開發人員指南

AWS Device Farm

API 版本 2015-06-23

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

AWS Device Farm: 開發人員指南

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

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

Table of Contents

什麼是 AWS Device Farm？

Device Farm 是一種應用程式測試服務，可讓您在 Amazon Web Services () 託管的真實實體手機和平板電腦上，測試 Android、iOS 和 Web 應用程式並與之互動AWS。

使用 Device Farm 有兩種主要方式：

- 從本機電腦遠端存取裝置，可在 Web 瀏覽器中以互動方式存取，或從本機用戶端使用 Appium 自動測試裝置。
- 使用 Device Farm 的受管測試執行環境自動執行應用程式測試。

> **Note**
>
> Device Farm 僅適用於 `us-west-2`（奧勒岡） 區域。

遠端存取

遠端存取可讓您透過 Web 瀏覽器即時與裝置互動。遠端存取也可讓您使用受管 Appium 端點，從本機用戶端對遠端 Device Farm 裝置執行 Appium 測試。

與裝置的即時互動對於多種案例很有用，例如手動應用程式測試、在特定裝置上重現錯誤、檢查不同螢幕類型的應用程式視覺化轉譯，以及應用程式安裝和升級序列。Device Farm 的全受管 Appium 端點可讓您開發、測試和偵錯 Appium 測試，提供快速意見回饋。

Appium 端點支援您選擇的任何語言、任何本機 IDE、具有中斷點的即時偵錯、即時視訊和日誌，以及 Appium Inspector 等工具。您可以在遠端存取工作階段期間，以 150 分鐘的限制在相同裝置上執行任意次數的測試。

在遠端存取工作階段期間，Device Farm 會記錄您與裝置互動時所發生動作的詳細資訊。日誌搭配這些詳細資訊和影片擷取的工作階段，會在工作階段結束時產生。

自動化應用程式測試

Device Farm 可讓您透過上傳應用程式和測試，在多個裝置上平行執行自動化測試。這些測試會在測試主機上的全受管環境中自動執行，您可以設定測試規格檔案。環境使用 Device Farm 的測試主機，因

此您不需要擔心佈建自己的基礎設施來執行測試。測試主機和裝置可以安全地連接到您的 VPC，以存取您的私有端點。

隨著測試完成，會產生測試報告，其中包含高階結果、低階日誌、螢幕擷取畫面和您的測試成品。

Device Farm 支援測試原生和混合 Android 和 iOS 應用程式。如需支援測試類型的詳細資訊，請參閱 AWS Device Farm 中的測試架構和內建測試。

術語

Device Farm 推出下列詞彙，定義資訊的組織方式：

裝置集區

裝置的集合通常有類似的性質，例如平台、製造商或型號。

job

要求 Device Farm 針對單一裝置測試單一應用程式。任務包含一或多個套件。

計量

指裝置的計費。您可以在文件和 API 參考中查看計量裝置或無限制裝置的參考。如需定價的詳細資訊，請參閱 AWS Device Farm 定價。

project

邏輯工作空間包含多個執行，每個執行皆為針對一或多個裝置執行的單一應用程式各項測試。您可以使用專案以自己選擇的方式整理工作空間。例如，每個應用程式標題可以有一個專案，或每個平台有一個專案。您可以視需要建立任意數量的專案。

報告

包含執行的相關資訊，這是 Device Farm 針對一或多個裝置測試單一應用程式的要求。如需詳細資訊，請參閱AWS Device Farm 中的報告。

run

應用程式的特定建置，它有一組特定的測試要在一組特定的裝置上執行。執行會產生結果的報告。執行包含一或多個任務。如需詳細資訊，請參閱執行。

工作階段

透過 Web 瀏覽器與實際實體裝置進行即時互動。如需詳細資訊，請參閱工作階段。

套件

測試套件中的測試階層組織。套件包含一或多個測試。

test

測試套件中的個別測試案例。

如需 Device Farm 的詳細資訊，請參閱 概念。

設定

若要使用 Device Farm，請參閱 設定。

設定 AWS Device Farm

第一次使用 Device Farm 之前，您必須先完成下列任務：

主題

步驟 1：註冊 AWS

註冊 Amazon Web Services (AWS)。

如果您沒有 AWS 帳戶，請完成下列步驟來建立一個。

註冊 AWS 帳戶

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

   部分註冊程序需接收來電或簡訊，並在電話鍵盤輸入驗證碼。

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

步驟 2：在您的帳戶中建立或使用 IAM 使用者 AWS

我們建議您不要使用 AWS 根帳戶來存取 Device Farm。請改為在 AWS 帳戶中建立 AWS Identity and Access Management (IAM) 使用者 （或使用現有的使用者），然後使用該 IAM 使用者存取 Device Farm。

如需詳細資訊，請參閱建立 IAM 使用者 (AWS 管理主控台)。

步驟 3：授予 IAM 使用者存取 Device Farm 的許可

授予 IAM 使用者存取 Device Farm 的許可。若要這樣做，請在 IAM 中建立存取政策，然後將存取政策指派給 IAM 使用者，如下所示。

> **Note**
>
> 您用來完成下列步驟的 AWS 根帳戶或 IAM 使用者必須具有建立下列 IAM 政策並將其連接至 IAM 使用者的許可。如需詳細資訊，請參閱使用政策。

1. 使用下列 JSON 內文建立政策。指定描述性的標題，例如 *DeviceFarmAdmin*。

   如需建立 IAM 政策的詳細資訊，請參閱《IAM 使用者指南》中的建立 IAM 政策。
2. 將您建立的 IAM 政策連接至新使用者。如需將 IAM 政策連接至使用者的詳細資訊，請參閱《IAM 使用者指南》中的新增和移除 IAM 政策。

連接政策可讓 IAM 使用者存取與該 IAM 使用者相關聯的所有 Device Farm 動作和資源。如需如何將 IAM 使用者限制為一組有限的 Device Farm 動作和資源的詳細資訊，請參閱 AWS Device Farm 中的身分和存取管理。

下一步驟

您現在可以開始使用 Device Farm。請參閱 Device Farm 入門。

Device Farm 入門

本演練說明如何使用 Device Farm 測試原生 Android 或 iOS 應用程式。您可以使用 Device Farm 主控台來建立專案、上傳 .apk 或 .ipa 檔案、執行一組標準測試，然後檢視結果。

> **Note**
>
> Device Farm 僅適用於 `us-west-2`（奧勒岡）AWS 區域。

主題

先決條件

開始之前，請確定您已完成下列要求：

- 完成「設定」中的步驟。您需要 AWS 帳戶和具有存取 Device Farm 許可的 AWS Identity and Access Management (IAM) 使用者。

- 對於 Android，您可以攜帶 .apk (Android 應用程式套件）檔案，或使用我們提供的範例應用程式。若為 iOS，您需要 .ipa (iOS 應用程式存檔) 檔案。您稍後在此演練中將檔案上傳至 Device Farm。

  > **Note**
  >
  > 請確定您的 .ipa 檔案是針對 iOS 裝置所建置，而非模擬器。

- （選用）您需要從 Device Farm 支援的其中一個測試架構進行測試。您可以將此測試套件上傳到 Device Farm，然後在本演練稍後執行測試。如果您沒有可用的測試套件，您可以指定並執行標準內建測試套件。如需詳細資訊，請參閱AWS Device Farm 中的測試架構和內建測試。

步驟 1：登入 主控台

您可以使用 Device Farm 主控台來建立和管理專案並執行以進行測試。您稍後會在此逐步教學中了解專案與執行。

- 登入 Device Farm 主控台，網址為 https：//https://console.aws.amazon.com/devicefarm。

步驟 2：建立專案

若要在 Device Farm 中測試應用程式，您必須先建立專案。

1. 在導覽窗格中，選擇行動裝置測試，然後選擇專案。
2. 在行動裝置測試專案下，選擇建立專案。
3. 在建立專案下，輸入專案名稱（例如，**`MyDemoProject`**)。
4. 選擇建立。

   主控台會開啟新建立專案的自動化測試頁面。

步驟 3：建立並開始執行

現在您有一個專案，您就可以建立然後開始執行。如需詳細資訊，請參閱執行。

1. 在自動化測試索引標籤上，選擇建立執行。或者，您也可以選取使用教學建立執行來遵循主控台內教學。
2. （選用）在執行設定下，於執行名稱區段中，輸入執行的名稱。如果未提供名稱，則 Device Farm 主控台預設會將您的執行命名為「My Device Farm run」。
3. 在執行設定下，於執行類型區段中，選取您的執行類型。如果您沒有準備好進行測試的應用程式，或者您正在測試 Android 應用程式 (.apk)，請選取 Android 應用程式。如果您要測試 iOS (.ipa) 應用程式，請選取 iOS 應用程式。
4. 在選取應用程式下，在應用程式選取選項區段中，如果您沒有可供測試的應用程式，請選擇選取 Device Farm 提供的範例應用程式。如果您要使用自己的應用程式，請選取上傳自己的應用程式，然後選擇您的應用程式檔案。如果您上傳的是 iOS 應用程式，請務必選擇 iOS device (iOS 裝置)，而非模擬器。
5. 在設定測試下，在選取測試架構區段中，選擇其中一個測試架構或內建測試套件。如需每個選項的詳細資訊，請參閱測試架構和內建測試。

- 如果您尚未封裝 Device Farm 的測試，請選擇內建：模糊以執行標準的內建測試套件。您可以保留事件計數、事件調節和 Randomizer 種子的預設值。如需詳細資訊，請參閱the section called “內建：模糊 (Android 和 iOS)”。
- 如果您有其中一個受支援測試架構的測試套件，請選擇對應的測試架構，然後上傳包含測試的檔案。

6. 在選取裝置下，選擇使用裝置集區和熱門裝置。
7. （選用）若要新增其他組態，請開啟其他組態下拉式清單。在本節中，您可以執行下列任何動作：

   - 若要提供其他資料供 Device Farm 在執行期間使用，請在新增額外資料旁選擇選擇檔案，然後瀏覽並選擇包含資料的 .zip 檔案。
   - 若要在執行期間安裝 Device Farm 的其他應用程式，請在安裝其他應用程式旁選擇選擇檔案，然後瀏覽並選擇包含應用程式的 .apk 或 .ipa 檔案。對於其他您要安裝的應用程式重複此動作。您可以在上傳應用程式之後，藉由拖放它們來變更安裝順序。
   - 若要指定執行期間是否啟用 Wi-Fi、藍牙、GPS 或 NFC，請在 Set radio states (設定無線電狀態) 旁選取適當的方塊。
   - 若要預設執行的裝置經緯度，請在 Device location (裝置位置) 旁輸入座標。
   - 若要預設執行的裝置地區設定，請在裝置地區設定中選擇地區設定。
   - 選取啟用影片錄製以在測試期間錄製影片。
   - 選取啟用應用程式效能資料擷取，以從裝置擷取效能資料。

   > **Note**
   >
   > 設定裝置無線電狀態和地區設定目前僅適用於 Android 原生測試的選項。

   > **Note**
   >
   > 如果您有私有裝置，也會顯示私有裝置特定的組態。

8. 在頁面底部，選擇建立執行以排程執行。

Device Farm 會在裝置可用時立即啟動執行，通常在幾分鐘內。若要檢視執行狀態，請在專案的自動化測試頁面上，選擇執行的名稱。其中一個執行頁面，在裝置下，每個裝置會從裝置資料

表 中
的待定圖示開始，然後
在 測
試開始時切換到執行圖示。當每個測試完成時，主控台會在裝置名稱旁顯示測試結果圖示。所有測試完成後，執行旁的待定圖示會變更為測試結果圖示。

步驟 4：檢視執行的結果

若要檢視執行中的測試結果，請在專案的自動化測試頁面上，選擇執行的名稱。系統會顯示摘要頁面：

- 測試結果總數，依結果排序。
- 具有唯一警告或故障之測試的清單。
- 每個裝置都有測試結果的裝置清單。
- 執行時所擷取的任何螢幕擷取畫面，依裝置分組。
- 下載剖析結果的區段。

如需詳細資訊，請參閱在 Device Farm 中檢視測試報告。

後續步驟

如需 Device Farm 的詳細資訊，請參閱 概念。

在 Device Farm 中購買裝置插槽

您可以使用 Device Farm 主控台、 AWS Command Line Interface (AWS CLI) 或 Device Farm API 來購買裝置插槽。

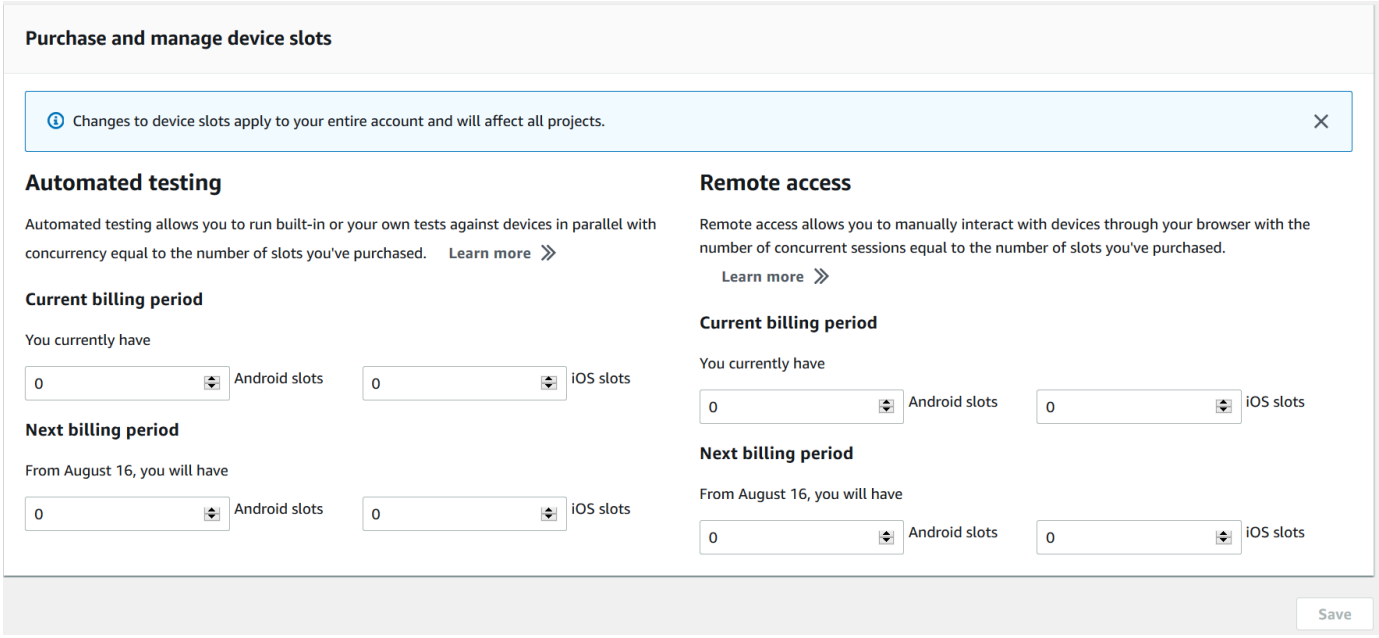
購買裝置插槽 （主控台）

1. 登入 Device Farm 主控台，網址為 https：//https://console.aws.amazon.com/devicefarm。
2. 在導覽窗格中，選擇行動裝置測試，然後選擇裝置插槽。
3. 在購買和管理裝置插槽頁面上，您可以選擇您要購買的自動測試和遠端存取裝置的插槽數量，以建立自己的自訂套件。指定目前和下一個計費期間的槽數量。

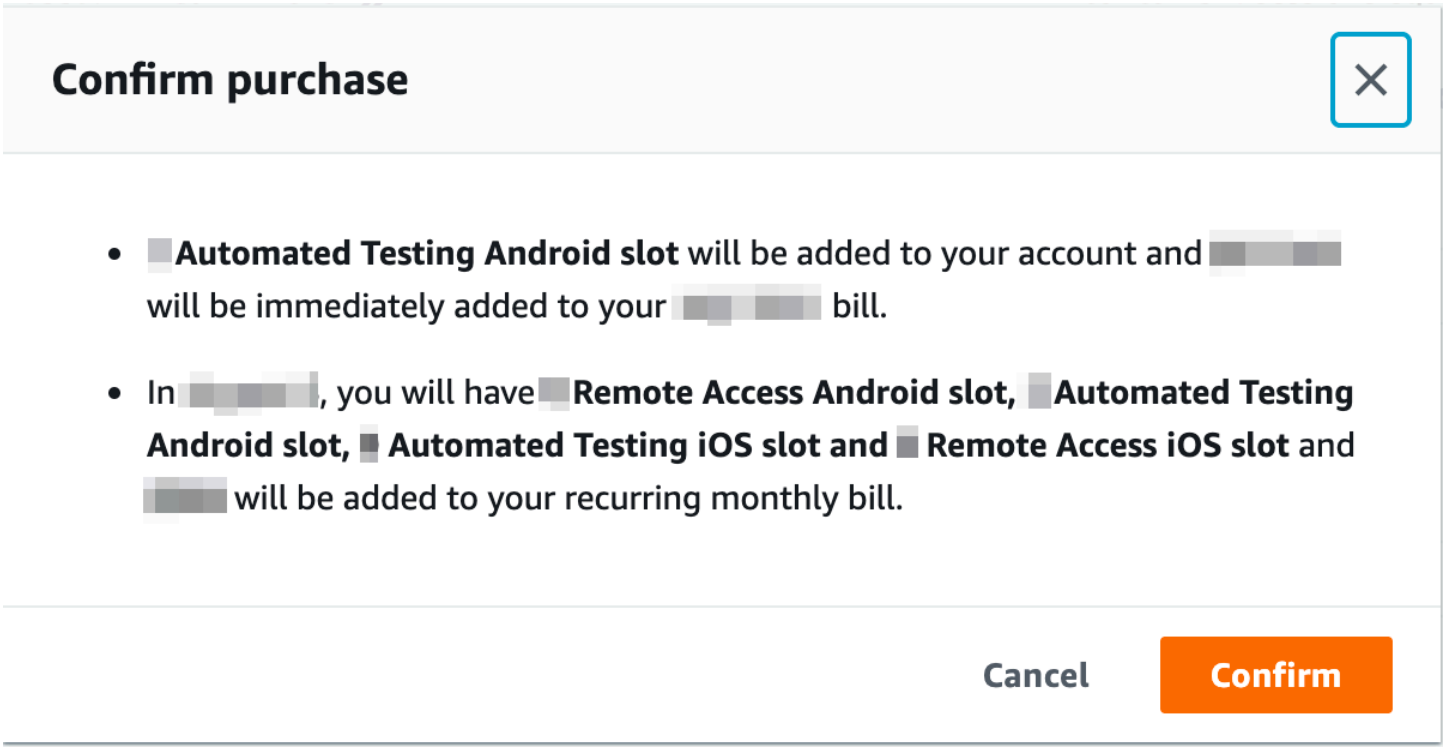
   當您變更槽數量時，文字會動態更新帳單金額。如需詳細資訊，請參閱 AWS Device Farm 定價。

> **Important**
>
> 如果您變更裝置插槽數量，但看到聯絡我們或聯絡我們來購買訊息， AWS 您的帳戶尚未獲准購買您請求的裝置插槽數量。
> 這些選項會提示您傳送電子郵件給 Device Farm 支援團隊。在電子郵件中，指定您要購買的每個裝置類型數量，以及哪個計費週期。

> **Note**
>
> 裝置插槽的變更會套用至您的整個帳戶，並影響所有專案。



4. 選擇 Purchase (購買)。確認購買時段隨即出現。檢閱資訊，然後選擇確認以完成交易。



在購買和管理裝置插槽頁面上，您可以查看目前擁有的裝置插槽數量。如果插槽數量有所增減，則您將看到變更日期後一個月內會擁有的插槽數量。

購買裝置插槽 (AWS CLI)

您可以執行 purchase-offering 命令來購買產品。

若要列出您的 Device Farm 帳戶設定，包括您可以購買的裝置插槽數目上限，以及剩餘的免費試用分鐘數，請執行 get-account-settings命令。輸出結果會類似如下：

```
{
    "accountSettings": {
        "maxSlots": {
            "GUID": 1,
            "GUID": 1,
            "GUID": 1,
            "GUID": 1
        },
        "unmeteredRemoteAccessDevices": {
            "ANDROID": 0,
            "IOS": 0
        },
        "maxJobTimeoutMinutes": 150,
        "trialMinutes": {
            "total": 1000.0,
            "remaining": 954.1
        },
        "defaultJobTimeoutMinutes": 150,
        "awsAccountNumber": "AWS-ACCOUNT-NUMBER",
        "unmeteredDevices": {
            "ANDROID": 0,
            "IOS": 0
        }
    }
}
```

若要列出可供您使用的裝置插槽產品，請執行 list-offerings 命令。您應該會看到類似下列的輸出：

```
{
    "offerings": [
        {
            "recurringCharges": [
                {
                    "cost": {
                        "amount": 250.0,
                        "currencyCode": "USD"
```

```
                },
                "frequency": "MONTHLY"
            }
        ],
        "platform": "IOS",
        "type": "RECURRING",
        "id": "GUID",
        "description": "iOS Unmetered Device Slot"
    },
    {
        "recurringCharges": [
            {
                "cost": {
                    "amount": 250.0,
                    "currencyCode": "USD"
                },
                "frequency": "MONTHLY"
            }
        ],
        "platform": "ANDROID",
        "type": "RECURRING",
        "id": "GUID",
        "description": "Android Unmetered Device Slot"
    },
    {
        "recurringCharges": [
            {
                "cost": {
                    "amount": 250.0,
                    "currencyCode": "USD"
                },
                "frequency": "MONTHLY"
            }
        ],
        "platform": "ANDROID",
        "type": "RECURRING",
        "id": "GUID",
        "description": "Android Remote Access Unmetered Device Slot"
    },
    {
        "recurringCharges": [
            {
                "cost": {
                    "amount": 250.0,
```

```
                    "currencyCode": "USD"
                },
                "frequency": "MONTHLY"
            }
        ],
        "platform": "IOS",
        "type": "RECURRING",
        "id": "GUID",
        "description": "iOS Remote Access Unmetered Device Slot"
    }
]
}
```

若要列出可用的優惠促銷，請執行 list-offering-promotions命令。

> **Note**
>
> 此命令只會傳回您尚未購買的促銷。只要您使用促銷在任何產品中購買一或多個插槽，該促銷就不會再出現在結果中。

您應該會看到類似下列的輸出：

```
{
    "offeringPromotions": [
        {
            "id": "2FREEMONTHS",
            "description": "New device slot customers get 3 months for the price of 1."
        }
    ]
}
```

若要取得產品狀態，請執行 get-offering-status 命令。您應該會看到類似下列的輸出：

```
{
    "current": {
        "GUID": {
            "offering": {
                "platform": "IOS",
                "type": "RECURRING",
                "id": "GUID",
                "description": "iOS Unmetered Device Slot"
```

```
            },
            "quantity": 1
        },
        "GUID": {
            "offering": {
                "platform": "ANDROID",
                "type": "RECURRING",
                "id": "GUID",
                "description": "Android Unmetered Device Slot"
            },
            "quantity": 1
        }
    },
    "nextPeriod": {
        "GUID": {
            "effectiveOn": 1459468800.0,
            "offering": {
                "platform": "IOS",
                "type": "RECURRING",
                "id": "GUID",
                "description": "iOS Unmetered Device Slot"
            },
            "quantity": 1
        },
        "GUID": {
            "effectiveOn": 1459468800.0,
            "offering": {
                "platform": "ANDROID",
                "type": "RECURRING",
                "id": "GUID",
                "description": "Android Unmetered Device Slot"
            },
            "quantity": 1
        }
    }
}
```

renew-offering 和 list-offering-transactions命令也可用於此功能。如需更多資訊，請參閱AWS CLI 參考。

購買裝置插槽 (API)

1. 呼叫 GetAccountSettings 操作來列出您的帳戶設定。

2. 呼叫 ListOfferings 操作，來列出可供您使用的裝置插槽產品。
3. 呼叫 ListOfferingPromotions 操作，即可列出可用的產品促銷。

> **Note**
>
> 此命令只會傳回您尚未購買的促銷。只要您使用產品促銷來購買一或多個插槽，該促銷就不會再出現在結果中。

4. 呼叫 PurchaseOffering 操作來購買產品。
5. 呼叫 GetOfferingStatus 操作來取得產品狀態。

RenewOffering 和 ListOfferingTransactions 命令也可供此功能使用。

如需使用 Device Farm API 的詳細資訊，請參閱 自動化Device Farm。

取消 Device Farm 中的裝置插槽

您可以取消自動測試和遠端存取的裝置插槽數量。如需說明，請參閱下列其中一個章節。下一個帳單週期向您的 帳戶收取的金額會列在帳單期間欄位下方。

如需裝置插槽的詳細資訊，請參閱 在 Device Farm 中購買裝置插槽。

取消裝置插槽 （主控台）

1. 登入 Device Farm 主控台，網址為 https：//https://console.aws.amazon.com/devicefarm。
2. 在導覽窗格中，選擇行動裝置測試，然後選擇裝置插槽。
3. 在購買和管理裝置插槽頁面上，您可以透過減少下一個計費期間下的值，減少自動測試和遠端存取的裝置插槽數量。下一個帳單週期向您的 帳戶收取的金額會列在帳單期間欄位下方。
4. 選擇儲存。確認變更視窗隨即出現。檢閱資訊，然後選擇確認以完成交易。

取消裝置插槽 (AWS CLI)

您可以執行 renew-offering命令來變更下一個計費週期的裝置數量。

取消裝置插槽 (API)

呼叫 RenewOffering 操作來變更帳戶中的裝置數量。

AWS Device Farm 概念

Device Farm 是一種應用程式測試服務，可讓您在 Amazon Web Services () 託管的實體手機和平板電腦上，測試 Android、iOS 和 Web 應用程式並與之互動AWS。

本節說明重要的 Device Farm 概念。

- AWS Device Farm 中的裝置支援
- 在 AWS Device Farm 中測試環境
- 執行
- 應用程式
- AWS Device Farm 中的報告
- 工作階段

如需 Device Farm 中支援測試類型的詳細資訊，請參閱 AWS Device Farm 中的測試架構和內建測試。

AWS Device Farm 中的裝置支援

下列各節提供有關 Device Farm 中裝置支援的資訊。

主題

- 支援的裝置
- 裝置集區
- 私有裝置
- 裝置品牌
- 裝置插槽
- 預先安裝的裝置應用程式
- 裝置功能

支援的裝置

Device Farm 支援數百種獨特的熱門 Android 和 iOS 裝置和作業系統組合。隨著新裝置在市場上推出，清單中的可用裝置也會增加。如需裝置的完整清單，請參閱 AWS 主控台中的互動式裝置清單。

裝置集區

Device Farm 會將其裝置組織成可用於測試的裝置集區。這些裝置集區包含相關裝置，例如僅在 Android 上執行或僅在 iOS 上執行的裝置。Device Farm 提供精選的裝置集區，例如適用於熱門裝置的集區。您也可以建立混合公有和私有裝置的裝置集區。

私有裝置

私有裝置允許您確切指定硬體和軟體組態，以滿足您的測試需求。某些組態，例如根 Android 裝置，可以支援為私有裝置。每個私有裝置都是 Device Farm 在 Amazon 資料中心代表您部署的實體裝置。您的私有裝置專為您提供自動和手動測試。在您選擇終止訂閱後，硬體就會從我們的環境中移除。如需詳細資訊，請參閱私有裝置和 AWS Device Farm 中的私有裝置。

裝置品牌

Device Farm 會在來自各種 OEMs 的實體行動裝置和平板電腦裝置上進行測試。

裝置插槽

裝置插槽會對應到同時執行數量，您購買的裝置插槽數量，會決定您可以在測試或遠端存取工作階段中執行的裝置數量。

有兩種裝置插槽類型：

- 遠端存取裝置插槽 是您可以在遠端存取工作階段中同時執行的項目。

  如果您只有一個遠端存取裝置插槽，您一次只能執行一個遠端存取工作階段。如果您購買額外遠端測試裝置插槽，您就可以同時執行多個工作階段。
- 自動測試裝置插槽 是您可以同時執行測試的項目。

  如果您只有一個自動測試裝置插槽，您一次只能在一個裝置上執行測試。如果您購買額外的自動測試裝置插槽，您可以在多個裝置上同時執行測試，就能更快獲得測試結果。

您可以根據裝置系列購買裝置插槽 (用於自動測試的 Android 或 iOS 裝置，以及用於遠端存取的 Android 或 iOS 裝置)。如需詳細資訊，請參閱 Device Farm 定價。

預先安裝的裝置應用程式

Device Farm 中的裝置包含製造商和電信業者已安裝的少量應用程式。

裝置功能

所有裝置都有網際網路連線。裝置未配有電信業者連線，因此無法撥打電話或傳送 SMS。

您可以使用任何支援前後鏡頭的裝置拍照。由於裝置掛載的方式，照片可能會看起來較暗且模糊。

Google Play Services 和 Google Chrome 安裝在 Android 裝置上。

在 AWS Device Farm 中測試環境

AWS Device Farm 提供自訂和標準測試環境，以執行您的自動化測試。您可以選擇自訂測試環境，來完全控制您的自動化測試。或者，您可以選擇 Device Farm 預設標準測試環境，提供自動化測試套件中每個測試的精細報告。

主題

標準測試環境

當您在標準環境中執行測試時，Device Farm 會提供測試套件中每個案例的詳細日誌和報告。您可以檢視每個測試的效能資料、視訊、螢幕擷取畫面和日誌，以在應用程式中找出問題並加以修正。

> **Note**
>
> 由於 Device Farm 在標準環境中提供精細的報告，因此測試執行時間可能比在本機執行測試的時間更長。如果您想要更快的執行時間，請在自訂測試環境中執行您的測試。

自訂測試環境

當您自訂測試環境時，您可以指定 Device Farm 應執行的命令來執行您的測試。這可確保 Device Farm 上的測試執行的方式類似於本機電腦上執行的測試。在這個模式上執行您的測試，也會啟用此測試的即時日誌和視訊串流。在自訂測試環境中執行測試時，您不會取得每個測試案例的精細報告。如需詳細資訊，請參閱AWS Device Farm 中的自訂測試環境。

當您使用 Device Farm 主控台、 或 Device Farm API 建立測試執行時 AWS CLI，您可以選擇使用自訂測試環境。

如需詳細資訊，請參閱使用 和 上傳自訂測試規格 AWS CLI在 Device Farm 中建立測試執行。

在 AWS Device Farm 中執行

下列各節包含 Device Farm 中執行的相關資訊。

Device Farm 中的執行代表應用程式的特定建置，以及要在特定裝置集上執行的特定一組測試。執行會產生報告，其中包含執行結果的相關資訊。執行包含一或多個任務。

主題

執行組態

在執行過程中，您可以提供 Device Farm 可用來覆寫目前裝置設定的設定。這些包括經緯度座標、額外資料（包含在 .zip 檔案中）和輔助應用程式（應該在要測試的應用程式之前安裝的應用程式）。在 Android 上，可以變更一些額外的設定，例如地區設定和無線電狀態（藍牙、GPS、NFC 和 Wi-Fi)。

執行檔案保留

Device Farm 會儲存您的應用程式和檔案 30 天，然後將其從系統中刪除。不過，您可以隨時刪除檔案。

Device Farm 會儲存您的執行結果、日誌和螢幕擷取畫面 400 天，然後將其從系統中刪除。

執行裝置狀態

Device Farm 一律會重新啟動裝置，再讓裝置可供下一個任務使用。

平行執行

Device Farm 會平行執行測試。

設定執行逾時

您可以設定一值，表示在您停止每個裝置的執行測試前，測試執行應該執行多長時間。例如，如果您的測試需要每個裝置 20 分鐘來完成，則您應該選擇每個裝置 30 分鐘逾時。

如需詳細資訊，請參閱在 AWS Device Farm 中設定測試執行的執行逾時。

執行中的廣告

建議您先從應用程式移除廣告，再將它們上傳至 Device Farm。我們無法保證執行期間顯示廣告。

執行中的媒體

您可以提供媒體或其他資料來伴隨您的應用程式。提供額外資料的 .zip 檔案，其大小不得超過 4 GB。

執行的常見任務

如需詳細資訊，請參閱在 Device Farm 中建立測試執行及測試在 AWS Device Farm 中執行。

AWS Device Farm 中的應用程式

下列各節包含 Device Farm 中應用程式行為的相關資訊。

主題

- 檢測應用程式
- 在執行中重新簽署應用程式
- 執行中混淆的應用程式

檢測應用程式

您不需要檢測您的應用程式，也不需要為 Device Farm 提供應用程式的原始碼。您可以提交未經修改的 Android 應用程式。iOS 應用程式必須搭配 iOS Device (iOS 裝置) 目標進行建置，而非搭配模擬器。

在執行中重新簽署應用程式

對於 iOS 應用程式，您不需要將任何 Device Farm UUIDs新增至您的佈建設定檔。Device Farm 會將內嵌佈建設定檔取代為萬用字元設定檔，然後重新簽署應用程式。如果您提供輔助資料，Device Farm 會在 Device Farm 安裝之前將其新增至應用程式的套件，以便輔助存在於應用程式的沙盒中。重新簽署應用程式會移除如下授權：App Group、Associated Domains、Game Center、HealthKit、HomeKit、Wireless Accessory Configuration、In-App Purchase、Inter-App Audio、Apple Pay、Push Notifications，以及 VPN Configuration & Control。

對於 Android 應用程式，Device Farm 會重新簽署應用程式。這可能會中斷取決於應用程式簽章的功能 (例如 Google Maps Android API)，或其可能從 DexGuard 這類產品中觸發反盜版或反竄改偵測。

執行中混淆的應用程式

對於 Android 應用程式，如果應用程式混淆，如果您使用 ProGuard，仍然可以使用 Device Farm 對其進行測試。不過，如果您使用 DexGuard 搭配反盜版措施，則 Device Farm 無法針對應用程式重新簽署和執行測試。

AWS Device Farm 中的報告

下列各節提供 Device Farm 測試報告的相關資訊。

主題

報告保留

Device Farm 會儲存您的報告 400 天。這些報告包含中繼資料、日誌、螢幕擷取畫面和效能資料。

報告元件

Device Farm 中的報告包含傳遞和失敗資訊、當機報告、測試和裝置日誌、螢幕擷取畫面和效能資料。

這些報告包含詳細的每一裝置資料和簡要的結果，例如特定問題的出現次數。

報告中的日誌

這些報告包含 Android 測試的完整 logcat 擷取，以及 iOS 測試的完整裝置主控台日誌。

報告的常見任務

如需詳細資訊，請參閱在 Device Farm 中檢視測試報告。

AWS Device Farm 中的工作階段

您可以使用 Device Farm 透過遠端存取工作階段對 Android 和 iOS 應用程式執行互動式測試。這包括 Web 瀏覽器中的手動互動，以及從本機用戶端對遠端裝置執行 Appium 測試。開發人員可以在特定裝置上重現應用程式或 Appium 測試的問題，以隔離和解決問題。

主題

- 支援遠端存取的裝置
- 工作階段檔案保留
- 檢測應用程式
- 在工作階段中重新簽署應用程式
- 工作階段中的混淆應用程式

支援遠端存取的裝置

Device Farm 支援多種獨特的熱門 Android 和 iOS 裝置。隨著新裝置在市場上推出，清單中的可用裝置也會增加。Device Farm 主控台會顯示目前可供遠端存取的 Android 和 iOS 裝置清單。如需詳細資訊，請參閱AWS Device Farm 中的裝置支援。

工作階段檔案保留

Device Farm 會儲存您的應用程式和檔案 30 天，然後將其從系統中刪除。不過，您可以隨時刪除檔案。

Device Farm 會儲存您的工作階段日誌和擷取的影片 400 天，然後從系統中刪除它們。

檢測應用程式

您不需要檢測應用程式，也不需要為 Device Farm 提供應用程式的原始程式碼。您可以提交未經修改的 Android 和 iOS 應用程式。

在工作階段中重新簽署應用程式

Device Farm 會重新簽署 Android 和 iOS 應用程式。這可能會中斷取決於應用程式簽章的功能。例如，Android 的 Google Maps API 取決於您應用程式的簽章。應用程式重新簽署也可以從 DexGuard for Android 裝置的這類產品中觸發反盜版或反竄改偵測。

工作階段中的混淆應用程式

對於 Android 應用程式，如果應用程式混淆，如果您使用 ProGuard，仍然可以使用 Device Farm 對其進行測試。不過，如果您使用 DexGuard 搭配反盜版措施，則 Device Farm 無法重新簽署應用程式。

AWS Device Farm 中的專案

Device Farm 中的專案代表 Device Farm 中的邏輯工作區，其中包含執行，每個單一應用程式針對一或多個裝置的測試各執行一次。專案可讓您以您選擇的任何方式組織工作區。例如，每個應用程式標題可以有一個專案，或者每個平台可以有一個專案。您可以視需要建立任意數量的專案。

您可以使用 AWS Device Farm 主控台、 AWS Command Line Interface (AWS CLI) 或 AWS Device Farm API 來處理專案。

主題

在 AWS Device Farm 中建立專案

您可以使用 AWS Device Farm 主控台 AWS CLI或 AWS Device Farm API 來建立專案。

先決條件

- 完成「設定」中的步驟。

建立專案 （主控台）

1. 登入 Device Farm 主控台，網址為 https：//https://console.aws.amazon.com/devicefarm。
2. 在 Device Farm 導覽面板上，選擇行動裝置測試，然後選擇專案。
3. 選擇 New project (新專案)。
4. 輸入專案的名稱。或者，您可以提供以下一個或多個參數，然後選擇提交。

   虛擬私有雲端 (VPC) 設定

   選取要套用至待測裝置的 VPC、子網路和安全群組及其配對的測試主機。此功能僅支援私有裝置。如需詳細資訊，請參閱AWS Device Farm 中的 VPC-ENI。

   執行角色 ARN

   由測試執行器在自訂測試環境中擔任的 IAM 角色。如需詳細資訊，請參閱使用 IAM 執行角色存取 AWS 資源。

環境變數

要插入測試執行器程序環境的一或多個變數。以 "DEVICEFARM_" 開頭的變數名稱會保留供服務使用。我們建議不要在這些環境變數中存放敏感值，而是建議您在測試期間使用 IAM 執行角色從 AWS Secrets Manager 擷取這些值。

建立專案 (AWS CLI)

- 執行 create-project，並指定專案名稱。

  範例：

  ```
  aws devicefarm create-project --name MyProjectName
  ```

  AWS CLI 回應包含專案的 Amazon Resource Name (ARN)。

  ```
  {
      "project": {
          "name": "MyProjectName",
          "arn": "arn:aws:devicefarm:us-west-2:123456789101:project:5e01a8c7-
  c861-4c0a-b1d5-12345EXAMPLE",
          "created": 1535675814.414
      }
  }
  ```

  如需詳細資訊，請參閱create-project及AWS CLI 參考。

建立專案 (API)

- 呼叫 `CreateProject` API。

如需使用 Device Farm API 的詳細資訊，請參閱 自動化Device Farm。

在 AWS Device Farm 中檢視專案清單

您可以使用 AWS Device Farm 主控台 AWS CLI或 AWS Device Farm API 來檢視專案清單。

主題

先決條件

- 在 Device Farm 中建立至少一個專案。請遵循在 AWS Device Farm 中建立專案中的指示，然後返回此頁面。

檢視專案清單（主控台）

1. 登入 Device Farm 主控台，網址為 https：//https://console.aws.amazon.com/devicefarm。
2. 若要尋找可用專案的清單，請執行下列動作：
   - 對於行動裝置測試專案，在 Device Farm 導覽功能表上，選擇行動裝置測試，然後選擇專案。
   - 對於桌面瀏覽器測試專案，在 Device Farm 導覽功能表上，選擇桌面瀏覽器測試，然後選擇專案。

檢視專案清單 (AWS CLI)

- 若要檢視專案清單，請執行 list-projects 命令。

  若要檢視單一專案的相關資訊，請執行 get-project 命令。

如需搭配 使用 Device Farm 的詳細資訊 AWS CLI，請參閱 AWS CLI 參考。

檢視專案清單 (API)

- 若要檢視專案清單，請呼叫 `ListProjects` API。

  若要檢視單一專案的相關資訊，請呼叫 `GetProject` API。

如需 AWS Device Farm API 的相關資訊，請參閱 自動化Device Farm。

測試在 AWS Device Farm 中執行

Device Farm 中的執行代表應用程式的特定組建，具有一組特定的測試，可在一組特定的裝置上執行。執行會產生報告，其中包含執行結果的相關資訊。執行包含一或多個任務。如需詳細資訊，請參閱執行。

您可以使用 AWS Device Farm 主控台、 AWS Command Line Interface (AWS CLI) 或 AWS Device Farm API 來使用測試執行。

主題

在 Device Farm 中建立測試執行

您可以使用 Device Farm 主控台 AWS CLI或 Device Farm API 來建立測試執行。您也可以使用支援的外掛程式，例如 Device Farm 的 Jenkins 或 Gradle 外掛程式。如需外掛程式的詳細資訊，請參閱 工具和外掛程式。如需執行的詳細資訊，請參閱執行。

主題

先決條件

您必須在 Device Farm 中擁有專案。請遵循在 AWS Device Farm 中建立專案中的指示，然後返回此頁面。

建立測試執行（主控台）

1. 登入 Device Farm 主控台，網址為 https：//https://console.aws.amazon.com/devicefarm。
2. 在導覽窗格中，選擇行動裝置測試，然後選擇專案。
3. 如果您已有專案，則可以將您的測試上傳至其中。否則，請選擇新增專案，輸入專案名稱，然後選擇建立。
4. 開啟您的專案，然後選擇建立執行。
5. （選用）在執行設定下，於執行名稱區段中，輸入執行的名稱。如果未提供名稱，則 Device Farm 主控台預設會將您的執行命名為「My Device Farm run」。
6. （選用）在執行設定下，在任務逾時區段中，您可以指定測試執行的執行逾時。如果您使用的是無限制的測試槽，請確認已在 Billing 方法下選取 Unmetered。
7. 在執行設定下，於執行類型區段中，選取您的執行類型。如果您沒有準備好進行測試的應用程式，或者您正在測試 Android 應用程式 (.apk)，請選取 Android 應用程式。如果您要測試 iOS (.ipa) 應用程式，請選取 iOS 應用程式。如果您想要測試 Web 應用程式，請選取 Web 應用程式。
8. 在選取應用程式下，在應用程式選取選項區段中，如果您沒有可供測試的應用程式，請選擇選取 Device Farm 提供的範例應用程式。如果您要使用自己的應用程式，請選取上傳自己的應用程式，然後選擇您的應用程式檔案。如果您上傳的是 iOS 應用程式，請務必選擇 iOS device (iOS 裝置)，而非模擬器。
9. 在設定測試下，選擇其中一個可用的測試架構。

   > **Note**
   >
   > 如果沒有任何可用的測試，請選擇 Built-in: Fuzz (內建：Fuzz)，以執行標準內建測試套件。如果您選擇 Built-in: Fuzz (內建：Fuzz)，且出現 Event count (事件計數)、Event throttle (事件調節) 和 Randomizer seed (亂數種子) 方塊，則可變更或保留值。

   如需可用測試套件的詳細資訊，請參閱AWS Device Farm 中的測試架構和內建測試。
10. 如果您未選擇內建：模糊，請選取選取測試套件下的選擇檔案。瀏覽並選擇包含測試的檔案。

11. 針對您的測試環境，選擇在我們的標準環境中執行您的測試，或在自訂環境中執行您的測試。如需詳細資訊，請參閱在 AWS Device Farm 中測試環境。
12. 如果您使用的是自訂測試環境，您可以選擇執行下列動作：
    - 如果您想要在自訂測試環境中編輯預設測試規格，請選擇 Edit (編輯) 來更新預設 YAML 規格。
    - 如果您變更了測試規格，請選擇另存新檔來更新它。
    - 您可以設定環境變數。此處提供的變數將優先於父專案上可能設定的任何 。
13. 在選取裝置下，執行下列其中一項操作：
    - 若要選擇內建的裝置集區來對其執行測試，請對於 Device pool (裝置集區)，選擇 Top Devices (熱門裝置)。
    - 若要建立您自己的裝置集區來對其執行測試，請依照建立裝置集區中的指示，然後返回此頁面。
    - 如果您先前已建立自己的裝置集區，請對於 Device pool (裝置集區)，選擇您的裝置集區。
    - 選取手動選取裝置，然後選擇您要執行的目標裝置。此組態將不會儲存。

    如需詳細資訊，請參閱AWS Device Farm 中的裝置支援。
14. （選用）若要新增其他組態，請開啟其他組態下拉式清單。在本節中，您可以執行下列任何動作：
    - 若要提供執行角色 ARN，或覆寫父專案上設定的執行角色 ARN，請使用 Exectuion 角色 ARN 欄位。
    - 若要提供其他資料供 Device Farm 在執行期間使用，請在新增額外資料旁選擇選擇檔案，然後瀏覽並選擇包含資料的 .zip 檔案。
    - 若要在執行期間安裝 Device Farm 的其他應用程式，請在安裝其他應用程式旁選擇選擇檔案，然後瀏覽並選擇包含應用程式的 .apk 或 .ipa 檔案。對於其他您要安裝的應用程式重複此動作。您可以在上傳應用程式之後，藉由拖放它們來變更安裝順序。
    - 若要指定執行期間是否啟用 Wi-Fi、藍牙、GPS 或 NFC，請在 Set radio states (設定無線電狀態) 旁選取適當的方塊。
    - 若要預設執行的裝置經緯度，請在 Device location (裝置位置) 旁輸入座標。
    - 若要預設執行的裝置地區設定，請在裝置地區設定中選擇地區設定。
    - 選取啟用影片錄製以在測試期間錄製影片。
    - 選取啟用應用程式效能資料擷取，以從裝置擷取效能資料。

> **Note**
>
> 設定裝置無線電狀態和地區設定目前僅適用於 Android 原生測試的選項。

> **Note**
>
> 如果您有私有裝置，也會顯示私有裝置特定的組態。

15. 在頁面底部，選擇建立執行以排程執行。

Device Farm 會在裝置可用時立即啟動執行，通常在幾分鐘內。在測試執行期間，Device Farm 主控台會在執行資料

表 中

顯示待處理圖示。執行中的每個裝置也會以待定圖示開頭，然後

在 測

試開始時切換到執行中圖示。當每個測試完成時，裝置名稱旁會顯示測試結果圖示。完成所有測試後，執行旁的待定圖示會變更為測試結果圖示。

如果您想要停止測試執行，請參閱 在 AWS Device Farm 中停止執行。

建立測試執行 (AWS CLI)

您可以使用 AWS CLI 來建立測試執行。

主題

- 步驟 1：選擇專案
- 步驟 2：選擇裝置集區
- 步驟 3：上傳您的應用程式檔案
- 步驟 4：上傳您的測試指令碼套件
- 步驟 5：（選用）上傳您的自訂測試規格
- 步驟 6：排程測試執行

步驟 1：選擇專案

您必須將測試執行與 Device Farm 專案建立關聯。

1. 若要列出 Device Farm 專案，請執行 list-projects。如果您沒有專案，請參閱在 AWS Device Farm 中建立專案。

   範例：

   ```
   aws devicefarm list-projects
   ```

   回應包含 Device Farm 專案的清單。

   ```
   {
       "projects": [
           {
               "name": "MyProject",
               "arn": "arn:aws:devicefarm:us-west-2:123456789101:project:5e01a8c7-
   c861-4c0a-b1d5-12345EXAMPLE",
               "created": 1503612890.057
           }
       ]
   }
   ```

2. 選擇要與測試執行建立關聯的專案，並記下其 Amazon Resource Name (ARN)。

步驟 2：選擇裝置集區

您必須選擇要與測試執行建立關聯的裝置集區。

1. 若要檢視裝置集區，請執行 list-device-pools 並指定專案 ARN。

   範例：

   ```
   aws devicefarm list-device-pools --arn arn:MyProjectARN
   ```

   回應包含內建的 Device Farm 裝置集區，例如 Top Devices，以及先前為此專案建立的任何裝置集區：

   ```
   {
   ```

```
    "devicePools": [
        {
            "rules": [
                {
                    "attribute": "ARN",
                    "operator": "IN",
                    "value": "[\"arn:aws:devicefarm:us-west-2::device:example1\",
\"arn:aws:devicefarm:us-west-2::device:example2\",\"arn:aws:devicefarm:us-
west-2::device:example3\"]"
                }
            ],
            "type": "CURATED",
            "name": "Top Devices",
            "arn": "arn:aws:devicefarm:us-west-2::devicepool:example",
            "description": "Top devices"
        },
        {
            "rules": [
                {
                    "attribute": "PLATFORM",
                    "operator": "EQUALS",
                    "value": "\"ANDROID\""
                }
            ],
            "type": "PRIVATE",
            "name": "MyAndroidDevices",
            "arn": "arn:aws:devicefarm:us-west-2:605403973111:devicepool:example2"
        }
    ]
}
```

2. 選擇裝置集區，並記下其 ARN。

   您也可以建立裝置集區，然後返回此步驟。如需詳細資訊，請參閱建立裝置集區 (AWS CLI)。

步驟 3：上傳您的應用程式檔案

若要建立上傳請求並取得 Amazon Simple Storage Service (Amazon S3) 預先簽章的上傳 URL，您需要：

- 您的專案 ARN。
- 您的應用程式檔案名稱。

- 上傳的類型。

如需詳細資訊，請參閱create-upload。

1. 若要上傳檔案，請搭配 `--project-arn`、`--name` 和 `--type` 參數執行 create-upload。

   此範例會建立 Android 應用程式的上傳：

   ```
   aws devicefarm create-upload --project-arn arn:MyProjectArn --name MyAndroid.apk --
   type ANDROID_APP
   ```

   回應包含您的應用程式上傳 ARN 和預先簽章的 URL。

   ```
   {
       "upload": {
           "status": "INITIALIZED",
           "name": "MyAndroid.apk",
           "created": 1535732625.964,
           "url": "https://prod-us-west-2-uploads.s3-us-west-2.amazonaws.com/
   ExampleURL",
           "type": "ANDROID_APP",
           "arn": "arn:aws:devicefarm:us-west-2:123456789101:upload:5e01a8c7-
   c861-4c0a-b1d5-12345EXAMPLE"
       }
   }
   ```

2. 記下應用程式上傳 ARN 和預先簽章的 URL。
3. 使用 Amazon S3 預先簽章的 URL 上傳您的應用程式檔案。此範例會使用 curl 來上傳 Android .apk 檔案：

   ```
   curl -T MyAndroid.apk "https://prod-us-west-2-uploads.s3-us-west-2.amazonaws.com/
   ExampleURL"
   ```

   如需詳細資訊，請參閱《Amazon Simple Storage Service 使用者指南》中的使用預先簽章URLs上傳物件。
4. 若要檢查應用程式的上傳狀態，請執行 get-upload，並指定上傳的應用程式 ARN。

   ```
   aws devicefarm get-upload --arn arn:MyAppUploadARN
   ```

等到回應中的狀態為 SUCCEEDED，然後再上傳您的測試指令碼套件。

```
{
    "upload": {
        "status": "SUCCEEDED",
        "name": "MyAndroid.apk",
        "created": 1535732625.964,
        "url": "https://prod-us-west-2-uploads.s3-us-west-2.amazonaws.com/
ExampleURL",
        "type": "ANDROID_APP",
        "arn": "arn:aws:devicefarm:us-west-2:123456789101:upload:5e01a8c7-
c861-4c0a-b1d5-12345EXAMPLE",
        "metadata": "{"valid": true}"
    }
}
```

步驟 4：上傳您的測試指令碼套件

接下來，您會上傳您的測試指令碼套件。

1. 若要建立您的上傳請求並取得 Amazon S3 預先簽章的上傳 URL，create-upload請使用 `--project-arn`、 `--name`和 `--type` 參數執行 。

   此範例會建立 Appium Java TestNG 測試套件上傳：

   ```
   aws devicefarm create-upload --project-arn arn:MyProjectARN --name MyTests.zip --
   type APPIUM_JAVA_TESTNG_TEST_PACKAGE
   ```

   回應包含您的測試套件上傳 ARN 和預先簽章的 URL。

   ```
   {
       "upload": {
           "status": "INITIALIZED",
           "name": "MyTests.zip",
           "created": 1535738627.195,
           "url": "https://prod-us-west-2-uploads.s3-us-west-2.amazonaws.com/
   ExampleURL",
           "type": "APPIUM_JAVA_TESTNG_TEST_PACKAGE",
           "arn": "arn:aws:devicefarm:us-west-2:123456789101:upload:5e01a8c7-
   c861-4c0a-b1d5-12345EXAMPLE"
   ```

```
    }
}
```

2. 記下測試套件上傳的 ARN 和預先簽章的 URL。
3. 使用 Amazon S3 預先簽章的 URL 上傳您的測試指令碼套件檔案。此範例會使用 curl 來上傳壓縮的 Appium TestNG 指令碼檔案：

```
curl -T MyTests.zip "https://prod-us-west-2-uploads.s3-us-west-2.amazonaws.com/
ExampleURL"
```

4. 若要檢查測試指令碼套件上傳的狀態，請執行 get-upload，並指定測試套件上傳的 ARN，其來自步驟 1。

```
aws devicefarm get-upload --arn arn:MyTestsUploadARN
```

等到回應中的狀態為 SUCCEEDED，然後再繼續下一步 (選用步驟)。

```
{
    "upload": {
        "status": "SUCCEEDED",
        "name": "MyTests.zip",
        "created": 1535738627.195,
        "url": "https://prod-us-west-2-uploads.s3-us-west-2.amazonaws.com/
ExampleURL",
        "type": "APPIUM_JAVA_TESTNG_TEST_PACKAGE",
        "arn": "arn:aws:devicefarm:us-west-2:123456789101:upload:5e01a8c7-
c861-4c0a-b1d5-12345EXAMPLE",
        "metadata": "{"valid": true}"
    }
}
```

步驟 5：（選用）上傳您的自訂測試規格

如果您是在標準測試環境中執行測試，則可跳過此步驟。

Device Farm 會維護每個支援測試類型的預設測試規格檔案。接著，您會下載預設測試規格，並將其用來建立自訂測試規格，以在自訂測試環境中執行您的測試。如需詳細資訊，請參閱[在 AWS Device Farm 中測試環境]。

1. 若要尋找預設測試規格的上傳 ARN，請執行 list-uploads 並指定專案 ARN。

```
aws devicefarm list-uploads --arn arn:MyProjectARN
```

回應會包含每個預設測試規格的項目：

```
{
    "uploads": [
        {

            {
                "status": "SUCCEEDED",
                "name": "Default TestSpec for Android Appium Java TestNG",
                "created": 1529498177.474,
                "url": "https://prod-us-west-2-uploads.s3-us-west-2.amazonaws.com/
ExampleURL",
                "type": "APPIUM_JAVA_TESTNG_TEST_SPEC",
                "arn": "arn:aws:devicefarm:us-west-2:123456789101:upload:5e01a8c7-
c861-4c0a-b1d5-12345EXAMPLE"
            }
        }
    ]
}
```

2. 從清單中選擇您的預設測試規格。請記下上傳 ARN。
3. 執行 get-upload 並指定上傳 ARN，即可下載預設測試規格。

   範例：

   ```
   aws devicefarm get-upload --arn arn:MyDefaultTestSpecARN
   ```

   回應包含預先簽章的 presigned URL，其中您可以下載預設測試規格。
4. 此範例會使用 curl 來下載預設測試規格，並將其儲存為 `MyTestSpec.yml`：

   ```
   curl "https://prod-us-west-2-uploads.s3-us-west-2.amazonaws.com/ExampleURL" >
    MyTestSpec.yml
   ```
5. 您可以編輯預設測試規格，以符合您的測試要求，然後在未來測試執行中使用您已修改的測試規格。請跳過此步驟以使用預設測試規格，如同在自訂測試環境中一般。
6. 若要建立自訂測試規格的上傳，請執行 create-upload，並指定測試規格名稱、測試規格類型和專案 ARN。

此範例會建立 Appium Java TestNG 自訂測試規格的上傳：

```
aws devicefarm create-upload --name MyTestSpec.yml --type
 APPIUM_JAVA_TESTNG_TEST_SPEC --project-arn arn:MyProjectARN
```

回應包含測試規格上傳 ARN 和預先簽章的 URL：

```
{
    "upload": {
        "status": "INITIALIZED",
        "category": "PRIVATE",
        "name": "MyTestSpec.yml",
        "created": 1535751101.221,
        "url": "https://prod-us-west-2-uploads.s3-us-west-2.amazonaws.com/
ExampleURL",
        "type": "APPIUM_JAVA_TESTNG_TEST_SPEC",
        "arn": "arn:aws:devicefarm:us-west-2:123456789101:upload:5e01a8c7-
c861-4c0a-b1d5-12345EXAMPLE"
    }
}
```

7. 記下測試規格上傳的 ARN 和預先簽章的 URL。
8. 使用 Amazon S3 預先簽章的 URL 上傳您的測試規格檔案。此範例會使用 curl 來上傳 Appium JavaTestNG 測試規格：

```
curl -T MyTestSpec.yml "https://prod-us-west-2-uploads.s3-us-west-2.amazonaws.com/
ExampleURL"
```

9. 若要檢查測試規格上傳的狀態，請執行 get-upload，並指定上傳 ARN。

```
aws devicefarm get-upload --arn arn:MyTestSpecUploadARN
```

等到回應中的狀態變成 SUCCEEDED，再排定測試執行。

```
{
    "upload": {
        "status": "SUCCEEDED",
        "name": "MyTestSpec.yml",
        "created": 1535732625.964,
```

```
        "url": "https://prod-us-west-2-uploads.s3-us-west-2.amazonaws.com/
ExampleURL",
        "type": "APPIUM_JAVA_TESTNG_TEST_SPEC",
        "arn": "arn:aws:devicefarm:us-west-2:123456789101:upload:5e01a8c7-
c861-4c0a-b1d5-12345EXAMPLE",
        "metadata": "{"valid": true}"
    }
}
```

若要更新自訂測試規格，請執行 update-upload，並指定測試規格的上傳 ARN。如需詳細資訊，請參閱update-upload。

步驟 6：排程測試執行

若要使用 排程測試執行 AWS CLI，請執行 schedule-run，指定：

- 來自步驟 1 的專案 ARN。
- 來自步驟 2 的裝置集區 ARN。
- 來自步驟 3 的應用程式上傳 ARN。
- 來自步驟 4 的測試套件上傳 ARN。

如果您是在自訂測試環境中執行測試，則也需要來自步驟 5 的測試規格 ARN。

在標準測試環境中排定執行

- 執行 schedule-run，並指定專案 ARN、裝置集區 ARN、應用程式上傳 ARN，以及測試套件資訊。

  範例：

  ```
  aws devicefarm schedule-run --project-arn arn:MyProjectARN --app-
  arn arn:MyAppUploadARN --device-pool-arn arn:MyDevicePoolARN --name MyTestRun --
  test type=APPIUM_JAVA_TESTNG,testPackageArn=arn:MyTestPackageARN
  ```

  回應包含一個執行 ARN，您可以用來檢查測試執行的狀態。

  ```
  {
      "run": {
          "status": "SCHEDULING",
  ```

```
        "appUpload": "arn:aws:devicefarm:us-west-2:123456789101:upload:5e01a8c7-
c861-4c0a-b1d5-12345appEXAMPLE",
        "name": "MyTestRun",
        "radios": {
            "gps": true,
            "wifi": true,
            "nfc": true,
            "bluetooth": true
        },
        "created": 1535756712.946,
        "totalJobs": 179,
        "completedJobs": 0,
        "platform": "ANDROID_APP",
        "result": "PENDING",
        "devicePoolArn": "arn:aws:devicefarm:us-
west-2:123456789101:devicepool:5e01a8c7-c861-4c0a-b1d5-12345devicepoolEXAMPLE",
        "jobTimeoutMinutes": 150,
        "billingMethod": "METERED",
        "type": "APPIUM_JAVA_TESTNG",
        "testSpecArn": "arn:aws:devicefarm:us-west-2:123456789101:upload:5e01a8c7-
c861-4c0a-b1d5-12345specEXAMPLE",
        "arn": "arn:aws:devicefarm:us-west-2:123456789101:run:5e01a8c7-c861-4c0a-
b1d5-12345runEXAMPLE",
        "counters": {
            "skipped": 0,
            "warned": 0,
            "failed": 0,
            "stopped": 0,
            "passed": 0,
            "errored": 0,
            "total": 0
        }
    }
}
```

如需詳細資訊，請參閱schedule-run。

在自訂測試環境中排定執行

- 此處步驟與標準測試環境的步驟幾乎相同，但 `--test` 參數中需加入額外的 `testSpecArn` 屬性。

範例：

```
aws devicefarm schedule-run --project-arn arn:MyProjectARN --app-
arn arn:MyAppUploadARN --device-pool-arn arn:MyDevicePoolARN --name MyTestRun --
test
 testSpecArn=arn:MyTestSpecUploadARN,type=APPIUM_JAVA_TESTNG,testPackageArn=arn:MyTestPacka
```

檢查測試執行的狀態

- 執行 get-run 命令並指定執行 ARN：

```
aws devicefarm get-run --arn arn:aws:devicefarm:us-
west-2:111122223333:run:5e01a8c7-c861-4c0a-b1d5-12345runEXAMPLE
```

如需詳細資訊，請參閱get-run。如需搭配 使用 Device Farm 的詳細資訊 AWS CLI，請參閱 AWS CLI 參考。

建立測試執行 (API)

這些步驟與 AWS CLI 章節中所述的步驟相同。請參閱 建立測試執行 (AWS CLI)。

您需要此資訊，才能呼叫 ScheduleRun API：

- 專案 ARN。請參閱 建立專案 (API) 和 CreateProject。
- 應用程式上傳 ARN。請參閱 CreateUpload。
- 測試套件上傳 ARN。請參閱 CreateUpload。
- 裝置集區 ARN。請參閱 建立裝置集區 和 CreateDevicePool。

> **Note**
>
> 如果您是在自訂測試環境中執行測試，則也需要您的測試規格上傳 ARN。如需詳細資訊，請參閱步驟 5：（選用）上傳您的自訂測試規格及CreateUpload。

如需使用 Device Farm API 的詳細資訊，請參閱 自動化Device Farm。

後續步驟

在 Device Farm 主控台中，時鐘圖示會變更為結果圖示，例如在執行完成時成功。測試一完成，執行的報告就會出現。如需詳細資訊，請參閱AWS Device Farm 中的報告。

若要使用報告，請依照在 Device Farm 中檢視測試報告中的指示。

在 AWS Device Farm 中設定測試執行的執行逾時

您可以設定一值，表示在您停止每個裝置的執行測試前，測試執行應該執行多長時間。每個裝置的預設執行逾時為 150 分鐘，但您可以設定的最低值為 5 分鐘。您可以使用 AWS Device Farm 主控台 AWS CLI或 AWS Device Farm API 來設定執行逾時。

> **Important**
>
> 您應該將執行逾時選項設定為測試執行的「最大持續時間」，並設定一些緩衝。例如，如果您的測試需要每個裝置 20 分鐘，則您應該選擇每個裝置 30 分鐘逾時。

如果執行超過您的逾時，則該裝置上的執行會被迫停止。可能的話，提供部分結果。如果您是使用計量計費選項，則執行的計費最多算到該點。如需定價的詳細資訊，請參閱 Device Farm 定價。

如果您知道測試執行假設要在每個裝置上執行所需的時間，則可能想要使用此功能。如果您指定測試執行的執行逾時，則可以避免測試執行由於某些原因而停滯的情況，而且當沒有測試執行時，以裝置分鐘數計費。換言之，如果執行花費的時間超過預期，則可使用執行逾時功能來停止該執行。

您可以在兩個地方設定執行逾時，一個是專案層級，另一個是測試執行層級。

先決條件

1. 完成「設定」中的步驟。
2. 在 Device Farm 中建立專案。請遵循在 AWS Device Farm 中建立專案中的指示，然後返回此頁面。

設定專案的執行逾時

1. 登入 Device Farm 主控台，網址為 https：//https://console.aws.amazon.com/devicefarm。
2. 在 Device Farm 導覽面板上，選擇行動裝置測試，然後選擇專案。
3. 如果您已經有專案，請從清單中選擇。否則，請選擇新增專案，輸入專案的名稱，然後選擇提交。
4. 選擇 Project settings (專案設定)。
5. 在 General (一般) 標籤上，對於 Execution timeout (執行逾時)，輸入一值或使用捲軸。
6. 選擇儲存。

   您專案中的所有測試執行現在都會使用您指定的執行逾時值，除非您在排定執行時覆寫該逾時值。

設定測試執行的執行逾時

1. 登入 Device Farm 主控台，網址為 https：//https://console.aws.amazon.com/devicefarm。
2. 在 Device Farm 導覽面板上，選擇行動裝置測試，然後選擇專案。
3. 如果您已經有專案，請從清單中選擇。否則，請選擇新增專案，輸入專案的名稱，然後選擇提交。
4. 選擇 Create a new run (建立新執行)。
5. 依照步驟來選擇應用程式、設定您的測試、選取您的裝置，然後指定裝置狀態。
6. 在檢閱和開始執行時，針對設定執行逾時，輸入值或使用滑桿。
7. 選擇 Confirm and start run (確認並開始執行)。

模擬 AWS Device Farm 執行的網路連線和條件

您可以使用網路塑造來模擬網路連線和條件，同時在 Device Farm 中測試 Android、iOS 和 Web 應用程式。例如，您可以模擬失真或間歇性網際網路連線。

當您使用預設網路設定來建立執行時，每個裝置都有完整、不受限的 Wi-Fi 連線，與網際網路進行連線。當您使用網路打造時，您可以變更 Wi-Fi 連線來指定網路設定檔 (像是 3G 或 Lossy WiFi (失真 WiFi))，其會同時控制傳入和傳出流量的傳輸量、延遲、抖動和損失。

主題

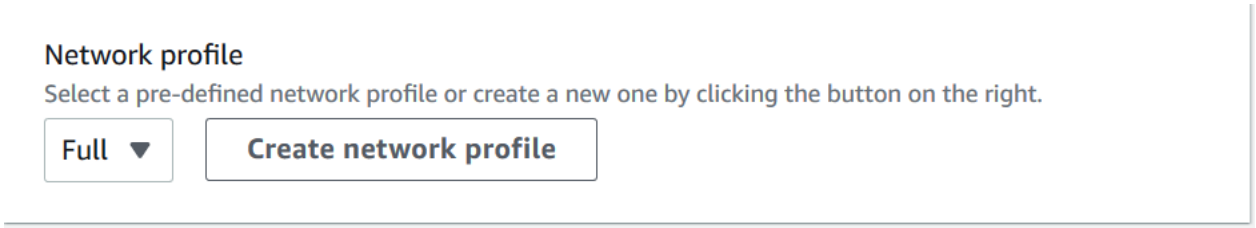
- 在排程測試執行時設定網路形狀
- 建立網路設定檔
- 在測試期間變更網路條件

在排程測試執行時設定網路形狀

排程執行時，您可以從任何 Device Farm 策劃的設定檔中選擇，也可以建立和管理自己的設定檔。

1. 從任何 Device Farm 專案中，選擇建立新的執行。

   如果您尚未有專案，請參閱在 AWS Device Farm 中建立專案。
2. 選擇您的應用程式，然後選擇下一步。
3. 設定您的測試，然後選擇下一步。
4. 選取您的裝置，然後選擇下一步。
5. 在位置和網路設定區段中，選擇網路設定檔或選擇建立網路設定檔以建立您自己的設定檔。

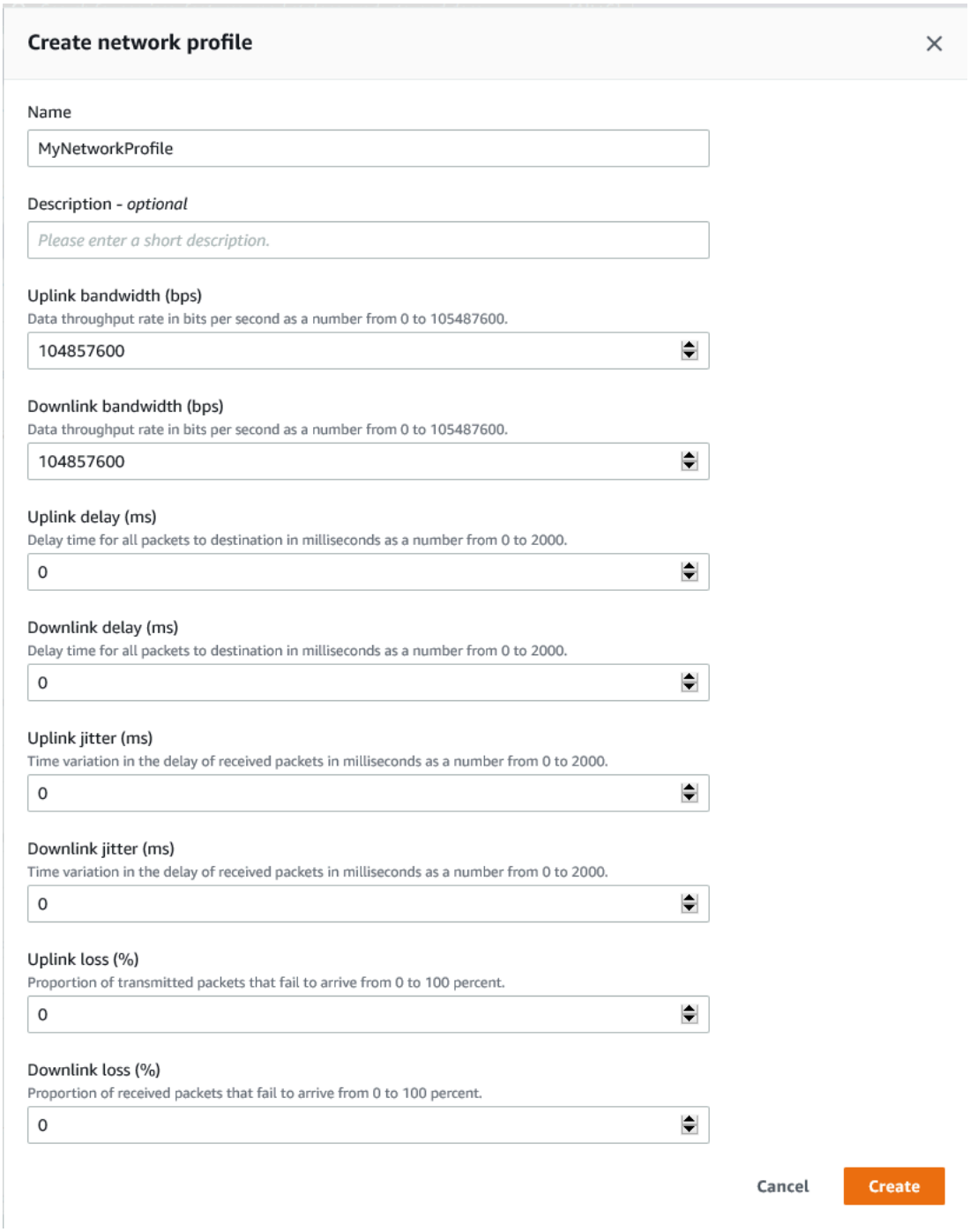


6. 選擇下一步。
7. 檢閱並開始您的執行測試。

建立網路設定檔

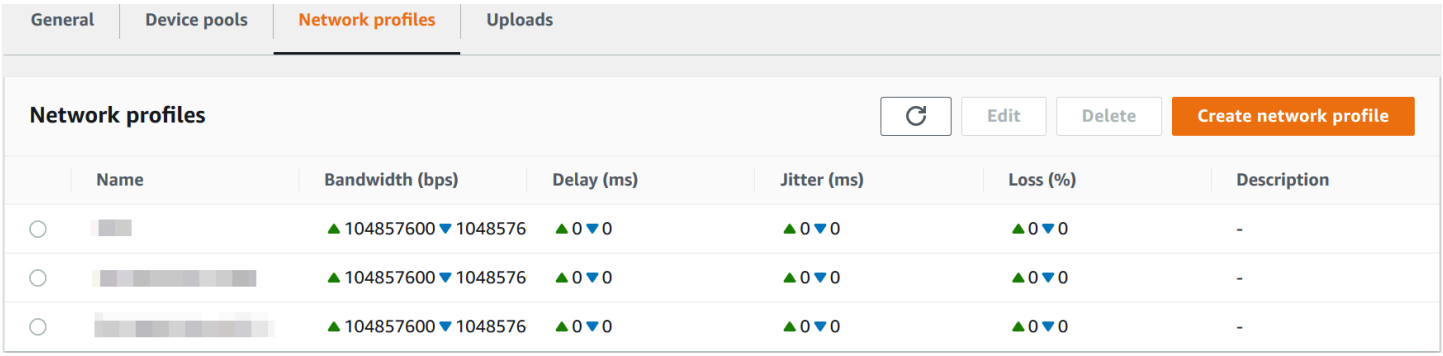
建立測試執行時，您可以建立網路設定檔。

1. 選擇建立網路設定檔。



2. 輸入網路設定檔的名稱和設定。
3. 選擇建立。
4. 完成建立您的測試執行並開始執行。

在您建立了網路設定檔之後，您將能夠在 Project settings(專案設定) 頁面上查看和管理它。



在測試期間變更網路條件

您可以使用 Appium 等架構從裝置主機呼叫 API，以模擬動態網路條件，例如在測試執行期間減少頻寬。如需詳細資訊，請參閱 CreateNetworkProfile。

在 AWS Device Farm 中停止執行

在啟動了執行之後，您可能想要停止它。例如，如果您在測試執行時注意到問題，則可能想要使用更新的測試指令碼，來重新啟動執行。

您可以使用 Device Farm 主控台 AWS CLI或 API 來停止執行。

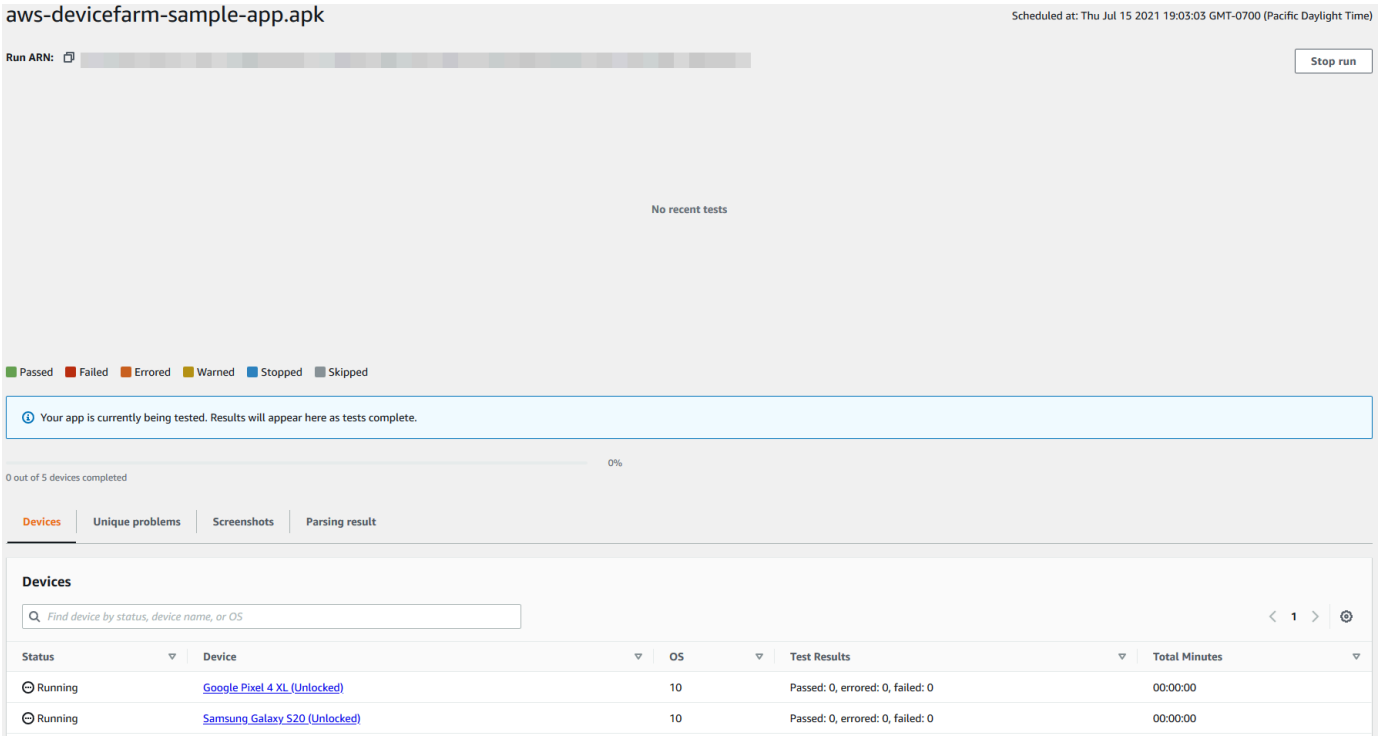
主題

- 停止執行 （ 主控台 ）
- 停止執行 (AWS CLI)
- 停止執行 (API)

停止執行 （ 主控台 ）

1. 登入 Device Farm 主控台，網址為 https：//https://console.aws.amazon.com/devicefarm。
2. 在 Device Farm 導覽面板上，選擇行動裝置測試，然後選擇專案。
3. 選擇您有作用中測試執行的專案。
4. 在自動化測試頁面上，選擇測試執行。

   待定或執行中的圖示應該會出現在裝置名稱的左側。

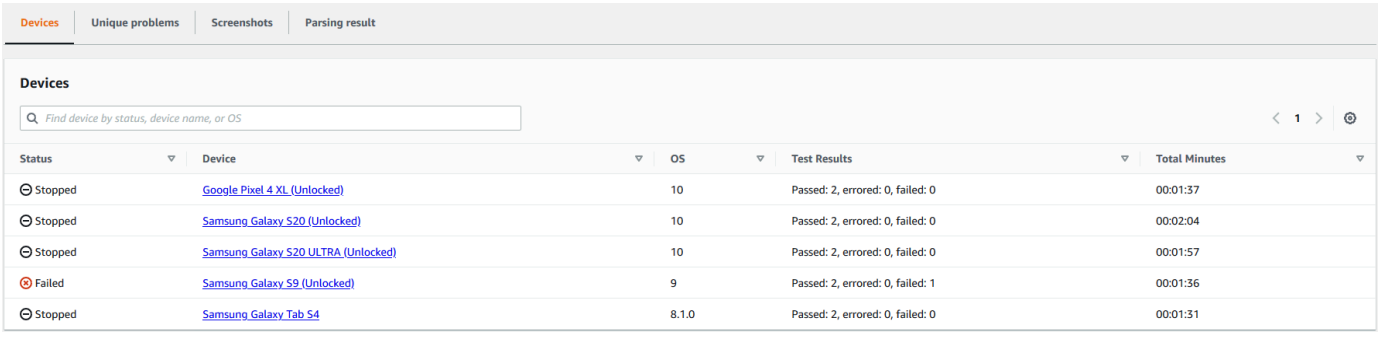


5. 選擇 Stop run (停止執行)。

一小段時間後，裝置名稱旁會出現一個帶有減號的紅色圓圈圖示。當執行停止時，圖示顏色會從紅色變更為黑色。

> **Important**
>
> 如果測試已執行，則 Device Farm 無法停止它。如果測試進行中，Device Farm 會停止測試。將計費的分鐘總數會出現在 Devices (裝置) 區段中。此外，也會向您收取 Device Farm 執行設定套件和遞減套件所需的總分鐘數。如需詳細資訊，請參閱 Device Farm 定價。

下圖顯示在測試執行成功停止之後的範例 Devices (裝置) 區段。



停止執行 (AWS CLI)

您可以執行以下命令來停止指定的測試執行，其中 *myARN* 是執行測試的 Amazon Resource Name (ARN)。

```
$ aws devicefarm stop-run --arn myARN
```

您應該會看到類似下列的輸出：

```
{
    "run": {
        "status": "STOPPING",
        "name": "Name of your run",
        "created": 1458329687.951,
        "totalJobs": 7,
        "completedJobs": 5,
        "deviceMinutes": {
            "unmetered": 0.0,
            "total": 0.0,
            "metered": 0.0
        },
        "platform": "ANDROID_APP",
        "result": "PENDING",
        "billingMethod": "METERED",
        "type": "BUILTIN_EXPLORER",
        "arn": "myARN",
        "counters": {
            "skipped": 0,
            "warned": 0,
            "failed": 0,
            "stopped": 0,
            "passed": 0,
            "errored": 0,
            "total": 0
        }
    }
}
```

若要取得執行的 ARN，請使用 `list-runs` 命令。輸出格式應類似以下內容：

```
{
    "runs": [
```

```
        {
            "status": "RUNNING",
            "name": "Name of your run",
            "created": 1458329687.951,
            "totalJobs": 7,
            "completedJobs": 5,
            "deviceMinutes": {
                "unmetered": 0.0,
                "total": 0.0,
                "metered": 0.0
            },
            "platform": "ANDROID_APP",
            "result": "PENDING",
            "billingMethod": "METERED",
            "type": "BUILTIN_EXPLORER",
            "arn": "Your ARN will be here",
            "counters": {
                "skipped": 0,
                "warned": 0,
                "failed": 0,
                "stopped": 0,
                "passed": 0,
                "errored": 0,
                "total": 0
            }
        }
    ]
}
```

如需搭配 使用 Device Farm 的詳細資訊 AWS CLI，請參閱 AWS CLI 參考。

停止執行 (API)

- 呼叫 StopRun 操作來進行測試執行。

如需使用 Device Farm API 的詳細資訊，請參閱 自動化Device Farm。

檢視 AWS Device Farm 中的執行清單

您可以使用 Device Farm 主控台 AWS CLI或 API 來檢視專案的執行清單。

主題

檢視執行清單 （ 主控台 ）

1. 登入 Device Farm 主控台，網址為 https：//https://console.aws.amazon.com/devicefarm。
2. 在 Device Farm 導覽面板上，選擇行動裝置測試，然後選擇專案。
3. 在專案清單中，選擇對應至您要檢視之清單的專案。

> **Tip**
>
> 您可以使用搜尋列，依名稱篩選專案清單。

檢視執行的清單 (AWS CLI)

- 執行 list-runs 命令。

  若要檢視單一執行的相關資訊，請執行 get-run 命令。

如需搭配 使用 Device Farm 的詳細資訊 AWS CLI，請參閱 AWS CLI 參考。

檢視執行清單 (API)

- 呼叫 `ListRuns` API。

  若要檢視單一執行的相關資訊，請呼叫 `GetRun` API。

如需 Device Farm API 的相關資訊，請參閱 自動化Device Farm。

在 AWS Device Farm 中建立裝置集區

您可以使用 Device Farm 主控台 AWS CLI或 API 來建立裝置集區。

主題

先決條件

- 在 Device Farm 主控台中建立執行。請遵循中的說明進行在 Device Farm 中建立測試執行 在您到達 Select devices (選擇裝置) 頁面時，請繼續遵循本節中的指示。

建立裝置集區 （主控台）

1. 在專案頁面上，選擇您的專案。在專案詳細資訊頁面中，選擇專案設定。在裝置集區索引標籤中，選擇建立裝置集區。
2. 針對 Name (名稱)，輸入可輕鬆識別此裝置集區的名稱。
3. 針對 Description (描述)，輸入可輕鬆識別此裝置集區的描述。
4. 如果您想要在此裝置集區中的裝置上使用一或多個選取條件，請執行下列動作：

   a. 選擇建立動態裝置集區。

   b. 選擇新增規則。

   c. 針對欄位 （第一個下拉式清單），選擇下列其中一項：

      - 若要依裝置製造商名稱包含裝置，請選擇裝置製造商。
      - 若要依裝置形式因素 （平板電腦或手機） 包含裝置，請選擇形式因素。
      - 若要根據負載依裝置的可用性狀態包含裝置，請選擇可用性。
      - 若要僅包含公有或私有裝置，請選擇機群類型。
      - 若要依其作業系統包含裝置，請選擇平台。
      - 有些裝置具有關於裝置的額外標籤標籤或描述。您可以選擇執行個體標籤，根據裝置標籤內容來尋找裝置。
      - 若要依其作業系統版本包含裝置，請選擇作業系統版本。
      - 若要依模型包含裝置，請選擇模型。

   d. 針對運算子 （第二個下拉式清單），選擇邏輯操作 (EQUALS、CONTAINS 等），以根據查詢包含裝置。例如，您可以選擇*## EQUALS ##*，以包含目前狀態為 的裝置Available。

e. 針對值（第三個下拉式清單），輸入或選擇您要為欄位和運算子值指定的值。值會根據您的欄位選擇而受限。例如，如果您為 欄位選擇平台，則唯一可用的選項是 ANDROID 和 IOS。同樣地，如果您選擇欄位的表單係數，則唯一可用的選項是 PHONE 和 TABLET。

f. 若要新增另一個規則，請選擇新增規則。

在您建立第一個規則之後，系統會選取裝置清單中每個符合規則裝置旁的方塊。在您建立或變更規則之後，系統會選取裝置清單中每個符合合併規則裝置旁的方塊。具有選取方塊的裝置會包含在裝置集區中。具有空白方塊的裝置則會受到排除。

g. 在最大裝置下，輸入您要在裝置集區中使用的裝置數量。如果您未輸入裝置數量上限 (Device Farm)，則 Device Farm 會挑選機群中符合您建立之規則的所有裝置。為了避免產生額外費用，請將此數字設定為符合您實際平行執行和裝置多樣性需求的數量。

h. 若要刪除規則，請選擇移除規則。

5. 如果您想要手動包含或排除個別裝置，請執行下列動作：

a. 選擇建立靜態裝置集區。

b. 選取或清除每個裝置旁的方塊。只有在您未指定任何規則的情況下，才能選取或清除方塊。

6. 如果您想要包含或排除所有顯示的裝置，請選取或清除清單標頭列中的方塊。如果您只想檢視私有裝置執行個體，請選擇僅查看私有裝置執行個體。

> **Important**
>
> 雖然您可以使欄標頭列中的方塊來變更顯示裝置的清單，但這不表示只會包含或排除顯示中剩下的裝置。若要確認包含或排除了哪些裝置，請務必清除所有欄標頭列方塊的內容，然後瀏覽方塊。

7. 選擇建立。

建立裝置集區 (AWS CLI)

> **Tip**
>
> 如果您未輸入裝置數量上限 (Device Farm)，則 Device Farm 會挑選機群中符合您建立之規則的所有裝置。為了避免產生額外費用，請將此數字設定為符合您實際平行執行和裝置多樣性需求的數量。

- 執行 create-device-pool 命令。

如需搭配 使用 Device Farm 的詳細資訊 AWS CLI，請參閱 AWS CLI 參考。

建立裝置集區 (API)

> **Tip**
>
> 如果您未輸入裝置數量上限 (Device Farm)，則 Device Farm 會挑選機群中符合您建立之規則的所有裝置。為了避免產生額外費用，請將此數字設定為符合您實際平行執行和裝置多樣性需求的數量。

- 呼叫 `CreateDevicePool` API。

如需使用 Device Farm API 的詳細資訊，請參閱 自動化Device Farm。

在 AWS Device Farm 中分析測試結果

在標準測試環境中，您可以使用 Device Farm 主控台來檢視測試執行中每個測試的報告。清空報告可協助您了解哪些測試通過或失敗，並為您提供不同裝置組態中應用程式效能和行為的詳細資訊。

Device Farm 也會收集可在測試執行完成時下載的其他成品，例如檔案、日誌和映像。此資訊可協助您分析應用程式在真實裝置上的行為、識別問題或錯誤，以及診斷問題。

主題

- 在 Device Farm 中檢視測試報告
- 在 Device Farm 中下載成品

在 Device Farm 中檢視測試報告

使用 Device Farm 主控台檢視您的測試報告。如需詳細資訊，請參閱AWS Device Farm 中的報告。

主題

- 先決條件
- 檢視報告
- Device Farm 測試結果狀態

先決條件

設定測試執行並驗證其是否已完成。

1. 若要建立執行，請參閱在 Device Farm 中建立測試執行，然後返回此頁面。
2. 確認執行已完成。在測試執行期間，Device Farm 主控台會顯示進行中 執行的待處理圖示。執行中的每個裝置也會以待定圖示開頭，然後在測試開始時切換到執行中 圖示。當每個測試完成時，裝置名稱旁會顯示測試結果圖示。完成所有測試後，執行旁的待定圖示會變更為測試結果圖示。如需詳細資訊，請參閱Device Farm 測試結果狀態。

檢視報告

您可以在 Device Farm 主控台中檢視測試結果。

主題

- 檢視測試執行摘要頁面
- 檢視唯一的問題報告
- 檢視裝置報告
- 檢視測試套件報告
- 檢視測試報告
- 檢視報告中問題、裝置、套件或測試的日誌資訊

檢視測試執行摘要頁面

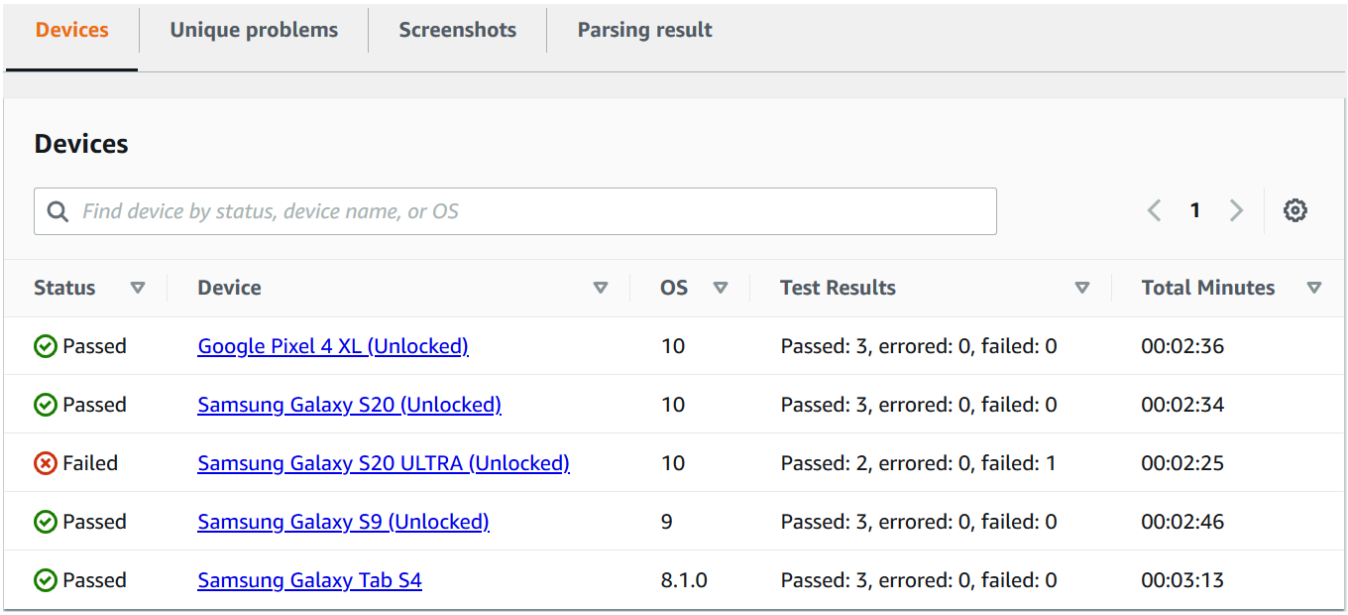
1. 登入 Device Farm 主控台，網址為 https：//https://console.aws.amazon.com/devicefarm。
2. 在導覽窗格中，選擇行動裝置測試，然後選擇專案。
3. 在專案清單中，選擇用於執行的專案。

> **Tip**
>
> 若要依名稱篩選專案清單，請使用搜尋列。

4. 選擇已完成的執行來檢視其摘要報告頁面。

5. 測試執行摘要頁面會顯示測試結果的概觀。

   - Unique problems (唯一問題) 區段會列出唯一警告和失敗。若要檢視唯一問題，請按照檢視唯一的問題報告中的指示。
   - Devices (裝置) 區段會顯示每個裝置的測試總數，依結果排列。



   在此範例中，有數個裝置。在第一個資料表項目中，執行 Android 版本 10 的 Google Pixel 4 XL 裝置報告了三個成功的測試，需要 02：36 分鐘才能執行。

   若要依裝置檢視結果，請按照檢視裝置報告中的指示。

   - 螢幕擷取畫面區段會顯示 Device Farm 在執行期間擷取的任何螢幕擷取畫面清單，依裝置分組。
   - 在剖析結果區段中，您可以下載剖析結果。

檢視唯一的問題報告

1. 在 Unique problems (唯一問題) 中，選擇您想要檢視的問題。
2. 選擇裝置。報告會顯示問題的相關資訊。

   Video (視訊) 區段顯示測試的可下載影片錄製。

   結果區段會顯示測試結果。狀態會以結果圖示表示。如需詳細資訊，請參閱個別測試的狀態。

日誌區段會顯示 Device Farm 在測試期間記錄的任何資訊。若要檢視此資訊，請按照檢視報告中問題、裝置、套件或測試的日誌資訊中的指示。

檔案索引標籤會顯示您可以下載的任何測試關聯檔案（例如日誌檔案）的清單。若要下載檔案，請在清單中選擇檔案的連結。

螢幕擷取畫面索引標籤會顯示 Device Farm 在測試期間擷取的任何螢幕擷取畫面清單。

檢視裝置報告

- 在 Devices (裝置) 區段中，選擇裝置。

  Video (視訊) 區段顯示測試的可下載影片錄製。

  套件區段會顯示包含裝置套件相關資訊的資料表。

  在此資料表中，測試結果欄會針對已在裝置上執行的每個測試套件，依結果摘要測試的數量。此資料也有圖形元件。如需詳細資訊，請參閱多個測試的狀態。

  若要依套件檢視完整結果，請遵循 中的指示檢視測試套件報告。

  日誌區段會顯示 Device Farm 在執行期間為裝置記錄的任何資訊。若要檢視此資訊，請按照檢視報告中問題、裝置、套件或測試的日誌資訊中的指示。

  檔案區段會顯示裝置套件的清單，以及您可以下載的任何相關檔案（例如日誌檔案）。若要下載檔案，請在清單中選擇檔案的連結。

  螢幕擷取畫面區段會顯示 Device Farm 在裝置執行期間擷取的任何螢幕擷取畫面清單，依套件分組。

檢視測試套件報告

1. 在 Devices (裝置) 區段中，選擇裝置。
2. 在套件區段中，從資料表中選擇套件。

   Video (視訊) 區段顯示測試的可下載影片錄製。

   測試區段會顯示包含套件中測試相關資訊的資料表。

在表格中，測試結果欄會顯示結果。此資料也有圖形元件。如需詳細資訊，請參閱多個測試的狀態。

若要依測試檢視完整結果，請遵循 中的指示檢視測試報告。

日誌區段會顯示 Device Farm 在套件執行期間記錄的任何資訊。若要檢視此資訊，請按照檢視報告中問題、裝置、套件或測試的日誌資訊中的指示。

檔案區段會顯示套件的測試清單，以及您可以下載的任何相關檔案（例如日誌檔案）。若要下載檔案，請在清單中選擇檔案的連結。

螢幕擷取畫面區段會顯示 Device Farm 在套件執行期間擷取的任何螢幕擷取畫面清單，依測試分組。

檢視測試報告

1. 在 Devices (裝置) 區段中，選擇裝置。
2. 在 Suites (套件) 區段中，選擇套件。
3. 在測試區段中，選擇測試。
4. Video (視訊) 區段顯示測試的可下載影片錄製。

結果區段會顯示測試結果。狀態會以結果圖示表示。如需詳細資訊，請參閱個別測試的狀態。

日誌區段會顯示 Device Farm 在測試期間記錄的任何資訊。若要檢視此資訊，請按照檢視報告中問題、裝置、套件或測試的日誌資訊中的指示。

檔案索引標籤會顯示您可以下載的任何測試關聯檔案（例如日誌檔案）的清單。若要下載檔案，請在清單中選擇檔案的連結。

螢幕擷取畫面索引標籤會顯示 Device Farm 在測試期間擷取的任何螢幕擷取畫面清單。

檢視報告中問題、裝置、套件或測試的日誌資訊

日誌區段會顯示下列資訊：

- Source (來源) 代表日誌項目的來源。可能的值包括：
  - Harness 代表 Device Farm 建立的日誌項目。這些日誌項目通常是在啟動和停止事件期間建立的。

  - 裝置代表裝置建立的日誌項目。若為 Android，這些是與 logcat 相容的日誌項目。若為 iOS，這些是與 syslog 相容的日誌項目。
  - Test (測試) 代表測試或其測試架構所建立的日誌項目。
- Time (時間) 代表第一個日誌項目與此日誌項目之間的經歷時間。此時間是以 *MM:SS.SSS* 格式表示，其中 *M* 代表分鐘，而 *S* 代表秒。
- PID 代表已建立日誌項目的程序識別碼 (PID)。裝置上由應用程式建立的所有日誌項目都具有相同的 PID。
- Level (層級) 代表日誌項目的記錄層級。例如， 會`Logger.debug("This is a message!")`記錄 層級`Debug`。以下是可能值：
  - 警示
  - 嚴重
  - 偵錯
  - Emergency (緊急)
  - 錯誤
  - Errored (錯誤)
  - 失敗
  - 資訊
  - 內部 (Internal)
  - Notice (注意)
  - Passed (通過)
  - 略過
  - 已停止
  - 詳細資訊
  - Warned (警告)
  - 警告
- Tag (標籤) 代表日誌項目的任意中繼資料。例如，Android logcat 可以使用此項，描述系統哪個部分已建立日誌項目 (例如，`ActivityManager`)。
- Message (訊息) 代表日誌項目的訊息或資料。例如， 會`Logger.debug("Hello, World!")`記錄訊息`"Hello, World!"`。

若要只顯示局部資訊：

- 若要顯示符合特定資料欄值的所有日誌項目，請在搜尋列中輸入該值。例如，若要顯示來源值為 的所有日誌項目`Harness`，請在搜尋列**`Harness`**中輸入 。
- 若要從資料欄標頭方塊中移除所有字元，請選擇資料欄標頭方塊中的 X。從資料欄標頭方塊中移除所有字元與*在該資料欄標頭方塊中輸入相同。

若要下載裝置的所有日誌資訊，包括您執行的所有套件和測試，請選擇下載日誌。

Device Farm 測試結果狀態

Device Farm 主控台會顯示圖示，協助您快速評估已完成測試執行的狀態。如需 Device Farm 中測試的詳細資訊，請參閱 AWS Device Farm 中的報告。

主題

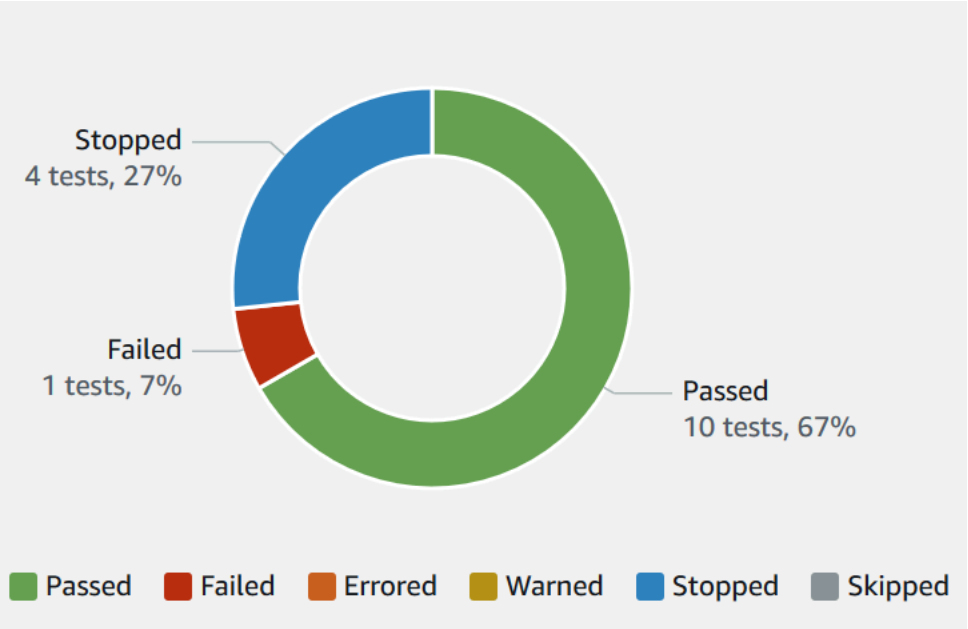
個別測試的狀態

對於描述個別測試的報告，Device Farm 會顯示代表測試結果狀態的圖示：

Description	圖示
測試成功。	
測試失敗。	
Device Farm 已略過測試。	
測試已停止。	
Device Farm 傳回警告。	
Device Farm 傳回錯誤。	

多個測試的狀態

如果您選擇完成執行，Device Farm 會顯示摘要圖表，顯示各種狀態的測試百分比。



例如，此測試執行結果圖表顯示執行有 4 個停止的測試、1 個失敗的測試，以及 10 個成功的測試。

圖形一律以顏色編碼和標記。

在 Device Farm 中下載成品

Device Farm 會收集執行中每個測試的成品，例如報告、日誌檔案和映像。

您可以下載測試執行期間所建立的成品：

檔案

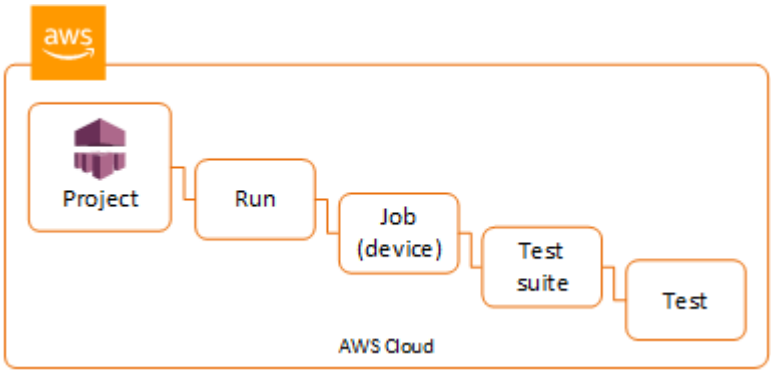
測試執行期間產生的檔案，包括 Device Farm 報告。如需詳細資訊，請參閱在 Device Farm 中檢視測試報告。

日誌

測試執行中每次測試的輸出。

螢幕擷取畫面

系統會記錄測試執行中每次測試的螢幕影像。



下載成品（主控台）

1. 在測試執行報告頁面的 Devices (裝置) 中，選擇行動裝置。
2. 若要下載檔案，請在 Files (檔案) 中選擇。
3. 要下載您測試執行的日誌，請在 Logs (日誌) 中選擇 Download logs (下載日誌)。
4. 若要下載螢幕擷取畫面，請在 Screenshots (螢幕擷取畫面) 中選擇螢幕擷取畫面。

如需在自訂測試環境中下載成品的詳細資訊，請參閱 在自訂測試環境中下載成品。

下載成品 (AWS CLI)

您可以使用 AWS CLI 列出您的測試執行成品。

主題

- 步驟 1：取得您的 Amazon Resource Name (ARN)
- 步驟 2：列出您的成品
- 步驟 3：下載您的成品

步驟 1：取得您的 Amazon Resource Name (ARN)

您可以透過執行、工作、測試套件或測試，列出您的成品。您需要對應的 ARN。此資料表顯示每個 AWS CLI 清單命令的輸入 ARN：

AWS CLI List 命令	需要 ARN
list-projects	此命令會傳回所有專案，而且不需要 ARN。

AWS CLI List 命令	需要 ARN
list-runs	`project`
list-jobs	`run`
list-suites	`job`
list-tests	`suite`

例如，若要尋找測試 ARN，請在輸入參數使用您的測試套件 ARN 來執行 list-tests。

範例：

```
aws devicefarm list-tests --arn arn:MyTestSuiteARN
```

測試套件中每次測試的回應中皆會包含測試 ARN。

```
{
    "tests": [
        {
            "status": "COMPLETED",
            "name": "Tests.FixturesTest.testExample",
            "created": 1537563725.116,
            "deviceMinutes": {
                "unmetered": 0.0,
                "total": 1.89,
                "metered": 1.89
            },
            "result": "PASSED",
            "message": "testExample passed",
            "arn": "arn:aws:devicefarm:us-west-2:123456789101:test:5e01a8c7-c861-4c0a-
b1d5-12345EXAMPLE",
            "counters": {
                "skipped": 0,
                "warned": 0,
                "failed": 0,
                "stopped": 0,
                "passed": 1,
                "errored": 0,
                "total": 1
```

```
            }
        }
    ]
}
```

步驟 2：列出您的成品

The AWS CLI list-artifacts 命令會傳回成品清單，例如檔案、螢幕擷取畫面和日誌。每個成品都會有 URL，因此您可以下載檔案。

- 呼叫 list-artifacts 指定執行、工作、測試套件或測試 ARN。指定檔案、日誌或螢幕快照的類型。

  此範例會傳回個別測試中，每個可供下載成品的 URL：

  ```
  aws devicefarm list-artifacts --arn arn:MyTestARN --type "FILE"
  ```

  每個成品的回應皆包含下載 URL。

  ```
  {
      "artifacts": [
          {
              "url": "https://prod-us-west-2-uploads.s3-us-west-2.amazonaws.com/
  ExampleURL",
              "extension": "txt",
              "type": "APPIUM_JAVA_OUTPUT",
              "name": "Appium Java Output",
              "arn": "arn:aws:devicefarm:us-west-2:123456789101:artifact:5e01a8c7-
  c861-4c0a-b1d5-12345EXAMPLE",
          }
      ]
  }
  ```

步驟 3：下載您的成品

- 使用先前步驟的 URL 下載您的成品。此範例使用 curl 來下載 Android Appium Java 輸出檔：

  ```
  curl "https://prod-us-west-2-uploads.s3-us-west-2.amazonaws.com/ExampleURL"
   > MyArtifactName.txt
  ```

下載成品 (API)

Device Farm API ListArtifacts 方法會傳回成品清單，例如檔案、螢幕擷取畫面和日誌。每個成品都會有 URL，因此您可以下載檔案。

在自訂測試環境中下載成品

在自訂測試環境中，Device Farm 會收集成品，例如自訂報告、日誌檔案和映像。這些成品可供測試執行中的每個裝置使用。

您可以下載這些在測試執行期間所建立的成品：

測試規格輸出

在測試規格 YAML 檔案中執行命令的輸出。

客戶成品

包含測試執行成品的壓縮檔案。這可在測試規格 YAML 檔案中的 artifacts: (成品：) 區段設定。

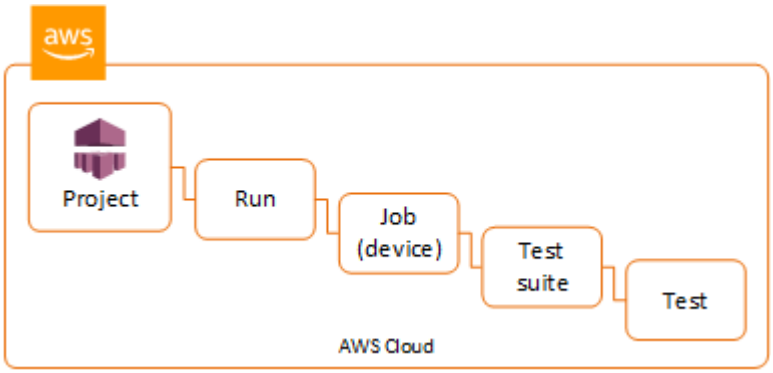
測試規格 shell 指令碼

從 YAML 檔案建立的中繼 shell 指令碼檔案。因為 shell 指令碼檔案會用於測試執行，所以其也可用於偵錯 YAML 檔案。

測試規格檔案

用於執行測試的 YAML 檔案。

如需詳細資訊，請參閱在 Device Farm 中下載成品。



標記 AWS Device Farm 資源

AWS Device Farm 使用 AWS 資源群組標記 API。此 API 可讓您使用標籤來管理 AWS 帳戶中的資源。您可以將標籤新增至資源，例如專案和測試執行。

您可以使用標籤執行以下動作：

- 整理 AWS 帳單，以反映成本結構。方式是註冊以取得包含標籤鍵值的 AWS 帳戶帳單。接著，若要查看合併資源的成本，請根據具有相同標籤鍵值的資源來整理您的帳單資訊。例如，您可以使用應用程式名稱來標記數個資源，然後整理帳單資訊以查看該應用程式跨多項服務的總成本。如需詳細資訊，請參閱《關於 AWS Billing and Cost Management》中的成本分配和標記。
- 透過 IAM 政策控制存取。若要執行此作業，請建立允許使用標籤值條件來存取資源或資源集的政策。
- 識別並管理具有特定屬性做為標籤的執行，例如用於測試的分支。

如需標記資源的詳細資訊，請參閱標記最佳實務白皮書。

主題

標記資源

AWS 資源群組標記 API 可讓您在資源上新增、移除或修改標籤。如需詳細資訊，請參閱 AWS 資源群組標記 API 參考。

若要標記資源，請使用`resourcegroupstaggingapi`端點中的 `TagResources`操作。此操作會取得支援服務的 ARN 清單以及金鑰/值對的清單。此值是選用的。空字串表示該標籤不應有任何值。例如：以下 Python 範例使用值為 `release` 的標籤 `build-config` 來標記一連串的專案 ARN：

```
import boto3

client = boto3.client('resourcegroupstaggingapi')

client.tag_resources(ResourceARNList=["arn:aws:devicefarm:us-
west-2:111122223333:project:123e4567-e89b-12d3-a456-426655440000",
```

```
                                        "arn:aws:devicefarm:us-
west-2:111122223333:project:123e4567-e89b-12d3-a456-426655441111",
                                        "arn:aws:devicefarm:us-
west-2:111122223333:project:123e4567-e89b-12d3-a456-426655442222"]
                      Tags={"build-config":"release", "git-commit":"8fe28cb"})
```

標籤值為非必要。若要設定沒有值的標籤，請在指定值時使用空字串 ("")。一個標籤只能有一個值。資源的標籤先前具有的任何值都將被新值覆寫。

依標籤查詢資源

若要依標籤查閱資源，請從 `resourcegrouptaggingapi` 端點使用 `GetResources` 操作。此操作採用一連串非必要的篩選條件，傳回符合指定條件的資源。如果沒有篩選條件，則會傳回所有標記的資源。`GetResources` 操作允許您根據下列條件篩選資源

- 標籤值
- 資源類型 (例如 `devicefarm:run`)

如需詳細資訊，請參閱 AWS 資源群組標記 API 參考。

下列範例會使用值`stack`為 的標籤查詢 Device Farm 桌面瀏覽器測試工作階段 (`devicefarm:testgrid-session` 資源) `production`：

```
import boto3
client = boto3.client('resourcegroupstaggingapi')
sessions = client.get_resources(ResourceTypeFilters=['devicefarm:testgrid-session'],
                                TagFilters=[
                                  {"Key":"stack","Values":["production"]}
                                ])
```

移除資源的標籤

若要移除標籤，請使用 `UntagResources` 操作，指定資源清單以及要移除的標籤：

```
import boto3
client = boto3.client('resourcegroupstaggingapi')
client.UntagResources(ResourceARNList=["arn:aws:devicefarm:us-
west-2:111122223333:project:123e4567-e89b-12d3-a456-426655440000"], TagKeys=["RunCI"])
```

AWS Device Farm 中的測試架構和內建測試

本節說明 Device Farm 對測試架構和內建測試類型的支援。

Device Farm 會讓您將應用程式和測試上傳到由 服務管理的安全 Amazon S3 儲存貯體，以執行自動化測試。上傳後，它會啟動基礎基礎設施，包括服務受管測試主機，並在多個裝置上平行執行測試。測試結果會儲存在服務受管 S3 儲存貯體中。此架構稱為服務端執行，是在靠近裝置的主機上執行測試的快速且有效率方式，而不需要自行管理測試主機基礎設施。此方法非常適合在許多裝置上獨立進行測試，以及根據 CI/CD 管道的內容進行測試。

如需 Device Farm 如何執行測試的詳細資訊，請參閱 在 AWS Device Farm 中測試環境。

> **Note**
>
> 對於 Appium 測試人員，您可能偏好從本機環境執行 Appium 測試。透過遠端存取工作階段，您可以執行用戶端 Appium 測試。如需詳細資訊，請參閱用戶端 Appium 測試。

測試架構

Device Farm 支援這些行動自動化測試架構：

Android 應用程式測試架構

- 自動 Appium 測試
- 檢測

iOS 應用程式測試架構

- 自動 Appium 測試
- XCTest
- XCTest UI

Web 應用程式測試架構

Web 應用程式支援使用 Appium。如需將測試帶入 Appium 的詳細資訊，請參閱 在 Device Farm 中自動執行 Appium 測試。

自訂測試環境中的架構

Device Farm 不支援自訂 XCTest 架構的測試環境。如需詳細資訊，請參閱AWS Device Farm 中的自訂測試環境。

Appium 版本支援

對於在自訂環境中執行的測試，Device Farm 支援 Appium 第 1 版。如需詳細資訊，請參閱在 AWS Device Farm 中測試環境。

內建測試類型

透過內建測試，您可以在多個裝置上測試應用程式，而無需撰寫和維護測試自動化指令碼。Device Farm 提供一種內建測試類型：

- 內建：模糊 (Android 和 iOS)

在 Device Farm 中自動執行 Appium 測試

> **Note**
>
> 此頁面涵蓋在 Device Farm 的受管伺服器端執行環境中執行 Appium 測試。若要在遠端存取工作階段期間從本機用戶端環境執行 Appium 測試，請參閱用戶端 Appium 測試。

本節說明如何設定、封裝和上傳 Appium 測試，以便在 Device Farm 的受管伺服器端環境中執行。Appium 是一種開放原始碼工具，用於自動化原生和行動 Web 應用程式。如需詳細資訊，請參閱 Appium 網站上的 Appium 簡介。

如需範例應用程式和工作測試的連結，請參閱 GitHub 上的適用於 Android 的 Device Farm Sample App 和適用於 iOS 的 Device Farm Sample App。

如需在 Device Farm 中測試及伺服器端運作方式的詳細資訊，請參閱 AWS Device Farm 中的測試架構和內建測試。

選取 Appium 版本

> **Note**
>
> 支援特定的 Appium 版本、Appium 驅動程式或程式設計 SDKs 將取決於為測試執行選取的裝置和測試主機。

Device Farm 測試主機已預先安裝 Appium，以便針對更直接的使用案例更快速地設定測試。不過，使用測試規格檔案可讓您視需要安裝不同版本的 Appium。

案例 1：預先設定的 Appium 版本

Device Farm 根據測試主機預先設定不同的 Appium 伺服器版本。主機隨附工具，可啟用具有裝置平台預設驅動程式的預先設定版本（適用於 Android 的 UiAutomator2 和適用於 iOS 的 XCUITest)。

```
phases:
  install:
    commands:
      - export APPIUM_VERSION=2
      - devicefarm-cli use appium $APPIUM_VERSION
```

若要檢視支援的軟體清單，請參閱 上的 主題自訂測試環境中支援的軟體。

案例 2：自訂 Appium 版本

若要選取自訂版本的 Appium，請使用 npm命令來安裝它。下列範例示範如何安裝最新版的 Appium 2。

```
phases:
  install:
    commands:
      - export APPIUM_VERSION=2
      - npm install -g appium@$APPIUM_VERSION
```

案例 3：舊版 iOS 主機上的 Appium

在 上舊版 iOS 測試主機，您可以選擇具有 的特定 Appium 版本avm。例如，若要使用 avm命令將 Appium 伺服器版本設定為 2.1.2，請將這些命令新增至您的測試規格 YAML 檔案。

```
phases:
  install:
    commands:
      - export APPIUM_VERSION=2.1.2
      - avm $APPIUM_VERSION
```

為 iOS 測試選取 WebDriverAgent 版本

若要在 iOS 裝置上執行 Appium 測試，必須使用 WebDriverAgent。此應用程式必須經過簽署，才能安裝在 iOS 裝置上。Device Farm 提供預先簽章的 WebDriverAgent 版本，可在自訂測試環境執行期間使用。

下列程式碼片段可用於在與 XCTestUI 驅動程式版本相容的測試規格檔案中，選取 Device Farm 上的 WebDriverAgent 版本。

```
phases:
  pre_test:
    commands:
      - |-
        APPIUM_DRIVER_VERSION=$(appium driver list --installed --json | jq -r
 ".xcuitest.version" | cut -d "." -f 1);
        CORRESPONDING_APPIUM_WDA=$(env | grep
 "DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V${APPIUM_DRIVER_VERSION}")
        if [[ ! -z "$APPIUM_DRIVER_VERSION" ]] && [[ ! -z
 "$CORRESPONDING_APPIUM_WDA" ]]; then
          echo "Using Device Farm's prebuilt WDA version ${APPIUM_DRIVER_VERSION}.x,
 which corresponds with your driver";
          DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH=$(echo $CORRESPONDING_APPIUM_WDA |
 cut -d "=" -f2)
        else
          LATEST_SUPPORTED_WDA_VERSION=$(env | grep
 "DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V" | sort -V -r | head -n 1)
          echo "Unknown driver version $APPIUM_DRIVER_VERSION; falling back to the
 Device Farm default version of $LATEST_SUPPORTED_WDA_VERSION";
          DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH=$(echo $LATEST_SUPPORTED_WDA_VERSION
 | cut -d "=" -f2)
        fi;
```

如需 WebDriverAgent 的詳細資訊，請參閱 Appium 的文件。

將 Appium 測試與 Device Farm 整合

使用下列指示將 Appium 測試與 AWS Device Farm 整合。如需在 Device Farm 中使用 Appium 測試的詳細資訊，請參閱 在 Device Farm 中自動執行 Appium 測試。

設定您的 Appium 測試套件

使用以下指示來設定您的測試套件。

Java (JUnit)

1. 修改 pom.xml 以將封裝設定為 JAR 檔案：

```
<groupId>com.acme</groupId>
<artifactId>acme-myApp-appium</artifactId>
<version>1.0-SNAPSHOT</version>
<packaging>jar</packaging>
```

2. 修改 pom.xml 以使用 maven-jar-plugin 將測試建置到 JAR 檔案。

   下列外掛程式會將您的測試原始程式碼 (src/test目錄中的任何項目) 建置為 JAR 檔案：

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-jar-plugin</artifactId>
  <version>2.6</version>
  <executions>
    <execution>
      <goals>
        <goal>test-jar</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

3. 修改 maven-dependency-plugin 以pom.xml使用 建置相依性做為 JAR 檔案。

   下列外掛程式會將您的相依性複製到 dependency-jars目錄：

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-dependency-plugin</artifactId>
  <version>2.10</version>
```

```
  <executions>
    <execution>
      <id>copy-dependencies</id>
      <phase>package</phase>
      <goals>
        <goal>copy-dependencies</goal>
      </goals>
      <configuration>
        <outputDirectory>${project.build.directory}/dependency-jars/</
outputDirectory>
      </configuration>
    </execution>
  </executions>
</plugin>
```

4. 將下列 XML 組件儲存至 `src/main/assembly/zip.xml`。

   下列 XML 是組件定義，設定後會指示 Maven 建置 .zip 檔案，其中包含建置輸出目錄根目錄和`dependency-jars`目錄中的所有內容：

```
<assembly
    xmlns="http://maven.apache.org/plugins/maven-assembly-plugin/assembly/1.1.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/plugins/maven-assembly-plugin/
assembly/1.1.0 http://maven.apache.org/xsd/assembly-1.1.0.xsd">
  <id>zip</id>
  <formats>
    <format>zip</format>
  </formats>
  <includeBaseDirectory>false</includeBaseDirectory>
  <fileSets>
    <fileSet>
      <directory>${project.build.directory}</directory>
      <outputDirectory>./</outputDirectory>
      <includes>
        <include>*.jar</include>
      </includes>
    </fileSet>
    <fileSet>
      <directory>${project.build.directory}</directory>
      <outputDirectory>./</outputDirectory>
      <includes>
        <include>/dependency-jars/</include>
```

```
      </includes>
    </fileSet>
  </fileSets>
</assembly>
```

5. 修改 pom.xml 以使用 maven-assembly-plugin 將測試和所有相依性封裝至單一 .zip 檔案。

   下列外掛程式使用上述組件，在每次執行建置輸出目錄中建立名為 zip-with-dependencies的 mvn package .zip 檔案：

```
<plugin>
  <artifactId>maven-assembly-plugin</artifactId>
  <version>2.5.4</version>
  <executions>
    <execution>
      <phase>package</phase>
      <goals>
        <goal>single</goal>
      </goals>
      <configuration>
        <finalName>zip-with-dependencies</finalName>
        <appendAssemblyId>false</appendAssemblyId>
        <descriptors>
          <descriptor>src/main/assembly/zip.xml</descriptor>
        </descriptors>
      </configuration>
    </execution>
  </executions>
</plugin>
```

> **Note**
>
> 如果您收到錯誤，指明 1.3 中不支援註釋，請將以下內容新增至 pom.xml：
>
> ```
> <plugin>
>   <artifactId>maven-compiler-plugin</artifactId>
>   <configuration>
>     <source>1.7</source>
>     <target>1.7</target>
>   </configuration>
> ```

```
</plugin>
```

Java (TestNG)

1. 修改 pom.xml 以將封裝設定為 JAR 檔案：

```
<groupId>com.acme</groupId>
<artifactId>acme-myApp-appium</artifactId>
<version>1.0-SNAPSHOT</version>
<packaging>jar</packaging>
```

2. 修改 pom.xml 以使用 maven-jar-plugin 將測試建置到 JAR 檔案。

   下列外掛程式會將您的測試原始程式碼 (src/test目錄中的任何項目) 建置為 JAR 檔案：

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-jar-plugin</artifactId>
  <version>2.6</version>
  <executions>
    <execution>
      <goals>
        <goal>test-jar</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

3. 修改 maven-dependency-plugin 以pom.xml使用 建置相依性做為 JAR 檔案。

   下列外掛程式會將您的相依性複製到 dependency-jars目錄：

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-dependency-plugin</artifactId>
  <version>2.10</version>
  <executions>
    <execution>
      <id>copy-dependencies</id>
      <phase>package</phase>
      <goals>
```

```
        <goal>copy-dependencies</goal>
      </goals>
      <configuration>
        <outputDirectory>${project.build.directory}/dependency-jars/</
outputDirectory>
      </configuration>
    </execution>
  </executions>
</plugin>
```

4. 將下列 XML 組件儲存至 `src/main/assembly/zip.xml`。

   下列 XML 是組件定義，設定後會指示 Maven 建置 .zip 檔案，其中包含建置輸出目錄根目錄和`dependency-jars`目錄中的所有內容：

```
<assembly
    xmlns="http://maven.apache.org/plugins/maven-assembly-plugin/assembly/1.1.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/plugins/maven-assembly-plugin/
assembly/1.1.0 http://maven.apache.org/xsd/assembly-1.1.0.xsd">
  <id>zip</id>
  <formats>
    <format>zip</format>
  </formats>
  <includeBaseDirectory>false</includeBaseDirectory>
  <fileSets>
    <fileSet>
      <directory>${project.build.directory}</directory>
      <outputDirectory>./</outputDirectory>
      <includes>
        <include>*.jar</include>
      </includes>
    </fileSet>
    <fileSet>
      <directory>${project.build.directory}</directory>
      <outputDirectory>./</outputDirectory>
      <includes>
        <include>/dependency-jars/</include>
      </includes>
    </fileSet>
  </fileSets>
</assembly>
```

5. 修改 pom.xml 以使用 maven-assembly-plugin 將測試和所有相依性封裝至單一 .zip 檔案。

   下列外掛程式使用上述組件，在每次執行建置輸出目錄中建立名為 zip-with-dependencies的 mvn package .zip 檔案：

```
<plugin>
  <artifactId>maven-assembly-plugin</artifactId>
  <version>2.5.4</version>
  <executions>
    <execution>
      <phase>package</phase>
      <goals>
        <goal>single</goal>
      </goals>
      <configuration>
        <finalName>zip-with-dependencies</finalName>
        <appendAssemblyId>false</appendAssemblyId>
        <descriptors>
          <descriptor>src/main/assembly/zip.xml</descriptor>
        </descriptors>
      </configuration>
    </execution>
  </executions>
</plugin>
```

> **Note**
>
> 如果您收到錯誤，指明 1.3 中不支援註釋，請將以下內容新增至 pom.xml：
>
> ```
> <plugin>
>   <artifactId>maven-compiler-plugin</artifactId>
>   <configuration>
>     <source>1.7</source>
>     <target>1.7</target>
>   </configuration>
> </plugin>
> ```

Node.JS

若要封裝 Appium Node.js 測試並將其上傳至 Device Farm，您必須在本機電腦上安裝下列項目：

- Node Version Manager (nvm)

  請使用此工具開發和封裝測試，以便在測試套件中排除不必要的相依性。
- Node.js
- npm-bundle (全域安裝)

1. 確認 nvm 已存在

   ```
   command -v nvm
   ```

   您應該會在輸出中看到 nvm。

   如需詳細資訊，請參閱 GitHub 上的 nvm。
2. 執行此命令以安裝 Node.js：

   ```
   nvm install node
   ```

   您可以指定特定版本的 Node.js：

   ```
   nvm install 11.4.0
   ```
3. 確認使用的是正確版本的節點：

   ```
   node -v
   ```
4. 全域安裝 npm-bundle：

   ```
   npm install -g npm-bundle
   ```

Python

1. 強烈建議您設定 Python virtualenv 來進行開發和封裝測試，這樣您的應用程式套件就不會包含不必要的相依性。

```
$ virtualenv workspace
$ cd workspace
$ source bin/activate
```

> **Tip**
>
> - 請不要使用 `--system-site-packages` 選項建立 Python virtualenv，因為它會從全域 site-packages 目錄繼承套件。這可能會導致您的虛擬環境包含測試不需要的相依性。
> - 您還必須驗證您的測試未使用相依於原生程式庫的相依性，因為這些原生程式庫可能不會出現在執行測試的執行個體上。

2. 將 py.test 安裝到虛擬環境中。

```
$ pip install pytest
```

3. 在您的虛擬環境中安裝 Appium Python 用戶端。

```
$ pip install Appium-Python-Client
```

4. 除非您在自訂模式中指定不同的路徑，否則 Device Farm 預期您的測試會存放在 中`tests/`。您可以使用 `find` 來顯示資料夾內的所有檔案：

```
$ find tests/
```

確認這些檔案包含您要在 Device Farm 上執行的測試套件

```
tests/
tests/my-first-tests.py
tests/my-second-tests/py
```

5. 從虛擬環境工作區資料夾中執行此命令，以顯示測試的清單而不予以執行。

```
$ py.test --collect-only tests/
```

確認輸出顯示您想要在 Device Farm 上執行的測試。

6. 清理您的測試資料夾下的所有快取檔案：

```
$ find . -name '__pycache__' -type d -exec rm -r {} +
$ find . -name '*.pyc' -exec rm -f {} +
$ find . -name '*.pyo' -exec rm -f {} +
$ find . -name '*~' -exec rm -f {} +
```

7. 在工作區中執行下列命令，以產生 requirements.txt 檔案：

```
$ pip freeze > requirements.txt
```

Ruby

若要封裝 Appium Ruby 測試並將其上傳至 Device Farm，您必須在本機電腦上安裝下列項目：

- Ruby Version Manager (RVM)

  請使用此命令列工具開發和封裝測試，以便在測試套件中排除不必要的相依性。
- Ruby
- Bundler (此 gem 套件通常會隨 Ruby 安裝)。

1. 安裝所需的金鑰、RVM 和 Ruby。如需詳細資訊，請參閱 RVM 網站上的安裝 RVM。

   在安裝完成時，藉由登出再重新登入，來重新載入終端機。

   > **Note**
   >
   > RVM 只會做為 bash shell 的函數載入。

2. 確認已正確安裝 rvm

```
command -v rvm
```

   您應該會在輸出中看到 `rvm`。

3. 如果您想要安裝特定版本的 Ruby，例如 *`2.5.3`*，請執行下列命令：

```
rvm install ruby 2.5.3 --autolibs=0
```

   確認使用的是要求的 Ruby 版本：

```
ruby -v
```

4. 設定 Bundler 來編譯所需測試平台的套件：

```
bundle config specific_platform true
```

5. 更新您的 .lock 檔案以新增執行測試所需的平台。

   - 如果您要編譯測試以在 Android 裝置上執行，請執行此命令來設定 Gemfile 以使用 Android 測試主機的相依性：

```
bundle lock --add-platform x86_64-linux
```

   - 如果您要編譯測試以在 iOS 裝置上執行，請執行此命令來設定 Gemfile 以使用 iOS 測試主機的相依性：

```
bundle lock --add-platform x86_64-darwin
```

6. 通常預設為安裝 bundler gem。如果不是，請安裝它：

```
gem install bundler -v 2.3.26
```

建立壓縮的測試套件檔案

> **Warning**
>
> 在 Device Farm 中，壓縮測試套件中檔案的資料夾結構很重要，有些封存工具會隱含地變更 ZIP 檔案的結構。我們建議您遵循以下指定的命令列公用程式，而不是使用本機桌面 （例如 Finder 或 Windows Explorer) 的檔案管理員內建的封存公用程式。

現在，針對 Device Farm 將您的測試綁定在一起。

Java (JUnit)

建置並封裝您的測試：

```
$ mvn clean package -DskipTests=true
```

結果會建立檔案 zip-with-dependencies.zip。這是您的測試套件。

Java (TestNG)

建置並封裝您的測試：

```
$ mvn clean package -DskipTests=true
```

結果會建立檔案 zip-with-dependencies.zip。這是您的測試套件。

Node.JS

1. 檢查您的專案。

   確定您位於專案的根目錄。您可以在根目錄看到 package.json。

2. 執行此命令來安裝本機相依性。

   ```
   npm install
   ```

   此命令也會在目前的目錄中建立 node_modules 資料夾。

   > **Note**
   >
   > 此時，您應該能夠在本機執行測試。

3. 執行此命令將目前資料夾中的檔案封裝成 *.tgz 檔案。檔案會使用 package.json 檔案中的 name 屬性來命名。

   ```
   npm-bundle
   ```

   這個 tarball (.tgz) 檔案包含所有程式碼和相依性。

4. 執行此命令將前一步驟所產生的 tarball (*.tgz 檔案) 套裝成單一壓縮的存檔：

   ```
   zip -r MyTests.zip *.tgz
   ```

   這是您在下列程序中上傳至 Device Farm MyTests.zip 的檔案。

Python

Python 2

使用 pip 產生所需 Python 套件的存檔 (稱為「wheelhouse」)：

```
$ pip wheel --wheel-dir wheelhouse -r requirements.txt
```

針對 Device Farm 將您的 wheelhouse、測試和 pip 要求封裝到 zip 存檔中：

```
$ zip -r test_bundle.zip tests/ wheelhouse/ requirements.txt
```

Python 3

將您的測試和 pip 要求封裝到一個 zip 檔案中：

```
$ zip -r test_bundle.zip tests/ requirements.txt
```

Ruby

1. 執行此命令來建立虛擬 Ruby 環境：

   ```
   # myGemset is the name of your virtual Ruby environment
   rvm gemset create myGemset
   ```

2. 執行此命令來使用您剛建立的環境：

   ```
   rvm gemset use myGemset
   ```

3. 檢查您的原始程式碼。

   確定您位於專案的根目錄。您可以在根目錄看到 `Gemfile`。

4. 執行此命令從 `Gemfile` 安裝本機相依性和所有 gem 套件：

   ```
   bundle install
   ```

   > **Note**
   >
   > 此時，您應該能夠在本機執行測試。使用此命令從本機執行測試：

```
bundle exec $test_command
```

5. 將您的 gem 套件封裝在 vendor/cache 資料夾中。

```
# This will copy all the .gem files needed to run your tests into the vendor/
cache directory
bundle package --all-platforms
```

6. 執行以下命令將您的原始程式碼及所有相依性套裝到單一壓縮的存檔：

```
zip -r MyTests.zip Gemfile vendor/ $(any other source code directory files)
```

這是您在下列程序中上傳至 Device Farm MyTests.zip 的檔案。

將您的測試套件上傳至 Device Farm

您可以使用 Device Farm 主控台上傳您的測試。

1. 登入 Device Farm 主控台，網址為 https：//https://console.aws.amazon.com/devicefarm。
2. 在 Device Farm 導覽面板上，選擇行動裝置測試，然後選擇專案。
3. 如果您是新使用者，請選擇新增專案，輸入專案的名稱，然後選擇提交。

   如果您已經有專案，您可以選擇它來上傳您的測試到該專案。
4. 開啟您的專案，然後選擇建立執行。
5. 在執行設定下，為您的測試指定適當的名稱。這可包含空格或標點符號的任何組合。
6. 若是原生 Android 和 iOS 測試

   在執行設定下，如果您要測試 Android (.apk) 應用程式，請選擇 Android 應用程式；如果您要測試 iOS (.ipa) 應用程式，請選擇 iOS 應用程式。然後，在選取應用程式下，選取上傳自己的應用程式以上傳應用程式的可分發套件。

> **Note**
>
> 檔案必須是 Android .apk 或 iOS .ipa。iOS 應用程式必須是專為真實裝置建置，而不是專為模擬器建置。

若是行動 Web 應用程式測試

在執行設定下，選擇 Web 應用程式。

7. 在設定測試下，在選取測試架構區段中，選擇您使用的測試 Appium 架構，然後上傳您自己的測試套件。
8. 瀏覽並選擇包含測試的 .zip 檔案。.zip 檔案必須遵循設定您的 Appium 測試套件中所述的格式。
9. 依照指示選取裝置並開始執行。如需詳細資訊，請參閱在 Device Farm 中建立測試執行。

> **Note**
>
> Device Farm 不會修改 Appium 測試。

擷取測試的螢幕擷取畫面 （選用）

您可以在測試時取得螢幕擷取畫面。

Device Farm 會將 `DEVICEFARM_SCREENSHOT_PATH` 屬性設為本機檔案系統上的完整路徑，這是 Device Farm 預期的 Appium 螢幕擷取畫面儲存位置。用於存放螢幕擷取畫面的測試專用目錄是在執行時間定義。系統會自動將螢幕擷取畫面提取到您的 Device Farm 報告。若要檢視螢幕擷取畫面，在 Device Farm 主控台中選擇 Screenshots (螢幕擷取畫面) 區段。

如需在 Appium 測試中擷取螢幕擷取畫面的詳細資訊，請參閱 Appium API 文件中的 擷取螢幕擷取畫面。

AWS Device Farm 中的 Android 測試

Device Farm 支援多種 Android 裝置的自動化測試類型，以及兩個內建測試。

如需在 Device Farm 中測試的詳細資訊，請參閱 AWS Device Farm 中的測試架構和內建測試。

Android 應用程式測試架構

下列測試適用於 Android 裝置。

- 自動 Appium 測試

- 檢測

適用於 Android 的內建測試類型

Android 裝置可使用一種內建測試類型：

- 內建：模糊 (Android 和 iOS)

適用於 Android 和 AWS Device Farm 的檢測

Device Farm 支援 Android 的檢測 (JUnit、Espresso、Robotium 或任何檢測型測試）。

Device Farm 也提供範例 Android 應用程式，以及三個 Android 自動化架構中的工作測試連結，包括檢測 (Espresso)。適用於 Android 的 Device Farm 範例應用程式可在 GitHub 下載。

如需在 Device Farm 中測試的詳細資訊，請參閱 AWS Device Farm 中的測試架構和內建測試。

主題

- 什麼是檢測？
- Android 檢測測試的考量事項
- 標準模式測試剖析
- 將 Android Instrumentation 與 Device Farm 整合

什麼是檢測？

Android 檢測可讓您在測試程式碼中叫用回呼方法，這樣您就可以逐步執行元件的生命週期，類似於偵錯元件。如需詳細資訊，請參閱 Android 開發人員工具文件的測試類型和位置一節中的檢測測試。

Android 檢測測試的考量事項

使用 Android 檢測時，請考慮下列建議和備註。

檢查 Android 作業系統相容性

檢查 Android 文件，以確保檢測與您的 Android 作業系統版本相容。

從命令列執行

若要從命令列執行檢測測試，請遵循 Android 文件。

System Animations (動畫系統)

根據 Espresso 測試的 Android 文件，建議在實際裝置上測試時關閉系統動畫。Device Farm 會在使用 android.support.test.runner.AndroidJUnitRunner 檢測測試執行器執行時，自動停用視窗動畫縮放、轉換動畫縮放和動畫持續時間縮放設定。

Test Recorders (測試錄製器)

Device Farm 支援具有record-and-playback指令碼工具的架構，例如 Robotium。

標準模式測試剖析

在標準執行模式中，Device Farm 會剖析您的測試套件，並識別其將執行的唯一測試類別和方法。這是透過名為 Dex Test Parser 的工具來完成。

將 Android 檢測 .apk 檔案指定為輸入時，剖析器會傳回符合 JUnit 3 和 JUnit 4 慣例之測試的完整方法名稱。

若要在本機環境中測試此項目：

1. 下載`dex-test-parser`二進位檔。
2. 執行下列命令以取得將在 Device Farm 上執行的測試方法清單：

```
java -jar parser.jar path/to/apk path/for/output
```

將 Android Instrumentation 與 Device Farm 整合

> **Note**
>
> 使用下列指示將 Android 檢測測試與 AWS Device Farm 整合。如需在 Device Farm 中使用檢測測試的詳細資訊，請參閱 適用於 Android 和 AWS Device Farm 的檢測。

上傳您的 Android 檢測測試

使用 Device Farm 主控台上傳您的測試。

1. 登入 Device Farm 主控台，網址為 https：//https://console.aws.amazon.com/devicefarm。
2. 在 Device Farm 導覽面板上，選擇行動裝置測試，然後選擇專案。

3. 在專案清單中，選擇要上傳測試的專案。

> **Tip**
>
> 您可以使用搜尋列，依名稱篩選專案清單。
> 若要建立專案，請按照 在 AWS Device Farm 中建立專案 中的說明進行操作。

4. 選取建立執行。
5. 在選取應用程式下，於應用程式選取選項區段中，選取上傳自己的應用程式。
6. 瀏覽並選擇您的 Android 應用程式檔案。此檔案必須是 .pak 檔案。
7. 在設定測試下，在選取測試架構區段中，選擇檢測，然後選擇選擇檔案。
8. 瀏覽並選擇包含測試的 .apk 檔案。
9. 完成剩餘的說明以選取裝置並開始執行。

（選用）在 Android 檢測測試中擷取螢幕擷取畫面

您可以在 Android 檢測測試中擷取螢幕擷取畫面。

若要擷取螢幕擷取畫面，請呼叫下列其中一個方法：

- 若您使用 Robotium，請呼叫 `takeScreenShot` 方法 (例如，`solo.takeScreenShot();`)。
- 若您使用 Spoon，請呼叫 `screenshot` 方法，例如：

```
Spoon.screenshot(activity, "initial_state");
/* Normal test code... */
Spoon.screenshot(activity, "after_login");
```

在測試執行期間， Device Farm 會從裝置上存在的下列位置取得螢幕擷取畫面，然後將它們新增至測試報告：

- `/sdcard/robotium-screenshots`
- `/sdcard/test-screenshots`
- `/sdcard/Download/spoon-screenshots/`*`test-class-name`*`/`*`test-method-name`*
- `/data/data/`*`application-package-name`*`/app_spoon-screenshots/`*`test-class-name`*`/`*`test-method-name`*

AWS Device Farm 中的 iOS 測試

Device Farm 支援 iOS 裝置的多種自動化測試類型，以及內建測試。

如需在 Device Farm 中測試的詳細資訊，請參閱 AWS Device Farm 中的測試架構和內建測試。

iOS 應用程式測試架構

下列測試適用於 iOS 裝置。

- 自動 Appium 測試
- XCTest
- XCTest UI

適用於 iOS 的內建測試類型

目前有一種內建測試類型適用於 iOS 裝置。

- 內建：模糊 (Android 和 iOS)

將 Device Farm 與適用於 iOS 的 XCTest 整合

使用 Device Farm，您可以使用 XCTest 架構在實際裝置上測試您的應用程式。如需 XCTest 的詳細資訊，請參閱使用 Xcode 在測試中測試基本概念。

若要執行測試，請為測試執行建立套件，並將這些套件上傳至 Device Farm。

如需在 Device Farm 中測試的詳細資訊，請參閱 AWS Device Farm 中的測試架構和內建測試。

主題

- 為您的 XCTest 執行建立套件
- 將 XCTest 執行的套件上傳至 Device Farm

為您的 XCTest 執行建立套件

若要使用 XCTest 架構測試您的應用程式，Device Farm 需要下列項目：

- 以 `.ipa` 檔案提供的應用程式套件。

- 以 .zip 檔案提供的 XCTest 套件。

您使用 Xcode 產生的組建輸出來建立這些套件。請完成下列步驟來建立套件，以便您可以將它們上傳到 Device Farm。

為應用程式產生組建輸出

1. 在 Xcode 中打开应用程序项目。
2. 在 Xcode 工具列的配置下拉式功能表中，選擇 Generic iOS Device (一般 iOS 裝置) 做為目的地。
3. 在 Product (產品) 功能表中，選擇 Build For (建置對象)，然後選擇 Testing (測試)。

建立應用程式套件

1. 在 Xcode 的專案導覽器中，在 Products (產品) 下方開啟名為 *app-project-name*.app 之檔案的內容功能表。然後，選擇 Show in Finder (在尋找工具中顯示)。Finder 會開啟名為 Debug-iphoneos 的資料夾，其中包含 Xcode 為您的測試組建產生的輸出。此資料夾包含您的 .app 檔案。
2. 在 Finder 中，建立一個新資料夾並將其命名為 Payload.
3. 複製 *app-project-name*.app 檔案，並將其貼至 Payload 資料夾。
4. 開啟 Payload 資料夾的內容功能表，然後選擇 Compress "Payload" (壓縮 "Payload")。名為 Payload.zip 的檔案已建立。
5. 將 Payload.zip 的檔案名稱和副檔名變更為 *app-project-name*.ipa。

   在後續步驟中，您會將此檔案提供給 Device Farm。為了能更容易找到此檔案，您可以將它移到另一個位置，例如桌面。
6. 或者，您也可以刪除 Payload 資料夾和其中的 .app 檔案。

建立 XCTest 套件

1. 在 Finder 的 Debug-iphoneos 目錄中，開啟 *app-project-name*.app 檔案的內容功能表。然後，選擇 Show Package Contents (顯示套件內容)。
2. 在套件內容中，開啟 Plugins 資料夾。此資料夾包含名為 *app-project-name*.xctest 的檔案。
3. 開啟此檔案的內容選單，然後選擇壓縮「***app-project-name*.xctest**」。名為 *app-project-name*.xctest.zip 的檔案已建立。

在後續步驟中，您會將此檔案提供給 Device Farm。為了能更容易找到此檔案，您可以將它移到另一個位置，例如桌面。

將 XCTest 執行的套件上傳至 Device Farm

使用 Device Farm 主控台上傳測試的套件。

1. 登入 Device Farm 主控台，網址為 https：//https://console.aws.amazon.com/devicefarm。
2. 如果您還沒有專案，請加以建立。如需建立專案的步驟，請參閱在 AWS Device Farm 中建立專案。

   否則，在 Device Farm 導覽面板上，選擇行動裝置測試，然後選擇專案。
3. 選擇您要用來執行測試的專案。
4. 選擇建立執行。
5. 在執行設定下，於執行類型區段中，選擇 iOS 應用程式。
6. 在選取應用程式下，於應用程式選取選項區段中，選取上傳自己的應用程式。然後，選取上傳應用程式下的選擇檔案。
7. 瀏覽至您應用程式的 `.ipa` 檔案並上傳。

   > **Note**
   >
   > 您的 `.ipa` 套件必須是專為測試而建置。

8. 在設定測試下，於選取測試架構區段中，選擇 XCTest。然後，選取上傳應用程式下的選擇檔案。
9. 瀏覽到其中包含用於您應用程式之 XCTest 套件的 `.zip` 檔案，並將其上傳。
10. 完成專案建立程序的其餘步驟。您將會選擇您想要進行測試的裝置，並指定裝置狀態。
11. 選擇建立執行。Device Farm 會執行您的測試，並在主控台中顯示結果。

將適用於 iOS 的 XCTest UI 與 Device Farm 整合

Device Farm 支援 XCTest UI 測試架構。具體而言，Device Farm 支援以 Objective-C 和 Swift 編寫的 XCTest UI 測試。

XCTest UI 架構可在 iOS 開發中啟用 UI 測試，建置於 XCTest 之上。如需詳細資訊，請參閱 iOS Developer Library 中的 使用者界面測試。

如需在 Device Farm 中測試的一般資訊，請參閱 AWS Device Farm 中的測試架構和內建測試。

使用以下指示，將 Device Farm 與適用於 iOS 的 XCTest UI 測試架構整合。

主題

準備您的 iOS XCTest UI 測試

您可以上傳`.ipa`檔案或 XCTEST_UI 測試套件`.zip`的檔案。

`.ipa` 檔案是應用程式封存，其中包含套件格式的 iOS Runner 應用程式。檔案內不可包含其他*`.ipa`*檔案。

如果您上傳`.zip`檔案，它可以直接包含 iOS Runner 應用程式或 `.ipa` 檔案。如果您想要在測試期間使用檔案，也可以在`.zip`檔案中包含其他檔案。例如，您可以包含 或 等`.zip`檔案`.xctestrun`.`xcworkspace`.`xcodeproj`，在裝置陣列上執行 XCUI 測試計劃。有關如何執行 Test Plans 的詳細指示可在 XCUI 測試類型的預設測試規格檔案中取得。

選項 1：建立 XCTest UI .ipa 套件

當您建置專案進行測試時，Xcode 會產生 yourAppNameUITest-Runner.app 套件。您可以在專案的「產品」目錄中找到該 bundle。

若要建立 .ipa 檔案：

1. 建立名為*##*的目錄。
2. 將您的應用程式目錄新增至承載目錄。
3. 將承載目錄封存至`.zip`檔案，然後將副檔名變更為 `.ipa`。

下列資料夾結構顯示名為 *`my-project-nameUITest-Runner.app`* 的範例應用程式如何封裝為 `.ipa` 檔案：

```
.
### my-project-nameUITest.ipa
```

```
    ### Payload (directory)
        ### my-project-nameUITest-Runner.app
```

選項 2：建立 XCTest UI .zip 套件

Device Farm 會自動為您產生執行完整 XCTest UI 測試套件`.xctestrun`的檔案。如果您想要在 Device Farm 上使用自己的`.xctestrun`檔案，您可以將`.xctestrun`檔案和應用程式目錄壓縮為`.zip` 檔案。如果您已有測試套件的檔案，您可以在這裡包含該`.ipa`檔案，而不是 *`*-Runner.app`*。

```
.
### swift-sample-UI.zip (directory)
   ### my-project-nameUITest-Runner.app [OR] my-project-nameUITest.ipa
   ### SampleTestPlan_2.xctestrun
   ### SampleTestPlan_1.xctestrun
   ### (any other files)
```

如果您想要在 Device Farm 上執行 XCUI 測試的 Xcode 測試計畫，您可以建立包含 my-project-nameUITest-Runner.app 或 my-project-nameUITest.ipa 檔案的 zip，以及使用測試計畫執行 XCTEST_UI 所需的 xcode 原始程式碼檔案，包括 `.xcworkspace`或 `.xcodeproj` 檔案。

以下是使用 `.xcodeproj` 檔案的範例 zip：

```
.
### swift-sample-UI.zip (directory)
   ### my-project-nameUITest-Runner.app [OR] my-project-nameUITest.ipa
   ### (any directory)
   ### SampleXcodeProject.xcodeproj
        ### Testplan_1.xctestplan
        ### Testplan_2.xctestplan
        ### (any other source code files created by xcode with .xcodeproj)
```

以下是使用 `.xcworkspace` 檔案的範例 zip：

```
.
###swift-sample-UI.zip (directory)
   ### my-project-nameUITest-Runner.app [OR] my-project-nameUITest.ipa
   ### (any directory)
```

```
#   ### SampleXcodeProject.xcodeproj
#   ### Testplan_1.xctestplan
#   ### Testplan_2.xctestplan
|   ### (any other source code files created by xcode with .xcodeproj)
### SampleWorkspace.xcworkspace
    ### contents.xcworkspacedata
```

> **Note**
>
> 請確定 XCTest UI .zip 套件內沒有名為 "Payload" 的目錄。

上傳您的 iOS XCTest UI 測試

使用 Device Farm 主控台上傳您的測試。

1. 登入 Device Farm 主控台，網址為 https：//https://console.aws.amazon.com/devicefarm。
2. 在 Device Farm 導覽面板上，選擇行動裝置測試，然後選擇專案。
3. 在專案清單中，選擇要上傳測試的專案。

   > **Tip**
   >
   > 您可以使用搜尋列，依名稱篩選專案清單。
   > 若要建立專案，請遵循 中的指示 在 AWS Device Farm 中建立專案

4. 選擇建立執行。
5. 在執行設定下，於執行類型區段中，選擇 iOS 應用程式。
6. 在選取應用程式下，於應用程式選取選項區段中，選取上傳自己的應用程式。然後，選取上傳應用程式下的選擇檔案。
7. 瀏覽並選擇您的 iOS 應用程式檔案。該檔案必須是 .ipa 檔案。

   > **Note**
   >
   > 請確定您的 .ipa 檔案是針對 iOS 裝置所建置，而非模擬器。

8. 在設定測試下，於選取測試架構區段中，選擇 XCTest UI。然後，選取上傳應用程式下的選擇檔案。

9. 瀏覽並選擇包含 iOS XCTest UI 測試執行器的 .ipa 或 .zip 檔案。
10. 完成執行建立程序中剩餘的步驟。您將選取要測試的裝置，並選擇性地指定其他組態。
11. 選擇建立執行。Device Farm 會執行您的測試，並在主控台中顯示結果。

AWS Device Farm 中的 Web 應用程式測試

Device Farm 提供適用於 Web 應用程式的 Appium 測試。如需在 Device Farm 上設定 Appium 測試的詳細資訊，請參閱 the section called “自動 Appium 測試”。

如需在 Device Farm 中測試的詳細資訊，請參閱 AWS Device Farm 中的測試架構和內建測試。

計量和未計量裝置的規則

計量是指裝置的計費。根據預設，會測量 Device Farm 裝置，並在免費試用分鐘用盡之後，每分鐘向您收取費用。您也可以選擇購買無限制裝置，每月付固定費用即可享有無限制測試。如需定價的詳細資訊，請參閱 AWS Device Farm 定價。

如果您選擇使用包含 iOS 和 Android 裝置的裝置集區來啟動執行，則另有針對計量和無限制裝置的規則。例如，如果您有五個無限制的 Android 裝置和五個無限制的 iOS 裝置，則 Web 測試執行會使用無限制裝置。

以下是另一個範例：假設您有五個無限制的 Android 裝置和零個無限制的 iOS 裝置。如果您僅選擇 Android 裝置用於 Web 執行，則系統會使用您的無限制裝置。如果您同時選擇 Android 和 iOS 裝置用於 Web 執行，則系統會依計量收費，且不使用您的無限制裝置。

AWS Device Farm 中的內建測試

Device Farm 支援 Android 和 iOS 裝置的內建測試類型。

透過內建測試，您可以在多個裝置上測試應用程式，而無需撰寫和維護測試自動化指令碼。這可以節省您的時間和精力，尤其是當您開始使用 Device Farm 時。Device Farm 提供下列內建測試類型：

- 內建：模糊 (Android 和 iOS) – 模糊測試會隨機將使用者介面事件傳送到裝置，然後報告結果。

如需 Device Farm 中測試和測試架構的詳細資訊，請參閱AWS Device Farm 中的測試架構和內建測試。

執行 Device Farm 的內建模糊測試 (Android 和 iOS)

Device Farm 的內建模糊測試會隨機將使用者介面事件傳送到裝置，然後報告結果。

如需在 Device Farm 中測試的詳細資訊，請參閱 AWS Device Farm 中的測試架構和內建測試。

執行內建模糊測試

1. 登入 Device Farm 主控台，網址為 https：//https://console.aws.amazon.com/devicefarm。
2. 在 Device Farm 導覽面板上，選擇行動裝置測試，然後選擇專案。
3. 在專案清單中，選擇您要執行內建模糊測試的專案。

   > **Tip**
   >
   > 您可以使用搜尋列，依名稱篩選專案清單。
   > 若要建立專案，請按照 在 AWS Device Farm 中建立專案 中的說明進行操作。

4. 選擇建立執行。
5. 在執行設定下，選取執行類型區段中的執行類型。如果您沒有準備好進行測試的應用程式，或正在測試 Android 應用程式 (.apk)，請選取 Android 應用程式。如果您要測試 iOS (.ipa) 應用程式，請選取 iOS 應用程式。
6. 在選取應用程式下，如果您沒有可供測試的應用程式，請選擇選取 Device Farm 提供的範例應用程式。如果您要使用自己的應用程式，請選取上傳自己的應用程式 ，然後選擇您的應用程式檔案。
7. 在設定測試下，在選取測試架構區段中，選擇內建：模糊。
8. 如果系統顯示以下任何設定，您可以接受預設值或指定自己的值：

   - 事件計數：指定 1 到 10,000 之間的數字，表示模糊測試所要執行的使用者界面事件數。
   - 事件調節：指定介於 0 到 1，000 之間的數字，表示模糊測試在執行下一個使用者介面事件之前等待的毫秒數。
   - 亂數種子：指定數字讓模糊測試用於隨機化使用者界面事件。為後續的模糊測試指定相同數字，可確保一致的事件順序。

9. 完成剩餘的說明以選取裝置並開始執行。

AWS Device Farm 中的自訂測試環境

AWS Device Farm 可設定自訂環境以進行自動化測試（自訂模式），這是所有 Device Farm 使用者的建議方法。若要進一步了解 Device Farm 中的環境，請參閱測試環境。

與標準模式相反的自訂模式優點包括：

- 更快速end-to-end測試執行：測試套件不會剖析來偵測套件中的每個測試，避免預先處理/後製處理額外負荷。
- 即時日誌和影片串流：使用自訂模式時，您的用戶端測試日誌和影片會進行即時串流。此功能不適用於標準模式。
- 擷取所有成品：在主機和裝置上，自訂模式可讓您擷取所有測試成品。這可能無法在標準模式中執行。
- 更一致且可複製的本機環境：在標準模式中，將分別為每個個別測試提供成品，這在某些情況下可能會有所幫助。不過，由於 Device Farm 以不同的方式處理每個執行的測試，因此您的本機測試環境可能會偏離原始組態。

  相反地，自訂模式可讓您讓 Device Farm 測試執行環境與本機測試環境一致。

自訂環境是使用 YAML 格式的測試規格（測試規格）檔案來設定。Device Farm 為每種支援的測試類型提供預設測試規格檔案，可依原樣使用或自訂；自訂如測試篩選條件或組態檔案可新增至測試規格。可儲存編輯的測試規格以供未來測試執行使用。

如需詳細資訊，請參閱使用 和 上傳自訂測試規格 AWS CLI在 Device Farm 中建立測試執行。

主題

- 測試規格參考和語法
- 自訂測試環境的主機
- 使用 IAM 執行角色存取 AWS 資源
- 自訂測試環境的環境變數
- 自訂測試環境執行的最佳實務
- 將測試從標準遷移至自訂測試環境
- 在 Device Farm 中擴展自訂測試環境

測試規格參考和語法

測試規格 （ 測試規格 ） 是您用來定義 Device Farm 中自訂測試環境的檔案。

測試規格工作流程

Device Farm 測試規格會以預先決定的順序執行階段及其命令，讓您自訂環境的準備和執行方式。執行每個階段時，其命令會依測試規格檔案中列出的順序執行。階段會依下列順序執行

1. `install` - 這是應該定義下載、安裝和設定工具等動作的位置。
2. `pre_test` - 這是應該定義測試前動作的位置，例如啟動背景程序。
3. `test` - 此處應定義叫用測試的 命令。
4. `post_test` - 此處應定義測試結束後需要執行的任何最終任務，例如測試報告產生和成品檔案彙總。

測試規格語法

以下是測試規格檔案的 YAML 結構描述

```
version: 0.1

android_test_host: "string"
ios_test_host: "string"

phases:
  install:
    commands:
      - "string"
      - "string"
  pre_test:
    commands:
      - "string"
      - "string"
  test:
    commands:
      - "string"
      - "string"
  post_test:
    commands:
      - "string"
```

```
      - "string"

artifacts:
    - "string"
    - "string"
```

version

（必要，數字）

反映 Device Farm 支援的測試規格版本。目前的版本編號為 `0.1`。

android_test_host

（選用，字串）

為在 Android 裝置上執行的測試執行選取的測試主機。在 Android 裝置上執行的測試需要此欄位。如需詳細資訊，請參閱適用於自訂測試環境的測試主機。

ios_test_host

（選用，字串）

為在 iOS 裝置上執行的測試執行選取的測試主機。在主要版本大於 26 的 iOS 裝置上執行測試時，需要此欄位。如需詳細資訊，請參閱適用於自訂測試環境的測試主機。

phases

本節包含在測試執行期間執行的命令群組，其中每個階段都是選用的測試階段名稱為：`install`、`test`、 `pre_test` 和 `post_test`。

- `install` - Device Farm 支援之測試架構的預設相依性已安裝。此階段包含 Device Farm 在安裝期間執行的其他命令，如果有的話。
- `pre_test` - 命令，如果有的話，會在您的自動化測試之前執行。
- `test` - 在自動測試執行期間執行的命令。如果測試階段中的任何命令失敗（表示傳回非零結束碼），則測試會標記為失敗
- `post_test` - 命令，如果有的話，會在自動測試執行之後執行。無論您在`test`階段中的測試成功或失敗，都會執行此操作。

commands

（選用，List【string】）

在階段期間以 shell 命令執行的字串清單。

`artifacts`

（選用，List【string】）

Device Farm 會從此處指定的位置收集成品，例如自訂報告、日誌檔案和映像。成品位置不支援萬用字元，因此您必須為每個位置指定有效路徑。

這些測試成品可供測試執行中的每個裝置使用。如需擷取測試成品的詳細資訊，請參閱 在自訂測試環境中下載成品。

> **Important**
>
> 測試規格的格式必須是有效的 YAML 檔案。如果縮排或空格在您的測試規格中無效，您的測試可能會失敗。YAML 檔案不允許標籤。您可以使用 YAML 驗證程式來測試您的測試規格是否為有效的 YAML 檔案。如需詳細資訊，請參閱 YAML 網站。

測試規格範例

下列範例顯示可在 Device Farm 上執行的測試規格。

Simple Demo

以下是僅記錄`Hello world!`為測試執行成品的範例測試規格檔案。

```
version: 0.1

android_test_host: amazon_linux_2
ios_test_host: macos_sequoia

phases:
  install:
    commands:
      # Setup your environment by installing and/or validating software
      - devicefarm-cli use python 3.11
      - python --version

  pre_test:
    commands:
      # Setup your tests by starting background tasks or setting up
      # additional environment variables.
      - OUTPUT_FILE="/tmp/hello.log"
```

```
  test:
    commands:
      # Run your tests within this phase.
      - python -c 'print("Hello world!")' &> $OUTPUT_FILE

  post_test:
    commands:
      # Perform any remaining tasks within this phase, such as copying
      # artifacts to the DEVICEFARM_LOG_DIR for upload
      - cp $OUTPUT_FILE $DEVICEFARM_LOG_DIR

artifacts:
  # By default, Device Farm will collect your artifacts from the $DEVICEFARM_LOG_DIR
 directory.
  - $DEVICEFARM_LOG_DIR
```

Appium Android

以下是範例測試規格檔案，可設定在 Android 上執行的 Appium Java TestNG 測試。

```
version: 0.1

# The following fields(s) allow you to select which Device Farm test host is used
 for your test run.
android_test_host: amazon_linux_2

phases:

  # The install phase contains commands for installing dependencies to run your
 tests.
  # Certain frequently used dependencies are preinstalled on the test host to
 accelerate and
  # simplify your test setup. To find these dependencies, versions supported and
 additional
  # software installation please see:
  # https://docs.aws.amazon.com/devicefarm/latest/developerguide/custom-test-
environments-hosts-software.html
  install:
    commands:
      # The Appium server is written using Node.js. In order to run your desired
 version of Appium,
      # you first need to set up a Node.js environment that is compatible with your
 version of Appium.
```

```
      - devicefarm-cli use node 20
      - node --version

      # Use the devicefarm-cli to select a preinstalled major version of Appium.
      - devicefarm-cli use appium 2
      - appium --version

      # The Device Farm service periodically updates the preinstalled Appium
versions over time to
      # incorporate the latest minor and patch versions for each major version. If
you wish to
      # select a specific version of Appium, you can use NPM to install it.
      # - npm install -g appium@2.19.0

      # When running Android tests with Appium version 2, the uiautomator2 driver is
preinstalled using driver
      # version 2.44.1 for Appium 2.5.1  If you want to install a different version
of the driver,
      # you can use the Appium extension CLI to uninstall the existing uiautomator2
driver
      # and install your desired version:
      # - |-
      #   if [ $DEVICEFARM_DEVICE_PLATFORM_NAME = "Android" ];
      #   then
      #     appium driver uninstall uiautomator2;
      #     appium driver install uiautomator2@2.34.0;
      #   fi;

      # Based on Appium framework's recommendation, we recommend setting the Appium
server's
      # base path explicitly for accepting commands. If you prefer the legacy base
path of /wd/hub,
      # please set it here.
      - export APPIUM_BASE_PATH=

      # Use the devicefarm-cli to setup a Java environment, with which you can run
your test suite.
      - devicefarm-cli use java 17
      - java -version

  # The pre-test phase contains commands for setting up your test environment.
  pre_test:
    commands:
      # Setup the CLASSPATH so that Java knows where to find your test classes.
```

```
      - export CLASSPATH=$CLASSPATH:$DEVICEFARM_TEST_PACKAGE_PATH/*
      - export CLASSPATH=$CLASSPATH:$DEVICEFARM_TEST_PACKAGE_PATH/dependency-jars/*

      # We recommend starting the Appium server process in the background using the
 command below.
      # The Appium server log will be written to the $DEVICEFARM_LOG_DIR directory.
      # The environment variables passed as capabilities to the server will be
 automatically assigned
      # during your test run based on your test's specific device.
      # For more information about which environment variables are set and how
 they're set, please see
      # https://docs.aws.amazon.com/devicefarm/latest/developerguide/custom-test-
environment-variables.html
      - |-
        appium --base-path=$APPIUM_BASE_PATH --log-timestamp \
          --log-no-colors --relaxed-security --default-capabilities \
          "{\"appium:deviceName\": \"$DEVICEFARM_DEVICE_NAME\", \
          \"platformName\": \"$DEVICEFARM_DEVICE_PLATFORM_NAME\", \
          \"appium:udid\":\"$DEVICEFARM_DEVICE_UDID\", \
          \"appium:platformVersion\": \"$DEVICEFARM_DEVICE_OS_VERSION\", \
          \"appium:chromedriverExecutableDir\":
 \"$DEVICEFARM_CHROMEDRIVER_EXECUTABLE_DIR\", \
          \"appium:automationName\": \"UiAutomator2\"}" \
          >> $DEVICEFARM_LOG_DIR/appium.log 2>&1 &;

      # This code snippet is to wait until the Appium server starts.
      - |-
        appium_initialization_time=0;
        until curl --silent --fail "http://0.0.0.0:4723${APPIUM_BASE_PATH}/status";
 do
          if [[ $appium_initialization_time -gt 30 ]]; then
            echo "Appium did not start within 30 seconds. Exiting...";
            exit 1;
          fi;
          appium_initialization_time=$((appium_initialization_time + 1));
          echo "Waiting for Appium to start on port 4723...";
          sleep 1;
        done;

  # The test phase contains commands for running your tests.
  test:
    commands:
      # Your test package is downloaded and unpackaged into the
 $DEVICEFARM_TEST_PACKAGE_PATH directory.
```

```
      - echo "Navigate to test package directory"
      - cd $DEVICEFARM_TEST_PACKAGE_PATH
      - echo "Starting the Appium TestNG test"

      # The following command runs your Appium Java TestNG test.
      # For more information, please see TestNG's documentation here:
      # https://testng.org/#_running_testng
      - |-
        java -Dappium.screenshots.dir=$DEVICEFARM_SCREENSHOT_PATH org.testng.TestNG
 -testjar *-tests.jar \
          -d $DEVICEFARM_LOG_DIR/test-output -verbose 10

      # To run your tests with a testng.xml file that is a part of your test
 package,
      # use the following commands instead:

      # - echo "Unzipping the tests JAR file"
      # - unzip *-tests.jar
      # - |-
      #   java -Dappium.screenshots.dir=$DEVICEFARM_SCREENSHOT_PATH
 org.testng.TestNG -testjar *-tests.jar \
      #     testng.xml -d $DEVICEFARM_LOG_DIR/test-output -verbose 10

  # The post-test phase contains commands that are run after your tests have
 completed.
  # If you need to run any commands to generating logs and reports on how your test
 performed,
  # we recommend adding them to this section.
  post_test:
    commands:

# Artifacts are a list of paths on the filesystem where you can store test output
 and reports.
# All files in these paths will be collected by Device Farm, with certain limits
 (see limit details
# here: https://docs.aws.amazon.com/devicefarm/latest/developerguide/
limits.html#file-limits).
# These files will be available through the ListArtifacts API as your "Customer
 Artifacts".
artifacts:
  # By default, Device Farm will collect your artifacts from the $DEVICEFARM_LOG_DIR
 directory.
  - $DEVICEFARM_LOG_DIR
```

Appium iOS

以下是測試規格檔案範例，可設定在 iOS 上執行的 Appium Java TestNG 測試。

```
version: 0.1

# The following fields(s) allow you to select which Device Farm test host is used
 for your test run.
ios_test_host: macos_sequoia

phases:

  # The install phase contains commands for installing dependencies to run your
 tests.
  # Certain frequently used dependencies are preinstalled on the test host to
 accelerate and
  # simplify your test setup. To find these dependencies, versions supported and
 additional
  # software installation please see:
  # https://docs.aws.amazon.com/devicefarm/latest/developerguide/custom-test-
environments-hosts-software.html
  install:
    commands:
      # The Appium server is written using Node.js. In order to run your desired
 version of Appium,
      # you first need to set up a Node.js environment that is compatible with your
 version of Appium.
      - devicefarm-cli use node 20
      - node --version

      # Use the devicefarm-cli to select a preinstalled major version of Appium.
      - devicefarm-cli use appium 2
      - appium --version

      # The Device Farm service periodically updates the preinstalled Appium
 versions over time to
      # incorporate the latest minor and patch versions for each major version. If
 you wish to
      # select a specific version of Appium, you can use NPM to install it.
      # - npm install -g appium@2.19.0

      # When running iOS tests with Appium version 2, the XCUITest driver is
 preinstalled using driver
```

```
    # version 9.10.5 for Appium 2.5.4. If you want to install a different version
of the driver,
    # you can use the Appium extension CLI to uninstall the existing XCUITest
driver
    # and install your desired version:
    # - |-
    #   if [ $DEVICEFARM_DEVICE_PLATFORM_NAME = "iOS" ];
    #   then
    #     appium driver uninstall xcuitest;
    #     appium driver install xcuitest@10.0.0;
    #   fi;

    # Based on Appium framework's recommendation, we recommend setting the Appium
server's
    # base path explicitly for accepting commands. If you prefer the legacy base
path of /wd/hub,
    # please set it here.
    - export APPIUM_BASE_PATH=

    # Use the devicefarm-cli to setup a Java environment, with which you can run
your test suite.
    - devicefarm-cli use java 17
    - java -version

 # The pre-test phase contains commands for setting up your test environment.
 pre_test:
   commands:
    # Setup the CLASSPATH so that Java knows where to find your test classes.
    - export CLASSPATH=$CLASSPATH:$DEVICEFARM_TEST_PACKAGE_PATH/*
    - export CLASSPATH=$CLASSPATH:$DEVICEFARM_TEST_PACKAGE_PATH/dependency-jars/*

    # Device Farm provides multiple pre-built versions of WebDriverAgent (WDA), a
required
    # Appium dependency for iOS, where each version corresponds to the XCUITest
driver version selected.
    # If Device Farm cannot find a corresponding version of WDA for your XCUITest
driver,
    # the latest available version is selected by default.
    - |-
      APPIUM_DRIVER_VERSION=$(appium driver list --installed --json | jq -r
".xcuitest.version" | cut -d "." -f 1);
      CORRESPONDING_APPIUM_WDA=$(env | grep
"DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V${APPIUM_DRIVER_VERSION}")
```

```
       if [[ ! -z "$APPIUM_DRIVER_VERSION" ]] && [[ ! -z
 "$CORRESPONDING_APPIUM_WDA" ]]; then
         echo "Using Device Farm's prebuilt WDA version ${APPIUM_DRIVER_VERSION}.x,
 which corresponds with your driver";
         DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH=$(echo $CORRESPONDING_APPIUM_WDA |
 cut -d "=" -f2)
       else
         LATEST_SUPPORTED_WDA_VERSION=$(env | grep
 "DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V" | sort -V -r | head -n 1)
         echo "Unknown driver version $APPIUM_DRIVER_VERSION; falling back to the
 Device Farm default version of $LATEST_SUPPORTED_WDA_VERSION";
         DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH=$(echo
 $LATEST_SUPPORTED_WDA_VERSION | cut -d "=" -f2)
       fi;

     # For iOS versions 16 and below only, the device unique identifier (UDID)
 needs to modified for Appium tests
     # on Device Farm to remove the hypens.
     - |-
       if [ $DEVICEFARM_DEVICE_PLATFORM_NAME = "iOS" ]; then
         DEVICEFARM_DEVICE_UDID_FOR_APPIUM=$DEVICEFARM_DEVICE_UDID;
         if [ $(echo $DEVICEFARM_DEVICE_OS_VERSION | cut -d "." -f 1) -le 16 ];
 then
           DEVICEFARM_DEVICE_UDID_FOR_APPIUM=$(echo $DEVICEFARM_DEVICE_UDID | tr -d
 "-");
         fi;
       fi;

     # We recommend starting the Appium server process in the background using the
 command below.
     # The Appium server log will be written to the $DEVICEFARM_LOG_DIR directory.
     # The environment variables passed as capabilities to the server will be
 automatically assigned
     # during your test run based on your test's specific device.
     # For more information about which environment variables are set and how
 they're set, please see
     # https://docs.aws.amazon.com/devicefarm/latest/developerguide/custom-test-
environment-variables.html
     - |-
       appium --base-path=$APPIUM_BASE_PATH --log-timestamp \
         --log-no-colors --relaxed-security --default-capabilities \
         "{\"appium:deviceName\": \"$DEVICEFARM_DEVICE_NAME\", \
         \"platformName\": \"$DEVICEFARM_DEVICE_PLATFORM_NAME\", \
         \"appium:app\": \"$DEVICEFARM_APP_PATH\", \
```

```
          \"appium:udid\":\"$DEVICEFARM_DEVICE_UDID_FOR_APPIUM\", \
          \"appium:platformVersion\": \"$DEVICEFARM_DEVICE_OS_VERSION\", \
          \"appium:derivedDataPath\": \"$DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH\",
\
          \"appium:usePrebuiltWDA\": true, \
          \"appium:automationName\": \"XCUITest\"}" \
          >> $DEVICEFARM_LOG_DIR/appium.log 2>&1 &

     # This code snippet is to wait until the Appium server starts.
     - |-
       appium_initialization_time=0;
       until curl --silent --fail "http://0.0.0.0:4723${APPIUM_BASE_PATH}/status";
do
         if [[ $appium_initialization_time -gt 30 ]]; then
           echo "Appium did not start within 30 seconds. Exiting...";
           exit 1;
         fi;
         appium_initialization_time=$((appium_initialization_time + 1));
         echo "Waiting for Appium to start on port 4723...";
         sleep 1;
       done;

 # The test phase contains commands for running your tests.
 test:
   commands:
     # Your test package is downloaded and unpackaged into the
$DEVICEFARM_TEST_PACKAGE_PATH directory.
     - echo "Navigate to test package directory"
     - cd $DEVICEFARM_TEST_PACKAGE_PATH
     - echo "Starting the Appium TestNG test"

     # The following command runs your Appium Java TestNG test.
     # For more information, please see TestNG's documentation here:
     # https://testng.org/#_running_testng
     - |-
       java -Dappium.screenshots.dir=$DEVICEFARM_SCREENSHOT_PATH org.testng.TestNG
-testjar *-tests.jar \
         -d $DEVICEFARM_LOG_DIR/test-output -verbose 10

     # To run your tests with a testng.xml file that is a part of your test
package,
     # use the following commands instead:

     # - echo "Unzipping the tests JAR file"
```

```
      # - unzip *-tests.jar
      # - |-
      #   java -Dappium.screenshots.dir=$DEVICEFARM_SCREENSHOT_PATH
 org.testng.TestNG -testjar *-tests.jar \
      #     testng.xml -d $DEVICEFARM_LOG_DIR/test-output -verbose 10

  # The post-test phase contains commands that are run after your tests have
 completed.
  # If you need to run any commands to generating logs and reports on how your test
 performed,
  # we recommend adding them to this section.
  post_test:
    commands:

# Artifacts are a list of paths on the filesystem where you can store test output
 and reports.
# All files in these paths will be collected by Device Farm, with certain limits
 (see limit details
# here: https://docs.aws.amazon.com/devicefarm/latest/developerguide/
limits.html#file-limits).
# These files will be available through the ListArtifacts API as your "Customer
 Artifacts".
artifacts:
  # By default, Device Farm will collect your artifacts from the $DEVICEFARM_LOG_DIR
 directory.
  - $DEVICEFARM_LOG_DIR
```

Appium (Both Platforms)

以下是測試規格檔案範例，可設定在 Android 和 iOS 上執行的 Appium Java TestNG 測試。

```
version: 0.1

# The following fields(s) allow you to select which Device Farm test host is used
 for your test run.
android_test_host: amazon_linux_2
ios_test_host: macos_sequoia

phases:

  # The install phase contains commands for installing dependencies to run your
 tests.
  # Certain frequently used dependencies are preinstalled on the test host to
 accelerate and
```

```
  # simplify your test setup. To find these dependencies, versions supported and
 additional
  # software installation please see:
  # https://docs.aws.amazon.com/devicefarm/latest/developerguide/custom-test-
environments-hosts-software.html
  install:
    commands:
      # The Appium server is written using Node.js. In order to run your desired
 version of Appium,
      # you first need to set up a Node.js environment that is compatible with your
 version of Appium.
      - devicefarm-cli use node 20
      - node --version

      # Use the devicefarm-cli to select a preinstalled major version of Appium.
      - devicefarm-cli use appium 2
      - appium --version

      # The Device Farm service periodically updates the preinstalled Appium
 versions over time to
      # incorporate the latest minor and patch versions for each major version. If
 you wish to
      # select a specific version of Appium, you can use NPM to install it.
      # - npm install -g appium@2.19.0

      # When running Android tests with Appium version 2, the uiautomator2 driver is
 preinstalled using driver
      # version 2.44.1 for Appium 2.5.1  If you want to install a different version
 of the driver,
      # you can use the Appium extension CLI to uninstall the existing uiautomator2
 driver
      # and install your desired version:
      # - |-
      #   if [ $DEVICEFARM_DEVICE_PLATFORM_NAME = "Android" ];
      #   then
      #     appium driver uninstall uiautomator2;
      #     appium driver install uiautomator2@2.34.0;
      #   fi;

      # When running iOS tests with Appium version 2, the XCUITest driver is
 preinstalled using driver
      # version 9.10.5 for Appium 2.5.4. If you want to install a different version
 of the driver,
```

```
    # you can use the Appium extension CLI to uninstall the existing XCUITest
driver
    # and install your desired version:
    # - |-
    #   if [ $DEVICEFARM_DEVICE_PLATFORM_NAME = "iOS" ];
    #   then
    #     appium driver uninstall xcuitest;
    #     appium driver install xcuitest@10.0.0;
    #   fi;

    # Based on Appium framework's recommendation, we recommend setting the Appium
server's
    # base path explicitly for accepting commands. If you prefer the legacy base
path of /wd/hub,
    # please set it here.
    - export APPIUM_BASE_PATH=

    # Use the devicefarm-cli to setup a Java environment, with which you can run
your test suite.
    - devicefarm-cli use java 17
    - java -version

 # The pre-test phase contains commands for setting up your test environment.
 pre_test:
   commands:
    # Setup the CLASSPATH so that Java knows where to find your test classes.
    - export CLASSPATH=$CLASSPATH:$DEVICEFARM_TEST_PACKAGE_PATH/*
    - export CLASSPATH=$CLASSPATH:$DEVICEFARM_TEST_PACKAGE_PATH/dependency-jars/*

    # Device Farm provides multiple pre-built versions of WebDriverAgent (WDA), a
required
    # Appium dependency for iOS, where each version corresponds to the XCUITest
driver version selected.
    # If Device Farm cannot find a corresponding version of WDA for your XCUITest
driver,
    # the latest available version is selected by default.
    - |-
      if [ $DEVICEFARM_DEVICE_PLATFORM_NAME = "iOS" ]; then
        APPIUM_DRIVER_VERSION=$(appium driver list --installed --json | jq -r
".xcuitest.version" | cut -d "." -f 1);
        CORRESPONDING_APPIUM_WDA=$(env | grep
"DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V${APPIUM_DRIVER_VERSION}")
        if [[ ! -z "$APPIUM_DRIVER_VERSION" ]] && [[ ! -z
"$CORRESPONDING_APPIUM_WDA" ]]; then
```

```
            echo "Using Device Farm's prebuilt WDA version
 ${APPIUM_DRIVER_VERSION}.x, which corresponds with your driver";
            DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH=$(echo $CORRESPONDING_APPIUM_WDA
 | cut -d "=" -f2)
          else
            LATEST_SUPPORTED_WDA_VERSION=$(env | grep
 "DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V" | sort -V -r | head -n 1)
            echo "Unknown driver version $APPIUM_DRIVER_VERSION; falling back to the
 Device Farm default version of $LATEST_SUPPORTED_WDA_VERSION";
            DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH=$(echo
 $LATEST_SUPPORTED_WDA_VERSION | cut -d "=" -f2)
          fi;
        fi;

      # For iOS versions 16 and below only, the device unique identifier (UDID)
 needs to modified for Appium tests
      # on Device Farm to remove the hypens.
      - |-
        if [ $DEVICEFARM_DEVICE_PLATFORM_NAME = "iOS" ]; then
          DEVICEFARM_DEVICE_UDID_FOR_APPIUM=$DEVICEFARM_DEVICE_UDID;
          if [ $(echo $DEVICEFARM_DEVICE_OS_VERSION | cut -d "." -f 1) -le 16 ];
 then
            DEVICEFARM_DEVICE_UDID_FOR_APPIUM=$(echo $DEVICEFARM_DEVICE_UDID | tr -d
 "-");
          fi;
        fi;

      # We recommend starting the Appium server process in the background using the
 command below.
      # The Appium server log will be written to the $DEVICEFARM_LOG_DIR directory.
      # The environment variables passed as capabilities to the server will be
 automatically assigned
      # during your test run based on your test's specific device.
      # For more information about which environment variables are set and how
 they're set, please see
      # https://docs.aws.amazon.com/devicefarm/latest/developerguide/custom-test-
environment-variables.html
      - |-
        if [ $DEVICEFARM_DEVICE_PLATFORM_NAME = "Android" ]; then
          appium --base-path=$APPIUM_BASE_PATH --log-timestamp \
            --log-no-colors --relaxed-security --default-capabilities \
            "{\"appium:deviceName\": \"$DEVICEFARM_DEVICE_NAME\", \
            \"platformName\": \"$DEVICEFARM_DEVICE_PLATFORM_NAME\", \
            \"appium:udid\":\"$DEVICEFARM_DEVICE_UDID\", \
```

```
            \"appium:platformVersion\": \"$DEVICEFARM_DEVICE_OS_VERSION\", \
            \"appium:chromedriverExecutableDir\":
\"$DEVICEFARM_CHROMEDRIVER_EXECUTABLE_DIR\", \
            \"appium:automationName\": \"UiAutomator2\"}" \
            >> $DEVICEFARM_LOG_DIR/appium.log 2>&1 &
        else
          appium --base-path=$APPIUM_BASE_PATH --log-timestamp \
            --log-no-colors --relaxed-security --default-capabilities \
            "{\"appium:deviceName\": \"$DEVICEFARM_DEVICE_NAME\", \
            \"platformName\": \"$DEVICEFARM_DEVICE_PLATFORM_NAME\", \
            \"appium:udid\":\"$DEVICEFARM_DEVICE_UDID_FOR_APPIUM\", \
            \"appium:platformVersion\": \"$DEVICEFARM_DEVICE_OS_VERSION\", \
            \"appium:derivedDataPath\": \"$DEVICEFARM_WDA_DERIVED_DATA_PATH\", \
            \"appium:usePrebuiltWDA\": true, \
            \"appium:automationName\": \"XCUITest\"}" \
            >> $DEVICEFARM_LOG_DIR/appium.log 2>&1 &
        fi;

      # This code snippet is to wait until the Appium server starts.
      - |-
        appium_initialization_time=0;
        until curl --silent --fail "http://0.0.0.0:4723${APPIUM_BASE_PATH}/status";
do
          if [[ $appium_initialization_time -gt 30 ]]; then
            echo "Appium did not start within 30 seconds. Exiting...";
            exit 1;
          fi;
          appium_initialization_time=$((appium_initialization_time + 1));
          echo "Waiting for Appium to start on port 4723...";
          sleep 1;
        done;

 # The test phase contains commands for running your tests.
 test:
   commands:
     # Your test package is downloaded and unpackaged into the
$DEVICEFARM_TEST_PACKAGE_PATH directory.
     - echo "Navigate to test package directory"
     - cd $DEVICEFARM_TEST_PACKAGE_PATH
     - echo "Starting the Appium TestNG test"

     # The following command runs your Appium Java TestNG test.
     # For more information, please see TestNG's documentation here:
     # https://testng.org/#_running_testng
```

```
      - |-
        java -Dappium.screenshots.dir=$DEVICEFARM_SCREENSHOT_PATH org.testng.TestNG
 -testjar *-tests.jar \
          -d $DEVICEFARM_LOG_DIR/test-output -verbose 10

      # To run your tests with a testng.xml file that is a part of your test
 package,
      # use the following commands instead:

      # - echo "Unzipping the tests JAR file"
      # - unzip *-tests.jar
      # - |-
      #   java -Dappium.screenshots.dir=$DEVICEFARM_SCREENSHOT_PATH
 org.testng.TestNG -testjar *-tests.jar \
      #     testng.xml -d $DEVICEFARM_LOG_DIR/test-output -verbose 10

  # The post-test phase contains commands that are run after your tests have
 completed.
  # If you need to run any commands to generating logs and reports on how your test
 performed,
  # we recommend adding them to this section.
  post_test:
    commands:

# Artifacts are a list of paths on the filesystem where you can store test output
 and reports.
# All files in these paths will be collected by Device Farm, with certain limits
 (see limit details
# here: https://docs.aws.amazon.com/devicefarm/latest/developerguide/
limits.html#file-limits).
# These files will be available through the ListArtifacts API as your "Customer
 Artifacts".
artifacts:
  # By default, Device Farm will collect your artifacts from the $DEVICEFARM_LOG_DIR
 directory.
  - $DEVICEFARM_LOG_DIR
```

自訂測試環境的主機

Device Farm 透過使用測試主機環境，支援一組具有預先設定軟體的作業系統。在測試執行期間，Device Farm 會利用 Amazon 受管執行個體 （主機），以動態方式連線至待測的所選裝置。此執行個體會在執行之間完全清除且不重複使用，並在測試執行結束後以其產生的成品終止。

主題

適用於自訂測試環境的測試主機

測試主機完全由 Device Farm 管理。下表列出適用於自訂測試環境的目前可用和支援的 Device Farm 測試主機。

裝置平台	測試主機	作業系統	Architecture(s)	支援的裝置
Android	amazon_linux_2	Amazon Linux 2	x86_64	Android 6 及更高版本
iOS	macos_sequoia	macOS Sequoia（版本 15)	arm64	iOS 15 到 26

> **Note**
>
> Device Farm 會定期為裝置平台新增測試主機，以支援較新的裝置作業系統版本及其相依性。發生這種情況時，個別裝置平台的較舊測試主機會終止支援。

作業系統版本

每個可用的測試主機都會使用 Device Farm 當時支援的特定作業系統版本。雖然我們嘗試使用最新的作業系統版本，但這可能不是最新的公開分散式版本。Device Farm 會定期使用次要版本更新和安全性修補程式來更新作業系統。

若要了解測試執行期間使用之作業系統的特定版本（包括次要版本），您可以將下列程式碼片段新增至任何測試規格檔案的階段。

Example

```
phases:
  install:
    commands:
      # The following example prints the instance's operating system version details
      - |-
        if [[ "Darwin" == "$(uname)" ]]; then
          echo "$(sw_vers --productName) $(sw_vers --productVersion) ($(sw_vers --
buildVersion))";
        else
          echo "$(. /etc/os-release && echo $PRETTY_NAME) ($(uname -r))";
        fi
```

選取自訂測試環境的測試主機

您可以在測試規格檔案的適當 `android_test_host`和 `ios_test_host`變數中指定 Android 和 iOS 測試主機。 ???

如果您未為指定的裝置平台指定測試主機選擇，則測試將在 Device Farm 已設定為指定裝置和測試組態預設值的測試主機上執行。

> **Important**
>
> 在 iOS 18 及更低版本上測試時，若未選取主機，則會使用舊版測試主機。如需詳細資訊，請參閱 上的 主題舊版 iOS 測試主機。

舉例來說，請檢閱下列程式碼片段：

Example

```
version: 0.1
android_test_host: amazon_linux_2
ios_test_host: macos_sequoia

phases:
  # ...
```

自訂測試環境中支援的軟體

Device Farm 使用預先安裝許多必要軟體程式庫的主機機器來執行我們服務支援的測試架構，在啟動時提供準備好的測試環境。Device Farm 透過使用我們的軟體選擇機制支援多種語言，並定期更新環境中包含的語言版本。

如需任何其他必要的軟體，您可以修改測試規格檔案，以從測試套件安裝、從網際網路下載，或存取 VPC 中的私有來源（如需詳細資訊，請參閱 VPC ENI)。如需詳細資訊，請參閱測試規格範例。

預先設定的軟體

為了在每個平台上促進裝置測試，測試主機上提供下列工具：

工具	裝置平台 (s)
Android SDK Build-Tools	Android
Android SDK Platform-Tools（包括 adb)	Android
Xcode	iOS

可選取的軟體

除了主機上的預先設定軟體之外，Device Farm 還提供一種方法，透過 `devicefarm-cli`工具選取特定版本的支援軟體。

下表包含可選取的軟體和包含這些軟體的測試主機。

軟體/工具	支援此軟體的主機	要在測試規格中使用的命令
Java 17	amazon_linux_2 macos_sequoia	`devicefarm-cli use java 17`
Java 11	amazon_linux_2 macos_sequoia	`devicefarm-cli use java 11`
Java 8	amazon_linux_2	`devicefarm-cli use java 8`

軟體/工具	支援此軟體的主機	要在測試規格中使用的命令
	macos_sequoia	
Node.js 20	amazon_linux_2 macos_sequoia	`devicefarm-cli use node 20`
Node.js 18	amazon_linux_2 macos_sequoia	`devicefarm-cli use node 18`
Node.js 16	amazon_linux_2	`devicefarm-cli use node 16`
Python 3.11	amazon_linux_2 macos_sequoia	`devicefarm-cli use python 3.11`
Python 3.10	amazon_linux_2 macos_sequoia	`devicefarm-cli use python 3.10`
Python 3.9	amazon_linux_2 macos_sequoia	`devicefarm-cli use python 3.9`
Python 3.8	amazon_linux_2	`devicefarm-cli use python 3.8`
Ruby 3.2	amazon_linux_2 macos_sequoia	`devicefarm-cli use ruby 3.2`
Ruby 2.7	amazon_linux_2	`devicefarm-cli use ruby 2.7`
Appium 3	amazon_linux_2	`devicefarm-cli use appium 3`

軟體/工具	支援此軟體的主機	要在測試規格中使用的命令
Appium 2	amazon_linux_2 macos_sequoia	`devicefarm-cli use appium 2`
Appium 1	amazon_linux_2	`devicefarm-cli use appium 1`
Xcode 26	macos_sequoia	`devicefarm-cli use xcode 26`
Xcode 16	macos_sequoia	`devicefarm-cli use xcode 16`

測試主機也包含每個軟體版本的常用支援工具，例如 pip和 npm套件管理員（分別包含在 Python 和 Node.js 中），以及 Appium 等工具的相依性（例如 Appium UIAutomator2 驅動程式）。這可確保您擁有使用支援的測試架構所需的工具。

在自訂測試環境中使用 devicefarm-cli 工具

測試主機使用稱為 的標準化版本管理工具 `devicefarm-cli`來選取軟體版本。此工具與 分開，AWS CLI 僅適用於 Device Farm 測試主機。使用 `devicefarm-cli`，您可以在測試主機上切換到任何預先安裝的軟體版本。這可讓您直接維護 Device Farm 測試規格檔案一段時間，並為您提供可預測的機制，讓您在未來升級軟體版本。

> **Important**
>
> 此命令列工具不適用於舊版 iOS 主機。如需詳細資訊，請參閱 上的 主題舊版 iOS 測試主機。

以下程式碼片段顯示 的help頁面devicefarm-cli：

```
$ devicefarm-cli help
 Usage: devicefarm-cli COMMAND [ARGS]

     Commands:
         help                          Prints this usage message.
         list                          Lists all versions of software configurable
                                       via this CLI.
```

```
        use <software> <version>     Configures the software for usage within the
                                     current shell's environment.
```

讓我們使用 來檢閱幾個範例`devicefarm-cli`。若要使用 工具將測試規格檔案中的 Python 版本從 *3.10* 變更為 *3.9*，請執行下列命令：

```
$ python --version
Python 3.10.12
$ devicefarm-cli use python 3.9
$ python --version
Python 3.9.17
```

若要將 Appium 版本從 *1* 變更為 *2*：

```
$ appium --version
1.22.3
$ devicefarm-cli use appium 2
$ appium --version
2.1.2
```

> **Tip**
>
> 請注意，當您選取軟體版本時，`devicefarm-cli`也會切換這些語言的支援工具，例如`pip`適用於 Python 和`npm`適用於 NodeJS 的工具。

如需測試主機上預先安裝軟體的詳細資訊，請參閱 自訂測試環境中支援的軟體。

Android 裝置的測試環境

AWS Device Farm 利用執行 Amazon Linux 2 的 Amazon Elastic Compute Cloud (EC2) 主機機器來執行 Android 測試。當您排程測試執行時，Device Farm 會為每個裝置配置專用主機，以獨立執行測試。主機機器會在測試執行後與任何產生的成品一起終止。

Amazon Linux 2 主機提供數種優點：

- 更快、更可靠的測試：相較於舊版主機，新的測試主機可大幅改善測試速度，尤其是縮短測試開始時間。Amazon Linux 2 主機也會在測試期間展現更高的穩定性和可靠性。
- 用於手動測試的增強型遠端存取：升級至最新的測試主機和改進功能，可降低 Android 手動測試的延遲並改善影片效能。

- 標準軟體版本選擇：Device Farm 現在會在測試主機和 Appium 架構版本上標準化主要程式設計語言支援。對於支援的語言（目前為 Java、Python、Node.js 和 Ruby) 和 Appium，新的測試主機會在啟動後立即提供長期穩定的版本。透過 `devicefarm-cli`工具的集中式版本管理，可讓測試規格檔案開發具有跨架構的一致體驗。

主題

Device Farm 中 Amazon Linux 2 測試環境支援的 IP 範圍

客戶通常需要知道 Device Farm 流量來源的 IP 範圍，特別是在設定其防火牆和安全設定時。對於 Amazon EC2 測試主機，IP 範圍包含整個`us-west-2`區域。對於 Amazon Linux 2 測試主機，這是新 Android 執行的預設選項，範圍已受到限制。流量現在來自一組特定的 NAT 閘道，將 IP 範圍限制為下列地址：

IP 範圍
44.236.137.143
52.13.151.244
52.35.189.191
54.201.250.26

如需 Device Farm 中 Android 測試環境的詳細資訊，請參閱 Android 裝置的測試環境。

iOS 裝置的測試環境

Device Farm 利用在測試執行期間動態連線至 iOS 裝置的 Amazon 受管 macOS 執行個體（主機）。每個主機都已預先設定軟體，可在各種熱門測試平台上進行裝置測試，例如 XCTestUI 和 Appium。

相較於舊版，iOS 測試主機目前的反覆運算已改善測試體驗，包括：

- iOS 15 到 iOS 26 的一致主機作業系統和工具體驗 在此之前，測試主機是由使用中的裝置決定，導致在多個 iOS 版本上執行時產生分段的軟體環境。目前的體驗允許簡單的主機選擇，以跨裝置啟用一致的環境。這將使相同的 macOS 版本和工具（例如 Xcode) 可在每個 iOS 裝置中使用。

- iOS 15 和 16 測試的效能改善 透過更新的基礎設施，iOS 15 和 16 測試的設定時間已大幅改善。
- 受支援相依性的標準化可選取軟體版本 現在在 iOS 和 Android 測試主機上都有`devicefarm-cli`軟體選擇系統，可讓您選取我們支援的相依性的偏好版本。對於支援的相依性 （ 例如 Java、Python、Node.js、Ruby 和 Appium)，可以透過測試規格選取版本。如需此功能運作方式的概念，請參閱 上的 主題自訂測試環境中支援的軟體。

> **Important**
>
> 如果在 iOS 18 及以下版本上執行，您的測試預設會在舊版測試主機上執行。請參閱以下主題，了解如何從舊版主機遷移。

舊版 iOS 測試主機

對於 iOS 18 及以下版本的現有測試，預設會針對自訂測試環境選取舊版測試主機。下表包含由 iOS 裝置版本使用 執行的測試主機版本。

作業系統	Architecture(s)	裝置的預設
macOS Sonoma （ 第 14 版 ）	arm64	iOS 18
macOS Ventura （ 第 13 版 ）	arm64	iOS 17
macOS Monterey （ 第 12 版 ）	x86_64	iOS 16 和以下

若要選取較新的測試主機，請參閱有關 的主題將自訂測試環境遷移至新的 iOS 測試主機。

iOS 裝置支援的 軟體

為了支援 iOS 裝置測試，iOS 裝置的 Device Farm 測試主機會預先設定 Xcode 及其相關聯的命令列工具。如需其他可用的軟體，請檢閱有關 的主題自訂測試環境中支援的軟體。

將自訂測試環境遷移至新的 iOS 測試主機

若要將現有測試從舊版主機遷移到新的 macOS 測試主機，您需要根據現有的測試規格檔案來開發新的測試規格檔案。

建議的方法是從所需測試類型的範例測試規格檔案開始，然後將相關命令從舊測試規格檔案遷移至新的測試規格檔案。這可讓您利用新主機範例測試規格的新功能和最佳化，同時重複使用現有程式碼片段。

主題

教學課程：使用主控台遷移 iOS 測試規格檔案

在此範例中，Device Farm 主控台將用於加入現有的 iOS 裝置測試規格，以使用新的測試主機。

步驟 1：使用主控台建立新的測試規格檔案

1. 登入 AWS Device Farm 主控台。
2. 導覽至包含自動化測試的 Device Farm 專案。
3. 下載您要加入的現有測試規格副本。
   a. 按一下「專案設定」選項，然後導覽至上傳索引標籤。
   b. 導覽至您要加入的測試規格檔案。
   c. 按一下下載按鈕來製作此檔案的本機副本。
4. 導覽回專案頁面，然後按一下建立執行。
5. 在精靈上填寫選項，就好像您要開始新的執行一樣，但在選取測試規格選項中停止。
6. 使用預設選取的 iOS 測試規格，按一下建立測試規格按鈕。
7. 修改文字編輯器中預設選取的測試規格。
   a. 如果尚未存在，請使用下列方式修改測試規格檔案以選取新的主機：

      ```
      ios_test_host: macos_sequoia
      ```

   b. 從上一個步驟中下載的測試規格副本中，檢閱每個 `phase`。
   c. 從舊測試規格的階段將命令複製到新測試規格中的每個個別階段，忽略與安裝或選取 Java、Python、Node.js、Ruby、Appium 或 Xcode 相關的命令。
8. 在另存為文字方塊中輸入新的檔案名稱。
9. 按一下另存新檔按鈕以儲存變更。

如需可用作參考的測試規格檔案範例，請參閱 中提供的範例測試規格範例。

步驟 2：選取軟體預先安裝的軟體

在新的測試主機中，會使用稱為 的新標準化版本管理工具來選取預先安裝的軟體版本`devicefarm-cli`。此工具現在是使用我們在測試主機上提供的各種軟體的建議方法。

例如，您可以新增以下行來使用不同的 JDK 17 您的測試環境：

```
- devicefarm-cli use java 17
```

如需可用軟體的詳細資訊，請參閱：自訂測試環境中支援的軟體。

步驟 3：透過軟體選取工具使用 Appium 及其相依性

新的測試主機僅支援 Appium 2.x 及更高版本。請使用 明確選取 Appium 版本`devicefarm-cli`，同時移除舊版工具，例如 `avm`。例如：

```
# This line using 'avm' should be removed
# - avm 2.3.1

# And the following lines should be added
- devicefarm-cli use appium 2 # Selects the version
- appium --version            # Prints the version
```

搭配 選取的 Appium 版本`devicefarm-cli`已預先安裝適用於 iOS 的相容 XCUITest 驅動程式版本。

此外，您將需要更新您的測試規格以使用 ，`DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V9`而非`DEVICEFARM_WDA_DERIVED_DATA_PATH`。新的環境變數指向預先建置的 WebDriverAgent 9.x 版本，這是 Appium 2 測試的最新支援版本。

如需詳細資訊，請檢閱 為 iOS 測試選取 WebDriverAgent 版本和 Appium 測試的環境變數。

新測試主機和舊版測試主機之間的差異

當您編輯測試規格檔案以使用新的 iOS 測試主機，並從舊版測試主機轉換測試時，請注意下列主要環境差異：

- Xcode 版本：在舊版測試主機環境中，可用的 Xcode 版本是以用於測試的裝置 iOS 版本為基礎。例如，在 iOS 18 裝置上的測試在舊版主機中使用 Xcode 16，而在 iOS 17 上的測試則使用 Xcode 15。在新的主機環境中，所有裝置都可以存取相同版本的 Xcode，允許在具有不同版本的裝置上進行測試的一致環境。如需目前可用的 Xcode 版本清單，請參閱 支援的軟體。

- 選取軟體版本：在許多執行個體中，預設軟體版本已變更，因此如果您之前沒有在舊版測試主機中明確選取軟體版本，建議您現在使用 在新的測試主機中指定它devicefarm-cli。在絕大多數的使用案例中，我們建議客戶明確選取其使用的軟體版本。透過使用 選取軟體版本devicefarm-cli，您將擁有可預測且一致的使用體驗，並在 Device Farm 計劃從測試主機移除該版本時收到大量警告。

  此外，rvm已移除 nvm、 avm、 pyenv和 等軟體選取工具，以支持新的 devicefarm-cli軟體選取系統。

- 可用的軟體版本：已移除許多先前預先安裝的軟體版本，並已新增許多新版本。因此，請務必在使用 devicefarm-cli 選取軟體版本時，選取支援版本清單中的版本。

- 已移除 工具**libimobiledevice**套件，以使用較新的/第一方工具來追蹤目前的 iOS 裝置測試和業界標準。對於 iOS 17 及更高版本，您可以遷移大多數命令來使用類似的 Xcode 工具，稱為 devicectl。如需 的資訊devicectl，您可以從已安裝 Xcode 的xcrun devicectl help機器執行 。

- 在舊版主機測試規格檔案中以絕對路徑硬式編碼的檔案路徑，很可能無法如預期在新的測試主機中運作，而且通常不建議用於測試規格檔案。建議您對所有測試規格檔案程式碼使用相對路徑和環境變數。如需詳細資訊，請參閱 上的 主題自訂測試環境執行的最佳實務。

- 作業系統版本和架構：舊版測試主機根據指派的裝置使用各種 macOS 版本和 CPU 架構。因此，使用者可能會在環境中可用的系統程式庫中注意到一些差異。如需舊版主機作業系統的詳細資訊，請參閱 舊版 iOS 測試主機。

- 對於 Appium 使用者，選取 WebDriverAgent 的方式已變更為使用環境變數字首 DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V，而不是舊字 DEVICEFARM_WDA_DERIVED_DATA_PATH_V首。如需更新變數的詳細資訊，請檢閱 Appium 測試的環境變數。

- 對於 Appium Java 使用者，新的測試主機在其類別路徑中不包含任何預先安裝的 JAR 檔案，而先前的主機包含一個用於 TestNG 架構的 JAR 檔案 （透過環境變數 $DEVICEFARM_TESTNG_JAR)。我們建議客戶在其測試套件中封裝其測試架構所需的 JAR 檔案，並從其測試規格檔案中移除 $DEVICEFARM_TESTNG_JAR變數的執行個體。

如果您對測試主機與軟體之間的差異有任何意見回饋或問題，我們建議您透過支援案例與服務團隊聯絡。

使用 IAM 執行角色存取 AWS 資源

Device Farm 支援指定由自訂測試執行期環境在測試執行期間擔任的 IAM 角色。此功能可讓您的測試安全地存取帳戶中的 AWS 資源，例如 Amazon S3 儲存貯體、DynamoDB 資料表或應用程式依賴的其他 AWS 服務。

主題

概觀

當您指定 IAM 執行角色時，Device Farm 會在測試執行期間擔任此角色，讓您的測試使用角色中定義的許可與 AWS 服務互動。

IAM 執行角色的常見使用案例包括：

- 存取存放在 Amazon S3 儲存貯體中的測試資料
- 將測試成品推送至 Amazon S3 儲存貯體
- 從 AWS AppConfig 擷取應用程式組態
- 將測試日誌和指標寫入 Amazon CloudWatch
- 將測試結果或狀態訊息傳送至 Amazon SQS 佇列
- 在測試工作流程中呼叫 AWS Lambda 函數

IAM 角色需求

若要搭配 Device Farm 使用 IAM 執行角色，您的角色必須符合下列要求：

- 信任關係：必須信任 Device Farm 服務主體才能擔任該角色。信任政策必須包含 `devicefarm.amazonaws.com`做為信任的實體。
- 許可：該角色必須具備必要的許可，才能存取您的測試所需的 AWS 資源。

- 工作階段持續時間：只要 Device Farm 專案的任務逾時設定，角色的工作階段持續時間上限必須至少為 。根據預設，Device Farm 專案的任務逾時為 150 分鐘，因此您的角色必須支援至少 150 分鐘的工作階段持續時間。
- 相同的帳戶需求：IAM 角色必須與用來呼叫 Device Farm 的 AWS 帳戶位於相同的 AWS 帳戶中。不支援跨帳戶角色假設。
- PassRole 許可：呼叫者必須獲授權，才能透過允許對指定執行角色`iam:PassRole`執行動作的政策傳遞 IAM 角色。

範例信任政策

下列範例顯示允許 Device Farm 擔任執行角色的信任政策。此信任政策應僅連接至您想要與 Device Farm 搭配使用的特定 IAM 角色，而非您帳戶中的其他角色：

Example

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "devicefarm.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

許可政策範例

下列範例顯示許可政策，授予測試中常用 AWS 服務的存取權：

Example

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::my-test-bucket",
        "arn:aws:s3:::my-test-bucket/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "appconfig:GetConfiguration",
        "appconfig:StartConfigurationSession"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "arn:aws:logs:*:*:log-group:/devicefarm/test-*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "sqs:SendMessage",
        "sqs:GetQueueUrl"
      ],
      "Resource": "arn:aws:sqs:*:*:test-results-*"
    }
  ]
}
```

設定 IAM 執行角色

您可以在專案層級或個別測試執行中指定 IAM 執行角色。在專案層級設定時，該專案內的所有執行都會繼承執行角色。在執行中設定的執行角色將取代其父專案上設定的任何 。

如需設定執行角色的詳細說明，請參閱：

- 在 AWS Device Farm 中建立專案 - 在專案層級設定執行角色
- 在 Device Farm 中建立測試執行 - 用於設定個別執行的執行角色

您也可以使用 Device Farm API 設定執行角色。如需詳細資訊，請參閱 Device Farm API 參考。

最佳實務

為 Device Farm 測試設定 IAM 執行角色時，請遵循下列最佳實務：

- 最低權限原則：僅授予測試運作所需的最低許可。避免使用過於廣泛的許可，例如*動作或資源。
- 使用資源特定的許可：盡可能限制特定資源 （ 例如特定 S3 儲存貯體或 DynamoDB 資料表 ） 的許可，而不是類型的所有資源。
- 獨立測試和生產資源：使用專用測試資源和角色，以避免在測試期間意外影響生產系統。
- 定期角色檢閱：定期檢閱和更新執行角色，以確保其仍符合您的測試需求，並遵循安全最佳實務。
- 使用條件索引鍵：考慮使用 IAM 條件索引鍵進一步限制何時及如何使用角色。

疑難排解

如果您遇到 IAM 執行角色的問題，請檢查下列項目：

- 信任關係：確認角色的信任政策包含 `devicefarm.amazonaws.com`做為信任的服務。
- 許可：檢查角色是否具有測試嘗試存取之 AWS 服務的必要許可。
- 測試日誌：檢閱測試執行日誌，了解與 AWS API 呼叫或許可拒絕相關的特定錯誤訊息。

自訂測試環境的環境變數

Device Farm 會動態設定數個環境變數，做為自訂測試環境執行的一部分。

主題

自訂環境變數

Device Farm 支援在測試主機上套用為環境變數的鍵/值對組態。這些可以在 Device Farm 專案上或在執行建立期間設定；在執行上設定的任何變數都會取代其父專案上可能設定的任何變數。適用以下限制：

- 舊版 iOS 測試主機不支援自訂環境變數。如需詳細資訊，請參閱舊版 iOS 測試主機。
- 以 開頭的變數名稱$DEVICEFARM_會保留供內部服務使用。
- 自訂環境變數可能無法用於在測試規格中設定測試主機運算選擇。

常見環境變數

本節說明 Device Farm 中所有測試常見的環境變數。

$DEVICEFARM_DEVICE_NAME

測試執行所在的裝置。它代表裝置的唯一裝置識別符 (UDID)。

$DEVICEFARM_DEVICE_UDID

裝置的唯一識別符。

$DEVICEFARM_DEVICE_PLATFORM_NAME

裝置平台名稱。它是 Android或 iOS。

$DEVICEFARM_DEVICE_OS_VERSION

裝置作業系統版本。

$DEVICEFARM_APP_PATH

（行動應用程式測試）

主機上行動應用程式的路徑，測試會在此處執行。此變數在 Web 測試期間無法使用。

$DEVICEFARM_LOG_DIR

儲存客戶日誌、成品和其他所需檔案的預設目錄路徑，以供日後擷取。使用範例測試規格，此目錄中的檔案會封存在 ZIP 檔案中，並在測試執行後做為成品提供。

$DEVICEFARM_SCREENSHOT_PATH

在測試執行期間擷取之螢幕擷取畫面的路徑 (若有)。

$DEVICEFARM_PROJECT_ARN

任務父專案的 ARN。

$DEVICEFARM_RUN_ARN

任務父項執行的 ARN。

$DEVICEFARM_DEVICE_ARN

待測裝置的 ARN。

$DEVICEFARM_TOTAL_JOBS

與其父系 Device Farm 執行相關聯的任務總數。

$DEVICEFARM_JOB_NUMBER

此任務在 中的號碼$DEVICEFARM_TOTAL_JOBS。例如，執行可能包含 5 個任務，每個任務都有 0 到 4 之間的唯一$DEVICEFARM_JOB_NUMBER範圍。

$AWS_REGION

AWS 區域。服務會將此設定為符合待測裝置所在的區域。如有需要，自訂環境變數可以覆寫它。

$ANDROID_HOME

（僅限 Android)

Android SDK 安裝目錄的路徑。

Appium 測試的環境變數

本節說明 Device Farm 中自訂測試環境中任何 Appium 測試所使用的環境變數。

$DEVICEFARM_CHROMEDRIVER_EXECUTABLE_DIR

（僅限 Android)

目錄的位置，其中包含在 Appium Web 和混合測試中使用的必要 ChromeDriver 可執行檔。

$DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V<N>

（僅限 iOS)

建置在 Device Farm 上執行之 WebDriverAgent 版本的衍生資料路徑。變數上的編號將對應至 WebDriverAgent 的主要版本。例如， DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V9會指向 9.x 的 WebDriverAgent 版本。如需詳細資訊，請參閱為 iOS 測試選取 WebDriverAgent 版本。

> **Note**
>
> $DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V<N> 環境變數僅存在於非舊版 iOS 主機上。如需詳細資訊，請參閱舊版 iOS 測試主機。

$DEVICEFARM_WDA_DERIVED_DATA_PATH_V9

（僅限 iOS，已棄用）

建置在 Device Farm 上執行之 WebDriverAgent 版本的衍生資料路徑。如需替代命名結構$DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V<N>，請參閱 。

XCUITest 測試的環境變數

本節說明 XCUITest 測試在 Device Farm 的自訂測試環境中使用的環境變數。

$DEVICEFARM_XCUITESTRUN_FILE

Device Farm .xctestun 檔案的路徑。該檔案是由您的應用程式和測試套件所產生。

$DEVICEFARM_DERIVED_DATA_PATH

Device Farm xcodebuild 輸出的預期路徑。

$DEVICEFARM_XCTEST_BUILD_DIRECTORY

測試套件檔案之未解壓縮內容的路徑。

自訂測試環境執行的最佳實務

下列主題涵蓋搭配 Device Farm 使用自訂測試執行的建議最佳實務。

執行組態

- 盡可能依賴 Device Farm 受管軟體和 API 功能來執行組態，而不是透過測試規格檔案中的 shell 命令套用類似的組態。這包括測試主機和裝置的組態，因為這在測試主機和裝置之間將更具永續性和一致性。

  雖然 Device Farm 鼓勵您盡可能地自訂測試規格檔案以執行測試，但隨著更自訂的命令新增至測試規格檔案，測試規格檔案可能會隨著時間而變得難以維護。使用 Device Farm 受管軟體（透過 中的工具，例如 `devicefarm-cli`和預設可用工具$PATH)，並使用 受管功能（例如 `deviceProxy`請求參數），透過將維護責任轉移到 Device Farm 本身來簡化測試規格檔案。

測試規格和測試套件程式碼

- 請勿使用絕對路徑或依賴測試規格檔案或測試套件程式碼中的特定次要版本。Device Farm 會將例行更新套用至選取的測試主機及其包含的軟體版本。使用特定或絕對路徑（例如 `/usr/local/bin/python` 而非 `python`) 或需要特定次要版本（例如 Node.js `20.3.1` 而非僅 `20`) 可能會導致您的測試找不到所需的可執行檔 / 檔案。

  作為自訂測試執行的一部分，Device Farm 會設定各種環境變數和 $PATH變數，以確保測試在我們的動態環境中具有一致的體驗。如需詳細資訊，請參閱 自訂測試環境的環境變數 和 自訂測試環境中支援的軟體。
- 在測試執行期間，將產生的或複製的檔案儲存在暫存目錄中。 今天，我們確保使用者在測試執行期間可以存取臨時目錄 (`/tmp`)（除了受管目錄之外，例如 `$DEVICEFARM_LOG_DIR`)。由於服務或使用中的作業系統的需求，使用者可存取的其他目錄可能會隨著時間而變更。
- 將您的測試執行日誌儲存至 **`$DEVICEFARM_LOG_DIR`**。這是為您的執行提供的預設成品目錄，用於新增執行日誌/成品。我們提供的測試規格範例預設會使用此目錄做為成品。
- 確保您的命令在測試規格的 階段期間失敗時傳回非零代碼。 `test`我們會檢查`test`階段期間叫用的每個 shell 命令是否有非零的結束碼，以判斷您的執行是否失敗。您應該確保邏輯或測試架構會傳回所有所需案例的非零結束程式碼，這可能需要額外的組態。

  例如，某些測試架構（例如 JUnit5) 不會將零測試執行視為失敗，這會導致偵測到您的測試已成功執行，即使未執行任何測試也一樣。使用 JUnit5 作為範例，您需要指定命令列選項`--fail-if-no-tests`，以確保此案例以非零的結束代碼結束。
- 檢閱軟體與您將用於測試執行的裝置作業系統版本和測試主機版本的相容性。例如，在測試軟體架構（即 Appium) 中，有些功能可能無法在所測試裝置的所有作業系統版本上如預期般運作。

安全

- 避免在測試規格檔案中存放或記錄敏感變數（例如 AWS 金鑰）。 測試規格檔案、測試規格產生的指令碼和測試規格指令碼的日誌都會在測試執行結束時以可下載成品的形式提供。這可能會導致帳戶中具有測試執行讀取存取權的其他使用者秘密意外暴露。

將測試從標準遷移至自訂測試環境

您可以從標準測試執行模式切換到 AWS Device Farm 中的自訂執行模式。遷移主要涉及兩種不同的執行形式：

1. 標準模式：此測試執行模式主要是為了為客戶提供精細的報告和全受管環境。
2. 自訂模式：此測試執行模式適用於需要更快速執行測試、能夠提升和轉移並實現與其本機環境同位，以及即時影片串流的不同使用案例。

如需 Device Farm 中標準和自訂模式的詳細資訊，請參閱 在 AWS Device Farm 中測試環境和 AWS Device Farm 中的自訂測試環境。

移轉時的考量

本節列出遷移至自訂模式時要考慮的一些重要使用案例：

1. 速度：在標準執行模式中，Device Farm 會使用特定架構的封裝指示，剖析您已封裝和上傳之測試的中繼資料。剖析會偵測套件中的測試數量。之後，Device Farm 會個別執行每個測試，並針對每個測試個別顯示日誌、影片和其他結果成品。不過，這會穩定地新增end-to-end測試執行總時間，因為服務端有測試和結果成品的預處理和後處理。

   相反地，自訂執行模式不會剖析您的測試套件；這表示測試或結果成品沒有預先處理和最少的後製處理。這會導致end-to-end執行時間總計接近您的本機設定。測試的執行格式與在本機機器上執行相同（與它們相同）。測試結果與您從本機取得的結果相同，可在任務執行結束時下載。
2. 自訂或彈性：標準執行模式會剖析您的測試套件，以偵測測試數量，然後分別執行每個測試。請注意，無法保證測試將按照您指定的順序執行。因此，需要特定執行序列的測試可能無法如預期般運作。此外，無法自訂主機環境或傳遞可能需要的組態檔案，以特定方式執行測試。

   相反地，自訂模式可讓您設定主機機器環境，包括安裝其他軟體、將篩選條件傳遞至測試、傳遞組態檔案，以及控制測試執行設定。它透過 yaml 檔案（也稱為 testpec 檔案）達成此目的，您可以將 shell 命令新增至該檔案。此 yaml 檔案會轉換為在測試主機機器上執行的 shell 指令碼。您可以儲存多個 yaml 檔案，並在排程執行時根據您的需求動態選擇一個。

3. 即時影片和記錄：標準和自訂執行模式都為您提供測試的影片和日誌。不過，在標準模式中，只有在測試完成後，您才會取得測試的影片和預先定義日誌。

   相反地，自訂模式可讓您即時串流測試的視訊和用戶端日誌。此外，您可以在測試結束時（測試）下載影片和其他成品。

> **Tip**
>
> 如果您的使用案例至少涉及上述其中一個因素，強烈建議切換到自訂執行模式。

移轉步驟

若要從標準遷移到自訂模式，請執行下列動作：

1. 登入 AWS 管理主控台 並開啟 Device Farm 主控台，網址為 https：//https://console.aws.amazon.com/devicefarm/。
2. 選擇您的專案，然後啟動新的自動化執行。
3. 上傳您的應用程式（或選取 web app)、選擇您的測試架構類型、上傳您的測試套件，然後在 `Choose your execution environment` 參數下選擇 選項。 `Run your test in a custom environment`
4. 根據預設，Device Farm 的範例測試規格檔案會顯示供您檢視和編輯。此範例檔案可以用作在自訂環境模式下試用測試的起點。然後，一旦從主控台驗證測試是否正常運作，您就可以變更任何與 Device Farm 的 API、CLI 和管道整合，以在排程測試執行時使用此測試規格檔案做為參數。如需有關如何新增測試規格檔案做為執行參數的資訊，請參閱 API 指南中的 `ScheduleRun` API `testSpecArn` 參數區段。

Appium 架構

在自訂測試環境中，Device Farm 不會插入或覆寫 Appium 架構測試中的任何 Appium 功能。您必須以測試規格 YAML 檔案或測試程式碼指定您測試的 Appium 功能。

Android 檢測

您不需要進行任何變更，就可將您的 Android 檢測測試移動到自訂測試環境。

iOS XCUITest

您不需要進行任何變更，就可將您的 iOS XCUITest 測試移動到自訂測試環境。

在 Device Farm 中擴展自訂測試環境

AWS Device Farm 啟用設定自訂環境以進行自動測試（自訂模式），這是所有 Device Farm 使用者的建議方法。Device Farm 自訂模式可讓您執行的不只是測試套件。在本節中，您將了解如何擴展測試套件並最佳化測試。

如需 Device Farm 中自訂測試環境的詳細資訊，請參閱 AWS Device Farm 中的自訂測試環境。

主題

- 在 Device Farm 中執行測試時設定裝置 PIN
- 透過所需的功能加速 Device Farm 中的 Appium 型測試
- 在 Device Farm 中執行測試後，使用 Webhooks APIs
- 在 Device Farm 中將額外檔案新增至您的測試套件

在 Device Farm 中執行測試時設定裝置 PIN

有些應用程式會要求您在裝置上設定 PIN 碼。Device Farm 不支援在原生裝置上設定 PIN。不過，以下警告是可行的：

- 裝置必須執行 Android 8 或更新版本。
- 測試完成後，必須移除 PIN 碼。

若要在測試中設定 PIN，請使用 `pre_test`和 `post_test`階段來設定和移除 PIN，如下所示：

```
phases:
    pre_test:
      - # ... among your pre_test commands
      - DEVICE_PIN_CODE="1234"
      - adb shell locksettings set-pin "$DEVICE_PIN_CODE"
    post_test:
      - # ... Among your post_test commands
      - adb shell locksettings clear --old "$DEVICE_PIN_CODE"
```

當您的測試套件開始時，會設定 PIN 1234。測試套件結束之後，將會移除 PIN 碼。

> **Warning**
>
> 如果您在測試完成後沒有從裝置中移除 PIN，裝置和您的帳戶將會隔離。

如需擴展測試套件和最佳化測試的更多方法，請參閱 在 Device Farm 中擴展自訂測試環境。

透過所需的功能加速 Device Farm 中的 Appium 型測試

使用 Appium 時，您可能會發現標準模式測試套件非常慢。這是因為 Device Farm 會套用預設設定，而且不會對您想要如何使用 Appium 環境做出任何假設。雖然這些預設值是以產業最佳實務為基礎建置，但可能不適用於您的情況。若要微調 Appium 伺服器的參數，您可以調整測試規格中的預設 Appium 功能。例如，以下將 iOS 測試套件`true`中的 `usePrebuildWDA`功能設定為 ，以加快初始開始時間：

```
phases:
  pre_test:
    - # ... Start up Appium
    - >-
    appium --log-timestamp
    --default-capabilities "{\"usePrebuiltWDA\": true, \"derivedDataPath\":
\"$DEVICEFARM_WDA_DERIVED_DATA_PATH\",
    \"deviceName\": \"$DEVICEFARM_DEVICE_NAME\", \"platformName\":
\"$DEVICEFARM_DEVICE_PLATFORM_NAME\", \"app\":\"$DEVICEFARM_APP_PATH\",
    \"automationName\":\"XCUITest\", \"udid\":\"$DEVICEFARM_DEVICE_UDID_FOR_APPIUM\",
 \"platformVersion\":\"$DEVICEFARM_DEVICE_OS_VERSION\"}"
    >> $DEVICEFARM_LOG_DIR/appiumlog.txt 2>&1 &
```

Appium 功能必須是殼層逸出的 JSON 結構。

下列 Appium 功能是效能改善的常見來源：

`noReset` 和 `fullReset`

這兩個功能是互斥的，描述每個工作階段完成後 Appium 的行為。當 `noReset` 設為時`true`，Appium 伺服器不會在 Appium 工作階段結束時從您的應用程式移除資料，實際上不會進

行任何清理。 會在工作階段關閉後從裝置fullReset解除安裝並清除所有應用程式資料。如需詳細資訊，請參閱 Appium 文件中的重設策略。

ignoreUnimportantViews （僅限 Android)

指示 Appium 僅將 Android UI 階層壓縮至測試的相關檢視，以加速特定元素查詢。不過，這可能會破壞某些以 XPath 為基礎的測試套件，因為 UI 配置的階層已變更。

skipUnlock （僅限 Android)

通知 Appium 目前未設定 PIN 碼，這會加速螢幕關閉事件或其他鎖定事件之後的測試。

webDriverAgentUrl （僅限 iOS)

指示 Appium 假設基本 iOS 相依性 已在執行webDriverAgent中，並可用於在指定的 URL 接受 HTTP 請求。如果 webDriverAgent 尚未啟動並執行，則在測試套件開始時，Appium 可能需要一些時間才能啟動 webDriverAgent。如果您在啟動 Appium http://localhost:8100時webDriverAgent自行啟動並將 webDriverAgentUrl設定為 ，您可以更快地啟動測試套件。請注意，此功能不應與 useNewWDA 功能搭配使用。

您可以使用下列程式碼webDriverAgent從裝置本機連接埠 上的測試規格檔案開始8100，然後將其轉送至測試主機的本機連接埠 8100 （這可讓您將 webDriverAgentUrl的值設定為 http://localhost:8100)。在定義任何用於設定 Appium 和webDriverAgent環境變數的程式碼之後，應該在安裝階段執行此程式碼：

```
      # Start WebDriverAgent and iProxy
      - >-
        xcodebuild test-without-building -project /usr/local/avm/versions/
$APPIUM_VERSION/node_modules/appium/node_modules/appium-webdriveragent/
WebDriverAgent.xcodeproj
        -scheme WebDriverAgentRunner -derivedDataPath
 $DEVICEFARM_WDA_DERIVED_DATA_PATH
        -destination id=$DEVICEFARM_DEVICE_UDID_FOR_APPIUM
 IPHONEOS_DEPLOYMENT_TARGET=$DEVICEFARM_DEVICE_OS_VERSION
        GCC_TREAT_WARNINGS_AS_ERRORS=0 COMPILER_INDEX_STORE_ENABLE=NO >>
 $DEVICEFARM_LOG_DIR/webdriveragent_log.txt 2>&1 &

        iproxy 8100 8100 >> $DEVICEFARM_LOG_DIR/iproxy_log.txt 2>&1 &
```

然後，您可以將下列程式碼新增至您的測試規格檔案，以確保 成功webDriverAgent啟動。在確保 Appium 成功啟動之後，應該在測試前階段結束時執行此程式碼：

```
      # Wait for WebDriverAgent to start
```

```
      - >-
        start_wda_timeout=0;
        while [ true ];
        do
          if [ $start_wda_timeout -gt 60 ];
          then
              echo "WebDriverAgent server never started in 60 seconds.";
              exit 1;
          fi;
          grep -i "ServerURLHere" $DEVICEFARM_LOG_DIR/webdriveragent_log.txt >> /
dev/null 2>&1;
          if [ $? -eq 0 ];
          then
              echo "WebDriverAgent REST http interface listener started";
              break;
          else
              echo "Waiting for WebDriverAgent server to start. Sleeping for 1
 seconds";
              sleep 1;
              start_wda_timeout=$((start_wda_timeout+1));
          fi;
        done;
```

如需 Appium 支援功能的詳細資訊，請參閱 Appium 文件中的 Appium 預期功能。

如需擴展測試套件和最佳化測試的更多方法，請參閱 在 Device Farm 中擴展自訂測試環境。

在 Device Farm 中執行測試後，使用 Webhooks APIs

您可以在每個測試套件使用 完成後，讓 Device Farm 呼叫 Webhookcurl。執行此操作的程序會因目的地和格式而異。如需特定 Webhook，請參閱該 Webhook 的文件。下列範例會在測試套件完成時發佈訊息至 Slack Webhook：

```
phases:
  post_test:
    - curl -X POST -H 'Content-type: application/json' --data '{"text":"Tests on
 '$DEVICEFARM_DEVICE_NAME' have finished!"}' https://hooks.slack.com/services/
T00000000/B00000000/XXXXXXXXXXXXXXXXXXXXXXXX
```

如需搭配 Slack 使用 Webhook 的詳細資訊，請參閱 Slack API 參考中的使用 Webhook 傳送您的第一個 Slack 訊息。

如需擴展測試套件和最佳化測試的更多方法，請參閱 在 Device Farm 中擴展自訂測試環境。

您並非僅限於使用 curl 呼叫 Webhook。測試套件可以包含額外的指令碼和工具，只要它們與 Device Farm 執行環境相容。例如，您的測試套件可能包含向其他 APIs輔助指令碼。請確定任何必要的套件都與您的測試套件需求一起安裝。若要新增測試套件完成後執行的指令碼，請在測試套件中包含指令碼，並將下列項目新增至您的測試規格：

```
phases:
  post_test:
    - python post_test.py
```

> **Note**
>
> 維護測試套件中使用的任何 API 金鑰或其他身分驗證字符是您的責任。我們建議您將任何形式的安全登入資料保留在來源控制之外、盡可能使用最低權限的登入資料，以及盡可能使用可撤銷的短期權杖。若要驗證安全需求，請參閱您使用的第三方 APIs 文件。

如果您打算使用 AWS 服務做為測試執行套件的一部分，您應該使用測試套件外部產生的 IAM 臨時憑證，並包含在測試套件中。這些登入資料應具有最短的授予許可和最短的生命週期。如需建立臨時登入資料的詳細資訊，請參閱《IAM 使用者指南》中的請求臨時安全登入資料。

如需擴展測試套件和最佳化測試的更多方法，請參閱 在 Device Farm 中擴展自訂測試環境。

在 Device Farm 中將額外檔案新增至您的測試套件

您可能想要使用其他檔案做為測試的一部分，無論是額外的組態檔案或其他測試資料。您可以將這些額外的檔案新增至測試套件，然後再上傳至 AWS Device Farm，然後從自訂環境模式存取這些檔案。基本上，所有測試套件上傳格式 (ZIP、IPA、APK、JAR 等） 都是支援標準 ZIP 操作的套件封存格式。

您可以使用 AWS Device Farm 下列命令，在將檔案上傳至 之前，將檔案新增至您的測試封存：

```
$ zip zip-with-dependencies.zip extra_file
```

對於額外檔案的目錄：

```
$ zip -r zip-with-dependencies.zip extra_files/
```

這些命令適用於所有測試套件上傳格式，但 IPA 檔案除外。對於 IPA 檔案，特別是與 XCUITests 搭配使用時，我們建議您將任何額外的檔案放在稍微不同的位置，因為 會 AWS Device Farm 重新簽署 iOS 測試套件。建置 iOS 測試時，測試應用程式目錄將位於另一個名為##的目錄中。

例如，以下是這類 iOS 測試目錄的外觀：

```
$ tree
.
### Payload
    ### ADFiOSReferenceAppUITests-Runner.app
        ### ADFiOSReferenceAppUITests-Runner
        ### Frameworks
        #   ### XCTAutomationSupport.framework
        #   #   ### Info.plist
        #   #   ### XCTAutomationSupport
        #   #   ### _CodeSignature
        #   #   #   ### CodeResources
        #   #   ### version.plist
        #   ### XCTest.framework
        #       ### Info.plist
        #       ### XCTest
        #       ### _CodeSignature
        #       #   ### CodeResources
        #       ### en.lproj
        #       #   ### InfoPlist.strings
        #       ### version.plist
        ### Info.plist
        ### PkgInfo
        ### PlugIns
        #   ### ADFiOSReferenceAppUITests.xctest
        #   #   ### ADFiOSReferenceAppUITests
        #   #   ### Info.plist
        #   #   ### _CodeSignature
        #   #       ### CodeResources
        #   ### ADFiOSReferenceAppUITests.xctest.dSYM
        #       ### Contents
        #           ### Info.plist
        #           ### Resources
        #               ### DWARF
        #                   ### ADFiOSReferenceAppUITests
        ### _CodeSignature
        #   ### CodeResources
        ### embedded.mobileprovision
```

對於這些 XCUITest 套件，請將任何額外的檔案新增至*##*目錄內以 *.app* 結尾的目錄。例如，下列命令示範如何將檔案新增至此測試套件：

```
$ mv extra_file Payload/*.app/
$ zip -r my_xcui_tests.ipa Payload/
```

當您將檔案新增至測試套件時，您可以 AWS Device Farm 根據檔案的上傳格式，預期 中的互動行為會略有不同。如果上傳使用 ZIP 副檔名， AWS Device Farm 會在測試之前自動解壓縮上傳，並將解壓縮的檔案保留在具有 *$DEVICEFARM_TEST_PACKAGE_PATH* 環境變數的位置。（這表示如果您將名為 *extra_file* 的檔案新增至封存的根目錄，如第一個範例所示，它將在測試期間位於 *$DEVICEFARM_TEST_PACKAGE_PATH/extra_file*)。

若要使用更實用的範例，如果您是 Appium TestNG 使用者，想要在測試中包含 *testng.xml* 檔案，您可以使用下列命令將其包含在封存中：

```
$ zip zip-with-dependencies.zip testng.xml
```

然後，您可以在自訂環境模式中將測試命令變更為下列項目：

```
java -D appium.screenshots.dir=$DEVICEFARM_SCREENSHOT_PATH org.testng.TestNG -testjar
 *-tests.jar -d $DEVICEFARM_LOG_DIR/test-output $DEVICEFARM_TEST_PACKAGE_PATH/
testng.xml
```

如果您的測試套件上傳副檔名不是 ZIP（例如 APK、IPA 或 JAR 檔案），則可在 *$DEVICEFARM_TEST_PACKAGE_PATH* 找到上傳的套件檔案本身。由於這些檔案仍然是封存格式檔案，因此您可以解壓縮檔案，以便從內部存取其他檔案。例如，下列命令會將測試套件的內容（適用於 APK、IPA 或 JAR 檔案）解壓縮至 */tmp* 目錄：

```
unzip $DEVICEFARM_TEST_PACKAGE_PATH -d /tmp
```

如果是 APK 或 JAR 檔案，您會發現額外的檔案解壓縮到 */tmp* 目錄（例如 */tmp/extra_file*)。對於 IPA 檔案，如前所述，額外的檔案會位於 *.app* 結尾資料夾內的略有不同位置，該資料夾位於*##*目錄內。例如，根據上述 IPA 範例，檔案會位於位置 */tmp/Payload/ADFiOSReferenceAppUITests-Runner.app/extra_file*（參考為 */tmp/Payload/*.app/extra_file)#*

如需擴展測試套件和最佳化測試的更多方法，請參閱 在 Device Farm 中擴展自訂測試環境。

AWS Device Farm 中的遠端存取

遠端存取可讓您透過 Web 瀏覽器即時使用滑動、手勢與裝置互動，以測試功能並重現客戶問題。您可以與特定裝置互動，方法為利用該裝置建立遠端偵錯工作階段。

Device Farm 中的工作階段是與 Web 瀏覽器中託管的實際實體裝置進行即時互動。工作階段會顯示您在啟動工作階段時選取的單一裝置。使用者可以一次啟動多個工作階段，但同步裝置的總數受到您具有的裝置插槽數目限制。您可以根據裝置系列 (Android 或 iOS 裝置) 購買裝置插槽。如需詳細資訊，請參閱 Device Farm 定價。

Device Farm 目前提供一部分裝置進行遠端存取測試。新的裝置始終都會新增到裝置集區。

Device Farm 會擷取每個遠端存取工作階段的影片，並在工作階段期間產生活動日誌。這些結果包含您在工作階段提供的任何資訊。

> **Note**
>
> 基於安全性考量，建議您避免提供或輸入敏感資訊，例如帳戶號碼、個人登入資訊，以及遠端存取工作階段的其他詳細資訊。如果可能，請使用專為測試而開發的替代方案，例如測試帳戶。

主題

在 AWS Device Farm 中建立遠端存取工作階段

如需遠端存取工作階段的詳細資訊，請參閱工作階段。

先決條件

- 在 Device Farm 中建立專案。請遵循在 AWS Device Farm 中建立專案中的指示，然後返回此頁面。

建立遠端工作階段

Console

1. 登入 Device Farm 主控台，網址為 https：//https://console.aws.amazon.com/devicefarm。
2. 在 Device Farm 導覽面板上，選擇行動裝置測試，然後選擇專案。
3. 如果您已有專案，請從清單中選擇。否則，請依照 中的指示建立專案在 AWS Device Farm 中建立專案。
4. 在遠端存取索引標籤上，選擇建立遠端存取工作階段。
5. 為您的工作階段選擇一個裝置。您可以從可用裝置清單中選擇，或使用清單頂端的搜尋列搜尋裝置。
6. （選用）在工作階段中包含應用程式和輔助應用程式。這些可以是新上傳的應用程式，或過去 30 天內在此專案中上傳的應用程式 (30 天後，應用程式上傳將會過期)。
7. 在 Session name (工作階段名稱) 中，輸入工作階段的名稱。
8. 選擇 Confirm and start session (確認並啟動工作階段)。

AWS CLI

注意：這些指示僅著重於建立遠端存取工作階段。如需如何上傳應用程式以在工作階段期間使用的指示，請參閱自動化應用程式上傳。

首先，下載並安裝最新版本，以確認您的 AWS CLI 版本是up-to-date。

> **Important**
>
> 本文件中提及的某些命令不適用於舊版的 AWS CLI。

然後，您可以決定要在哪個裝置上測試：

```
$ aws devicefarm list-devices
```

這會顯示如下所示的輸出：

```
{
    "devices":
    [
        {
            "arn": "arn:aws:devicefarm:us-
west-2::device:DE5BD47FF3BD42C3A14BF7A6EFB1BFE7",
            "name": "Google Pixel 8",
            "remoteAccessEnabled": true,
            "availability": "HIGHLY_AVAILABLE"
            ...
        },
        ...
    ]
}
```

然後，您可以使用您選擇的裝置 ARN 建立遠端存取工作階段：

```
$ aws devicefarm create-remote-access-session \
  --project-arn arn:aws:devicefarm:us-
west-2:111122223333:project:12345678-1111-2222-333-456789abcdef \
  --device-arn arn:aws:devicefarm:us-west-2::device:DE5BD47FF3BD42C3A14BF7A6EFB1BFE7
 \
  --app-arn arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
 \
  --configuration '{
      "auxiliaryApps": [
          "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
          "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
      ]
  }'
```

這會顯示如下所示的輸出：

```
{
    "remoteAccessSession": {
        "arn": "arn:aws:devicefarm:us-
west-2:111122223333:session:abcdef123456-1234-5678-abcd-abcdef123456/
abcdef123456-1234-5678-abcd-abcdef123456/00000",
        "name": "Google Pixel 8",
```

```
        "status": "PENDING",
        ...
}
```

現在，您可以選擇輪詢並等待工作階段準備就緒：

```
$ POLL_INTERVAL=3
TIMEOUT=600
DEADLINE=$(( $(date +%s) + TIMEOUT ))

while [[ "$(date +%s)" -lt "$DEADLINE" ]]; do

  STATUS=$(aws devicefarm get-remote-access-session \
    --arn "$DEVICE_FARM_SESSION_ARN" \
    --query 'remoteAccessSession.status' \
    --output text)

  case "$STATUS" in
    RUNNING)
      echo "Session is ready with status: $STATUS"
      break
      ;;
    STOPPING|COMPLETED)
      echo "Session ended early with status: $STATUS"
      exit 1
      ;;
  esac

done
```

Python

注意：這些指示僅著重於建立遠端存取工作階段。如需如何上傳應用程式以在工作階段期間使用的指示，請參閱自動化應用程式上傳。

此範例會先在 Device Farm 上尋找任何可用的 Google Pixel 裝置，然後使用它建立遠端存取工作階段，並等待工作階段執行。

```
import random
import time
import boto3

client = boto3.client("devicefarm", region_name="us-west-2")
```

```
# 1) Gather all matching devices via paginated ListDevices with filters
filters = [
    {"attribute": "MODEL",        "operator": "CONTAINS", "values": ["Pixel"]},
    {"attribute": "AVAILABILITY", "operator": "EQUALS",   "values": ["AVAILABLE"]},
]

matching_arns = []
next_token = None
while True:
    args = {"filters": filters}
    if next_token:
        args["nextToken"] = next_token
    page = client.list_devices(**args)
    for d in page.get("devices", []):
        matching_arns.append(d["arn"])
    next_token = page.get("nextToken")
    if not next_token:
        break

if not matching_arns:
    raise RuntimeError("No available Google Pixel device found.")

# Randomly select one device from the full matching set
device_arn = random.choice(matching_arns)
print("Selected device ARN:", device_arn)

# 2) Create remote access session and wait until RUNNING
resp = client.create_remote_access_session(
    projectArn="arn:aws:devicefarm:us-
west-2:111122223333:project:12345678-1111-2222-333-456789abcdef",
    deviceArn=device_arn,
    appArn="arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
  # optional
    configuration={
        "auxiliaryApps": [  # optional
            "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
            "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
        ]
    },
)
```

```
session_arn = resp["remoteAccessSession"]["arn"]
print(f"Created Remote Access Session: {session_arn}")

poll_interval = 3
timeout = 600
deadline = time.time() + timeout
terminal_states = ["STOPPING", "COMPLETED"]

while True:
    out = client.get_remote_access_session(arn=session_arn)
    status = out["remoteAccessSession"]["status"]
    print(f"Current status: {status}")

    if status == "RUNNING":
        print(f"Session is ready with status: {status}")
        break
    if status in terminal_states:
        raise RuntimeError(f"Session ended early with status: {status}")
    if time.time() >= deadline:
        raise RuntimeError("Timed out waiting for session to be ready.")
    time.sleep(poll_interval)
```

Java

注意：這些指示僅著重於建立遠端存取工作階段。如需如何上傳應用程式以在工作階段期間使用的指示，請參閱自動化應用程式上傳。

注意：此範例使用適用於 Java v2 的 AWS 開發套件，並與 JDK 第 11 版及更高版本相容。

此範例會先在 Device Farm 上尋找任何可用的 Google Pixel 裝置，然後使用它建立遠端存取工作階段，並等待工作階段執行。

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.concurrent.ThreadLocalRandom;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.devicefarm.DeviceFarmClient;
import
 software.amazon.awssdk.services.devicefarm.model.CreateRemoteAccessSessionConfiguration;
import
 software.amazon.awssdk.services.devicefarm.model.CreateRemoteAccessSessionRequest;
```

```
import
 software.amazon.awssdk.services.devicefarm.model.CreateRemoteAccessSessionResponse;
import software.amazon.awssdk.services.devicefarm.model.Device;
import software.amazon.awssdk.services.devicefarm.model.DeviceFilter;
import software.amazon.awssdk.services.devicefarm.model.DeviceFilterAttribute;
import
 software.amazon.awssdk.services.devicefarm.model.GetRemoteAccessSessionRequest;
import
 software.amazon.awssdk.services.devicefarm.model.GetRemoteAccessSessionResponse;
import software.amazon.awssdk.services.devicefarm.model.ListDevicesRequest;
import software.amazon.awssdk.services.devicefarm.model.ListDevicesResponse;
import software.amazon.awssdk.services.devicefarm.model.RuleOperator;

public class CreateRemoteAccessSession {
  public static void main(String[] args) throws Exception {
    DeviceFarmClient client = DeviceFarmClient.builder()
        .region(Region.US_WEST_2)
        .build();

    String projectArn = "arn:aws:devicefarm:us-
west-2:111122223333:project:12345678-1111-2222-333-456789abcdef";
    String appArn     = "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
    String aux1       = "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
    String aux2       = "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789

    // 1) Gather all matching devices via paginated ListDevices with filters
    List<DeviceFilter> filters = Arrays.asList(
        DeviceFilter.builder()
            .attribute(DeviceFilterAttribute.MODEL)
            .operator(RuleOperator.CONTAINS)
            .values("Pixel")
            .build(),
        DeviceFilter.builder()
            .attribute(DeviceFilterAttribute.AVAILABILITY)
            .operator(RuleOperator.EQUALS)
            .values("AVAILABLE")
            .build()
    );

    List<String> matchingDeviceArns = new ArrayList<>();
    String next = null;
```

```
    do {
      ListDevicesResponse page = client.listDevices(
          ListDevicesRequest.builder().filters(filters).nextToken(next).build());
      for (Device d : page.devices()) {
        matchingDeviceArns.add(d.arn());
      }
      next = page.nextToken();
    } while (next != null);

    if (matchingDeviceArns.isEmpty()) {
      throw new RuntimeException("No available Google Pixel device found.");
    }

    // Randomly select one device from the full matching set
    String deviceArn = matchingDeviceArns.get(
        ThreadLocalRandom.current().nextInt(matchingDeviceArns.size()));
    System.out.println("Selected device ARN: " + deviceArn);

    // 2) Create Remote Access session and wait until it is RUNNING
    CreateRemoteAccessSessionConfiguration cfg =
CreateRemoteAccessSessionConfiguration.builder()
        .auxiliaryApps(Arrays.asList(aux1, aux2))
        .build();

    CreateRemoteAccessSessionResponse res = client.createRemoteAccessSession(
        CreateRemoteAccessSessionRequest.builder()
            .projectArn(projectArn)
            .deviceArn(deviceArn)
            .appArn(appArn)       // optional
            .configuration(cfg)   // optional
            .build());

    String sessionArn = res.remoteAccessSession().arn();
    System.out.println("Created Remote Access Session: " + sessionArn);

    int pollIntervalMs = 3000;
    long timeoutMs = 600_000L;
    long deadline = System.currentTimeMillis() + timeoutMs;

    while (true) {
      GetRemoteAccessSessionResponse get = client.getRemoteAccessSession(
          GetRemoteAccessSessionRequest.builder().arn(sessionArn).build());
      String status = get.remoteAccessSession().statusAsString();
      System.out.println("Current status: " + status);
```

```
      if ("RUNNING".equals(status)) {
        System.out.println("Session is ready with status: " + status);
        break;
      }
      if ("STOPPING".equals(status) || "COMPLETED".equals(status)) {
        throw new RuntimeException("Session ended early with status: " + status);
      }
      if (System.currentTimeMillis() >= deadline) {
        throw new RuntimeException("Timed out waiting for session to be ready.");
      }
      Thread.sleep(pollIntervalMs);
    }
  }
}
```

JavaScript

注意：這些指示僅著重於建立遠端存取工作階段。如需如何上傳應用程式以在工作階段期間使用的指示，請參閱自動化應用程式上傳。

注意：此範例使用適用於 JavaScript v3 的 AWS 開發套件。

此範例會先在 Device Farm 上尋找任何可用的 Google Pixel 裝置，然後使用它建立遠端存取工作階段，並等待工作階段執行。

```
import {
  DeviceFarmClient,
  ListDevicesCommand,
  CreateRemoteAccessSessionCommand,
  GetRemoteAccessSessionCommand,
} from "@aws-sdk/client-device-farm";

const client = new DeviceFarmClient({ region: "us-west-2" });

// 1) Gather all matching devices via paginated ListDevices with filters
const filters = [
  { attribute: "MODEL",        operator: "CONTAINS", values: ["Pixel"] },
  { attribute: "AVAILABILITY", operator: "EQUALS",   values: ["AVAILABLE"] },
];

let nextToken;
const matching = [];
```

```
while (true) {
  const page = await client.send(new ListDevicesCommand({ filters, nextToken }));
  for (const d of page.devices ?? []) {
    matching.push(d.arn);
  }
  nextToken = page.nextToken;
  if (!nextToken) break;
}

if (matching.length === 0) {
  throw new Error("No available Google Pixel device found.");
}

// Randomly select one device from the full matching set
const deviceArn = matching[Math.floor(Math.random() * matching.length)];
console.log("Selected device ARN:", deviceArn);

// 2) Create remote access session and wait until RUNNING
const out = await client.send(new CreateRemoteAccessSessionCommand({
  projectArn: "arn:aws:devicefarm:us-
west-2:111122223333:project:12345678-1111-2222-333-456789abcdef",
  deviceArn,
  appArn: "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
 optional
  configuration: {
    auxiliaryApps: [ // optional
      "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
      "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
    ],
  },
}));

const sessionArn = out.remoteAccessSession?.arn;
console.log("Created Remote Access Session:", sessionArn);

const pollIntervalMs = 3000;
const timeoutMs = 600000;
const deadline = Date.now() + timeoutMs;

while (true) {
```

```
  const get = await client.send(new GetRemoteAccessSessionCommand({ arn:
 sessionArn }));
  const status = get.remoteAccessSession?.status;
  console.log("Current status:", status);

  if (status === "RUNNING") {
    console.log("Session is ready with status:", status);
    break;
  }
  if (status === "STOPPING" || status === "COMPLETED") {
    throw new Error(`Session ended early with status: ${status}`);
  }
  if (Date.now() >= deadline) {
    throw new Error("Timed out waiting for session to be ready.");
  }
  await new Promise((r) => setTimeout(r, pollIntervalMs));
}
```

C#

注意：這些指示僅著重於建立遠端存取工作階段。如需如何上傳應用程式以在工作階段期間使用的指示，請參閱自動化應用程式上傳。

此範例會先在 Device Farm 上尋找任何可用的 Google Pixel 裝置，然後使用它建立遠端存取工作階段，並等待工作階段執行。

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon;
using Amazon.DeviceFarm;
using Amazon.DeviceFarm.Model;

class Program
{
    static async Task Main()
    {
        var client = new AmazonDeviceFarmClient(RegionEndpoint.USWest2);

        // 1) Gather all matching devices via paginated ListDevices with filters
        var filters = new List<DeviceFilter>
        {
```

```
            new DeviceFilter { Attribute = DeviceAttribute.MODEL,        Operator =
 RuleOperator.CONTAINS, Values = new List<string>{ "Pixel" } },
            new DeviceFilter { Attribute = DeviceAttribute.AVAILABILITY, Operator =
 RuleOperator.EQUALS,   Values = new List<string>{ "AVAILABLE" } },
        };

        var matchingArns = new List<string>();
        string nextToken = null;

        do
        {
            var list = await client.ListDevicesAsync(new ListDevicesRequest
            {
                Filters = filters,
                NextToken = nextToken
            });

            foreach (var d in list.Devices)
                matchingArns.Add(d.Arn);

            nextToken = list.NextToken;
        }
        while (nextToken != null);

        if (matchingArns.Count == 0)
            throw new Exception("No available Google Pixel device found.");

        // Randomly select one device from the full matching set
        var rnd = new Random();
        var deviceArn = matchingArns[rnd.Next(matchingArns.Count)];
        Console.WriteLine($"Selected device ARN: {deviceArn}");

        // 2) Create remote access session and wait until RUNNING
        var request = new CreateRemoteAccessSessionRequest
        {
            ProjectArn = "arn:aws:devicefarm:us-
west-2:111122223333:project:12345678-1111-2222-333-456789abcdef",
            DeviceArn  = deviceArn,
            AppArn     = "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
 optional
            Configuration = new CreateRemoteAccessSessionConfiguration
            {
                AuxiliaryApps = new List<string>
```

```
                {
                    "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
                    "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
                }
            }
        };

        request.Configuration.AuxiliaryApps.RemoveAll(string.IsNullOrWhiteSpace);

        var response = await client.CreateRemoteAccessSessionAsync(request);
        var sessionArn = response.RemoteAccessSession.Arn;
        Console.WriteLine($"Created Remote Access Session: {sessionArn}");

        var pollIntervalMs = 3000;
        var timeoutMs = 600000;
        var deadline = DateTime.UtcNow.AddMilliseconds(timeoutMs);

        while (true)
        {
            var get = await client.GetRemoteAccessSessionAsync(new
 GetRemoteAccessSessionRequest { Arn = sessionArn });
            var status = get.RemoteAccessSession.Status.Value;
            Console.WriteLine($"Current status: {status}");

            if (status == "RUNNING")
            {
                Console.WriteLine($"Session is ready with status: {status}");
                break;
            }
            if (status == "STOPPING" || status == "COMPLETED")
            {
                throw new Exception($"Session ended early with status: {status}");
            }
            if (DateTime.UtcNow >= deadline)
            {
                throw new TimeoutException("Timed out waiting for session to be
 ready.");
            }

            await Task.Delay(pollIntervalMs);
        }
    }
```

```
}
```

Ruby

注意：這些指示僅著重於建立遠端存取工作階段。如需如何上傳應用程式以在工作階段期間使用的指示，請參閱自動化應用程式上傳。

此範例會先在 Device Farm 上尋找任何可用的 Google Pixel 裝置，然後使用它建立遠端存取工作階段，並等待工作階段執行。

```
require 'aws-sdk-devicefarm'

client = Aws::DeviceFarm::Client.new(region: 'us-west-2')

# 1) Gather all matching devices via paginated ListDevices with filters
filters = [
  { attribute: 'MODEL',        operator: 'CONTAINS', values: ['Pixel'] },
  { attribute: 'AVAILABILITY', operator: 'EQUALS',   values: ['AVAILABLE'] },
]

matching_arns = []
next_token = nil
loop do
  resp = client.list_devices(filters: filters, next_token: next_token)
  resp.devices&.each { |d| matching_arns << d.arn }
  next_token = resp.next_token
  break unless next_token
end

abort "No available Google Pixel device found." if matching_arns.empty?

# Randomly select one device from the full matching set
device_arn = matching_arns.sample
puts "Selected device ARN: #{device_arn}"

# 2) Create remote access session and wait until RUNNING
resp = client.create_remote_access_session(
  project_arn: "arn:aws:devicefarm:us-
west-2:111122223333:project:12345678-1111-2222-333-456789abcdef",
  device_arn:  device_arn,
  app_arn:     "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
 # optional
```

```
  configuration: {
    auxiliary_apps: [ # optional
      "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
      "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
    ].compact
  }
)

session_arn = resp.remote_access_session.arn
puts "Created Remote Access Session: #{session_arn}"

poll_interval = 3
timeout = 600
deadline = Time.now + timeout
terminal = %w[STOPPING COMPLETED]

loop do
  get = client.get_remote_access_session(arn: session_arn)
  status = get.remote_access_session.status
  puts "Current status: #{status}"

  if status == 'RUNNING'
    puts "Session is ready with status: #{status}"
    break
  end

  abort "Session ended early with status: #{status}" if terminal.include?(status)
  abort "Timed out waiting for session to be ready." if Time.now >= deadline
  sleep poll_interval
end
```

後續步驟

Device Farm 會在請求的裝置和基礎設施可用時立即啟動工作階段，通常在幾分鐘內。裝置請求對話方塊會出現，直到工作階段啟動為止。若要取消工作階段請求，請選擇 Cancel request (取消請求)。

如果選取的裝置無法使用或忙碌，工作階段狀態會顯示為待定裝置，表示您可能需要等待一段時間，裝置才能進行測試。

如果您的帳戶已達到公有計量或未計量裝置的並行限制，工作階段狀態會顯示為待定並行。對於未計量的裝置插槽，您可以透過購買更多裝置插槽來增加並行。對於按pay-as-you-go計費的裝置，請透過支援票證聯絡 AWS，以請求提高服務配額。

當工作階段設定開始時，它會先顯示進行中的狀態，然後在本機 Web 瀏覽器嘗試開啟與裝置的遠端連線時顯示連線狀態。

在工作階段開始之後，如果您應該關閉瀏覽器或瀏覽器標籤，而無需停止工作階段，或如果失去瀏覽器與網際網路之間的連線，則工作階段仍有五分鐘保持作用中狀態。之後，Device Farm 會結束工作階段。您的帳戶需要為閒置時間付費。

工作階段開始後，您可以在 Web 瀏覽器中與裝置互動，或使用 Appium 測試裝置。

在 AWS Device Farm 中使用遠端存取工作階段

如需透過遠端存取工作階段執行 Android 和 iOS 應用程式互動式測試的詳細資訊，請參閱工作階段。

- 先決條件
- 在 Device Farm 主控台中使用工作階段
- 後續步驟
- 秘訣和技巧

先決條件

- 建立工作階段。請遵循建立工作階段中的指示，然後返回此頁面。

在 Device Farm 主控台中使用工作階段

一旦您針對遠端存取工作階段請求的裝置可供使用，主控台就會顯示裝置畫面。工作階段的長度上限為 150 分鐘。工作階段中剩餘的時間會出現在裝置名稱附近的左時間欄位中。

安裝應用程式

若要在工作階段裝置上安裝應用程式，請在安裝應用程式中選取選擇檔案，然後選擇您要安裝的 .apk 檔案 (Android) 或 .ipa 檔案 (iOS)。您在遠端存取工作階段中執行的應用程式不需進行任何測試檢測或佈建作業。

> **Note**
>
> 當您上傳應用程式時，有時在應用程式可供使用前會發生延遲。系統會顯示確認訊息，讓您知道應用程式是否已成功安裝。

控制裝置

如同操作實際的實體裝置一般，您可使用滑鼠或相容裝置進行觸控，也可使用裝置的螢幕鍵盤，藉此與主控台中顯示的裝置互動。若為 Android 裝置，View controls (檢視控制) 中有按鈕，其功能類似 Android 裝置上的 Home (首頁) 和 Back (返回) 按鈕。若為 iOS 裝置，有一個 Home (首頁) 按鈕，其功能類似 iOS 裝置上的首頁按鈕。您也可以選擇最近應用程式，在裝置上執行的應用程式之間切換。

在縱向和橫向模式之間切換

您也可以為您正在使用的裝置切換縱向 （ 垂直 ） 和橫向 （ 水平 ） 模式。

後續步驟

Device Farm 會繼續工作階段，直到您手動停止或達到 150 分鐘的時間限制為止。若要結束工作階段，請選擇停止工作階段。當工作階段停止後，您即可存取擷取的影片或產生的日誌。如需詳細資訊，請參閱擷取工作階段結果。

秘訣和技巧

在某些 AWS 區域中，您可能遇到遠端存取工作階段的效能問題。部分原因是某些區域中發生延遲所致。如果您遇到效能問題，請等待遠端存取工作階段追上進度，再重新與應用程式互動。

在 AWS Device Farm 中擷取遠端存取工作階段的結果

如需工作階段的詳細資訊，請參閱 工作階段。

- 先決條件
- 檢視工作階段詳細資訊
- 下載工作階段影片或日誌

先決條件

- 完成工作階段。請遵循在 AWS Device Farm 中使用遠端存取工作階段中的指示，然後返回此頁面。

檢視工作階段詳細資訊

當遠端存取工作階段結束時，Device Farm 主控台會顯示包含工作階段期間活動詳細資訊的資料表。如需詳細資訊，請參閱分析日誌資訊。

若之後要回到工作階段的詳細資訊：

1. 在 Device Farm 導覽面板上，選擇行動裝置測試，然後選擇專案。
2. 選擇包含工作階段的專案。
3. 選擇遠端存取，然後從清單中選擇您要檢閱的工作階段。

下載工作階段影片或日誌

當遠端存取工作階段結束時，Device Farm 主控台會提供工作階段和活動日誌的影片擷取存取權。在工作階段結果中，選擇 Files (檔案) 標籤以取得工作階段影片和日誌的連結。您可以在瀏覽器中檢視這些檔案或將其儲存在本機。

AWS Device Farm 中的 Appium 測試

在遠端存取工作階段期間，您可以從本機環境執行 Appium 測試，並使用受管 Appium 端點鎖定工作階段的裝置。透過 Appium 端點，您可以利用快速回饋和快速迭代來開發、測試和執行 Appium 程式碼。此用戶端測試方法可讓您靈活地從您選擇的任何 Appium 用戶端環境連線至 Device Farm 裝置。

為了補充用戶端測試，Device Farm 也支援在由 服務管理的基礎設施上執行測試，稱為伺服器端執行。在此方法中，您可以將應用程式和測試上傳至服務，然後使用服務受管測試主機在多個裝置上平行執行測試。此方法非常適合在許多裝置上獨立進行測試，以及根據 CI/CD 管道的內容進行測試。

若要進一步了解伺服器端執行，請參閱 測試架構和內建測試。

主題

什麼是 Appium 端點？

Appium 是熱門的開放原始碼軟體測試架構，可在不同的裝置上測試 iOS 和 Android 的原生、混合和行動 Web 應用程式，包括行動電話和平板電腦。它可讓開發人員和 QA （品質保證） 工程師撰寫指令碼，以遠端控制裝置、模擬使用者互動，並確認測試中的應用程式如預期般運作。Appium 從最終使用者的角度與應用程式互動，讓測試人員能夠開發測試，模擬真實使用者將如何使用應用程式進行測試。

Appium 是以用戶端-伺服器模型為基礎，其中本機用戶端會請求 （本機或遠端） Appium 伺服器代其命令裝置。Appium 伺服器會管理與裝置通訊的驅動程式，例如適用於 Android 的 UIAutomator2 驅動程式或適用於 iOS 的 XCUITest 驅動程式。所有命令都遵循 W3C WebDriver 標準，了解如何控制裝置。

Device Farm 的 Appium 端點會在遠端存取工作階段中公開裝置的 Appium 伺服器 URL。Appium 端點 URL 將專屬於該工作階段中的該裝置，並在工作階段期間保持有效，可讓您在相同裝置上迭代，而無需額外的設定時間。如需遠端存取的詳細資訊，請參閱 遠端存取。

Appium 測試入門

對於大多數 Appium 使用者，使用 Device Farm for Appium 測試只需要對現有測試組態進行次要變更。

在高階，使用 Device Farm 進行用戶端 Appium 測試有三個步驟：

1. 首先，您需要建立遠端存取工作階段來測試 Device Farm 裝置。您可以在遠端存取請求中包含您的應用程式，或在工作階段開始後安裝應用程式。
2. 工作階段執行後，您可以複製 Appium 端點 URL，並透過獨立工具（例如 Appium Inspector) 或從 IDE 中的 Appium 測試程式碼使用它。URL 在遠端存取工作階段期間有效。
3. 最後，一旦您的 Appium 測試開始，您就可以在測試執行期間與裝置的影片串流一起即時檢閱 Appium 伺服器日誌。

使用 Appium 與裝置互動

建立遠端存取工作階段後，裝置將可用於 Appium 測試。在遠端存取工作階段的整個期間，您可以視需要在裝置上執行任意數量的 Appium 工作階段，而不會限制您使用的用戶端。例如，您可以從 IDE 使用本機 Appium 程式碼執行測試，然後切換到使用 Appium Inspector 來疑難排解您遇到的任何問題。工作階段最多可持續 150 分鐘，不過，如果超過 5 分鐘沒有活動（透過互動式主控台或透過 Appium 端點），工作階段將會逾時。

使用應用程式搭配 Appium 工作階段進行測試

Device Farm 可讓您使用應用程式作為遠端存取工作階段建立請求的一部分，或在遠端存取工作階段本身期間安裝應用程式。這些應用程式會自動安裝在待測裝置上，並插入為任何 Appium 工作階段請求的預設功能。當您建立遠端存取工作階段時，您可以選擇傳入應用程式 ARN，該應用程式 ARN 預設會用作所有後續 Appium 工作階段`appium:app`的功能，以及輔助應用程式 ARNs，其將用作`appium:otherApps`功能。

例如，如果您使用 應用程式`com.aws.devicefarm.sample`建立遠端存取工作階段，並`com.aws.devicefarm.other.sample`做為其中一個輔助應用程式，則當您前往建立 Appium 工作階段時，它將具有類似下列的功能：

```
{
    "value":
    {
```

```
        "sessionId": "abcdef123456-1234-5678-abcd-abcdef123456",
        "capabilities":
        {
            "app": "/tmp/com.aws.devicefarm.sample.apk",
            "otherApps": "[\"/tmp/com.aws.devicefarm.other.sample.apk\"]",
            ...
        }
    }
}
```

在工作階段期間，您可以安裝其他應用程式（在主控台內或使用 InstallToRemoteAccessSession API)。這些將覆寫先前用作 appium:app功能的任何現有應用程式。如果先前使用的應用程式具有不同的套件名稱，它們將保留在裝置上，並用作appium:otherApps功能的一部分。

例如，如果您最初在建立遠端存取工作階段com.aws.devicefarm.sample時使用應用程式，然後在工作階段com.aws.devicefarm.other.sample期間安裝名為 的新應用程式，則您的 Appium 工作階段將具有類似下列的功能：

```
{
    "value":
    {
        "sessionId": "abcdef123456-1234-5678-abcd-abcdef123456",
        "capabilities":
        {
            "app": "/tmp/com.aws.devicefarm.other.sample.apk",
            "otherApps": "[\"/tmp/com.aws.devicefarm.sample.apk\"]",
            ...
        }
    }
}
```

如果您願意，您可以使用應用程式名稱明確指定應用程式的功能（分別使用 Android 和 iOS 的 appium:appPackage或 appium:bundleId功能）。

如果您要測試 Web 應用程式，請指定 Appium 工作階段建立請求browserName的功能。Chrome 瀏覽器適用於所有 Android 裝置，瀏覽器Safari適用於所有 iOS 裝置。

Device Farm 不支援在遠端存取工作階段appium:app期間在 中傳遞遠端 URL 或本機檔案系統路徑。將應用程式上傳至 Device Farm，並改為將其包含在工作階段中。

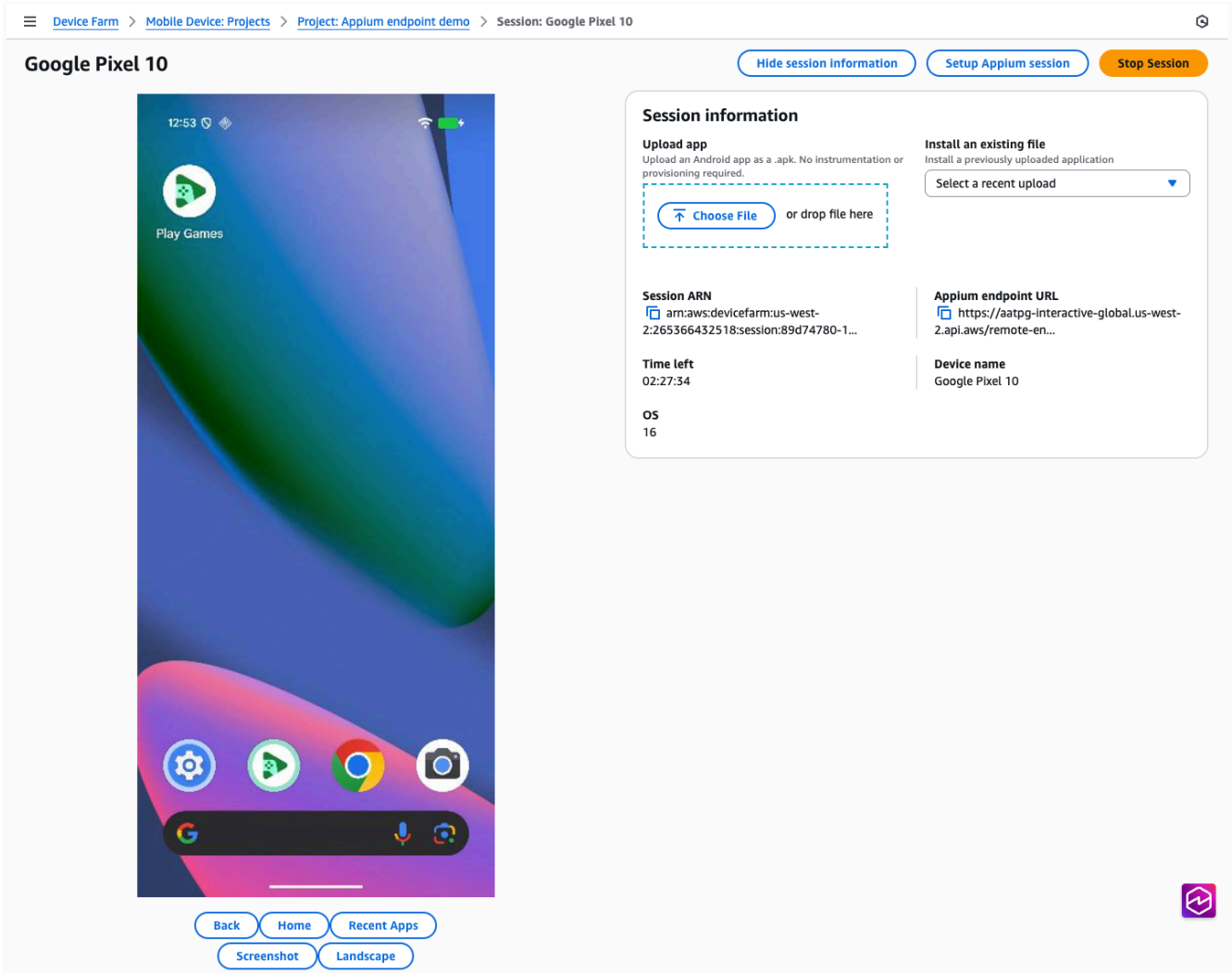
> **Note**
>
> 如需在遠端存取工作階段中自動上傳應用程式的詳細資訊，請參閱自動上傳應用程式。

如何使用 Appium 端點

以下是從主控台 AWS CLI、 和 AWS SDKs 存取工作階段 Appium 端點的步驟。這些步驟包括如何使用各種 Appium 用戶端測試架構開始執行測試：

Console

1. 在 Web 瀏覽器中開啟遠端存取工作階段頁面：



2. 若要使用 Appium Inspector 執行工作階段，請執行下列動作：

a. 按一下按鈕設定 Appium 工作階段

b. 遵循頁面上的指示，了解如何使用 Appium Inspector 啟動工作階段。

3. 若要從本機 IDE 執行 Appium 測試，請執行下列動作：

a. 按一下文字 Appium 端點 URL 旁的「複製」圖示

b. 將此 URL 貼到您的本機 Appium 程式碼，只要您目前指定遠端地址或命令執行器。如需特定語言的範例，請按一下此範例視窗中您所選語言的其中一個標籤。

AWS CLI

首先，下載並安裝最新版本，以確認您的 AWS CLI 版本是up-to-date。

> **Important**
>
> Appium 端點欄位不適用於舊版的 AWS CLI。

一旦您的工作階段啟動並執行，Appium 端點 URL 將透過 `GetRemoteAccessSession` API 呼叫回應`remoteDriverEndpoint`中名為 的欄位提供：

```
$ aws devicefarm get-remote-access-session \
    --arn "arn:aws:devicefarm:us-west-2:123456789876:session:abcdef123456-1234-5678-abcd-abcdef123456/abcdef123456-1234-5678-abcd-abcdef123456/00000"
```

這會顯示如下所示的輸出：

```
{
    "remoteAccessSession": {
        "arn": "arn:aws:devicefarm:us-west-2:111122223333:session:abcdef123456-1234-5678-abcd-abcdef123456/abcdef123456-1234-5678-abcd-abcdef123456/00000",
        "name": "Google Pixel 8",
        "status": "RUNNING",
        "endpoints": {
            "remoteDriverEndpoint": "https://devicefarm-interactive-global.us-west-2.api.aws/remote-endpoint/ABCD1234...",
        ...
}
```

無論您目前在何處指定遠端地址或命令執行器，都可以在本機 Appium 程式碼中使用此 URL。如需特定語言的範例，請按一下此範例視窗中您所選語言的其中一個標籤。

如需如何直接從命令列與端點互動的範例，您可以使用命令列工具 curl 直接呼叫 WebDriver 端點：

```
$ curl "https://devicefarm-interactive-global.us-west-2.api.aws/remote-endpoint/
ABCD1234.../status"
```

這會顯示如下所示的輸出：

```
{
    "value":
    {
        "ready": true,
        "message": "The server is ready to accept new connections",
        "build":
        {
            "version": "2.5.1"
        }
    }
}
```

Python

一旦您的工作階段啟動並執行，Appium 端點 URL 將透過 GetRemoteAccessSession API 呼叫回應remoteDriverEndpoint中名為 的欄位提供：

```
# To get the URL
import sys
import boto3
from botocore.exceptions import ClientError

def get_appium_endpoint() -> str:
    session_arn = "arn:aws:devicefarm:us-
west-2:111122223333:session:abcdef123456-1234-5678-abcd-abcdef123456/
abcdef123456-1234-5678-abcd-abcdef123456/00000"
    device_farm_client = boto3.client("devicefarm", region_name="us-west-2")

    try:
        resp = device_farm_client.get_remote_access_session(arn=session_arn)
    except ClientError as exc:
        sys.exit(f"Failed to call Device Farm: {exc}")
```

```
    remote_access_session = resp.get("remoteAccessSession", {})
    endpoints = remote_access_session.get("endpoints", {})
    endpoint = endpoints.get("remoteDriverEndpoint")

    if not endpoint:
        sys.exit("Device Farm response did not include
 endpoints.remoteDriverEndpoint")

    return endpoint

# To use the URL
from appium import webdriver
from appium.options.android import UiAutomator2Options

opts = UiAutomator2Options()
driver = webdriver.Remote(get_appium_endpoint(), options=opts)
# ...
driver.quit()
```

Java

注意：此範例使用適用於 Java v2 的 AWS 開發套件，並與 JDK 第 11 版及更新版本相容。

一旦您的工作階段啟動並執行，Appium 端點 URL 將透過 GetRemoteAccessSession API 呼叫回應`remoteDriverEndpoint`中名為 的欄位提供：

```
// To get the URL
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.devicefarm.DeviceFarmClient;
import
 software.amazon.awssdk.services.devicefarm.model.GetRemoteAccessSessionRequest;
import
 software.amazon.awssdk.services.devicefarm.model.GetRemoteAccessSessionResponse;

public class AppiumEndpointBuilder {
    public static String getAppiumEndpoint() throws Exception {
        String session_arn = "arn:aws:devicefarm:us-
west-2:111122223333:session:abcdef123456-1234-5678-abcd-abcdef123456/
abcdef123456-1234-5678-abcd-abcdef123456/00000";

        try (DeviceFarmClient client = DeviceFarmClient.builder()
                .region(Region.US_WEST_2)
                .credentialsProvider(DefaultCredentialsProvider.create())
```

```
                .build()) {

            GetRemoteAccessSessionResponse resp = client.getRemoteAccessSession(
                    GetRemoteAccessSessionRequest.builder().arn(session_arn).build()
            );

            String endpoint =
 resp.remoteAccessSession().endpoints().remoteDriverEndpoint();
            if (endpoint == null || endpoint.isEmpty()) {
                throw new IllegalStateException("remoteDriverEndpoint missing from
 response");
            }
            return endpoint;
        }
    }
}

// To use the URL
import io.appium.java_client.android.AndroidDriver;
import io.appium.java_client.android.options.UiAutomator2Options;

import java.net.URL;

public class ExampleTest {
    public static void main(String[] args) throws Exception {
        String endpoint = AppiumEndpointBuilder.getAppiumEndpoint();
        UiAutomator2Options options = new UiAutomator2Options();
        AndroidDriver driver = new AndroidDriver(new URL(endpoint), options);

        try {
            // ... your test ...
        } finally {
            driver.quit();
        }
    }
}
```

JavaScript

注意：此範例使用適用於 JavaScript v3 的 AWS SDK，以及使用 Node 18+ 的 WebdriverIO v8+。

一旦您的工作階段啟動並執行，Appium 端點 URL 將透過 GetRemoteAccessSession API 呼叫回應`remoteDriverEndpoint`中名為 的欄位提供：

```
// To get the URL
import { DeviceFarmClient, GetRemoteAccessSessionCommand } from "@aws-sdk/client-
device-farm";

export async function getAppiumEndpoint() {
  const sessionArn = "arn:aws:devicefarm:us-
west-2:111122223333:session:abcdef123456-1234-5678-abcd-abcdef123456/
abcdef123456-1234-5678-abcd-abcdef123456/00000";

  const client = new DeviceFarmClient({ region: "us-west-2" });
  const resp = await client.send(new GetRemoteAccessSessionCommand({ arn:
 sessionArn }));

  const endpoint = resp?.remoteAccessSession?.endpoints?.remoteDriverEndpoint;
  if (!endpoint) throw new Error("remoteDriverEndpoint missing from response");
  return endpoint;
}

// To use the URL with WebdriverIO
import { remote } from "webdriverio";

(async () => {
  const endpoint = await getAppiumEndpoint();
  const u = new URL(endpoint);

  const driver = await remote({
    protocol: u.protocol.replace(":", ""),
    hostname: u.hostname,
    port: u.port ? Number(u.port) : (u.protocol === "https:" ? 443 : 80),
    path: u.pathname + u.search,
    capabilities: {
      platformName: "Android",
      "appium:automationName": "UiAutomator2",
      // ...other caps...
    },
  });

  try {
    // ... your test ...
  } finally {
    await driver.deleteSession();
  }
})();
```

C#

一旦您的工作階段啟動並執行，Appium 端點 URL 將透過 GetRemoteAccessSession API 呼叫回應remoteDriverEndpoint中名為 的欄位提供：

```
// To get the URL
using System;
using System.Threading.Tasks;
using Amazon;
using Amazon.DeviceFarm;
using Amazon.DeviceFarm.Model;

public static class AppiumEndpointBuilder
{
    public static async Task<string> GetAppiumEndpointAsync()
    {
        var sessionArn = "arn:aws:devicefarm:us-
west-2:111122223333:session:abcdef123456-1234-5678-abcd-abcdef123456/
abcdef123456-1234-5678-abcd-abcdef123456/00000";

        var config = new AmazonDeviceFarmConfig
        {
            RegionEndpoint = RegionEndpoint.USWest2
        };
        using var client = new AmazonDeviceFarmClient(config);

        var resp = await client.GetRemoteAccessSessionAsync(new
 GetRemoteAccessSessionRequest { Arn = sessionArn });
        var endpoint = resp?.RemoteAccessSession?.Endpoints?.RemoteDriverEndpoint;

        if (string.IsNullOrWhiteSpace(endpoint))
            throw new InvalidOperationException("RemoteDriverEndpoint missing from
 response");

        return endpoint;
    }
}

// To use the URL
using OpenQA.Selenium.Appium;
using OpenQA.Selenium.Appium.Android;

class Example
{
```

```
    static async Task Main()
    {
        var endpoint = await AppiumEndpointBuilder.GetAppiumEndpointAsync();

        var options = new AppiumOptions();
        options.PlatformName = "Android";
        options.AutomationName = "UiAutomator2";

        using var driver = new AndroidDriver(new Uri(endpoint), options);
        try
        {
            // ... your test ...
        }
        finally
        {
            driver.Quit();
        }
    }
}
```

Ruby

一旦您的工作階段啟動並執行，Appium 端點 URL 將透過 GetRemoteAccessSession API 呼叫回應remoteDriverEndpoint中名為 的欄位提供：

```
# To get the URL
require 'aws-sdk-devicefarm'

def get_appium_endpoint
  session_arn = "arn:aws:devicefarm:us-
west-2:111122223333:session:abcdef123456-1234-5678-abcd-abcdef123456/
abcdef123456-1234-5678-abcd-abcdef123456/00000"

  client = Aws::DeviceFarm::Client.new(region: 'us-west-2')
  resp = client.get_remote_access_session(arn: session_arn)
  endpoint = resp.remote_access_session.endpoints.remote_driver_endpoint
  raise "remote_driver_endpoint missing from response" if endpoint.nil? ||
 endpoint.empty?
  endpoint
end

# To use the URL
require 'appium_lib_core'
```

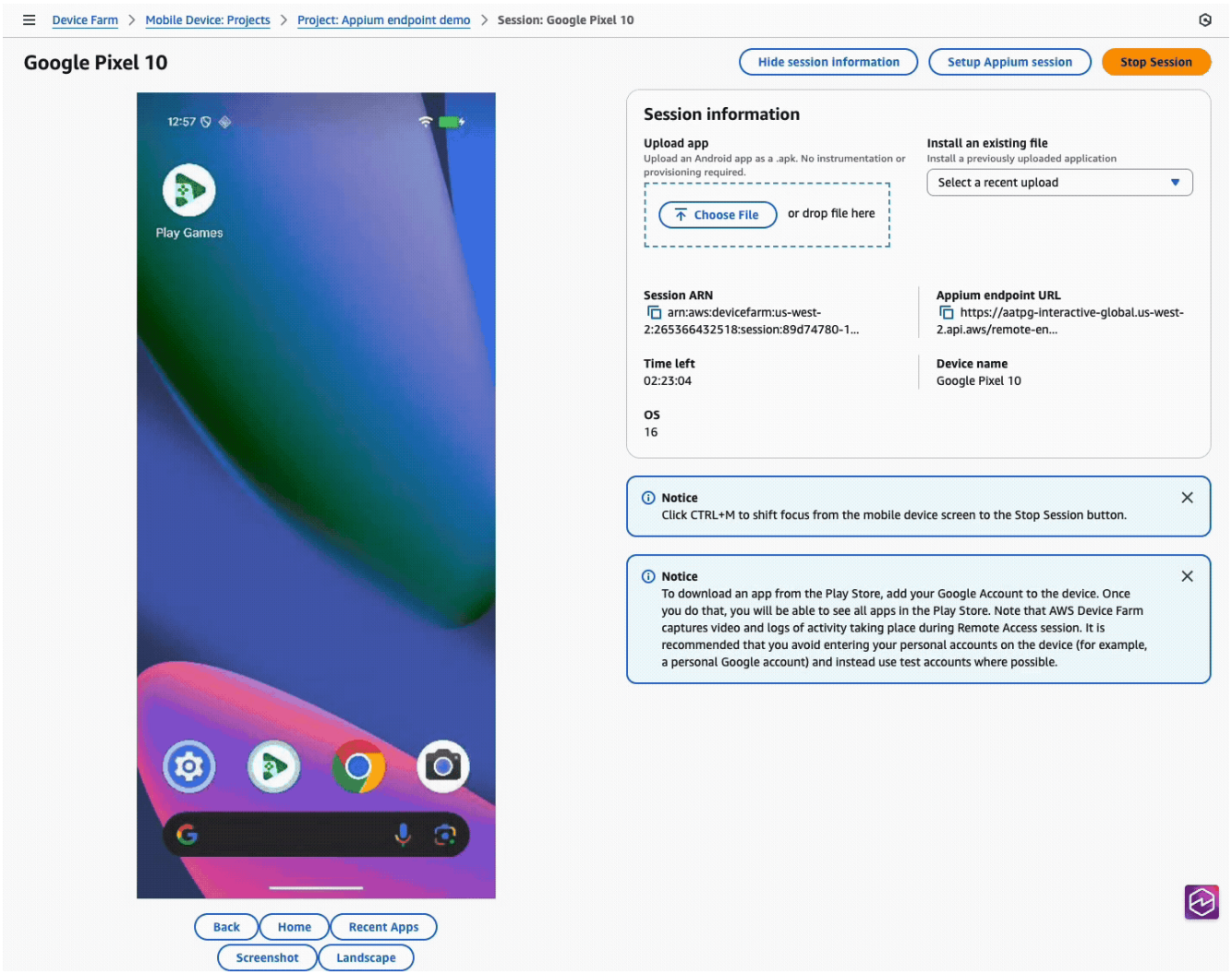
```
endpoint = get_appium_endpoint
opts = {
  server_url: endpoint,
  capabilities: {
    'platformName' => 'Android',
    'appium:automationName' => 'UiAutomator2'
  }
}

driver = Appium::Core.for(opts).start_driver
begin
  # ... your test ...
ensure
  driver.quit
end
```

檢閱您的 Appium 伺服器日誌

啟動 Appium 工作階段後，您可以在 Device Farm 主控台中即時檢視 Appium 伺服器日誌，或在遠端存取工作階段結束後下載日誌。以下是執行此作業的指示：

Console

1. 在 Device Farm 主控台中，開啟裝置的遠端存取工作階段。
2. 從本機 IDE 或 Appium Inspector 使用裝置啟動 Appium 端點工作階段
3. 然後，Appium 伺服器日誌將與裝置一起顯示在遠端存取工作階段頁面中，並在裝置下方的頁面底部提供「工作階段資訊」：



AWS CLI

注意：此範例使用命令列工具*curl*從 Device Farm 提取日誌。

在工作階段期間或之後，您可以使用 Device Farm 的 `ListArtifacts` API 下載 Appium 伺服器日誌。

```
$ aws devicefarm list-artifacts \
  --type FILE \
  --arn arn:aws:devicefarm:us-
west-2:111122223333:session:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-45678
```

這會在工作階段期間顯示如下的輸出：

```
{
```

```
    "artifacts": [
        {
            "arn": "arn:aws:devicefarm:us-
west-2:111122223333:artifact:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-4567
            "name": "AppiumServerLogOutput",
            "type": "APPIUM_SERVER_LOG_OUTPUT",
            "extension": "",
            "url": "https://prod-us-west-2-results.s3.dualstack.us-
west-2.amazonaws.com/111122223333/12345678..."
        }
    ]
}
```

工作階段完成後，還有以下項目：

```
{
    "artifacts": [
        {
            "arn": "arn:aws:devicefarm:us-
west-2:111122223333:artifact:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-4567
            "name": "Appium Server Output",
            "type": "APPIUM_SERVER_OUTPUT",
            "extension": "log",
            "url": "https://prod-us-west-2-results.s3.dualstack.us-
west-2.amazonaws.com/111122223333/12345678..."
        }
    ]
}
```

```
$ curl "https://prod-us-west-2-results.s3.dualstack.us-
west-2.amazonaws.com/111122223333/12345678..."
```

這會顯示如下所示的輸出：

```
info Appium Welcome to Appium v2.5.4
info Appium Non-default server args:
info Appium { address: '127.0.0.1',
info Appium   allowInsecure:
info Appium    [ 'execute_driver_script',
info Appium      'session_discovery',
info Appium      'perf_record',
info Appium      'adb_shell',
```

```
info Appium      'chromedriver_autodownload',
info Appium      'get_server_logs' ],
info Appium   keepAliveTimeout: 0,
info Appium   logNoColors: true,
info Appium   logTimestamp: true,
info Appium   longStacktrace: true,
info Appium   sessionOverride: true,
info Appium   strictCaps: true,
info Appium   useDrivers: [ 'uiautomator' ] }
```

Python

注意：此範例使用第三方*requests*套件來下載日誌，以及適用於 Python 的 AWS SDK*boto3*。

在工作階段期間或之後，您可以使用 Device Farm 的 ListArtifacts API 擷取 Appium 伺服器日誌 URL，然後將其下載。

```
import pathlib
import requests
import boto3

def download_appium_log():
    session_arn = "arn:aws:devicefarm:us-
west-2:111122223333:session:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-45678
    client = boto3.client("devicefarm", region_name="us-west-2")

    # 1) List artifacts for the session (FILE artifacts), handling pagination
    artifacts = []
    token = None
    while True:
        kwargs = {"arn": session_arn, "type": "FILE"}
        if token:
            kwargs["nextToken"] = token
        resp = client.list_artifacts(**kwargs)
        artifacts.extend(resp.get("artifacts", []))
        token = resp.get("nextToken")
        if not token:
            break

    if not artifacts:
        raise RuntimeError("No artifacts found in this session")

    # Filter strictly to Appium server logs
    allowed = {"APPIUM_SERVER_OUTPUT", "APPIUM_SERVER_LOG_OUTPUT"}
```

```
    filtered = [a for a in artifacts if a.get("type") in allowed]
    if not filtered:
        raise RuntimeError("No Appium server log artifacts found (expected
 APPIUM_SERVER_OUTPUT or APPIUM_SERVER_LOG_OUTPUT)")

    # Prefer the final 'OUTPUT' log, else the live 'LOG_OUTPUT'
    chosen = (next((a for a in filtered if a.get("type") == "APPIUM_SERVER_OUTPUT"),
 None)
              or next((a for a in filtered if a.get("type") ==
 "APPIUM_SERVER_LOG_OUTPUT"), None))

    url = chosen["url"]
    ext = chosen.get("extension") or "log"
    out = pathlib.Path(f"./appium_server_log.{ext}")

    # 2) Download the artifact
    with requests.get(url, stream=True) as r:
        r.raise_for_status()
        with open(out, "wb") as fh:
            for chunk in r.iter_content(chunk_size=1024 * 1024):
                if chunk:
                    fh.write(chunk)

    print(f"Saved Appium server log to: {out.resolve()}")

download_appium_log()
```

這會顯示如下所示的輸出：

```
info Appium Welcome to Appium v2.5.4
info Appium Non-default server args:
info Appium { address: '127.0.0.1', allowInsecure: [ 'execute_driver_script', ... ],
 useDrivers: [ 'uiautomator' ] }
```

Java

注意：此範例使用適用於 Java v2 的 AWS SDK 和 *HttpClient* 來下載日誌，並與 JDK 第 11 版及更高版本相容。

在工作階段期間或之後，您可以使用 Device Farm 的 ListArtifacts API 擷取 Appium 伺服器日誌 URL，然後將其下載。

```
import java.io.IOException;
```

```
import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.nio.file.Path;
import java.time.Duration;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.devicefarm.DeviceFarmClient;
import software.amazon.awssdk.services.devicefarm.model.Artifact;
import software.amazon.awssdk.services.devicefarm.model.ArtifactCategory;
import software.amazon.awssdk.services.devicefarm.model.ListArtifactsRequest;
import software.amazon.awssdk.services.devicefarm.model.ListArtifactsResponse;

public class AppiumLogDownloader {

    public static void main(String[] args) throws Exception {
        String sessionArn = "arn:aws:devicefarm:us-
west-2:111122223333:session:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-45678

        try (DeviceFarmClient client = DeviceFarmClient.builder()
                .region(Region.US_WEST_2)
                .build()) {

            // 1) List artifacts for the session (FILE artifacts) with pagination
            List<Artifact> all = new ArrayList<>();
            String token = null;
            do {
                ListArtifactsRequest.Builder b = ListArtifactsRequest.builder()
                        .arn(sessionArn)
                        .type(ArtifactCategory.FILE);
                if (token != null) b.nextToken(token);
                ListArtifactsResponse page = client.listArtifacts(b.build());
                all.addAll(page.artifacts());
                token = page.nextToken();
            } while (token != null && !token.isBlank());

            // Filter strictly to Appium logs
            List<Artifact> filtered = all.stream()
                    .filter(a -> {
                        String t = a.typeAsString();
```

```
                        return "APPIUM_SERVER_OUTPUT".equals(t) ||
"APPIUM_SERVER_LOG_OUTPUT".equals(t);
                    })
                    .toList();

            if (filtered.isEmpty()) {
                throw new RuntimeException("No Appium server log artifacts found
(expected APPIUM_SERVER_OUTPUT or APPIUM_SERVER_LOG_OUTPUT).");
            }

            // Prefer OUTPUT; else LOG_OUTPUT
            Artifact chosen = filtered.stream()
                    .filter(a -> "APPIUM_SERVER_OUTPUT".equals(a.typeAsString()))
                    .findFirst()
                    .orElseGet(() -> filtered.stream()
                            .filter(a ->
"APPIUM_SERVER_LOG_OUTPUT".equals(a.typeAsString()))
                            .findFirst()
                            .get());

            String url = chosen.url();
            String ext = (chosen.extension() == null ||
chosen.extension().isBlank()) ? "log" : chosen.extension();
            Path out = Path.of("appium_server_log." + ext);

            // 2) Download the artifact with HttpClient
            HttpClient http = HttpClient.newBuilder()
                    .connectTimeout(Duration.ofSeconds(10))
                    .build();

            HttpRequest get = HttpRequest.newBuilder(URI.create(url))
                    .timeout(Duration.ofMinutes(5))
                    .GET()
                    .build();

            HttpResponse<Path> resp = http.send(get,
HttpResponse.BodyHandlers.ofFile(out));
            if (resp.statusCode() / 100 != 2) {
                throw new IOException("Failed to download log, HTTP " +
resp.statusCode());
            }
            System.out.println("Saved Appium server log to: " +
out.toAbsolutePath());
        }
```

```
    }
}
```

這會顯示如下所示的輸出：

```
info Appium Welcome to Appium v2.5.4
info Appium Non-default server args:
info Appium { address: '127.0.0.1', ..., useDrivers: [ 'uiautomator' ] }
```

JavaScript

注意：此範例使用適用於 JavaScript 的 AWS SDK (v3) 和 Node 18+ *fetch*來下載日誌。

在工作階段期間或之後，您可以使用 Device Farm 的 ListArtifacts API 來擷取 Appium 伺服器日誌 URL，然後將其下載。

```
import { DeviceFarmClient, ListArtifactsCommand } from "@aws-sdk/client-device-
farm";
import { createWriteStream } from "fs";
import { pipeline } from "stream";
import { promisify } from "util";

const pipe = promisify(pipeline);
const client = new DeviceFarmClient({ region: "us-west-2" });

const sessionArn = "arn:aws:devicefarm:us-
west-2:111122223333:session:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-45678

// 1) List artifacts for the session (FILE artifacts), handling pagination
const artifacts = [];
let nextToken;
do {
  const page = await client.send(new ListArtifactsCommand({
    arn: sessionArn,
    type: "FILE",
    nextToken
  }));
  artifacts.push(...(page.artifacts ?? []));
  nextToken = page.nextToken;
} while (nextToken);

if (!artifacts.length) throw new Error("No artifacts found");
```

```
// Strict filter to Appium logs
const filtered = (artifacts ?? []).filter(a =>
  a.type === "APPIUM_SERVER_OUTPUT" || a.type === "APPIUM_SERVER_LOG_OUTPUT"
);
if (!filtered.length) {
  throw new Error("No Appium server log artifacts found (expected
 APPIUM_SERVER_OUTPUT or APPIUM_SERVER_LOG_OUTPUT).");
}

// Prefer OUTPUT; else LOG_OUTPUT
const chosen =
  filtered.find(a => a.type === "APPIUM_SERVER_OUTPUT") ??
  filtered.find(a => a.type === "APPIUM_SERVER_LOG_OUTPUT");

const url = chosen.url;
const ext = chosen.extension || "log";
const outPath = `./appium_server_log.${ext}`;

// 2) Download the artifact
const resp = await fetch(url);
if (!resp.ok) {
  throw new Error(`Failed to download log: ${resp.status} ${await
 resp.text().catch(()=>"")}`);
}
await pipe(resp.body, createWriteStream(outPath));
console.log("Saved Appium server log to:", outPath);
```

這會顯示如下所示的輸出：

```
info Appium Welcome to Appium v2.5.4
info Appium Non-default server args:
info Appium { address: '127.0.0.1', allowInsecure: [ 'execute_driver_script', ... ],
 useDrivers: [ 'uiautomator' ] }
```

C#

注意：此範例使用適用於 .NET 的 AWS SDK 和 *HttpClient*下載日誌。

在工作階段期間或之後，您可以使用 Device Farm 的 ListArtifacts API 擷取 Appium 伺服器日誌 URL，然後將其下載。

```
using System;
using System.Collections.Generic;
```

```
using System.IO;
using System.Net.Http;
using System.Threading.Tasks;
using System.Linq;
using Amazon;
using Amazon.DeviceFarm;
using Amazon.DeviceFarm.Model;

class AppiumLogDownloader
{
    static async Task Main()
    {
        var sessionArn = "arn:aws:devicefarm:us-
west-2:111122223333:session:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-45678

        using var client = new AmazonDeviceFarmClient(RegionEndpoint.USWest2);

        // 1) List artifacts for the session (FILE artifacts), handling pagination
        var all = new List<Artifact>();
        string? token = null;
        do
        {
            var page = await client.ListArtifactsAsync(new ListArtifactsRequest
            {
                Arn = sessionArn,
                Type = ArtifactCategory.FILE,
                NextToken = token
            });
            if (page.Artifacts != null) all.AddRange(page.Artifacts);
            token = page.NextToken;
        } while (!string.IsNullOrEmpty(token));

        if (all.Count == 0)
            throw new Exception("No artifacts found");

        // Strict filter to Appium logs
        var filtered = all.Where(a =>
            a.Type == "APPIUM_SERVER_OUTPUT" || a.Type ==
 "APPIUM_SERVER_LOG_OUTPUT").ToList();

        if (filtered.Count == 0)
            throw new Exception("No Appium server log artifacts found (expected
 APPIUM_SERVER_OUTPUT or APPIUM_SERVER_LOG_OUTPUT).");
```

```
        // Prefer OUTPUT; else LOG_OUTPUT
        var chosen = filtered.FirstOrDefault(a => a.Type == "APPIUM_SERVER_OUTPUT")
                    ?? filtered.First(a => a.Type == "APPIUM_SERVER_LOG_OUTPUT");

        var url = chosen.Url;
        var ext = string.IsNullOrWhiteSpace(chosen.Extension) ? "log" :
 chosen.Extension;
        var outPath = $"./appium_server_log.{ext}";

        // 2) Download the artifact
        using var http = new HttpClient();
        using var resp = await http.GetAsync(url,
 HttpCompletionOption.ResponseHeadersRead);
        resp.EnsureSuccessStatusCode();
        await using (var fs = File.Create(outPath))
        {
            await resp.Content.CopyToAsync(fs);
        }
        Console.WriteLine($"Saved Appium server log to:
 {Path.GetFullPath(outPath)}");
    }
}
```

這會顯示如下所示的輸出：

```
info Appium Welcome to Appium v2.5.4
info Appium Non-default server args:
info Appium { address: '127.0.0.1', ..., useDrivers: [ 'uiautomator' ] }
```

Ruby

注意：此範例使用適用於 Ruby 的 AWS SDK 和 *Net::HTTP*下載日誌。

在工作階段期間或之後，您可以使用 Device Farm 的 `ListArtifacts` API 來擷取 Appium 伺服器日誌 URL，然後將其下載。

```
require "aws-sdk-devicefarm"
require "net/http"
require "uri"

client = Aws::DeviceFarm::Client.new(region: "us-west-2")
session_arn = "arn:aws:devicefarm:us-
west-2:111122223333:session:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-45678
```

```
# 1) List artifacts for the session (FILE artifacts), handling pagination
artifacts = []
token = nil
loop do
  page = client.list_artifacts(arn: session_arn, type: "FILE", next_token: token)
  artifacts.concat(page.artifacts || [])
  token = page.next_token
  break if token.nil? || token.empty?
end

raise "No artifacts found" if artifacts.empty?

# Strict filter to Appium logs
filtered = (artifacts || []).select { |a| ["APPIUM_SERVER_OUTPUT",
 "APPIUM_SERVER_LOG_OUTPUT"].include?(a.type) }
raise "No Appium server log artifacts found (expected APPIUM_SERVER_OUTPUT or
 APPIUM_SERVER_LOG_OUTPUT)." if filtered.empty?

# Prefer OUTPUT; else LOG_OUTPUT
chosen = filtered.find { |a| a.type == "APPIUM_SERVER_OUTPUT" } ||
         filtered.find { |a| a.type == "APPIUM_SERVER_LOG_OUTPUT" }

url = chosen.url
ext = (chosen.extension && !chosen.extension.empty?) ? chosen.extension : "log"
out_path = "./appium_server_log.#{ext}"

# 2) Download the artifact
uri = URI.parse(url)
Net::HTTP.start(uri.host, uri.port, use_ssl: (uri.scheme == "https")) do |http|
  req = Net::HTTP::Get.new(uri)
  http.request(req) do |resp|
    raise "Failed GET: #{resp.code} #{resp.body}" unless resp.code.to_i / 100 == 2
    File.open(out_path, "wb") { |f| resp.read_body { |chunk| f.write(chunk) } }
  end
end
puts "Saved Appium server log to: #{File.expand_path(out_path)}"
```

這會顯示如下所示的輸出：

```
info Appium Welcome to Appium v2.5.4
info Appium Non-default server args:
```

```
info Appium { address: '127.0.0.1', allowInsecure: [ 'execute_driver_script', ... ],
 useDrivers: [ 'uiautomator' ] }
```

支援的 Appium 功能和命令

Device Farm 的 Appium 端點支援您在本機裝置上使用的大多數相同命令和所需功能，但有少數例外。下列清單顯示目前不支援的功能和命令。如果您的測試因為功能受限而無法如預期執行，請開啟支援案例以取得其他指導。

支援的功能

在 Device Farm 上建立 Appium 工作階段時，我們建議您擁有一組不同的功能，排除本機裝置特有的任何功能。在 Device Farm 上，如果設定某些不支援的功能，工作階段建立可能會失敗。這包括裝置特定的功能，例如 `udid`和 `platformVersion`。此外，不支援與 Android 上的 ChromeDriver 和 iOS 上的 WebDriverAgent 相關的特定功能，以及僅支援模擬器和模擬器的功能。

支援的命令

在實際 Android 和 iOS 裝置上正確執行的大多數 Appium 命令都會在 Device Farm 上如預期執行，但有下列排除項目：

Appium 裝置命令 (**`/appium/device`**)

- `install_app`
- `finger_print`
- `send_sms`
- `gsm_call`
- `gsm_signal`
- `gsm_voice`
- `power_ac`
- `power_capacity`
- `network_speed`
- `shake`

Appium 執行方法和指令碼 (**/execute**)

- installApp
- execEmuConsoleCommand
- fingerprint
- gsmCall
- gsmSignal
- sendSms
- gsmVoice
- powerAC
- powerCapacity
- networkSpeed
- sensorSet
- injectEmulatorCameraImage
- isGpsEnabled
- shake
- clearApp
- clearKeychains
- configureLocalization
- enrollBiometric
- getPasteboard
- installXCTestBundle
- listXCTestBundles
- listXCTestsInTestBundle
- runXCTest
- sendBiometricMatch
- setPasteboard
- setPermission
- startAudioRecording
- startLogsBroadcast
- startRecordingScreen

- startScreenStreaming
- startXCTestScreenRecording
- stopAudioRecording
- stopLogsBroadcast
- stopRecordingScreen
- stopScreenStreaming
- stopXCTestScreenRecording
- updateSafariPreferences

AWS Device Farm 中的私有裝置

私有裝置是 AWS Device Farm 在 Amazon 資料中心代表您部署的實體行動裝置。此裝置專屬於 AWS 您的帳戶。

> **Note**
>
> 目前，私有裝置僅適用於 AWS 美國西部（奧勒岡）區域 ()`us-west-2`。

如果您有私有裝置機群，您可以使用您的私有裝置來建立遠端存取工作階段並排程測試執行。如需詳細資訊，請參閱在 AWS Device Farm 中建立測試執行或啟動遠端存取工作階段。您還可以建立執行個體描述檔，以在遠端存取工作階段或測試執行期間控制私有裝置的行為。如需詳細資訊，請參閱在 AWS Device Farm 中建立執行個體描述檔。或者，您可以請求將特定 Android 私有裝置部署為根裝置。

您也可以建立 Amazon Virtual Private Cloud 端點服務，以測試您公司可存取但無法透過網際網路連線的私有應用程式。例如，您可能會在 VPC 中執行要在行動裝置上測試的 Web 應用程式。如需詳細資訊，請參閱搭配使用 Amazon VPC 端點服務與 Device Farm - Legacy（不建議）。

如果您有興趣使用私有裝置的機群，請聯絡我們。Device Farm 團隊必須與您合作，為您的 AWS 帳戶設定和部署私有裝置的機群。

主題

- 在 AWS Device Farm 中建立執行個體描述檔
- 在 AWS Device Farm 中請求其他私有裝置
- 在 AWS Device Farm 中建立測試執行或啟動遠端存取工作階段
- 在 AWS Device Farm 的裝置集區中選取私有裝置
- 在 AWS Device Farm 中的私有裝置上略過應用程式重新簽署
- AWS Device Farm 中跨 AWS 區域的 Amazon VPC
- 終止 Device Farm 中的私有裝置

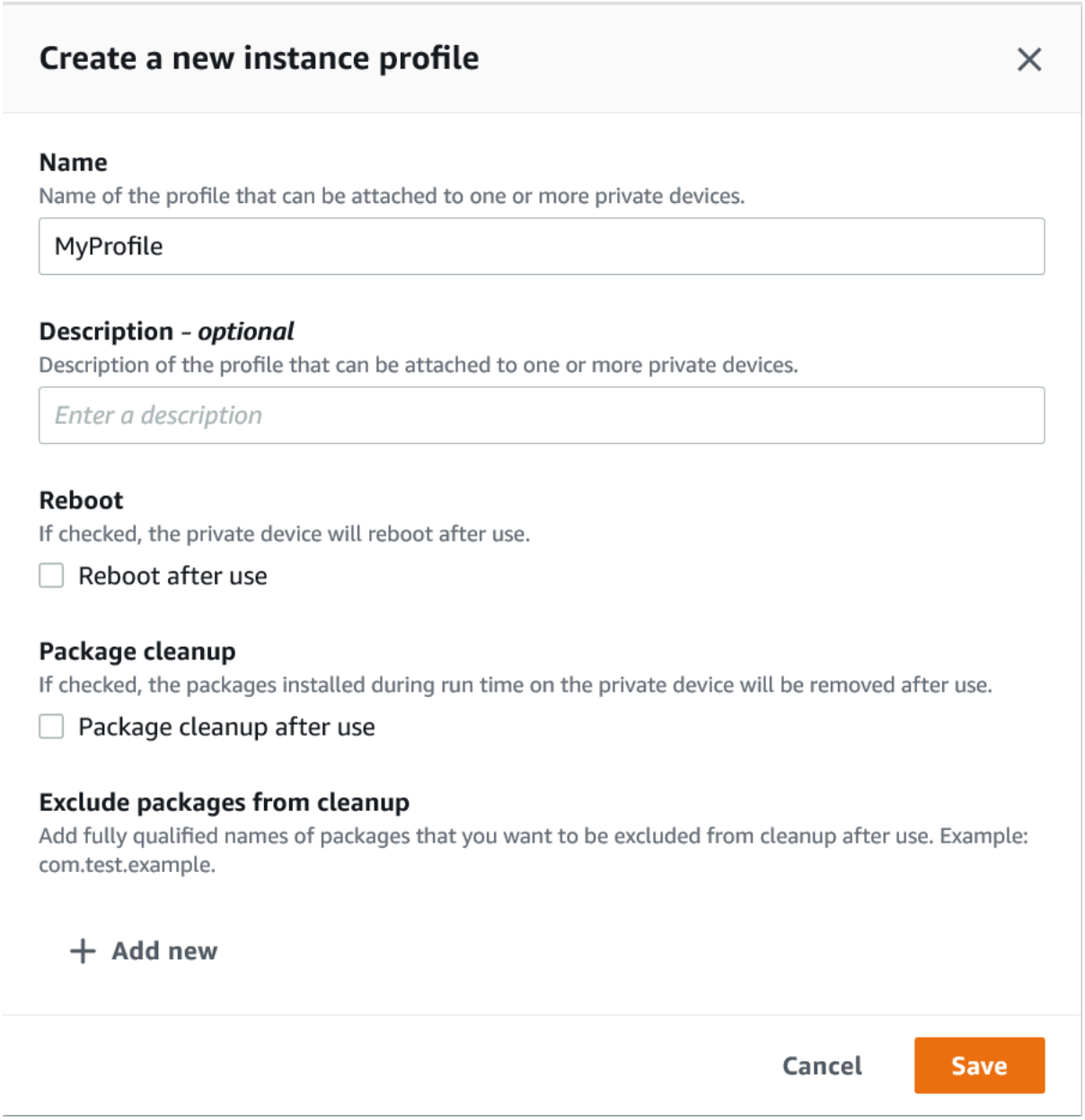
在 AWS Device Farm 中建立執行個體描述檔

您可以設定機群，其中包含一或多個私有裝置。這些裝置為您的 AWS 帳戶專用。設定裝置之後，您可以選擇性地為裝置建立一或多個執行個體描述檔。執行個體描述檔可協助您將測試執行自動化，並一致

地對裝置執行個體套用相同的設定。執行個體描述檔也可以協助您控制遠端存取工作階段的行為。如需 Device Farm 中私有裝置的詳細資訊，請參閱 AWS Device Farm 中的私有裝置。

若要建立 執行個體

1. 在 https：//https://console.aws.amazon.com/devicefarm/ 開啟 Device Farm 主控台。
2. 在 Device Farm 導覽面板上，選擇行動裝置測試，然後選擇私有裝置。
3. 選擇執行個體設定檔。
4. 選擇建立執行個體設定檔。
5. 輸入執行個體描述檔的名稱。



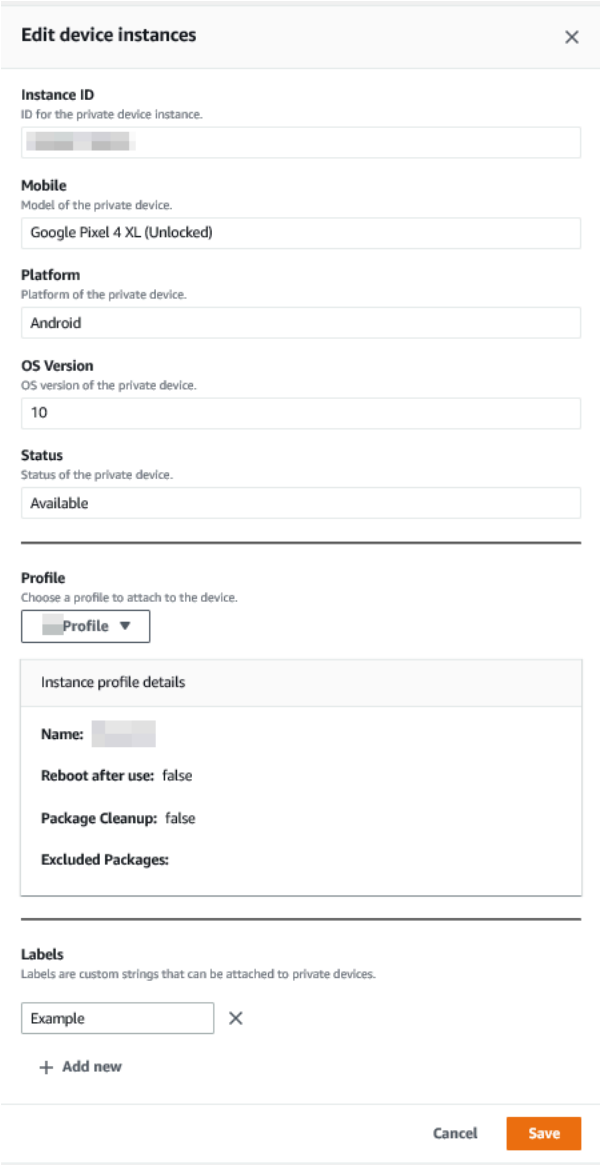
6. (選用) 輸入執行個體描述檔的描述。
7. （選用）變更下列任何設定，以指定您希望 Device Farm 在每次測試執行或工作階段結束後對裝置採取的動作：
   - 使用後重新啟動 – 若要重新啟動裝置，請選取此核取方塊。根據預設，會清除此核取方塊 (`false`)。
   - 套件清除 – 若要移除您在裝置上安裝的所有應用程式套件，請選取此核取方塊。根據預設，會清除此核取方塊 (`false`)。若要保留您在裝置上安裝的所有應用程式套件，請將此核取方塊保留為未選取。
   - 從清除中排除套件 – 若要僅在裝置上保留選取的應用程式套件，請選取套件清除核取方塊，然後選擇新增。對於套件名稱，輸入您要保留在裝置上之應用程式套件的完整名稱 (例如，`com.test.example`)。若要在裝置上保留更多應用程式套件，請選擇 Add new (新增)，然後輸入每個套件的完整名稱。
8. 選擇儲存。

在 AWS Device Farm 中請求其他私有裝置

在 AWS Device Farm 中，您可以請求將額外的私有裝置執行個體新增至您的機群。您也可以檢視和變更機群中現有私有裝置執行個體的設定。如需私有裝置的詳細資訊，請參閱 AWS Device Farm 中的私有裝置。

請求其他私有裝置或變更其設定

1. 在 https：//https://console.aws.amazon.com/devicefarm/ 開啟 Device Farm 主控台。
2. 在 Device Farm 導覽面板上，選擇行動裝置測試，然後選擇私有裝置。
3. 選擇 Device instances (裝置執行個體)。Device instances (裝置執行個體) 標籤會顯示您的機群中私有裝置的表格。若要快速搜尋或篩選資料表，請在資料欄上方的搜尋列中輸入搜尋詞彙。
4. 若要請求新的私有裝置執行個體，請選擇請求裝置執行個體或聯絡我們。私有裝置需要在 Device Farm 團隊協助下進行額外設定。
5. 在裝置執行個體的表格中，選擇您要檢視或管理資訊的執行個體旁的切換選項，然後選擇編輯。



6. 若要將執行個體描述檔連接至裝置執行個體，請從描述檔下拉式清單中選擇它。例如，如果您想要一律從清除任務中排除特定應用程式套件，連接執行個體描述檔會很有幫助。如需搭配 裝置使用執行個體描述檔的詳細資訊，請參閱 在 AWS Device Farm 中建立執行個體描述檔。
7. (選用) 在 Labels (標籤) 中，選擇 Add new) 來新增標籤至裝置執行個體。標籤可協助您分類裝置並更輕鬆地找到特定裝置。
8. 選擇儲存。

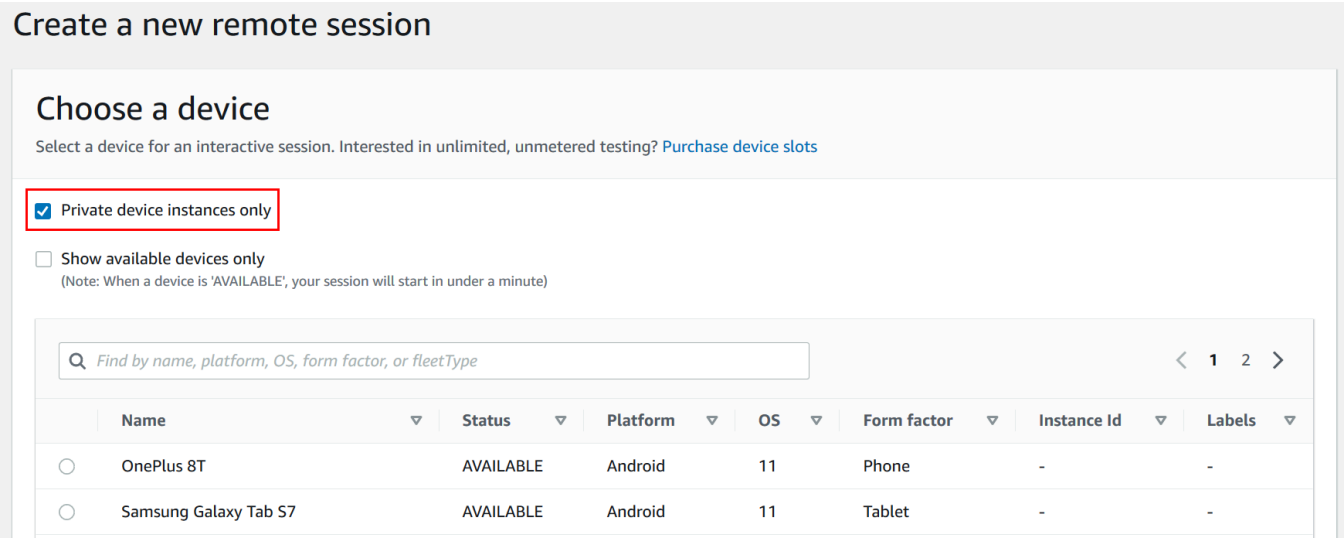
在 AWS Device Farm 中建立測試執行或啟動遠端存取工作階段

在 AWS Device Farm 中，設定私有裝置機群後，您可以使用機群中的一或多個私有裝置建立測試執行或啟動遠端存取工作階段。如需私有裝置的詳細資訊，請參閱 AWS Device Farm 中的私有裝置。

建立測試執行或啟動遠端存取工作階段

1. 在 https：//https://console.aws.amazon.com/devicefarm/ 開啟 Device Farm 主控台。
2. 在 Device Farm 導覽面板上，選擇行動裝置測試，然後選擇專案。
3. 從清單中選擇現有的專案，或建立新的專案。若要建立新專案，請選擇新增專案，輸入專案的名稱，然後選擇提交。
4. 執行以下任意一項：

   - 若要建立測試執行，請選擇 Automated tests (自動測試)，然後選擇 Create a new run (建立新執行)。精靈會逐步引導您進行建立執行的步驟。在選取裝置步驟中，您可以編輯現有的裝置集區，或建立新的裝置集區，只包含 Device Farm 團隊設定並與 AWS 您的帳戶相關聯的私有裝置。如需詳細資訊，請參閱the section called “建立私有裝置集區”。
   - 若要開始遠端存取工作階段，請選擇 Remote access (遠端存取)，然後選擇 Start a new session (開始新的工作階段)。在選擇裝置頁面上，選取私有裝置執行個體，僅將清單限制為 Device Farm團隊設定並與 AWS 您的帳戶相關聯的私有裝置。然後，選擇您要存取的裝置，輸入遠端存取工作階段的名稱，並選擇 Confirm and start session (確認並啟動工作階段)。



在 AWS Device Farm 的裝置集區中選取私有裝置

若要在測試執行中使用私有裝置，您可以建立選取私有裝置的裝置集區。裝置集區可讓您主要透過三種類型的裝置集區規則來選取私有裝置：

1. 以裝置 ARN 為基礎的規則

2. 以裝置執行個體標籤為基礎的規則
3. 以裝置執行個體 ARN 為基礎的規則

在下列各節中，每個規則類型及其使用案例都會詳細說明。您可以使用 Device Farm 主控台、 AWS 命令列界面 (AWS CLI) 或 Device Farm API，使用這些規則建立或修改具有私有裝置的裝置集區。

主題

裝置 ARN

裝置 ARN 是代表裝置類型的識別符，而不是任何特定的實體裝置執行個體。裝置類型由下列屬性定義：

- 裝置機群 ID
- 裝置 OEM
- 裝置型號
- 裝置作業系統版本
- 裝置的狀態，指出裝置是否已根目錄

許多實體裝置執行個體可由單一裝置類型表示，其中該類型的每個執行個體都有這些屬性的相同值。例如，如果您的私有機群中有三個 iOS 版本 *16.1.0* 上的 *Apple iPhone 13* 裝置，則每個裝置都會共用相同的裝置 ARN。如果從具有這些相同屬性的機群新增或移除任何裝置，裝置 ARN 會繼續代表該裝置類型機群中任何可用的裝置。

裝置 ARN 是最強大的裝置集區選擇私有裝置的方式，因為它允許裝置集區繼續選擇裝置，無論您在任何特定時間部署的特定裝置執行個體為何。個別私有裝置執行個體可能會遇到硬體故障，提示 Device Farm 自動將其取代為相同裝置類型的新工作執行個體。在這些情況下，裝置 ARN 規則可確保您的裝置集區可以在發生硬體故障時繼續選取裝置。

當您針對裝置集區中的私有裝置使用裝置 ARN 規則，並排程使用該集區的測試執行時，Device Farm 會自動檢查哪些私有裝置執行個體是由該裝置 ARN 表示。在目前可用的執行個體中，其中一個將被指派執行您的測試。如果目前沒有可用的執行個體，Device Farm 會等待該裝置 ARN 的第一個可用執行個體變成可用，並指派它來執行您的測試。

裝置執行個體標籤

裝置執行個體標籤是文字識別符，可作為裝置執行個體的中繼資料附加。您可以將多個標籤連接到每個裝置執行個體，並將相同的標籤連接到多個裝置執行個體。如需從裝置執行個體新增、修改或移除裝置標籤的詳細資訊，請參閱管理私有裝置。

裝置執行個體標籤可以是為裝置集區選取私有裝置的強大方式，因為如果您有多個裝置執行個體具有相同的標籤，則允許裝置集區從其中任一個選取以供測試。如果裝置 ARN 不是適合您使用案例的良好規則（例如，如果您想要從多個裝置類型的裝置中選取，或者如果您想要從裝置類型的所有裝置子集中選取），則裝置執行個體標籤可讓您為裝置集區選取更精細的多個裝置。個別私有裝置執行個體可能會遇到硬體故障，提示 Device Farm 自動將其取代為相同裝置類型的新工作執行個體。在這些情況下，替換裝置執行個體不會保留替換裝置的任何執行個體標籤中繼資料。因此，如果您將相同的裝置執行個體標籤套用至多個裝置執行個體，則裝置執行個體標籤規則可確保您的裝置集區可在發生硬體故障時繼續選取裝置執行個體。

當您針對裝置集區中的私有裝置使用裝置執行個體標籤規則，並排程使用該集區執行測試時，Device Farm 會自動檢查哪些私有裝置執行個體是由該裝置執行個體標籤表示，而在這些執行個體中，隨機選取一個可用於執行測試的執行個體。如果沒有可用，Device Farm 會隨機選取任何具有裝置執行個體標籤的裝置執行個體，以執行您的測試，並在測試可用時將測試排入佇列。

執行個體 ARN

裝置執行個體 ARN 是識別符，代表部署在私有機群中的實體裸機裝置執行個體。例如，如果您在私有機群的 OS *15.0.0 #### iPhone 13* 裝置，而每個裝置都會共用相同的裝置 ARN，則每個裝置也會有自己的執行個體 ARN，僅代表該執行個體。

裝置執行個體 ARN 是為裝置集區選取私有裝置最不穩健的方式，只有在裝置 ARNs 和裝置執行個體標籤不符合您的使用案例時才建議使用。裝置執行個體 ARNs 通常用作裝置集區的規則，當特定裝置執行個體以唯一且特定的方式設定為測試的先決條件，且如果在測試執行之前需要已知和驗證該組態。個別私有裝置執行個體可能會遇到硬體故障，提示 Device Farm 自動將其取代為相同裝置類型的新工作執行個體。在這些情況下，替換裝置執行個體將具有與替換裝置不同的裝置執行個體 ARN。因此，如果您依賴裝置集區的裝置執行個體 ARNs，則需要手動將裝置集區的規則定義從使用舊的 ARN 變更為使用新的 ARN。如果您需要手動預先設定裝置進行測試，則這可能是有效的工作流程（相較於裝置

ARNs)。對於大規模測試，建議嘗試調整這些使用案例以使用裝置執行個體標籤，如果可能，請預先設定多個裝置執行個體進行測試。

當您將裝置執行個體 ARN 規則用於裝置集區中的私有裝置，並使用該集區排程測試執行時，Device Farm 會自動將該測試指派給該裝置執行個體。如果該裝置執行個體無法使用，則 Device Farm 會在裝置可用時將測試排入佇列。

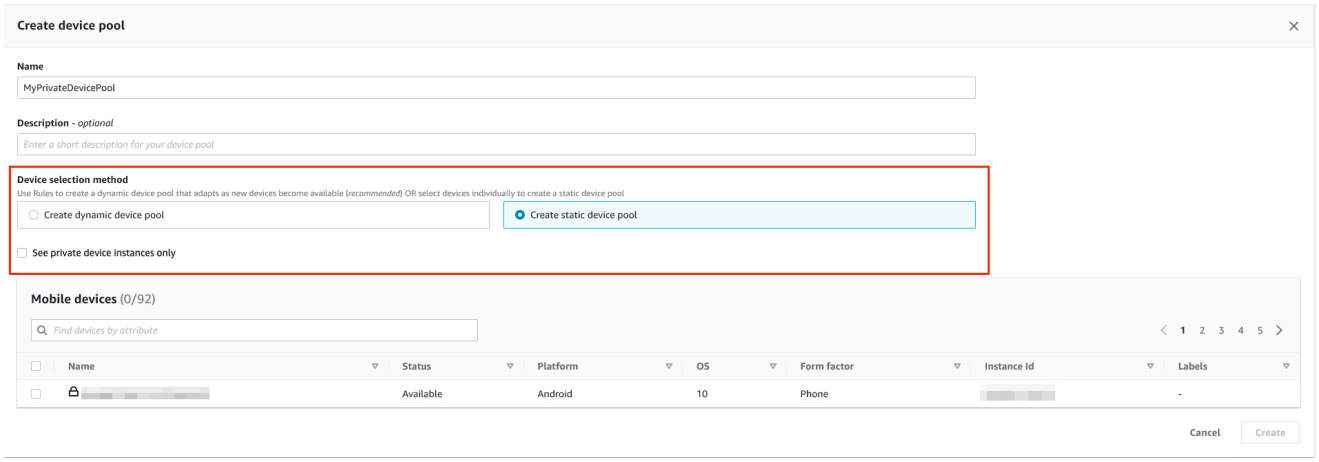
使用私有裝置建立私有裝置集區（主控台）

建立測試執行時，您可以為測試執行建立一個裝置集區，並確保該集區僅包含您的私有裝置。

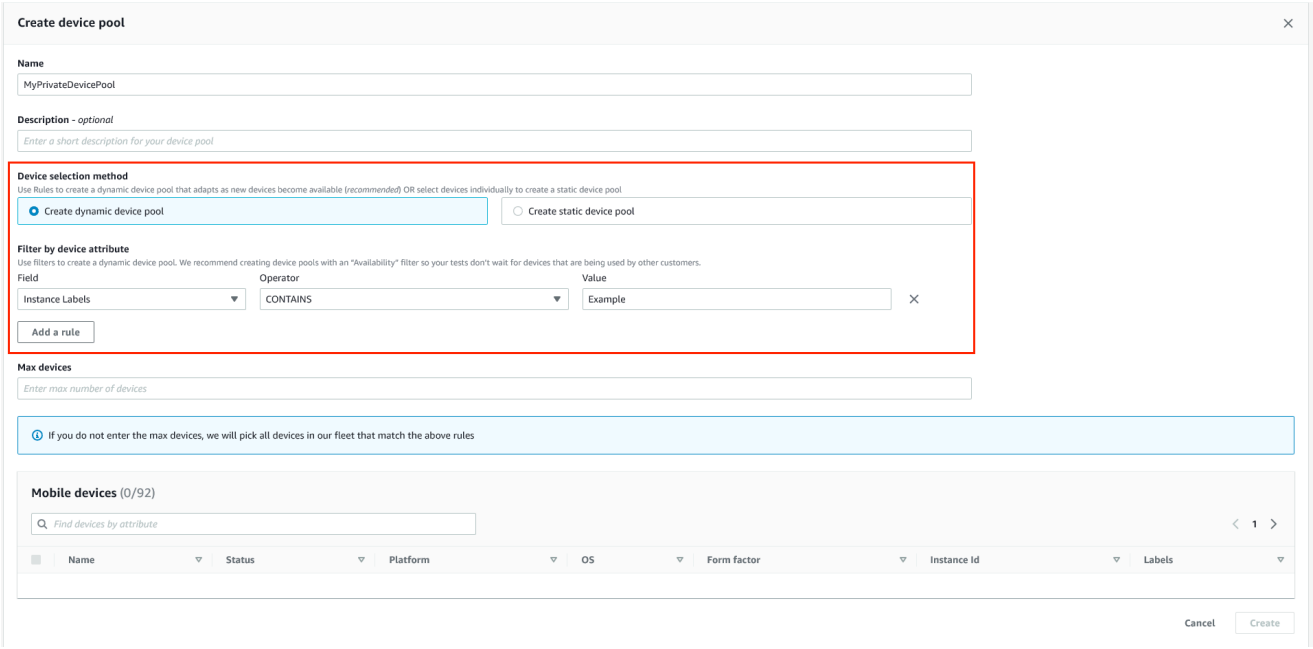
> **Note**
>
> 在主控台中使用私有裝置建立裝置集區時，您只能使用三個可用規則中的任何一個來選取私有裝置。如果您想要建立包含私有裝置多種類型規則的裝置集區（例如，包含裝置 ARNs 和裝置執行個體 ARNs 規則的裝置集區），則需要透過 CLI 或 API 建立集區。

1. 在 https：//https://console.aws.amazon.com/devicefarm/ 開啟 Device Farm 主控台。
2. 在 Device Farm 導覽面板上，選擇行動裝置測試，然後選擇專案。
3. 從清單中選擇現有的專案，或建立新的專案。若要建立新專案，請選擇新增專案，輸入專案的名稱，然後選擇提交。
4. 選擇專案設定，然後導覽至裝置集區索引標籤。
5. 選擇建立裝置集區，然後輸入裝置集區的名稱和選用描述。

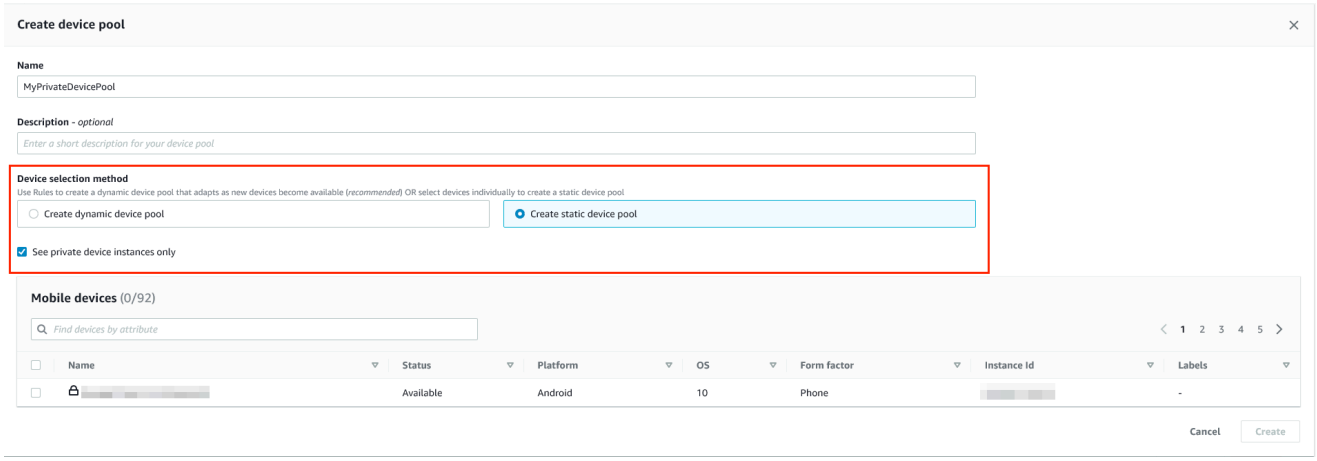
    a. 若要為裝置集區使用裝置 ARN 規則，請選擇建立靜態裝置集區，然後從清單中選擇您想要在裝置集區中使用的特定裝置類型。請勿僅選取私有裝置執行個體，因為此選項會導致使用裝置執行個體 ARN 規則（而非裝置 ARN 規則）建立裝置集區。



b. 若要使用裝置集區的裝置執行個體標籤規則，請選擇建立動態裝置集區。然後，針對您想要在裝置集區中使用的每個標籤，選擇新增規則。針對每個規則，選擇執行個體標籤做為 `Field`，選擇包含做為 `Operator`，然後將您想要的裝置執行個體標籤指定為 `Value`。



c. 若要為裝置集區使用裝置執行個體 ARN 規則，請選擇建立靜態裝置集區，然後選取私有裝置執行個體，僅將裝置清單限制為 Device Farm 與您 AWS 帳戶相關聯的私有裝置執行個體。



6. 選擇建立。

使用私有裝置建立私有裝置集區 (AWS CLI)

- 執行 create-device-pool 命令。

如需搭配 使用 Device Farm 的詳細資訊 AWS CLI，請參閱 AWS CLI 參考。

使用私有裝置 (API) 建立私有裝置集區

- 呼叫 `CreateDevicePool` API。

如需使用 Device Farm API 的詳細資訊，請參閱 自動化Device Farm。

在 AWS Device Farm 中的私有裝置上略過應用程式重新簽署

應用程式簽署程序涉及使用私有金鑰數位簽署應用程式套件（例如 APK、IPA)，然後才能安裝在裝置上或發佈到應用程式存放區，例如 Google Play Store 或 Apple App Store。為了透過減少所需的簽章和設定檔數量並提高遠端裝置上的資料安全性來簡化測試，AWS Device Farm 將在應用程式上傳至服務之後重新簽署您的應用程式。

將應用程式上傳至 AWS Device Farm 後，服務將使用自己的簽署憑證和佈建設定檔，為應用程式產生新的簽章。此程序會將原始應用程式簽章取代為 AWS Device Farm 的簽章。然後，重新簽署的應用程式會安裝在 AWS Device Farm 提供的測試裝置上。新的簽章允許在這些裝置上安裝和執行應用程式，而不需要原始開發人員的憑證。

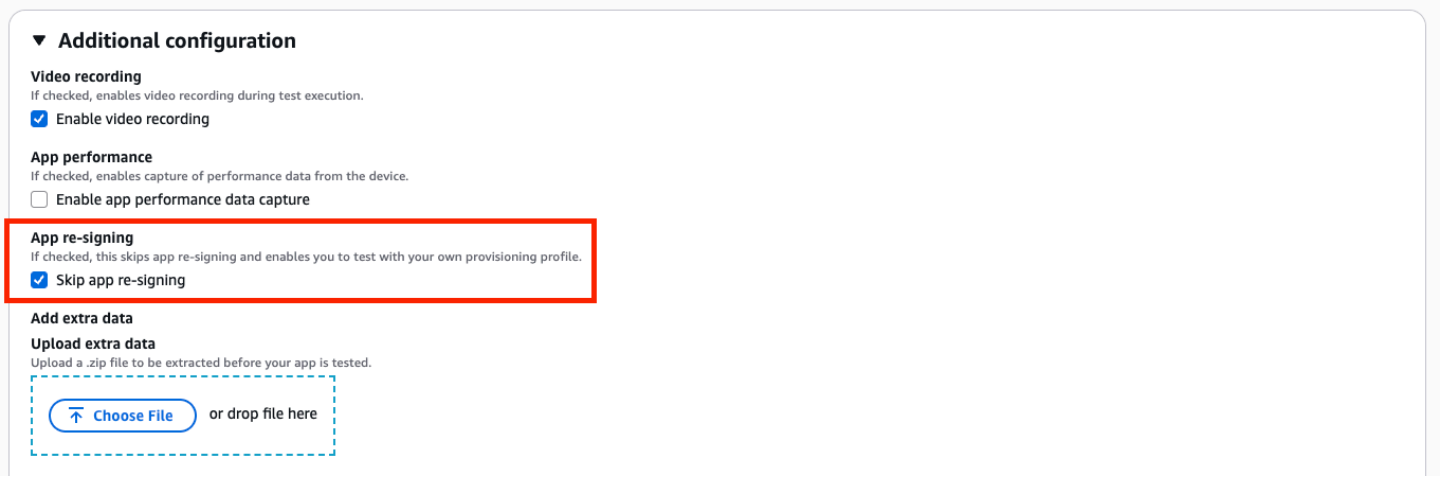
在 iOS 上，我們將內嵌佈建設定檔取代為萬用字元設定檔，並重新簽署應用程式。如果您提供它，我們會在安裝之前將輔助資料新增至應用程式套件，以便資料會出現在您應用程式的沙盒中。重新簽署 iOS 應用程式會導致移除所有權利。

在 Android 上，我們會重新簽署應用程式。這可能會中斷依賴應用程式簽章的功能，例如 Google Maps Android API。它也可能觸發 DexGuard 等產品提供的反盜版和反竄改偵測。對於內建測試，我們可能會修改資訊清單，以包含擷取和儲存螢幕擷取畫面所需的許可。

使用私有裝置時，您可以略過 AWS Device Farm 重新簽署應用程式的步驟。這與公有裝置不同，其中 Device Farm 一律會在 Android 和 iOS 平台上重新簽署您的應用程式。

您可以在建立遠端存取工作階段或測試執行時略過應用程式重新簽署。如果您的應用程式具有在 Device Farm 重新簽署應用程式時中斷的功能，這會很有幫助。例如，在重新簽署之後，推送通知可能無法運作。如需有關 Device Farm 在測試應用程式時所做的變更的詳細資訊，請參閱 AWS Device Farm FAQs或應用程式頁面。

若要略過測試執行的應用程式重新簽署，請選取其他組態下的略過應用程式重新簽署。此選項僅適用於私有裝置。



> **Note**
>
> 如果您是使用 XCTest 架構，則無法使用 Skip app re-signing (略過應用程式重新簽署) 選項。如需詳細資訊，請參閱將 Device Farm 與適用於 iOS 的 XCTest 整合。

設定您的應用程式簽署設定所需的其他步驟可能不同，取決於您使用的是私有 Android 或 iOS 裝置而定。

在 Android 裝置上略過應用程式重新簽署

如果您是在私有 Android 裝置上測試您的應用程式，請在建立您的測試執行或遠端存取工作階段時，選取 Skip app re-signing (略過應用程式重新簽署)。無需其他組態。

在 iOS 裝置上略過應用程式重新簽署

Apple 要求您在將應用程式載入裝置之前，先對應用程式簽署以進行測試。若為 iOS 裝置，您有兩個簽署應用程式的選項。

- 如果您使用的是內部 (Enterprise) 開發人員描述檔，您可以跳到下一個小節，the section called “建立遠端存取工作階段以信任您的應用程式”。

- 如果您是使用特定 iOS 應用程式開發描述檔，則必須先使用您的 Apple 開發人員帳戶註冊裝置，然後更新您的佈建描述檔以包含私有裝置。然後您必須使用您更新的佈建描述檔重新簽署應用程式。然後，您可以在 Device Farm 中執行重新簽署的應用程式。

使用特定/iOS 應用程式開發佈建設定檔註冊裝置

1. 登入您的 Apple 開發人員帳戶。
2. 導覽至主控台的 Certificates, IDs, and Profiles (憑證、ID 和設定檔) 區段。
3. 移至 Devices (裝置)。
4. 在您的 Apple 開發人員帳戶中註冊裝置。若要取得裝置的名稱和 UDID，請使用 Device Farm API `ListDeviceInstances`的操作。
5. 移至您的佈建描述檔，然後選擇 Edit (編輯)。
6. 從清單選擇裝置。
7. 在 XCode 中，擷取您更新的佈建描述檔，然後重新簽署應用程式。

無需其他組態。您現在可以建立一個遠端存取工作階段或測試執行，並選取 Skip app re-signing (略過應用程式重新簽署)。

建立遠端存取工作階段以信任您的 iOS 應用程式

如果您是使用內部 (Enterprise) 開發人員佈建描述檔，則必須執行一次性程序，來信任每個私有裝置上的內部應用程式開發人員憑證。

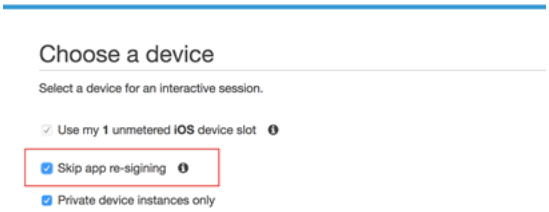
若要這樣做，您必須安裝與您要測試的應用程式使用相同憑證簽署的預留位置應用程式。裝置信任組態描述檔或企業應用程式開發人員之後，該開發人員的所有應用程式都會在私有裝置上受信任，直到您將其刪除為止。因此，當您安裝要測試的新版本應用程式時，您不必每次都再次信任應用程式開發人員。如果您執行測試自動化，而且不想要在每次測試應用程式時建立遠端存取工作階段，則這樣做特別有用。

許多客戶使用的常見程序是重新簽署適用於 iOS 的 Device Farm 範例應用程式，然後將其安裝到其裝置上做為預留位置應用程式。

在開始遠端存取工作階段之前，請遵循中的步驟在 AWS Device Farm 中建立執行個體描述檔，在 Device Farm 中建立或修改執行個體描述檔。在執行個體描述檔中，將預留位置應用程式的套件 ID 新增至從清除設定中排除套件。然後，將執行個體描述檔連接到私有裝置執行個體，以確保 Device Farm 在啟動新的測試執行之前不會從裝置中移除此應用程式。這可確保您的開發人員憑證仍得到信任。

您可以使用遠端存取工作階段將預留位置應用程式上傳至裝置，這可讓您啟動應用程式並信任開發人員。

1. 請按照建立工作階段中的指示，使用您建立的私有裝置執行個體描述檔，來建立遠端存取工作階段。當您建立工作階段時，請務必選取 Skip app re-signing (略過應用程式重新簽署)。



> **Important**
>
> 若要篩選裝置的清單以僅包括私有裝置，請選取 Private device instances only (僅限私有裝置執行個體)，以確保您是使用私有裝置與用正確的執行個體設定檔搭配。

也請務必將預留位置應用程式或您要測試的應用程式新增至此執行個體所連接之執行個體描述檔的排除套件清除設定。

2. 當您的遠端工作階段開始時，請選擇選擇檔案以安裝使用您的內部佈建設定檔的應用程式。
3. 啟動您剛上傳的應用程式。
4. 確認出現 iOS 對話方塊，指出企業應用程式開發人員不受信任。
5. 然後，如果 iOS 裝置位於 iOS 版本 18 或更高版本，請向 AWS Device Farm團隊開立支援票證，讓團隊為您信任應用程式，因為這些裝置需要手動信任應用程式。否則，如果 iOS 版本為 17 或更低，您可以前往設定應用程式，並在一般設定下，從 VPN 和設定檔選單中信任應用程式。

現在這個私有裝置信任來自這個組態設定檔或企業應用程式開發人員的所有應用程式，直到您將其刪除為止。

AWS Device Farm 中跨 AWS 區域的 Amazon VPC

Device Farm 服務僅位於美國西部 （ 奧勒岡 ） (`us-west-2`) 區域。您可以使用 Amazon Virtual Private Cloud (Amazon VPC)，使用 Device Farm 連線到另一個 AWS 區域中 Amazon Virtual Private Cloud 中的服務。如果 Device Farm 和您的服務位於相同區域，請參閱 搭配使用 Amazon VPC 端點服務與 Device Farm - Legacy （ 不建議 ）。

有兩種方式可存取位於不同 區域的私有服務。如果您的服務位於另一個非 的區域`us-west-2`，您可以使用 VPC 對等互連，將該區域的 VPC 對等到另一個與 中的 Device Farm 相連接的 VPC`us-west-2`。不過，如果您在多個區域中有 服務，Transit Gateway 可讓您使用更簡單的網路組態來存取這些服務。

如需詳細資訊，請參閱《Amazon VPC 對等互連指南》中的 VPC 對等互連案例。

AWS Device Farm 中不同區域中 VPCs 的 VPC 互連概觀

您可以對不同區域中的任兩個 VPCs 進行對等互連，只要它們具有不同的、不重疊的 CIDR 區塊即可。這可確保所有私有 IP 地址都是唯一的，並允許 VPCs 中的所有資源彼此處理，而不需要任何形式的網路地址轉譯 (NAT)。如需 CIDR 表示法的詳細資訊，請參閱 RFC 4632。

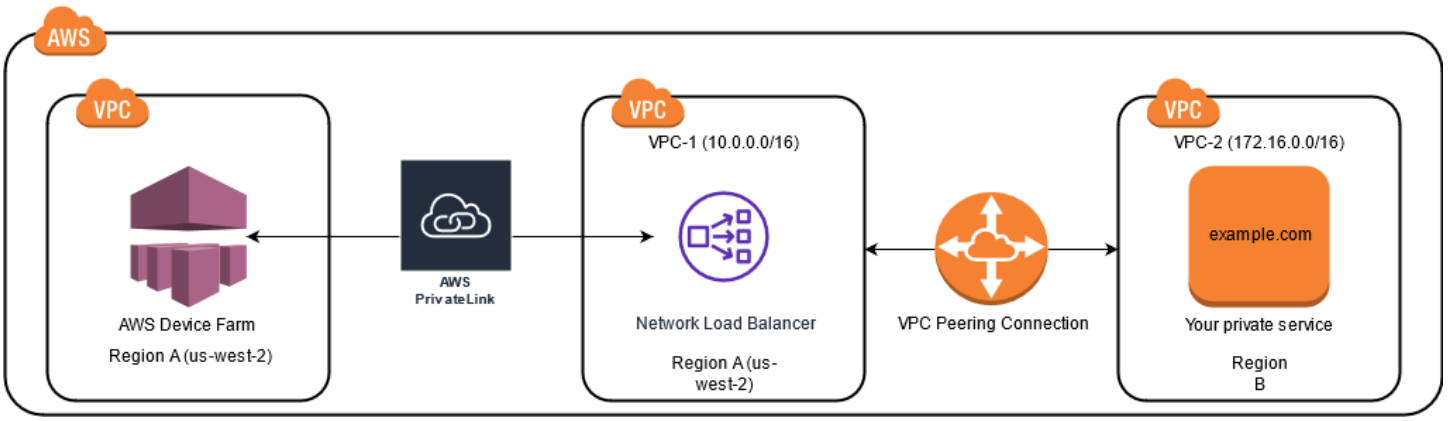
本主題包含跨區域範例案例，其中 Device Farm （稱為 VPC-1) 位於美國西部 （奧勒岡） (us-west-2) 區域。此範例中的第二個 VPC （稱為 VPC-2) 位於另一個區域。

Device Farm VPC 跨區域範例

VPC 元件	VPC-1	VPC-2
CIDR	10.0.0.0/16	172.16.0.0/16

> **Important**
>
> 在兩個 VPCs 之間建立對等連線可以變更 VPCs 的安全狀態。此外，將新項目新增至其路由表可能會變更 VPCs 內資源的安全狀態。您有責任以符合組織安全需求的方式實作這些組態。如需詳細資訊，請參閱共同責任模型。

下圖顯示範例中的元件，以及這些元件之間的互動。



主題

- 在 AWS Device Farm 中使用 Amazon VPC 的先決條件
- 步驟 1：設定 VPC-1 和 VPC-2 之間的對等互連

- 步驟 2：更新 VPC-1 和 VPC-2 中的路由表
- 步驟 3：建立目標群組
- 步驟 4：建立 Network Load Balancer
- 步驟 5：建立 VPC 端點服務，將您的 VPC 連線至 Device Farm
- 步驟 6：在您的 VPC 和 Device Farm 之間建立 VPC 端點組態
- 步驟 7：建立測試執行以使用 VPC 端點組態
- 使用 Transit Gateway 建立可擴展的網路

在 AWS Device Farm 中使用 Amazon VPC 的先決條件

此範例需要以下資訊：

- 使用包含非重疊 CIDR 區塊的子網路設定的兩個 VPCs。
- VPC-1 必須位於 us-west-2區域，並包含可用區域 us-west-2a、 us-west-2b和 的子網路us-west-2c。

如需建立 VPCs和設定子網路的詳細資訊，請參閱《Amazon VPCs 對等互連指南》中的使用 VPC 和子網路。

步驟 1：設定 VPC-1 和 VPC-2 之間的對等互連

在包含非重疊 CIDR 區塊的兩個 VPCs 之間建立對等連線。若要這樣做，請參閱《Amazon VPC 對等互連指南》中的建立和接受 VPC 對等互連。 使用此主題的跨區域案例和 Amazon VPC 對等互連指南，會建立下列對等互連組態範例：

名稱

```
Device-Farm-Peering-Connection-1
```

VPC ID（請求者）

```
vpc-0987654321gfedcba (VPC-2)
```

帳戶

```
My account
```

區域

```
US West (Oregon) (us-west-2)
```

VPC ID（接受者）

```
vpc-1234567890abcdefg (VPC-1)
```

> **Note**
>
> 建立任何新的互連連線時，請務必參閱 VPC 互連連線配額。如需詳細資訊，請參閱《Amazon VPC 對等互連指南》中的 Amazon VPC 配額。

步驟 2：更新 VPC-1 和 VPC-2 中的路由表

設定對等連線之後，您必須在兩個 VPCs之間建立目的地路由，以便在兩者之間傳輸資料。若要建立此路由，您可以手動更新 VPC-1 的路由表，以指向 VPC-2 的子網路，反之亦然。若要這樣做，請參閱《Amazon VPC 對等互連指南》中的更新 VPC 對等互連的路由表。使用此主題的跨區域案例和 Amazon VPC 對等互連指南，會建立下列範例路由表組態：

Device Farm VPC 路由表範例

VPC 元件	VPC-1	VPC-2
路由表 ID	rtb-1234567890abcdefg	rtb-0987654321gfedcba
本機地址範圍	10.0.0.0/16	172.16.0.0/16
目的地地址範圍	172.16.0.0/16	10.0.0.0/16

步驟 3：建立目標群組

設定目的地路由之後，您可以在 VPC-1 中設定 Network Load Balancer，將請求路由到 VPC-2。

Network Load Balancer 必須先包含目標群組，其中包含傳送請求的 IP 地址。

建立目標群組

1. 識別您要在 VPC-2 中鎖定的服務 IP 地址。
   - 這些 IP 地址必須是對等連線中使用的子網路成員。
   - 目標 IP 地址必須是靜態且不變的。如果您的服務具有動態 IP 地址，請考慮以靜態資源（例如 Network Load Balancer) 為目標，並將該靜態資源路由請求傳送至您的真實目標。

> **Note**
>
> - 如果您要鎖定一或多個獨立 Amazon Elastic Compute Cloud (Amazon EC2) 執行個體，請在 https：//https://console.aws.amazon.com/ec2/ 開啟 Amazon EC2 主控台，然後選擇執行個體。
> - 如果您要鎖定 Amazon EC2 Auto Scaling 群組的 Amazon EC2 執行個體，則必須將 Amazon EC2 Auto Scaling 群組與 Network Load Balancer 建立關聯。如需詳細資訊，請參閱《Amazon EC2 Auto Scaling 使用者指南》中的將負載平衡器附加到您的 Auto Scaling 群組。
>
> 然後，您可以在 https://console.aws.amazon.com/ec2/：// 開啟 Amazon EC2 主控台，然後選擇網路介面。您可以在該處檢視每個可用區域中每個 Network Load Balancer 網路介面的 IP 地址。

2. 在 VPC-1 中建立目標群組。若要這樣做，請參閱 Network Load Balancer 使用者指南中的為 Network Load Balancer 建立目標群組。

   不同 VPC 中服務的目標群組需要下列組態：

   - 針對選擇目標類型，選擇 IP 地址。
   - 針對 VPC，選擇將託管負載平衡器的 VPC。對於主題範例，這會是 VPC-1。
   - 在註冊目標頁面上，為 VPC-2 中的每個 IP 地址註冊目標。

     針對網路，選擇其他私有 IP 地址。

     針對可用區域，在 VPC-1 中選擇所需的區域。

     針對 IPv4 地址，選擇 VPC-2 IP 地址。

     針對連接埠，選擇您的連接埠。
   - 選擇包含為下方待處理項目。完成指定地址後，請選擇註冊待定目標。

使用此主題的跨區域案例和 Network Load Balancer 使用者指南，目標群組組態中會使用下列值：

Target type (目標類型)

```
IP addresses
```

目標群組名稱

```
my-target-group
```

通訊協定/連接埠

```
TCP : 80
```

VPC

```
vpc-1234567890abcdefg (VPC-1)
```

網路

```
Other private IP address
```

可用區域

```
all
```

IPv4 地址

```
172.16.100.60
```

連接埠

```
80
```

步驟 4：建立 Network Load Balancer

使用步驟 3 中所述的目標群組建立 Network Load Balancer。若要這樣做，請參閱建立 Network Load Balancer。

使用此主題的跨區域案例，下列值用於 Network Load Balancer 組態範例：

Load balancer name (負載平衡器名稱)

```
my-nlb
```

結構描述

```
Internal
```

VPC

```
vpc-1234567890abcdefg (VPC-1)
```

映射

```
us-west-2a - subnet-4i23iuufkdiufsloi

us-west-2b - subnet-7x989pkjj78nmn23j

us-west-2c - subnet-0231ndmas12bnnsds
```

通訊協定/連接埠

```
TCP : 80
```

目標群組

```
my-target-group
```

步驟 5：建立 VPC 端點服務，將您的 VPC 連線至 Device Farm

您可以使用 Network Load Balancer 來建立 VPC 端點服務。透過此 VPC 端點服務，Device Farm 可以在 VPC-2 中連線至您的服務，而不需要任何其他基礎設施，例如網際網路閘道、NAT 執行個體或 VPN 連線。

若要這樣做，請參閱建立 Amazon VPC 端點服務。

步驟 6：在您的 VPC 和 Device Farm 之間建立 VPC 端點組態

現在您可以在 VPC 和 Device Farm 之間建立私有連線。您可以使用 Device Farm 測試私有服務，而無需透過公有網際網路公開。若要這樣做，請參閱在 Device Farm 中建立 VPC 端點組態。

使用此主題的跨區域案例，下列值用於範例 VPC 端點組態：

名稱

```
My VPCE Configuration
```

VPCE 服務名稱

```
com.amazonaws.vpce.us-west-2.vpce-svc-1234567890abcdefg
```

服務 DNS 名稱

```
devicefarm.com
```

步驟 7：建立測試執行以使用 VPC 端點組態

您可以使用步驟 6 中所述的 VPC 端點組態來建立測試執行。如需詳細資訊，請參閱 在 Device Farm 中建立測試執行 或 建立工作階段 。

使用 Transit Gateway 建立可擴展的網路

若要使用兩個以上的 VPCs 建立可擴展的網路，您可以使用 Transit Gateway 做為網路傳輸中樞來互連 VPCs 和內部部署網路。若要在與 Device Farm 相同的區域中設定 VPC 以使用 Transit Gateway，您可以使用 Device Farm 指南遵循 Amazon VPC 端點服務，根據其私有 IP 地址來鎖定另一個區域中的資源。

如需 Transit Gateway 的詳細資訊，請參閱《Amazon VPC Transit Gateways 指南》中的什麼是傳輸閘道？。

終止 Device Farm 中的私有裝置

若要在初始同意的期限後終止私有裝置，您必須透過電子郵件 <aws-devicefarm-support@amazon.com> 提供 30 天的不續約通知。如需私有裝置的詳細資訊，請參閱 AWS Device Farm 中的私有裝置。

> **Important**
>
> 這些指示僅適用於終止私有裝置協議。如需所有其他 AWS 服務和帳單問題，請參閱這些產品的個別文件或聯絡 AWS 支援。

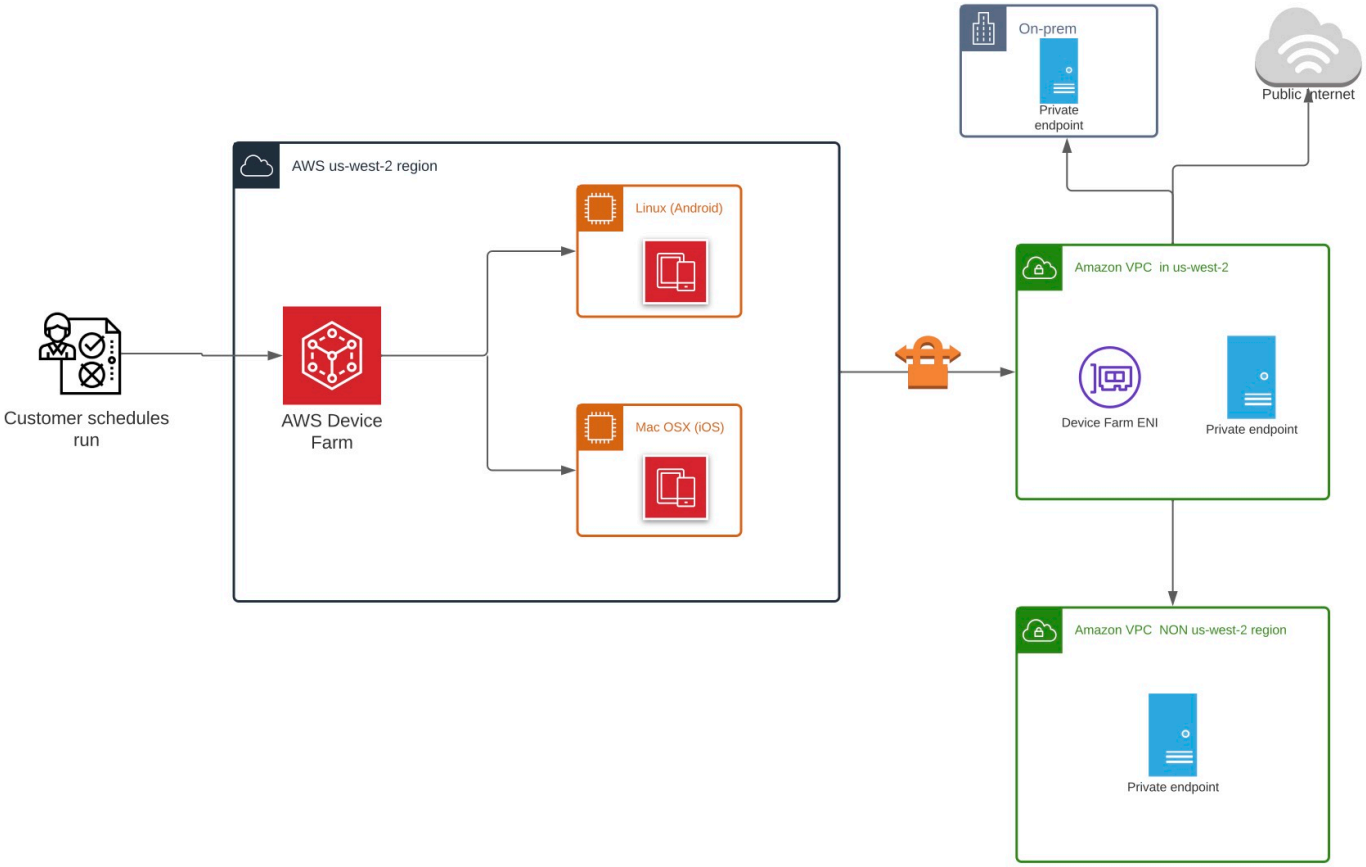
AWS Device Farm 中的 VPC-ENI

> **Warning**
>
> 此功能僅適用於私有裝置。若要請求在您的 AWS 帳戶中使用私有裝置，請聯絡我們。如果您已將私有裝置新增至 AWS 您的帳戶，強烈建議使用此 VPC 連線方法。

AWS Device Farm 的 VPC-ENI 連線功能可協助客戶安全地連線至託管於 AWS、內部部署軟體或其他雲端供應商的私有端點。

您可以將 Device Farm 行動裝置及其主機機器連線到 `us-west-2` 區域中的 Amazon Virtual Private Cloud (Amazon VPC) 環境，以便透過彈性網路界面存取隔離non-internet-facing服務和應用程式。如需 VPCs的詳細資訊，請參閱《Amazon VPC 使用者指南》。

如果您的私有端點或 VPC 不在 `us-west-2`區域中，您可以使用 Transit Gateway 或 VPC 對等互連等解決方案，將其與 `us-west-2` 區域中的 VPC 連結。 https://docs.aws.amazon.com/vpc/latest/peering/what-is-vpc-peering.html在這種情況下，Device Farm 會在您為`us-west-2`區域 VPC 提供的子網路中建立 ENI，而且您將負責確保可在`us-west-2`區域 VPC 與其他區域中的 VPC 之間建立連線。



如需有關使用 AWS CloudFormation 自動建立和對等 VPCs 的資訊，請參閱 GitHub 上範本儲存庫中的 VPCPeering AWS CloudFormation 範本。

> **Note**
>
> Device Farm 在 的客戶 VPC 中建立 ENIs 不收取任何費用`us-west-2`。跨區域或外部跨 VPC 連線的成本不包含在此功能中。

設定 VPC 存取後，您用於測試的裝置和主機機器將無法連線至 VPC 外部的資源 （例如公有 CDNs），除非您在 VPC 中指定 NAT 閘道。如需詳細資訊，請參閱 《Amazon VPC 使用者指南》中的 NAT 閘道。

主題

AWS 存取控制和 IAM

AWS Device Farm 可讓您使用 AWS Identity and Access Management(IAM) 建立政策，以授予或限制對 Device Farm 功能的存取。若要搭配 AWS Device Farm 使用 VPC 連線功能，您用來存取 AWS Device Farm 的使用者帳戶或角色需要下列 IAM 政策：

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [{
      "Effect": "Allow",
      "Action": [
        "devicefarm:*",
        "ec2:DescribeVpcs",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups",
        "ec2:CreateNetworkInterface"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "iam:CreateServiceLinkedRole",
      "Resource": "arn:aws:iam::*:role/aws-service-role/devicefarm.amazonaws.com/
AWSServiceRoleForDeviceFarm",
      "Condition": {
        "StringLike": {
```

```
          "iam:AWSServiceName": "devicefarm.amazonaws.com"
        }
      }
    }
  ]
}
```

若要使用 VPC 組態建立或更新 Device Farm 專案，您的 IAM 政策必須允許您針對 VPC 組態中列出的資源呼叫下列動作：

```
"ec2:DescribeVpcs"
"ec2:DescribeSubnets"
"ec2:DescribeSecurityGroups"
"ec2:CreateNetworkInterface"
```

此外，您的 IAM 政策也必須允許建立服務連結角色：

```
"iam:CreateServiceLinkedRole"
```

> **Note**
>
> 對於在其專案中不使用 VPC 組態的使用者，不需要這些許可。

服務連結角色

AWS Device Farm 使用 AWS Identity and Access Management (IAM) 服務連結角色。服務連結角色是直接連結至 Device Farm 的唯一 IAM 角色類型。服務連結角色是由 Device Farm 預先定義，並包含該服務代表您呼叫其他 AWS 服務所需的所有許可。

服務連結角色可讓您更輕鬆地設定 Device Farm，因為您不必手動新增必要的許可。Device Farm 定義其服務連結角色的許可，除非另有定義，否則只有 Device Farm 可以擔任其角色。定義的許可包括信任政策和許可政策，且該許可政策無法附加至其他 IAM 實體。

您必須先刪除服務連結角色的相關資源，才能將其刪除。這可保護您的 Device Farm 資源，因為您不會不小心移除存取資源的許可。

如需支援服務連結角色其他服務的資訊，請參閱可搭配 IAM 運作的 AWS 服務，並尋找 Service-Linked Role (服務連結角色) 欄顯示 Yes (是) 的服務。選擇具有連結的 Yes (是)，以檢視該服務的服務連結角色文件。

Device Farm 的服務連結角色許可

Device Farm 使用名為 AWSServiceRoleForDeviceFarm 的服務連結角色 – 允許 Device Farm 代表您存取 AWS 資源。

AWSServiceRoleForDeviceFarm 服務連結角色信任下列服務擔任該角色：

- `devicefarm.amazonaws.com`

角色許可政策允許 Device Farm 完成下列動作：

- 針對您的帳戶
  - 建立網路介面
  - 描述網路介面
  - 描述 VPCs
  - 描述子網路
  - 描述安全群組
  - 刪除界面
  - 修改網路介面
- 對於網路介面
  - 建立標籤
- 適用於 Device Farm 管理的 EC2 網路介面
  - 建立網路介面許可

完整的 IAM 政策會讀取：

JSON

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

```
    "Effect": "Allow",
    "Action": [
     "ec2:DescribeNetworkInterfaces",
     "ec2:DescribeVpcs",
     "ec2:DescribeSubnets",
     "ec2:DescribeSecurityGroups"
    ],
    "Resource": "*"
   },
   {
    "Effect": "Allow",
    "Action": [
     "ec2:CreateNetworkInterface"
    ],
    "Resource": [
     "arn:aws:ec2:*:*:subnet/*",
     "arn:aws:ec2:*:*:security-group/*"
    ]
   },
   {
    "Effect": "Allow",
    "Action": [
     "ec2:CreateNetworkInterface"
    ],
    "Resource": [
     "arn:aws:ec2:*:*:network-interface/*"
    ],
    "Condition": {
     "StringEquals": {
      "aws:RequestTag/AWSDeviceFarmManaged": "true"
     }
    }
   },
   {
    "Effect": "Allow",
    "Action": [
     "ec2:CreateTags"
    ],
    "Resource": "arn:aws:ec2:*:*:network-interface/*",
    "Condition": {
     "StringEquals": {
      "ec2:CreateAction": "CreateNetworkInterface"
     }
    }
```

```
  },
  {
   "Effect": "Allow",
   "Action": [
    "ec2:CreateNetworkInterfacePermission",
    "ec2:DeleteNetworkInterface"
   ],
   "Resource": "arn:aws:ec2:*:*:network-interface/*",
   "Condition": {
    "StringEquals": {
     "aws:ResourceTag/AWSDeviceFarmManaged": "true"
    }
   }
  },
  {
   "Effect": "Allow",
   "Action": [
    "ec2:ModifyNetworkInterfaceAttribute"
   ],
   "Resource": [
    "arn:aws:ec2:*:*:security-group/*",
    "arn:aws:ec2:*:*:instance/*"
   ]
  },
  {
   "Effect": "Allow",
   "Action": [
    "ec2:ModifyNetworkInterfaceAttribute"
   ],
   "Resource": "arn:aws:ec2:*:*:network-interface/*",
   "Condition": {
    "StringEquals": {
     "aws:ResourceTag/AWSDeviceFarmManaged": "true"
    }
   }
  }
 ]
}
```

您必須設定許可，IAM 實體 (如使用者、群組或角色) 才可建立、編輯或刪除服務連結角色。如需詳細資訊，請參閱《IAM 使用者指南》中的服務連結角色許可。

為 Device Farm 建立服務連結角色

當您為行動測試專案提供 VPC 組態時，您不需要手動建立服務連結角色。當您在 AWS 管理主控台、AWS CLI或 AWS API 中建立第一個 Device Farm 資源時，Device Farm 會為您建立服務連結角色。

若您刪除此服務連結角色，之後需要再次建立，您可以在帳戶中使用相同程序重新建立角色。當您建立第一個 Device Farm 資源時，Device Farm 會再次為您建立服務連結角色。

您也可以使用 IAM 主控台，透過 Device Farm 使用案例建立服務連結角色。在 AWS CLI 或 AWS API 中，使用服務名稱建立`devicefarm.amazonaws.com`服務連結角色。如需詳細資訊，請參閱 IAM 使用者指南中的建立服務連結角色。如果您刪除此服務連結角色，您可以使用此相同的程序以再次建立該角色。

編輯 Device Farm 的服務連結角色

Device Farm 不允許您編輯 AWSServiceRoleForDeviceFarm 服務連結角色。因為有各種實體可能會參考服務連結角色，所以您無法在建立角色之後變更角色名稱。然而，您可使用 IAM 來編輯角色描述。如需詳細資訊，請參閱《IAM 使用者指南》中的編輯服務連結角色。

刪除 Device Farm 的服務連結角色

若您不再使用需要服務連結角色的功能或服務，我們建議您刪除該角色。如此一來，您就沒有未主動監控或維護的未使用實體。然而，在手動刪除服務連結角色之前，您必須先清除資源。

> **Note**
>
> 如果 Device Farm 服務在您嘗試刪除資源時使用角色，則刪除可能會失敗。若此情況發生，請等待數分鐘後並再次嘗試操作。

使用 IAM 手動刪除服務連結角色

使用 IAM 主控台 AWS CLI、 或 AWS API 來刪除 AWSServiceRoleForDeviceFarm 服務連結角色。如需詳細資訊，請參閱《IAM 使用者指南》中的刪除服務連結角色。

Device Farm 服務連結角色支援的區域

Device Farm 支援在提供服務的所有區域中使用服務連結角色。如需詳細資訊，請參閱 AWS 區域與端點。

Device Farm 不支援在提供服務的每個區域中使用服務連結角色。您可以在下列區域中使用 AWSServiceRoleForDeviceFarm 角色。

區域名稱	區域身分	Device Farm 中的支援
美國東部 (維吉尼亞北部)	us-east-1	否
美國東部 (俄亥俄)	us-east-2	否
美國西部 (加利佛尼亞北部)	us-west-1	否
美國西部 (奧勒岡)	us-west-2	是
亞太區域 (孟買)	ap-south-1	否
亞太地區 (大阪)	ap-northeast-3	否
亞太區域 (首爾)	ap-northeast-2	否
亞太區域 (新加坡)	ap-southeast-1	否
亞太地區 (悉尼)	ap-southeast-2	否
亞太區域 (東京)	ap-northeast-1	否
加拿大 (中部)	ca-central-1	否
歐洲 (法蘭克福)	eu-central-1	否
歐洲 (愛爾蘭)	eu-west-1	否
歐洲 (倫敦)	eu-west-2	否
Europe (Paris)	eu-west-3	否
南美洲 (聖保羅)	sa-east-1	否
AWS GovCloud (US)	us-gov-west-1	否

先決條件

下列清單說明在建立 VPC-ENI 組態時要檢閱的一些需求和建議：

- 私有裝置必須指派給 AWS 您的帳戶。
- 您必須擁有具有建立服務連結角色許可 AWS 的帳戶使用者或角色。搭配 Device Farm 行動測試功能使用 Amazon VPC 端點時，Device Farm 會建立 AWS Identity and Access Management (IAM) 服務連結角色。
- Device Farm 只能在 `us-west-2` 區域中連線至 VPCs。如果您在 `us-west-2` 區域中沒有 VPC，則需要建立一個 VPC。然後，若要存取另一個區域中 VPC 中的資源，您必須在 `us-west-2` 區域中的 VPC 與另一個區域中的 VPC 之間建立對等連線。如需對等 VPCs 的詳細資訊，請參閱 Amazon VPC 對等互連指南。

  您應該在設定連線時，確認您有權存取指定的 VPC。您必須為 Device Farm 設定特定 Amazon Elastic Compute Cloud (Amazon EC2) 許可。
- 在您使用的 VPC 中需要 DNS 解析。
- 建立 VPC 之後，您將需要 區域中 VPC 的下列相關資訊`us-west-2`：
  - VPC ID
  - 子網路 IDs （僅限私有子網路）
  - 安全群組 IDs
- 您必須根據專案設定 Amazon VPC 連線。目前，每個專案只能設定一個 VPC 組態。當您設定 VPC 時，Amazon VPC 會在 VPC 內建立界面，並將其指派給指定的子網路和安全群組。所有與專案相關聯的未來工作階段都會使用設定的 VPC 連線。
- 您無法將 VPC-ENI 組態與舊版 VPCE 功能搭配使用。
- 我們強烈建議不要使用 VPC-ENI 組態更新現有專案，因為現有專案可能會有持續執行層級的 VPCE 設定。相反地，如果您已經使用現有的 VPCE 功能，請為所有新專案使用 VPC-ENI。

連線至 Amazon VPC

您可以設定和更新專案以使用 Amazon VPC 端點。VPC-ENI 組態是以每個專案為基礎進行設定。專案在任何指定時間只能有一個 VPC-ENI 端點。若要設定專案的 VPC 存取，您必須知道下列詳細資訊：

- `us-west-2` 如果您的應用程式託管在那裡，則為 中的 VPC ID，或連接到不同區域中其他 VPC 的 `us-west-2` VPC ID。
- 要套用至連線的適用安全群組。

- 將與連線相關聯的子網路。當工作階段開始時，會使用最大的可用子網路。我們建議您擁有多個與不同可用區域相關聯的子網路，以改善 VPC 連線的可用性狀態。
- 使用 VPC-ENI 時，Device Farm 測試主機和裝置所使用的 DNS 解析程式將是客戶子網路中 DHCP 服務提供的伺服器。在預設組態中，這會是 VPC 的預設解析程式。想要指定自訂 DNS 解析程式的客戶可以在其 VPC 中設定 DHCP 選項集。

建立 VPC-ENI 組態後，您可以使用主控台或 CLI 更新其詳細資訊，步驟如下。

Console

1. 登入 Device Farm 主控台，網址為 https：//https://console.aws.amazon.com/devicefarm。
2. 在 Device Farm 導覽面板上，選擇行動裝置測試，然後選擇專案。
3. 在行動測試專案下，從清單中選擇專案的名稱。
4. 選擇 Project settings (專案設定)。
5. 在虛擬私有雲端 (VPC) 設定區段中，您可以變更 VPC、 Subnets（僅限私有子網路） 和 Security Groups。
6. 選擇儲存。

CLI

使用下列 AWS CLI 命令來更新 Amazon VPC：

```
$  aws devicefarm update-project \
--arn arn:aws:devicefarm:us-
west-2:111122223333:project:12345678-1111-2222-333-456789abcdef \
--vpc-config \
securityGroupIds=sg-02c1537701a7e3763,sg-005dadf9311efda25,\
subnetIds=subnet-09b1a45f9cac53717,subnet-09b1a45f9cac12345,\
vpcId=vpc-0238fb322af81a368
```

您也可以在建立專案時設定 Amazon VPC：

```
$  aws devicefarm create-project \
--name VPCDemo \
--vpc-config \
securityGroupIds=sg-02c1537701a7e3763,sg-005dadf9311efda25,\
subnetIds=subnet-09b1a45f9cac53717,subnet-09b1a45f9cac12345,\
vpcId=vpc-0238fb322af81a368
```

限制

下列限制適用於 VPC-ENI 功能：

- 您可以在 Device Farm 專案的 VPC 組態中提供最多五個安全群組。
- 您可以在 Device Farm 專案的 VPC 組態中提供最多八個子網路。
- 設定 Device Farm 專案以使用 VPC 時，您可以提供的最小子網路必須至少有五個可用的 IPv4 地址。
- 目前不支援公有 IP 地址。反之，我們建議您在 Device Farm 專案中使用私有子網路。如果您在測試期間需要公有網際網路存取，請使用網路位址轉譯 (NAT) 閘道。使用公有子網路設定 Device Farm 專案並不會提供測試網際網路存取或公有 IP 地址。
- VPC-ENI 整合僅支援 VPC 中的私有子網路。
- 僅支援來自服務受管 ENI 的傳出流量。這表示 ENI 無法從 VPC 接收未經請求的傳入請求。

搭配使用 Amazon VPC 端點服務與 Device Farm - Legacy （不建議）

> **Warning**
>
> 我們強烈建議將此頁面所述的 VPC-ENI 連線用於私有端點連線，因為 VPCE 現在被視為舊版功能。與 VPCE 連線方法相比，VPC-ENI 提供更多彈性、更簡單的組態、更具成本效益，而且需要的維護開銷也大幅降低。

> **Note**
>
> 只有已設定私有裝置的客戶，才支援將 Amazon VPC Endpoint Services 與 Device Farm 搭配使用。若要讓您的 AWS 帳戶搭配私有裝置使用此功能，請聯絡我們。

Amazon Virtual Private Cloud (Amazon VPC) 是一種 AWS 服務，可用來在您定義的虛擬網路中啟動 AWS 資源。使用 VPC，您可以控制網路設定，例如 IP 地址範圍、子網路、路由表和網路閘道。

如果您使用 Amazon VPC 在美國西部（奧勒岡）(`us-west-2`) AWS 區域託管私有應用程式，您可以在 VPC 和 Device Farm 之間建立私有連線。透過此連線，您可以使用 Device Farm 測試私有應用程式，而無需透過公有網際網路公開。若要讓 AWS 您的帳戶搭配私有裝置使用此功能，請聯絡我們。

若要將 VPC 中的資源連線至 Device Farm，您可以使用 Amazon VPC 主控台來建立 VPC 端點服務。此端點服務可讓您透過 Device Farm VPC 端點，將 VPC 中的資源提供給 Device Farm。端點服務為 Device Farm 提供可靠、可擴展的連線，而不需要網際網路閘道、網路位址轉譯 (NAT) 執行個體或 VPN 連線。如需詳細資訊，請參閱《 AWS PrivateLink 指南》中的 VPC 端點服務 (AWS PrivateLink)。

> **Important**
>
> Device Farm VPC 端點功能可協助您使用 AWS PrivateLink 連線，將 VPC 中的私有內部服務安全地連線至 Device Farm 公有 VPC。雖然連線受到保護且為私有，該安全性取決於對 AWS 登入資料的保護。如果您的 AWS 登入資料遭到入侵，攻擊者可以存取或向外界公開您的服務資料。

在 Amazon VPC 中建立 VPC 端點服務之後，您可以使用 Device Farm 主控台在 Device Farm 中建立 VPC 端點組態。本主題說明如何在 Device Farm 中建立 Amazon VPC 連線和 VPC 端點組態。

開始之前

下列資訊適用於美國西部 （ 奧勒岡 ） (`us-west-2`) 區域的 Amazon VPC 使用者，子網路位於下列每個可用區域：us-west-2a、us-west-2b 和 us-west-2c。

Device Farm 對您可以使用的 VPC 端點服務有其他要求。當您建立和設定 VPC 端點服務以使用 Device Farm 時，請務必選擇符合下列需求的選項：

- 服務的可用區域必須包含 us-west-2a、us-west-2b 和 us-west-2c。與 VPC 端點服務相關聯的 Network Load Balancer 會決定該 VPC 端點服務的可用區域。如果您的 VPC 端點服務未顯示所有三個可用區域，您必須重新建立 Network Load Balancer 以啟用這三個區域，然後將 Network Load Balancer 與端點服務重新建立關聯。
- 端點服務的允許主體必須包含 Device Farm VPC 端點 （ 服務 ARN) 的 Amazon Resource Name (ARN)。建立端點服務之後，請將 Device Farm VPC 端點服務 ARN 新增至允許清單，以授予 Device Farm 存取 VPC 端點服務的許可。若要取得 Device Farm VPC 端點服務 ARN，請聯絡我們。

此外，如果您在建立 VPC 端點服務時保持開啟接受必要設定，則必須手動接受 Device Farm 傳送至端點服務的每個連線請求。若要變更現有端點服務的此設定，請在 Amazon VPC 主控台上選擇端點服務，選擇動作，然後選擇修改端點接受設定。如需詳細資訊，請參閱《 AWS PrivateLink 指南》中的變更負載平衡器和接受設定。

下一節說明如何建立符合這些要求的 Amazon VPC 端點服務。

步驟 1：建立 Network Load Balancer

在 VPC 和 Device Farm 之間建立私有連線的第一步是建立 Network Load Balancer，將請求路由到目標群組。

New console

使用新主控台建立 Network Load Balancer

1. 開啟位於 https：//https://console.aws.amazon.com/ec2/ 的 Amazon Elastic Compute Cloud (Amazon EC2) 主控台。
2. 在導覽窗格的負載平衡下，選擇負載平衡器。
3. 選擇 Create load balancer (建立負載平衡器)。
4. 在網路負載平衡器下，選擇建立。
5. 在建立網路負載平衡器頁面的基本組態下，執行下列動作：
   a. 輸入負載平衡器名稱。
   b. 針對結構描述，選擇內部。
6. 在 Network mappings (網路映射) 下，執行下列動作：
   a. 選擇目標群組的 VPC。
   b. 選取下列映射：
      - `us-west-2a`
      - `us-west-2b`
      - `us-west-2c`
7. 在接聽程式和路由下，使用通訊協定和連接埠選項來選擇您的目標群組。

   > **Note**
   > 預設會停用跨可用區域負載平衡。
   > 由於負載平衡器使用可用區域 `us-west-2a`、 `us-west-2b`和 `us-west-2c`，因此需要在每個可用區域中註冊目標，或者，如果您在少於所有三個區域中註冊目標，則需要啟用跨區域負載平衡。否則，負載平衡器可能無法如預期般運作。

8. 選擇 Create load balancer (建立負載平衡器)。

Old console

使用舊主控台建立 Network Load Balancer

1. 開啟位於 https：//https://console.aws.amazon.com/ec2/ 的 Amazon Elastic Compute Cloud (Amazon EC2) 主控台。
2. 在導覽窗格的負載平衡下，選擇負載平衡器。
3. 選擇 Create load balancer (建立負載平衡器)。
4. 在網路負載平衡器下，選擇建立。
5. 在設定負載平衡器頁面的基本組態下，執行下列動作：
   a. 輸入負載平衡器名稱。
   b. 針對結構描述，選擇內部。
6. 在接聽程式下，選取目標群組正在使用的通訊協定和連接埠。
7. 在可用區域下，執行下列動作：
   a. 選擇目標群組的 VPC。
   b. 選取下列可用區域：
      - `us-west-2a`
      - `us-west-2b`
      - `us-west-2c`
   c. 選擇下一步：設定安全設定。
8. （選用）設定您的安全設定，然後選擇下一步：設定路由。
9. 在 Configure Routing (設定路由) 頁面上，執行下列作業：
   a. 對於 目標群組，選擇 現有的目標群組。
   b. 針對名稱，選擇您的目標群組。
   c. 選擇下一步：註冊目標。
10. 在註冊目標頁面上，檢閱您的目標，然後選擇下一步：檢閱。

> **Note**
>
> 預設會停用跨可用區域負載平衡。

> 由於負載平衡器使用可用區域 us-west-2a、 us-west-2b和 us-west-2c，因此需要在每個可用區域中註冊目標，或者，如果您在少於所有三個區域中註冊目標，則需要啟用跨區域負載平衡。否則，負載平衡器可能無法如預期般運作。

11. 檢閱您的負載平衡器組態，然後選擇建立。

步驟 2：建立 Amazon VPC 端點服務

建立 Network Load Balancer 之後，請使用 Amazon VPC 主控台在您的 VPC 中建立端點服務。

1. 在 https://console.aws.amazon.com/vpc/ 開啟 Amazon VPC 主控台。
2. 在依區域的資源下，選擇端點服務。
3. 選擇 Create Endpoint Service (建立端點服務)。
4. 執行以下任意一項：
   - 如果您已有希望端點服務使用的 Network Load Balancer，請在可用負載平衡器下選擇它，然後繼續步驟 5。
   - 如果您尚未建立 Network Load Balancer，請選擇建立新的負載平衡器。Amazon EC2 主控台隨即開啟。請遵循從步驟 3 開始建立 Network Load Balancer 中的步驟，然後在 Amazon VPC 主控台中繼續這些步驟。
5. 對於包含的可用區域，請確認 us-west-2a、 us-west-2b和 us-west-2c出現在清單中。
6. 如果您不想手動接受或拒絕傳送到端點服務的每個連線請求，請在其他設定下清除需要接受。如果您清除此核取方塊，端點服務會自動接受它收到的每個連線請求。
7. 選擇建立。
8. 在新的端點服務中，選擇允許主體。
9. 聯絡我們以取得 Device Farm VPC 端點的 ARN （ 服務 ARN)，以新增至端點服務的允許清單，然後將該服務 ARN 新增至服務的允許清單。
10. 在端點服務的 Details (詳細資訊) 標籤上，記下服務的名稱 (服務名稱)。在下一個步驟中建立 VPC 端點組態時您會需要此名稱。

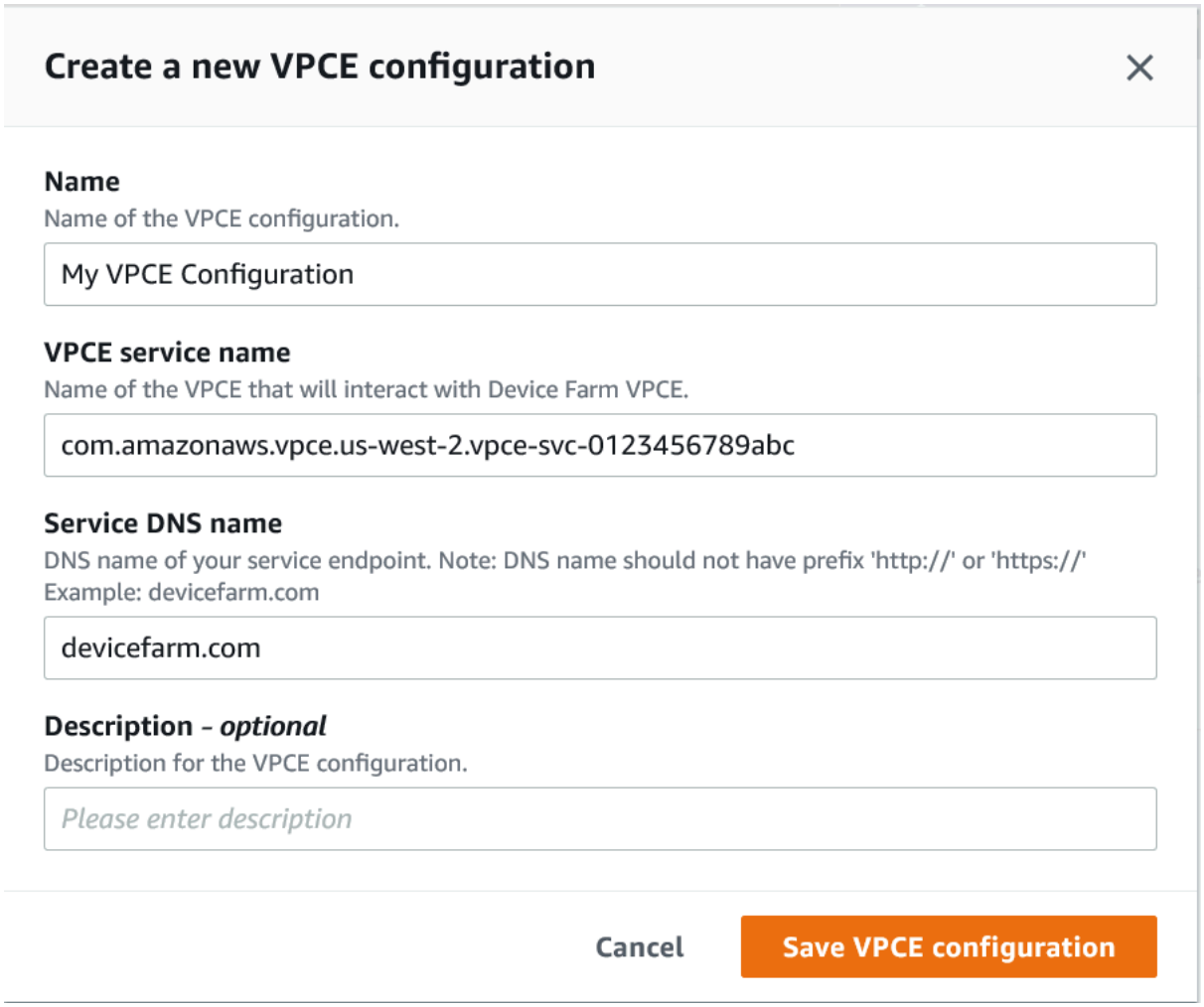
您的 VPC 端點服務現在可以與 Device Farm 搭配使用。

步驟 3：在 Device Farm 中建立 VPC 端點組態

在 Amazon VPC 中建立端點服務之後，您可以在 Device Farm 中建立 Amazon VPC 端點組態。

1. 登入 Device Farm 主控台，網址為 https：//https://console.aws.amazon.com/devicefarm。
2. 在導覽窗格中，選擇行動裝置測試，然後選擇私有裝置。
3. 選擇 VPCE 組態。
4. 選擇建立 VPCE 組態。
5. 在建立新的 VPCE 組態下，輸入 VPC 端點組態的名稱。
6. 針對 VPCE 服務名稱，輸入您在 Amazon VPC 主控台中記下的 Amazon VPC 端點服務名稱 (服務名稱)。名称的形式类似于 `com.amazonaws.vpce.us-west-2.vpce-svc-id`。
7. 針對服務 DNS 名稱，輸入您要測試的應用程式的服務 DNS 名稱 （ 例如 `devicefarm.com`)。請勿在服務 DNS 名稱前面指定 `http` 或 `https`。

   網域名稱無法透過公有網際網路存取。此外，此新網域名稱會映射至您的 VPC 端點服務，由 Amazon Route 53 產生，並僅供您在 Device Farm 工作階段中使用。
8. 選擇儲存。



步驟 4：建立測試執行

儲存 VPC 端點組態後，您可以使用組態來建立測試執行或遠端存取工作階段。如需詳細資訊，請參閱 在 Device Farm 中建立測試執行 或 建立工作階段 。

使用 記錄 AWS Device Farm API 呼叫 AWS CloudTrail

AWS Device Farm 已與 服務整合 AWS CloudTrail，此服務可提供由使用者、角色或 AWS Device Farm 中的 AWS 服務所採取之動作的記錄。CloudTrail 會將 AWS Device Farm 的所有 API 呼叫擷取為事件。擷取的呼叫包括來自 AWS Device Farm 主控台的呼叫，以及對 AWS Device Farm API 操作的程式碼呼叫。如果您建立線索，您可以啟用 CloudTrail 事件持續交付至 Amazon S3 儲存貯體，包括 AWS Device Farm 的事件。即使您未設定追蹤，依然可以透過 CloudTrail 主控台中的事件歷史記錄檢視最新事件。您可以使用 CloudTrail 所收集的資訊，判斷向 AWS Device Farm 提出的請求、提出請求的 IP 地址、提出請求的人員、提出請求的時間，以及其他詳細資訊。

若要進一步了解 CloudTrail，請參閱「AWS CloudTrail 使用者指南」。

CloudTrail 中的 AWS Device Farm 資訊

當您建立 AWS 帳戶時，會在您的帳戶上啟用 CloudTrail。當活動在 AWS Device Farm 中發生時，該活動會記錄在 CloudTrail 事件中，以及事件歷史記錄中的其他服務 AWS 事件。您可以在 AWS 帳戶中檢視、搜尋和下載最近的事件。如需詳細資訊，請參閱《使用 CloudTrail 事件歷史記錄檢視事件》https://docs.aws.amazon.com/awscloudtrail/latest/userguide/view-cloudtrail-events.html。

若要持續記錄您 AWS 帳戶中的事件，包括 AWS Device Farm 的事件，請建立追蹤。線索能讓 CloudTrail 將日誌檔案交付至 Amazon S3 儲存貯體。根據預設，當您在主控台建立追蹤記錄時，追蹤記錄會套用到所有 AWS 區域。線索會記錄 AWS 分割區中所有區域的事件，並將日誌檔案交付至您指定的 Amazon S3 儲存貯體。此外，您可以設定其他 AWS 服務，以進一步分析和處理 CloudTrail 日誌中所收集的事件資料。如需詳細資訊，請參閱下列內容：

- 建立追蹤的概觀
- CloudTrail 支援的服務和整合
- 設定 CloudTrail 的 Amazon SNS 通知
- 從多個區域接收 CloudTrail 日誌檔案，以及從多個帳戶接收 CloudTrail 日誌檔案

當 AWS 您的帳戶中啟用 CloudTrail 記錄時，對 Device Farm 動作發出的 API 呼叫會在日誌檔案中追蹤。Device Farm 記錄會與其他 AWS 服務記錄一起寫入日誌檔案中。CloudTrail 會根據期間與檔案大小，決定何時建立與寫入新檔案。

所有 Device Farm 動作都會記錄在 AWS CLI 參考和 中自動化Device Farm。例如，在 Device Farm 中建立新專案或執行的呼叫會在 CloudTrail 日誌檔案中產生項目。

每一筆事件或日誌專案都會包含產生請求者的資訊。身分資訊可協助您判斷下列事項：

- 請求是使用根或 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。

了解 AWS Device Farm 日誌檔案項目

追蹤是一種組態，能讓事件以日誌檔案的形式交付到您指定的 Amazon S3 儲存貯體。CloudTrail 日誌檔案包含一或多個日誌專案。一個事件為任何來源提出的單一請求，並包含請求動作、請求的日期和時間、請求參數等資訊。CloudTrail 日誌檔並非依公有 API 呼叫的堆疊追蹤排序，因此不會以任何特定順序出現。

下列範例顯示示範 Device Farm `ListRuns`動作的 CloudTrail 日誌項目：

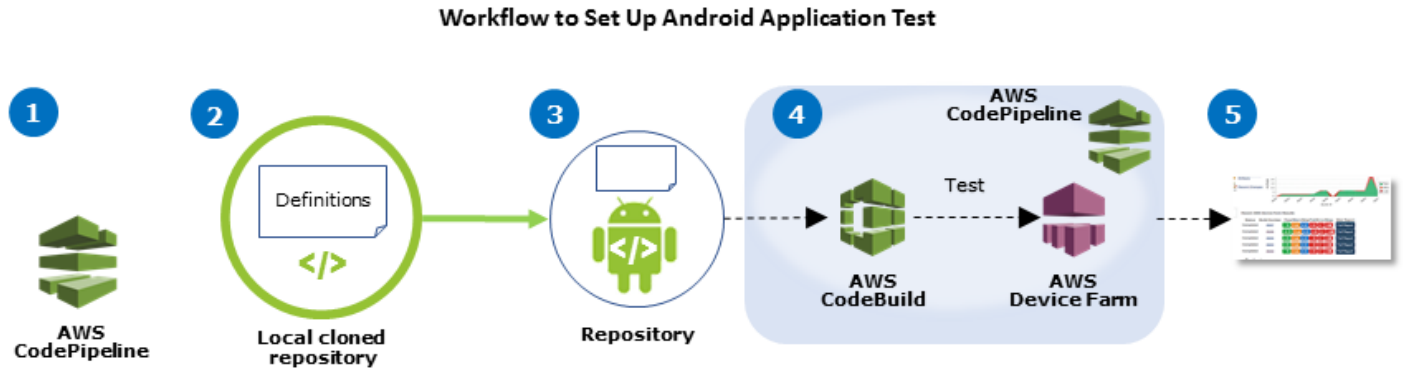
```
{
  "Records": [
    {
      "eventVersion": "1.03",
      "userIdentity": {
        "type": "Root",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "arn:aws:iam::123456789012:root",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
          "attributes": {
            "mfaAuthenticated": "false",
            "creationDate": "2015-07-08T21:13:35Z"
          }
        }
      },
      "eventTime":"2015-07-09T00:51:22Z",
      "eventSource": "devicefarm.amazonaws.com",
      "eventName":"ListRuns",
      "awsRegion":"us-west-2",
      "sourceIPAddress":"203.0.113.11",
      "userAgent":"example-user-agent-string",
      "requestParameters": {
```

```
        "arn":"arn:aws:devicefarm:us-west-2:123456789012:project:a9129b8c-
df6b-4cdd-8009-40a25EXAMPLE"},
        "responseElements": {
          "runs": [
            {
              "created": "Jul 8, 2015 11:26:12 PM",
              "name": "example.apk",
              "completedJobs": 2,
              "arn": "arn:aws:devicefarm:us-west-2:123456789012:run:a9129b8c-
df6b-4cdd-8009-40a256aEXAMPLE/1452d105-e354-4e53-99d8-6c993EXAMPLE",
              "counters": {
                "stopped": 0,
                "warned": 0,
                "failed": 0,
                "passed": 4,
                "skipped": 0,
                "total": 4,
                "errored": 0
              },
              "type": "BUILTIN_FUZZ",
              "status": "RUNNING",
              "totalJobs": 3,
              "platform": "ANDROID_APP",
              "result": "PENDING"
            },
            ... additional entries ...
          ]
        }
      }
    }
  ]
}
```

在 CodePipeline 測試階段中整合 AWS Device Farm

您可以使用 AWS CodePipeline 將 Device Farm 中設定的行動應用程式測試併入 AWS 受管的自動發行管道。您可以將管道的執行測試設定為隨需、排程，或做為持續整合流程的一部分。

下圖顯示持續整合流程，而每次推送遞交至儲存庫時，皆會進行 Android 應用程式建置和測試。若要建立此管道組態，請參閱 教學：在推送至 GitHub 時建置及測試 Android 應用程式。



1。設定	2. 新增定義	3. 推送	4. 建置和測試	5. 報告
設定管道資源	將建置及測試定義新增至您的套件	將套件推送至您的儲存庫	建置輸出成品的應用程式建置及測試會自動開始執行	檢視測試結果

若要了解如何設定管道持續測試已編譯的應用程式 (例如 iOS `.ipa` 或 Android `.apk` 檔案) 做為其來源的詳細資訊，請參閱教學：在每次您上傳 .ipa 檔案到 Amazon S3 儲存貯體時測試 iOS 應用程式。

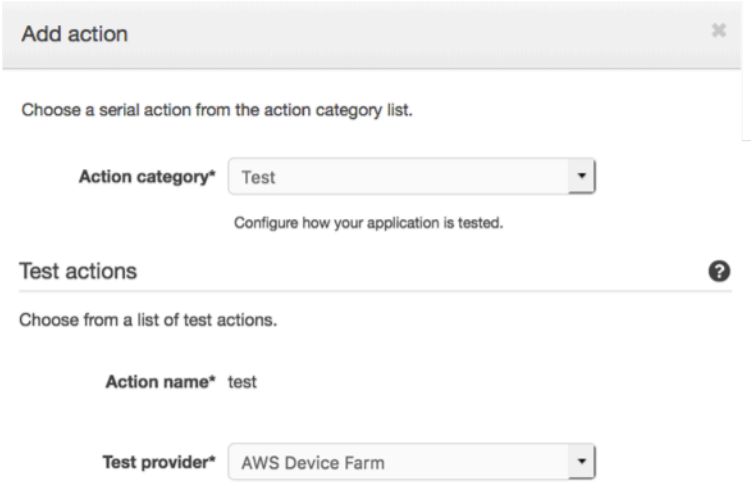
設定 CodePipeline 以使用您的 Device Farm 測試

在這些步驟中，我們假設您已設定 Device Farm 專案並建立管道。管道應該設定接收輸入成品的測試階段，其中包含您的測試定義和已編譯應用程式套件檔案。測試階段輸入成品可以是管道中來源或建置階段設定的輸出成品。

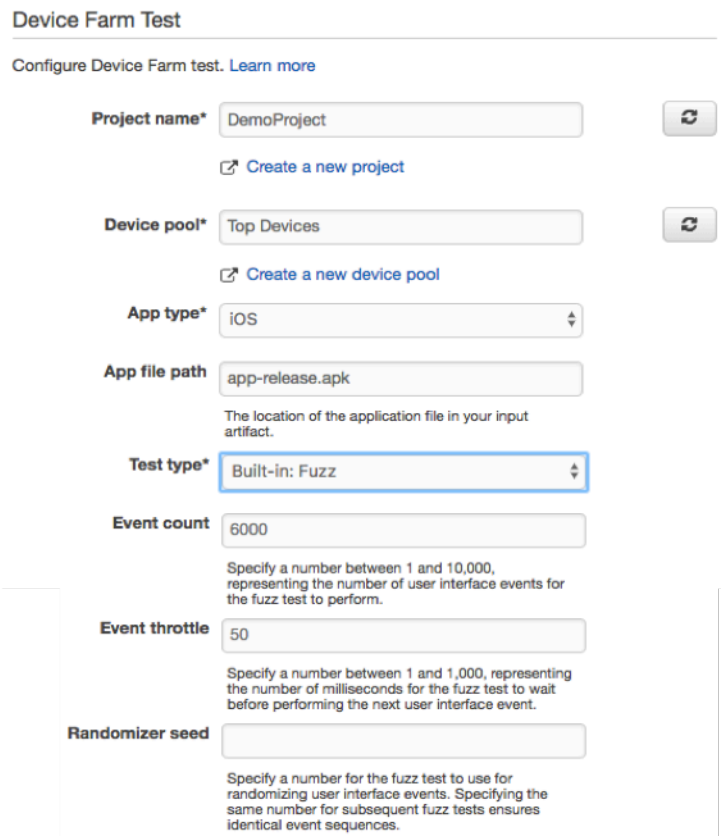
將 Device Farm 測試執行設定為 CodePipeline 測試動作

1. 登入 AWS 管理主控台 ，並在 https：//https://console.aws.amazon.com/codepipeline/ 開啟 CodePipeline 主控台。

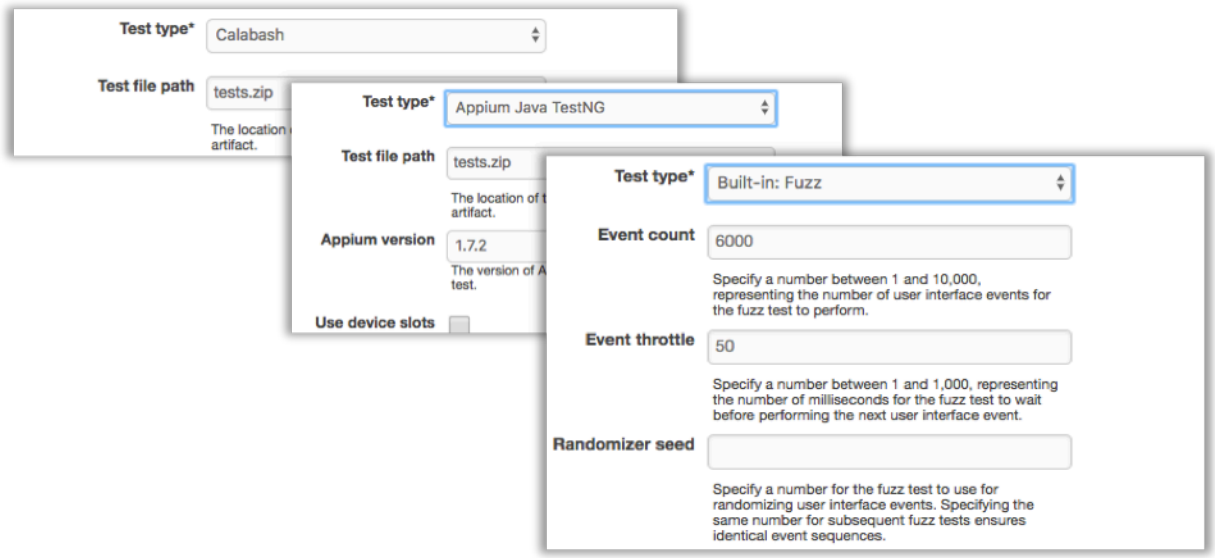
2. 選擇應用程式的發行管道。
3. 在測試階段面板中，選擇鉛筆圖示，然後選擇 Action (動作)。
4. 在 Add action (新增動作) 面板中，針對 Action category (動作類別)，選擇 Test (測試)。
5. 在 Action name (動作名稱) 中，輸入名稱。
6. 在 Test provider (測試提供者) 中，選擇 AWS Device Farm。



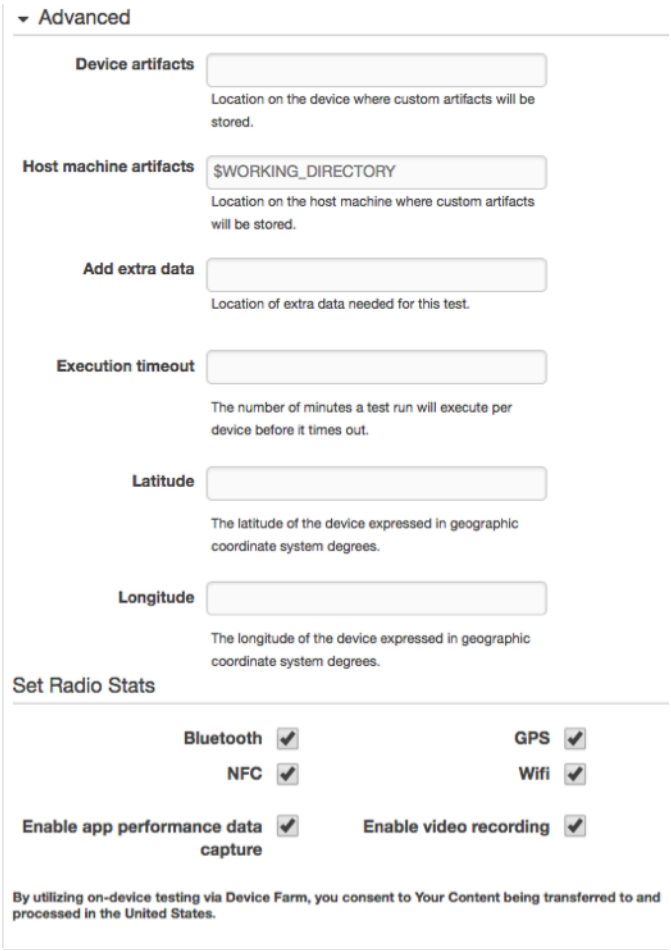
7. 在專案名稱中，選擇現有的 Device Farm 專案，或選擇建立新專案。
8. 在 Device pool (裝置集區) 中，選擇您現有的裝置集區，或是選擇 Create a new device pool (新增新裝置集區)。若您建立裝置集區，您需要選取一組測試裝置。
9. 在 App type (應用程式類型) 中，選擇應用程式的平台。



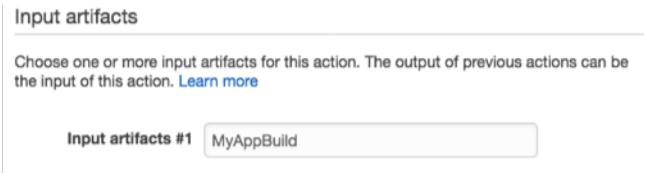
10. 在 App file path (應用程式檔案路徑) 中，輸入已編譯的應用程式套件路徑。路徑為相對於您測試輸入成品根的相對路徑。
11. 在 Test type (測試類型) 中，執行下列其中一項作業：
    - 如果您使用的是其中一個內建 Device Farm 測試，請選擇在 Device Farm 專案中設定的測試類型。
    - 如果您未使用其中一個 Device Farm 內建測試，請在測試檔案路徑中輸入測試定義檔案的路徑。路徑為相對於您測試輸入成品根的相對路徑。



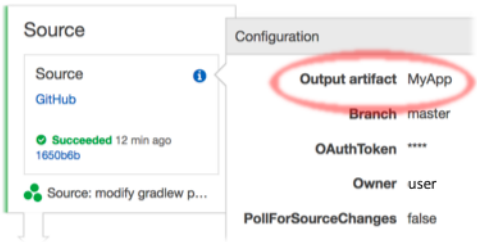
12. 在剩餘欄位中，提供適用於您測試及應用程式類型的組態。
13. (選用) 在 Advanced (進階) 中，提供測試執行的詳細組態。



14. 在 Input artifacts (輸入成品) 中，選擇與管道中測試階段前階段輸出成品相符的輸入成品。



在 CodePipeline 主控台中，將滑鼠暫留在管道圖表中的資訊圖示上，即可找到每個階段的輸出成品名稱。若您的管道是從 Source (來源) 階段直接測試您的應用程式，請選擇 MyApp。若您的管道包含 Build (建置) 階段，請選擇 MyAppBuild。



15. 在面板底部，選擇 Add Action (新增動作)。
16. 在 CodePipeline 窗格中，選擇儲存管道變更，然後選擇儲存變更。
17. 若要提交您的變更並啟動管道建置，請選擇 Release change (發行變更)，然後選擇 Release (發行)。

AWS CLI AWS Device Farm 的參考

若要使用 AWS Command Line Interface (AWS CLI) 執行 Device Farm 命令，請參閱 AWS CLI AWS Device Farm 的參考。

如需 的一般資訊 AWS CLI，請參閱 AWS Command Line Interface 使用者指南和 AWS CLI 命令參考。

AWS Device Farm 的 Windows PowerShell 參考

若要使用 Windows PowerShell 執行 Device Farm 命令，請參閱 Cmdlet Reference 中的 Device Farm AWS Tools for Windows PowerShell Cmdlet Reference。如需詳細資訊，請參閱AWS Tools for PowerShell 《使用者指南》中的設定適用於 Windows PowerShell 的 AWS 工具。

自動化 AWS Device Farm

以程式設計方式存取 Device Farm 是自動化您需要完成之常見任務的強大方式，例如排程執行或下載執行、套件或測試的成品。 AWS 開發套件和 AWS CLI 提供執行此操作的方法。

AWS 開發套件可讓您存取每項 AWS 服務，包括 Device Farm、Amazon S3 等。如需詳細資訊，請參閱

- AWS 工具和軟體開發套件
- AWS Device Farm API 參考

範例：使用 AWS CLI 或 SDK 將應用程式或測試上傳至 Device Farm

下列範例示範如何使用 CLI 或各種語言的 AWS SDK，在 Device Farm AWS 上建立上傳。上傳是在 Device Farm 上排程測試執行的核心建置區塊，並包含下列項目：

- 您的應用程式
- 您的測試
- 您的測試規格檔案

使用 `CreateUpload` API 建立上傳。此 API 會傳回 S3 預先簽章的 URL，您可以使用 HTTP PUT 請求將上傳推送至 。URL 會在 24 小時後過期。

AWS CLI

注意：此範例使用命令列工具*curl*將應用程式推送至 Device Farm。

首先，如果您尚未建立專案，請先建立專案。

```
$ aws devicefarm create-project --name MyProjectName
```

這會顯示如下所示的輸出：

```
{
```

```
    "project": {
        "name": "MyProjectName",
        "arn": "arn:aws:devicefarm:us-west-2:123456789101:project:5e01a8c7-
c861-4c0a-b1d5-12345EXAMPLE",
        "created": 1535675814.414
    }
}
```

然後，執行下列動作來建立您的上傳並將其推送至 Device Farm。在此範例中，我們將使用本機 APK 檔案建立 Android 應用程式上傳。如需更多上傳類型資訊，包括 iOS 應用程式上傳類型的詳細資訊，請參閱我們用於建立 的 API 文件Upload。

```
$ export APP_PATH="/local/path/to/my_sample_app.apk"
$ export APP_TYPE="ANDROID_APP"
```

首先，我們會在 Device Farm 中建立上傳：

```
$ aws devicefarm create-upload \
  --project-arn "arn:aws:devicefarm:us-west-2:123456789101:project:5e01a8c7-
c861-4c0a-b1d5-12345EXAMPLE" \
  --name "$(basename "$APP_PATH")" \
  --type "$APP_TYPE"
```

這會顯示如下所示的輸出：

```
{
    "upload": {
        "arn": "arn:aws:devicefarm:us-
west-2:385076942068:upload:490a6350-0ba3-43e5-83f5-d2896b069a34/a120e848-c57b-4e8d-
a720-d750a0c4d936",
        "name": "my_sample_app.apk",
        "created": 1760747318.266,
        "type": "ANDROID_APP",
        "status": "INITIALIZED",
        "url": "https://prod-us-west-2-uploads.s3.dualstack.us-west-2.amazonaws.com/
arn%3Aaws%3Adevicefarm%3Aus-west-2...",
        "category": "PRIVATE"
    }
}
```

然後，使用 curl 執行 PUT 呼叫，將應用程式推送至 Device Farm 的 S3 儲存貯體：

```
$ curl -T "$APP_PATH" "https://prod-us-west-2-uploads.s3.dualstack.us-
west-2.amazonaws.com/arn%3Aaws%3Adevicefarm%3Aus-west-2..."
```

最後，等待應用程式處於「成功」狀態：

```
$ aws devicefarm get-upload --arn "arn:aws:devicefarm:us-
west-2:385076942068:upload:490a6350-0ba3-43e5-83f5-d2896b069a34/a120e848-c57b-4e8d-
a720-d750a0c4d936"
```

這會顯示如下所示的輸出：

```
{
    "upload": {
        "arn": "arn:aws:devicefarm:us-
west-2:385076942068:upload:490a6350-0ba3-43e5-83f5-d2896b069a34/a120e848-c57b-4e8d-
a720-d750a0c4d936",
        "name": "my_sample_app.apk",
        "created": 1760747318.266,
        "type": "ANDROID_APP",
        "status": "SUCCEEDED",
        "url": "https://prod-us-west-2-uploads.s3.dualstack.us-west-2.amazonaws.com/
arn%3Aaws%3Adevicefarm%3Aus-west-2...",
        "metadata": "{\"activity_name\":
\"com.amazonaws.devicefarm.android.referenceapp.Activities.MainActivity\",
\"package_name\":\"com.amazonaws.devicefarm.android.referenceapp\",...}",
        "category": "PRIVATE"
    }
}
```

Python

注意：此範例使用第三方*requests*套件將應用程式推送至 Device Farm，以及適用於 Python 的 AWS SDK*boto3*。

首先，如果您尚未建立專案，請先建立專案。

```
import boto3

client = boto3.client("devicefarm", region_name="us-west-2")
resp = client.create_project(name="MyProjectName")

print(resp)
```

```
# Response will be something like:
# {
#     "project": {
#         "name": "MyProjectName",
#         "arn": "arn:aws:devicefarm:us-west-2:123456789101:project:5e01a8c7-
c861-4c0a-b1d5-12345EXAMPLE",
#         "created": 1535675814.414
#     }
# }
```

然後，執行下列動作來建立您的上傳並將其推送至 Device Farm。在此範例中，我們將使用本機 APK 檔案建立 Android 應用程式上傳。如需上傳類型的詳細資訊，包括 iOS 應用程式上傳類型的詳細資訊，請參閱我們用於建立 的 API 文件Upload。

```
import os
import time
import datetime
import requests
from pathlib import Path
import boto3


def upload_device_farm_file():
    project_arn = "arn:aws:devicefarm:us-west-2:123456789101:project:5e01a8c7-
c861-4c0a-b1d5-12345EXAMPLE"
    app_path = Path("/local/path/to/my_sample_app.apk")
    file_type = "ANDROID_APP"

    if not app_path.is_file():
        raise RuntimeError(f"{app_path} is not a valid app file path")

    client = boto3.client("devicefarm", region_name="us-west-2")

    # 1) Create the upload in Device Farm
    create = client.create_upload(
        projectArn=project_arn,
        name=app_path.name,
        type=file_type,
        contentType="application/octet-stream",
    )
    upload = create["upload"]
    upload_arn = upload["arn"]
    upload_url = upload["url"]
```

```
    # This will show output such as the following:
    # { "upload": { "arn": "...", "name": "my_sample_app.apk", "type":
 "ANDROID_APP", "status": "INITIALIZED", "url": "https://..." } }

    # 2) Do an HTTP PUT command to push the file to the pre-signed S3 URL
    with app_path.open("rb") as fh:
        print(f"Uploading {app_path.name} to Device Farm...")
        put_resp = requests.put(upload_url, data=fh, headers={"Content-Type":
 "application/octet-stream"})
        put_resp.raise_for_status()

    # 3) Wait for the app to be in "SUCCEEDED" status (or fail/timeout)
    timeout_seconds = 30
    start = time.time()
    while True:
        get_resp = client.get_upload(arn=upload_arn)
        status = get_resp["upload"]["status"]
        msg = get_resp["upload"].get("message") or
 get_resp["upload"].get("metadata") or ""
        elapsed = datetime.timedelta(seconds=int(time.time() - start))
        print(f"[{elapsed}] status={status}{' - ' + msg if msg else ''}")

        if status == "SUCCEEDED":
            print(f"Upload complete: {upload_arn}")
            return upload_arn
        if status == "FAILED":
            raise RuntimeError(f"Device Farm failed to process upload: {msg}")

        if (time.time() - start) > timeout_seconds:
            raise RuntimeError(f"Timed out after {timeout_seconds}s waiting for
 upload to process (last status={status}).")

        time.sleep(1)

upload_device_farm_file()
```

Java

注意：此範例使用適用於 Java v2 的 AWS 開發套件*HttpClient*，並將應用程式推送至 Device Farm，並與 JDK 第 11 版及更新版本相容。

首先，如果您尚未建立專案，請先建立專案。

```
import software.amazon.awssdk.regions.Region;
```

```
import software.amazon.awssdk.services.devicefarm.DeviceFarmClient;
import software.amazon.awssdk.services.devicefarm.model.CreateProjectRequest;
import software.amazon.awssdk.services.devicefarm.model.CreateProjectResponse;

try (DeviceFarmClient client = DeviceFarmClient.builder()
        .region(Region.US_WEST_2)
        .build()) {
    CreateProjectResponse resp = client.createProject(
        CreateProjectRequest.builder().name("MyProjectName").build());
    System.out.println(resp.project());
    // Response will be something like:
    // Project{name=MyProjectName, arn=arn:aws:devicefarm:us-
west-2:123456789101:project:5e01a8c7-..., created=...}
}
```

然後，執行下列動作來建立您的上傳並將其推送至 Device Farm。在此範例中，我們將使用本機 APK 檔案建立 Android 應用程式上傳。如需更多上傳類型資訊，包括 iOS 應用程式上傳類型的詳細資訊，請參閱我們用於建立 的 API 文件Upload。

```
import java.io.IOException;
import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.time.Duration;
import java.time.Instant;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.devicefarm.DeviceFarmClient;
import software.amazon.awssdk.services.devicefarm.model.CreateUploadRequest;
import software.amazon.awssdk.services.devicefarm.model.CreateUploadResponse;
import software.amazon.awssdk.services.devicefarm.model.GetUploadRequest;
import software.amazon.awssdk.services.devicefarm.model.GetUploadResponse;
import software.amazon.awssdk.services.devicefarm.model.Upload;
import software.amazon.awssdk.services.devicefarm.model.UploadType;

public class DeviceFarmUploader {

    public static String upload(String projectArn, Path appPath) throws Exception {
        if (projectArn == null || projectArn.isEmpty()) {
```

```
            throw new IllegalArgumentException("Missing projectArn");
        }
        if (!Files.isRegularFile(appPath)) {
            throw new IllegalArgumentException("Invalid app path: " + appPath);
        }

        String fileName = appPath.getFileName().toString().trim();
        UploadType type = UploadType.ANDROID_APP;

        // Build a reusable HttpClient
        HttpClient http = HttpClient.newBuilder()
                .version(HttpClient.Version.HTTP_1_1)
                .connectTimeout(Duration.ofSeconds(10))
                .build();

        try (DeviceFarmClient client = DeviceFarmClient.builder()
                .region(Region.US_WEST_2)
                .build()) {

            // 1) Create the upload in Device Farm
            CreateUploadResponse create =
client.createUpload(CreateUploadRequest.builder()
                    .projectArn(projectArn)
                    .name(fileName)
                    .type(type)
                    .contentType("application/octet-stream")
                    .build());

            Upload upload = create.upload();
            String uploadArn = upload.arn();
            String url = upload.url();
            // This will show output such as the following:
            // { "upload": { "arn": "...", "name": "my_sample_app.apk", "type":
"ANDROID_APP", "status": "INITIALIZED", "url": "https://..." } }

            // 2) PUT file to pre-signed URL using HttpClient
            HttpRequest put = HttpRequest.newBuilder(URI.create(url))
                    .timeout(Duration.ofMinutes(15))
                    .header("Content-Type", "application/octet-stream")
                    .PUT(HttpRequest.BodyPublishers.ofFile(appPath))
                    .build();

            HttpResponse<Void> resp = http.send(put,
HttpResponse.BodyHandlers.discarding());
```

```
            int code = resp.statusCode();
            if (code / 100 != 2) {
                throw new IOException("Failed PUT to S3 pre-signed URL, HTTP " +
 code);
            }

            // 3) Wait for the app to be in "SUCCEEDED" status (or fail/timeout)
            Instant deadline = Instant.now().plusSeconds(30); // 30-second timeout
            while (true) {
                GetUploadResponse got = client.getUpload(GetUploadRequest.builder()
                        .arn(uploadArn)
                        .build());

                String status = got.upload().statusAsString();
                String msg = got.upload().metadata();
                System.out.println("status=" + status + (msg != null ? " - " + msg :
 ""));

                if ("SUCCEEDED".equals(status)) return uploadArn;
                if ("FAILED".equals(status)) throw new RuntimeException("Upload
 failed: " + msg);
                if (Instant.now().isAfter(deadline)) {
                    throw new RuntimeException("Timeout waiting for processing, last
 status=" + status);
                }
                Thread.sleep(2000);
            }
        }
    }

    public static void main(String[] args) throws Exception {
        String projectArn = "arn:aws:devicefarm:us-
west-2:123456789101:project:5e01a8c7-c861-4c0a-b1d5-12345EXAMPLE";
        Path appPath = Paths.get("/local/path/to/my_sample_app.apk");
        String result = upload(projectArn, appPath);
        System.out.println("Upload ARN: " + result);
    }
}
```

JavaScript

注意：此範例使用適用於 JavaScript (v3) 和 Node 18+ 的 AWS SDK 將應用程式*fetch*推送至 Device Farm。

首先，如果您尚未建立專案，請先建立專案。

```
import { DeviceFarmClient, CreateProjectCommand } from "@aws-sdk/client-device-
farm";

const df = new DeviceFarmClient({ region: "us-west-2" });
const resp = await df.send(new CreateProjectCommand({ name: "MyProjectName" }));
console.log(resp);
// Response will be something like:
// { project: { name: 'MyProjectName', arn: 'arn:aws:devicefarm:us-
west-2:123456789101:project:5e01a8c7-...', created: 1535675814.414 } }
```

然後，執行下列動作來建立您的上傳並將其推送至 Device Farm。在此範例中，我們將使用本機 APK 檔案建立 Android 應用程式上傳。如需上傳類型的詳細資訊，包括 iOS 應用程式上傳類型的詳細資訊，請參閱我們用於建立 的 API 文件Upload。

```
import { DeviceFarmClient, CreateUploadCommand, GetUploadCommand } from "@aws-sdk/
client-device-farm";
import { createReadStream } from "fs";
import { basename } from "path";

const projectArn = "arn:aws:devicefarm:us-west-2:123456789101:project:5e01a8c7-
c861-4c0a-b1d5-12345EXAMPLE";
const appPath = "/local/path/to/my_sample_app.apk";
const name = basename(appPath).trim();
const type = "ANDROID_APP";

const client = new DeviceFarmClient({ region: "us-west-2" });

// 1) Create the upload in Device Farm
const create = await client.send(new CreateUploadCommand({
  projectArn,
  name,
  type,
  contentType: "application/octet-stream",
}));

const uploadArn = create.upload.arn;
const url = create.upload.url;
// This will show output such as the following:
// { upload: { arn: '...', name: 'my_sample_app.apk', type: 'ANDROID_APP', status:
 'INITIALIZED', url: 'https://...' } }
```

```
// 2) PUT to pre-signed URL
const putResp = await fetch(url, {
  method: "PUT",
  headers: { "Content-Type": "application/octet-stream" },
  body: createReadStream(appPath),
});
if (!putResp.ok) {
  throw new Error(`Failed PUT to pre-signed URL: ${putResp.status} ${await
 putResp.text().catch(()=>"")}`);
}

// 3) Wait for the app to be in "SUCCEEDED" status (or fail/timeout)
const deadline = Date.now() + (30 * 1000); // 30-second timeout
while (true) {
  const response = await client.send(new GetUploadCommand({ arn: uploadArn }));
  const { status, message, metadata } = response.upload;
  console.log(`status=${status}${message ? " - " + message : metadata ? " - " +
 metadata : ""}`);
  if (status === "SUCCEEDED") {
    console.log("Upload complete:", uploadArn);
    break;
  }
  if (status === "FAILED") {
    throw new Error(`Upload failed: ${message || metadata || "unknown"}`);
  }
  if (Date.now() > deadline) throw new Error(`Timeout waiting for processing (last
 status=${status})`);
  await new Promise(r => setTimeout(r, 2000));
}
```

C#

注意：此範例使用適用於 .NET 的 AWS 開發套件*HttpClient*，並將應用程式推送至 Device Farm。

首先，如果您尚未建立專案，請先建立專案。

```
using System;
using Amazon;
using Amazon.DeviceFarm;
using Amazon.DeviceFarm.Model;

using var client = new AmazonDeviceFarmClient(RegionEndpoint.USWest2);
```

```
var resp = await client.CreateProjectAsync(new CreateProjectRequest { Name =
 "MyProjectName" });
Console.WriteLine(resp.Project);
// Response will be something like:
// { Name = MyProjectName, Arn = arn:aws:devicefarm:us-
west-2:123456789101:project:5e01a8c7-..., Created = ... }
```

然後，執行下列動作來建立您的上傳並將其推送至 Device Farm。在此範例中，我們將使用本機 APK 檔案建立 Android 應用程式上傳。如需更多上傳類型資訊，包括 iOS 應用程式上傳類型的詳細資訊，請參閱我們用於建立 的 API 文件Upload。

```
using System;
using System.IO;
using System.Net.Http;
using System.Threading.Tasks;
using System.Net.Http.Headers;
using Amazon;
using Amazon.DeviceFarm;
using Amazon.DeviceFarm.Model;

class DeviceFarmUploader
{
    public static async Task<string> UploadAsync(string projectArn, string appPath)
    {
        if (string.IsNullOrWhiteSpace(projectArn)) throw new
 ArgumentException("Missing projectArn");
        if (!File.Exists(appPath)) throw new ArgumentException($"Invalid app path:
 {appPath}");
        var type = UploadType.ANDROID_APP;

        using var client = new AmazonDeviceFarmClient(RegionEndpoint.USWest2);
        // 1) Create the upload in Device Farm
        var create = await client.CreateUploadAsync(new CreateUploadRequest
        {
            ProjectArn = projectArn,
            Name = Path.GetFileName(appPath),
            Type = type,
            ContentType = "application/octet-stream"
        });

        var uploadArn = create.Upload.Arn;
        var url = create.Upload.Url;
        // This will show output such as the following:
```

```
        // { Upload: { Arn = ..., Name = my_sample_app.apk, Type = ANDROID_APP,
 Status = INITIALIZED, Url = https://... } }

        // 2) PUT file to pre-signed URL
        using (var http = new HttpClient())
        using (var fs = File.OpenRead(appPath))
        using (var content = new StreamContent(fs))
        {
            content.Headers.Add("Content-Type", "application/octet-stream");
            var resp = await http.PutAsync(url, content);
            if (!resp.IsSuccessStatusCode)
                throw new Exception($"Failed PUT to pre-signed URL:
 {(int)resp.StatusCode} {await resp.Content.ReadAsStringAsync()}");
        }

        // 3) Wait for the app to be in "SUCCEEDED" status (or fail/timeout)
        var deadline = DateTime.UtcNow.AddSeconds(30); // 30-second timeout
        while (true)
        {
            var got = await client.GetUploadAsync(new GetUploadRequest { Arn =
 uploadArn });
            var status = got.Upload.Status.Value;
            var msg = got.Upload.Message ?? got.Upload.Metadata;
            Console.WriteLine($"status={status}{(string.IsNullOrEmpty(msg) ? "" : "
 - " + msg)}");

            if (status == UploadStatus.SUCCEEDED.Value) return uploadArn;
            if (status == UploadStatus.FAILED.Value) throw new Exception($"Upload
 failed: {msg}");
            if (DateTime.UtcNow > deadline) throw new TimeoutException($"Timeout
 waiting for processing (last status={status})");
            await Task.Delay(2000);
        }
    }

    static async Task Main()
    {
        var projectArn = "arn:aws:devicefarm:us-
west-2:123456789101:project:5e01a8c7-c861-4c0a-b1d5-12345EXAMPLE";
        var appPath = "/local/path/to/my_sample_app.apk";
        var result = await UploadAsync(projectArn!, appPath!);
        Console.WriteLine("Upload ARN: " + result);
    }
```

```
}
```

Ruby

注意：此範例使用適用於 Ruby 的 AWS 開發套件*Net::HTTP*，並將應用程式推送至 Device Farm。

首先，如果您尚未建立專案，請先建立專案。

```
require "aws-sdk-devicefarm"

client = Aws::DeviceFarm::Client.new(region: "us-west-2")
resp = client.create_project(name: "MyProjectName")
puts resp.project.inspect
# Response will be something like:
# #<struct Aws::DeviceFarm::Types::Project name="MyProjectName",
 arn="arn:aws:devicefarm:us-west-2:123456789101:project:5e01a8c7-...",
 created=1535675814.414>
```

然後，執行下列動作來建立您的上傳並將其推送至 Device Farm。在此範例中，我們將使用本機 APK 檔案建立 Android 應用程式上傳。如需更多上傳類型資訊，包括 iOS 應用程式上傳類型的詳細資訊，請參閱我們用於建立 的 API 文件Upload。

```
require "aws-sdk-devicefarm"
require "net/http"
require "uri"

project_arn = "arn:aws:devicefarm:us-west-2:123456789101:project:5e01a8c7-c861-4c0a-
b1d5-12345EXAMPLE"
app_path    = "/local/path/to/my_sample_app.apk"
raise "Invalid APP_PATH: #{app_path}" unless File.file?(app_path)
type = "ANDROID_APP"

client = Aws::DeviceFarm::Client.new(region: "us-west-2")

# 1) Create the upload in Device Farm
create = client.create_upload(
  project_arn: project_arn,
  name: File.basename(app_path),
  type: type,
  content_type: "application/octet-stream"
)
```

```
upload_arn = create.upload.arn
url = create.upload.url
# This will show output such as the following:
# #<Upload arn="...", name="my_sample_app.apk", type="ANDROID_APP",
 status="INITIALIZED", url="https://...">

# 2) PUT the file to the pre-signed URL
uri = URI.parse(url)
Net::HTTP.start(uri.host, uri.port, use_ssl: (uri.scheme == "https")) do |http|
  req = Net::HTTP::Put.new(uri)
  req["Content-Type"] = "application/octet-stream"
  req.body_stream = File.open(app_path, "rb")
  req.content_length = File.size(app_path)
  resp = http.request(req)
  raise "Failed PUT: #{resp.code} #{resp.body}" unless resp.code.to_i / 100 == 2
end

# 3) Wait for the app to be in "SUCCEEDED" status (or fail/timeout)
deadline = Time.now + 30 # 30-second timeout
loop do
  got = client.get_upload(arn: upload_arn)
  status = got.upload.status
  msg = got.upload.message || got.upload.metadata
  puts "status=#{status}#{msg ? " - #{msg}" : ""}"

  case status
  when "SUCCEEDED" then puts "Upload complete: #{upload_arn}"; break
  when "FAILED"    then raise "Upload failed: #{msg}"
  end
  raise "Timeout waiting for processing (last status=#{status})" if Time.now >
 deadline
  sleep 2
end
```

範例：使用 AWS SDK 啟動 Device Farm 執行並收集成品

下列範例提供如何使用 AWS SDK 搭配 Device Farm beginning-to-end示範。此範例會執行下列操作：

- 將測試和應用程式套件上傳至 Device Farm
- 開始測試執行並等待其完成 (或失敗)
- 下載測試套件產生的所有成品

此範例取決於與 HTTP 互動的第三方 `requests` 套件。

```
import boto3
import os
import requests
import string
import random
import time
import datetime
import time
import json

# The following script runs a test through Device Farm
#
# Things you have to change:
config = {
    # This is our app under test.
    "appFilePath":"app-debug.apk",
    "projectArn": "arn:aws:devicefarm:us-
west-2:111122223333:project:1b99bcff-1111-2222-ab2f-8c3c733c55ed",
    # Since we care about the most popular devices, we'll use a curated pool.
    "testSpecArn":"arn:aws:devicefarm:us-west-2::upload:101e31e8-12ac-11e9-ab14-
d663bd873e83",
    "poolArn":"arn:aws:devicefarm:us-west-2::devicepool:082d10e5-d7d7-48a5-ba5c-
b33d66efa1f5",
    "namePrefix":"MyAppTest",
    # This is our test package. This tutorial won't go into how to make these.
    "testPackage":"tests.zip"
}

client = boto3.client('devicefarm')

unique =
 config['namePrefix']+"-"+(datetime.date.today().isoformat())+(''.join(random.sample(string.asc

print(f"The unique identifier for this run is going to be {unique} -- all uploads will
 be prefixed with this.")

def upload_df_file(filename, type_, mime='application/octet-stream'):
    response = client.create_upload(projectArn=config['projectArn'],
        name = (unique)+"_"+os.path.basename(filename),
        type=type_,
        contentType=mime
        )
```

```
    # Get the upload ARN, which we'll return later.
    upload_arn = response['upload']['arn']
    # We're going to extract the URL of the upload and use Requests to upload it
    upload_url = response['upload']['url']
    with open(filename, 'rb') as file_stream:
        print(f"Uploading {filename} to Device Farm as {response['upload']['name']}...
 ",end='')
        put_req = requests.put(upload_url, data=file_stream, headers={"content-
type":mime})
        print(' done')
        if not put_req.ok:
            raise Exception("Couldn't upload, requests said we're not ok. Requests
 says: "+put_req.reason)
    started = datetime.datetime.now()
    while True:
        print(f"Upload of {filename} in state {response['upload']['status']} after
 "+str(datetime.datetime.now() - started))
        if response['upload']['status'] == 'FAILED':
            raise Exception("The upload failed processing. DeviceFarm says reason
 is: \n"+(response['upload']['message'] if 'message' in response['upload'] else
 response['upload']['metadata']))
        if response['upload']['status'] == 'SUCCEEDED':
            break
        time.sleep(5)
        response = client.get_upload(arn=upload_arn)
    print("")
    return upload_arn

our_upload_arn = upload_df_file(config['appFilePath'], "ANDROID_APP")
our_test_package_arn = upload_df_file(config['testPackage'],
 'APPIUM_PYTHON_TEST_PACKAGE')
print(our_upload_arn, our_test_package_arn)
# Now that we have those out of the way, we can start the test run...
response = client.schedule_run(
    projectArn = config["projectArn"],
    appArn = our_upload_arn,
    devicePoolArn = config["poolArn"],
    name=unique,
    test = {
        "type":"APPIUM_PYTHON",
        "testSpecArn": config["testSpecArn"],
        "testPackageArn": our_test_package_arn
        }
    )
```

```
run_arn = response['run']['arn']
start_time = datetime.datetime.now()
print(f"Run {unique} is scheduled as arn {run_arn} ")

try:

    while True:
        response = client.get_run(arn=run_arn)
        state = response['run']['status']
        if state == 'COMPLETED' or state == 'ERRORED':
            break
        else:
            print(f" Run {unique} in state {state}, total time
 "+str(datetime.datetime.now()-start_time))
            time.sleep(10)
except:
    # If something goes wrong in this process, we stop the run and exit.

    client.stop_run(arn=run_arn)
    exit(1)
print(f"Tests finished in state {state} after "+str(datetime.datetime.now() -
 start_time))
# now, we pull all the logs.
jobs_response = client.list_jobs(arn=run_arn)
# Save the output somewhere. We're using the unique value, but you could use something
 else
save_path = os.path.join(os.getcwd(), unique)
os.mkdir(save_path)
# Save the last run information
for job in jobs_response['jobs'] :
    # Make a directory for our information
    job_name = job['name']
    os.makedirs(os.path.join(save_path, job_name), exist_ok=True)
    # Get each suite within the job
    suites = client.list_suites(arn=job['arn'])['suites']
    for suite in suites:
        for test in client.list_tests(arn=suite['arn'])['tests']:
            # Get the artifacts
            for artifact_type in ['FILE','SCREENSHOT','LOG']:
                artifacts = client.list_artifacts(
                    type=artifact_type,
                    arn = test['arn']
                )['artifacts']
                for artifact in artifacts:
```

```
                    # We replace : because it has a special meaning in Windows & macos
                    path_to = os.path.join(save_path, job_name, suite['name'],
 test['name'].replace(':','_') )
                    os.makedirs(path_to, exist_ok=True)
                    filename =
 artifact['type']+"_"+artifact['name']+"."+artifact['extension']
                    artifact_save_path = os.path.join(path_to, filename)
                    print("Downloading "+artifact_save_path)
                    with open(artifact_save_path, 'wb') as fn,
 requests.get(artifact['url'],allow_redirects=True) as request:
                        fn.write(request.content)
                    #/for artifact in artifacts
                #/for artifact type in []
            #/ for test in ()[]
        #/ for suite in suites
    #/ for job in _[]
# done
print("Finished")
```

故障診斷 Device Farm 錯誤

在本節中，您會找到錯誤訊息和程序，協助您修正 Device Farm 的常見問題。

> **Note**
>
> 若要對 Device Farm 上意外失敗的 Appium 測試進行故障診斷，請參閱我們的用戶端 Appium 測試指南

主題

針對 AWS Device Farm 中的 Android 應用程式測試進行故障診斷

下列主題會列出在上傳 Android 應用程式測試期間出現的錯誤訊息，並建議解決每個錯誤的解決方法。

> **Note**
>
> 以下說明以 Linux x86_64 和 Mac 為基礎。

ANDROID_APP_UNZIP_FAILED

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 無法開啟您的應用程式。請確認檔案是否有效，然後再試一次。

請確認您可以正確解壓縮應用程式套件。在下列範例中，套件的名稱是 app-debug.apk。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip app-debug.apk
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

有效的 Android 應用程式套件應產生輸出如下：

```
.
|-- AndroidManifest.xml
|-- classes.dex
|-- resources.arsc
|-- assets (directory)
|-- res (directory)
`-- META-INF (directory)
```

如需詳細資訊，請參閱AWS Device Farm 中的 Android 測試。

ANDROID_APP_AAPT_DEBUG_BADGING_FAILED

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 無法擷取應用程式的相關資訊。請執行命令 `aapt debug badging` *`<path to your test package>`*，確認應用程式是否有效，並在命令未列印任何錯誤後再試一次。

在上傳驗證程序期間，AWS Device Farm 會從 `aapt debug badging` *`<path to your package>`*命令的輸出中剖析資訊。

請確認您可以在 Android 應用程式上成功執行此命令。在下列範例中，套件的名稱是 app-debug.apk。

- 將您的應用程式套件複製到工作目錄，然後執行命令：

```
$ aapt debug badging app-debug.apk
```

有效的 Android 應用程式套件應產生輸出如下：

```
package: name='com.amazon.aws.adf.android.referenceapp' versionCode='1'
 versionName='1.0' platformBuildVersionName='5.1.1-1819727'
sdkVersion:'9'
application-label:'ReferenceApp'
application: label='ReferenceApp' icon='res/mipmap-mdpi-v4/ic_launcher.png'
application-debuggable
launchable-activity:
 name='com.amazon.aws.adf.android.referenceapp.Activities.MainActivity'
 label='ReferenceApp' icon=''
uses-feature: name='android.hardware.bluetooth'
uses-implied-feature: name='android.hardware.bluetooth' reason='requested
 android.permission.BLUETOOTH permission, and targetSdkVersion > 4'
main
supports-screens: 'small' 'normal' 'large' 'xlarge'
supports-any-density: 'true'
locales: '--_--'
densities: '160' '213' '240' '320' '480' '640'
```

如需詳細資訊，請參閱AWS Device Farm 中的 Android 測試。

ANDROID_APP_PACKAGE_NAME_VALUE_MISSING

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 在應用程式中找不到套件名稱值。請執行命令 `aapt debug badging` *`<path to your test package>`* 驗證應用程式是否有效，並在以關鍵字「package：name」找到套件名稱值後再試一次。

在上傳驗證程序期間，AWS Device Farm 會從 `aapt debug badging <path to your package>`命令的輸出剖析套件名稱值。

請確認您可以在 Android 應用程式上執行此命令，並成功找到套件名稱值。在下列範例中，套件的名稱是 app-debug.apk。

- 將您的應用程式套件複製到工作目錄，然後執行下列命令：

  ```
  $ aapt debug badging app-debug.apk | grep "package: name="
  ```

  有效的 Android 應用程式套件應產生輸出如下：

  ```
  package: name='com.amazon.aws.adf.android.referenceapp' versionCode='1'
   versionName='1.0' platformBuildVersionName='5.1.1-1819727'
  ```

  如需詳細資訊，請參閱AWS Device Farm 中的 Android 測試。

ANDROID_APP_SDK_VERSION_VALUE_MISSING

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 在應用程式中找不到軟體開發套件版本值。請執行命令 `aapt debug badging <path to your test package>` 驗證應用程式是否有效，並在以關鍵字 `sdkVersion` 找到軟體開發套件版本值後再試一次。

在上傳驗證程序期間，AWS Device Farm 會從 `aapt debug badging <path to your package>`命令的輸出剖析 SDK 版本值。

請確認您可以在 Android 應用程式上執行此命令，並成功找到套件名稱值。在下列範例中，套件的名稱是 app-debug.apk。

- 將您的應用程式套件複製到工作目錄，然後執行下列命令：

  ```
  $ aapt debug badging app-debug.apk | grep "sdkVersion"
  ```

  有效的 Android 應用程式套件應產生輸出如下：

```
sdkVersion:'9'
```

如需詳細資訊，請參閱AWS Device Farm 中的 Android 測試。

ANDROID_APP_AAPT_DUMP_XMLTREE_FAILED

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 在應用程式中找不到有效的 AndroidManifest.xml。請執行命令 `aapt dump xmltree <path to your test package> AndroidManifest.xml`，確認測試套件是否有效，並在命令未列印任何錯誤後再試一次。

在上傳驗證程序期間，AWS Device Farm 會使用 命令，從 XML 剖析樹狀目錄中剖析包含在套件中的 XML 檔案的資訊`aapt dump xmltree <path to your package> AndroidManifest.xml`。

請確認您可以在 Android 應用程式上成功執行此命令。在下列範例中，套件的名稱是 app-debug.apk。

- 將您的應用程式套件複製到工作目錄，然後執行下列命令：

  ```
  $ aapt dump xmltree app-debug.apk. AndroidManifest.xml
  ```

  有效的 Android 應用程式套件應產生輸出如下：

  ```
  N: android=http://schemas.android.com/apk/res/android
    E: manifest (line=2)
      A: android:versionCode(0x0101021b)=(type 0x10)0x1
      A: android:versionName(0x0101021c)="1.0" (Raw: "1.0")
      A: package="com.amazon.aws.adf.android.referenceapp" (Raw:
   "com.amazon.aws.adf.android.referenceapp")
      A: platformBuildVersionCode=(type 0x10)0x16 (Raw: "22")
      A: platformBuildVersionName="5.1.1-1819727" (Raw: "5.1.1-1819727")
      E: uses-sdk (line=7)
        A: android:minSdkVersion(0x0101020c)=(type 0x10)0x9
        A: android:targetSdkVersion(0x01010270)=(type 0x10)0x16
      E: uses-permission (line=11)
  ```

```
      A: android:name(0x01010003)="android.permission.INTERNET" (Raw:
 "android.permission.INTERNET")
    E: uses-permission (line=12)
      A: android:name(0x01010003)="android.permission.CAMERA" (Raw:
 "android.permission.CAMERA")
```

如需詳細資訊，請參閱AWS Device Farm 中的 Android 測試。

ANDROID_APP_DEVICE_ADMIN_PERMISSIONS

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 您的應用程式需要裝置管理許可。請執行命令 `aapt dump xmltree <path to your test package> AndroidManifest.xml`，確認是否不需要許可，並在確認輸出不包含關鍵字 `android.permission.BIND_DEVICE_ADMIN` 後再試一次。

在上傳驗證程序期間，AWS Device Farm 會使用命令 從套件中包含的 xml 檔案的 xml 剖析樹中剖析許可資訊`aapt dump xmltree <path to your package> AndroidManifest.xml`。

請確認您的應用程式是否不需要裝置管理許可。在下列範例中，套件的名稱是 app-debug.apk。

- 將您的應用程式套件複製到工作目錄，然後執行下列命令：

  ```
  $ aapt dump xmltree app-debug.apk AndroidManifest.xml
  ```

  輸出應顯示如下：

  ```
  N: android=http://schemas.android.com/apk/res/android
    E: manifest (line=2)
      A: android:versionCode(0x0101021b)=(type 0x10)0x1
      A: android:versionName(0x0101021c)="1.0" (Raw: "1.0")
      A: package="com.amazonaws.devicefarm.android.referenceapp" (Raw:
   "com.amazonaws.devicefarm.android.referenceapp")
      A: platformBuildVersionCode=(type 0x10)0x16 (Raw: "22")
      A: platformBuildVersionName="5.1.1-1819727" (Raw: "5.1.1-1819727")
      E: uses-sdk (line=7)
        A: android:minSdkVersion(0x0101020c)=(type 0x10)0xa
  ```

```
      A: android:targetSdkVersion(0x01010270)=(type 0x10)0x16
    E: uses-permission (line=11)
      A: android:name(0x01010003)="android.permission.INTERNET" (Raw:
"android.permission.INTERNET")
    E: uses-permission (line=12)
      A: android:name(0x01010003)="android.permission.CAMERA" (Raw:
"android.permission.CAMERA")
        ……
```

如果 Android 應用程式是有效的，輸出應不包含下列項目：`A: android:name(0x01010003)="android.permission.BIND_DEVICE_ADMIN" (Raw: "android.permission.BIND_DEVICE_ADMIN")`。

如需詳細資訊，請參閱AWS Device Farm 中的 Android 測試。

Android 應用程式中的某些視窗會顯示空白或黑色畫面

如果您正在測試 Android 應用程式，並注意到應用程式中的某些視窗在 Device Farm 測試的影片錄製中顯示黑色畫面，您的應用程式可能正在使用 Android FLAG_SECURE的功能。此標記（如 Android 官方文件所述）用於防止螢幕錄製工具記錄應用程式的特定時段。因此，如果視窗使用此旗標，Device Farm 的畫面錄製功能（適用於自動化和遠端存取測試）可能會顯示黑色畫面，以取代應用程式的視窗。

開發人員通常會將此標記用於應用程式中包含登入頁面等敏感資訊的頁面。如果您看到黑螢幕取代應用程式登入頁面等特定頁面的螢幕，請與您的開發人員合作，以取得未使用此標記進行測試的應用程式組建。

此外，請注意 Device Farm 仍可與具有此旗標的應用程式視窗互動。因此，如果您應用程式的登入頁面顯示為黑色畫面，您仍然可以輸入登入資料以登入應用程式（因此檢視未被FLAG_SECURE旗標封鎖的頁面）。

對 AWS Device Farm 中的 Appium Java JUnit 測試進行故障診斷

下列主題會列出在上傳 Appium Java JUnit 測試期間出現的錯誤訊息，並建議解決每個錯誤的解決方法。

> **Note**
>
> 以下說明以 Linux x86_64 和 Mac 為基礎。

APPIUM_JAVA_JUNIT_TEST_PACKAGE_UNZIP_FAILED

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 無法開啟您的測試 ZIP 檔。請確認檔案是否有效，然後再試一次。

請確認您可以正確解壓縮測試套件。在下列範例中，套件的名稱是 zip-with-dependencies.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

   ```
   $ tree .
   ```

   有效的 Appium Java JUnit 套件應產生輸出如下：

   ```
   .
   |— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
   |— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
   |— zip-with-dependencies.zip (this .zip file contains all of the items)
   `— dependency-jars  (this is the directory that contains all of your dependencies,
    built as JAR files)
         |— com.some-dependency.bar-4.1.jar
         |— com.another-dependency.thing-1.0.jar
         |— joda-time-2.7.jar
         `— log4j-1.2.14.jar
   ```

   如需詳細資訊，請參閱在 Device Farm 中自動執行 Appium 測試。

APPIUM_JAVA_JUNIT_TEST_PACKAGE_DEPENDENCY_DIR_MISSING

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 在測試套件找不到相依性 jar 目錄。請解壓縮您的測試套件，確認相依性 jar 目錄位於套件中，然後再試一次。

在下列範例中，套件的名稱是 zip-with-dependencies.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip zip-with-dependencies.zip
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

如果 Appium Java JUnit 套件是有效的，您可以在工作目錄中找到 *### jar* 目錄：

```
.
|— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|— zip-with-dependencies.zip (this .zip file contains all of the items)
`— dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |— com.some-dependency.bar-4.1.jar
      |— com.another-dependency.thing-1.0.jar
      |— joda-time-2.7.jar
      `— log4j-1.2.14.jar
```

如需詳細資訊，請參閱在 Device Farm 中自動執行 Appium 測試。

APPIUM_JAVA_JUNIT_TEST_PACKAGE_JAR_MISSING_IN_DEPENDENCY_DI

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 在相依性 jar 目錄樹狀結構中找不到 JAR 檔案。請解壓縮您的測試套件，開啟相依性 jar 目錄，確認目錄中至少有一個 JAR 檔案，然後再試一次。

在下列範例中，套件的名稱是 zip-with-dependencies.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip zip-with-dependencies.zip
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

如果 Appium Java JUnit 套件有效，您會在 dependency-*jar* s目錄中找到至少一個 jar 檔案：

```
.
|— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|— zip-with-dependencies.zip (this .zip file contains all of the items)
`— dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |— com.some-dependency.bar-4.1.jar
      |— com.another-dependency.thing-1.0.jar
      |— joda-time-2.7.jar
      `— log4j-1.2.14.jar
```

如需詳細資訊，請參閱在 Device Farm 中自動執行 Appium 測試。

APPIUM_JAVA_JUNIT_TEST_PACKAGE_TESTS_JAR_FILE_MISSING

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 在您的測試套件中找不到 *-tests.jar 檔案。請解壓縮您的測試套件，確認至少一個 *-tests.jar 檔案位於套件中，然後再試一次。

在下列範例中，套件的名稱是 zip-with-dependencies.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip zip-with-dependencies.zip
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

如果 Appium Java JUnit 套件是有效的，您可以找到至少一個 *jar* 檔案，如同我們的範例中的 *acme-android-appium-1.0-SNAPSHOT-tests.jar*。檔案的名稱可能不同，但應以 *-tests.jar* 結尾。

```
.
|— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|— zip-with-dependencies.zip (this .zip file contains all of the items)
`— dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |— com.some-dependency.bar-4.1.jar
      |— com.another-dependency.thing-1.0.jar
      |— joda-time-2.7.jar
      `— log4j-1.2.14.jar
```

如需詳細資訊，請參閱在 Device Farm 中自動執行 Appium 測試。

APPIUM_JAVA_JUNIT_TEST_PACKAGE_CLASS_FILE_MISSING_IN_TESTS_JA

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 在測試 JAR 檔案中找不到類別檔案。請解壓縮您的測試套件，反編譯測試 JAR 檔案，確認 JAR 檔案中至少有一個類別檔案，然後再試一次。

在下列範例中，套件的名稱是 zip-with-dependencies.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip zip-with-dependencies.zip
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

   您應該會找到至少一個 jar 檔案，如範例中的 *acme-android-appium-1.0-SNAPSHOT-tests.jar*。檔案的名稱可能不同，但應以 *-tests.jar* 結尾。

```
.
|— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|— zip-with-dependencies.zip (this .zip file contains all of the items)
`— dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |— com.some-dependency.bar-4.1.jar
      |— com.another-dependency.thing-1.0.jar
      |— joda-time-2.7.jar
      `— log4j-1.2.14.jar
```

3. 成功擷取檔案後，您可以透過執行下列命令，在工作目錄樹狀結構中找到至少一個類別：

```
$ tree .
```

   您應該會看到輸出，如下所示：

```
.
```

```
|— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
everything built from the ./src/test directory)
|- one-class-file.class
|- folder
|    `-another-class-file.class
|— zip-with-dependencies.zip (this .zip file contains all of the items)
`— dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |— com.some-dependency.bar-4.1.jar
      |— com.another-dependency.thing-1.0.jar
      |— joda-time-2.7.jar
      `— log4j-1.2.14.jar
```

如需詳細資訊，請參閱在 Device Farm 中自動執行 Appium 測試。

APPIUM_JAVA_JUNIT_TEST_PACKAGE_JUNIT_VERSION_VALUE_UNKNOWN

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 找不到 JUnit 版本值。請解壓縮您的測試套件，開啟相依性 jar 目錄，確認目錄中有 JUnit JAR 檔案，然後再試一次。

在下列範例中，套件的名稱是 zip-with-dependencies.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. 成功解壓縮套件後，您可以透過執行下列命令找到工作目錄樹狀結構：

   ```
   tree .
   ```

   輸出應如下所示：

   ```
   .
   ```

```
|– acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|– acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|– zip-with-dependencies.zip (this .zip file contains all of the items)
`– dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
    |– junit-4.10.jar
    |– com.some-dependency.bar-4.1.jar
    |– com.another-dependency.thing-1.0.jar
    |– joda-time-2.7.jar
    `– log4j-1.2.14.jar
```

如果 Appium Java JUnit 套件是有效的，您可以找到 JUnit 相依性檔案，類似於範例中的 *junit-4.10.jar* JAR 檔案。名稱應包含關鍵字 *junit* 及其版本編號，在此範例中為 4.10。

如需詳細資訊，請參閱在 Device Farm 中自動執行 Appium 測試。

APPIUM_JAVA_JUNIT_TEST_PACKAGE_INVALID_JUNIT_VERSION

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> JUnit 版本低於支援的最低版本 4.10。請變更 JUnit 版本，然後再試一次。

在下列範例中，套件的名稱是 zip-with-dependencies.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip zip-with-dependencies.zip
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

您應可找到 JUnit 相依性檔案，如範例中的 *junit-4.10.jar* 及其版本編號，在此範例中為 4.10：

```
.
|— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|— zip-with-dependencies.zip (this .zip file contains all of the items)
`— dependency-jars (this is the directory that contains all of your dependencies,
 built as JAR files)
      |— junit-4.10.jar
      |— com.some-dependency.bar-4.1.jar
      |— com.another-dependency.thing-1.0.jar
      |— joda-time-2.7.jar
      `— log4j-1.2.14.jar
```

> **Note**
>
> 如果測試套件中指定的 JUnit 版本低於我們支援的最低版本 4.10，則測試可能無法正確執行。

如需詳細資訊，請參閱在 Device Farm 中自動執行 Appium 測試。

故障診斷 AWS Device Farm 中的 Appium Java JUnit Web 應用程式測試

下列主題會列出在上傳 Appium Java JUnit Web 應用程式測試期間出現的錯誤訊息，並建議解決每個錯誤的解決方法。如需搭配 Device Farm 使用 Appium 的詳細資訊，請參閱 the section called “自動 Appium 測試”。

APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_UNZIP_FAILED

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 無法開啟您的測試 ZIP 檔。請確認檔案是否有效，然後再試一次。

請確認您可以正確解壓縮測試套件。在下列範例中，套件的名稱是 zip-with-dependencies.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip zip-with-dependencies.zip
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

有效的 Appium Java JUnit 套件應產生輸出如下：

```
.
|— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|— zip-with-dependencies.zip (this .zip file contains all of the items)
`— dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |— com.some-dependency.bar-4.1.jar
      |— com.another-dependency.thing-1.0.jar
      |— joda-time-2.7.jar
      `— log4j-1.2.14.jar
```

APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_DEPENDENCY_DIR_MISSING

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 在測試套件找不到相依性 jar 目錄。請解壓縮您的測試套件，確認相依性 jar 目錄位於套件中，然後再試一次。

在下列範例中，套件的名稱是 zip-with-dependencies.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip zip-with-dependencies.zip
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

如果 Appium Java JUnit 套件是有效的，您可以在工作目錄中找到 *### jar* 目錄：

```
.
|— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|— zip-with-dependencies.zip (this .zip file contains all of the items)
`— dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |— com.some-dependency.bar-4.1.jar
      |— com.another-dependency.thing-1.0.jar
      |— joda-time-2.7.jar
      `— log4j-1.2.14.jar
```

APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_JAR_MISSING_IN_DEPENDEN

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 在相依性 jar 目錄樹狀結構中找不到 JAR 檔案。請解壓縮您的測試套件，開啟相依性 jar 目錄，確認目錄中至少有一個 JAR 檔案，然後再試一次。

在下列範例中，套件的名稱是 zip-with-dependencies.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip zip-with-dependencies.zip
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

如果 Appium Java JUnit 套件有效，您將在 dependency-*jar* s目錄中找到至少一個 jar 檔案：

```
.
|— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|— zip-with-dependencies.zip (this .zip file contains all of the items)
`— dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |— com.some-dependency.bar-4.1.jar
      |— com.another-dependency.thing-1.0.jar
      |— joda-time-2.7.jar
      `— log4j-1.2.14.jar
```

APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_TESTS_JAR_FILE_MISSING

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 在您的測試套件中找不到 *-tests.jar 檔案。請解壓縮您的測試套件，確認至少一個 *-tests.jar 檔案位於套件中，然後再試一次。

在下列範例中，套件的名稱是 zip-with-dependencies.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip zip-with-dependencies.zip
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

如果 Appium Java JUnit 套件是有效的，您可以找到至少一個 *jar* 檔案，如同我們的範例中的 *acme-android-appium-1.0-SNAPSHOT-tests.jar*。檔案的名稱可能不同，但應以 *-tests.jar* 結尾。

```
.
|– acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|– acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|– zip-with-dependencies.zip (this .zip file contains all of the items)
`– dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |– com.some-dependency.bar-4.1.jar
      |– com.another-dependency.thing-1.0.jar
      |– joda-time-2.7.jar
      `– log4j-1.2.14.jar
```

APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_CLASS_FILE_MISSING_IN_TES

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 在測試 JAR 檔案中找不到類別檔案。請解壓縮您的測試套件，反編譯測試 JAR 檔案，確認 JAR 檔案中至少有一個類別檔案，然後再試一次。

在下列範例中，套件的名稱是 zip-with-dependencies.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

   ```
   $ tree .
   ```

   您應該會找到至少一個 jar 檔案，如範例中的 *acme-android-appium-1.0-SNAPSHOT-tests.jar*。檔案的名稱可能不同，但應以 *-tests.jar* 結尾。

```
.
|— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|— zip-with-dependencies.zip (this .zip file contains all of the items)
`— dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |— com.some-dependency.bar-4.1.jar
      |— com.another-dependency.thing-1.0.jar
      |— joda-time-2.7.jar
      `— log4j-1.2.14.jar
```

3. 成功擷取檔案後，您可以透過執行下列命令，在工作目錄樹狀結構中找到至少一個類別：

```
$ tree .
```

您應該會看到輸出，如下所示：

```
.
|— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
everything built from the ./src/test directory)
|- one-class-file.class
|- folder
|    `-another-class-file.class
|— zip-with-dependencies.zip (this .zip file contains all of the items)
`— dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |— com.some-dependency.bar-4.1.jar
      |— com.another-dependency.thing-1.0.jar
      |— joda-time-2.7.jar
      `— log4j-1.2.14.jar
```

APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_JUNIT_VERSION_VALUE_UNK

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 找不到 JUnit 版本值。請解壓縮您的測試套件，開啟相依性 jar 目錄，確認目錄中有 JUnit JAR 檔案，然後再試一次。

在下列範例中，套件的名稱是 zip-with-dependencies.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip zip-with-dependencies.zip
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到工作目錄樹狀結構：

```
tree .
```

輸出應如下所示：

```
.
|— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|— zip-with-dependencies.zip (this .zip file contains all of the items)
`— dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |— junit-4.10.jar
      |— com.some-dependency.bar-4.1.jar
      |— com.another-dependency.thing-1.0.jar
      |— joda-time-2.7.jar
      `— log4j-1.2.14.jar
```

如果 Appium Java JUnit 套件是有效的，您可以找到 JUnit 相依性檔案，類似於範例中的 *junit-4.10.jar* JAR 檔案。名稱應包含關鍵字 *junit* 及其版本編號，在此範例中為 4.10。

APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_INVALID_JUNIT_VERSION

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> JUnit 版本低於支援的最低版本 4.10。請變更 JUnit 版本，然後再試一次。

在下列範例中，套件的名稱是 zip-with-dependencies.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip zip-with-dependencies.zip
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

您應可找到 JUnit 相依性檔案，如範例中的 *junit-4.10.jar* 及其版本編號，在此範例中為 4.10：

```
.
|— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|— zip-with-dependencies.zip (this .zip file contains all of the items)
`— dependency-jars (this is the directory that contains all of your dependencies,
 built as JAR files)
      |— junit-4.10.jar
      |— com.some-dependency.bar-4.1.jar
      |— com.another-dependency.thing-1.0.jar
      |— joda-time-2.7.jar
      `— log4j-1.2.14.jar
```

> **Note**
>
> 如果測試套件中指定的 JUnit 版本低於我們支援的最低版本 4.10，則測試可能無法正確執行。

如需詳細資訊，請參閱在 Device Farm 中自動執行 Appium 測試。

對 AWS Device Farm 中的 Appium Java TestNG 測試進行故障診斷

下列主題會列出在上傳 Appium Java TestNG 測試期間出現的錯誤訊息，並建議解決每個錯誤的解決方法。

> **Note**
>
> 以下說明以 Linux x86_64 和 Mac 為基礎。

APPIUM_JAVA_TESTNG_TEST_PACKAGE_UNZIP_FAILED

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 無法開啟您的測試 ZIP 檔。請確認檔案是否有效，然後再試一次。

請確認您可以正確解壓縮測試套件。在下列範例中，套件的名稱是 zip-with-dependencies.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

   ```
   $ tree .
   ```

   有效的 Appium Java JUnit 套件應產生輸出如下：

   ```
   .
   |— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
   |— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
   |— zip-with-dependencies.zip (this .zip file contains all of the items)
   `— dependency-jars  (this is the directory that contains all of your dependencies,
    built as JAR files)
        |— com.some-dependency.bar-4.1.jar
   ```

```
      |— com.another-dependency.thing-1.0.jar
      |— joda-time-2.7.jar
      `— log4j-1.2.14.jar
```

如需詳細資訊，請參閱在 Device Farm 中自動執行 Appium 測試。

APPIUM_JAVA_TESTNG_TEST_PACKAGE_DEPENDENCY_DIR_MISSING

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 在測試套件找不到 dependency-jars 目錄。請解壓縮您的測試套件，確認 dependency-jars 目錄位於套件中，然後再試一次。

在下列範例中，套件的名稱是 zip-with-dependencies.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip zip-with-dependencies.zip
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

如果 Appium Java JUnit 套件是有效的，您可以在工作目錄中找到 *### jar* 目錄。

```
.
|— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|— zip-with-dependencies.zip (this .zip file contains all of the items)
`— dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |— com.some-dependency.bar-4.1.jar
      |— com.another-dependency.thing-1.0.jar
      |— joda-time-2.7.jar
      `— log4j-1.2.14.jar
```

如需詳細資訊，請參閱在 Device Farm 中自動執行 Appium 測試。

APPIUM_JAVA_TESTNG_TEST_PACKAGE_JAR_MISSING_IN_DEPENDENCY_

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 在相依性 jar 目錄樹狀結構中找不到 JAR 檔案。請解壓縮您的測試套件，開啟相依性 jar 目錄，確認目錄中至少有一個 JAR 檔案，然後再試一次。

在下列範例中，套件的名稱是 zip-with-dependencies.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip zip-with-dependencies.zip
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

如果 Appium Java JUnit 套件有效，您會在 dependency-*jar* s目錄中找到至少一個 jar 檔案。

```
.
|— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|— zip-with-dependencies.zip (this .zip file contains all of the items)
`— dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |— com.some-dependency.bar-4.1.jar
      |— com.another-dependency.thing-1.0.jar
      |— joda-time-2.7.jar
      `— log4j-1.2.14.jar
```

如需詳細資訊，請參閱在 Device Farm 中自動執行 Appium 測試。

APPIUM_JAVA_TESTNG_TEST_PACKAGE_TESTS_JAR_FILE_MISSING

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 在您的測試套件中找不到 *-tests.jar 檔案。請解壓縮您的測試套件，確認至少一個 *-tests.jar 檔案位於套件中，然後再試一次。

在下列範例中，套件的名稱是 zip-with-dependencies.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

   ```
   $ tree .
   ```

   如果 Appium Java JUnit 套件是有效的，您可以找到至少一個 *jar* 檔案，如同我們的範例中的 *acme-android-appium-1.0-SNAPSHOT-tests.jar*。檔案的名稱可能不同，但應以 *-tests.jar* 結尾。

   ```
   .
   |— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
   |— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
   |— zip-with-dependencies.zip (this .zip file contains all of the items)
   `— dependency-jars  (this is the directory that contains all of your dependencies,
    built as JAR files)
         |— com.some-dependency.bar-4.1.jar
         |— com.another-dependency.thing-1.0.jar
         |— joda-time-2.7.jar
         `— log4j-1.2.14.jar
   ```

   如需詳細資訊，請參閱在 Device Farm 中自動執行 Appium 測試。

APPIUM_JAVA_TESTNG_TEST_PACKAGE_CLASS_FILE_MISSING_IN_TESTS

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 在測試 JAR 檔案中找不到類別檔案。請解壓縮您的測試套件，反編譯測試 JAR 檔案，確認 JAR 檔案中至少有一個類別檔案，然後再試一次。

在下列範例中，套件的名稱是 zip-with-dependencies.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip zip-with-dependencies.zip
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

您應該會找到至少一個 jar 檔案，如範例中的 *acme-android-appium-1.0-SNAPSHOT-tests.jar*。檔案的名稱可能不同，但應以 *-tests.jar* 結尾。

```
.
|— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|— zip-with-dependencies.zip (this .zip file contains all of the items)
`— dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |— com.some-dependency.bar-4.1.jar
      |— com.another-dependency.thing-1.0.jar
      |— joda-time-2.7.jar
      `— log4j-1.2.14.jar
```

3. 若要從 JAR 檔案中擷取檔案，您可執行以下命令：

```
$ jar xf acme-android-appium-1.0-SNAPSHOT-tests.jar
```

4. 成功擷取檔案後，請執行下列命令：

```
$ tree .
```

您應可在工作目錄樹狀結構中找到至少一個類別：

```
.
|— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
everything built from the ./src/test directory)
|- one-class-file.class
|- folder
|    `— another-class-file.class
|— zip-with-dependencies.zip (this .zip file contains all of the items)
`— dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |— com.some-dependency.bar-4.1.jar
      |— com.another-dependency.thing-1.0.jar
      |— joda-time-2.7.jar
      `— log4j-1.2.14.jar
```

如需詳細資訊，請參閱在 Device Farm 中自動執行 Appium 測試。

故障診斷 AWS Device Farm 中的 Appium Java TestNG Web 應用程式

下列主題會列出在上傳 Appium Java TestNG Web 應用程式測試期間出現的錯誤訊息，並建議解決每個錯誤的解決方法。

APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_UNZIP_FAILED

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 無法開啟您的測試 ZIP 檔。請確認檔案是否有效，然後再試一次。

請確認您可以正確解壓縮測試套件。在下列範例中，套件的名稱是 zip-with-dependencies.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip zip-with-dependencies.zip
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

有效的 Appium Java JUnit 套件應產生輸出如下：

```
.
|— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|— zip-with-dependencies.zip (this .zip file contains all of the items)
`— dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |— com.some-dependency.bar-4.1.jar
      |— com.another-dependency.thing-1.0.jar
      |— joda-time-2.7.jar
      `— log4j-1.2.14.jar
```

如需詳細資訊，請參閱在 Device Farm 中自動執行 Appium 測試。

APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_DEPENDENCY_DIR_MISSIN

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 在測試套件找不到相依性 jar 目錄。請解壓縮您的測試套件，確認相依性 jar 目錄位於套件中，然後再試一次。

在下列範例中，套件的名稱是 zip-with-dependencies.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip zip-with-dependencies.zip
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

如果 Appium Java JUnit 套件是有效的，您可以在工作目錄中找到 *### jar* 目錄。

```
.
|— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|— zip-with-dependencies.zip (this .zip file contains all of the items)
`— dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |— com.some-dependency.bar-4.1.jar
      |— com.another-dependency.thing-1.0.jar
      |— joda-time-2.7.jar
      `— log4j-1.2.14.jar
```

如需詳細資訊，請參閱在 Device Farm 中自動執行 Appium 測試。

APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_JAR_MISSING_IN_DEPEND

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 在相依性 jar 目錄樹狀結構中找不到 JAR 檔案。請解壓縮您的測試套件，開啟相依性 jar 目錄，確認目錄中至少有一個 JAR 檔案，然後再試一次。

在下列範例中，套件的名稱是 zip-with-dependencies.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip zip-with-dependencies.zip
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

如果 Appium Java JUnit 套件有效，您將在 dependency-*jar* s目錄中找到至少一個 jar 檔案。

```
.
|— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|— zip-with-dependencies.zip (this .zip file contains all of the items)
`— dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |— com.some-dependency.bar-4.1.jar
      |— com.another-dependency.thing-1.0.jar
      |— joda-time-2.7.jar
      `— log4j-1.2.14.jar
```

如需詳細資訊，請參閱在 Device Farm 中自動執行 Appium 測試。

APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_TESTS_JAR_FILE_MISSING

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 在您的測試套件中找不到 *-tests.jar 檔案。請解壓縮您的測試套件，確認至少一個 *-tests.jar 檔案位於套件中，然後再試一次。

在下列範例中，套件的名稱是 zip-with-dependencies.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip zip-with-dependencies.zip
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

如果 Appium Java JUnit 套件是有效的，您可以找到至少一個 *jar* 檔案，如同我們的範例中的 *acme-android-appium-1.0-SNAPSHOT-tests.jar*。檔案的名稱可能不同，但應以 *-tests.jar* 結尾。

```
.
|– acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|– acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|– zip-with-dependencies.zip (this .zip file contains all of the items)
`– dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |– com.some-dependency.bar-4.1.jar
      |– com.another-dependency.thing-1.0.jar
      |– joda-time-2.7.jar
      `– log4j-1.2.14.jar
```

如需詳細資訊，請參閱在 Device Farm 中自動執行 Appium 測試。

APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_CLASS_FILE_MISSING_IN_T

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 在測試 JAR 檔案中找不到類別檔案。請解壓縮您的測試套件，反編譯測試 JAR 檔案，確認 JAR 檔案中至少有一個類別檔案，然後再試一次。

在下列範例中，套件的名稱是 zip-with-dependencies.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip zip-with-dependencies.zip
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

您應該會找到至少一個 jar 檔案，如範例中的 *acme-android-appium-1.0-SNAPSHOT-tests.jar*。檔案的名稱可能不同，但應以 *-tests.jar* 結尾。

```
.
|— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|— zip-with-dependencies.zip (this .zip file contains all of the items)
`— dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |— com.some-dependency.bar-4.1.jar
      |— com.another-dependency.thing-1.0.jar
      |— joda-time-2.7.jar
      `— log4j-1.2.14.jar
```

3. 若要從 JAR 檔案中擷取檔案，您可執行以下命令：

```
$ jar xf acme-android-appium-1.0-SNAPSHOT-tests.jar
```

4. 成功擷取檔案後，請執行下列命令：

```
$ tree .
```

您應可在工作目錄樹狀結構中找到至少一個類別：

```
.
|— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
everything built from the ./src/test directory)
|- one-class-file.class
|- folder
|    `— another-class-file.class
|— zip-with-dependencies.zip (this .zip file contains all of the items)
`— dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |— com.some-dependency.bar-4.1.jar
```

```
      |— com.another-dependency.thing-1.0.jar
      |— joda-time-2.7.jar
      `— log4j-1.2.14.jar
```

如需詳細資訊，請參閱在 Device Farm 中自動執行 Appium 測試。

對 AWS Device Farm 中的 Appium Python 測試進行故障診斷

下列主題會列出在上傳 Appium Python 測試期間出現的錯誤訊息，並建議解決每個錯誤的解決方法。

APPIUM_PYTHON_TEST_PACKAGE_UNZIP_FAILED

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 無法開啟您的 Appium ZIP 檔。請確認檔案是否有效，然後再試一次。

請確認您可以正確解壓縮測試套件。在下列範例中，套件的名稱是 test_bundle.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip test_bundle.zip
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

有效的 Appium Python 套件應產生輸出如下：

```
.
|-- requirements.txt
|-- test_bundle.zip
|-- tests (directory)
|       `-- test_unittest.py
`-- wheelhouse (directory)
        |-- Appium_Python_Client-0.20-cp27-none-any.whl
        |-- py-1.4.31-py2.py3-none-any.whl
        |-- pytest-2.9.0-py2.py3-none-any.whl
```

```
        |-- selenium-2.52.0-cp27-none-any.whl
        `-- wheel-0.26.0-py2.py3-none-any.whl
```

如需詳細資訊，請參閱在 Device Farm 中自動執行 Appium 測試。

APPIUM_PYTHON_TEST_PACKAGE_DEPENDENCY_WHEEL_MISSING

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 在 wheelhouse 目錄樹狀結構中找不到相依性 wheel 檔案。請解壓縮您的測試套件，開啟 wheelhouse 目錄，確認目錄中至少有一個 wheel 檔案，然後再試一次。

請確認您可以正確解壓縮測試套件。在下列範例中，套件的名稱是 test_bundle.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip test_bundle.zip
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

如果 Appium Python 套件是有效的，您可以找到至少一個 *.whl* 檔案，類似於 *wheelhouse* 目錄中反白顯示的檔案。

```
.
|-- requirements.txt
|-- test_bundle.zip
|-- tests (directory)
|       `-- test_unittest.py
`-- wheelhouse (directory)
        |-- Appium_Python_Client-0.20-cp27-none-any.whl
        |-- py-1.4.31-py2.py3-none-any.whl
        |-- pytest-2.9.0-py2.py3-none-any.whl
        |-- selenium-2.52.0-cp27-none-any.whl
        `-- wheel-0.26.0-py2.py3-none-any.whl
```

如需詳細資訊，請參閱在 Device Farm 中自動執行 Appium 測試。

APPIUM_PYTHON_TEST_PACKAGE_INVALID_PLATFORM

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 至少有一個 wheel 檔案指定了我們不支援的平台。請解壓縮您的測試套件後開啟 wheelhouse 目錄，確認 wheel 檔案的名稱以 -any.whl 或 -linux_x86_64.whl 結尾，然後再試一次。

請確認您可以正確解壓縮測試套件。在下列範例中，套件的名稱是 test_bundle.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip test_bundle.zip
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

如果 Appium Python 套件是有效的，您可以找到至少一個 *.whl* 檔案，類似於 *wheelhouse* 目錄中反白顯示的檔案。檔案的名稱可能不同，但應以 *-any.whl* 或 *-linux_x86_64.whl* 結尾 (用於指定平台)。不支援 windows 等任何其他平台。

```
.
|-- requirements.txt
|-- test_bundle.zip
|-- tests (directory)
|       `-- test_unittest.py
`-- wheelhouse (directory)
        |-- Appium_Python_Client-0.20-cp27-none-any.whl
        |-- py-1.4.31-py2.py3-none-any.whl
        |-- pytest-2.9.0-py2.py3-none-any.whl
        |-- selenium-2.52.0-cp27-none-any.whl
        `-- wheel-0.26.0-py2.py3-none-any.whl
```

如需詳細資訊，請參閱在 Device Farm 中自動執行 Appium 測試。

APPIUM_PYTHON_TEST_PACKAGE_TEST_DIR_MISSING

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 在測試套件找不到測試目錄。請解壓縮您的測試套件，確認測試目錄位於套件中，然後再試一次。

請確認您可以正確解壓縮測試套件。在下列範例中，套件的名稱是 test_bundle.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

   ```
   $ unzip test_bundle.zip
   ```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

   ```
   $ tree .
   ```

   如果 Appium Python 套件是有效的，您可以在工作目錄中找到##目錄。

   ```
   .
   |-- requirements.txt
   |-- test_bundle.zip
   |-- tests (directory)
   |       `-- test_unittest.py
   `-- wheelhouse (directory)
           |-- Appium_Python_Client-0.20-cp27-none-any.whl
           |-- py-1.4.31-py2.py3-none-any.whl
           |-- pytest-2.9.0-py2.py3-none-any.whl
           |-- selenium-2.52.0-cp27-none-any.whl
           `-- wheel-0.26.0-py2.py3-none-any.whl
   ```

   如需詳細資訊，請參閱在 Device Farm 中自動執行 Appium 測試。

APPIUM_PYTHON_TEST_PACKAGE_INVALID_TEST_FILE_NAME

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 在測試目錄樹狀結構中找不到有效的測試檔案。請解壓縮您的測試套件，開啟測試目錄，確認至少一個檔案的名稱以關鍵字「test」開頭或結尾，然後再試一次。

請確認您可以正確解壓縮測試套件。在下列範例中，套件的名稱是 test_bundle.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip test_bundle.zip
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

如果 Appium Python 套件是有效的，您可以在工作目錄中找到##目錄。檔案的名稱可能不同，但應以 *test_* 開頭，或以 *_test.py* 結尾。

```
.
|-- requirements.txt
|-- test_bundle.zip
|-- tests (directory)
|       `-- test_unittest.py
`-- wheelhouse (directory)
        |-- Appium_Python_Client-0.20-cp27-none-any.whl
        |-- py-1.4.31-py2.py3-none-any.whl
        |-- pytest-2.9.0-py2.py3-none-any.whl
        |-- selenium-2.52.0-cp27-none-any.whl
        `-- wheel-0.26.0-py2.py3-none-any.whl
```

如需詳細資訊，請參閱在 Device Farm 中自動執行 Appium 測試。

APPIUM_PYTHON_TEST_PACKAGE_REQUIREMENTS_TXT_FILE_MISSING

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 在測試套件找不到 requirements.txt 檔案。請解壓縮您的測試套件，確認 requirements.txt 檔案位於套件中，然後再試一次。

請確認您可以正確解壓縮測試套件。在下列範例中，套件的名稱是 test_bundle.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip test_bundle.zip
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

如果 Appium Python 套件是有效的，您可以在工作目錄中找到 *requirements.txt* 檔案。

```
.
|-- requirements.txt
|-- test_bundle.zip
|-- tests (directory)
|       `-- test_unittest.py
`-- wheelhouse (directory)
        |-- Appium_Python_Client-0.20-cp27-none-any.whl
        |-- py-1.4.31-py2.py3-none-any.whl
        |-- pytest-2.9.0-py2.py3-none-any.whl
        |-- selenium-2.52.0-cp27-none-any.whl
        `-- wheel-0.26.0-py2.py3-none-any.whl
```

如需詳細資訊，請參閱在 Device Farm 中自動執行 Appium 測試。

APPIUM_PYTHON_TEST_PACKAGE_INVALID_PYTEST_VERSION

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> pytest 版本低於支援的最低版本 2.8.0。請變更 requirements.txt 中的 pytest 版本，然後再試一次。

請確認您可以正確解壓縮測試套件。在下列範例中，套件的名稱是 test_bundle.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip test_bundle.zip
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

   您應該會在工作目錄中找到 *requirements.txt* 檔案。

```
.
|-- requirements.txt
|-- test_bundle.zip
|-- tests (directory)
|       `--test_unittest.py
`-- wheelhouse (directory)
        |-- Appium_Python_Client-0.20-cp27-none-any.whl
        |-- py-1.4.31-py2.py3-none-any.whl
        |-- pytest-2.9.0-py2.py3-none-any.whl
        |-- selenium-2.52.0-cp27-none-any.whl
        `-- wheel-0.26.0-py2.py3-none-any.whl
```

3. 若要取得 pytest 版本，您可以執行下列命令：

```
$ grep "pytest" requirements.txt
```

   輸出應顯示如下：

```
pytest==2.9.0
```

   系統會顯示 pytest 版本，在此範例中為 2.9.0。如果 Appium Python 套件是有效的，則 pytest 版本應高於或等於 2.8.0。

如需詳細資訊，請參閱在 Device Farm 中自動執行 Appium 測試。

APPIUM_PYTHON_TEST_PACKAGE_INSTALL_DEPENDENCY_WHEELS_FAIL

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 無法安裝相依性 wheel。請解壓縮您的測試套件，開啟 requirements.txt 檔案和 wheelhouse 目錄，確認 requirements.txt 檔案中指定的相依性 wheel 完全符合 wheelhouse 目錄中的相依性 wheel，然後再試一次。

我們強烈建議您設定 Python virtualenv 來封裝測試。以下是使用 Python virtualenv 建立虛擬環境然後啟用的範例流程：

```
$ virtualenv workspace
$ cd workspace
$ source bin/activate
```

請確認您可以正確解壓縮測試套件。在下列範例中，套件的名稱是 test_bundle.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

   ```
   $ unzip test_bundle.zip
   ```

2. 若要測試安裝 wheel 檔案，您可以執行下列命令：

   ```
   $ pip install --use-wheel --no-index --find-links=./wheelhouse --requirement=./
   requirements.txt
   ```

   有效的 Appium Python 套件應產生輸出如下：

   ```
   Ignoring indexes: https://pypi.python.org/simple
   Collecting Appium-Python-Client==0.20 (from -r ./requirements.txt (line 1))
   Collecting py==1.4.31 (from -r ./requirements.txt (line 2))
   Collecting pytest==2.9.0 (from -r ./requirements.txt (line 3))
   Collecting selenium==2.52.0 (from -r ./requirements.txt (line 4))
   ```

```
Collecting wheel==0.26.0 (from -r ./requirements.txt (line 5))
Installing collected packages: selenium, Appium-Python-Client, py, pytest, wheel
  Found existing installation: wheel 0.29.0
    Uninstalling wheel-0.29.0:
      Successfully uninstalled wheel-0.29.0
Successfully installed Appium-Python-Client-0.20 py-1.4.31 pytest-2.9.0
 selenium-2.52.0 wheel-0.26.0
```

3. 若要停用虛擬環境，您可以執行下列命令：

```
$ deactivate
```

如需詳細資訊，請參閱在 Device Farm 中自動執行 Appium 測試。

APPIUM_PYTHON_TEST_PACKAGE_PYTEST_COLLECT_FAILED

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 無法在測試目錄中收集測試。請解壓縮您的測試套件，執行命令 `py.test --collect-only <path to your tests directory>`，確認測試套件是否有效，並在命令未列印任何錯誤後再試一次。

我們強烈建議您設定 Python virtualenv 來封裝測試。以下是使用 Python virtualenv 建立虛擬環境然後啟用的範例流程：

```
$ virtualenv workspace
$ cd workspace
$ source bin/activate
```

請確認您可以正確解壓縮測試套件。在下列範例中，套件的名稱是 test_bundle.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip test_bundle.zip
```

2. 若要安裝 wheel 檔案，您可以執行下列命令：

```
$ pip install --use-wheel --no-index --find-links=./wheelhouse --requirement=./
requirements.txt
```

3. 若要收集測試，您可以執行下列命令：

```
$ py.test --collect-only tests
```

有效的 Appium Python 套件應產生輸出如下：

```
==================== test session starts ====================
platform darwin -- Python 2.7.11, pytest-2.9.0, py-1.4.31, pluggy-0.3.1
rootdir: /Users/zhena/Desktop/Ios/tests, inifile:
collected 1 items
<Module 'test_unittest.py'>
  <UnitTestCase 'DeviceFarmAppiumWebTests'>
    <TestCaseFunction 'test_devicefarm'>

==================== no tests ran in 0.11 seconds ====================
```

4. 若要停用虛擬環境，您可以執行下列命令：

```
$ deactivate
```

如需詳細資訊，請參閱在 Device Farm 中自動執行 Appium 測試。

APPIUM_PYTHON_TEST_PACKAGE_DEPENDENCY_WHEELS_INSUFFICIENT

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 我們在 wheelhouse 目錄中找不到足夠的 wheel 相依性。請解壓縮您的測試套件，然後開啟 wheelhouse 目錄。確認您已在 requirements.txt 檔案中指定所有 wheel 相依性。

請確認您可以正確解壓縮測試套件。在下列範例中，套件的名稱是 test_bundle.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip test_bundle.zip
```

2. 檢查 *requirements.txt* 檔案的長度，以及 wheelhouse 目錄中的 *.whl* 相依檔案數目：

```
$ cat requirements.txt | egrep "." |wc -l
    12
$ ls wheelhouse/ | egrep ".+\.whl" | wc -l
    11
```

如果 *.whl* 相依檔案的數量少於您 *requirements.txt* 檔案中非空白資料列的數量，則您需要確保下列事項：

- 有一個 *.whl* 相依檔案，對應至 *requirements.txt* 檔案中的每一列。
- requirements.*txt* 檔案中沒有包含相依性套件名稱以外資訊的其他行。
- requirements.*txt* 檔案的多行中不會複製任何相依性名稱，因此檔案中的兩行可能對應至一個 *.whl* 相依檔案。

AWS Device Farm 不支援 *requirements.txt* 檔案中未直接對應至相依性套件的行，例如指定`pip install`命令全域選項的行。如需全域選項清單，請參閱需求檔案格式。

如需詳細資訊，請參閱在 Device Farm 中自動執行 Appium 測試。

故障診斷 AWS Device Farm 中的 Appium Python Web 應用程式測試

下列主題會列出在上傳 Appium Python Web 應用程式測試期間出現的錯誤訊息，並建議解決每個錯誤的解決方法。

APPIUM_WEB_PYTHON_TEST_PACKAGE_UNZIP_FAILED

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 無法開啟您的 Appium ZIP 檔。請確認檔案是否有效，然後再試一次。

請確認您可以正確解壓縮測試套件。在下列範例中，套件的名稱是 test_bundle.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip test_bundle.zip
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

有效的 Appium Python 套件應產生輸出如下：

```
.
|-- requirements.txt
|-- test_bundle.zip
|-- tests (directory)
|       `-- test_unittest.py
`-- wheelhouse (directory)
        |-- Appium_Python_Client-0.20-cp27-none-any.whl
        |-- py-1.4.31-py2.py3-none-any.whl
        |-- pytest-2.9.0-py2.py3-none-any.whl
        |-- selenium-2.52.0-cp27-none-any.whl
        `-- wheel-0.26.0-py2.py3-none-any.whl
```

如需詳細資訊，請參閱在 Device Farm 中自動執行 Appium 測試。

APPIUM_WEB_PYTHON_TEST_PACKAGE_DEPENDENCY_WHEEL_MISSING

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 在 wheelhouse 目錄樹狀結構中找不到相依性 wheel 檔案。請解壓縮您的測試套件，開啟 wheelhouse 目錄，確認目錄中至少有一個 wheel 檔案，然後再試一次。

請確認您可以正確解壓縮測試套件。在下列範例中，套件的名稱是 test_bundle.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip test_bundle.zip
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

如果 Appium Python 套件是有效的，您可以找到至少一個 *.whl* 檔案，類似於 *wheelhouse* 目錄中反白顯示的檔案。

```
.
|-- requirements.txt
|-- test_bundle.zip
|-- tests (directory)
|       `-- test_unittest.py
`-- wheelhouse (directory)
        |-- Appium_Python_Client-0.20-cp27-none-any.whl
        |-- py-1.4.31-py2.py3-none-any.whl
        |-- pytest-2.9.0-py2.py3-none-any.whl
        |-- selenium-2.52.0-cp27-none-any.whl
        `-- wheel-0.26.0-py2.py3-none-any.whl
```

如需詳細資訊，請參閱在 Device Farm 中自動執行 Appium 測試。

APPIUM_WEB_PYTHON_TEST_PACKAGE_INVALID_PLATFORM

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 至少有一個 wheel 檔案指定了我們不支援的平台。請解壓縮您的測試套件後開啟 wheelhouse 目錄，確認 wheel 檔案的名稱以 -any.whl 或 -linux_x86_64.whl 結尾，然後再試一次。

請確認您可以正確解壓縮測試套件。在下列範例中，套件的名稱是 test_bundle.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip test_bundle.zip
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

如果 Appium Python 套件是有效的，您可以找到至少一個 *.whl* 檔案，類似於 *wheelhouse* 目錄中反白顯示的檔案。檔案的名稱可能不同，但應以 *-any.whl* 或 *-linux_x86_64.whl* 結尾 (用於指定平台)。不支援 windows 等任何其他平台。

```
.
|-- requirements.txt
|-- test_bundle.zip
|-- tests (directory)
|       `-- test_unittest.py
`-- wheelhouse (directory)
        |-- Appium_Python_Client-0.20-cp27-none-any.whl
        |-- py-1.4.31-py2.py3-none-any.whl
        |-- pytest-2.9.0-py2.py3-none-any.whl
        |-- selenium-2.52.0-cp27-none-any.whl
        `-- wheel-0.26.0-py2.py3-none-any.whl
```

如需詳細資訊，請參閱在 Device Farm 中自動執行 Appium 測試。

APPIUM_WEB_PYTHON_TEST_PACKAGE_TEST_DIR_MISSING

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 在測試套件找不到測試目錄。請解壓縮您的測試套件，確認測試目錄位於套件中，然後再試一次。

請確認您可以正確解壓縮測試套件。在下列範例中，套件的名稱是 test_bundle.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip test_bundle.zip
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

如果 Appium Python 套件是有效的，您可以在工作目錄中找到##目錄。

```
.
|-- requirements.txt
|-- test_bundle.zip
|-- tests (directory)
|       `-- test_unittest.py
`-- wheelhouse (directory)
        |-- Appium_Python_Client-0.20-cp27-none-any.whl
        |-- py-1.4.31-py2.py3-none-any.whl
        |-- pytest-2.9.0-py2.py3-none-any.whl
        |-- selenium-2.52.0-cp27-none-any.whl
        `-- wheel-0.26.0-py2.py3-none-any.whl
```

如需詳細資訊，請參閱在 Device Farm 中自動執行 Appium 測試。

APPIUM_WEB_PYTHON_TEST_PACKAGE_INVALID_TEST_FILE_NAME

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 在測試目錄樹狀結構中找不到有效的測試檔案。請解壓縮您的測試套件，開啟測試目錄，確認至少一個檔案的名稱以關鍵字「test」開頭或結尾，然後再試一次。

請確認您可以正確解壓縮測試套件。在下列範例中，套件的名稱是 test_bundle.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip test_bundle.zip
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

如果 Appium Python 套件是有效的，您可以在工作目錄中找到##目錄。檔案的名稱可能不同，但應以 *test_* 開頭，或以 *_test.py* 結尾。

```
.
|-- requirements.txt
|-- test_bundle.zip
|-- tests (directory)
|       `-- test_unittest.py
`-- wheelhouse (directory)
        |-- Appium_Python_Client-0.20-cp27-none-any.whl
        |-- py-1.4.31-py2.py3-none-any.whl
        |-- pytest-2.9.0-py2.py3-none-any.whl
        |-- selenium-2.52.0-cp27-none-any.whl
        `-- wheel-0.26.0-py2.py3-none-any.whl
```

如需詳細資訊，請參閱在 Device Farm 中自動執行 Appium 測試。

APPIUM_WEB_PYTHON_TEST_PACKAGE_REQUIREMENTS_TXT_FILE_MISS

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 在測試套件找不到 requirements.txt 檔案。請解壓縮您的測試套件，確認 requirements.txt 檔案位於套件中，然後再試一次。

請確認您可以正確解壓縮測試套件。在下列範例中，套件的名稱是 test_bundle.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip test_bundle.zip
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

如果 Appium Python 套件是有效的，您可以在工作目錄中找到 *requirements.txt* 檔案。

```
.
|-- requirements.txt
|-- test_bundle.zip
|-- tests (directory)
|       `-- test_unittest.py
`-- wheelhouse (directory)
        |-- Appium_Python_Client-0.20-cp27-none-any.whl
        |-- py-1.4.31-py2.py3-none-any.whl
        |-- pytest-2.9.0-py2.py3-none-any.whl
        |-- selenium-2.52.0-cp27-none-any.whl
        `-- wheel-0.26.0-py2.py3-none-any.whl
```

如需詳細資訊，請參閱在 Device Farm 中自動執行 Appium 測試。

APPIUM_WEB_PYTHON_TEST_PACKAGE_INVALID_PYTEST_VERSION

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> pytest 版本低於支援的最低版本 2.8.0。請變更 requirements.txt 中的 pytest 版本，然後再試一次。

請確認您可以正確解壓縮測試套件。在下列範例中，套件的名稱是 test_bundle.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

   ```
   $ unzip test_bundle.zip
   ```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

   ```
   $ tree .
   ```

   您應可在工作目錄中找到 *requirement.txt* 檔案。

   ```
   .
   |-- requirements.txt
   |-- test_bundle.zip
   ```

```
|-- tests (directory)
|       `--test_unittest.py
`-- wheelhouse (directory)
        |-- Appium_Python_Client-0.20-cp27-none-any.whl
        |-- py-1.4.31-py2.py3-none-any.whl
        |-- pytest-2.9.0-py2.py3-none-any.whl
        |-- selenium-2.52.0-cp27-none-any.whl
        `-- wheel-0.26.0-py2.py3-none-any.whl
```

3. 若要取得 pytest 版本，您可以執行下列命令：

```
$ grep "pytest" requirements.txt
```

輸出應顯示如下：

```
pytest==2.9.0
```

系統會顯示 pytest 版本，在此範例中為 2.9.0。如果 Appium Python 套件是有效的，則 pytest 版本應高於或等於 2.8.0。

如需詳細資訊，請參閱在 Device Farm 中自動執行 Appium 測試。

APPIUM_WEB_PYTHON_TEST_PACKAGE_INSTALL_DEPENDENCY_WHEELS

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 無法安裝相依性 wheel。請解壓縮您的測試套件，開啟 requirements.txt 檔案和 wheelhouse 目錄，確認 requirements.txt 檔案中指定的相依性 wheel 完全符合 wheelhouse 目錄中的相依性 wheel，然後再試一次。

我們強烈建議您設定 Python virtualenv 來封裝測試。以下是使用 Python virtualenv 建立虛擬環境然後啟用的範例流程：

```
$ virtualenv workspace
$ cd workspace
$ source bin/activate
```

請確認您可以正確解壓縮測試套件。在下列範例中，套件的名稱是 test_bundle.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip test_bundle.zip
```

2. 若要測試安裝 wheel 檔案，您可以執行下列命令：

```
$ pip install --use-wheel --no-index --find-links=./wheelhouse --requirement=./
requirements.txt
```

有效的 Appium Python 套件應產生輸出如下：

```
Ignoring indexes: https://pypi.python.org/simple
Collecting Appium-Python-Client==0.20 (from -r ./requirements.txt (line 1))
Collecting py==1.4.31 (from -r ./requirements.txt (line 2))
Collecting pytest==2.9.0 (from -r ./requirements.txt (line 3))
Collecting selenium==2.52.0 (from -r ./requirements.txt (line 4))
Collecting wheel==0.26.0 (from -r ./requirements.txt (line 5))
Installing collected packages: selenium, Appium-Python-Client, py, pytest, wheel
  Found existing installation: wheel 0.29.0
    Uninstalling wheel-0.29.0:
      Successfully uninstalled wheel-0.29.0
Successfully installed Appium-Python-Client-0.20 py-1.4.31 pytest-2.9.0
 selenium-2.52.0 wheel-0.26.0
```

3. 若要停用虛擬環境，您可以執行下列命令：

```
$ deactivate
```

如需詳細資訊，請參閱在 Device Farm 中自動執行 Appium 測試。

APPIUM_WEB_PYTHON_TEST_PACKAGE_PYTEST_COLLECT_FAILED

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 無法在測試目錄中收集測試。請解壓縮您的測試套件，執行命令「py.test --collect-only <path to your tests directory>」，確認測試套件是否有效，並在命令未列印任何錯誤後再試一次。

我們強烈建議您設定 Python virtualenv 來封裝測試。以下是使用 Python virtualenv 建立虛擬環境然後啟用的範例流程：

```
$ virtualenv workspace
$ cd workspace
$ source bin/activate
```

請確認您可以正確解壓縮測試套件。在下列範例中，套件的名稱是 test_bundle.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

   ```
   $ unzip test_bundle.zip
   ```

2. 若要安裝 wheel 檔案，您可以執行下列命令：

   ```
   $ pip install --use-wheel --no-index --find-links=./wheelhouse --requirement=./
   requirements.txt
   ```

3. 若要收集測試，您可以執行下列命令：

   ```
   $ py.test --collect-only tests
   ```

   有效的 Appium Python 套件應產生輸出如下：

   ```
   ==================== test session starts ====================
   platform darwin -- Python 2.7.11, pytest-2.9.0, py-1.4.31, pluggy-0.3.1
   rootdir: /Users/zhena/Desktop/Ios/tests, inifile:
   collected 1 items
   <Module 'test_unittest.py'>
     <UnitTestCase 'DeviceFarmAppiumWebTests'>
       <TestCaseFunction 'test_devicefarm'>

   ==================== no tests ran in 0.11 seconds ====================
   ```

4. 若要停用虛擬環境，您可以執行下列命令：

```
$ deactivate
```

如需詳細資訊，請參閱在 Device Farm 中自動執行 Appium 測試。

故障診斷 AWS Device Farm 中的檢測測試

下列主題會列出在上傳檢測測試期間出現的錯誤訊息，並建議解決每個錯誤的解決方法。

> **Note**
>
> 如需在 AWS Device Farm 中使用檢測測試的重要考量，請參閱 適用於 Android 和 AWS Device Farm 的檢測。

INSTRUMENTATION_TEST_PACKAGE_UNZIP_FAILED

如果您看到下列訊息，請依照以下步驟修復問題。

```
Warning: We could not open your test APK file. Please verify that the file is valid and
    try again.
```

請確認您可以正確解壓縮測試套件。在下列範例中，套件的名稱是 app-debug-androidTest-unaligned.apk。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

   ```
   $ unzip app-debug-androidTest-unaligned.apk
   ```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

   ```
   $ tree .
   ```

   有效的檢測測試套件將產生輸出如下：

   ```
   .
   |-- AndroidManifest.xml
   |--  classes.dex
   ```

```
|-- resources.arsc
|-- LICENSE-junit.txt
|-- junit (directory)
`-- META-INF (directory)
```

如需詳細資訊，請參閱適用於 Android 和 AWS Device Farm 的檢測。

INSTRUMENTATION_TEST_PACKAGE_AAPT_DEBUG_BADGING_FAILED

如果您看到下列訊息，請依照以下步驟修復問題。

```
We could not extract information about your test package. Please verify that the
      test package is valid by running the command "aapt debug badging <path to your
 test
      package>", and try again after the command does not print any error.
```

在上傳驗證程序期間，Device Farm 會從`aapt debug badging <path to your package>`命令的輸出中剖析資訊。

請確認您可以在檢測測試套件上成功執行此命令。

在下列範例中，套件的名稱是 app-debug-androidTest-unaligned.apk。

- 將您的測試套件複製到工作目錄，然後執行下列命令：

  ```
  $ aapt debug badging app-debug-androidTest-unaligned.apk
  ```

  有效的檢測測試套件將產生輸出如下：

  ```
  package: name='com.amazon.aws.adf.android.referenceapp.test' versionCode=''
   versionName='' platformBuildVersionName='5.1.1-1819727'
  sdkVersion:'9'
  targetSdkVersion:'22'
  application-label:'Test-api'
  application: label='Test-api' icon=''
  application-debuggable
  uses-library:'android.test.runner'
  feature-group: label=''
  uses-feature: name='android.hardware.touchscreen'
  uses-implied-feature: name='android.hardware.touchscreen' reason='default feature
   for all apps'
  ```

```
supports-screens: 'small' 'normal' 'large' 'xlarge'
supports-any-density: 'true'
locales: '--_--'
densities: '160'
```

如需詳細資訊，請參閱適用於 Android 和 AWS Device Farm 的檢測。

INSTRUMENTATION_TEST_PACKAGE_INSTRUMENTATION_RUNNER_VALU

如果您看到下列訊息，請依照以下步驟修復問題。

```
We could not find the instrumentation runner value in the AndroidManifest.xml.
      Please verify the test package is valid by running the command "aapt dump xmltree
 <path to
      your test package> AndroidManifest.xml", and try again after finding the
 instrumentation
      runner value behind the keyword "instrumentation."
```

在上傳驗證程序期間，Device Farm 會從 XML 剖析樹狀目錄中剖析包含在套件中的 XML 檔案的檢測執行器值。您可以使用下列命令：`aapt dump xmltree <path to your package> AndroidManifest.xml`。

請確認您可以在檢測測試套件上執行此命令，並成功找到檢測值。

在下列範例中，套件的名稱是 app-debug-androidTest-unaligned.apk。

- 將您的測試套件複製到工作目錄，然後執行下列命令：

  ```
  $ aapt dump xmltree app-debug-androidTest-unaligned.apk AndroidManifest.xml | grep
   -A5 "instrumentation"
  ```

  有效的檢測測試套件將產生輸出如下：

  ```
  E: instrumentation (line=9)
        A: android:label(0x01010001)="Tests for
   com.amazon.aws.adf.android.referenceapp" (Raw: "Tests for
   com.amazon.aws.adf.android.referenceapp")
        A:
   android:name(0x01010003)="android.support.test.runner.AndroidJUnitRunner" (Raw:
   "android.support.test.runner.AndroidJUnitRunner")
  ```

```
      A:
 android:targetPackage(0x01010021)="com.amazon.aws.adf.android.referenceapp" (Raw:
 "com.amazon.aws.adf.android.referenceapp")
      A: android:handleProfiling(0x01010022)=(type 0x12)0x0
      A: android:functionalTest(0x01010023)=(type 0x12)0x0
```

如需詳細資訊，請參閱適用於 Android 和 AWS Device Farm 的檢測。

INSTRUMENTATION_TEST_PACKAGE_AAPT_DUMP_XMLTREE_FAILED

如果您看到下列訊息，請依照以下步驟修復問題。

```
We could not find the valid AndroidManifest.xml in your test package. Please
      verify that the test package is valid by running the command "aapt dump xmltree
 <path to
      your test package> AndroidManifest.xml", and try again after the command does not
 print any
      error.
```

在上傳驗證程序期間，Device Farm 會使用下列命令，從 XML 剖析樹狀目錄中剖析包含在套件中的 XML 檔案的資訊：`aapt dump xmltree <path to your package> AndroidManifest.xml`。

請確認您可以在檢測測試套件上成功執行此命令。

在下列範例中，套件的名稱是 app-debug-androidTest-unaligned.apk。

- 將您的測試套件複製到工作目錄，然後執行下列命令：

  ```
  $ aapt dump xmltree app-debug-androidTest-unaligned.apk AndroidManifest.xml
  ```

  有效的檢測測試套件將產生輸出如下：

  ```
  N: android=http://schemas.android.com/apk/res/android
    E: manifest (line=2)
      A: package="com.amazon.aws.adf.android.referenceapp.test" (Raw:
   "com.amazon.aws.adf.android.referenceapp.test")
      A: platformBuildVersionCode=(type 0x10)0x16 (Raw: "22")
      A: platformBuildVersionName="5.1.1-1819727" (Raw: "5.1.1-1819727")
      E: uses-sdk (line=5)
  ```

```
      A: android:minSdkVersion(0x0101020c)=(type 0x10)0x9
      A: android:targetSdkVersion(0x01010270)=(type 0x10)0x16
    E: instrumentation (line=9)
      A: android:label(0x01010001)="Tests for
com.amazon.aws.adf.android.referenceapp" (Raw: "Tests for
com.amazon.aws.adf.android.referenceapp")
      A:
android:name(0x01010003)="android.support.test.runner.AndroidJUnitRunner" (Raw:
"android.support.test.runner.AndroidJUnitRunner")
      A:
android:targetPackage(0x01010021)="com.amazon.aws.adf.android.referenceapp" (Raw:
"com.amazon.aws.adf.android.referenceapp")
      A: android:handleProfiling(0x01010022)=(type 0x12)0x0
      A: android:functionalTest(0x01010023)=(type 0x12)0x0
    E: application (line=16)
      A: android:label(0x01010001)=@0x7f020000
      A: android:debuggable(0x0101000f)=(type 0x12)0xffffffff
      E: uses-library (line=17)
        A: android:name(0x01010003)="android.test.runner" (Raw:
"android.test.runner")
```

如需詳細資訊，請參閱適用於 Android 和 AWS Device Farm 的檢測。

INSTRUMENTATION_TEST_PACKAGE_TEST_PACKAGE_NAME_VALUE_MISS

如果您看到下列訊息，請依照以下步驟修復問題。

```
We could not find the package name in your test package. Please verify that the
      test package is valid by running the command "aapt debug badging <path to your
 test
      package>", and try again after finding the package name value behind the keyword
 "package:
      name."
```

在上傳驗證程序期間，Device Farm 會從下列命令的輸出中剖析套件名稱值：`aapt debug badging <path to your package>`。

請確認您可以在檢測測試套件上執行此命令，並成功找到套件名稱值。

在下列範例中，套件的名稱是 app-debug-androidTest-unaligned.apk。

- 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ aapt debug badging app-debug-androidTest-unaligned.apk | grep "package: name="
```

有效的檢測測試套件將產生輸出如下：

```
package: name='com.amazon.aws.adf.android.referenceapp.test' versionCode=''
 versionName='' platformBuildVersionName='5.1.1-1819727'
```

如需詳細資訊，請參閱適用於 Android 和 AWS Device Farm 的檢測。

對 AWS Device Farm 中的 iOS 應用程式測試進行故障診斷

下列主題會列出在上傳 iOS 應用程式測試期間出現的錯誤訊息，並建議解決每個錯誤的解決方法。

> **Note**
>
> 以下說明以 Linux x86_64 和 Mac 為基礎。

IOS_APP_UNZIP_FAILED

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 無法開啟您的應用程式。請確認檔案是否有效，然後再試一次。

請確認您可以正確解壓縮應用程式套件。在下列範例中，套件的名稱為 AWSDeviceFarmiOSReferenceApp.ipa。

1. 將您的應用程式套件複製到工作目錄，然後執行下列命令：

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

有效的 iOS 應用程式套件應產生如下輸出：

```
.
`-- Payload (directory)
        `-- AWSDeviceFarmiOSReferenceApp.app (directory)
                        |-- Info.plist
                        `-- (any other files)
```

如需詳細資訊，請參閱AWS Device Farm 中的 iOS 測試。

IOS_APP_PAYLOAD_DIR_MISSING

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 在應用程式中找不到承載目錄。請解壓縮您的應用程式，確認承載目錄位於套件中，然後再試一次。

在下列範例中，套件的名稱為 AWSDeviceFarmiOSReferenceApp.ipa。

1. 將您的應用程式套件複製到工作目錄，然後執行下列命令：

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

如果 iOS 應用程式套件是有效的，您可以在工作目錄中找到##目錄。

```
.
`-- Payload (directory)
        `-- AWSDeviceFarmiOSReferenceApp.app (directory)
                        |-- Info.plist
                        `-- (any other files)
```

如需詳細資訊，請參閱AWS Device Farm 中的 iOS 測試。

IOS_APP_APP_DIR_MISSING

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 在承載目錄中找不到 .app 目錄。請解壓縮您的應用程式，開啟承載目錄，確認目錄中有 .app 目錄，然後再試一次。

在下列範例中，套件的名稱為 AWSDeviceFarmiOSReferenceApp.ipa。

1. 將您的應用程式套件複製到工作目錄，然後執行下列命令：

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

如果 iOS 應用程式套件有效，您會在*##*目錄內的範例中找到 *.app* 目錄，例如 *AWSDeviceFarmiOSReferenceApp.app*

```
.
`-- Payload (directory)
        `-- AWSDeviceFarmiOSReferenceApp.app (directory)
                        |-- Info.plist
                        `-- (any other files)
```

如需詳細資訊，請參閱AWS Device Farm 中的 iOS 測試。

IOS_APP_PLIST_FILE_MISSING

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 在 .app 目錄中找不到 Info.plist 檔案。請解壓縮您的應用程式，開啟 .app 目錄，確認目錄中有 Info.plist 檔案，然後再試一次。

在下列範例中，套件的名稱為 AWSDeviceFarmiOSReferenceApp.ipa。

1. 將您的應用程式套件複製到工作目錄，然後執行下列命令：

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

如果 iOS 應用程式套件有效，您會在範例中的 *.app* 目錄中找到 *Info.plist* 檔案，例如 *AWSDeviceFarmiOSReferenceApp.app*。

```
.
`-- Payload (directory)
        `-- AWSDeviceFarmiOSReferenceApp.app (directory)
                        |-- Info.plist
                        `-- (any other files)
```

如需詳細資訊，請參閱AWS Device Farm 中的 iOS 測試。

IOS_APP_CPU_ARCHITECTURE_VALUE_MISSING

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 在 Info.plist 檔案中找不到 CPU 架構值。請解壓縮您的應用程式，開啟 .app 目錄中的 Info.plist 檔案，確認已指定金鑰「UIRequiredDeviceCapabilities」，然後再試一次。

在下列範例中，套件的名稱為 AWSDeviceFarmiOSReferenceApp.ipa。

1. 將您的應用程式套件複製到工作目錄，然後執行下列命令：

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

您應該會在範例中的 *.app* 目錄中找到 *Info.plist* 檔案，例如 *AWSDeviceFarmiOSReferenceApp.app*：

```
.
`-- Payload (directory)
        `-- AWSDeviceFarmiOSReferenceApp.app (directory)
                        |-- Info.plist
                        `-- (any other files)
```

3. 若要找到 CPU 架構值，您可以使用 Xcode 或 Python 開啟 Info.plist。

若您是使用 Python，則可透過執行下列命令安裝 biplist 模組：

```
$ pip install biplist
```

4. 接著，開啟 Python 並執行下列命令：

```
import biplist
info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
Info.plist')
print info_plist['UIRequiredDeviceCapabilities']
```

有效的 iOS 應用程式套件應產生如下輸出：

```
['armv7']
```

如需詳細資訊，請參閱AWS Device Farm 中的 iOS 測試。

IOS_APP_PLATFORM_VALUE_MISSING

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 在 Info.plist 檔案中找不到平台值。請解壓縮您的應用程式，開啟 .app 目錄中的 Info.plist 檔案，確認已指定金鑰「CFBundleSupportedPlatforms」，然後再試一次。

在下列範例中，套件的名稱為 AWSDeviceFarmiOSReferenceApp.ipa。

1. 將您的應用程式套件複製到工作目錄，然後執行下列命令：

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

   您應該會在範例中的 *.app* 目錄中找到 *Info.plist* 檔案，例如 *AWSDeviceFarmiOSReferenceApp.app*：

```
.
`-- Payload (directory)
        `-- AWSDeviceFarmiOSReferenceApp.app (directory)
                    |-- Info.plist
                    `-- (any other files)
```

3. 若要找到平台架構值，您可以使用 Xcode 或 Python 開啟 Info.plist。

   若您是使用 Python，則可透過執行下列命令安裝 biplist 模組：

```
$ pip install biplist
```

4. 接著，開啟 Python 並執行下列命令：

```
import biplist
info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
Info.plist')
print info_plist['CFBundleSupportedPlatforms']
```

   有效的 iOS 應用程式套件應產生如下輸出：

```
['iPhoneOS']
```

如需詳細資訊，請參閱AWS Device Farm 中的 iOS 測試。

IOS_APP_WRONG_PLATFORM_DEVICE_VALUE

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> Info.plist 檔案中的平台裝置值錯誤。請解壓縮您的應用程式，開啟 .app 目錄中的 Info.plist 檔案，確認金鑰「CFBundleSupportedPlatforms」的值未包含關鍵字「simulator」，然後再試一次。

在下列範例中，套件的名稱為 AWSDeviceFarmiOSReferenceApp.ipa。

1. 將您的應用程式套件複製到工作目錄，然後執行下列命令：

   ```
   $ unzip AWSDeviceFarmiOSReferenceApp.ipa
   ```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

   ```
   $ tree .
   ```

   您應該會在範例中的 *.app* 目錄中找到 *Info.plist* 檔案，例如 *AWSDeviceFarmiOSReferenceApp.app*：

   ```
   .
   `-- Payload (directory)
           `-- AWSDeviceFarmiOSReferenceApp.app (directory)
                       |-- Info.plist
                       `-- (any other files)
   ```

3. 若要找到平台架構值，您可以使用 Xcode 或 Python 開啟 Info.plist。

   若您是使用 Python，則可透過執行下列命令安裝 biplist 模組：

```
$ pip install biplist
```

4. 接著，開啟 Python 並執行下列命令：

```
import biplist
info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
Info.plist')
print info_plist['CFBundleSupportedPlatforms']
```

有效的 iOS 應用程式套件應產生如下輸出：

```
['iPhoneOS']
```

如果 iOS 應用程式是有效的，值應不包含關鍵字 `simulator`。

如需詳細資訊，請參閱AWS Device Farm 中的 iOS 測試。

IOS_APP_FORM_FACTOR_VALUE_MISSING

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 在 Info.plist 檔案中找不到表單係數值。請解壓縮您的應用程式，開啟 .app 目錄中的 Info.plist 檔案，確認已指定金鑰「UIDeviceFamily」，然後再試一次。

在下列範例中，套件的名稱為 AWSDeviceFarmiOSReferenceApp.ipa。

1. 將您的應用程式套件複製到工作目錄，然後執行下列命令：

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

您應該會在範例中的 *.app* 目錄中找到 *Info.plist* 檔案，例如 *AWSDeviceFarmiOSReferenceApp.app*：

```
.
`-- Payload (directory)
        `-- AWSDeviceFarmiOSReferenceApp.app (directory)
                        |-- Info.plist
                        `-- (any other files)
```

3. 若要找到表單係數值，您可以使用 Xcode 或 Python 開啟 Info.plist。

   若您是使用 Python，則可透過執行下列命令安裝 biplist 模組：

   ```
   $ pip install biplist
   ```

4. 接著，開啟 Python 並執行下列命令：

   ```
   import biplist
   info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
   Info.plist')
   print info_plist['UIDeviceFamily']
   ```

   有效的 iOS 應用程式套件應產生如下輸出：

   ```
   [1, 2]
   ```

   如需詳細資訊，請參閱AWS Device Farm 中的 iOS 測試。

IOS_APP_PACKAGE_NAME_VALUE_MISSING

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 在 Info.plist 檔案中找不到套件名稱值。請解壓縮您的應用程式，開啟 .app 目錄中的 Info.plist 檔案，確認已指定金鑰「CFBundleIdentifier」，然後再試一次。

在下列範例中，套件的名稱為 AWSDeviceFarmiOSReferenceApp.ipa。

1. 將您的應用程式套件複製到工作目錄，然後執行下列命令：

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

您應該會在範例中的 *.app* 目錄中找到 *Info.plist* 檔案，例如 *AWSDeviceFarmiOSReferenceApp.app*：

```
.
`-- Payload (directory)
        `-- AWSDeviceFarmiOSReferenceApp.app (directory)
                        |-- Info.plist
                        `-- (any other files)
```

3. 若要找到套件名稱值，您可以使用 Xcode 或 Python 開啟 Info.plist。

若您是使用 Python，則可透過執行下列命令安裝 biplist 模組：

```
$ pip install biplist
```

4. 接著，開啟 Python 並執行下列命令：

```
import biplist
info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
Info.plist')
print info_plist['CFBundleIdentifier']
```

有效的 iOS 應用程式套件應產生如下輸出：

```
Amazon.AWSDeviceFarmiOSReferenceApp
```

如需詳細資訊，請參閱AWS Device Farm 中的 iOS 測試。

IOS_APP_EXECUTABLE_VALUE_MISSING

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 在 Info.plist 檔案中找不到可執行的值。請解壓縮您的應用程式，開啟 .app 目錄中的 Info.plist 檔案，確認已指定金鑰「CFBundleExecutable」，然後再試一次。

在下列範例中，套件的名稱為 AWSDeviceFarmiOSReferenceApp.ipa。

1. 將您的應用程式套件複製到工作目錄，然後執行下列命令：

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

   您應該會在範例中的 *.app* 目錄中找到 *Info.plist* 檔案，例如 *AWSDeviceFarmiOSReferenceApp.app*：

```
.
`-- Payload (directory)
        `-- AWSDeviceFarmiOSReferenceApp.app (directory)
                    |-- Info.plist
                    `-- (any other files)
```

3. 若要找到可執行的值，您可以使用 Xcode 或 Python 開啟 Info.plist。

   若您是使用 Python，則可透過執行下列命令安裝 biplist 模組：

```
$ pip install biplist
```

4. 接著，開啟 Python 並執行下列命令：

```
import biplist
info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
Info.plist')
print info_plist['CFBundleExecutable']
```

   有效的 iOS 應用程式套件應產生如下輸出：

```
AWSDeviceFarmiOSReferenceApp
```

如需詳細資訊，請參閱AWS Device Farm 中的 iOS 測試。

針對 AWS Device Farm 中的 XCTest 測試進行故障診斷

下列主題會列出在上傳 XCTest 測試期間出現的錯誤訊息，並建議解決每個錯誤的解決方法。

> **Note**
>
> 下列說明假設您使用 MacOS。

XCTEST_TEST_PACKAGE_UNZIP_FAILED

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 無法開啟您的測試 ZIP 檔。請確認檔案是否有效，然後再試一次。

請確認您可以正確解壓縮應用程式套件。在下列範例中，套件的名稱為 swiftExampleTests.xctest-1.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

   ```
   $ unzip swiftExampleTests.xctest-1.zip
   ```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

   ```
   $ tree .
   ```

   有效的 XCTest 套件應產生輸出如下：

   ```
   .
   `-- swiftExampleTests.xctest (directory)
   ```

```
                |-- Info.plist
                `-- (any other files)
```

如需詳細資訊，請參閱將 Device Farm 與適用於 iOS 的 XCTest 整合。

XCTEST_TEST_PACKAGE_XCTEST_DIR_MISSING

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 在測試套件中找不到 .xctest 目錄。請解壓縮您的測試套件，確認 .xctest 目錄位於套件中，然後再試一次。

在下列範例中，套件的名稱為 swiftExampleTests.xctest-1.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip swiftExampleTests.xctest-1.zip
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

如果 XCTest 套件是有效的，您可以在工作目錄中找到名稱類似於 *swiftExampleTests.xctest* 的目錄。該名稱應以 *.xctest* 結尾。

```
.
`-- swiftExampleTests.xctest (directory)
                |-- Info.plist
                `-- (any other files)
```

如需詳細資訊，請參閱將 Device Farm 與適用於 iOS 的 XCTest 整合。

XCTEST_TEST_PACKAGE_PLIST_FILE_MISSING

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 在 .xctest 目錄中找不到 .plist 檔案。請解壓縮您的測試套件，開啟相依性 .xctest 目錄，確認目錄中有 Info.plist 檔案，然後再試一次。

在下列範例中，套件的名稱為 swiftExampleTests.xctest-1.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

   ```
   $ unzip swiftExampleTests.xctest-1.zip
   ```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

   ```
   $ tree .
   ```

   如果 XCTest 套件有效，您會在 *.xctest* 目錄中找到 *Info.plist* 檔案。在以下範例中，目錄的名稱為 *swiftExampleTests.xctest*。

   ```
   .
   `-- swiftExampleTests.xctest (directory)
                |-- Info.plist
                `-- (any other files)
   ```

   如需詳細資訊，請參閱將 Device Farm 與適用於 iOS 的 XCTest 整合。

XCTEST_TEST_PACKAGE_PACKAGE_NAME_VALUE_MISSING

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 在 Info.plist 檔案中找不到套件名稱值。請解壓縮您的測試套件，開啟 Info.plist 檔案，確認已指定金鑰「CFBundleIdentifier」，然後再試一次。

在下列範例中，套件的名稱為 swiftExampleTests.xctest-1.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip swiftExampleTests.xctest-1.zip
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

您應該會在 *.xctest* 目錄中找到 *Info.plist* 檔案，例如 *swiftExampleTests.xctest*，在我們的範例中：

```
.
`-- swiftExampleTests.xctest (directory)
              |-- Info.plist
              `-- (any other files)
```

3. 若要找到套件名稱值，您可以使用 Xcode 或 Python 開啟 Info.plist。

若您是使用 Python，則可透過執行下列命令安裝 biplist 模組：

```
$ pip install biplist
```

4. 接著，開啟 Python 並執行下列命令：

```
import biplist
info_plist = biplist.readPlist('swiftExampleTests.xctest/Info.plist')
print info_plist['CFBundleIdentifier']
```

有效的 XCtest 應用程式套件應產生輸出如下：

```
com.amazon.kanapka.swiftExampleTests
```

如需詳細資訊，請參閱將 Device Farm 與適用於 iOS 的 XCTest 整合。

XCTEST_TEST_PACKAGE_EXECUTABLE_VALUE_MISSING

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 在 Info.plist 檔案中找不到可執行的值。請解壓縮您的測試套件，開啟 Info.plist 檔案，確認已指定金鑰「CFBundleExecutable」，然後再試一次。

在下列範例中，套件的名稱為 swiftExampleTests.xctest-1.zip。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip swiftExampleTests.xctest-1.zip
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

   您應該會在 *.xctest* 目錄中找到 *Info.plist* 檔案，例如 *swiftExampleTests.xctest*，在我們的範例中：

```
.
`-- swiftExampleTests.xctest (directory)
              |-- Info.plist
              `-- (any other files)
```

3. 若要找到套件名稱值，您可以使用 Xcode 或 Python 開啟 Info.plist。

   若您是使用 Python，則可透過執行下列命令安裝 biplist 模組：

```
$ pip install biplist
```

4. 接著，開啟 Python 並執行下列命令：

```
import biplist
info_plist = biplist.readPlist('swiftExampleTests.xctest/Info.plist')
print info_plist['CFBundleExecutable']
```

   有效的 XCtest 應用程式套件應產生輸出如下：

```
swiftExampleTests
```

如需詳細資訊，請參閱將 Device Farm 與適用於 iOS 的 XCTest 整合。

對 AWS Device Farm 中的 XCTest UI 測試進行故障診斷

下列主題會列出在上傳 XCTest UI 測試期間出現的錯誤訊息，並建議解決每個錯誤的解決方法。

> **Note**
>
> 以下說明以 Linux x86_64 和 Mac 為基礎。

XCTEST_UI_TEST_PACKAGE_UNZIP_FAILED

如果您看到下列訊息，請依照以下步驟修復問題。

```
We could not open your test IPA file. Please verify that the file is valid and try again.
```

請確認您可以正確解壓縮應用程式套件。在下列範例中，套件的名稱為 swift-sample-UI.ipa。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

   ```
   $ tree .
   ```

   有效的 iOS 應用程式套件應產生如下輸出：

   ```
   .
   `-- Payload (directory)
           `-- swift-sampleUITests-Runner.app (directory)
                       |-- Info.plist
                       |-- Plugins (directory)
                       |       `swift-sampleUITests.xctest (directory)
                       |                       |-- Info.plist
                       |                       `-- (any other files)
                       `-- (any other files)
   ```

如需詳細資訊，請參閱將適用於 iOS 的 XCTest UI 與 Device Farm 整合。

XCTEST_UI_TEST_PACKAGE_PAYLOAD_DIR_MISSING

如果您看到下列訊息，請依照以下步驟修復問題。

```
We could not find the Payload directory inside your test package. Please unzip your test package, verify that the Payload directory is inside the package, and try again.
```

在下列範例中，套件的名稱為 swift-sample-UI.ipa。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip swift-sample-UI.ipa
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

如果 XCTest UI 套件是有效的，您可以在工作目錄中找到##目錄。

```
.
`-- Payload (directory)
        `-- swift-sampleUITests-Runner.app (directory)
                    |-- Info.plist
                    |-- Plugins (directory)
                    |       `swift-sampleUITests.xctest (directory)
                    |                       |-- Info.plist
                    |                       `-- (any other files)
                    `-- (any other files)
```

如需詳細資訊，請參閱將適用於 iOS 的 XCTest UI 與 Device Farm 整合。

XCTEST_UI_TEST_PACKAGE_APP_DIR_MISSING

如果您看到下列訊息，請依照以下步驟修復問題。

```
We could not find the .app directory inside the Payload directory. Please
unzip your test package and then open the Payload directory, verify that
the .app directory is inside the directory, and try again.
```

在下列範例中，套件的名稱為 swift-sample-UI.ipa。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip swift-sample-UI.ipa
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

如果 XCTest UI 套件有效，您可以在*##*目錄內的範例中找到 *.app* 目錄，例如 *swift-sampleUITests-Runner.app#*

```
.
`-- Payload (directory)
        `-- swift-sampleUITests-Runner.app (directory)
                        |-- Info.plist
                        |-- Plugins (directory)
                        |       `swift-sampleUITests.xctest (directory)
                        |                         |-- Info.plist
                        |                         `-- (any other files)
                        `-- (any other files)
```

如需詳細資訊，請參閱[將適用於 iOS 的 XCTest UI 與 Device Farm 整合](#)。

XCTEST_UI_TEST_PACKAGE_PLUGINS_DIR_MISSING

如果您看到下列訊息，請依照以下步驟修復問題。

```
We could not find the Plugins directory inside the .app directory. Please
unzip your test package and then open the .app directory, verify that the
Plugins directory is inside the directory, and try again.
```

在下列範例中，套件的名稱為 swift-sample-UI.ipa。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip swift-sample-UI.ipa
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

如果 XCTest UI 套件有效，您會在 *.app* 目錄中找到*####*目錄。在範例中，目錄的名稱為 *swift-sampleUITests-Runner.app*。

```
.
`-- Payload (directory)
        `-- swift-sampleUITests-Runner.app (directory)
                        |-- Info.plist
                        |-- Plugins (directory)
                        |       `swift-sampleUITests.xctest (directory)
                        |                         |-- Info.plist
                        |                         `-- (any other files)
                        `-- (any other files)
```

如需詳細資訊，請參閱將適用於 iOS 的 XCTest UI 與 Device Farm 整合。

XCTEST_UI_TEST_PACKAGE_XCTEST_DIR_MISSING_IN_PLUGINS_DIR

如果您看到下列訊息，請依照以下步驟修復問題。

```
We could not find the .xctest directory inside the plugins directory.
Please unzip your test package and then open the plugins directory, verify
that the .xctest directory is inside the directory, and try again.
```

在下列範例中，套件的名稱為 swift-sample-UI.ipa。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip swift-sample-UI.ipa
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

如果 XCTest UI 套件有效，您會在*####*目錄中找到 *.xctest* 目錄。在範例中，目錄的名稱為 *swift-sampleUITests.xctest*。

```
.
`-- Payload (directory)
        `-- swift-sampleUITests-Runner.app (directory)
                    |-- Info.plist
                    |-- Plugins (directory)
                    |       `swift-sampleUITests.xctest (directory)
                    |                       |-- Info.plist
                    |                       `-- (any other files)
                    `-- (any other files)
```

如需詳細資訊，請參閱將適用於 iOS 的 XCTest UI 與 Device Farm 整合。

XCTEST_UI_TEST_PACKAGE_PLIST_FILE_MISSING

如果您看到下列訊息，請依照以下步驟修復問題。

```
We could not find the Info.plist file inside the .app directory. Please unzip your test package and then open the .app directory, verify that the Info.plist file is inside the directory, and try again.
```

在下列範例中，套件的名稱為 swift-sample-UI.ipa。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip swift-sample-UI.ipa
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

如果 XCTest UI 套件有效，您會在 *.app* 目錄中找到 *Info.plist* 檔案。在以下範例中，目錄的名稱為 *swift-sampleUITests-Runner.app*。

```
.
`-- Payload (directory)
        `-- swift-sampleUITests-Runner.app (directory)
```

```
                    |-- Info.plist
                    |-- Plugins (directory)
                    |       `swift-sampleUITests.xctest (directory)
                    |                       |-- Info.plist
                    |                       `-- (any other files)
                    `-- (any other files)
```

如需詳細資訊，請參閱將適用於 iOS 的 XCTest UI 與 Device Farm 整合。

XCTEST_UI_TEST_PACKAGE_PLIST_FILE_MISSING_IN_XCTEST_DIR

如果您看到下列訊息，請依照以下步驟修復問題。

```
We could not find the Info.plist file inside the .xctest directory. Please
unzip your test package and then open the .xctest directory, verify that
the Info.plist file is inside the directory, and try again.
```

在下列範例中，套件的名稱為 swift-sample-UI.ipa。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip swift-sample-UI.ipa
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

如果 XCTest UI 套件有效，您會在 *.xctest* 目錄中找到 *Info.plist* 檔案。在以下範例中，目錄的名稱為 *swift-sampleUITests.xctest*。

```
.
`-- Payload (directory)
        `-- swift-sampleUITests-Runner.app (directory)
                    |-- Info.plist
                    |-- Plugins (directory)
                    |       `swift-sampleUITests.xctest (directory)
                    |                       |-- Info.plist
                    |                       `-- (any other files)
                    `-- (any other files)
```

如需詳細資訊，請參閱將適用於 iOS 的 XCTest UI 與 Device Farm 整合。

XCTEST_UI_TEST_PACKAGE_CPU_ARCHITECTURE_VALUE_MISSING

如果您看到下列訊息，請依照以下步驟修復問題。

```
We could not the CPU architecture value in the Info.plist file. Please
unzip your test package and then open the Info.plist file inside the .app
directory, verify that the key "UIRequiredDeviceCapabilities" is specified,
and try again.
```

在下列範例中，套件的名稱為 swift-sample-UI.ipa。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip swift-sample-UI.ipa
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

   您應該會在 *.app* 目錄中找到 *Info.plist* 檔案，例如 *swift-sampleUITests-Runner.app*，在我們的範例中：

```
.
`-- Payload (directory)
        `-- swift-sampleUITests-Runner.app (directory)
                        |-- Info.plist
                        |-- Plugins (directory)
                        |       `swift-sampleUITests.xctest (directory)
                        |                         |-- Info.plist
                        |                         `-- (any other files)
                        `-- (any other files)
```

3. 若要找到 CPU 架構值，您可以使用 Xcode 或 Python 開啟 Info.plist。

   若您是使用 Python，則可透過執行下列命令安裝 biplist 模組：

```
$ pip install biplist
```

4. 接著，開啟 Python 並執行下列命令：

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/
Info.plist')
print info_plist['UIRequiredDeviceCapabilities']
```

有效的 XCtest UI 套件應產生輸出如下：

```
['armv7']
```

如需詳細資訊，請參閱將適用於 iOS 的 XCTest UI 與 Device Farm 整合。

XCTEST_UI_TEST_PACKAGE_PLATFORM_VALUE_MISSING

如果您看到下列訊息，請依照以下步驟修復問題。

```
We could not find the platform value in the Info.plist. Please unzip your
test package and then open the Info.plist file inside the .app directory,
verify that the key "CFBundleSupportedPlatforms" is specified, and try
again.
```

在下列範例中，套件的名稱為 swift-sample-UI.ipa。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip swift-sample-UI.ipa
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

您應該會在 *.app* 目錄中找到 *Info.plist* 檔案，例如 *swift-sampleUITests-Runner.app*，在我們的範例中：

```
.
`-- Payload (directory)
        `-- swift-sampleUITests-Runner.app (directory)
                        |-- Info.plist
```

```
                        |-- Plugins (directory)
                        |       `swift-sampleUITests.xctest (directory)
                        |                        |-- Info.plist
                        |                        `-- (any other files)
                        `-- (any other files)
```

3. 若要找到平台架構值，您可以使用 Xcode 或 Python 開啟 Info.plist。

   若您是使用 Python，則可透過執行下列命令安裝 biplist 模組：

   ```
   $ pip install biplist
   ```

4. 接著，開啟 Python 並執行下列命令：

   ```
   import biplist
   info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
   print info_plist['CFBundleSupportedPlatforms']
   ```

   有效的 XCtest UI 套件應產生輸出如下：

   ```
   ['iPhoneOS']
   ```

   如需詳細資訊，請參閱將適用於 iOS 的 XCTest UI 與 Device Farm 整合。

XCTEST_UI_TEST_PACKAGE_WRONG_PLATFORM_DEVICE_VALUE

如果您看到下列訊息，請依照以下步驟修復問題。

```
We found the platform device value was wrong in the Info.plist file. Please
unzip your test package and then open the Info.plist file inside the .app
directory, verify that the value of the key "CFBundleSupportedPlatforms"
does not contain the keyword "simulator", and try again.
```

在下列範例中，套件的名稱為 swift-sample-UI.ipa。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

您應該會在 *.app* 目錄中找到 *Info.plist* 檔案，例如 *swift-sampleUITests-Runner.app*，在我們的範例中：

```
.
`-- Payload (directory)
        `-- swift-sampleUITests-Runner.app (directory)
                        |-- Info.plist
                        |-- Plugins (directory)
                        |       `swift-sampleUITests.xctest (directory)
                        |                         |-- Info.plist
                        |                         `-- (any other files)
                        `-- (any other files)
```

3. 若要找到平台架構值，您可以使用 Xcode 或 Python 開啟 Info.plist。

   若您是使用 Python，則可透過執行下列命令安裝 biplist 模組：

   ```
   $ pip install biplist
   ```

4. 接著，開啟 Python 並執行下列命令：

   ```
   import biplist
   info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
   print info_plist['CFBundleSupportedPlatforms']
   ```

   有效的 XCtest UI 套件應產生輸出如下：

   ```
   ['iPhoneOS']
   ```

   如果 XCTest UI 套件是有效的，值應不包含關鍵字 `simulator`。

   如需詳細資訊，請參閱將適用於 iOS 的 XCTest UI 與 Device Farm 整合。

XCTEST_UI_TEST_PACKAGE_FORM_FACTOR_VALUE_MISSING

如果您看到下列訊息，請依照以下步驟修復問題。

```
We could not the form factor value in the Info.plist. Please unzip your
test package and then open the Info.plist file inside the .app directory,
verify that the key "UIDeviceFamily" is specified, and try again.
```

在下列範例中，套件的名稱為 swift-sample-UI.ipa。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip swift-sample-UI.ipa
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

您應該會在 *.app* 目錄中找到 *Info.plist* 檔案，例如 *swift-sampleUITests-Runner.app*，在我們的範例中：

```
.
`-- Payload (directory)
        `-- swift-sampleUITests-Runner.app (directory)
                        |-- Info.plist
                        |-- Plugins (directory)
                        |       `swift-sampleUITests.xctest (directory)
                        |                         |-- Info.plist
                        |                         `-- (any other files)
                        `-- (any other files)
```

3. 若要找到表單係數值，您可以使用 Xcode 或 Python 開啟 Info.plist。

若您是使用 Python，則可透過執行下列命令安裝 biplist 模組：

```
$ pip install biplist
```

4. 接著，開啟 Python 並執行下列命令：

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
print info_plist['UIDeviceFamily']
```

有效的 XCtest UI 套件應產生輸出如下：

```
[1, 2]
```

如需詳細資訊，請參閱將適用於 iOS 的 XCTest UI 與 Device Farm 整合。

XCTEST_UI_TEST_PACKAGE_PACKAGE_NAME_VALUE_MISSING

如果您看到下列訊息，請依照以下步驟修復問題。

```
We could not find the package name value in the Info.plist file. Please unzip your test package and then open the Info.plist file inside the .app directory, verify that the key "CFBundleIdentifier" is specified, and try again.
```

在下列範例中，套件的名稱為 swift-sample-UI.ipa。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip swift-sample-UI.ipa
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

您應該會在 *.app* 目錄中找到 *Info.plist* 檔案，例如 *swift-sampleUITests-Runner.app*，在我們的範例中：

```
.
`-- Payload (directory)
        `-- swift-sampleUITests-Runner.app (directory)
                    |-- Info.plist
                    |-- Plugins (directory)
                    |       `swift-sampleUITests.xctest (directory)
                    |                       |-- Info.plist
                    |                       `-- (any other files)
                    `-- (any other files)
```

3. 若要找到套件名稱值，您可以使用 Xcode 或 Python 開啟 Info.plist。

若您是使用 Python，則可透過執行下列命令安裝 biplist 模組：

```
$ pip install biplist
```

4. 接著，開啟 Python 並執行下列命令：

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
print info_plist['CFBundleIdentifier']
```

有效的 XCtest UI 套件應產生輸出如下：

```
com.apple.test.swift-sampleUITests-Runner
```

如需詳細資訊，請參閱將適用於 iOS 的 XCTest UI 與 Device Farm 整合。

XCTEST_UI_TEST_PACKAGE_EXECUTABLE_VALUE_MISSING

如果您看到下列訊息，請依照以下步驟修復問題。

```
We could not find the executable value in the Info.plist file. Please
unzip your test package and then open the Info.plist file inside the .app
directory, verify that the key "CFBundleExecutable" is specified, and try
again.
```

在下列範例中，套件的名稱為 swift-sample-UI.ipa。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip swift-sample-UI.ipa
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

您應該會在 *.app* 目錄中找到 *Info.plist* 檔案，例如 *swift-sampleUITests-Runner.app*，在我們的範例中：

```
.
`-- Payload (directory)
```

```
        `-- swift-sampleUITests-Runner.app (directory)
                    |-- Info.plist
                    |-- Plugins (directory)
                    |       `swift-sampleUITests.xctest (directory)
                    |                       |-- Info.plist
                    |                       `-- (any other files)
                    `-- (any other files)
```

3. 若要找到可執行的值，您可以使用 Xcode 或 Python 開啟 Info.plist。

   若您是使用 Python，則可透過執行下列命令安裝 biplist 模組：

   ```
   $ pip install biplist
   ```

4. 接著，開啟 Python 並執行下列命令：

   ```
   import biplist
   info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
   print info_plist['CFBundleExecutable']
   ```

   有效的 XCtest UI 套件應產生輸出如下：

   ```
   XCTRunner
   ```

   如需詳細資訊，請參閱將適用於 iOS 的 XCTest UI 與 Device Farm 整合。

XCTEST_UI_TEST_PACKAGE_TEST_PACKAGE_NAME_VALUE_MISSING

如果您看到下列訊息，請依照以下步驟修復問題。

```
We could not find the package name value in the Info.plist file inside
the .xctest directory. Please unzip your test package and then open
the Info.plist file inside the .xctest directory, verify that the key
"CFBundleIdentifier" is specified, and try again.
```

在下列範例中，套件的名稱為 swift-sample-UI.ipa。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

您應該會在 *.app* 目錄中找到 *Info.plist* 檔案，例如 *swift-sampleUITests-Runner.app*，在我們的範例中：

```
.
`-- Payload (directory)
        `-- swift-sampleUITests-Runner.app (directory)
                        |-- Info.plist
                        |-- Plugins (directory)
                        |       `swift-sampleUITests.xctest (directory)
                        |                         |-- Info.plist
                        |                         `-- (any other files)
                        `-- (any other files)
```

3. 若要找到套件名稱值，您可以使用 Xcode 或 Python 開啟 Info.plist。

若您是使用 Python，則可透過執行下列命令安裝 biplist 模組：

```
$ pip install biplist
```

4. 接著，開啟 Python 並執行下列命令：

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Plugins/
swift-sampleUITests.xctest/Info.plist')
print info_plist['CFBundleIdentifier']
```

有效的 XCtest UI 套件應產生輸出如下：

```
com.amazon.swift-sampleUITests
```

如需詳細資訊，請參閱將適用於 iOS 的 XCTest UI 與 Device Farm 整合。

XCTEST_UI_TEST_PACKAGE_TEST_EXECUTABLE_VALUE_MISSING

如果您看到下列訊息，請依照以下步驟修復問題。

```
We could not find the executable value in the Info.plist file inside
the .xctest directory. Please unzip your test package and then open
the Info.plist file inside the .xctest directory, verify that the key
"CFBundleExecutable" is specified, and try again.
```

在下列範例中，套件的名稱為 swift-sample-UI.ipa。

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip swift-sample-UI.ipa
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

您應該會在 *.app* 目錄中找到 *Info.plist* 檔案，例如 *swift-sampleUITests-Runner.app*，在我們的範例中：

```
.
`-- Payload (directory)
        `-- swift-sampleUITests-Runner.app (directory)
                        |-- Info.plist
                        |-- Plugins (directory)
                        |       `swift-sampleUITests.xctest (directory)
                        |                         |-- Info.plist
                        |                         `-- (any other files)
                        `-- (any other files)
```

3. 若要找到可執行的值，您可以使用 Xcode 或 Python 開啟 Info.plist。

若您是使用 Python，則可透過執行下列命令安裝 biplist 模組：

```
$ pip install biplist
```

4. 接著，開啟 Python 並執行下列命令：

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Plugins/
swift-sampleUITests.xctest/Info.plist')
print info_plist['CFBundleExecutable']
```

有效的 XCtest UI 套件應產生輸出如下：

```
swift-sampleUITests
```

如需詳細資訊，請參閱將適用於 iOS 的 XCTest UI 與 Device Farm 整合。

XCTEST_UI_TEST_PACKAGE_MULTIPLE_APP_DIRS

如果您看到下列訊息，請依照以下步驟修復問題。

```
We found multiple .app directories inside your test package. Please unzip
your test package, verify that only a single .app directory is present
inside the package, then try again.
```

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip swift-sample-UI.zip
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

如果 XCTest UI 套件有效，您應該只會在 .zip 測試套件`swift-sampleUITests-Runner.app`中找到範例中的單一`.app`目錄。

```
.
`--swift-sample-UI.zip--(directory)
    `-- swift-sampleUITests-Runner.app (directory)
            |-- Info.plist
            |-- Plugins (directory)
            |       `swift-sampleUITests.xctest (directory)
           |            |-- Info.plist
           |            `-- (any other files)
            `-- (any other files)
    `-- (any other files)
```

如需詳細資訊，請參閱將適用於 iOS 的 XCTest UI 與 Device Farm 整合。

XCTEST_UI_TEST_PACKAGE_MULTIPLE_IPA_DIRS

如果您看到下列訊息，請依照以下步驟修復問題。

```
We found multiple .ipa directories inside your test package. Please unzip
your test package, verify that only a single .ipa directory is present
inside the package, then try again.
```

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

   ```
   $ unzip swift-sample-UI.zip
   ```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

   ```
   $ tree .
   ```

   如果 XCTest UI 套件有效，您應該只會在 .zip 測試套件sampleUITests.ipa中找到範例中的單一.ipa目錄。

   ```
   .
   `--swift-sample-UI.zip--(directory)
       `-- sampleUITests.ipa (directory)
               `-- Payload (directory)
                   `-- swift-sampleUITests-Runner.app (directory)
       `-- (any other files)
   ```

   如需詳細資訊，請參閱將適用於 iOS 的 XCTest UI 與 Device Farm 整合。

XCTEST_UI_TEST_PACKAGE_BOTH_APP_AND_IPA_DIR_PRESENT

如果您看到下列訊息，請依照以下步驟修復問題。

```
We found both .app and .ipa files inside your test package. Please unzip
your test package, verify that only a single .app or .ipa file is present
inside the package, then try again.
```

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

   ```
   $ unzip swift-sample-UI.zip
   ```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

如果 XCTest UI 套件有效，您應該會在 .zip 測試套件`swift-sampleUITests-Runner.app`中，在我們的範例中找到`.ipa`類似 `sampleUITests.ipa`或 的`.app`目錄。您可以在 上的文件中參考有效 XCTEST_UI 測試套件的範例將適用於 iOS 的 XCTest UI 與 Device Farm 整合。

```
.
`--swift-sample-UI.zip--(directory)
    `-- sampleUITests.ipa (directory)
            `-- Payload (directory)
                `-- swift-sampleUITests-Runner.app (directory)
   `-- (any other files)
```

或

```
.
`--swift-sample-UI.zip--(directory)
    `-- swift-sampleUITests-Runner.app (directory)
            |-- Info.plist
            |-- Plugins (directory)
            `-- (any other files)
   `-- (any other files)
```

如需詳細資訊，請參閱將適用於 iOS 的 XCTest UI 與 Device Farm 整合。

XCTEST_UI_TEST_PACKAGE_PAYLOAD_DIR_PRESENT_IN_ZIP

如果您看到下列訊息，請依照以下步驟修復問題。

```
We found a Payload directory inside your .zip test package. Please unzip your test package, ensure that a Payload directory is not present in the package, then try again.
```

1. 將您的測試套件複製到工作目錄，然後執行下列命令：

```
$ unzip swift-sample-UI.zip
```

2. 成功解壓縮套件後，您可以透過執行下列命令找到樹狀結構的工作目錄：

```
$ tree .
```

如果 XCTest UI 套件有效，則不應在測試套件中找到承載目錄。

```
.
`--swift-sample-UI.zip--(directory)
    `-- swift-sampleUITests-Runner.app (directory)
            |-- Info.plist
            |-- Plugins (directory)
            `-- (any other files)
   `-- Payload (directory) [This directory should not be present]
            |-- (any other files)
   `-- (any other files)
```

如需詳細資訊，請參閱將適用於 iOS 的 XCTest UI 與 Device Farm 整合。

中的安全性 AWS Device Farm

的雲端安全性 AWS 是最高優先順序。身為 AWS 客戶，您可以受益於資料中心和網路架構，該架構專為滿足最安全敏感組織的需求而建置。

安全性是 AWS 與您之間的共同責任。共同責任模式將其描述為雲端的安全性，和雲端中的安全性：

- 雲端的安全性 – AWS 負責保護在 AWS Cloud 中執行 AWS 服務的基礎設施。 AWS 也為您提供可安全使用的服務。在AWS 合規計劃中，第三方稽核人員會定期測試和驗證我們安全的有效性。若要了解適用的合規計劃 AWS Device Farm，請參閱合規計劃的 AWS 服務範圍合規。
- 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。

本文件可協助您了解如何在使用 Device Farm 時套用共同責任模型。下列主題說明如何設定 Device Farm 以符合您的安全與合規目標。您也會了解如何使用其他 AWS 服務來協助您監控和保護 Device Farm 資源。

主題

AWS Device Farm 中的身分和存取管理

目標對象

使用方式 AWS Identity and Access Management (IAM) 會根據您的角色而有所不同：

- 服務使用者 — 若無法存取某些功能，請向管理員申請所需許可 (請參閱 故障診斷 AWS Device Farm 身分和存取)
- 服務管理員 — 負責設定使用者存取權並提交相關許可請求 (請參閱 AWS Device Farm 如何與 IAM 搭配使用)
- IAM 管理員 — 撰寫政策以管理存取控制 (請參閱 AWS Device Farm 身分型政策範例)

使用身分驗證

身分驗證是您 AWS 使用身分憑證登入 的方式。您必須以 AWS 帳戶根使用者、IAM 使用者或擔任 IAM 角色身分進行身分驗證。

您可以使用身分來源的登入資料，例如 AWS IAM Identity Center (IAM Identity Center)、單一登入身分驗證或 Google/Facebook 登入資料，以聯合身分的形式登入。如需有關登入的詳細資訊，請參閱《AWS 登入 使用者指南》中的如何登入您的 AWS 帳戶。

對於程式設計存取， AWS 提供 SDK 和 CLI 以密碼編譯方式簽署請求。如需詳細資訊，請參閱《IAM 使用者指南》中的 API 請求的AWS 第 4 版簽署程序。

AWS 帳戶 根使用者

當您建立 時 AWS 帳戶，您會從一個名為 AWS 帳戶 theroot 使用者的登入身分開始，該身分具有對所有 AWS 服務 和 資源的完整存取權。強烈建議不要使用根使用者來執行日常任務。有關需要根使用者憑證的任務，請參閱《IAM 使用者指南》中的需要根使用者憑證的任務。

IAM 使用者和群組

IAM 使用者https://docs.aws.amazon.com/IAM/latest/UserGuide/id_users.html是一種身分具備單人或應用程式的特定許可權。建議以臨時憑證取代具備長期憑證的 IAM 使用者。如需詳細資訊，請參閱《IAM 使用者指南》中的要求人類使用者使用聯合身分提供者來 AWS 使用臨時憑證存取 。

IAM 群組會指定 IAM 使用者集合，使管理大量使用者的許可權更加輕鬆。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM 使用者的使用案例。

IAM 角色

IAM 角色https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles.html的身分具有特定許可權，其可以提供臨時憑證。您可以透過從使用者切換到 IAM 角色 （主控台） 或呼叫 AWS CLI 或 AWS API 操作來擔任角色。如需詳細資訊，請參閱《IAM 使用者指南》中的擔任角色的方法。

IAM 角色適用於聯合身分使用者存取、臨時 IAM 使用者許可、跨帳戶存取權與跨服務存取，以及在 Amazon EC2 執行的應用程式。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM 中的快帳戶資源存取。

AWS Device Farm 如何與 IAM 搭配使用

在您使用 IAM 管理 Device Farm 的存取權之前，您應該了解哪些 IAM 功能可與 Device Farm 搭配使用。若要全面了解 Device Farm 和其他 AWS 服務如何與 IAM 搭配使用，請參閱《IAM 使用者指南》中的與 IAM AWS 搭配使用的 服務。

主題

- Device Farm 身分型政策
- Device Farm 資源型政策
- 存取控制清單
- 以 Device Farm 標籤為基礎的授權
- Device Farm IAM 角色

Device Farm 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。Device Farm 支援特定動作、資源和條件索引鍵。若要了解您在 JSON 政策中使用的所有元素，請參閱 IAM 使用者指南中的 JSON 政策元素參考。

動作

管理員可以使用 AWS JSON 政策來指定誰可以存取內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

JSON 政策的 `Action` 元素描述您可以用來允許或拒絕政策中存取的動作。政策會使用動作來授予執行相關聯動作的許可。

Device Farm 中的政策動作在動作之前使用下列字首：`devicefarm:`。例如，若要授予某人使用 Device Farm 桌面瀏覽器測試 `CreateTestGridUrl` API 操作啟動 Selenium 工作階段的許可，請在政策中包含 `devicefarm:CreateTestGridUrl`動作。政策陳述式必須包含 `Action` 或 `NotAction` 元素。Device Farm 會定義自己的一組動作，描述您可以使用此服務執行的任務。

若要在單一陳述式中指定多個動作，請用逗號分隔，如下所示：

```
"Action": [
```

```
      "devicefarm:action1",
      "devicefarm:action2"
```

您也可以使用萬用字元 (*) 來指定多個動作。例如，若要指定開頭是 `List` 文字的所有動作，請包含以下動作：

```
"Action": "devicefarm:List*"
```

若要查看 Device Farm 動作的清單，請參閱《IAM 服務授權參考》中的 定義的動作 AWS Device Farm。

Resources

管理員可以使用 AWS JSON 政策來指定誰可以存取內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

`Resource` JSON 政策元素可指定要套用動作的物件。最佳實務是使用其 Amazon Resource Name (ARN) 來指定資源。若動作不支援資源層級許可，使用萬用字元 (*) 表示該陳述式適用於所有資源。

```
"Resource": "*"
```

Amazon EC2 執行個體資源具有下列 ARN：

```
arn:${Partition}:ec2:${Region}:${Account}:instance/${InstanceId}
```

如需 ARNs 格式的詳細資訊，請參閱 Amazon Resource Name (ARNs) AWS 和服務命名空間。

例如，若要在陳述式中指定 `i-1234567890abcdef0` 執行個體，請使用以下 ARN：

```
"Resource": "arn:aws:ec2:us-east-1:123456789012:instance/i-1234567890abcdef0"
```

若要指定屬於某帳戶的所有執行個體，請使用萬用字元 (*)：

```
"Resource": "arn:aws:ec2:us-east-1:123456789012:instance/*"
```

有些 Device Farm 動作無法在資源上執行，例如用於建立資源的動作。在這些情況下，您必須使用萬用字元 (*)。

```
"Resource": "*"
```

許多 Amazon EC2 API 動作都涉及多個資源。例如，`AttachVolume` 會將 Amazon EBS 磁碟區連接至執行個體，所以 IAM 使用者必須具備該磁碟區與執行個體的使用許可。若要在單一陳述式中指定多項資源，請使用逗號分隔 ARN。

```
"Resource": [
      "resource1",
      "resource2"
```

若要查看 Device Farm 資源類型及其 ARNs的清單，請參閱《IAM 服務授權參考》中的 定義的資源類型 AWS Device Farm。若要了解您可以使用哪些動作指定每個資源的 ARN，請參閱《IAM 服務授權參考》中的 定義的動作 AWS Device Farm。

條件索引鍵

管理員可以使用 AWS JSON 政策來指定誰可以存取內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

`Condition` 元素會根據定義的條件，指定陳述式的執行時機。您可以建立使用條件運算子的條件運算式 (例如等於或小於)，來比對政策中的條件和請求中的值。若要查看所有 AWS 全域條件索引鍵，請參閱《IAM 使用者指南》中的AWS 全域條件內容索引鍵。

Device Farm 會定義自己的一組條件金鑰，也支援使用一些全域條件金鑰。若要查看所有 AWS 全域條件金鑰，請參閱《IAM 使用者指南》中的AWS 全域條件內容金鑰。

若要查看 Device Farm 條件索引鍵的清單，請參閱《IAM 服務授權參考》中的 的條件索引鍵 AWS Device Farm。若要了解您可以使用條件金鑰的動作和資源，請參閱《IAM 服務授權參考》中的 定義的動作 AWS Device Farm。

範例

若要檢視 Device Farm 身分型政策的範例，請參閱 AWS Device Farm 身分型政策範例。

Device Farm 資源型政策

Device Farm 不支援以資源為基礎的政策。

存取控制清單

Device Farm 不支援存取控制清單 ACLs)。

以 Device Farm 標籤為基礎的授權

您可以將標籤連接至 Device Farm 資源，或在請求中將標籤傳遞至 Device Farm。如需根據標籤控制存取，請使用 `aws:ResourceTag/key-name`、`aws:RequestTag/key-name` 或 `aws:TagKeys` 條件索引鍵，在政策的條件元素中，提供標籤資訊。如需標記 Device Farm 資源的詳細資訊，請參閱 Device Farm中的標記。

若要檢視身分型政策範例，以根據該資源上的標籤來限制存取資源，請參閱 根據標籤檢視 Device Farm 桌面瀏覽器測試專案。

Device Farm IAM 角色

IAM 角色是您 AWS 帳戶中具有特定許可的實體。

搭配 Device Farm 使用臨時登入資料

Device Farm 支援使用臨時登入資料。

您可以使用臨時登入資料透過聯合身分登入，以擔任 IAM 角色或跨帳戶角色。您可以透過呼叫 AssumeRole 或 GetFederationToken 等 AWS STS API 操作來取得臨時安全登入資料。

服務連結角色

服務連結角色可讓 AWS 服務存取其他服務中的資源，以代表您完成 動作。服務連結角色會顯示在您的 IAM 帳戶中，並由該服務所擁有。IAM 管理員可以檢視但無法編輯服務連結角色的許可。

Device Farm 在 Device Farm 桌面瀏覽器測試功能中使用服務連結角色。如需這些角色的資訊，請參閱《 開發人員指南》中的在 Device Farm 桌面瀏覽器測試中使用服務連結角色。

服務角色

Device Farm 不支援 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。

使用政策管理存取權

您可以透過建立政策並將其連接到身分或資源 AWS 來控制 AWS 中的存取。政策定義與身分或資源相關聯的許可。當委託人提出請求時 AWS ， 會評估這些政策。大多數政策會以 JSON 文件 AWS 形式存放在 中。如需進一步了解 JSON 政策文件，請參閱《IAM 使用者指南》中的 JSON 政策概觀。

管理員會使用政策，透過定義哪些主體可在哪些條件下對哪些資源執行動作，以指定可存取的範圍。

預設情況下，使用者和角色沒有許可。IAM 管理員會建立 IAM 政策並將其新增至角色，供使用者後續擔任。IAM 政策定義動作的許可，無論採用何種方式執行。

身分型政策

身分型政策是附加至身分 (使用者、使用者群組或角色) 的 JSON 許可政策文件。這類政策控制身分可對哪些資源執行哪些動作，以及適用的條件。如需了解如何建立身分型政策，請參閱《IAM 使用者指南》中的透過客戶管理政策定義自訂 IAM 許可。

身分型政策可分為內嵌政策 (直接內嵌於單一身分) 與受管政策 (可附加至多個身分的獨立政策)。如需了解如何在受管政策及內嵌政策之間做選擇，請參閱《IAM 使用者指南》中的在受管政策與內嵌政策之間選擇。

下表概述 Device Farm AWS 受管政策。

變更	描述	Date
AWSDeviceFarmFullAccess	提供所有 AWS Device Farm 操作的完整存取權。	2015 年 7 月 15 日
AWSServiceRoleForDeviceFarmTestGrid	允許 Device Farm 代表您存取 AWS 資源。	2021 年 5 月 20 日

其他政策類型

AWS 支援其他政策類型，可設定更多常見政策類型授予的最大許可：

- 許可界限 — 設定身分型政策可授與 IAM 實體的最大許可。如需詳細資訊，請參閱《 IAM 使用者指南》中的 IAM 實體許可界限。
- 服務控制政策 (SCP) — 為 AWS Organizations中的組織或組織單位指定最大許可。如需詳細資訊，請參閱《AWS Organizations 使用者指南》中的服務控制政策。
- 資源控制政策 (RCP) — 設定您帳戶中資源可用許可的上限。如需詳細資訊，請參閱《AWS Organizations 使用者指南》中的資源控制政策 (RCP)。
- 工作階段政策 — 在以程式設計方式為角色或聯合身分使用者建立臨時工作階段時，以參數形式傳遞的進階政策。如需詳細資訊，請參《IAM 使用者指南》中的工作階段政策。

多種政策類型

當多種類型的政策適用於請求時，產生的許可會更複雜而無法理解。若要了解如何 AWS 在涉及多種政策類型時決定是否允許請求，請參閱《IAM 使用者指南》中的政策評估邏輯。

AWS Device Farm 身分型政策範例

根據預設，IAM 使用者和角色沒有建立或修改 Device Farm 資源的許可。他們也無法使用 AWS 管理主控台 AWS CLI或 AWS API 執行任務。IAM 管理員必須建立 IAM 政策，授予使用者和角色在指定資源上執行特定 API 作業的所需許可。管理員接著必須將這些政策連接至需要這些許可的 IAM 使用者或群組。

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。

主題

- 政策最佳實務
- 允許使用者檢視他們自己的許可
- 存取一個 Device Farm 桌面瀏覽器測試專案
- 根據標籤檢視 Device Farm 桌面瀏覽器測試專案

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 Device Farm 資源。這些動作可能會讓您的 AWS 帳戶產生費用。當您建立或編輯身分型政策時，請遵循下列準則及建議事項：

- 開始使用 AWS 受管政策並邁向最低權限許可 – 若要開始將許可授予您的使用者和工作負載，請使用將許可授予許多常見使用案例的 AWS 受管政策。它們可在您的 中使用 AWS 帳戶。我們建議您定義特定於使用案例 AWS 的客戶受管政策，以進一步減少許可。如需更多資訊，請參閱《IAM 使用者指南》中的 AWS 受管政策或任務職能的AWS 受管政策。
- 套用最低權限許可 – 設定 IAM 政策的許可時，請僅授予執行任務所需的許可。為實現此目的，您可以定義在特定條件下可以對特定資源採取的動作，這也稱為最低權限許可。如需使用 IAM 套用許可的更多相關資訊，請參閱《IAM 使用者指南》中的 IAM 中的政策和許可。
- 使用 IAM 政策中的條件進一步限制存取權 – 您可以將條件新增至政策，以限制動作和資源的存取。例如，您可以撰寫政策條件，指定必須使用 SSL 傳送所有請求。如果透過特定 等使用服務動作 AWS 服務，您也可以使用條件來授予其存取權 CloudFormation。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM JSON 政策元素：條件。

- 使用 IAM Access Analyzer 驗證 IAM 政策，確保許可安全且可正常運作 – IAM Access Analyzer 驗證新政策和現有政策，確保這些政策遵從 IAM 政策語言 (JSON) 和 IAM 最佳實務。IAM Access Analyzer 提供 100 多項政策檢查及切實可行的建議，可協助您撰寫安全且實用的政策。如需詳細資訊，請參閱《IAM 使用者指南》中的使用 IAM Access Analyzer 驗證政策。
- 需要多重要素驗證 (MFA) – 如果您的案例需要 IAM 使用者或 中的根使用者 AWS 帳戶，請開啟 MFA 以提高安全性。如需在呼叫 API 操作時請求 MFA，請將 MFA 條件新增至您的政策。如需詳細資訊，請參閱《IAM 使用者指南》中的透過 MFA 的安全 API 存取。

如需 IAM 中最佳實務的相關資訊，請參閱《IAM 使用者指南》中的 IAM 安全最佳實務。

允許使用者檢視他們自己的許可

此範例會示範如何建立政策，允許 IAM 使用者檢視附加到他們使用者身分的內嵌及受管政策。此政策包含在主控台或使用 或 AWS CLI AWS API 以程式設計方式完成此動作的許可。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
```

```
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

存取一個 Device Farm 桌面瀏覽器測試專案

在此範例中，您想要授予 AWS 帳戶中的 IAM 使用者存取其中一個 Device Farm destktop 瀏覽器測試專案 `arn:aws:devicefarm:us-west-2:111122223333:testgrid-project:123e4567-e89b-12d3-a456-426655441111`。您希望帳戶能夠查看與專案相關的項目。

除了 `devicefarm:GetTestGridProject` 端點之外，帳戶還必須具有 `devicefarm:ListTestGridSessions`、`devicefarm:GetTestGridSession`、`devicefarm:ListTe` 和 `devicefarm:ListTestGridSessionArtifacts` 端點。

如果使用 CI 系統，您應為每個 CI 執行者提供唯一的存取登入資料。例如，CI 系統需要的許可，不可能超越 `devicefarm:ScheduleRun` 或 `devicefarm:CreateUpload`。下列 IAM 政策概述了允許 CI 執行器透過建立上傳並使用它來排程測試執行來開始新 Device Farm 原生應用程式測試的最小政策：

根據標籤檢視 Device Farm 桌面瀏覽器測試專案

您可以在身分型政策中使用條件，根據標籤控制對 Device Farm 資源的存取。此範例示範如何建立允許檢視專案及工作階段的政策。如果所請求資源的 `Owner` 標籤符合請求帳戶的使用者名稱，即授予許可。

您可以將此政策連接到您帳戶中的 IAM 使用者。如果名為 的使用者`richard-roe`嘗試檢視 Device Farm 專案或工作階段，則該專案必須加上標籤 `Owner=richard-roe`或 `owner=richard-roe`。否則，便會拒絕該使用者存取。條件標籤金鑰 `Owner` 符合 `Owner` 和 `owner`，因為條件金鑰名稱不區分大小寫。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM JSON 政策元素：條件。

故障診斷 AWS Device Farm 身分和存取

使用以下資訊來協助您診斷和修正使用 Device Farm 和 IAM 時可能遇到的常見問題。

我無權在 Device Farm 中執行動作

如果您在 中收到錯誤 AWS 管理主控台 ，指出您無權執行 動作，您必須聯絡管理員尋求協助。您的管理員是提供您使用者名稱和密碼的人員。

當 IAM 使用者 嘗試使用主控台檢視有關執行的詳細資訊，但沒有devicefarm:GetRun許可時mateojackson，會發生下列範例錯誤。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 devicefarm:GetRun on resource: arn:aws:devicefarm:us-west-2:123456789101:run:123e4567-
e89b-12d3-a456-426655440000/123e4567-e89b-12d3-a456-426655441111
```

在本例中，Mateo 要求管理員更新政策，允許其使用 devicefarm:GetRun 動作存取資源 arn:aws:devicefarm:us-west-2:123456789101:run:123e4567-e89b-12d3-a456-426655440000/123e4567-e89b-12d3-a456-426655441111 上的 devicefarm:GetRun 資源。

我未獲得執行 iam:PassRole 的授權

如果您收到錯誤，告知您無權執行 iam:PassRole動作，您的政策必須更新，以允許您將角色傳遞至 Device Farm。

有些 AWS 服務 可讓您將現有角色傳遞給該服務，而不是建立新的服務角色或服務連結角色。如需執行此作業，您必須擁有將角色傳遞至該服務的許可。

當名為 的 IAM marymajor 使用者嘗試使用主控台在 Device Farm 中執行動作時，會發生下列範例錯誤。但是，動作請求服務具備服務角色授予的許可。Mary 沒有將角色傳遞給服務的許可。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

在這種情況下，Mary 的政策必須更新，允許她執行 iam:PassRole 動作。

如果您需要協助，請聯絡您的 AWS 管理員。您的管理員提供您的登入憑證。

我想要檢視我的存取金鑰

在您建立 IAM 使用者存取金鑰後，您可以隨時檢視您的存取金鑰 ID。但是，您無法再次檢視您的私密存取金鑰。若您遺失了密碼金鑰，您必須建立新的存取金鑰對。

存取金鑰包含兩個部分：存取金鑰 ID (例如 AKIAIOSFODNN7EXAMPLE) 和私密存取金鑰 (例如 wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY)。如同使用者名稱和密碼，您必須一起使用存取金鑰 ID 和私密存取金鑰來驗證您的請求。就如對您的使用者名稱和密碼一樣，安全地管理您的存取金鑰。

> **Important**
>
> 請勿將您的存取金鑰提供給第三方，甚至是協助尋找您的標準使用者 ID。透過這樣做，您可以讓某人永久存取您的 AWS 帳戶。

建立存取金鑰對時，您會收到提示，要求您將存取金鑰 ID 和私密存取金鑰儲存在安全位置。私密存取金鑰只會在您建立它的時候顯示一次。若您遺失了私密存取金鑰，您必須將新的存取金鑰新增到您的 IAM 使用者。您最多可以擁有兩個存取金鑰。若您已有兩個存取金鑰，您必須先刪除其中一個金鑰對，才能建立新的金鑰對。若要檢視說明，請參閱《IAM 使用者指南》中的管理存取金鑰。

我是管理員，想要允許其他人存取 Device Farm

若要允許其他人存取 Device Farm，您必須將許可授予需要存取的人員或應用程式。如果您使用 AWS IAM Identity Center 來管理人員和應用程式，您可以將許可集指派給使用者或群組，以定義其存取層級。許可集會自動建立 IAM 政策，並將其指派給與該人員或應用程式相關聯的 IAM 角色。如需詳細資訊，請參閱AWS IAM Identity Center 《 使用者指南》中的許可集。

如果您不是使用 IAM Identity Center，則必須為需要存取的人員或應用程式建立 IAM 實體（使用者或角色）。然後，您必須將政策連接至實體，以授予其在 Device Farm 中的正確許可。授予許可後，請將登入資料提供給使用者或應用程式開發人員。他們將使用這些登入資料來存取 AWS。若要進一步了解如何建立 IAM 使用者、群組、政策和許可，請參閱《IAM 使用者指南》中的 IAM 身分和政策和許可。

我想要允許 AWS 帳戶外的人員存取我的 Device Farm 資源

您可以建立一個角色，讓其他帳戶中的使用者或您組織外部的人員存取您的資源。您可以指定要允許哪些信任物件取得該角色。針對支援基於資源的政策或存取控制清單 (ACL) 的服務，您可以使用那些政策來授予人員存取您的資源的許可。

如需進一步了解，請參閱以下內容：

- 若要了解 Device Farm 是否支援這些功能，請參閱 AWS Device Farm 如何與 IAM 搭配使用。
- 若要了解如何在您擁有 AWS 帳戶 的 資源之間提供存取權，請參閱《IAM 使用者指南》中的在您擁有 AWS 帳戶 的另一個 中提供存取權給 IAM 使用者。
- 若要了解如何將資源的存取權提供給第三方 AWS 帳戶，請參閱《IAM 使用者指南》中的將存取權提供給第三方 AWS 帳戶 擁有。
- 如需了解如何透過聯合身分提供存取權，請參閱《IAM 使用者指南》中的將存取權提供給在外部進行身分驗證的使用者 (聯合身分)。

- 如需了解使用角色和資源型政策進行跨帳戶存取之間的差異，請參閱《IAM 使用者指南》中的 IAM 中的跨帳戶資源存取。

的合規驗證 AWS Device Farm

在多個合規計畫中 AWS Device Farm ，第三方稽核人員會評估 的安全與 AWS 合規。其中包括 SOC、PCI、FedRAMP、HIPAA 等。 AWS Device Farm 不在任何 AWS 合規計劃範圍內。

如需特定合規計劃範圍內 AWS 的服務清單，請參閱合規計劃範圍內的 AWS 服務。如需一般資訊，請參閱AWS 合規計劃。

您可以使用 下載第三方稽核報告 AWS Artifact。如需詳細資訊，請參閱下載 AWS Artifact 中的報告。

您使用 Device Farm 時的合規責任取決於資料的機密性、您公司的合規目標，以及適用的法律和法規。 AWS 提供下列資源以協助合規：

- 安全與合規快速入門指南：這些部署指南討論架構考量，並提供在 AWS上部署以安全及合規為重心之基準環境的步驟。
- AWS 合規資源 – 此工作手冊和指南集合可能適用於您的產業和位置。
- 《 AWS Config 開發人員指南》中的使用 規則評估資源 - AWS Config 評估資源組態符合內部實務、產業準則和法規的程度。
- AWS Security Hub CSPM – AWS 此服務提供 內安全狀態的完整檢視 AWS ，可協助您檢查是否符合安全產業標準和最佳實務。

中的資料保護 AWS Device Farm

AWS 共同責任模型適用於 AWS Device Farm (Device Farm) 中的資料保護。如此模型所述， AWS 負責保護執行所有 的全域基礎設施 AWS 雲端。您負責維護在此基礎設施上託管內容的控制權。您也同時負責所使用 AWS 服務 的安全組態和管理任務。如需資料隱私權的詳細資訊，請參閱資料隱私權常見問答集。如需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 共同責任模型和 GDPR 部落格文章。

基於資料保護目的，我們建議您保護 AWS 帳戶 登入資料，並使用 AWS IAM Identity Center 或 AWS Identity and Access Management (IAM) 設定個別使用者。如此一來，每個使用者都只會獲得授與完成其任務所必須的許可。我們也建議您採用下列方式保護資料：

- 每個帳戶均要使用多重要素驗證 (MFA)。

- 使用 SSL/TLS 與 AWS 資源通訊。我們需要 TLS 1.2 並建議使用 TLS 1.3。
- 使用 設定 API 和使用者活動記錄 AWS CloudTrail。如需有關使用 CloudTrail 追蹤擷取 AWS 活動的資訊，請參閱AWS CloudTrail 《 使用者指南》中的使用 CloudTrail 追蹤。
- 使用 AWS 加密解決方案，以及其中的所有預設安全控制 AWS 服務。
- 使用進階的受管安全服務 (例如 Amazon Macie)，協助探索和保護儲存在 Amazon S3 的敏感資料。
- 如果您在 AWS 透過命令列界面或 API 存取 時需要 FIPS 140-3 驗證的密碼編譯模組，請使用 FIPS 端點。如需有關 FIPS 和 FIPS 端點的更多相關資訊，請參閱聯邦資訊處理標準 (FIPS) 140-3。

我們強烈建議您絕對不要將客戶的電子郵件地址等機密或敏感資訊，放在標籤或自由格式的文字欄位中，例如名稱欄位。這包括當您使用 Device Farm 或使用主控台、API AWS CLI或其他 AWS 服務 AWS SDKs 時。您在標籤或自由格式文字欄位中輸入的任何資料都可能用於計費或診斷日誌。如果您提供外部伺服器的 URL，我們強烈建議請勿在驗證您對該伺服器請求的 URL 中包含憑證資訊。

傳輸中加密

Device Farm 端點僅支援已簽署的 HTTPS (SSL/TLS) 請求，除非另有說明。透過上傳 URL 從 Amazon S3 擷取或放置在 Amazon S3 中的所有內容都會使用 SSL/TLS 加密。 URLs 如需如何登入 HTTPS 請求的詳細資訊 AWS，請參閱《 AWS 一般參考》中的簽署 AWS API 請求。

您必須負責加密並保護受測應用程式所建立的所有通訊，以及在裝置上執行測試過程中所安裝的任何應用程式。

靜態加密

Device Farm 的桌面瀏覽器測試功能支援測試期間產生的成品靜態加密。

Device Farm 的實體行動裝置測試資料不會靜態加密。

資料保留

Device Farm 中的資料會保留一段時間。保留期間過期後，資料會從 Device Farm 的備份儲存中移除。

內容類型	保留期間 (天)	中繼資料保留期 （ 天 ）
上傳的應用程式	30	30
上傳的測試套件	30	30

內容類型	保留期間 (天)	中繼資料保留期 （天）
Logs (日誌)	400	400
影片和其他成品	400	400

您有責任儲存想保留較長時間的任何內容。

資料管理

Device Farm 中的資料管理方式會因使用的功能而有所不同。本節說明如何在使用 Device Farm 期間和之後管理資料。

桌面瀏覽器測試

不儲存 Selenium 工作階段期間使用的執行個體。工作階段結束後，即捨棄所有因瀏覽器互動而產生的資料。

此功能目前針對測試期間產生的成品支援靜態加密。

實體裝置測試

下列各節提供使用 Device Farm 後清除或銷毀裝置所 AWS 採取步驟的相關資訊。

Device Farm 的實體行動裝置測試資料不會靜態加密。

公有裝置機群

測試執行完成後，Device Farm 會在公有裝置機群中的每個裝置上執行一系列清除任務，包括解除安裝您的應用程式。如果我們無法驗證應用程式的解除安裝或任何其他清除步驟，則在重新使用裝置之前，會將其重設成出廠預設值。

> **Note**
>
> 在某些情況下，資料可能會在工作階段之間保留，尤其是當您在應用程式內容之外使用裝置系統時。因此，由於 Device Farm 會擷取在您使用每個裝置期間發生的活動影片和日誌，因此建議您不要在自動化測試和遠端存取工作階段期間輸入敏感資訊 （例如 Google 帳戶或 Apple ID)、個人資訊和其他安全敏感詳細資訊。

私有裝置

在您的私有裝置合約過期或終止之後，即無法使用該裝置，並會根據 AWS 銷毀政策安全地將其銷毀。如需詳細資訊，請參閱AWS Device Farm 中的私有裝置。

金鑰管理

目前，Device Farm 不提供任何外部金鑰管理來加密靜態或傳輸中的資料。

網際網路流量隱私權

Device Farm 只能針對私有裝置設定為使用 Amazon VPC 端點連線到 中的資源 AWS。存取與您的帳戶相關聯的任何非公有 AWS 基礎設施（例如，沒有公有 IP 地址的 Amazon EC2 執行個體）必須使用 Amazon VPC 端點。無論 VPC 端點組態為何，Device Farm 都會將您的流量與整個 Device Farm 網路的其他使用者隔離。

您 AWS 網路外的連線不保證安全無虞，而且您有責任保護應用程式建立的任何網際網路連線。

中的彈性 AWS Device Farm

AWS 全球基礎設施是以 AWS 區域和可用區域為基礎建置的。 AWS 區域提供多個實體隔離且隔離的可用區域，這些區域以低延遲、高輸送量和高度備援的網路連接。透過可用區域，您可以設計與操作的應用程式和資料庫，在可用區域之間自動容錯移轉而不會發生中斷。可用區域的可用性、容錯能力和擴展能力，均較單一或多個資料中心的傳統基礎設施還高。

如需 AWS 區域和可用區域的詳細資訊，請參閱 AWS 全球基礎設施。

由於 Device Farm 僅適用於 `us-west-2`區域，因此我們強烈建議您實作備份和復原程序。Device Farm 不應是任何上傳內容的唯一來源。

Device Farm 不保證公有裝置的可用性。這些裝置會因為故障率和隔離狀態等各種因素而放入或移出公有裝置集區。建議您不要依賴公有裝置集區中任何一部裝置的可用性。

中的基礎設施安全性 AWS Device Farm

作為受管服務， AWS Device Farm 受到 AWS 全球網路安全的保護。如需 AWS 安全服務以及如何 AWS 保護基礎設施的資訊，請參閱AWS 雲端安全。若要使用基礎設施安全的最佳實務來設計您的 AWS 環境，請參閱安全支柱 AWS Well-Architected Framework 中的基礎設施保護。

您可以使用 AWS 發佈的 API 呼叫，透過網路存取 Device Farm。使用者端必須支援下列專案：

- Transport Layer Security (TLS)。我們需要 TLS 1.2 並建議使用 TLS 1.3。
- 具備完美轉送私密(PFS)的密碼套件，例如 DHE (Ephemeral Diffie-Hellman)或 ECDHE (Elliptic Curve Ephemeral Diffie-Hellman)。現代系統(如 Java 7 和更新版本)大多會支援這些模式。

此外，請求必須使用存取金鑰 ID 和與 IAM 主體相關聯的私密存取金鑰來簽署。或者，您可以透過 AWS Security Token Service (AWS STS) 來產生暫時安全憑證來簽署請求。

實體裝置測試的基礎設施安全性

裝置在實體裝置測試期間實際上是隔開的。網路隔離可防止透過無線網路進行跨裝置通訊。

共用公有裝置，且 Device Farm 會盡最大努力確保裝置安全。諸如嘗試取得裝置的完整管理員權限 (被稱為 rooting 或 jailbreaking 的實務作法) 一類的某些動作會導致公有裝置遭到隔離。它們會被自動移出公有集區，等候進行人工審查。

只有明確授權 AWS 的帳戶才能存取私有裝置。Device Farm 會將這些裝置與其他裝置實體隔離，並將其保存在不同的網路上。

在私有受管裝置上，測試可以設定為使用 Amazon VPC 端點來保護您 AWS 帳戶內外的連線。

桌面瀏覽器測試的基礎設施安全性

當您使用桌面瀏覽器測試功能時，所有測試工作階段都會彼此分開。在沒有外部的中繼第三方的情況下，硒執行個體無法跨通訊 AWS。

所有通往 Selenium WebDriver 控制器的流量都必須通過使用 `createTestGridUrl` 產生的 HTTPS 端點。

您有責任確保每個 Device Farm 測試執行個體都能安全地存取其測試的資源。根據預設，Device Farm 的桌面瀏覽器測試執行個體可存取公有網際網路。當您將執行個體連接到 VPC 時，其行為就像任何其他 EC2 執行個體一樣，可存取由 VPC 組態及其相關聯聯網元件決定的資源。AWS 提供 安全群組和 網路存取控制清單 (ACLs) ，以提高 VPC 的安全性。安全群組控制資源的傳入與傳出流量，網路 ACL 則是控制子網的傳入與傳出流量。安全群組可為大多數子網路提供足夠的存取控制。如果您想讓 VPC 多一層安全，可以使用網路 ACL。如需使用 Amazon VPCs 時安全最佳實務的一般準則，請參閱《 Amazon Virtual Private Cloud 使用者指南 》中的 VPC 安全最佳實務。

Device Farm 中的組態漏洞分析和管理

Device Farm 可讓您執行廠商未主動維護或修補的軟體，例如作業系統廠商、硬體廠商或電信業者。Device Farm 會盡最大努力維護最新的軟體，但不保證實體裝置上任何特定版本的軟體都是最新的，方法是允許使用潛在易受攻擊的軟體。

例如，如果在執行 Android 4.4.2 的裝置上執行測試，Device Farm 不保證裝置會針對 Android 中稱為 StageFright 的漏洞進行修補。是否為裝置提供安全性更新則取決於裝置的廠商 (有時是電信業者)。我們的自動隔離不保證能捕獲利用此漏洞的惡意應用程式。

私有裝置會根據您與 的協議進行維護 AWS。

Device Farm 會盡最大努力防止客戶應用程式遭到破壞或破解等動作。Device Farm 會從公有集區中移除隔離的裝置，直到手動檢閱為止。

您有責任將測試中使用的任何程式庫或軟體版本保持在最新狀態，例如 Python wheel 和 Ruby gem。Device Farm 建議您更新測試程式庫。

這些資源有助於讓您的測試相依性保持最新狀態：

- 如需如何保護 Ruby gem 的詳細資訊，請參閱 RubyGems 網站上的安全實務 。
- 如需 Pipenv 使用且經 Python 封裝授權單位認可的安全套件，以掃描您的相依性是否有已知漏洞的詳細資訊，請參閱 GitHub 上的安全漏洞偵測。
- 如需開放式 Web 應用程式安全專案 (OWASP) Maven 相依性檢查工具的詳細資訊，請參閱 OWASP 網站上的 OWASP DependencyCheck。

請務必記住，即使自動化系統不認為有任何已知的安全問題，不表示真的沒有安全問題。使用第三方的程式庫或工具時，請務必盡職調查，並在可能或合理的情況下驗證加密簽名。

Device Farm 中的事件回應

Device Farm 會持續監控裝置是否有可能表示安全問題的行為。如果 AWS 注意到客戶資料，例如測試結果或寫入公有裝置的檔案，可由其他客戶存取， 會根據整個 AWS 服務中使用的標準事件提醒和報告政策 AWS 聯絡受影響的客戶。

在 Device Farm 中記錄和監控

此服務支援 AWS CloudTrail，這項服務會記錄 AWS 的呼叫 AWS 帳戶 ，並將日誌檔案傳送到 Amazon S3 儲存貯體。透過使用 CloudTrail 所收集的資訊，您可以判斷成功提出的請求 AWS 服務、

提出請求的人員、提出請求的時間等等。欲進一步了解 CloudTrail，包括如何將其開啟並尋找您的日誌檔案，請參閱《AWS CloudTrail 使用者指南》。

如需搭配 Device Farm 使用 CloudTrail 的詳細資訊，請參閱 使用 記錄 AWS Device Farm API 呼叫 AWS CloudTrail。

Device Farm 的安全最佳實務

Device Farm 提供多種安全功能，供您在開發和實作自己的安全政策時加以考量。以下最佳實務為一般準則，並不代表完整的安全解決方案。這些最佳實務可能不適用或無法滿足您的環境需求，因此請將其視為實用建議就好，而不要當作是指示。

- 將您在 IAM 下使用的最低權限，授予任何持續整合 (CI) 系統。考慮每個 CI 系統測試都使用暫時登入資料，如此一來，即使 CI 系統遭盜用，也不能發出虛假請求。如需暫時登入資料的詳細資訊，請參閱 IAM 使用者指南。
- 在自訂測試環境中使用 adb 命令，清除應用程式建立的所有內容。如需自訂測試環境的詳細資訊，請參閱自訂測試環境。

AWS Device Farm 中的限制

主題

服務限制

- 測試執行中可以包含的裝置數量沒有限制。不過，Device Farm 在測試執行期間同時測試的裝置數量上限為 5 個。服務團隊可應要求增加此數量，並依個別案例進行評估。
- 您可以排定的執行次數沒有限制。請注意，它們最多只能保持佇列 24 小時。
- 遠端存取工作階段的持續時間有 150 分鐘的硬性限制。
- 自動化測試執行期間有 150 分鐘的硬性限制
- 處理中的任務數量上限為 250 個，包括您帳戶中的待定佇列任務。這是軟性限制。
- 您可以在測試執行中包含的裝置數目沒有限制。在任何指定時間可以平行執行測試的裝置（任務）數目等於您的帳戶層級並行。Device Farm 中計量使用的預設帳戶層級並行為 5。
- 視使用案例而定，可依請求將計量並行限制提高到特定閾值。未計量使用的預設帳戶層級並行等於您為該平台訂閱的插槽數量。

  如需預設計量並行限制或一般配額的詳細資訊，請參閱配額頁面。
- 不使用自訂測試環境的自動化執行最多只能有 250 個個別測試案例。否則，可能會略過執行。

檔案限制

- 您可以上傳的應用程式檔案大小上限為 4 GB。請注意，我們目前不接受 Android 的 .aab 格式檔案。
- 在測試執行期間，Device Farm 自動產生的影片大小上限為 1GB。任何超過此大小的影片都會截斷所有剩餘的影片內容。如果存在，客戶仍然可以使用自己的影片錄製解決方案，並將其存放在 Device Farm 受管儲存體之外。

- 測試執行期間，Device Farm 自動產生的裝置日誌 (Android 上的日誌或 iOS 上的 syslog) 的大小上限為 1GB。任何超過此大小的日誌都會截斷所有剩餘的日誌。對於大於 1 GB 的日誌，客戶可以將這些日誌存放在 Device Farm 受管儲存之外。
- Device Farm 自訂環境模式客戶成品的累積大小上限為 1GB。如果您的成品超過此大小，則無法使用任何成品。
- 如果測試執行期間產生的所有成品的累積大小超過 4GB，則可能會捨棄某些成品（包括影片、裝置日誌和客戶成品）。

API 限制

- Device Farm 遵循權杖儲存貯體演算法來調節 API 呼叫率。例如，假設建立包含字符的儲存貯體。每個權杖代表一個交易，而一個 API 呼叫會使用權杖。權杖會以固定速率（例如每秒 10 個權杖）新增至儲存貯體，且儲存貯體具有最大容量（例如 100 個權杖）。請求或封包送達時，必須向要處理的儲存貯體請求權杖。如果有足夠的權杖，則允許透過 請求，並移除權杖。如果權杖不足，請求會延遲或捨棄，視實作而定。

  在 Device Farm 中，以下是演算法的實作方式：

  - 爆量 API 請求是服務能夠為指定客戶帳戶 ID 中的指定 API 回應的請求數目上限。換句話說，這是儲存貯體的容量。您可以多次呼叫 API，因為儲存貯體中還有權杖，而且每個請求都會使用一個權杖。
  - Transactions-per-second數 (TPS) 速率是可以執行 API 請求的最低速率。換句話說，這是儲存貯體每秒以字符重新填充的速率。例如，如果 API 的爆量為十，但 TPS 為一，您可以立即呼叫它十次。不過，除非您停止呼叫 API 讓儲存貯體重新填充，否則儲存貯體只會以每秒一個權杖的速率重新取得權杖，導致調節為每秒一個呼叫。

以下是 Device Farm APIs費率：

- 對於列出和取得 APIs，爆量 API 請求容量為 50，Transactions-per-second (TPS) 速率為 10。
- 對於所有其他 APIs，爆量 API 請求容量為 10，Transactions-per-second (TPS) 速率為 1。

Appium 端點限制

下列限制適用於所有 Appium 端點工作階段。有關如何最佳處理限制的問題和指導，請開啟支援案例。

- 每個 Appium 命令的執行持續時間限制為 4 分鐘，之後命令會逾時。

- 端點接受高達 20MB 的輸入承載大小，並允許高達 20MB 的輸出承載大小。輸入或輸出大小大於此值的任何請求都會收到 的 WebDriver 錯誤`'unsupported operation'`。
- 請求會依收到的順序在裝置上循序執行。因此，強烈建議您依序傳送命令，並在傳送新的命令之前等待每個命令的回應。也就是說，某些 Appium 伺服器命令可以平行傳送，特別是：
  - getStatus
  - getSessions
- 端點目前不支援 WebDriver BiDi 通訊協定。
- 端點不支援 Appium 外掛程式或 XCUITest 和 UIAutomator2 驅動程式以外的驅動程式。
- 最多可以用 3 個應用程式做為具有遠端存取工作階段建立請求的輔助應用程式。也就是說，使用 `InstallToRemoteAccessSession` API 的工作階段期間可以安裝多少應用程式沒有限制。

自訂環境變數限制

下列限制適用於所有自訂環境變數。有關如何最佳處理限制的問題和指導，請開啟支援案例。

- 指定 Device Farm 專案或執行中最多可設定 32 個變數。
- 變數名稱長度不可超過 256 個字元。
- 變數名稱受到 施加的限制`bash`。也就是說，它們只能包含英數字元和底線字元，而且不能以數字開頭。
- 以 開頭的變數名稱`$DEVICEFARM_`會保留供內部服務使用。
- 變數值長度不能超過 256 個字元。
- 環境變數無法用於設定測試規格檔案中的測試主機運算選擇。

AWS Device Farm 的工具和外掛程式

本節包含使用 AWS Device Farm 工具和外掛程式的連結和資訊。您可以在 GitHub 上的 AWS Labs 上找到 Device Farm 外掛程式。

如果您是 Android 開發人員，我們也在 GitHub 上提供適用於 Android 的 AWS Device Farm 範例應用程式。您可以使用應用程式和範例測試做為您自己的 Device Farm 測試指令碼的參考。

主題

- 將 Device Farm 與 Jenkins CI 伺服器整合
- 將 Device Farm 與 Gradle 建置系統整合

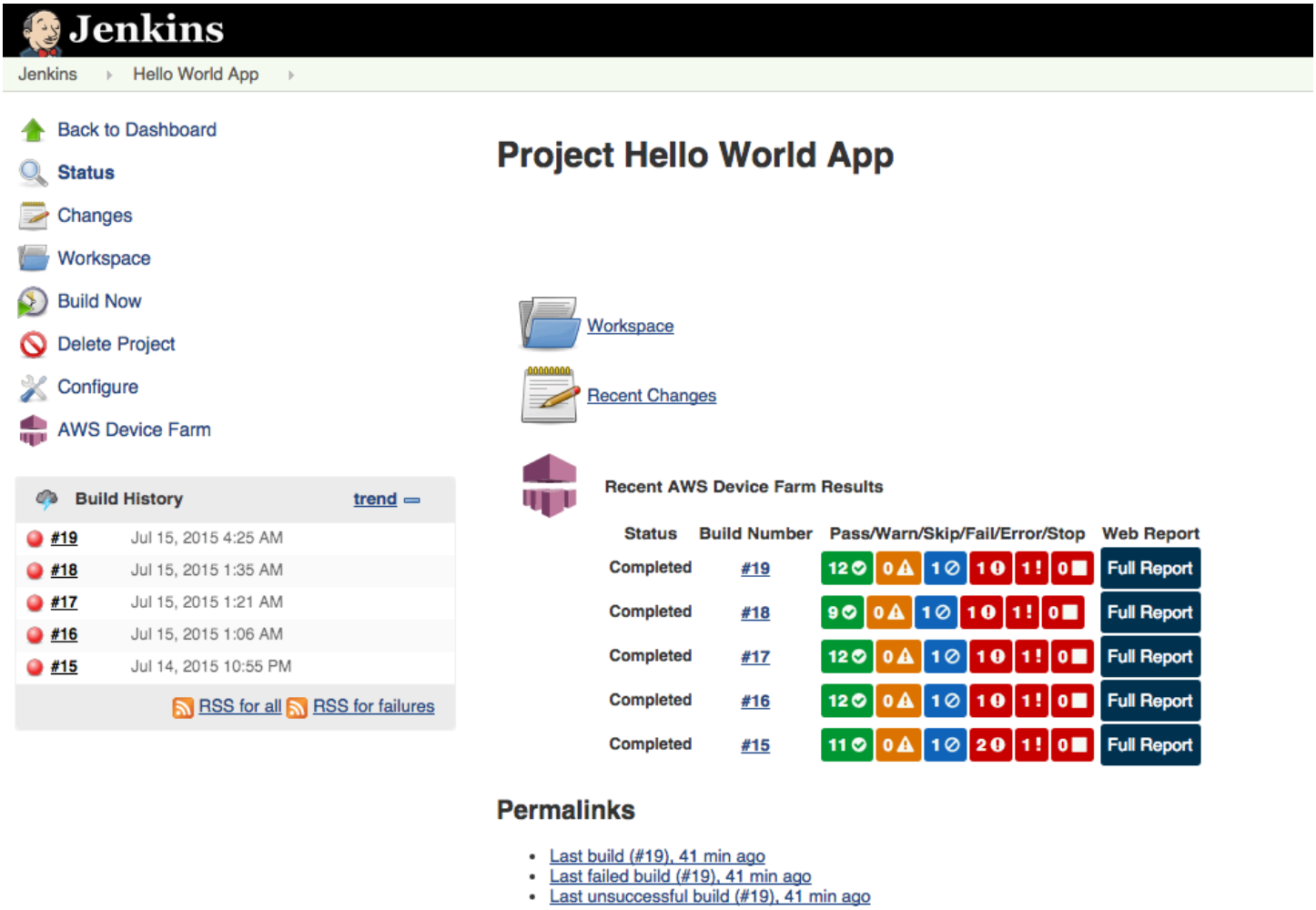
將 Device Farm 與 Jenkins CI 伺服器整合

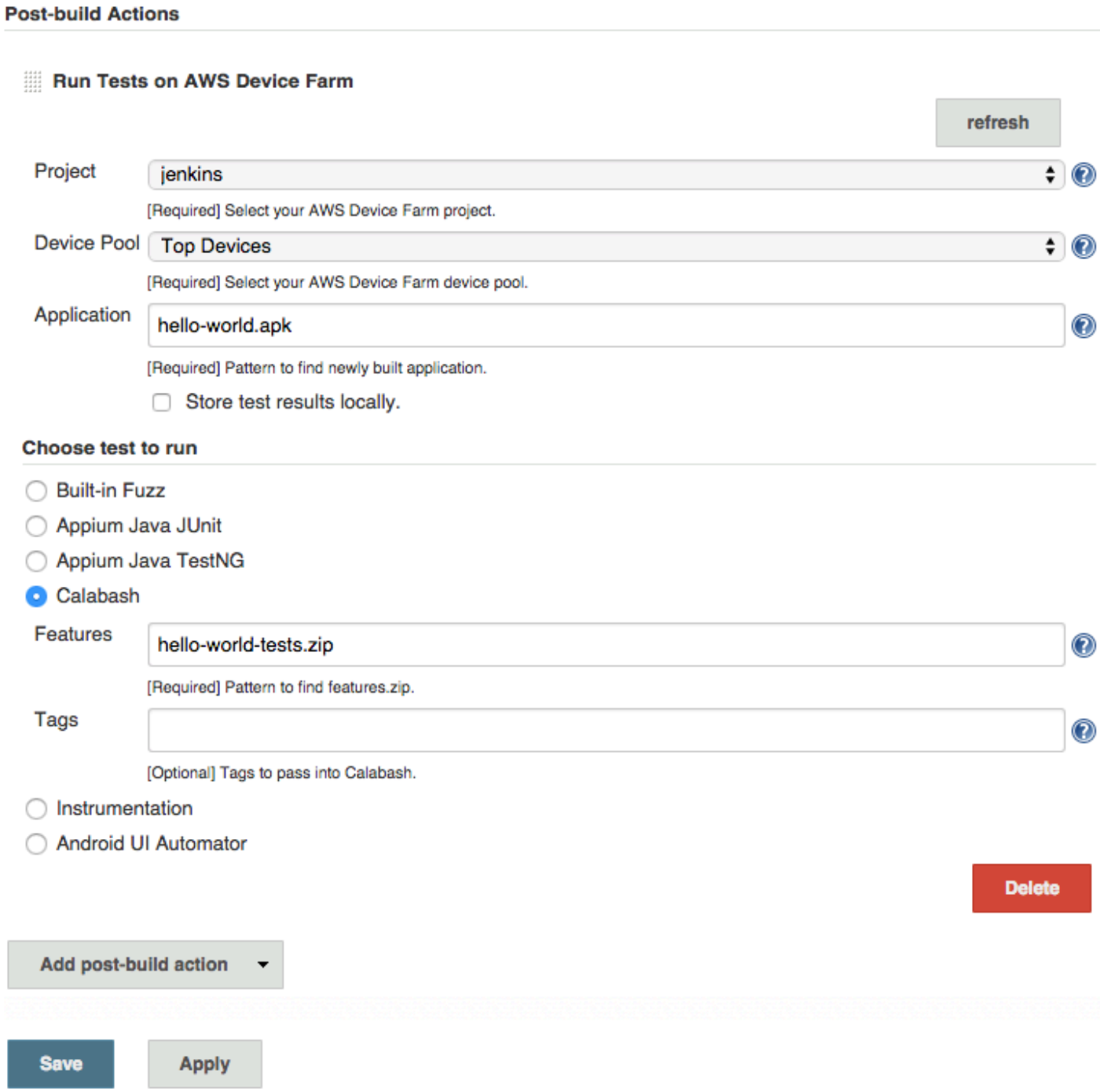
Jenkins CI 外掛程式可從您自己的 Jenkins 持續整合 (CI) 伺服器提供 AWS Device Farm 功能。如需詳細資訊，請參閱 Jenkins (軟體)。

> **Note**
>
> 若要下載 Jenkins 外掛程式，請前往 GitHub 並依照 步驟 1：安裝 AWS Device Farm 的 Jenkins CI 外掛程式 中的說明操作。

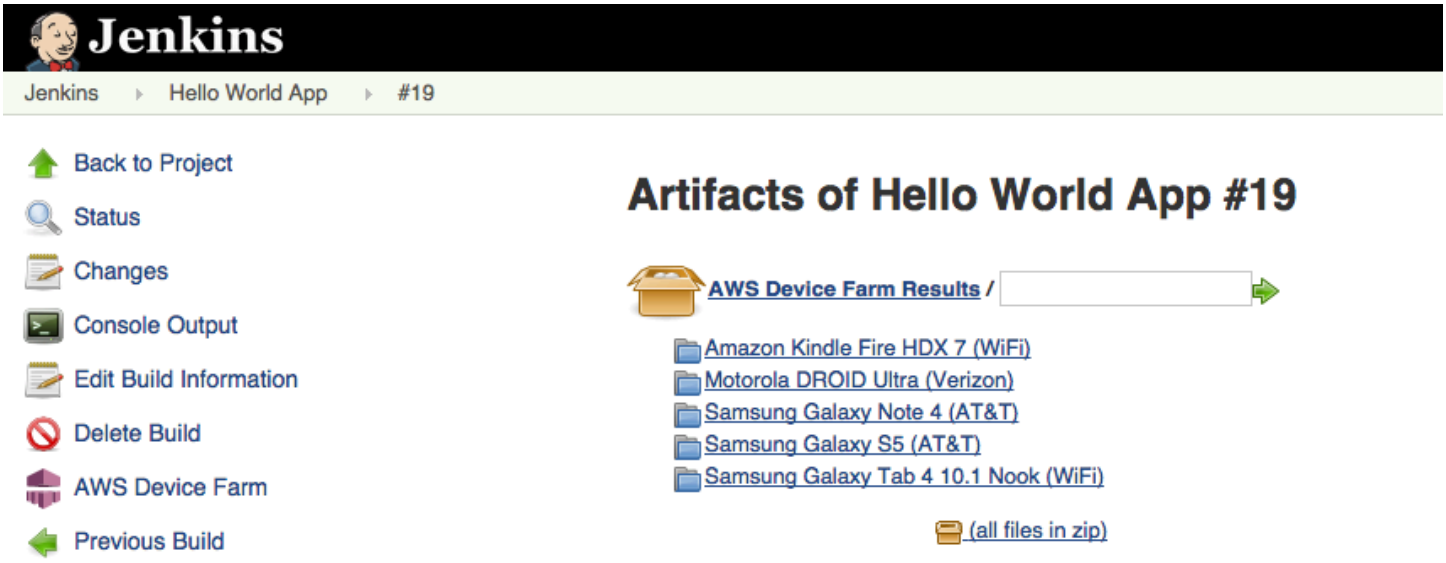
本節包含一系列設定和使用 Jenkins CI 外掛程式搭配 AWS Device Farm 的程序。

以下影像顯示 Jenkins CI 外掛程式的功能。


Jenkins
Jenkins ▸ Hello World App ▸
Back to Dashboard
Status
Changes
Workspace
Build Now
Delete Project
Configure
AWS Device Farm
Build History trend
#19 Jul 15, 2015 4:25 AM
#18 Jul 15, 2015 1:35 AM
#17 Jul 15, 2015 1:21 AM
#16 Jul 15, 2015 1:06 AM
#15 Jul 14, 2015 10:55 PM
RSS for all RSS for failures
Project Hello World App
Workspace
Recent Changes
Recent AWS Device Farm Results
Status Build Number Pass/Warn/Skip/Fail/Error/Stop Web Report
Completed #19 12 0 1 1 1 0 Full Report
Completed #18 9 0 1 1 1 0 Full Report
Completed #17 12 0 1 1 1 0 Full Report
Completed #16 12 0 1 1 1 0 Full Report
Completed #15 11 0 1 2 1 0 Full Report
Permalinks
Last build (#19), 41 min ago
Last failed build (#19), 41 min ago
Last unsuccessful build (#19), 41 min ago



外掛程式也可以在本機提取所有測試成品 (日誌、螢幕擷取畫面等)：



主題

相依性

Jenkins CI 外掛程式需要 AWS Mobile SDK 1.10.5 或更新版本。如需安裝軟體開發套件的詳細資訊，請參閱 AWS Mobile SDK。

步驟 1：安裝 AWS Device Farm 的 Jenkins CI 外掛程式

有兩種選項可安裝 AWS Device Farm 的 Jenkins 持續整合 (CI) 外掛程式。您可以從 Jenkins Web UI 中的 Available Plugins (可用外掛程式) 對話方塊內搜尋外掛程式，或您可以從 Jenkins 內下載 `hpi` 檔案並安裝它。

從 Jenkins UI 內安裝

1. 在 Jenkins UI 內尋找外掛程式，方法為選擇 Manage Jenkins (管理 Jenkins)、Manage Plugins (管理外掛程式)，然後選擇 Available (可用)。

2. 搜尋 aws-device-farm。
3. 安裝 AWS Device Farm 外掛程式。
4. 確保外掛程式是由 `Jenkins` 使用者擁有。
5. 重新啟動 Jenkins。

下載外掛程式

1. 直接從 https：//http://updates.jenkins-ci.org/latest/aws-device-farm.hpi 下載`hpi`檔案。
2. 確保外掛程式是由 `Jenkins` 使用者擁有。
3. 使用以下其中一個選項安裝外掛程式：
   - 選擇 Manage Jenkins (管理 Jenkins)、Manage Plugins (管理外掛程式)、Advanced (進階)，然後選擇 Upload plugin (上傳外掛程式) 來上傳外掛程式。
   - 將 `hpi` 檔案放在 Jenkins 外掛程式目錄 (通常為 `/var/lib/jenkins/plugins`) 中。
4. 重新啟動 Jenkins。

步驟 2：為 AWS Device Farm 的 Jenkins CI 外掛程式建立 AWS Identity and Access Management 使用者

我們建議您不要使用 AWS 根帳戶來存取 Device Farm。請改為在帳戶中 AWS 建立新的 AWS Identity and Access Management (IAM) 使用者 （或使用現有的 IAM 使用者），然後使用該 IAM 使用者存取 Device Farm。

若要建立新的 IAM 使用者，請參閱建立 IAM 使用者 (AWS 管理主控台)。務必為每個使用者產生存取金鑰，並下載或儲存使用者安全登入資料。稍後您需要登入資料。

授予 IAM 使用者存取 Device Farm 的許可

若要授予 IAM 使用者存取 Device Farm 的許可，請在 IAM 中建立新的存取政策，然後將存取政策指派給 IAM 使用者，如下所示。

> **Note**
>
> 您用來完成下列步驟的 AWS 根帳戶或 IAM 使用者必須具有建立下列 IAM 政策並將其連接至 IAM 使用者的許可。如需詳細資訊，請參閱使用政策

在 IAM 中建立存取政策

1. 前往 https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。
2. 選擇政策。
3. 選擇建立政策。(出現 Get Started (開始使用) 按鈕時先選擇它，然後選擇 Create Policy (建立政策)。)
4. 在建立您自己的政策旁邊，選擇選取。
5. 針對 Policy Name (政策名稱)，輸入政策的名稱 (例如，**AWSDeviceFarmAccessPolicy**)。
6. 針對描述，輸入可協助您將此 IAM 使用者與 Jenkins 專案建立關聯的描述。
7. 針對 Policy Document (政策文件)，輸入下列聲明：

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DeviceFarmAll",
            "Effect": "Allow",
            "Action": [ "devicefarm:*" ],
            "Resource": [ "*" ]
        }
    ]
}
```

8. 選擇建立政策。

將存取政策指派給 IAM 使用者

1. 前往 https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。
2. 選擇 Users (使用者)。
3. 選擇您要為其指派存取政策的 IAM 使用者。
4. 在 Permissions (許可) 區域中，針對 Managed Policies (受管政策)，選擇 Attach Policy (連接政策)。
5. 選取您剛建立的政策 （ 例如，AWSDeviceFarmAccessPolicy)。
6. 選擇 Attach Policy (連接政策)。

步驟 3：第一次在 AWS Device Farm 中設定 Jenkins CI 外掛程式

第一次執行 Jenkins 伺服器時，您將需要設定系統，如下所示。

> **Note**
>
> 如果您使用的是裝置插槽，則裝置插槽功能預設為停用。

1. 登入您的 Jenkins Web 使用者界面。
2. 在畫面左側，選擇 Manage Jenkins (管理 Jenkins)。
3. 選擇 Configure System (設定系統)。
4. 向下捲動至 AWS Device Farm 標頭。
5. 從 為您的 Jenkins CI 外掛程式建立 IAM 使用者 複製您的安全登入資料，並將您的存取金鑰 ID 和私密存取金鑰貼至其各自方塊。
6. 選擇儲存。

步驟 4：在 Jenkins 任務中使用外掛程式

一旦您已安裝 Jenkins 外掛程式，請按照這些指示，在 Jenkins 任務中使用這個外掛程式。

1. 登入您的 Jenkins Web UI。
2. 按一下您要編輯的任務。
3. 在畫面左側，選擇 Configure (設定)。
4. 向下捲動到 Post-build Actions (後置建置動作) 標頭。
5. 按一下新增建置後動作，然後選取在 AWS Device Farm 上執行測試。
6. 選擇您要使用的專案。
7. 選擇您要使用的裝置集區。
8. 選取您是否想要將測試成品 (例如，日誌和螢幕擷取畫面) 封存在本機。
9. 在 Application (應用程式) 中，填入您已編譯之應用程式的路徑。
10. 選取您要執行的測試，並填寫所有必要的欄位。
11. 選擇儲存。

將 Device Farm 與 Gradle 建置系統整合

Device Farm Gradle 外掛程式提供 AWS Device Farm 與 Android Studio 中 Gradle 建置系統的整合。如需詳細資訊，請參閱 Gradle。

> **Note**
>
> 若要下載 Gradle 外掛程式，請前往 GitHub 並依照 建置 Device Farm Gradle 外掛程式 中的說明操作。

Device Farm Gradle 外掛程式可從 Android Studio 環境提供 Device Farm 功能。您可以在 Device Farm 託管的真實 Android 手機和平板電腦上開始測試。

本節包含一系列設定和使用 Device Farm Gradle 外掛程式的程序。

主題

相依性

執行時間

- Device Farm Gradle 外掛程式需要 AWS Mobile SDK 1.10.15 或更新版本。如需安裝軟體開發套件的詳細資訊，請參閱 AWS Mobile SDK。
- Android tools builder test api 0.5.2
- Apache Commons Lang3 3.3.4

適用於單位測試

- Testng 6.8.8
- Jmockit 1.19

- Android gradle 工具 1.3.0

步驟 1：建置 AWS Device Farm Gradle 外掛程式

此外掛程式提供 AWS Device Farm 與 Android Studio 中 Gradle 建置系統的整合。如需詳細資訊，請參閱 Gradle。

> **Note**
>
> 建置外掛程式為選用。外掛程式會透過 Maven Central 發佈。如果您希望允許 Gradle 直接下載外掛程式，請略過此步驟並跳到 步驟 2：設定 AWS Device Farm Gradle 外掛程式。

建置外掛程式

1. 前往 GitHub 並複製儲存庫。
2. 使用 `gradle install` 建置外掛程式

   外掛程式已安裝至您的本機 Maven 儲存庫。

後續步驟：步驟 2：設定 AWS Device Farm Gradle 外掛程式

步驟 2：設定 AWS Device Farm Gradle 外掛程式

如果您尚未執行，請使用此處的程序複製儲存庫並安裝外掛程式：建置 Device Farm Gradle 外掛程式。

設定 AWS Device Farm Gradle 外掛程式

1. 將外掛程式成品中新增到 `build.gradle` 中的相依性清單。

```
buildscript {

    repositories {
        mavenLocal()
        mavenCentral()
    }

    dependencies {
        classpath 'com.android.tools.build:gradle:1.3.0'
```

```
            classpath 'com.amazonaws:aws-devicefarm-gradle-plugin:1.0'
        }
    }
```

2. 在 build.gradle 檔案中設定外掛程式。以下針對測試的組態應做為您的指南：

```
apply plugin: 'devicefarm'

devicefarm {

    // Required. The project must already exist. You can create a project in the
 AWS Device Farm console.
    projectName "My Project" // required: Must already exist.

    // Optional. Defaults to "Top Devices"
    // devicePool "My Device Pool Name"

    // Optional. Default is 150 minutes
    // executionTimeoutMinutes 150

    // Optional. Set to "off" if you want to disable device video recording during
 a run. Default is "on"
    // videoRecording "on"

    // Optional. Set to "off" if you want to disable device performance monitoring
 during a run. Default is "on"
    // performanceMonitoring "on"

    // Optional. Add this if you have a subscription and want to use your unmetered
 slots
    // useUnmeteredDevices()

    // Required. You must specify either accessKey and secretKey OR roleArn.
 roleArn takes precedence.
    authentication {
        accessKey "AKIAIOSFODNN7EXAMPLE"
        secretKey "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"

        // OR

        roleArn "arn:aws:iam::111122223333:role/DeviceFarmRole"
    }

    // Optionally, you can
```

```
    // - enable or disable Wi-Fi, Bluetooth, GPS, NFC radios
    // - set the GPS coordinates
    // - specify files and applications that must be on the device when your test
 runs
    devicestate {
        // Extra files to include on the device.
        // extraDataZipFile file("path/to/zip")

        // Other applications that must be installed in addition to yours.
        // auxiliaryApps files(file("path/to/app"), file("path/to/app2"))

        // By default, Wi-Fi, Bluetooth, GPS, and NFC are turned on.
        // wifi "off"
        // bluetooth "off"
        // gps "off"
        // nfc "off"

        // You can specify GPS location. By default, this location is 47.6204,
 -122.3491
        // latitude 44.97005
        // longitude -93.28872
    }

    // By default, the Instrumentation test is used.
    // If you want to use a different test type, configure it here.
    // You can set only one test type (for example, Calabash, Fuzz, and so on)

    // Fuzz
    // fuzz { }

    // Calabash
    // calabash { tests file("path-to-features.zip") }

}
```

3. 使用以下任務執行 Device Farm 測試：`gradle devicefarmUpload`。

   建置輸出會印出 Device Farm 主控台的連結，您可以在其中監控測試執行。

後續步驟：[在 Device Farm Gradle 外掛程式中產生 IAM 使用者]

步驟 3：在 Device Farm Gradle 外掛程式中產生 IAM 使用者

AWS Identity and Access Management (IAM) 可協助您管理使用 AWS 資源的許可和政策。本主題會逐步引導您產生具有存取 AWS Device Farm 資源許可的 IAM 使用者。

如果您尚未這麼做，請先完成步驟 1 和 2，再產生 IAM 使用者。

我們建議您不要使用 AWS 根帳戶來存取 Device Farm。請改為在帳戶中 AWS 建立新的 IAM 使用者（或使用現有的 IAM 使用者），然後使用該 IAM 使用者存取 Device Farm。

> **Note**
>
> 您用來完成下列步驟的 AWS 根帳戶或 IAM 使用者必須具有建立下列 IAM 政策並將其連接至 IAM 使用者的許可。如需詳細資訊，請參閱使用政策。

在 IAM 中建立具有適當存取政策的新使用者

1. 前往 https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。
2. 選擇 Users (使用者)。
3. 選擇 Create New Users (建立新的使用者)。
4. 輸入您選擇的使用者名稱。

   例如 **`GradleUser`**。
5. 選擇建立。
6. 選擇 Download Credentials (下載登入資料) 並將其儲存在您稍後可以輕鬆取得的位置。
7. 選擇 Close (關閉)。
8. 在清單中選擇使用者名稱。
9. 在 Permissions (許可) 下，按一下右側的向下箭頭展開 Inline Policies (內嵌政策) 標頭。
10. 選擇按一下它所說的這裡，沒有要顯示的內嵌政策。若要建立一個，請按一下此處。
11. 在 Set Permissions (設定許可) 畫面上，選擇 Custom Policy (自訂政策)。
12. 選擇選取。
13. 為您的政策命名，例如 **`AWSDeviceFarmGradlePolicy`**。
14. 將以下政策貼到 Policy Document (政策文件) 中。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DeviceFarmAll",
            "Effect": "Allow",
            "Action": [ "devicefarm:*" ],
            "Resource": [ "*" ]
        }
    ]
}
```

15. 選擇 Apply Policy (套用政策)

後續步驟：設定測試類型。

如需詳細資訊，請參閱建立 IAM 使用者 (AWS 管理主控台) 或 設定。

步驟 4：設定測試類型

根據預設，AWS Device Farm Gradle 外掛程式會執行適用於 Android 和 AWS Device Farm 的檢測測試。如果您想要執行自己的測試或指定其他參數，您可以選擇設定測試類型。本主題提供有關每個可用的測試類型的資訊，以及您必須在 Android Studio 中所進行的設定。如需 Device Farm 中可用測試類型的詳細資訊，請參閱 AWS Device Farm 中的測試架構和內建測試。

如果您尚未這麼做，請先完成步驟 1 – 3，再設定測試類型。

> **Note**
>
> 如果您使用的是裝置插槽，則裝置插槽功能預設為停用。

Appium

Device Farm 支援適用於 Android 的 Appium Java JUnit 和 TestNG。

- Appium （在 Java (JUnit) 下）

- Appium （在 Java (TestNG) 下）

您可以選擇 `useTestNG()` 或 `useJUnit()`。預設為 `JUnit` 且不需要明確指定。

```
    appium {
        tests file("path to zip file") // required
        useTestNG() // or useJUnit()
    }
```

內建：模糊

Device Farm 提供內建模糊測試類型，它會將使用者介面事件隨機傳送到裝置，然後報告結果。

```
    fuzz {

       eventThrottle 50 // optional default
       eventCount 6000  // optional default
       randomizerSeed 1234 // optional default blank

     }
```

如需詳細資訊，請參閱執行 Device Farm 的內建模糊測試 (Android 和 iOS)。

檢測

Device Farm 支援 Android 的檢測 (JUnit、Espresso、Robotium 或任何檢測型測試）。如需詳細資訊，請參閱適用於 Android 和 AWS Device Farm 的檢測。

在 Gradle 中執行檢測測試時，Device Farm 會使用從您的 androidTest 目錄產生的 `.apk` 檔案做為測試來源。

```
    instrumentation {

        filter "test filter per developer docs" // optional

    }
```

AWS Device Farm 文件歷史記錄

下表說明自上次發行本指南之後，文件內所進行的重要變更。

變更	描述	變更日期
Appium 端點支援	Device Farm 現在提供全受管 Appium 端點，用於遠端裝置測試，可實現快速測試開發和偵錯。這補充了現有的伺服器端執行方法，其中測試會上傳並直接在 Device Farm 上執行。雖然伺服器端執行非常適合 CI/CD 管道和大規模測試，但新的本機 Appium 端點允許在實際裝置上更快速地迭代和開發測試。	2025 年 11 月 17 日
iOS 測試主機改進	Device Farm 現在支援 iOS 測試環境的更新體驗，讓 Android 和 iOS 測試之間的設定保持一致。如需進一步了解，請參閱自訂測試環境的主機。 此外，已移除與淘汰 Android 測試主機相關的資訊。建議 Android 使用者使用 Amazon Linux 2 測試主機。	2025 年 10 月 31 日
AL2 支援	Device Farm 現在支援 Android 的 AL2 測試環境。進一步了解 AL2。	2023 年 11 月 6 日
從標準遷移到自訂測試環境	更新遷移指南，以記錄 2023 年 12 月標準模式測試的棄用。	2023 年 9 月 3 日
VPC ENI 支援	Device Farm 現在允許私有裝置使用 VPC-ENI 連線功能，協助客戶安全地連線至託管於 AWS、內部部署軟體或其他雲端供應商的私有端點。了解 VPC-ENI。	2023 年 5 月 15 日
microSDHC UI 更新	Device Farm 主控台現在支援架構架構。	2021 年 7 月 28 日
Python 3 支援	Device Farm 現在支援自訂模式測試中的 Python 3。了解關於在您的測試套件中使用 Python 3 的相關資訊： • Appium (Python) • Appium (Python)	2020 年 4 月 20 日

變更	描述	變更日期
新的安全性資訊和標記 AWS 資源的資訊。	為了更輕鬆且全面地保護 AWS 服務，已建立新的安全性章節。若要深入了解，請參閱中的安全性 AWS Device Farm 已新增在 Device Farm 中標記的新區段。如需進一步了解標記的資訊，請參閱 Device Farm中的標記。	2020 年 3 月 27 日
移除直接裝置存取。	直接裝置存取 (私有裝置上的遠端偵錯) 不再提供一般用途。如需查詢直接裝置存取的未來可用性，請聯絡我們。	2019 年 9 月 9 日
更新 Gradle 外掛程式組態	修訂版的 Gradle 外掛程式組態現在包含可自訂的 gradle 組態版本，且選用參數已變更為註解。進一步了解 設定 Device Farm Gradle 外掛程式。	2019 年 8 月 16 日
使用 XCTest 測試執行的新需求	對於使用 XCTest 架構的測試執行，Device Farm 現在需要專為測試而建置的應用程式套件。進一步了解 the section called “XCTest”。	2019 年 2 月 4 日
支援在自訂環境中的 Appium Node.js 和 Appium Ruby 測試類型	您現在可以在 Appium Node.js 和 Appium Ruby 自訂測試環境中執行測試。進一步了解 AWS Device Farm 中的測試架構和內建測試。	2019 年 1 月 10 日
在標準和自訂環境中支援 Appium 伺服器版本 1.7.2。版本 1.8.1 支援在自訂測試環境中使用自訂測試規格 YAML 檔案。	您現在可以在標準和自訂測試環境中使用 Appium 伺服器版本 1.72、1.71 和 1.6.5 執行測試。您也可以在自訂測試環境中使用版本 1.8.1 和 1.8.0 和自訂測試規格 YAML 檔案執行測試。進一步了解 AWS Device Farm 中的測試架構和內建測試。	2018 年 10 月 2 日
自訂測試環境	透過自訂測試環境，您可以確保測試如同在本機環境中一樣執行。Device Farm 現在支援即時日誌和影片串流，因此您可以立即取得在自訂測試環境中執行之測試的意見回饋。進一步了解 AWS Device Farm 中的自訂測試環境。	2018 年 8 月 16 日

變更	描述	變更日期
支援使用 Device Farm 做為 AWS CodePipeline 測試供應商	您現在可以在 中設定管道 AWS CodePipeline ，以在發行程序中使用 AWS Device Farm 執行做為測試動作。CodePipeline 可讓您快速連結儲存庫以建置和測試階段，以達成根據您的需求自訂的持續整合系統。進一步了解 在 CodePipeline 測試階段中整合 AWS Device Farm。	2018 年 7 月 19 日
支援私有裝置	您現在可以使用私有裝置來排程測試執行，並開始遠端存取工作階段。您可以管理這些裝置的設定檔和設定、建立 Amazon VPC 端點以測試私有應用程式，以及建立遠端偵錯工作階段。進一步了解 AWS Device Farm 中的私有裝置。	2018 年 5 月 2 日
支援 Appium 1.6.3	您現在可以設定 Appium 自訂測試的 Appium 版本。	2017 年 3 月 21 日
設定測試執行的執行逾時	您可以設定測試執行或專案中所有測試的執行逾時。進一步了解 在 AWS Device Farm 中設定測試執行的執行逾時。	2017 年 2 月 9 日
網路打造	您現在可以為測試執行模擬網路連線和狀況。進一步了解 模擬 AWS Device Farm 執行的網路連線和條件。	2016 年 12 月 8 日
新的故障排除章節	您現在可以使用一組程序對測試套件上傳進行故障診斷，這些程序旨在解決您在 Device Farm 主控台中可能遇到的錯誤訊息。進一步了解 故障診斷 Device Farm 錯誤。	2016 年 8 月 10 日
遠端存取工作階段	您現在可以遠端存取並與主控台中的單一裝置進行互動。進一步了解 遠端存取。	2016 年 4 月 19 日
裝置插槽自助服務	您現在可以使用 AWS 管理主控台、 AWS Command Line Interface或 API 購買裝置插槽。進一步了解如何 在 Device Farm 中購買裝置插槽。	2016 年 3 月 22 日
如何停止測試執行	您現在可以使用 AWS 管理主控台 AWS Command Line Interface、 或 API 停止測試執行。進一步了解如何 在 AWS Device Farm 中停止執行。	2016 年 3 月 22 日

變更	描述	變更日期
新的 XCTest UI 測試類型	您現在可以在 iOS 應用程式上執行 XCTest UI 自訂測試。進一步了解關於 將適用於 iOS 的 XCTest UI 與 Device Farm 整合 測試類型的資訊。	2016 年 3 月 8 日
新的 Appium Python 測試類型	您現在可以在 Android、iOS 和 web 應用程式上執行 Appium Python 自訂測試。進一步了解 AWS Device Farm 中的測試架構和內建測試。	2016 年 1 月 19 日
web 應用程式測試類型	您現在可以在 web 應用程式上執行 Appium Java JUnit 和 TestNG 自訂測試。進一步了解 AWS Device Farm 中的 Web 應用程式測試。	2015 年 11 月 19 日
AWS Device Farm Gradle 外掛程式	進一步了解如何安裝和使用 Device Farm Gradle 外掛程式。	2015 年 9 月 28 日
新的 Android 內建測試：Explorer	瀏覽器測試會以最終使用者的角度分析每個畫面，藉此測試爬取您的應用程式，並在探索時擷取螢幕擷取畫面。	2015 年 9 月 16 日
新增對 iOS 的支援	若要進一步了解測試 iOS 裝置和執行 iOS 測試 (包括 XCTest)，請參閱 AWS Device Farm 中的測試架構和內建測試。	2015 年 8 月 4 日
初始公有版本	這是 AWS Device Farm 開發人員指南的初始公開版本。	2015 年 7 月 13 日

AWS 詞彙表

如需最新的 AWS 術語，請參閱 AWS 詞彙表 參考中的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。