.aws.greengrass.util.exceptions.TLSAuthException: Failed to get trust manager
```

檢查您是否在提供給安裝程式的組態檔案中使用`rootCaPath`參數指定有效的根 CA 檔案。如需詳細資訊，請參閱 安裝 AWS IoT Greengrass 核心軟體。

com.aws.greengrass.deployment.IotJobsHelper: No connection available during subscribing to Iot Jobs descriptions topic. Will retry in sometime

當核心裝置無法連線以訂閱部署工作通知時，您可能會看到此警告訊息。 AWS IoT Core 請執行下列操作：

- 檢查核心裝置是否已連線至網際網路，並可連線到您設定的 AWS IoT 資料端點。如需核心裝置使用之端點的詳細資訊，請參閱允許裝置流量透過 Proxy 或防火牆。
- 檢查 Greengrass 日誌是否有其他顯示其他根本原因的錯誤。

software.amazon.awssdk.services.iam.model.IamException: The security token included in the request is invalid

當您使用自動佈建來安裝 AWS IoT Greengrass Core 軟體，且安裝程式使用無效的 AWS 工作階段 Token 時，可能會看到此錯誤。請執行下列操作：

- 如果您使用臨時安全憑據，請檢查會話令牌是否正確，以及是否正在複製並粘貼完整的會話令牌。
- 如果您使用長期安全憑據，請檢查設備是否沒有以前使用臨時憑據的會話令牌。請執行下列操作：
  1. 執行下列命令以取消設定工作階段 Token 環境變數。

Linux or Unix

```
unset AWS_SESSION_TOKEN
```

Windows Command Prompt (CMD)

```
set AWS_SESSION_TOKEN=
```

PowerShell

```
Remove-Item Env:\AWS_SESSION_TOKEN
```

2. 檢查 AWS 憑據文件是否~/.aws/credentials包含會話令牌，aws_session_token。如果是這樣，請從文件中刪除該行。

```
aws_session_token = AQoEXAMPLEH4aoAH0gNCAPyJxz4BlCFFxWNE1OPTgk5TthT
+FvwqnKwRcOIfrRh3c/LTo6UDdyJwOOvEVPvLXCrrrUtdnniCEXAMPLE/
IvU1dYUg2RVAJBanLiHb4IgRmpRV3zrkuWJOgQs8IZZaIv2BXIa2R4Olgk
```

您也可以在不提供 AWS 憑證的情況下安裝 AWS IoT Greengrass Core 軟體。如需詳細資訊，請參閱 透過手動佈建資源安裝 AWS IoT Greengrass 核心軟體 或 透過 AWS IoT 叢集佈建安裝 AWS IoT Greengrass 核心軟體 。

software.amazon.awssdk.services.iot.model.IotException: User: <user> is not authorized to perform: iot:GetPolicy

當您使用自動佈建來安裝 AWS IoT Greengrass Core 軟體，且安裝程式使用的 AWS 認證不具備必要權限時，可能會看到此錯誤。如需所需權限的詳細資訊，請參閱安裝程式佈建資源的最低 IAM 政策。

檢查登入資料 IAM 身分的許可，並授予 IAM 身分遺失的任何必要許可。

Error: com.aws.greengrass.shadowmanager.sync.model.FullShadowSyncRequest: Could not execute cloud shadow get request

當您使用陰影管理員元件與裝置陰影同步處理時，您可能會看到這個錯誤 AWS IoT Core。HTTP 403 狀態碼表示發生此錯誤，因為核心裝置的 AWS IoT 政策未授予呼叫權限GetThingShadow。

```
com.aws.greengrass.shadowmanager.sync.model.FullShadowSyncRequest: Could not execute
 cloud shadow get request. {thing name=MyGreengrassCore, shadow name=MyShadow}
2021-07-14T21:09:02.456Z [ERROR] (pool-2-thread-109)
 com.aws.greengrass.shadowmanager.sync.SyncHandler: sync. Skipping sync request. {thing
 name=MyGreengrassCore, shadow name=MyShadow}
com.aws.greengrass.shadowmanager.exception.SkipSyncRequestException:
 software.amazon.awssdk.services.iotdataplane.model.IotDataPlaneException:
 null (Service: IotDataPlane, Status Code: 403, Request ID:
 f6e713ba-1b01-414c-7b78-5beb3f3ad8f6, Extended Request ID: null)
```

若要將本機陰影與同步 AWS IoT Core，核心裝置的 AWS IoT 策略必須授予以下權限：

- `iot:GetThingShadow`
- `iot:UpdateThingShadow`
- `iot:DeleteThingShadow`

檢查核心設備的 AWS IoT 策略，並添加任何缺少的必需權限。如需詳細資訊，請參閱下列內容：

- AWS IoT Core AWS IoT 開發人員指南中的政策動作
- 更新核心裝置的AWS IoT政策

Operation aws.greengrass#<operation> is not supported by Greengrass

當您在自訂 Greengrass 元件中使用處理序間通訊 (IPC) 作業，且核心裝置上未安裝所需 AWS提供的元件時，您可能會看到這個錯誤。

若要修正此問題，請在元件方案中新增必要的元件做為相依性，以便 AWS IoT Greengrass 核心軟體在您部署元件時安裝必要的元件。

- 擷取秘密值 — `aws.greengrass.SecretManager`
- 與局部陰影互動 — `aws.greengrass.ShadowManager`
- 管理本機部署和元件 — `aws.greengrass.Cli` v2.6.0 或更新版本
- 驗證和授權用戶端裝置 — `aws.greengrass.clientdevices.Auth` v2.2.0 或更新版本

java.io.FileNotFoundException: <stream-manager-store-root-dir>/stream_manager_metadata_store (Permission denied)

當您設定串流管理員使用不存在或具有正確權限的根資料夾時，您可能會在串流管理員記錄檔 (`aws.greengrass.StreamManager.log`) 中看到這個錯誤。如需如何設定此資料夾的詳細資訊，請參閱串流管理員設定。

com.aws.greengrass.security.provider.pkcs11.PKCS11CryptoKeyService: Private key or certificate with label <label> does not exist

當 PKCS #11 提供者元件找不到或載入您在將 AWS IoT Greengrass Core 軟體設定為使用硬體安全性模組 (HSM) 時指定的私密金鑰或憑證，就會發生此錯誤。請執行下列操作：

- 使用您設定要使用的 AWS IoT Greengrass Core 軟體的插槽、使用者 PIN 碼和物件標籤，檢查私密金鑰和憑證是否儲存在 HSM 中。
- 檢查私密金鑰和憑證是否在 HSM 中使用相同的物件標籤。
- 如果您的 HSM 支援物件 ID，請檢查私密金鑰和憑證是否在 HSM 中使用相同的物件識別碼。

請查看 HSM 的文件，以了解如何查詢有關 HSM 中安全性權杖的詳細資料。如果您需要變更安全性權杖的插槽、物件標籤或物件 ID，請查看 HSM 的文件以瞭解如何執行此動作。

software.amazon.awssdk.services.secretsmanager.model.SecretsManagerExcep User: <user> is not authorized to perform: secretsmanager:GetSecretValue on resource: <arn>

當您使用秘密管理員元件部署密碼時，可能會發生這個 AWS Secrets Manager 錯誤。如果核心設備的令牌交換 IAM 角色未授予獲取密碼的權限，則部署失敗，並且 Greengrass 日誌包含此錯誤。

授權核心裝置下載密碼

1. 將`secretsmanager:GetSecretValue`權限新增至核心裝置的權杖交換角色。下列範例原則陳述式會授與取得密碼值的權限。

```
{
    "Effect": "Allow",
    "Action": [
        "secretsmanager:GetSecretValue"
```

```
    ],
    "Resource": [
        "arn:aws:secretsmanager:us-west-2:123456789012:secret:MyGreengrassSecret-
abcdef"
    ]
}
```

如需詳細資訊，請參閱 授權核心設備與AWS服務。

2. 將部署重新套用至核心裝置。執行以下任意一項：
   - 不進行任何變更即修訂部署。核心裝置在收到修訂後的部署時，會嘗試再次下載密碼。如需詳細資訊，請參閱 修訂部署。
   - 重新啟動 AWS IoT Greengrass 核心軟體以重試部署。如需更多資訊，請參閱執行AWS IoT Greengrass核心軟體

   如果密碼管理員成功下載密碼，則部署成功。

software.amazon.awssdk.services.secretsmanager.model.SecretsManagerExcep Access to KMS is not allowed

當您使用 Secret Manager 元件部署由 AWS Key Management Service 金鑰加密的密碼時 AWS Secrets Manager ，可能會發生這個錯誤。如果核心設備的令牌交換 IAM 角色未授予解密權限，則部署失敗，並且 Greengrass 日誌包含此錯誤。

若要修正此問題，請將`kms:Decrypt`權限新增至核心裝置的 Token 交換角色。如需詳細資訊，請參閱下列內容：

- AWS Secrets Manager 用戶指南中的秘密加密和解密
- 授權核心設備與AWS服務

java.lang.NoClassDefFoundError: com/aws/greengrass/security/CryptoKeySpi

當您嘗試安裝具有硬體安全性的 AWS IoT Greengrass Core 軟體，而您使用的舊版 Greengrass 核心版本不支援硬體安全性整合時，可能會看到這個錯誤。若要使用硬體安全性整合，您必須使用 Greengrass 核 v2.5.3 或更新版本。

com.aws.greengrass.security.provider.pkcs11.PKCS11CryptoKeyService: CKR_OPERATION_NOT_INITIALIZED

當您執行 AWS IoT Greengrass Core 做為系統服務時，使用 TPM2 程式庫時，可能會看到這個錯誤。

此錯誤表示您需要新增環境變數，以提供 PKCS #11 儲存區在 AWS IoT Greengrass 核心 systemd 服務檔案中的位置。

如需詳細資訊，請參閱PKCS #11 供應商元件文件的「需求」一節。

Greengrass core device stuck on nucleus v2.12.3

如果您的 Greengrass 核心裝置無法從核心 2.12.3 版本修改您的部署，您可能需要下載檔案，並使用 Greengrass 核心 2.12.2 版來取代`Greengrass.jar`檔案。請執行下列操作：

1. 在您的 Greengrass 核心裝置上，執行下列命令以停止 Greengrass 核心軟體。

   Linux or Unix

   ```
   sudo systemctl stop greengrass
   ```

   Windows Command Prompt (CMD)

   ```
   sc stop "greengrass"
   ```

   PowerShell

   ```
   Stop-Service -Name "greengrass"
   ```

2. 在您的核心裝置上，將 AWS IoT Greengrass 軟體下載到名為`greengrass-2.12.2.zip`.

   Linux or Unix

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-2.12.2.zip >
    greengrass-2.12.2.zip
   ```

   Windows Command Prompt (CMD)

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-2.12.2.zip >
    greengrass-2.12.2.zip
   ```

PowerShell

```
iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-2.12.2.zip -
OutFile greengrass-2.12.2.zip
```

3. 將 AWS IoT Greengrass Core 軟件解壓縮到設備上的文件夾。以您要使用的資料夾取*GreengrassInstaller*代。

   Linux or Unix

   ```
   unzip greengrass-2.12.2.zip -d GreengrassInstaller && rm greengrass-2.12.2.zip
   ```

   Windows Command Prompt (CMD)

   ```
   mkdir GreengrassInstaller && tar -xf greengrass-2.12.2.zip -
   C GreengrassInstaller && del greengrass-2.12.2.zip
   ```

   PowerShell

   ```
   Expand-Archive -Path greengrass-2.12.2.zip -DestinationPath .\
   \GreengrassInstaller
   rm greengrass-2.12.2.zip
   ```

4. 運行以下命令，以覆蓋核 2.12.3 版本的 Greengrass JAR 文件與核 2.12.2 Greengrass JAR 文件。

   Linux or Unix

   ```
   sudo cp ./GreengrassInstaller/lib/Greengrass.jar /greengrass/v2/packages/
   artifacts-unarchived/aws.greengrass.Nucleus/2.12.3/aws.greengrass.nucleus/lib
   ```

   Windows Command Prompt (CMD)

   ```
   robocopy ./GreengrassInstaller/lib/Greengrass.jar /greengrass/v2/packages/
   artifacts-unarchived/aws.greengrass.Nucleus/2.12.3/aws.greengrass.nucleus/lib /E
   ```

PowerShell

```
cp -Path ./GreengrassInstaller/lib/Greengrass.jar -Destination /
greengrass/v2/packages/artifacts-unarchived/aws.greengrass.Nucleus/2.12.3/
aws.greengrass.nucleus/lib
```

5. 運行以下命令來啟動 Greengrass 核心軟件。

   Linux or Unix

   ```
   sudo systemctl start greengrass
   ```

   Windows Command Prompt (CMD)

   ```
   sc start "greengrass"
   ```

   PowerShell

   ```
   Start-Service -Name "greengrass"
   ```

AWS IoT Greengrass 雲端問題

請使用下列資訊疑難排解 AWS IoT Greengrass 主控台和 API 的問題。每個項目都會對應到您在執行動作時可能會看到的錯誤訊息。

An error occurred (AccessDeniedException) when calling the CreateComponentVersion operation: User: arn:aws:iam::123456789012:user/<username> is not authorized to perform: null

當您從 AWS IoT Greengrass 主控台或使用作業建立元件版本時，您可能會看到這個錯[CreateComponentVersion](#)誤。

此錯誤表示您的方案不是有效的 JSON 或 YAML。請檢查方案的語法，修正任何語法問題，然後再試一次。您可以使用線上 JSON 或 YAML 語法檢查程式來識別方案中的語法問題。

Invalid Input: Encountered following errors in Artifacts: {<s3ArtifactUri> = Specified artifact resource cannot be accessed}

當您從 AWS IoT Greengrass 主控台或使用作業建立元件版本時，您可能會看到這個錯誤。此錯誤表示元件方案中的 S3 成品無效。

請執行下列操作：

- 檢查 S3 儲存貯體是否 AWS 區域 位於建立元件的位置。 AWS IoT Greengrass 不支援元件成品的跨區域要求。
- 檢查成品 URI 是否為有效的 S3 物件 URL，並檢查該 S3 物件 URL 是否存在成品。
- 檢查您是否 AWS 帳戶 有權在其 S3 物件 URL 存取成品。

INACTIVE deployment status

在沒有必要相依 AWS IoT 政策的情況下呼叫 ListDeploymentsAPI 時，您可能會取得`INACTIVE`部署狀態。您必須具備必要的權限，才能取得正確的部署狀態。您可以通過查看由定義的操作 AWS IoT Greengrass V2並遵循所需的權限來找到相依動作`ListDeployments`。如果沒有必要的相依 AWS IoT 權限，您仍會看到部署狀態，但可能會看到的不正確的部署狀態`INACTIVE`。

核心裝置部署問題

疑難排解 Greengrass 核心裝置上的部署問題。每個項目都對應到您可能在核心設備上看到的日誌消息。

主題

- Error: com.aws.greengrass.componentmanager.exceptions.PackageDownloadException: Failed to download artifact
- Error: com.aws.greengrass.componentmanager.exceptions.ArtifactChecksumMismatchException: Integrity check for downloaded artifact failed. Probably due to file corruption.
- Error: com.aws.greengrass.componentmanager.exceptions.NoAvailableComponentVersionException: Failed to negotiate component <name> version with cloud and no local applicable version satisfying requirement <requirements>

- software.amazon.awssdk.services.greengrassv2data.model.ResourceNotFoundException: The latest version of Component <componentName> doesn't claim platform <coreDevicePlatform> compatibility
- com.aws.greengrass.componentmanager.exceptions.PackagingException: The deployment attempts to update the nucleus from aws.greengrass.Nucleus-<version> to aws.greengrass.Nucleus-<version> but no component of type nucleus was included as target component
- Error: com.aws.greengrass.deployment.exceptions.DeploymentException: Unable to process deployment. Greengrass launch directory is not set up or Greengrass is not set up as a system service
- Info: com.aws.greengrass.deployment.exceptions.RetryableDeploymentDocumentDownloadException: Greengrass Cloud Service returned an error when getting full deployment configuration
- Warn: com.aws.greengrass.deployment.DeploymentService: Failed to get thing group hierarchy
- Info: com.aws.greengrass.deployment.DeploymentDocumentDownloader: Calling Greengrass cloud to get full deployment configuration
- Caused by: software.amazon.awssdk.services.greengrassv2data.model.GreengrassV2DataException: null (Service: GreengrassV2Data, Status Code: 403, Request ID: <some_request_id>, Extended Request ID: null)

Error: com.aws.greengrass.componentmanager.exceptions.PackageDownloadExceptio Failed to download artifact

當核心裝置套用部署時， AWS IoT Greengrass 核心軟體無法下載元件成品時，您可能會看到此錯誤。由於此錯誤，部署失敗。

當您收到此錯誤時，記錄檔也會包含可用來識別特定問題的堆疊追蹤。下列每個項目都會對應到您可能會在`Failed to download artifact`錯誤訊息的堆疊追蹤中看到的訊息。

主題

- software.amazon.awssdk.services.s3.model.S3Exception: null (Service: S3, Status Code: 403, Request ID: null, ...)

- software.amazon.awssdk.services.s3.model.S3Exception: Access Denied (Service: S3, Status Code: 403, Request ID: <requestID>

software.amazon.awssdk.services.s3.model.S3Exception: null (Service: S3, Status Code: 403, Request ID: null, ...)

在以下情況下，PackageDownloadException 錯誤可能包括此堆棧跟踪：

- 元件成品無法在您在元件的方案中指定的 S3 物件 URL 中使用。檢查您已將成品上傳到 S3 儲存貯體，以及成品 URI 是否與儲存貯體中成品的 S3 物件 URL 相符。
- 核心裝置的權杖交換角色不允許 AWS IoT Greengrass Core 軟體從您在元件方案中指定的 S3 物件 URL 下載元件成品。檢查令牌交換角色是否允許`s3:GetObject`存在成品可用的 S3 物件 URL。

software.amazon.awssdk.services.s3.model.S3Exception: Access Denied (Service: S3, Status Code: 403, Request ID: <requestID>

當核心設備沒有調用權限時，PackageDownloadException 錯誤可能包括此堆棧跟踪`s3:GetBucketLocation`。錯誤訊息也包含下列訊息。

```
reason: Failed to determine S3 bucket location
```

檢查核心裝置的權杖交換角色是否允許`s3:GetBucketLocation`可用成品的 S3 儲存貯體。

Error: com.aws.greengrass.componentmanager.exceptions.ArtifactChecksumMismatchI Integrity check for downloaded artifact failed. Probably due to file corruption.

當核心裝置套用部署時， AWS IoT Greengrass 核心軟體無法下載元件成品時，您可能會看到此錯誤。部署失敗，因為下載的成品檔案的總和檢查碼與您建立元件時所 AWS IoT Greengrass 計算的總和檢查碼不符。

請執行下列操作：

- 檢查您託管它的 S3 儲存貯體中的成品檔案是否已變更。如果檔案自建立元件後變更，請將其還原為核心裝置預期的先前版本。如果您無法將檔案還原至先前的版本，或者想要使用新版本的檔案，請使用人工因素檔案建立新版本的元件。

- 檢查核心設備的互聯網連接。如果成品檔案在下載時損毀，則可能會發生此錯誤。建立新部署，然後再試一次。

Error: com.aws.greengrass.componentmanager.exceptions.NoAvailableComponentVers Failed to negotiate component <name> version with cloud and no local applicable version satisfying requirement <requirements>

當核心裝置找不到符合該核心裝置部署需求的元件版本時，您可能會看到此錯誤。核心裝置會檢查 AWS IoT Greengrass 服務和本機裝置上的元件。錯誤訊息包含每個部署的目標以及元件的部署版本需求。部署目標可以是物件、物件群組LOCAL_DEPLOYMENT，或代表核心裝置上的本機部署。

在下列情況下，可能會發生這個問題：

- 核心裝置是具有衝突元件版本需求的多個部署的目標。例如，核心裝置可能是包含com.example.HelloWorld元件的多個部署目標，其中一個部署需要版本 1.0.0，而另一個部署則需要版本 1.0.1。不可能有一個符合這兩個需求的組件，因此部署失敗。
- 組件版本不存在於 AWS IoT Greengrass 服務或本地設備上。例如，元件可能已被刪除。
- 存在符合版本需求的組件版本，但沒有與核心設備的平台兼容。
- 核心設備的 AWS IoT 策略未授予greengrass:ResolveComponentCandidates權限。在錯誤記錄檔Status Code: 403中尋找以識別此問題。若要解決此問題，請將greengrass:ResolveComponentCandidates權限新增至核心裝置的 AWS IoT 原則。如需詳細資訊，請參閱 AWS IoT Greengrass V2核心裝置的最低AWS IoT原則。

若要解決此問題，請修訂部署以包含相容的元件版本或移除不相容的元件版本。如需如何修訂雲端部署的更多資訊，請參閱修訂部署。如需如何修訂本機部署的相關資訊，請參閱 AWS IoT Greengrass CLI 部署建立命令。

software.amazon.awssdk.services.greengrassv2data.model.ResourceNotFoundE The latest version of Component <componentName> doesn't claim platform <coreDevicePlatform> compatibility

當您將元件部署到核心裝置時，您可能會看到此錯誤，而且該元件未列出與核心裝置平台相容的平台。執行以下任意一項：

- 如果元件是自訂 Greengrass 元件，您可以將元件更新為與核心裝置相容。添加與核心設備平台匹配的新清單，或更新現有清單以匹配核心設備的平台。如需詳細資訊，請參閱 AWS IoT Greengrass元件配方參考。
- 如果元件由提供 AWS，請檢查元件的另一個版本是否與核心裝置相容。如果沒有版本兼容，請聯繫我們AWS re:Post使用標AWS IoT Greengrass 籤，或聯繫我們AWS Support。

com.aws.greengrass.componentmanager.exceptions.PackagingException: The deployment attempts to update the nucleus from aws.greengrass.Nucleus-<version> to aws.greengrass.Nucleus-<version> but no component of type nucleus was included as target component

當您部署依賴 Greengrass 核心的元件時，您可能會看到這個錯誤，而且核心裝置執行的是較舊的 Greengrass 核心版本，而不是最新的次要版本。之所以發生這個錯誤，是因為 AWS IoT Greengrass Core 軟體會嘗試自動將元件更新為最新的相容版本。但是， AWS IoT Greengrass 核心軟體會阻止 Greengrass 核更新為新的次要版本，因為幾個 AWS提供的元件依賴於 Greengrass 核的特定次要版本。如需詳細資訊，請參閱 Greengrass 核更新行為。

您必須修訂部署，以指定您要使用的 Greengrass 核心版本。執行以下任意一項：

- 修改部署以指定核心裝置目前執行的 Greengrass 核心核心版本。
- 修改部署以指定 Greengrass 核心的較新次要版本。如果您選擇此選項，則還必須更新依賴 Greengrass AWS核特定次要版本的所有提供元件的版本。如需詳細資訊，請參閱 AWS-提供的組件。

Error: com.aws.greengrass.deployment.exceptions.DeploymentException: Unable to process deployment. Greengrass launch directory is not set up or Greengrass is not set up as a system service

當您將 Greengrass 裝置從一個物件群組移至另一個物件群組，然後回到具有需要 Greengrass 重新啟動部署的原始群組時，您可能會看到此錯誤。

若要解決此問題，請重新建立裝置的啟動目錄。我們也強烈建議您升級至 Greengrass 核心 2.9.6 版或更新版本。

以下是用來重新建立啟動目錄的 Linux 指令碼。將腳本保存在名為的文件中`fix_directory.sh`。

```
#!/bin/bash

set -e

GG_ROOT=$1
GG_VERSION=$2

CURRENT="$GG_ROOT/alts/current"

if [ ! -L "$CURRENT" ]; then
  mkdir -p $GG_ROOT/alts/directory_fix
  echo "Relinking $GG_ROOT/alts/directory_fix to $CURRENT"
  ln -sf $GG_ROOT/alts/directory_fix $CURRENT
fi

TARGET=$(readlink $CURRENT)

if [[ ! -d "$TARGET" ]]; then
  echo "Creating directory: $TARGET"
  mkdir -p "$TARGET"
fi

DISTRO_LINK="$TARGET/distro"
DISTRO="$GG_ROOT/packages/artifacts-unarchived/aws.greengrass.Nucleus/$GG_VERSION/
aws.greengrass.nucleus/"
echo "Relinking Nucleus artifacts to $DISTRO_LINK"
ln -sf $DISTRO $DISTRO_LINK
```

若要執行指令碼，請執行下列命令：

```
[root@ip-172-31-27-165 ~]# ./fix_directory.sh /greengrass/v2 2.9.5
Relinking /greengrass/v2/alts/directory_fix to /greengrass/v2/alts/current
Relinking Nucleus artifacts to /greengrass/v2/alts/directory_fix/distro
```

Info: com.aws.greengrass.deployment.exceptions.RetryableDeploymentDocumentDow Greengrass Cloud Service returned an error when getting full deployment configuration

當核心裝置收到大型部署文件時，您可能會看到此錯誤，該文件是大於 7 KB (針對以物件為目標的部署) 或 31 KB (針對以物件群組為目標的部署)。若要擷取大型部署文件，核心裝置的 AWS IoT 原則必須允許`greengrass:GetDeploymentConfiguration`權限。當核心裝置沒有此權限時，可能會發生此錯誤。發生此錯誤時，部署會無限期地重試，且其狀態為 [進行中] () `IN_PROGRESS`。

若要解決此問題，請將`greengrass:GetDeploymentConfiguration`權限新增至核心裝置的 AWS IoT 原則。如需詳細資訊，請參閱 更新核心裝置的AWS IoT政策。

Warn: com.aws.greengrass.deployment.DeploymentService: Failed to get thing group hierarchy

當核心裝置收到部署且核心裝置的 AWS IoT 政策不允許`greengrass:ListThingGroupsForCoreDevice`權限時，您可能會看到此警告。當您建立部署時，核心裝置會使用此權限來識別其物群組，並移除您從中移除核心裝置之任何物件群組的元件。如果核心裝置執行 Greengrass 核心 v2.5.0，則部署會失敗。如果核心裝置執行 Greengrass 核心 v2.5.1 或更新版本，部署會繼續進行，但不會移除元件。如需有關物件群組移除行為的詳細資訊，請參閱將 AWS IoT Greengrass元件部署到裝置。

若要更新核心裝置的行為，以移除您從中移除核心裝置之物件群組的元件，請將`greengrass:ListThingGroupsForCoreDevice`權限新增至核心裝置的 AWS IoT 策略。如需詳細資訊，請參閱 更新核心裝置的AWS IoT政策。

Info: com.aws.greengrass.deployment.DeploymentDocumentDownloader: Calling Greengrass cloud to get full deployment configuration

您可能會看到多次列印此資訊訊息，而不會出現錯誤，因為核心裝置會在記錄層級記`DEBUG`錄錯誤。核心裝置收到大型部署文件時，可能會發生此問題。發生此問題時，部署會無限期地重試，且其狀態為 [進行中] () `IN_PROGRESS`。如需如何解決此問題的詳細資訊，請參閱此疑難排解項目。

Caused by: software.amazon.awssdk.services.greengrassv2data.model.GreengrassV2DataE
null (Service: GreengrassV2Data, Status Code: 403, Request ID: <some_request_id>, Extended Request ID: null)

當數據通道 API 沒有`iot:Connect`權限時，您可能會看到此錯誤。如果您沒有正確的政策，您將收到`GreengrassV2DataException: 403`. 若要建立權限原則，請依照下列指示執行：建立 AWS IoT 政策。

核心裝置元件問題

疑難排解核心裝置上的 Greengrass 元件問題。

主題

- Warn: '<command>' is not recognized as an internal or external command
- Python 腳本不會記錄消息
- 變更預設組態時，元件組態不會更新
- awsiot.greengrasscoreipc.model.UnauthorizedError
- com.aws.greengrass.authorization.exceptions.AuthorizationException: Duplicate policy ID "<id>" for principal "<componentList>"
- com.aws.greengrass.tes.CredentialRequestHandler: Error in retrieving AwsCredentials from TES (HTTP 400)
- com.aws.greengrass.tes.CredentialRequestHandler: Error in retrieving AwsCredentials from TES (HTTP 403)
- com.aws.greengrass.tes.CredentialsProviderError: Could not load credentials from any providers
- Received error when attempting to retrieve ECS metadata: Could not connect to the endpoint URL: "<tokenExchangeServiceEndpoint>"
- copyFrom: <configurationPath> is already a container, not a leaf
- com.aws.greengrass.componentmanager.plugins.docker.exceptions.DockerLoginException: Error logging into the registry using credentials - 'The stub received bad data.'
- java.io.IOException: Cannot run program "cmd" ...: [LogonUser] The password for this account has expired.

- aws.greengrass.StreamManager: Instant exceeds minimum or maximum instant

Warn: '<command>' is not recognized as an internal or external command

當 AWS IoT Greengrass 核心軟體無法在元件的生命週期指令碼中執行命令時，您可能會在 Greengrass 元件的記錄檔中看到這個錯誤。該組件的狀態變BROKEN為這個錯誤的結果。如果執行元件的系統使用者 (例如ggc_user，在 PATH 中的資料夾中找不到命令的可執行檔)，就會發生此錯誤。

在 Windows 裝置上，檢查包含可執行檔的資料夾是否位PATH於執行元件的系統使用者。如果缺少它PATH，請執行下列其中一個動作：

- 將可執行檔的資料夾加入至PATH系統變數，此變數可供所有使用者使用。然後，重新啟動元件。

  如果您執行 Greengrass 核心 2.5.0，則在更新PATH系統變數之後，您必須重新啟動 AWS IoT Greengrass 核心軟體，才能執行更新的元件。PATH如果 AWS IoT Greengrass 核心軟體PATH在您重新啟動軟體後未使用更新的軟體，請重新啟動裝置，然後再試一次。如需詳細資訊，請參閱 執行 AWS IoT Greengrass核心軟體。
- 將可執行檔的資料夾新增至執行元件之系統使用者的使用者變數。PATH

Python 腳本不會記錄消息

Greengrass 核心裝置會收集可用來識別元件問題的記錄檔。如果您的 Python 腳本stdout和stderr消息沒有出現在組件日誌中，則可能需要刷新緩衝區或禁用 Python 中這些標準輸出流的緩衝。執行下列任何一項：

- 使用 -u 參數運行 Python 以禁用和緩衝。stdout stderr

  Linux or Unix

  ```
  python3 -u hello_world.py
  ```

  Windows

  ```
  py -3 -u hello_world.py
  ```
- 在組件的配方中使用 Setenv 將 PYTHON緩衝環境變量設置為非空字符串。此環境變數會停用stdout和stderr的緩衝。
- 清除stdout或stderr串流的緩衝區。執行以下任意一項：

- 打印時刷新消息。

```
import sys

print('Hello, error!', file=sys.stderr, flush=True)
```

- 打印後刷新一條消息。您可以在刷新流之前發送多個消息。

```
import sys

print('Hello, error!', file=sys.stderr)
sys.stderr.flush()
```

如需如何驗證 Python 指令碼是否輸出記錄訊息的詳細資訊，請參閱監控AWS IoT Greengrass日誌。

變更預設組態時，元件組態不會更新

當您在元件的方案DefaultConfiguration中變更時，新的預設組態不會在部署期間取代元件的現有組態。若要套用新的預設模型組態，您必須將零組件的模型組態重設為其預設設定。部署元件時，請指定單一空字串做為重設更新。

Console

重設路徑

```
[""]
```

AWS CLI

下列指令會建立核心裝置的部署。

```
aws greengrassv2 create-deployment --cli-input-json file://reset-configuration-
deployment.json
```

該文reset-configuration-deployment.json件包含以下 JSON 文檔。

```
{
  "targetArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
  "deploymentName": "Deployment for MyGreengrassCore",
```

```
  "components": {
    "com.example.HelloWorld": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {,
        "reset": [""]
      }
    }
  }
}
```

Greengrass CLI

以下 Greengrass CLI 命令會在核心裝置上建立本機部署。

```
sudo greengrass-cli deployment create \
  --recipeDir recipes \
  --artifactDir artifacts \
  --merge "com.example.HelloWorld=1.0.0" \
  --update-config reset-configuration-deployment.json
```

該文`reset-configuration-deployment.json`件包含以下 JSON 文檔。

```
{
  "com.example.HelloWorld": {
    "RESET": [""]
  }
}
```

awsiot.greengrasscoreipc.model.UnauthorizedError

當元件沒有對資源執行 IPC 作業的權限時，您可能會在 Greengrass 元件的記錄檔中看到此錯誤。若要授與元件呼叫 IPC 作業的權限，請在元件的組態中定義 IPC 授權原則。如需詳細資訊，請參閱 授權元件執行 IPC 作業。

> **Tip**
>
> 如果您變更零組件方案`DefaultConfiguration`中的，您必須將零組件的模型組態重設為其新的預設組態。部署元件時，請指定單一空字串做為重設更新。如需詳細資訊，請參閱 變更預設組態時，元件組態不會更新。

com.aws.greengrass.authorization.exceptions.AuthorizationException: Duplicate policy ID "<id>" for principal "<componentList>"

如果多個 IPC 授權原則 (包括核心裝置上的所有元件) 使用相同的原則識別碼，您可能會看到此錯誤。

請檢查元件的 IPC 授權政策，修復任何重複項目，然後再試一次。若要建立唯一的原則識別碼，建議您結合元件名稱、IPC 服務名稱和計數器。如需詳細資訊，請參閱 授權元件執行 IPC 作業。

> **Tip**
>
> 如果您變更零組件方案`DefaultConfiguration`中的，您必須將零組件的模型組態重設為其新的預設組態。部署元件時，請指定單一空字串做為重設更新。如需詳細資訊，請參閱 變更預設組態時，元件組態不會更新。

com.aws.greengrass.tes.CredentialRequestHandler: Error in retrieving AwsCredentials from TES (HTTP 400)

當核心裝置無法從權杖交換服務取得認 AWS 證時，您可能會看到此錯誤。HTTP 400 狀態碼表示發生此錯誤，是因為核心設備的令牌交換 IAM 角色不存在，或者沒有允許 AWS IoT 憑據提供者承擔的信任關係。

請執行下列操作：

1. 識別核心裝置使用的權杖交換角色。該錯誤消息包括核心設備的 AWS IoT 角色別名，該別名指向令牌交換角色。在您的開發電腦上執行下列命令，並*MyGreengrassCoreTokenExchangeRoleAlias*以錯誤訊息中的 AWS IoT 角色別名名稱取代。

   ```
   aws iot describe-role-alias --role-alias MyGreengrassCoreTokenExchangeRoleAlias
   ```

   該響應包括令牌交換 IAM 角色的 Amazon 資源名稱（ARN）。

   ```
   {
     "roleAliasDescription": {
       "roleAlias": "MyGreengrassCoreTokenExchangeRoleAlias",
       "roleAliasArn": "arn:aws:iot:us-west-2:123456789012:rolealias/
   MyGreengrassCoreTokenExchangeRoleAlias",
   ```

```
    "roleArn": "arn:aws:iam::123456789012:role/MyGreengrassV2TokenExchangeRole",
    "owner": "123456789012",
    "credentialDurationSeconds": 3600,
    "creationDate": "2021-02-05T16:46:18.042000-08:00",
    "lastModifiedDate": "2021-02-05T16:46:18.042000-08:00"
  }
}
```

2. 檢查角色是否存在。執行下列命令，並以權杖交換角色的名稱取代 *MyGreengrassV2TokenExchangeRole*。

```
aws iam get-role --role-name MyGreengrassV2TokenExchangeRole
```

如果命令返回NoSuchEntity錯誤，則該角色不存在，您必須創建它。如需如何建立和設定此角色的詳細資訊，請參閱授權核心設備與AWS服務。

3. 檢查角色是否具有允許 AWS IoT 認證提供者承擔的信任關係。上一個AssumeRolePolicyDocument步驟的回應包含定義角色的信任關係。角色必須定義允許credentials.iot.amazonaws.com承擔它的信任關係。這份文件看起來應該類似於下列範例。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

如果角色的信任關係不credentials.iot.amazonaws.com允許假設，您必須將此信任關係新增至角色。如需更多資訊，請參閱 AWS Identity and Access Management IAM 使用者指南中的修改角色。

com.aws.greengrass.tes.CredentialRequestHandler: Error in retrieving AwsCredentials from TES (HTTP 403)

當核心裝置無法從權杖交換服務取得認 AWS 證時，您可能會看到此錯誤。HTTP 403 狀態碼表示發生此錯誤，因為核心裝置的 AWS IoT 原則未授予核心裝置 AWS IoT 角色別名的`iot:AssumeRoleWithCertificate`權限。

檢閱核心裝置的 AWS IoT 原則，並新增核心裝置 AWS IoT 角色別名的`iot:AssumeRoleWithCertificate`權限。錯誤訊息包含核心裝置的目前 AWS IoT 角色別名。如需有關此權限以及如何更新核心裝置 AWS IoT 原則的詳細資訊，請參閱AWS IoT Greengrass V2核心裝置的最低AWS IoT原則和更新核心裝置的AWS IoT政策。

com.aws.greengrass.tes.CredentialsProviderError: Could not load credentials from any providers

當元件嘗試要求 AWS 認證且無法連線至 Token 交換服務時，您可能會看到此錯誤。

請執行下列操作：

- 檢查組件是否聲明了令牌交換服務組件的依賴關係，`aws.greengrass.TokenExchangeService`. 如果沒有，請新增相依性並重新部署元件。
- 如果組件在 docker 中運行，請確保您應用正確的網絡設置和環境變量，根據在碼頭容器組件中使用 AWS憑據（Linux）.
- 如果組件是用 NodeJS 編寫的，請設置 DNS。 setDefaultResult訂購到**`ipv4first`**。
- 檢`/etc/hosts`查是否有以`::1`及包含開頭的項目`localhost`。移除項目，以查看是否造成元件在錯誤的位址連線至 Token 交換服務。

Received error when attempting to retrieve ECS metadata: Could not connect to the endpoint URL: "<tokenExchangeServiceEndpoint>"

當元件未執行 Token 交換服務且元件嘗試要求 AWS 認證時，您可能會看到此錯誤。

請執行下列操作：

- 檢查組件是否聲明了令牌交換服務組件的依賴關係，`aws.greengrass.TokenExchangeService`. 如果沒有，請新增相依性並重新部署元件。

- 檢查元件是否在其`install`生命週期中使用 AWS 認證。 AWS IoT Greengrass 不保證令牌交換服務在`install`生命週期中的可用性。更新元件，將使用 AWS 認證的程式碼移至`startup`或`run`生命週期，然後重新部署元件。

copyFrom: <configurationPath> is already a container, not a leaf

當您將組態值從容器類型 (清單或物件) 變更為非容器類型 (字串、數字或布林值) 時，您可能會看到此錯誤。請執行下列操作：

1. 檢查組件的方案，以查看其默認配置是否將該配置值設置為列表還是對象。如果是這樣，請移除或變更該組態值。
2. 建立部署以將該組態值重設為預設值。如需詳細資訊，請參閱 建立部署 及 更新零組件組態。

然後，您可以將該配置值設置為字符串，數字或布爾值。

com.aws.greengrass.componentmanager.plugins.docker.exceptions.DockerLoginError logging into the registry using credentials - 'The stub received bad data.'

當 Docker 應用程式管理員元件嘗試從 Amazon 彈性容器登錄 (Amazon ECR) 中的私有存放庫下載 Docker 映像時，您可能會在 Greengrass 核記錄中看到此錯誤。如果您使用 `wincred` Docker 認證協助程式 (`docker-credential-wincred`)，就會發生這個錯誤。因此，Amazon ECR 無法儲存登入登入資料。

執行下列其中一個動作：

- 如果您不使用 `wincred` Docker 憑證助手，請從核心設備中刪除`docker-credential-wincred`該程序。
- 如果您使用 `wincred` Docker 認證協助程式，請執行下列動作：
  1. 重新命名核心裝置上的`docker-credential-wincred`程式。`wincred`以 Windows 泊塢視窗認證協助程式的新名稱取代。例如，您可以將其重新命名為`docker-credential-wincredreal`。
  2. 更新 Docker 組態檔案 (`.docker/config.json`) 中的`credsStore`選項，以使用 Windows Docker 認證協助程式的新名稱。例如，如果將程式重新命名為`docker-credential-wincredreal`，請將選`credsStore`項更新為`wincredreal`。

```
{
  "credsStore": "wincredreal"
}
```

java.io.IOException: Cannot run program "cmd" ...: [LogonUser] The password for this account has expired.

當執行元件處理序的系統使用者 (例如) 的密碼過期時，您可能會在 Windows 核心裝置上看到這個錯誤。`ggc_user`因此， AWS IoT Greengrass Core 軟體無法以該系統使用者身分執行元件處理程序。

若要更新 Greengrass 系統使用者的密碼

1. 以系統管理員身分執行下列命令來設定使用者的密碼。以系統使用者取代 *ggc_user*，並以要設定的*##*取代密碼。

   ```
   net user ggc_user password
   ```

2. 使用此PsExec 公用程式將使用者的新密碼儲存在 LocalSystem 帳戶的認證管理員執行個體中。*##############*

   ```
   psexec -s cmd /c cmdkey /generic:ggc_user /user:ggc_user /pass:password
   ```

> **Tip**
>
> 視您的 Windows 組態而定，使用者的密碼可能會設定為在 future 的某個日期到期。為確保您的 Greengrass 應用程式繼續運作，請追蹤密碼何時到期，並在密碼到期之前進行更新。您也可以將使用者的密碼設定為永不過期。
>
> - 若要檢查使用者及其密碼何時到期，請執行下列命令。
>
>   ```
>   net user ggc_user | findstr /C:expires
>   ```
>
> - 若要將使用者的密碼設定為永不過期，請執行下列命令。
>
>   ```
>   wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
>   ```

- 如果您使用的是 Windows 10 或更新版本，而該wmic命令已被取代，請執行下列 PowerShell 命令。

```
Get-CimInstance -Query "SELECT * from Win32_UserAccount WHERE name =
 'ggc_user'" | Set-CimInstance -Property @{PasswordExpires="False"}
```

aws.greengrass.StreamManager: Instant exceeds minimum or maximum instant

當您將串流管理員 v2.0.7 升級到 v2.0.8 和 v2.0.11 之間的版本時，如果元件無法啟動，您可能會在串流管理員元件的記錄檔中看到下列錯誤。

```
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}
```

如果您已部署串流管理員 v2.0.7，且想要升級至更新版本，則必須直接升級至串流管理員 v2.0.12。如需串流管理員元件的詳細資訊，請參閱串流管理員。

核心裝置 Lambda 函數元件問題

解決核心裝置上的 Lambda 函數元件問題。

主題

- The following cgroup subsystems are not mounted: devices, memory
- ipc_client.py:64,HTTP Error 400:Bad Request, b'No subscription exists for the source <label-or-lambda-arn> and subject <label-or-lambda-arn>

The following cgroup subsystems are not mounted: devices, memory

在下列情況下，當您執行容器化 Lambda 函數時，可能會看到此錯誤：

- 核心裝置沒有為記憶體或裝置 cgroup 啟用 cgroup v1。
- 核心裝置已啟用 cgroup v2。Greengrass Lambda 函數需要 C 組 v1，C 組 v1 和 v2 是相互排斥的。

若要啟用 cgroup v1，請使用下列 Linux 核心參數來啟動裝置。

```
cgroup_enable=memory cgroup_memory=1 systemd.unified_cgroup_hierarchy=0
```

> **Tip**
>
> 在 Raspberry Pi 上，編輯`/boot/cmdline.txt`文件以設置設備的內核參數。

ipc_client.py:64,HTTP Error 400:Bad Request, b'No subscription exists for the source <label-or-lambda-arn> and subject <label-or-lambda-arn>

當您在 V2 核心裝置上執行 V1 Lambda 函數 (使用 AWS IoT Greengrass Core SDK)，但未在舊版訂閱路由器元件中指定訂閱時，可能會看到此錯誤。若要修正此問題，請部署並設定舊版訂閱路由器，以指定所需的訂閱。如需詳細資訊，請參閱 匯入 V1 Lambda 函數。

已停用元件版本

當核心裝置上的元件版本停止使用時，您可能會在 Personal Health Dashboard (PHD) 上看到通知。元件版本會在停產後 60 分鐘內將此通知傳送給您的 PHD。

若要查看需要修訂的部署，請使用執行下列操作 AWS Command Line Interface：

1. 執行下列命令以取得核心裝置的清單。

   ```
   aws greengrassv2 list-core-devices
   ```

2. 執行下列命令，從步驟 1 擷取每個核心裝置上的元件狀態。取代*coreDeviceName*為要查詢的每個核心裝置的名稱。

   ```
   aws greengrassv2 list-installed-components --core-device-thing-name coreDeviceName
   ```

3. 收集具有先前步驟中已停止使用的元件版本的核心裝置。
4. 執行下列命令，從步驟 3 擷取每個核心裝置的所有部署工作狀態。以要查詢的核心裝置名稱取*coreDeviceName*代。

   ```
   aws greengrassv2 list-effective-deployments --core-device-thing-name coreDeviceName
   ```

   回應包含核心裝置的部署工作清單。您可以修訂部署以選擇其他元件版本。如需如何修訂部署的相關資訊，請參閱版本修訂部署。

Greengrass 命令行界面問題

疑難排解使用 Greengr ass CLI 的問題。

主題

- java.lang.RuntimeException: Unable to create ipc client

java.lang.RuntimeException: Unable to create ipc client

當您執行 Greengrass CLI 命令，並指定與 AWS IoT Greengrass 核心軟體安裝位置不同的根資料夾時，您可能會看到這個錯誤。

執行下列其中一項動作來設定根路徑，並取代*/greengrass/v2*為 AWS IoT Greengrass Core 軟體安裝的路徑：

- 將 GGC_ROOT_PATH 環境變數設為 */greengrass/v2*。
- 如下列範例所示，將--ggcRootPath */greengrass/v2*引數新增至您的命令。

```
greengrass-cli --ggcRootPath /greengrass/v2 <command> <subcommand> [arguments]
```

AWS Command Line Interface 問題

疑 AWS CLI 難排解 AWS IoT Greengrass V2.

主題

- Error: Invalid choice: 'greengrassv2'

Error: Invalid choice: 'greengrassv2'

當您使用 AWS CLI (例如`aws greengrassv2 list-core-devices`) 執行 AWS IoT Greengrass V2 命令時，您可能會看到此錯誤。

此錯誤表示您擁有不支援的版本 AWS IoT Greengrass V2。 AWS CLI 若要 AWS IoT Greengrass V2 搭配使用 AWS CLI，您必須具有下列其中一個版本或更新版本：

- 最低版本 AWS CLI 第一版本：
- 最 AWS CLI 低版本：

> **Tip**
>
> 您可以運行以下命令來檢查您擁有 AWS CLI 的版本。
>
> ```
> aws --version
> ```

若要解決此問題，請更新 AWS CLI 至支援的更新版本 AWS IoT Greengrass V2。若要取得更多資訊，請參閱《AWS Command Line Interface 使用指南》 AWS CLI中的〈安裝、更新和解除安裝〉。

詳細的部署錯誤代碼

使用這些章節中的錯誤代碼和解決方案，以協助解決使用 Greengrass 核心 2.8.0 版或更新版本時的元件部署問題。

Greengrass 核心會將部署錯誤報告為最不特定於可用程式碼的階層。您可以使用此階層來協助找出部署錯誤的原因。例如，以下是一個可能的錯誤層次結構：

- 部署失敗 (_R)
  - 人工件下載錯誤
    - IO_ 錯誤
      - 磁碟空間關鍵

錯誤代碼被組織為類型。每種類型都代表可能發生的錯誤類別。AWS IoT Greengrass在主控台、API 和中報告這些錯誤類型AWS CLI。根據錯誤階層中報告的錯誤，可能會有多個錯誤類型。對於上述範例，傳回的錯誤類型為`DEVICE_ERROR`。

類型有：

- 權限錯誤 (_V)— 拒絕存取需要權限的作業。
- 請求錯誤— 由於部署文件中的問題而發生錯誤。
- 元件接收錯誤— 由於元件配方中的問題而發生錯誤。
- AW_ 元件錯誤— 啟動或移除時發生錯誤AWS提供的組件。
- 使用者元件錯誤— 啟動或移除使用者元件時發生錯誤。
- 元件錯誤— 啟動或移除元件時發生錯誤，但 Greengrass 核無法判斷元件是否為AWS提供的組件或用戶組件。
- 設備錯誤— 本機 I/O 發生錯誤或發生其他裝置錯誤。
- 依賴錯誤— 部署無法從 Amazon S3 下載成品或從 ECR 登錄提取映像。
- 錯誤— HTTP 要求發生錯誤。
- 網路錯誤— 裝置網路發生錯誤。
- 核錯誤— Greengrass 核無法找到組件或找不到活性細胞核版本.
- 伺服器錯誤— 伺服器回應要求時傳回 500 錯誤。
- 雲服務錯誤— 發生錯誤AWS IoT Greengrass雲端服務。
- 未知錯誤— 元件擲回未核取的例外狀況。

本節中的許多錯誤報告了AWS IoT Greengrass核心記錄檔。這些記錄會儲存在核心裝置的本機檔案系統上。有日誌AWS IoT Greengrass核心核心軟件和每個單獨的組件。如需存取記錄檔的資訊，請參閱存取檔案系統記錄。

許可錯誤

存取 (_拒絕)

你可能會得到這個錯誤AWS服務作業會傳回 403 錯誤，因為權限設定不正確。檢查更具體的錯誤代碼以獲取詳細信息。

獲取部署 _ 配置 _ 訪問 _ 拒絕

你可能會得到這個錯誤AWS IoT原則不允許呼叫`GetDeploymentConfiguration`操作。添加`greengrass::GetDeploymentConfiguration`核心裝置原則的權限。

獲取組件版本文件訪問被拒絕

核心設備時，您可能會收到此錯誤AWS IoT政策不允許`greengrass:GetComponentVersionArtifact`權限。將權限新增至核心裝置的原則。

解析組件候選人訪問被拒絕

核心設備時，您可能會收到此錯誤AWS IoT政策不允許`greengrass:ResolveComponentCandidates`權限。將權限新增至核心裝置的原則。

獲取 _ 電腦認證錯誤

當部署無法使用 ECR 中的私人登錄進行驗證時，您可能會收到此錯誤。檢查記錄檔是否有特定錯誤，然後再次嘗試部署。

用戶不授權用戶泊塢窗

當 Greengrass 使用者未獲授權使用 Docker 時，您可能會收到此錯誤。請確定您以 root 身分執行 Greengrass，或是將使用者新增至`docker`群組。然後再次嘗試部署。

拒絕訪問

當 Amazon S3 作業傳回 403 錯誤時，您可能會收到此錯誤。檢查任何其他錯誤代碼或日誌以獲取詳細信息。

S3_HEAD_對象存取_拒絕

當設備的令牌交換角色不允許AWS IoT Greengrass核心軟體，可從您在元件方案中指定的 S3 物件 URL 下載元件成品，或無法使用元件成品。檢查令牌交換角色是否允許`s3:GetObject`適用於成品可用且存在成品的 S3 物件 URL。

S3_GET_BUCKET_ 位置 _ 訪問被拒絕

當設備的令牌交換角色不允許`s3:GetBucketLocation`可取得成品之 Amazon S3 儲存貯體的許可。檢查設備是否允許該權限，然後再次嘗試部署。

S3_GET_ 對象 _ 訪問被拒絕

當設備的令牌交換角色不允許AWS IoT Greengrass核心軟體，可從您在元件方案中指定的 S3 物件 URL 下載元件成品，或無法使用元件成品。檢查令牌交換角色是否允許`s3:GetObject`適用於成品可用且存在成品的 S3 物件 URL。

請求錯誤

核缺少 _ 所需的能力

當部署中的核心版本無法執行要求的作業 (例如下載大型設定或設定 Linux 資源限制) 時，您可能會收到此錯誤。使用支援此作業的核心版本重試部署。

多重核 _ 解析錯誤 _

當部署嘗試部署多個核心元件時，您可能會收到此錯誤。檢查記錄檔以瞭解造成錯誤的原因，然後檢查核心軟體更新頁面，以查看問題是否已在較新版本的核心中修正，或是聯絡AWS Support。

元件循環相依性錯誤

當部署中的兩個元件彼此相依時，您可能會收到此錯誤。修訂元件設定，讓部署中的元件彼此不相依。

未授權的核心最小版本更新

當部署中的元件需要核心次要版本更新，但部署中未指定該版本時，您可能會收到此錯誤。這有助於減少依賴於不同版本的元件的意外次要版本更新。在部署中包含新的次要核心版本。

遺失碼頭 _ 應用程式管理員

當您部署 Docker 元件而不部署 Docker 應用程式管理員時，可能會收到此錯誤。請確定您的部署包含 Docker 應用程式管理員。

缺少令牌 _ 交換 _ 服務

當部署想要從私有 ECR 登錄下載 Docker 映像成品而不部署 Token 交換服務時，您可能會收到此錯誤。確保您的部署包括令牌交換服務。

元件版本需求不符合

當發生版本條件約束衝突或元件版本不存在時，您可能會收到此錯誤。如需詳細資訊，請參閱Error: com.aws.greengrass.componentmanager.exceptions.NoAvailableComponentVersionException: Failed to negotiate component <name> version with cloud and no local applicable version satisfying requirement <requirements>。

節流錯誤

你可能會得到這個錯誤AWS服務操作超過費率配額。重新嘗試部署。

衝突 (_R) 請求

你可能會得到這個錯誤AWS服務作業會傳回 409 錯誤，因為您的部署嘗試一次執行多項作業。重新嘗試部署。

未找到資源

你可能會得到這個錯誤AWS服務作業會傳回 404 錯誤，因為找不到資源。檢查記錄檔中是否有遺失的資源。

配置沒有有效

你可能會得到這個錯誤`posixUser`,`posixGroup`，或`windowsUser`指定執行元件的資訊無效。檢查使用者是否有效，然後重試部署。

不支援的區域 (_I)

當針對部署指定的區域不受支援時，您可能會收到此錯誤AWS IoT Greengrass。檢查「區域」，然後再次嘗試部署。

物聯網信用 _ 端點 _ 不有效

你可能會得到這個錯誤AWS IoT設定中指定的認證端點無效。請檢查端點，然後再試一次您的要求。

物聯網資料終點不有效

你可能會得到這個錯誤AWS IoT組態中指定的資料端點無效。請檢查端點，然後再試一次您的要求。

S3_头对象 _ 资源 _ 找不到

當元件成品在元件方案中指定的 S3 物件 URL 中無法使用時，您可能會收到此錯誤。檢查您已將成品上傳到 S3 儲存貯體，以及成品 URI 是否與儲存貯體中成品的 S3 物件 URL 相符。

S3_GET 儲存區位置 _ 資源 _ 找不到

找不到 Amazon S3 儲存貯體時，您可能會收到此錯誤。檢查值區是否存在，然後再次嘗試部署。

沒有找到 S3_ 獲取對象資源

當元件成品在元件方案中指定的 S3 物件 URL 中無法使用時，您可能會收到此錯誤。檢查您已將成品上傳到 S3 儲存貯體，以及成品 URI 是否與儲存貯體中成品的 S3 物件 URL 相符。

IO 映射錯誤

剖析部署文件或方案時發生 I/O 錯誤時，您可能會收到這個錯誤。檢查任何其他錯誤代碼或日誌以獲取詳細信息。

元件配方錯誤

接收剖析錯誤

無法剖析部署方案時，您可能會收到這個錯誤，因為方案的結構有錯誤。檢查方案格式是否正確，然後再次嘗試部署。

偏好元數據 _ 剖析錯誤

無法剖析從雲端下載的部署配方中繼資料時，您可能會收到此錯誤。請聯絡 AWS Support。

人工件無效 _ 無效

當配方中的成品 URI 格式不正確時，您可能會收到此錯誤。檢查記錄檔中是否有無效的 URI，更新方案中的 URI，然後再次嘗試部署。

S3_ 人工件 _ 不是有效的

當配方中的成品的 Amazon S3 URI 無效時，您可能會收到此錯誤。檢查記錄檔中是否有無效的 URI，更新方案中的 URI，然後再次嘗試部署。

碼頭工件沒有有效

當配方中的工件的 Docker URI 無效時，您可能會收到此錯誤。檢查記錄檔中是否有無效的 URI，更新方案中的 URI，然後再次嘗試部署。

空白人工件

當方案中未指定成品的 URI 時，您可能會收到此錯誤。檢查日誌中缺少 URI 的成品，更新方案中的 URI，然後再次嘗試部署。

空人工件方案

當未針對成品定義 URI 配置時，您可能會收到此錯誤。檢查記錄檔中是否有無效的 URI，更新方案中的 URI，然後再次嘗試部署。

不支援的人工件方案

當正在執行的核心版本不支援 URI 配置時，您可能會收到此錯誤。URI 無效，或者您需要更新核子核版本。如果 URI 無效，請檢查記錄檔中無效的 URI，更新方案中的 URI，然後再次嘗試部署。

接收遺失資訊清單

當清單部分未包含在配方中時，您可能會收到此錯誤。將資訊清單新增至方案，然後再次嘗試部署。

接收遺失 _ 人工雜湊演算法

如果在沒有雜湊演算法的配方內指定不是本機的成品，您可能會收到這個錯誤。將演算法新增至成品，然後再次嘗試要求。

人工因素 _ 檢查和不相符

當下載的成品摘要與方案中指定的摘要不同時，您可能會收到此錯誤。請確定方案包含正確的摘要，然後再次嘗試部署。如需詳細資訊，請參閱 Error: com.aws.greengrass.componentmanager.exceptions.ArtifactChecksumMismatchException: Integrity check for downloaded artifact failed. Probably due to file corruption.。

組件依賴關係無效

當部署方案中指定的相依性類型無效時，您可能會收到這個錯誤。請檢查食譜，然後再試一次。

配置插值錯誤

插值 recipe 變數時，您可能會看到這個錯誤。檢查日誌以獲取詳細信息。

IO 映射錯誤

剖析部署文件或方案時發生 I/O 錯誤時，您可能會收到這個錯誤。檢查任何其他錯誤代碼或日誌以獲取詳細信息。

AWS元件錯誤、使用者元件錯誤、元件錯誤

當元件發生問題時，會傳回下列錯誤碼。報告的實際錯誤類型取決於引發錯誤的特定組件。如果 Greengrass 核標識該組件作為一個提供AWS IoT Greengrass，它返回AWS_COMPONENT_ERROR。如果該組件被標識為一個用戶組件，Greengrass 核返回USER_COMPONENT_ERROR。如果格林格拉斯核無法分辨，它返回COMPONENT_ERROR。

組件更新錯誤

當元件在部署期間未更新時，您可能會收到此錯誤。檢查任何其他錯誤代碼或檢查日誌以查看導致錯誤的原因。

元件已損壞

當元件在部署期間中斷時，您可能會收到此錯誤。檢查元件記錄檔以取得錯誤詳細資料，然後再次嘗試部署。

移除元件錯誤

當核心無法在部署期間移除元件時，您可能會收到此錯誤。請檢查記錄檔中的錯誤詳細資料，然後再次嘗試部署。

組件引導程序超時

當元件的啟動程序工作花費的時間超過設定的逾時時間時，您可能會收到此錯誤。增加逾時或減少啟動程序工作的執行時間，然後再次嘗試部署。

組件引導程序錯誤

當元件的啟動程序工作發生錯誤時，您可能會收到此錯誤。檢查記錄檔以取得錯誤詳細資料，然後再次嘗試部署。

零組件組態無效

當核心無法驗證元件的已部署組態時，您可能會收到此錯誤。檢查記錄檔以取得錯誤詳細資料，然後再次嘗試部署。

裝置錯誤

IO_ 寫入錯誤

寫入檔案時可能會出現此錯誤。檢查日誌以獲取詳細信息。

讀取錯誤

從檔案讀取時，您可能會收到此錯誤。檢查日誌以獲取詳細信息。

磁碟空間關鍵

當沒有足夠的磁碟空間無法完成部署要求時，您可能會收到這個錯誤。您必須至少有 20 Mb 的可用空間，或足以容納較大的成品。請釋放一些磁碟空間，然後重試部署。

IO_ 文件屬性錯誤

當無法從檔案系統擷取現有檔案大小時，您可能會收到此錯誤。檢查日誌以獲取詳細信息。

設置權限錯誤

當無法在下載的成品或成品目錄上設定權限時，您可能會收到此錯誤。檢查日誌以獲取詳細信息。

解壓縮錯誤

無法解壓縮成品時，您可能會收到此錯誤。檢查日誌以獲取詳細信息。

本地接收 _ 找不到

當找不到 recipe 檔案的本機副本時，您可能會收到此錯誤。請再次嘗試部署。

本地 (_R) 已損毀

當食譜的本地副本自下載以來發生變化時，您可能會收到此錯誤。刪除方案的現有副本，然後再次嘗試部署。

找不到本地信息中繼資料

找不到 recipe 中繼資料檔案的本機副本時，您可能會收到此錯誤訊息。請再次嘗試部署。

啟動目錄已損壞

當用於啟動 Greengrass 核的目錄時，您可能會遇到此錯誤 (`/greengrass/v2/alts/current`) 自上次啟動原子核以來已被修改。重新啟動核心，然後重試部署。

哈希演算法不可用

當設備的 Java 發行版不支持所需的哈希算法或組件配方中指定的哈希算法無效時，您可能會遇到此錯誤。

設備配置沒有可用的文件下載

當裝置組態中出現錯誤，導致部署無法從 Amazon S3 或 Greengrass 雲端下載成品時，您可能會收到此錯誤。檢查記錄檔是否有特定組態錯誤，然後重試部署。

相依性錯誤

碼頭錯誤

在提取 Docker 圖像時，您可能會遇到此錯誤。檢查任何其他錯誤代碼或日誌以獲取詳細信息。

碼頭服務 _ 不可用

當 Greengrass 無法登錄到 Docker 註冊表時，您可能會收到此錯誤。檢查記錄檔是否有特定錯誤，然後再次嘗試部署。

碼頭登錄錯誤

登入 Docker 時發生未預期的錯誤時，您可能會收到此錯誤。檢查記錄檔是否有特定錯誤，然後再次嘗試部署。

碼頭拉錯誤

當從註冊表中提取 Docker 映像時發生意外錯誤時，您可能會收到此錯誤。檢查記錄檔是否有特定錯誤，然後再次嘗試部署。

碼頭圖像無效

當請求的 Docker 映像不存在時，您可能會收到此錯誤。檢查記錄檔是否有特定錯誤，然後再次嘗試部署。

碼頭圖像查詢錯誤

當查詢 Docker 尋找可用映像檔時，發生未預期的失敗時，您可能會收到此錯誤。檢查記錄檔中是否有特定錯誤，然後再次嘗試部署。

錯誤

下載 Amazon S3 成品時，您可能會收到此錯誤訊息。檢查任何其他錯誤代碼或日誌以獲取詳細信息。

沒有找到 S3 資源

當 Amazon S3 作業傳回 404 錯誤時，您可能會收到此錯誤。檢查任何其他錯誤代碼或日誌以獲取詳細信息。

請求

當 Amazon S3 作業傳回 400 錯誤時，您可能會收到這個錯誤。檢查記錄檔是否有特定錯誤，然後再試一次要求。

錯誤

請求錯誤

發出 HTTP 請求時發生錯誤時，您可能會收到此錯誤。檢查日誌中的特定錯誤。

下載 _ 部署 _ 文件 _ 錯誤

下載部署文件時發生 HTTP 錯誤時，您可能會收到這個錯誤。檢查日誌中是否有特定的 HTTP 錯誤。

獲取 _ 格蘭文件 _ 大小 _ 錯誤

當取得公用元件加工品的大小時，發生 HTTP 錯誤時，您可能會收到這個錯誤。檢查日誌中是否有特定的 HTTP 錯誤。

下載 _ 格蘭文件 _ 錯誤

下載公用元件加工品時，發生 HTTP 錯誤時，您可能會收到這個錯誤。檢查日誌中是否有特定的 HTTP 錯誤。

網路錯誤

網路錯誤

部署期間發生連線問題時，您可能會收到此錯誤。檢查裝置與網際網路的連線，然後再次嘗試部署。

核誤差

壞請求

你可能會得到這個錯誤AWS雲操作返回一個 400 錯誤。檢查日誌以查看哪個 API 導致錯誤，然後檢查核心軟件更新頁面，以查看問題是否已在較新版本的核心中更正，或聯繫AWS Support。

核心版本 _ 未找到

當核心設備找不到活動核心的版本時，您可能會收到此錯誤。檢查記錄檔以瞭解造成錯誤的原因，然後檢查核心軟體更新頁面，以查看問題是否已在較新版本的核心中修正，或是聯絡AWS Support。

核重新啟動失敗

在任何需要核心重新啟動的部署期間，如果核心未重新啟動，則可能會收到此錯誤。檢查載入程式記錄檔以瞭解造成錯誤的原因，然後檢查核心軟體更新頁面，看看問題是否已在較新版本的核心中得到修正，或是聯絡AWS Support。

未找到已安裝元件

當核心找不到已安裝的元件時，您可能會收到此錯誤。檢查記錄檔以瞭解造成錯誤的原因，然後檢查核心軟體更新頁面，以查看問題是否已在較新版本的核心中修正，或是聯絡AWS Support。

部署 _ 文件 _ 無效

當設備收到無效的部署文檔時，您可能會收到此錯誤。檢查任何其他錯誤代碼或檢查日誌以查看導致錯誤的原因。

空白部署請求

當設備收到空的部署請求時，您可能會收到此錯誤。檢查記錄檔以瞭解造成錯誤的原因，然後檢查核心軟體更新頁面，以查看問題是否已在較新版本的核心中修正，或是聯絡AWS Support。

部署 _ 文件 _ 剖析錯誤

當部署要求格式不符合預期的格式時，您可能會收到這個錯誤。檢查記錄檔以瞭解造成錯誤的原因，然後檢查核心軟體更新頁面，以查看問題是否已在較新版本的核心中修正，或是聯絡AWS Support。

部署元件中繼資料不是有效的

當部署要求包含無效的元件中繼資料時，您可能會收到這個錯誤。檢查記錄檔以瞭解造成錯誤的原因，然後檢查核心軟體更新頁面，以查看問題是否已在較新版本的核心中修正，或是聯絡AWS Support。

啟動目錄已損壞

當您將 Greengrass 裝置從一個物件群組移至另一個物件群組，然後回到具有需要 Greengrass 重新啟動部署的原始群組時，可能會收到此錯誤。若要解決此錯誤，請在裝置上重新建立 Greengrass 的啟動目錄。

如需詳細資訊，請參閱Error: com.aws.greengrass.deployment.exceptions.DeploymentException: Unable to process deployment. Greengrass launch directory is not set up or Greengrass is not set up as a system service。

伺服器錯誤

伺服器錯誤

你可能會得到這個錯誤AWS服務作業會傳回 500 錯誤，因為服務目前無法處理要求。稍後重試部署。

S3_ 伺服器錯誤

當 Amazon S3 作業傳回 500 錯誤時，您可能會收到這個錯誤。檢查任何其他錯誤代碼或日誌以獲取詳細信息。

雲端服務錯誤

解析組件候選人問題

當 Greengrass 雲端服務傳送不相容的回應給`ResolveComponentCandidates`操作。檢查記錄檔以瞭解造成錯誤的原因，然後檢查核心軟體更新頁面，以查看問題是否已在較新版本的核心中修正，或是聯絡AWS Support。

超過部署 _ 文件大小 _

當要求的部署文件超過大小配額上限時，您可能會收到這個錯誤。減少部署文件的大小，然後再次嘗試部署。

找不到大小 _ 文件 _ 大小 _

當 Greengrass 無法獲取公共組件成品的大小時，您可能會收到此錯誤。檢查記錄檔以瞭解造成錯誤的原因，然後檢查核心軟體更新頁面，以查看問題是否已在較新版本的核心中修正，或是聯絡AWS Support。

部署 _ 文件 _ 無效

當設備收到無效的部署文檔時，您可能會收到此錯誤。檢查任何其他錯誤代碼或檢查日誌以查看導致錯誤的原因。

空白部署請求

當設備收到空的部署請求時，您可能會收到此錯誤。檢查記錄檔以瞭解造成錯誤的原因，然後檢查核心軟體更新頁面，以查看問題是否已在較新版本的核心中修正，或是聯絡AWS Support。

部署 _ 文件 _ 剖析錯誤

當部署要求格式不符合預期的格式時，您可能會收到這個錯誤。檢查記錄檔以瞭解造成錯誤的原因，然後檢查核心軟體更新頁面，以查看問題是否已在較新版本的核心中修正，或是聯絡AWS Support。

部署元件中繼資料不是有效的

當部署要求包含無效的元件中繼資料時，您可能會收到這個錯誤。檢查記錄檔以瞭解造成錯誤的原因，然後檢查核心軟體更新頁面，以查看問題是否已在較新版本的核心中修正，或是聯絡AWS Support。

一般錯誤

這些一般錯誤沒有關聯的錯誤類型。

部署中斷 (_C)

因為核心關閉或其他外部事件而無法完成部署時，您可能會收到這個錯誤。檢查任何其他錯誤代碼或日誌以獲取詳細信息。

人工件下載錯誤

下載成品時發生問題時，您可能會收到此錯誤。檢查任何其他錯誤代碼或日誌以獲取詳細信息。

沒有可用的元件版本 (_R)

當元件版本不存在於雲端或本機時，或者發生相依性解決衝突時，您可能會收到這個錯誤。檢查任何其他錯誤代碼或日誌以獲取詳細信息。

元件封裝載入錯誤

當處理下載的成品時發生錯誤，您可能會收到此錯誤。檢查任何其他錯誤代碼或日誌以獲取詳細信息。

雲錯誤

當發生錯誤調用時，您可能會收到此錯誤AWS服務 API。檢查任何其他錯誤代碼或日誌以獲取詳細信息。

IO_ 錯誤

部署期間發生 I/O 錯誤時，您可能會收到此錯誤。檢查任何其他錯誤代碼或日誌以獲取詳細信息。

組件更新錯誤

當元件在部署期間未更新時，您可能會收到此錯誤。檢查任何其他錯誤代碼或檢查日誌以查看導致錯誤的原因。

未知的錯誤

部署失敗 (_R)

當部署失敗時，您可能會收到這個錯誤，因為未經核取的例外狀況擲回。檢查記錄檔以瞭解造成錯誤的原因，然後檢查核心軟體更新頁面，以查看問題是否已在較新版本的核心中修正，或是聯絡 AWS Support。

部署 _ 類型 _ 無效

當部署類型無效時，您可能會收到此錯誤。檢查記錄檔以瞭解造成錯誤的原因，然後檢查核心軟體更新頁面，以查看問題是否已在較新版本的核心中修正，或是聯絡AWS Support。

詳細的元件狀態代碼

使用 Greengrass 核心 2.8.0 版或更新版本時，請使用這些區段中的狀態碼和解決方案來協助解決元件的問題。

本主題中的許多狀態都會報告AWS IoT Greengrass核心記錄檔中的其他資訊。這些記錄會儲存在核心裝置的本機檔案系統上。每個單獨的元件都有記錄檔。如需存取記錄檔的資訊，請參閱存取檔案系統記錄。

安裝錯誤

當執行安裝指令碼時發生錯誤時，您可能會收到此訊息。錯誤碼會報告在元件記錄檔中。檢查安裝指令碼是否有錯誤，然後再次部署元件。

安裝配置不有效

因為配方的`Install`區段無效而無法完成元件安裝時，您可能會收到此錯誤。檢查方案的安裝部分是否有錯誤，然後再次嘗試部署。

安裝錯誤

安裝元件期間發生 I/O 錯誤時，您可能會收到此訊息。請檢查元件錯誤的詳細資訊。

安裝遺失 _ 預設 _ 執行

安裝元件時，AWS IoT Greengrass無法判斷要使用的使用者或群組時，您可能會收到此錯誤訊息。請檢查以確定您的安裝配方`runWith`章節包含有效的使用者或群組。

安裝逾時

當安裝指令碼未在設定的逾時期間內完成時，您可能會收到此錯誤。您可以增加方案`Install`區段中指定的`Timeout`期間，或修改您的安裝指令碼，以便在設定的逾時時間內完成。

啟動錯誤

當運行啟動腳本時發生錯誤時，您可能會得到這個問題。錯誤碼會報告在元件記錄檔中。檢查安裝指令碼是否有錯誤，然後再次部署元件。

啟動配置無效

因為配方的`Startup`區段無效而無法完成元件安裝時，您可能會收到此錯誤。檢查方案的啟動部分是否有錯誤，然後再次嘗試部署。

啟動 IO_ 錯誤

當元件啟動期間發生 I/O 錯誤時，您可能會收到此問題。請檢查元件錯誤的詳細資訊。

啟動缺少 _ 預設 _ 執行

當無法判斷執行元件時要使用的使用者或群組時，您AWS IoT Greengrass可能會收到此錯誤。請檢查以確定啟動方案中的runWith區段包含有效的使用者或群組。

啟動逾時

當啟動指令碼未在設定的逾時期間內完成時，您可能會收到此錯誤。您可以增加方案Startup區段中指定的Timeout期間，或修改您的啟動指令碼，以在設定的逾時內完成。

運行錯誤

當執行元件指令碼時發生錯誤時，您可能會得到這個問題。錯誤碼會報告在元件記錄檔中。檢查運行腳本是否有錯誤，然後再次部署您的組件。

運行 _ 缺少 _ 默認 _ 運行

當無法判斷執行元件時要使用的使用者或群組時，您AWS IoT Greengrass可能會收到此錯誤。請檢查以確定執行方案的runWith區段包含有效的使用者或群組。

運行配置不有效

當組件無法運行時，您可能會收到此錯誤，因為配方的Run部分無效。檢查方案的運行部分是否有錯誤，然後再次嘗試部署。

運行 _ IO_ 錯誤

當元件執行時發生 I/O 錯誤時，您可能會收到此訊息。請檢查元件錯誤的詳細資訊。

執行逾時

當執行指令碼未在設定的逾時期限內完成時，您可能會收到此錯誤。您可以增加方案Run區段中指定的Timeout期間，或修改執行指令碼，以在設定的逾時內完成。

關機 (_R) 錯誤

當關閉組件腳本時發生錯誤時，您可能會得到這個問題。錯誤碼會報告在元件記錄檔中。檢查關機指令碼是否有錯誤，然後再次部署元件。

關機逾時 (_I)

當關機指令碼未在設定的逾時期間內完成時，您可能會收到此錯誤。您可以增加方案Shutdown區段中指定的Timeout期間，或修改執行指令碼，以在設定的逾時內完成。

標記您的 AWS IoT Greengrass Version 2 資源

使用標籤，您可以在 AWS IoT Greengrass 中組織和管理您的資源。您可以使用標籤為資源指派中繼資料，也可以在 IAM 政策中使用標籤來定義資源的條件式存取。

> **Note**
>
> 目前，AWS IoT 帳單群組或成本分配報告不支援 Greengrass 資源標籤。

在 AWS IoT Greengrass V2 中使用標籤

您可以使用標籤，依據用途、擁有者、環境或任何其他使用案例的分類來分類 AWS IoT Greengrass 資源。當您擁有許多相同類型的資源時，標籤可協助您更輕鬆地識別特定資源。

每個標籤皆包含由您定義的一個金鑰與一個選用值。例如，您可以為您核心裝置定義一組標籤，協助您追蹤各個裝置的客戶。我們建議您為每種資源類型建立符合您需求的一組標籤金鑰。使用一致的標籤鍵組，可讓您的資源管理更輕鬆。

使用標籤AWS Management Console

中的標籤編輯器可讓您集中、整合地建立和管理標籤，協助您為各種AWS服務的資源建立與管理標籤。AWS Management Console若要取得更多資訊，請參閱AWS Resource Groups使用者指南中的標籤編輯器。

使用AWS IoT Greengrass V2 API 進行標記

您也可以使用AWS IoT Greengrass V2 API 來使用標籤。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AWS 一般參考。

- 若要在建立資源時新增標籤，請在資源的 `tags` 屬性中定義標籤。
- 若要將標籤新增至現有資源，或更新標籤值，請使用此TagResource作業。
- 若要移除標籤，請使用UntagResource操作。
- 若要擷取與資源相關聯的標籤，請使用ListTagsForResource作業，或描述資源並檢查其`tags`屬性。

下表列出您可以使用AWS IoT Greengrass V2 API 標記的資源及其對應的`Create` AND`Describe`或`Get`作業。

可標記的 AWS IoT Greengrass V2 資源

資源	建立操作	描述或獲取操作
核心裝置	無。在裝置上執行AWS IoT Greengrass Core 軟體以建立核心裝置。	GetCoreDevice
元件	CreateComponentVersion	DescribeComponent, GetComponent
部署	CreateDeployment	GetDeployment

使用下列操作檢視並管理支援標記之資源的標籤：

- TagResource— 將標籤新增至資源，或更新現有標籤的值。
- 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。新值會取代先前的值。您可以將數值設為空白字串，但您無法將數值設為 null。

在您刪除資源時，也會刪除與該資源建立關聯的標籤。

搭配 IAM 政策使用標籤

在 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。

> **Note**
>
> 如果您使用標籤來允許或拒絕使用者存取資源，您應該拒絕使用者為相同資源新增或移除這些標籤的能力。否則，使用者可能透過修改標籤來避開您的限制，並取得資源的存取。

您可以在原則陳述式的`Condition`元素 (也稱為`Condition`區塊) 中使用下列條件內容索引鍵和值。

`greengrassv2:ResourceTag/tag-key: tag-value`

允許或拒絕資源對具有特定標籤之資源的動作。

`aws:RequestTag/tag-key: tag-value`

建立或修改可加標籤資源時，需要使用或不使用特定標籤。

`aws:TagKeys: [tag-key, ...]`

建立或修改可加標籤資源時，需要使用或不使用特定的標籤鍵集。

> **Note**
>
> IAM 政策中的條件內容索引鍵和值僅適用於具有可標記資源作為必要參數的動作。例如，您可以設定以標籤為基礎的條件式存取ListCoreDevices。

如需詳細AWS資訊，請參閱 IAM 使用者指南中的使用資源標籤控制資源存取和 IAM JSON 政策參考。

透過 AWS CloudFormation 建立 AWS IoT Greengrass 資源

AWS IoT Greengrass 已與 AWS CloudFormation 整合，這項服務可協助您建立 AWS 資源的模型和設定，以減少建立和管理資源和基礎設施的時間。您創建一個描述所有的模板AWS您想要的資源 (例如元件版本和部署)，以及AWS CloudFormation為您佈建和配置這些資源。

當您使用 AWS CloudFormation 時，您可以重複使用您的範本，重複、一致的設定您的 AWS IoT Greengrass 資源。只需描述一次您的資源，即可在多個 AWS 帳戶 與區域內重複佈建相同資源。

AWS IoT Greengrass 和 AWS CloudFormation 範本

若要佈建和配置 AWS IoT Greengrass 與相關服務的資源，您必須了解 AWS CloudFormation 範本。範本是以 JSON 或 YAML 格式化的文本檔案。而您亦可以透過這些範本的說明，了解欲在 AWS CloudFormation 堆疊中佈建的資源。如果您不熟悉 JSON 或 YAML，您可以使用 AWS CloudFormation Designer 協助您開始使用 AWS CloudFormation 範本。如需詳細資訊，請參閱 AWS CloudFormation 使用者指南中的什麼是 AWS CloudFormation Designer？。

AWS IoT Greengrass支援在中建立元件版本和部署AWS CloudFormation。如需詳細資訊 (包括元件版本和部署的 JSON 和 YAML 範本範例)，請參閱AWS IoT Greengrass資源類型參考在AWS CloudFormation使用者指南。

ComponentVersion 範本範例

以下是簡單組件的版本的 YAML 模板。JSON 方案包括換行以方便閱讀。

```
Parameters:
  ComponentVersion:
    Type: String
Resources:
  TestSimpleComponentVersion:
    Type: AWS::GreengrassV2::ComponentVersion
    Properties:
      InlineRecipe: !Sub
        - "{\n
            \"RecipeFormatVersion\": \"2020-01-25\",\n
            \"ComponentName\": \"component1\",\n
            \"ComponentVersion\": \"${ComponentVersion}\",\n
            \"ComponentType\": \"aws.greengrass.generic\",\n
            \"ComponentDescription\": \"This\",\n
```

```
            \"ComponentPublisher\": \"You\",\n
            \"Manifests\": [\n
            {\n
              \"Platform\": {\n
              \"os\": \"darwin\"\n
            },\n
            \"Lifecycle\": {},\n
            \"Artifacts\": []\n
           },\n
           {\n
             \"Lifecycle\": {},\n
             \"Artifacts\": []\n
           }\n
          ],\n
          \"Lifecycle\": {\n
          \"install\": {\n
            \"script\": \"yuminstallpython\"\n
          }\n
         }\n
        }"
        - { ComponentVersion: !Ref ComponentVersion }
```

部署範本範例

以下是定義部署的簡單範本的 YAML 檔案。

```
Parameters:
  ComponentVersion:
    Type: String
  TargetArn:
    Type: String
Resources:
  TestDeployment:
    Type: AWS::GreengrassV2::Deployment
    Properties:
      Components:
        component1:
          ComponentVersion: !Ref ComponentVersion
      TargetArn: !Ref TargetArn
      DeploymentName: CloudFormationIntegrationTest
      DeploymentPolicies:
        FailureHandlingPolicy: DO_NOTHING
        ComponentUpdatePolicy:
```

```
          TimeoutInSeconds: 5000
          Action: SKIP_NOTIFY_COMPONENTS
        ConfigurationValidationPolicy:
          TimeoutInSeconds: 30000
Outputs:
  TestDeploymentArn:
    Value: !Sub
      - arn:${AWS::Partition}:greengrass:${AWS::Region}:${AWS::AccountId}:deployments:
${DeploymentId}
      - DeploymentId: !GetAtt TestDeployment.DeploymentId
```

進一步了解 AWS CloudFormation

若要進一步了解 AWS CloudFormation，請參閱下列資源：

- AWS CloudFormation
- 《AWS CloudFormation 使用者指南》
- AWS CloudFormation API 參考
- 《AWS CloudFormation 命令列介面使用者指南》

開放原始碼AWS IoT Greengrass核心軟體

邊AWS IoT Greengrass Version 2緣運行時（核）和AWS IoT Greengrass核心軟件的其他組件是開源的。這表示您可以檢閱程式碼，對與應用程式的互動進行疑難排解。您也可以自訂和擴充 AWS IoT Greengrass Core 軟體，以滿足您的特定軟體和硬體需求。

如需有關AWS IoT Greengrass核心軟體之開放原始碼儲存庫的詳細資訊，請參閱上的 aws-greengrass 組織。 GitHub您對開放原始碼軟體的使用受到相應 GitHub 儲存庫中的開放原始碼授權條款管理。

您對不受開放原始碼授權約束的AWS IoT Greengrass核心軟體和元件的使用受 AWS Greengrass 核心軟體授權管理。

AWS IoT Greengrass V2 開發人員指南的文件歷史記錄

下表說明此版本的文件 AWS IoT Greengrass Version 2。

- 應用程式介面版本：

變更	描述	日期
AWS IoT Device Tester 版本與 GGV2Q 版本發布	已提供適用於 AWS IoT Greengrass V2 的 IDT 版本 4.9.3。此版本包括 AWS IoT Greengrass V2 合格套件 (GGV2Q) v2.5.3 版，並支援 Greengrass 核的版本 2.12.0、2.11.0、2.10.0、2.9.5。	2024年4月5 日
Greengrass CLI 版本發布	綠色 CLI 組件 v2.12.4 是可用的。	2024年4月2日
AWS IoT Greengrass 核心 v2.12.4 軟體更新	此版本提供了 Greengrass 核組件的 2.12.4 版本和更新提供的組件。 AWS	2024年4月2日
影子管理器 v2.3.7 發布	可以使用陰影管理器 v2.3.7。此版本修正了陰影管理員在陰影管理員同步期間定期記錄`NullPointerException` 錯誤的問題。	2024年3月27日
平均 MQTT 3.1.1 經紀商 v2.3.6 發布	轉換 MQTT 3.1.1 經紀人組件 v2.3.6 是可用的。此版本包括一般錯誤修復和改進。	2024年3月27日
本地調試控制台 v2.4.2 發布	本地調試控制台組件 v2.4.2 可用。此版本包括一般錯誤修復和改進。	2024年3月27日

Lambda 管理器 v2.3.3 發布	Lambda 管理器組件 v2.3.3 可用。此版本包括一般錯誤修復和改進。	2024年3月27日
IP 檢測器 v2.1.9 發布	IP 檢測器組件 v2.1.9 可用。此版本會將取得的 IP 步驟調整為僅在偵錯記錄檔層級傳送記錄檔。	2024年3月27日
AWS IoT 發布車隊配置插件 v1.2.1	AWS IoT 車隊配置插件 v1.2.1 可用。此版本修正了叢集佈建外掛程式在 Greengrass 核心啟動期間處於離線狀態的問題。車隊佈建外掛程式現在可以無限期地重試 MQTT 連線呼叫。	2024年3月27日
AWS IoT Greengrass 核心版本 v2.12.3 軟體更新	此發行版本提供 Greengrass 核元件的 2.12.3 版，以及更新提供的元件。 AWS	2024年3月27日
Greengrass CLI 版本 2.12.3 發布	綠色 CLI 組件 v2.12.3 是可用的。	2024年3月25日
AWS IoT Device Tester v4.9.2 與 GGV2Q 版本發布	已提供適用於 AWS IoT Greengrass V2 的 IDT 版本 4.9.2。此版本包括 AWS IoT Greengrass V2 合格套件 (GGV2Q) v2.5.2，並支援 Greengrass 核的版本 2.12.0、2.11.0、2.10.0、2.9.5。	2024年3月18日
Lookout for Vision 邊緣代理 v1.2.0 發布	Lookout for Vision 邊緣代理 v1.2.0 是可用的。	2024年3月11日

AWS IoT Greengrass 核心版軟體更新	此版本提供 Greengrass 核心元件的 2.12.2 版，以及更新提供的元件。 AWS	2024年2月15日
影子管理器 v2.3.6 發布	陰影管理器 v2.3.6 可用。此版本修正了在裝置離線時透過 AWS 雲端 更新刪除的陰影屬性在重新取得連線後，仍會繼續存在於本機陰影中的問題。	2024年2月14日
Lambda 發射器 v2.0.13 發布	Lambda 啟動器元件的 2.0.13 版本可供使用。此版本包含一般錯誤修正和改良功能。	2024年2月14日
磁盤後台處理程序 v1.0.3 發布	磁盤後台處理程序組件 v1.0.3 可用。此版本透過重複使用資料庫連線來改善效能。	2024年2月14日
Lookout for Vision 邊緣劑 v1.1.9 發布	Lookout for Vision 邊緣代理 v1.1.9 可用。	2024年1月17日
Greengrass 開發工具包 CLI v1.6.2	Greengrass 開發工具包 CLI 的 1.6.2 版本可用。此版本修復了 Windows gradlew.bat 由於相對路徑而無法正常工作的問題。此版本還包含其他改進。	2024年1月16日
新 CloudTrail 資料事件	您現在可以記錄資 AWS CloudTrail 料事件以取得資源作業的相關資訊，例如取得元件或部署的組態。使用這些事件來深入了解 Greengrass 裝置的操作情況。	2023 年 12 月 20 日
Lookout for Vision 邊緣代理 v1.1.8 發布	Lookout for Vision 邊緣代理 v1.1.8 可用。	2023 年 12 月 12 日

流管理器 v2.1.12 發布	流管理器 v2.1.12 現在可用。此版本會變更 Greengrass 用來為服務呼叫選取一組認證的 AWS 順序。	2023 年 12 月 8 日
MQTT 橋接器 v2.3.1 發布	MQTT 橋接器 v2.3.1 是可用的。此版本修正了本機 MQTT 用戶端進入中斷連線迴圈的罕見問題。	2023 年 12 月 8 日
磁盤後台處理程序 v1.0.2 發布	磁盤後台處理程序組件 v1.0.2 可用。此版本修正了 MQTT 郵件格式欄位在特定情況下不會持續存在的問題。	2023 年 12 月 8 日
客戶端設備身份驗證組件 v2.4.5 發布	客戶端設備身份驗證組件 v2.4.5 可用。此版本在選取規則中新增對物件名稱結尾萬用字元的支援，並修正了在某些情況下憑證不會以新連線資訊更新的問題。	2023 年 12 月 8 日
AWS IoT Greengrass 核心軟體更新	此版本提供 Greengrass 核心元件的 2.12.1 版，以及更新提供的元件。 AWS	2023 年 12 月 8 日
Greengrass 開發工具包 CLI 版本 1.6.1	Greengrass 開發工具包 CLI 的 1.6.1 版本可用。此版本包含錯誤修復和改進。	2023 年 12 月 6 日
配方驗證	新增配方驗證功能，可在建立元件版本時驗證元件配方。	2023 年 11 月 16 日
出版商支援的元件	AWS IoT Greengrass 現在提供發行商支援的元件。這些組件由第三方供應商開發，提供和服務。	2023 年 11 月 16 日

Greengrass 測試框架 v1.2.0 發布	綠色測試框架 v1.2.0 是可用的。	2023 年 11 月 15 日
Greengrass 開發工具包 CLI 版本 1.6.0	Greengrass 開發工具包 CLI 的 1.6.0 版本可用。此版本會在和指令期間針對 Greengrass 方案結構描述新增配方驗證檢查。`component build` `component publish`此更新可協助開發人員在元件建立程序的早期元件配方中找出可行動的問題。此版本還將可信度測試套件添加到模板中，該模板可以通過`test-e2e init`命令下拉。此置信度測試套件包括八個通用測試，可以使用和擴展以滿足基本組件測試需求。	2023 年 11 月 15 日
AWS IoT Device Tester v4.9.1 支持 Greengrass 核版本 2.12.0	對於 AWS IoT Greengrass V2 的 IDT 4.9.1 版本現在支持 Greengrass 核 2.12.0 版本。	2023 年 11 月 7 日
AWS IoT Greengrass 核心 v2.12.0 軟體更新	此版本提供了 Greengrass 核組件的 2.12.0 版本和更新提供的組件。 AWS	2023 年 11 月 7 日
在 VPC 中操作 Greengrass 核心設備	可以在 VPC 中操作 Greengrass 核心設備。此功能使您可以在沒有公共 Internet 訪問的情況下在 VPC 中執行部署。	2023 年 11 月 3 日
Greengrass CLI 版本 12.0 發布	綠色 CLI 組件 v2.12.0 是可用的。	2023 年 10 月 30 日

流管理器 v2.1.10 發布	流管理器 v2.1.10 現在可用。此版本修正了 HTTPS 代理伺服器設定不信任 Greengrass CA 憑證鏈結的問題。	2023 年 10 月 26 日
Lambda 發射器 v2.0.12 發布	Lambda 啟動器元件的 2.0.12 版本可供使用。此版本修正了如果前一個程序未正確停止，Lambda 啟動器可能會擲回錯誤的問題。	2023 年 10 月 26 日
Greengrass 開發工具包 CLI V1.5.0	Greengrass 開發工具包 CLI 的 1.5.0 版本可用。此版本會更新`excludes`構建選項識別的模`build_system` 式`zip`。此版本現在將根據通配符符識別匹配路徑名的 glob 模式。這會啟用要排除哪些目錄的自訂規格。	2023 年 10 月 26 日
Lookout for Vision 邊緣代理 v1.1.7 發布	Lookout for Vision 邊緣代理 v1.1.7 可用。	2023 年 10 月 24 日
影子管理器 v2.3.4 發布	陰影管理器 v2.3.4 是可用的。此版本新增對空白和空白陰影狀態文件的支援。	2023 年 10 月 18 日
日誌管理器 v2.3.6 發布	日誌管理器組件 v2.3.6 可用。	2023 年 10 月 18 日
本地調試控制台 v2.4.0 發布	本地調試控制台組件 v2.4.0 可用。	2023 年 10 月 18 日
Lambda 管理器 v2.3.1 發布	Lambda 管理器組件 v2.3.1 可用。	2023 年 10 月 18 日
Greengrass CLI 版本 2.11.3 發布	綠色 CLI 組件 v2.11.3 是可用的。	2023 年 10 月 18 日

AWS IoT Greengrass 核心 v2.11.3 軟體更新	此版本提供了 Greengrass 核元件的 2.11.3 版，並更新提供的元件。 AWS	2023 年 10 月 18 日
安全隧道系統 1.0.17 版發佈	我們提供安全通道系統 v1.0.17 版本。	2023 年 10 月 4 日
Greengrass 開發工具包 CLI 版本 1.4.0	格林格拉斯開發工具包 CLI 的 1.4.0 版本可用。這個版本添加了一個新的`config`命令，啟動一個交互式提示來修改現有 GDK 配置文件中的字段。此版本還會修改`gdk component build`和命`gdk component publish`令，以在繼續之前驗證配方大小是否在 Greengrass 要求範圍內（ <= 16000 字節）。	2023 年 10 月 2 日
平均 MQTT 3.1.1 經紀商 V2.3.5 發布	平均 MQTT 3.1.1 經紀組件 V2.3.5 是可用的。此版本更新莫奎特 0.17 版本。	2023 年 9 月 28 日
MQTT 橋接器 v2.3.0 發布	MQTT 橋接器 v2.3.0 是可用的。此版本新增 MQTT 5 支援，可在 MQTT 來源 AWS IoT Core 與本地 MQTT 來源之間進行橋接。	2023 年 9 月 28 日
Lookout for Vision 邊緣劑 v1.1.6 發布	Lookout for Vision 邊緣代理 v1.1.6 可用。	2023 年 9 月 27 日
Lambda 管理器 v2.3.0 發布	Lambda 管理器組件 v2.3.0 可用。	2023 年 9 月 15 日
Lambda 發射器 v2.0.11 發布	Lambda 啟動器元件的 2.0.11 版本可供使用。此版本支持 Lambda 管理器 2.3.0。	2023 年 9 月 15 日

平均 MQTT 3.1.1 經紀商 V2.3.4 發布	平均 MQTT 3.1.1 經紀人組件 v2.3.4 是可用的。	2023 年 9 月 1 日
Greengrass 測試框架	GTF 是支援 end-to-end 自動化的建置區塊集合。它可讓 AWS IoT Greengrass Version 2 內部客戶使用與服務團隊相同的測試架構，以進行合格的軟體變更、自動接受和品質保證目的。	2023 年 8 月 11 日
AWS IoT Greengrass 核心軟體更新	此版本提供了 Greengrass 核元件的 2.11.2 版，以及更新提供的元件。 AWS	2023 年 8 月 9 日
Greengrass 開發工具包 CLI V1.3.0	Greengrass 開發工具包 CLI 的 1.3.0 版本可用。此版本增加了一個新的`test-e2e`命令，以支持使用開放 end-to-end 測試框架組件的測試。	2023 年 7 月 21 日
AWS IoT Greengrass 核心 v2.11.1 軟體更新	此發行版本提供 Greengrass 核元件的 2.11.1 版，以及更新提供的元件。 AWS	2023 年 7 月 21 日
磁盤後台處理程序 v1.0.0 發布	磁碟多工緩衝處理程式元件 v1.0.0 可用。	2023 年 6 月 28 日
AWS IoT Greengrass 核心軟體更新	此版本提供了 Greengrass 核組件的 2.11.0 版本和更新提供的組件。 AWS	2023 年 6 月 28 日
AWS IoT Greengrass 核心版本 v2.10.3 軟體更新	此版本提供了 Greengrass 核元件的 2.10.3 版，以及更新提供的元件。 AWS	2023 年 6 月 21 日

AWS IoT Greengrass 核心 v2.10.2 軟體更新	此發行版本提供 Greengrass 核元件的 2.10.2 版，以及更新提供的元件。 AWS	2023 年 6 月 5 日
AWS IoT Greengrass 核心 v2.10.1 軟體更新	此版本提供了 Greengrass 核組件的 2.10.1 版本和更新提供的組件。 AWS	2023 年 5 月 11 日
AWS IoT Greengrass 核心 v2.10.0 軟體更新	此版本提供了 Greengrass 核組件的 2.10.0 版本和更新提供的組件。 AWS	2023 年 5 月 9 日
SageMaker 邊緣管理員停止	Amazon SageMaker 邊緣管理器組件將於 2024 年 4 月 26 日停產。	2023 年 4 月 28 日
AWS IoT Greengrass 核心 v2.9.6 軟體更新	此版本提供 Greengrass 核心元件的 2.9.6 版，以及更新提供的元件。 AWS	2023 年 4 月 20 日
日誌管理器 v2.3.2 發布	日誌管理器組件 v2.3.2 可用。	2023 年 4 月 19 日
流管理器 v2.1.4 發布	流管理器 v2.1.4 現在可用。此版本修正了 SiteWise API 的單一批次傳回`ConflictingOperationException` 中具有相同時間戳記的相同屬性資產項目的問題，導致串流管理員持續重試。此版本也會將預設連線逾時從 3 秒更新為 1 分鐘。	2023 年 4 月 13 日
Greengrass 開發工具包 CLI 版本 1.2.3	Greengrass 開發工具包 CLI 的 1.2.3 版本可用。此版本包含錯誤修復。	2023 年 4 月 13 日

客戶端設備身份驗證組件 v2.4.0 發布	客戶端設備身份驗證組件 v2.4.0 可用。此版本新增了對用戶端裝置驗證的支援，以發出可顯示在 Greengrass 用戶端裝置儀表板上的操作指標。	2023 年 4 月 10 日
Greengrass 開發工具包 CLI v1.2.2	Greengrass 開發工具包 CLI 的 1.2.2 版本可用。此版本包含改進和錯誤修復。	2023 年 4 月 7 日
AWS IoT Greengrass 核心 v2.9.5 軟體更新	此版本提供了 Greengrass 核組件的 2.9.5 版本和更新提供的組件。 AWS	2023 年 3 月 30 日
流管理器 v2.1.3 發布	流管理器 v2.1.3 現在可用。此版本修復了以系統用戶身份運行時 Windows 操作系統上的啟動問題。	2023 年 3 月 7 日
通訊協定介面卡 v2.1.5 發行	Modbus RTU 協議適配器組件 v2.1.5 是可用的。此版本修正了`ReadDiscreteInput` 作業的問題。	2023 年 3 月 7 日
客戶端設備身份驗證組件 v2.3.2 發布	客戶端設備身份驗證組件 v2.3.2 可用。此版本新增了快取主機名稱資訊的支援，以便元件在離線時重新啟動時正確產生憑證主體。	2023 年 3 月 7 日
AWS IoT Device Tester v4.7.0 支持 Greengrass 核 2.9.4 版本	對於 AWS IoT Greengrass V2 的 IDT 4.7.0 版本現在支持 Greengrass 核 2.9.4 版本。	2023 年 3 月 2 日
綠色命令行界面 v1.2.0 發布	綠色命令行界面 v1.2.0 是可用的。	2023 年 2 月 28 日

AWS IoT Greengrass 核心 v2.9.4 軟體更新	此版本提供 Greengrass 核元件的 2.9.4 版本，並更新提供的元件。 AWS	2023 年 2 月 24 日
影子管理器 v2.3.1 發布	陰影管理器 v2.3.1 可用。此版本修正了可能無法同步雲端陰影更新的情況。此版本也修正具名陰影同步組態的變更只會套用至一個具名陰影的問題。	2023 年 2 月 21 日
AWS IoT Device Tester v4.7.0 支持 Greengrass 核 2.9.3 版本	對於 AWS IoT Greengrass V2 的 IDT 4.7.0 版本現在支持 Greengrass 核 2.9.3 版本。	2023 年 2 月 9 日
IAM 最佳做法已更新	更新了指南以符合 IAM 最佳實務。如需更多詳細資訊，請參閱 IAM 中的安全最佳實務。	2023 年 2 月 3 日
AWS IoT Greengrass 核心 v2.9.3 軟體更新	此版本提供 Greengrass 核元件的 2.9.3 版，並更新提供的元件。 AWS	2023 年 2 月 1 日
日誌管理器 v2.3.1 發布	日誌管理器 v2.3.1 可用。	2023 年 1 月 27 日
AWS IoT Device Tester v4.7.0 支持 Greengrass 核 2.9.2 版本	對於 AWS IoT Greengrass V2 的 IDT 4.7.0 版本現在支持 Greengrass 核 2.9.2 版本。	2023 年 1 月 3 日
影子管理器 v2.3.0 發布	陰影管理器 v2.3.0 可用。此版本修正了當裝置將 Greengrass 裝置私密金鑰儲存在硬體安全性模組中時，可能會導致陰影無法同步的問題。	2022 年 12 月 29 日
AWS IoT 車隊配置插件 v1.2.0 發布	AWS IoT 車隊配置插件 v1.2.0 是可用的。此版本透過具有可設定私密金鑰路徑的憑證簽章要求新增裝置佈建的支援。	2022 年 12 月 22 日

AWS IoT Greengrass 核心 v2.9.2 軟體更新	此版本提供了 Greengrass 核組件的 2.9.2 版本和更新提供的組件。 AWS	2022 年 12 月 22 日
AWS IoT Device Tester 版本 4.7.0 與 GGV2Q 版本發布	已提供適用於 AWS IoT Greengrass V2 的 IDT 版本 4.7.0。此發行版本包含 AWS IoT Greengrass V2 資格套件 (GGV2Q) v2.5.0 版，並支援 Greengrass 核心版本 2.9.1、2.9.0、2.8.1、2.8.0、2.7.0 和 2.6.0 版。	2022 年 12 月 13 日
影子管理器 v2.2.4 發布	修正更新本機陰影文件時，陰影大小驗證與雲端不一致的問題。這也可修正當部署 RESET在組態節點上執行時，陰影管理員會停止接聽組態更新的問題。	2022 年 12 月 8 日
Lookout for Vision 邊緣代理 1.1.1 發布	Lookout for Vision 邊緣代理組件 v1.1.1 可用。	2022 年 12 月 5 日
日誌管理器 v2.3.0 發布	日誌管理器組件 v2.3.0 可用。	2022 年 11 月 18 日
AWS IoT Device Tester v4.5.11 支持 Greengrass 核 2.9.1 版本	針對 AWS IoT Greengrass V2 的 IDT 版本 4.5.11 現在支援 Greengrass 核 2.9.1 版本。	2022 年 11 月 18 日
AWS IoT Greengrass 核心 v2.9.1 軟體更新	此版本提供了 Greengrass 核組件的 2.9.1 版本和更新提供的組件。 AWS	2022 年 11 月 18 日
AWS IoT Device Tester v4.5.11 支持 Greengrass 核 2.9.0 版本	針對 AWS IoT Greengrass V2 的 IDT 版本 4.5.11 現在支援 Greengrass 核 2.9.0 版本。	2022 年 11 月 17 日

流管理器 v2.1.2 發布	流管理器 v2.1.2 現在可用。此版本可修正 Windows 作業系統上使用非英文語言的問題。	2022 年 11 月 15 日
AWS IoT Greengrass 核心版軟體更新	此版本提供了 Greengrass 核組件的 2.9.0 版本和更新提供的組件。 AWS	2022 年 11 月 15 日
AWS IoT Device Tester v4.5.11 支持 Greengrass 核 2.8.1 版本	針對 AWS IoT Greengrass V2 的 IDT 版本 4.5.11 現在支援 Greengrass 核 2.8.1 版本。	2022 年 10 月 19 日
AWS IoT Device Tester 與 GGV2Q 版本 4.1 版一起發布	已提供適用於 AWS IoT Greengrass V2 的 IDT 版本 4.5.11。此版本包含 AWS IoT Greengrass V2 合格套件 (GGV2Q) v2.4.1，並支援 Greengrass 核心版本 2.8.0、2.7.0 和 2.6.0。	2022 年 10 月 13 日
AWS IoT Greengrass 核心 v2.8.1 軟體更新	此版本提供 Greengrass 核心元件的 2.8.1 版，以及更新提供的元件。 AWS	2022 年 10 月 13 日
AWS IoT Greengrass 核心 v2.8.0 軟體更新	此版本提供了 Greengrass 核元件的 2.8.0 版，並更新提供的元件。 AWS	2022 年 10 月 7 日
增加了對部署的 AWS CloudFormation 支持	AWS CloudFormation 現在支援作為資源進行 AWS IoT Greengrass 部署。	2022 年 10 月 6 日

SageMaker 邊緣管理器 v1.3.0 發布	Amazon SageMaker 邊緣管理器組件 v1.3.0 可用。此版本新增了對此元件的支援，可設定 TensorRT 模型快取的磁碟大小，並改善預測並行性，以更好地利用裝置加速器引擎 (例如 GPU)。	2022 年 9 月 1 日
使用處理序間通訊 (IPC) 用戶端 V2	已新增 IPC 用戶端 V2 的相關資訊，可減少使用 IPC 作業所需撰寫的程式碼量，並協助避免 IPC Client V1 可能發生的常見錯誤。	2022 年 8 月 12 日
AWS IoT Device Tester v4.5.8 與 GGV2Q 版本一起發布	已提供適用於 AWS IoT Greengrass V2 的 IDT 版本 4.5.8。此版本包含 AWS IoT Greengrass V2 合格套件 (GGV2Q) v2.4.0，並支援 Greengrass 核心版本 2.7.0、2.6.0 和 2.5.6。	2022 年 8 月 12 日
SageMaker 邊緣管理器 v1.2.0 發布	Amazon SageMaker 邊緣管理器組件 v1.2.0 可用。此版本新增對此元件的支援，可自動擷取您上傳到 Amazon S3 的新 SageMaker 編譯模型，因此您無需建立部署即可部署新模型。 AWS IoT Greengrass	2022 年 8 月 3 日
AWS IoT Device Tester v4.5.3 支持 Greengrass 核 2.7.0 版本	對於 AWS IoT Greengrass V2 的 IDT 4.5.3 版本現在支持 Greengrass 核 2.7.0 版本。	2022 年 8 月 1 日
流管理器 v2.1.0 發布	流管理器 v2.1.0 現在可用。此版本包含將遙測指標傳送至 Amazon 的支援 EventBridge。	2022 年 7 月 28 日

AWS IoT Greengrass 核心 v2.7.0 軟體更新	此版本提供 Greengrass 核心元件的 2.7.0 版，以及更新提供的元件。 AWS它包括將遙測指標傳送到 Amazon 的支援 EventBridge。	2022 年 7 月 28 日
IoT SiteWise 出版商 v2.2.0 發布	IoT SiteWise 發布者組件 v2.2.0 可用。此版本會更新元件，以便在將資料傳送至 AWS IoT SiteWise 服務之前壓縮資料，最多可減少 75% 的頻寬使用量。	2022 年 7 月 19 日
教學課程：開發與用戶端裝置陰影互動的元件	在教學課程中新增新模組：透過 MQTT 與本機 IoT 裝置互動，您可以遵循該模組瞭解如何開發與用戶端裝置陰影互動的元件。	2022 年 7 月 18 日
選擇當地的 MQTT 經紀商	已新增有關如何選擇用戶端裝置連線至核心裝置的本機 MQTT 代理程式的資訊。	2022 年 7 月 18 日
AWS IoT Device Tester v4.5.3 支持 Greengrass 核 2.6.0 版本	對於 AWS IoT Greengrass V2 的 IDT 4.5.3 版本現在支持 Greengrass 核 2.6.0 版本。	2022 年 6 月 29 日

AWS IoT Greengrass 核心 v2.6.0 軟體更新	此版本提供了 Greengrass 核組件的 2.6.0 版本和更新提供的組件。 AWS它包括用戶端裝置陰影的支援，以及用戶端裝置的本機 MQTT 5 代理程式。它還包括支援本機發佈/訂閱主題中的萬用字元、元件組態中的配方變數，以及 IPC 授權原則中的萬用字元。這些功能可讓您更輕鬆地開發和設定部署到核心裝置群組的元件。此版本還包括使用 IPC 操作的元件支援，這些操作可管理核心裝置上的本機部署和元件。	2022 年 6 月 27 日
用戶端裝置元件更新	客戶端設備身份驗證 v2.1.0，MQTT 代理商（模型）v2.1.0，MQTT 橋接器 v2.1.1 和 IP 檢測器 v2.1.2 可用。此版本可改善憑證輪替、改善 MQTT 代理程式效能，並修正這些元件如何處理組態重設更新的問題。	2022 年 6 月 14 日
AWS IoT Device Tester v4.5.3 支持 Greengrass 核 2.5.6 版本	對於 AWS IoT Greengrass V2 的 IDT 4.5.3 版本現在支持 Greengrass 核 2.5.6 版本。	2022 年 6 月 1 日
AWS IoT Greengrass 核心 v2.5.6 軟體更新	此版本提供了 Greengrass 核元件的 2.5.6 版，並更新提供的元件。 AWS它包括對帶有 ECC 密鑰的硬件安全模塊的支持。它還包括其他錯誤修復和改進。	2022 年 5 月 31 日

AWS IoT 車隊配置插件 v1.1.0 發布	AWS IoT 車隊配置插件 v1.1.0 可用。當您在 Windows 設備上配置插件時，此版本添加了對其他文件路徑格式的支持。	2022 年 5 月 12 日
推出新的 Lambda 執行階段	增加了對新的 Lambda 運行時的支持：Python 3.9，Java 11 和 NodeJS 14。	2022 年 5 月 10 日
開發延遲元件更新的 Greengrass 元件	已新增教學課程，您可以遵循該教學課程來學習如何開發可延遲部署元件更新的 Greengrass 元件。例如，當裝置電池電量不足或執行無法中斷的程序時，您可能會想要延遲更新。	2022 年 5 月 4 日
CloudWatch 發布的指標 v3.1.0 和 AWS IoT Device Defender 版本	CloudWatch 您可以使用測量結果元件 v3.1.0 和 AWS IoT Device Defender 元件 v3.1.0。這些版本新增了 HTTPS 網路代理伺服器組態的支援。如需詳細資訊，請參閱 Connect 埠 443 或透過網路 Proxy 進行連線和啟用核心裝置信任 HTTPS Proxy。	2022 年 4 月 27 日
移轉來源 AWS IoT Greengrass Version 1	已新增指南，您可以按照該指南移轉 AWS IoT Greengrass V1 至 AWS IoT Greengrass V2。	2022 年 4 月 26 日

AWS IoT Device Tester 已更新 GGV2Q v2.3.1 版的 v4.5.3，並將具有 GGV2Q 版本的 IDT v4.5.1 新增至支援的版本	使用 AWS IoT Greengrass V AWS IoT Greengrass 2 資格套件 (GGV2Q) v2.3.1 的 IDT 版本 4.5.3 已更新，以包含對 Greengrass 心 2.5.5、2.5.4 和 2.5.3 版的支援。此更新也包含 IDT 4.5.1，並以 AWS IoT Greengrass V2 資格套件 (GGV2Q) v2.3.0 做為支援的版本。IDT 4.5.1 與 AWS IoT Greengrass V2 合格套件 (GGV2Q) v2.3.0 支援 Greengrass 核 2.5.3 版本。	2022 年 4 月 25 日
通訊協定介面卡 v2.1.0 已發佈	Modbus-RTU 協議適配器組件 v2.1.0 是可用的。此版本增加了新的參數，您可以指定這些參數來配置與 Modbus RTU 設備的串行通信。	2022 年 4 月 20 日
CloudWatch 發布指標 v2.1.0、Firehose v2.1.0 和 Amazon SNS v2.1.0	CloudWatch 您可以使用指標元件 v2.1.0、Firehose 元件 v2.1.0 和 Amazon SNS 元件 v2.1.0。這些版本新增了 HTTPS 網路代理伺服器組態的支援。如需詳細資訊，請參閱 Connect 埠 443 或透過網路 Proxy 進行連線和啟用核心裝置信任 HTTPS Proxy。	2022 年 4 月 19 日
AWS IoT Device Tester 與 GGV2Q 版本 3.1 版本發布	已提供適用於 AWS IoT Greengrass V2 的 IDT 版本 4.5.3。此版本包括 AWS IoT Greengrass V2 合格套件（GGV2Q）v2.3.1，並支持 Greengrass 核 2.5.5 版本。	2022 年 4 月 15 日

AWS IoT Greengrass 核心 v2.5.5 軟體更新	此版本提供 Greengrass 核心元件的 2.5.5 版，並更新提供的元件。 AWS它增加了對使用英語以外的顯示語言的 Windows 設備的支持。它也修正了核心裝置在特定情況下佈建後未向 AWS IoT Greengrass 雲端服務報告其狀態的問題。	2022 年 4 月 6 日
AWS IoT Greengrass 核心 v2.5.4 軟體更新	此版本提供 Greengrass 核元件的 2.5.4 版，以及更新提供的元件。 AWS它包括錯誤修復和改進。	2022 年 3 月 23 日
AWS IoT Device Tester 編程下載	已新增有關如何 AWS IoT Greengrass V2 以程式設計方式下載 IDT 的資訊。	2022 年 3 月 15 日
Greengrass 開發工具包 CLI V1.1.0	Greengrass 開發工具包 CLI 的 1.1.0 版可用。此版本會將新引數加入至`component init`和`component publish`指令。如果未構建組件，此版本還會更新`component publish`命令以構建組件。	2022 年 2 月 24 日
影子管理器 v2.1.0 發布	陰影管理員元件 v2.1.0 可用。此版本新增了用於規劃元件與 AWS IoT Core陰影同步處理間隔的選項。例如，您可以指定較長的間隔來減少頻寬使用量和費用。	2022 年 2 月 3 日

核心軟件 v2.5.3 的 AWS IoT Greengrass 碼頭文件和碼頭圖像	AWS IoT Greengrass 核心軟件 v2.5.3 的碼頭文件和碼頭映像現已推出。	2022 年 1 月 12 日
AWS IoT Device Tester 版本 4.5.1 與 GGV2Q 版本發布	已提供適用於 AWS IoT Greengrass V2 的 IDT 版本 4.5.1。此版本包含 AWS IoT Greengrass V2 資格套件 (GGV2Q) v2.3.0，並支援使用硬體安全性模組 (HSM) 儲存 Core 軟體所使用之私密金鑰和憑證的 Linux 裝置進行驗證和限定。 AWS IoT Greengrass	2022 年 1 月 11 日
AWS IoT Greengrass 核心 v2.5.3 軟體更新	此版本提供 Greengrass 核心元件的 2.5.3 版，以及更新提供的元件。 AWS它支援您將 AWS IoT Greengrass Core 軟體設定為使用您安全地儲存在硬體安全模組 (HSM) 中的私密金鑰和憑證。	2022 年 1 月 6 日
核心軟件 v2.5.2 的 AWS IoT Greengrass 碼頭文件和碼頭圖像	AWS IoT Greengrass 核心軟件 v2.5.2 的碼頭文件和碼頭映像現已上市。	2021 年 12 月 20 日
AWS IoT Device Tester 版本 4.1 與 GGV2Q 版本 2.1 發布	已提供適用於 AWS IoT Greengrass V2 的 IDT 4.4.1 版本。此版本包含 AWS IoT Greengrass V2 合格套件 (GGV2Q) v2.2.1 版，並支援 Greengrass 核心 2.5.2 版以進行裝置認證。	2021 年 12 月 12 日

使用 Amazon Lookout for Vision 執行機器學習推論	已新增有關如何在 Greengrass 核心裝置上使 Lookout for Vision 執行機器學習推論的資訊。Lookout for Vision 使用電腦視覺來尋找工業產品中的視覺缺陷。	2021 年 12 月 8 日
AWS IoT Device Tester 版本 4.4.1 與 GGV2Q 版本 2.2.0 發布	已提供適用於 AWS IoT Greengrass V2 的 IDT 4.4.1 版本。此版本包含 AWS IoT Greengrass V2 合格套件 (GGV2Q) v2.2.0，並支援 Greengrass 核 2.5.2 版以進行裝置認證。	2021 年 12 月 6 日
AWS IoT Greengrass 核心 v2.5.2 軟體更新	此版本提供了 Greengrass 核組件的 2.5.2 版本和更新提供的組件。 AWS它修復了 Greengrass 核更新後發生的 Windows 服務的問題。它還包括對 Windows 設備上的 AWS IoT Device Defender 組件的支持。	2021 年 12 月 3 日
適用於 Kinesis Video Streams 元件的新邊緣連接器	適用於 Kinesis Video Streams 元件的邊緣連接器 1.0.0 版可供使用。此 AWS提供讀取本地攝像機的視頻源，並將流發佈到 Kinesis Video Streams。此元件與整合 AWS IoT TwinMaker，可讓您檢視和管理 Grafana 儀表板中的視訊串流和其他資料。	2021 年 11 月 30 日

使用 Greengrass 管理核心裝置 AWS Systems Manager	已新增有關如何使用管理 Greengrass 核心裝置的資訊。 AWS Systems Manager Systems Manager 是一項 AWS 服務，可讓您檢視作業資料、將作業工作自動化，以及維護安全性與合規性。	2021 年 11 月 29 日
Greengrass 開發工具包 CLI	已新增有關開 AWS IoT Greengrass 發套件命令列介面 (GDK CLI) 的資訊，您可以下載至本機開發電腦，以協助您開發自訂 Greengrass 元件的工具。您可以使用 GDK CLI 來建立、建置和發佈自訂元件。	2021 年 11 月 29 日
社群提供的 Greengrass 元件	已新增 Greengrass 軟體目錄的相關資訊，這是 Greengrass 社群所開發的 Greengrass 元件索引。您可以從此目錄下載、修改和部署元件，以建立 Greengrass 應用程式。	2021 年 11 月 29 日
AWS IoT Greengrass 核心 v2.5.1 軟體更新	此版本提供 Greengrass 核心元件的 2.5.1 版，並更新提供的元件。 AWS它包括對 Windows 設備上的 32 位 Java 的支持。它也會修正 Windows 裝置上新物件群組移除行為和載入系統環境變數的問題。	2021 年 11 月 23 日

AWS IoT Device Tester 版本 4.4.0 與 GGV2Q 版本 1.0 發布	已提供適用於 AWS IoT Greengrass V2 的 IDT 版本 4.4.0。此版本包括 AWS IoT Greengrass V2 合格套件（GGV2Q）v2.1.0，並支持運行格林格拉斯核 2.5.0 版本的基於 Windows 的 Greengrass 設備的限定。	2021 年 11 月 19 日
AWS IoT Greengrass 核心 v2.5.0 軟體更新	此發行版本提供 Greengrass 核心元件的 2.5.0 版，以及更新提供的元件。 AWS它包括在 Windows 設備上運行 AWS IoT Greengrass 核心軟件的支持。它也會變更物件群組移除行為，並新增 HTTPS 代理伺服器的支援。	2021 年 11 月 12 日
SageMaker 邊緣管理器 v1.1.0 發布	Amazon SageMaker 邊緣管理器組件 v1.1.0 可用。此版本新增對執行 Amazon Linux 2 之 Greengrass 核心裝置的支援，並新增組態參數以指定裝置上擷取資料資料夾的位置。	2021 年 11 月 3 日
跨服務混淆代理人預防更新	AWS IoT Greengrass V2 支援在 IAM 資源政策中使用 `aws:SourceArn` 和 `aws:SourceAccount` 全域條件內容金鑰，以防止混淆的副問題。	2021 年 11 月 1 日

用戶端裝置元件更新	客戶端設備身份驗證 v2.0.3，IP 檢測器 v2.1.0，MQTT 橋接器 v2.1.0 和 MQTT 代理商（模特）v2.0.2 可用。此版本新增了對非預設 MQTT 代理程式連接埠的完整支援，並包含其他錯誤修正和改進。	2021 年 10 月 28 日
影子管理器 v2.0.4 發布	陰影管理員元件 v2.0.4 可用。此版本修正了造成陰影管理員刪除先前刪除之任何陰影的新建立版本的問題。從此版本開始，DeleteThingShadow IPC 作業會增加陰影版本。	2021 年 10 月 20 日
日誌管理器 v2.2.0 發布	日誌管理器組件 v2.2.0 可用。日誌管理器現在支持使用配置映射來提供組件日誌配置。	2021 年 10 月 20 日
Lambda 管理器 v2.1.4 發布	Lambda 管理器組件 v2.1.4 可用。此版本修正了導致使用 NodeJS 執行階段的 Lambda 函數僅處理一則訊息的問題。	2021 年 10 月 20 日
在 Docker 容器組件中使用進程間通信， AWS 憑據和流管理器	已新增有關如何在自訂 Docker 容器元件中使用處理序間通訊 (IPC)、 AWS 認證和串流管理員的資訊。	2021 年 10 月 19 日
新的原子核遙測發射器元件	原子核遙測發射器元件的 1.0.0 版可供使用。此 AWS提供的元件會收集系統健康情況遙測資料，並持續將其發佈至本機主題和 AWS IoT Core MQTT 主題。	2021 年 9 月 30 日

允許裝置流量透過 Proxy 或防火牆	已新增 Greengrass 核心裝置所使用之端點和連接埠的相關資訊，因此您可以限制流量做為安全措施。	2021 年 9 月 16 日
AWS IoT Device Tester 與 GGV2Q V2.0.1 版發布	適用於 V2 的 IDT 版本 4.2.0 已更新為 AWS IoT Greengrass AWS IoT Greengrass V2 資格套件 (GGV2Q) v2.0.1。此版本支援 Greengrass 核 2.4.0 版以進行裝置限定。	2021 年 8 月 31 日
更新的機器學習安裝程式	DLR 安裝程序組件 v1.6.5 和 TensorFlow 精簡版安裝程序組件 v2.5.4 可用。這些元件版本包含新的`UseInstaller` 組態參數，可讓您停用預設安裝指令碼。	2021 年 8 月 30 日
嵌入式 Linux 支援 AWS IoT Greengrass	的 BitBake 方 AWS IoT Greengrass V2 法可在上的`meta-aws`專案中取得 GitHub。您可以使用此配方，使用 Yocto 專案建立自訂的 Linux 作業系統。	2021 年 8 月 20 日
代碼完整性	已新增有關如何 AWS IoT Greengrass V2 驗證 Greengrass 核心裝置從. AWS 雲端	2021 年 8 月 19 日
VPC 端端點 ()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 和 AWS IoT Greengrass 控制平面之間建立私人連接。	2021 年 8 月 16 日

流管理器 v2.0.12 發布	流管理器 v2.0.12 現在可用。此版本修正了無法從串流管理員元件 2.0.7 版升級至 v2.0.8 到 v2.0.11 之間的版本的問題。	2021 年 8 月 10 日
核心軟件 v2.4.0 的 AWS IoT Greengrass 碼頭文件和碼頭圖像	AWS IoT Greengrass 核心軟件 v2.4.0 的碼頭文件和碼頭映像現已上市。	2021 年 8 月 9 日
AWS IoT Greengrass 核心 v2.4.0 軟體更新	此版本提供了 Greengrass 核組件的 2.4.0 版本和更新提供的組件。 AWS它包括對元件系統資源限制的支援、暫停和恢復元件的 IPC 作業，以及佈建外掛程式。	2021 年 8 月 3 日
新 AWS IoT SiteWise 元件	新增以下 AWS提供的元件 AWS IoT SiteWise：IoT SiteWise OPC-UA 收集器、IoT SiteWise 發行者和 IoT 處理器。 SiteWise	2021 年 7 月 29 日
AWS IoT Device Tester 版本 4.2.0 與 GGV2Q 版本發布	已提供適用於 AWS IoT Greengrass V2 的 IDT 版本 4.2.0。此版本包含 AWS IoT Greengrass V2 資格套件 (GGV2Q) v2.0.0，並支援 Docker 元件、機器學習和串流管理員的選用資格測試。	2021 年 7 月 14 日
AWS IoT Greengrass 核心 IPC 庫可用 AWS IoT Device SDK 於 C ++ V2	C ++ v2 版本 1.13.0 支持 AWS IoT Greengrass 核心 IPC，因此您可以在 C ++ 中開發與核心軟件進行交互的 AWS IoT Greengrass 組件。 AWS IoT Device SDK	2021 年 7 月 14 日

SageMaker 邊緣管理器組件 v1.0.2 發布	Amazon SageMaker 邊緣管理器組件 v1.0.2 可用。此版本會更新元件生命週期中的安裝指令碼。您的核心裝置現在必須在裝置上安裝 Python 3.6 或更新版本 (包括pip您的 Python 版本)，然後才能部署此元件。	2021 年 7 月 12 日
對於 AWS IoT Greengrass V2 的 Sup AWS IoT Device Tester port 更新	適用於 AWS IoT Greengrass V2 版本 4.1.0 的 IDT 現在支援使用 Greengrass 核 2.3.0 版進行裝置限定。	2021 年 7 月 8 日
核心軟件 v2.3.0 的 AWS IoT Greengrass 碼頭文件和碼頭圖像	AWS IoT Greengrass 核心軟件 v2.3.0 的碼頭文件和碼頭映像現已上市。	2021 年 7 月 7 日
AWS 受管政策	已新增有關的 AWS 受管理原則的資訊 AWS IoT Greengrass。	2021 年 7 月 2 日
新推薦的 JVM 選項	已新增控制 AWS IoT Greengrass Core 軟體記憶體配置之建議 JVM 選項的相關資訊。	2021 年 6 月 30 日
AWS IoT Greengrass 核心 v2.3.0 軟體更新	此版本提供了 Greengrass 核組件的 2.3.0 版本和更新提供的組件。 AWS它包括對部署中的大型元件組態文件的支援。	2021 年 6 月 29 日
碼頭文件和碼頭圖像核心軟件 v2.2.0 AWS IoT Greengrass	AWS IoT Greengrass 核心軟件 v2.2.0 的碼頭文件和碼頭映像現已上市。	2021 年 6 月 28 日

AWS IoT Device Tester 與 GGV2Q 版本 1.1 版發布	已提供適用於 AWS IoT Greengrass V2 的 IDT 版本 4.1.0。此版本包含 AWS IoT Greengrass V2 合格套件 (GGV2Q) 1.1.1 版，並支援使用 Greengrass 核心 v2.2.0、2.1.0 和 v2.0.5 以取得裝置認證。	2021 年 6 月 18 日
AWS IoT Greengrass 核心 v2.2.0 軟體更新	此版本提供了 Greengrass 核組件的 2.2.0 版本和更新提供的組件。 AWS它包含您可以部署的元件，以新增對用戶端裝置的支援，並新增本機陰影服務。	2021 年 6 月 18 日
Lambda 發射器 v2.0.6 發布	Lambda 啟動器元件的 2.0.6 版本可供使用。此版本包括性能改進和錯誤修復。	2021年6月13日
已發行新 SageMaker 邊緣管理員元件	Amazon SageMaker 邊緣管理器組件的 1.0.0 版可用於 AWS IoT Greengrass. 此元件會在 Greengrass 核心裝置上安裝 SageMaker 邊緣管理員代理程式二進位檔。	2021 年 6 月 10 日
元件類型	已新增有關中元件類型的資訊 AWS IoT Greengrass。元件類型會指定 AWS IoT Greengrass 核心軟體執行元件的方式。	2021 年 6 月 3 日

AWS IoT Device Tester v4.0.2 與 GGV2Q 版本 1.1.0 發布	已提供適用於 AWS IoT Greengrass V2 的 IDT 版本 4.0.2。此版本包含 AWS IoT Greengrass V2 合格套件 (GGV2Q) 1.1.0 版，並支援使用 Greengrass 核心 v2.1.0 與 Greengrass 心 v2.1.0 以取得裝置限定。這也包括 MQTT 和 Lambda 的新必要測試群組，以及其他小錯誤修正和改進。	2021 年 5 月 5 日
碼頭文件和碼頭圖像核心軟件 v2.1.0 AWS IoT Greengrass	AWS IoT Greengrass 核心軟件 v2.1.0 的碼頭文件和碼頭映像現已上市。Docker 映像檔可讓您在使用 Amazon Linux 2 做為基礎作業系統的泊塢視窗容器中執行 AWS IoT Greengrass 核心軟體。	2021 年 4 月 27 日
AWS IoT Greengrass 核心 v2.1.0 軟體更新	此版本提供了 Greengrass 核組件的 2.1.0 版和更新提供的組件。 AWS其中包含一個新元件，您可以用來從私有 Amazon ECR 儲存庫下載 Docker 映像，以及使用 Lite 執行機器學習推論的新範例元件。 TensorFlow	2021 年 4 月 26 日
使用 Secrets Manager 的範例元件	已新增範例元件，列印您部署至核心裝置的 AWS Secrets Manager 密碼值。	2021 年 4 月 8 日
Greengrass 核心設備的最小 AWS IoT 策略	已新增有關在核心裝置上支援基本 Greengrass 功能所需的最小權限集合的資訊。	2021 年 4 月 2 日

訂閱 IPC 事件串流	已新增有關如何使用處理序間通訊 (IPC) 作業來訂閱 Greengrass 核心裝置上的事件串流的資訊。	2021 年 4 月 1 日
的 Support AWS IoT Device Tester 更新 AWS IoT Greengrass	針對 AWS IoT Greengrass V2 4.0.1 版本的 IDT 現在支援使用 Greengrass 核 2.0.5 版和 Greengrass 色核 2.0.5 版進行裝置限定。	二 O 一年三月十七日
建立使用串流管理員的自訂元件	已新增有關如何設定元件配方和成品以開發管理資料串流之應用程式的資訊。	2021 年 3 月 9 日
AWS IoT Greengrass 核心 v2.0.5 軟體更新	此版本提供了 Greengrass 核組件的 2.0.5 版本和更新提供的組件。 AWS它修正了網路 Proxy 支援的問題，以及中國區域中 Greengrass 資料平面端點的 AWS 問題。	2021 年 3 月 9 日
元件環境變數參考	已新增 AWS IoT Greengrass Core 軟體為元件設定之環境變數的相關資訊。您可以使用這些環境變數來取得物件名稱 AWS 區域、和 Greengrass 核心版本。	2021 年 2 月 23 日

手動安裝	已新增有關如何手動建立必要 AWS 資源或安裝在防火牆或網路 Proxy 後方的資訊。通過使用手動安裝，您不需要授予安裝程序權限來在您的中創建資源 AWS 帳戶，因為您創建了必要的 AWS IoT 和 IAM 資源。您也可以將裝置設定為透過連接埠 443 或透過網路 Proxy 進行連線。	2021 年 2 月 17 日
AWS IoT Greengrass 核心工 AWS IoT Device SDK 業電腦程式庫更新	AWS IoT Device SDK 適用於 Python V2 的 1.5.4 版簡化了連接到 AWS IoT Greengrass 核心 IPC 服務所需的步驟。	2021 年 2 月 11 日
的 Support AWS IoT Device Tester 更新 AWS IoT Greengrass	針對 AWS IoT Greengrass V2 版本 4.0.1 的 IDT 現在支援使用 Greengrass 核 2.0.4 版和 Greengrass 色核 2.0.4 CLI 本以進行裝置限定。	2021 年 2 月 5 日
匯入 Lambda 函數的新教學課程	新增以主控台為基礎的新教學課程，將 Lambda 函數匯入為在 Greengrass 核心裝置上執行的元件。	2021 年 2 月 5 日

AWS IoT Greengrass 核心 v2.0.4 軟體更新	此版本提供了 Greengrass 核成分的 2.0.4 版本。它包含用於透過連接埠 443 設定 HTTPS 通訊的新`greengrassDataPlanePort` 參數，並修正錯誤。現在，AWS IoT Greengrass 核心軟體安裝程式執`iam:GetPolicy` 行`sts:GetCallerIdentity` 時需要最低 IAM 政策，以及執行時間`--provision true` 。	2021 年 2 月 4 日
推出新的安全通道元件	安全通道元件的 1.0.0 版可供使用。 AWS IoT Greengrass 此 AWS提供的元件使用 AWS IoT 安全通道，與受限防火牆後方的 Greengrass 核心裝置建立安全的雙向通訊。	2021 年 1 月 21 日
AWS IoT Device Tester 對於版本 AWS IoT Greengrass 4.0.1 版本發布	已提供適用於 AWS IoT Greengrass V2 的 IDT 版本 4.0.1。此版本使您可以使用 IDT 開發和運行自定義測試套件以進行設備驗證。這也包括適用於 macOS 和視窗的程式碼簽署 IDT 應用程式。	2020 年 12 月 22 日
的初始版本 AWS IoT Greengrass Version 2	AWS IoT Greengrass V2 是的新主要版本發行版本 AWS IoT Greengrass。此版本增加了一些功能，例如模組化軟體元件和持續部署。這些功能可讓您更輕鬆地開發和管理邊緣應用程式。	2020 年 12 月 15 日

AWS 詞彙表

有關最新 AWS 術語，請參閱AWS 詞彙表 參考文獻中的AWS 詞彙表。

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。