開發人員指南

AWS Device Farm

API 版本 2015-06-23

AWS Device Farm: 開發人員指南

Table of Contents

什麼是 AWS 裝置伺服器陣列？

設備農場是一種應用程序測試服務，可用於在由亞馬遜網絡服務託管的真實物理手機和平板電腦上測試 Android，iOS 和 Web 應用程序並與之交互（ AWS).

使用裝置伺服器陣列的主要方式有兩種：

- 使用各種測試架構自動化應用程式測試。
- 遠端存取您可以即時載入、執行並與應用程式互動的裝置。

> **Note**
>
> 裝置伺服器陣列僅適用於`us-west-2`(俄勒岡州) 地區。

自動化應用測試

裝置伺服器陣列可讓您上傳自己的測試，或使用內建的無指令碼相容性測試。由於系統會平行執行測試，所以會在幾分鐘內開始多個裝置上的測試。

測試完成後，測試報告包含高級別結果，低級日誌，pixel-to-pixel屏幕截圖和性能數據被更新。

設備農場支持本地和混合 Android 和 iOS 應用程序的測試，包括使用創建的應用程序PhoneGap，鈦，Xamarin，統一，和其他框架。其支援遠端存取 Android 和 iOS 應用程式以進行互動測試。如需支援測試類型的詳細資訊，請參閱 在 AWS Device Farm 中使用測試類型。

遠端存取互動

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。與裝置即時互動在許多情況下很實用。例如，客服代表可以指導客戶使用或設定他們的裝置。他們還可以指導客戶使用在特定裝置上執行的應用程式。您可以將應用程式安裝在執行於遠端存取工作階段的裝置上，然後重現客戶問題或回報錯誤。

在遠端存取工作階段期間，裝置伺服器陣列會收集您與裝置互動時所發生之動作的詳細資料。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。

術語

裝置伺服器陣列引入下列定義資訊組織方式的術語：

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

要求裝置伺服器陣列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 裝置農場定價。

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

包含有關運行的信息，這是設備場對一個或多個設備測試單個應用程序的請求。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

若要使用裝置陣列，請參閱設定。

設定 AWS Device Farm

第一次使用 Device Farm 列之前，您必須完成下列工作：

主題

步驟 1：註冊 AWS

註冊 Amazon Web Services（AWS）。

如果您沒有 AWS 帳戶，請完成以下步驟來建立一個。

若要註冊成為 AWS 帳戶

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

   部分註冊程序需接收來電，並在電話鍵盤輸入驗證碼。

   當您註冊時 AWS 帳戶，會建立AWS 帳戶根使用者一個。根使用者有權存取該帳戶中的所有 AWS 服務 和資源。安全性最佳做法是將管理存取權指派給使用者，並僅使用 root 使用者來執行需要 root 使用者存取權的工作。

步驟 2：在您的 AWS 帳戶中建立或使用 IAM 使用者

建議您不要使用 AWS root 帳戶來存取 Device Farm。請改為在您的 AWS 帳戶中建立 AWS Identity and Access Management (IAM) 使用者 (或使用現有使用者)，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 Management Console)。

步驟 3：授與 IAM 使用者存取 Device Farm 的權限

授與 IAM 使用者存取 Device Farm 的權限。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 政策的權限，並將其附加到 IAM 使用者。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。給它一個描述性的標題，如*DeviceFarm###*。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "devicefarm:*"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

   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 政策附加到新使用者。如需將 IAM 政策附加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。

附加政策可讓 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 動作和資源的相關資訊，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 了。請參閱開始使用裝置伺服器陣列。

開始使用裝置伺服器陣列

本逐步解說說明如何使用裝置伺服器陣列來測試原生 Android 或 iOS 應用程式。您可以使用裝置伺服器陣列主控台建立專案、上傳 .apk 或 .ipa 檔案、執行一組標準測試，然後檢視結果。

> **Note**
>
> 裝置伺服器陣列僅適用於us-west-2(俄勒岡州)AWS地區。

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。你需要一個AWS帳戶和AWS Identity and Access Management具有存取裝置伺服器陣列權限的 (IAM) 使用者。

- 若為 Android，您需要 .apk (Android 應用程式套件) 檔案。若為 iOS，您需要 .ipa (iOS 應用程式存檔) 檔案。您稍後在本逐步解說中將檔案上傳到裝置伺服器陣列。

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

- (選擇性) 您需要從裝置伺服器陣列支援的其中一個測試架構進行測試。您將此測試套件上傳至裝置伺服器陣列，然後在本逐步解說稍後執行測試。如果您沒有可用的測試包，則可以指定並運行標準的內置測試套件。如需詳細資訊，請參閱在 AWS Device Farm 中使用測試類型。

步驟 1：登入 主控台

您可以使用裝置伺服器陣列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。

- 登入裝置伺服器陣列主控台，位於：https://console.aws.amazon.com/devicefarm。

步驟 2：建立專案

若要在裝置伺服器陣列中測試應用程式，您必須先建立專案。

1. 在導覽窗格中，選擇行動裝置測試，然後選擇項目。
2. 下行動裝置測試專案，選擇新項目。
3. 下建立專案，輸入 a專案名稱（例如，**MyDemoProject**).
4. 選擇 建立 。

   控制台打開自動化測試新創建項目的頁面。

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 Automated tests (自動測試) 頁面上，選擇 Create a new run (建立新執行)。
2. 在「」選擇應用頁面，下方移動應用，選擇選擇檔案，然後從您的計算機中選擇安卓（.apk 文件）或 iOS（.ipa）文件。或者，將檔案從電腦拖放到主機中。
3. 輸入一個執行名稱，例如**my first test**。依預設，裝置伺服器陣列主控台會使用檔案名稱。
4. 選擇 下一步。
5. 在「」配置頁面，下方設置測試框架，選擇其中一個測試框架或內置測試套件。如需每個選項的詳細資訊，請參閱測試類型和框架。
   - 如果您尚未封裝裝置伺服器陣列的測試，請選擇內置：模糊運行標準的內置測試套件。您可以保留下列項目的預設值事件計數,事件油門，以及隨機種子。如需詳細資訊，請參閱the section called “內置：模糊（安卓和 iOS）”。
   - 如果您有來自其中一個受支持的測試框架的測試包，請選擇相應的測試框架，然後上傳包含測試的文件。
6. 選擇 下一步。

7. 在「」選擇裝置頁面，用於裝置集區，選擇頂級設備。
8. 選擇 下一步。
9. 在 Specify device state (指定裝置狀態) 頁面上，請執行下列動作：
   - 若要提供在執行期間使用的裝置伺服器陣列的其他資料，請在新增額外資料」中，上傳一個 .zip 檔案。
   - 要安裝用於運行的其他應用程序，請在安裝其他應用，上傳應用程式的 .apk 或 .ipa 檔案。若要變更安裝順序，請拖放檔案。
   - 要打開 Wi-Fi，藍牙，GPS 或 NFC 無線電進行運行，請在設定無線電狀態」中，選取對應的核取方塊。

     > **Note**
     >
     > 設置設備無線電狀態目前僅適用於 Android 本機測試。

   - 若要在執行期間測試特定於位置的行為，請在裝置位置，指定預置緯度和經度坐標。
   - 要為運行預設設備語言和區域，請在裝置地區設中，選擇語言環境。
   - 若要為執行預設網路設定檔，請在網路設定檔」中，選擇精選的個人檔案。或者，選擇建立網路設定檔創建自己的。
10. 選擇 下一步。
11. 在 Review and start run (檢閱並開始執行) 頁面上，選擇 Confirm and start run (確認並開始執行)。

裝置伺服器陣列會在裝置可用時立即開始執行，通常會在幾分鐘內完成。若要檢視執行狀態，請在自動化測試您的項目頁面，選擇運行的名稱。一個運行頁面，下设备，每個裝置都會以擱置圖示開頭 在設備表中，然後切換到正在運行的圖標 測試開始的時間。每次測試完成時，控制台都會在設備名稱旁邊顯示測試結果圖標。完成所有測試後，運行旁邊的待處理圖標將更改為測試結果圖標。

步驟 4：查看跑步結果

若要檢視執行中的測試結果，請在自動化測試您的項目頁面，選擇運行的名稱。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 中使用測試報告。

下一步驟

如需 Device Farm 的詳細資訊，請參閱 概念。

在裝置伺服器陣列中購買裝置插槽

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。

主題

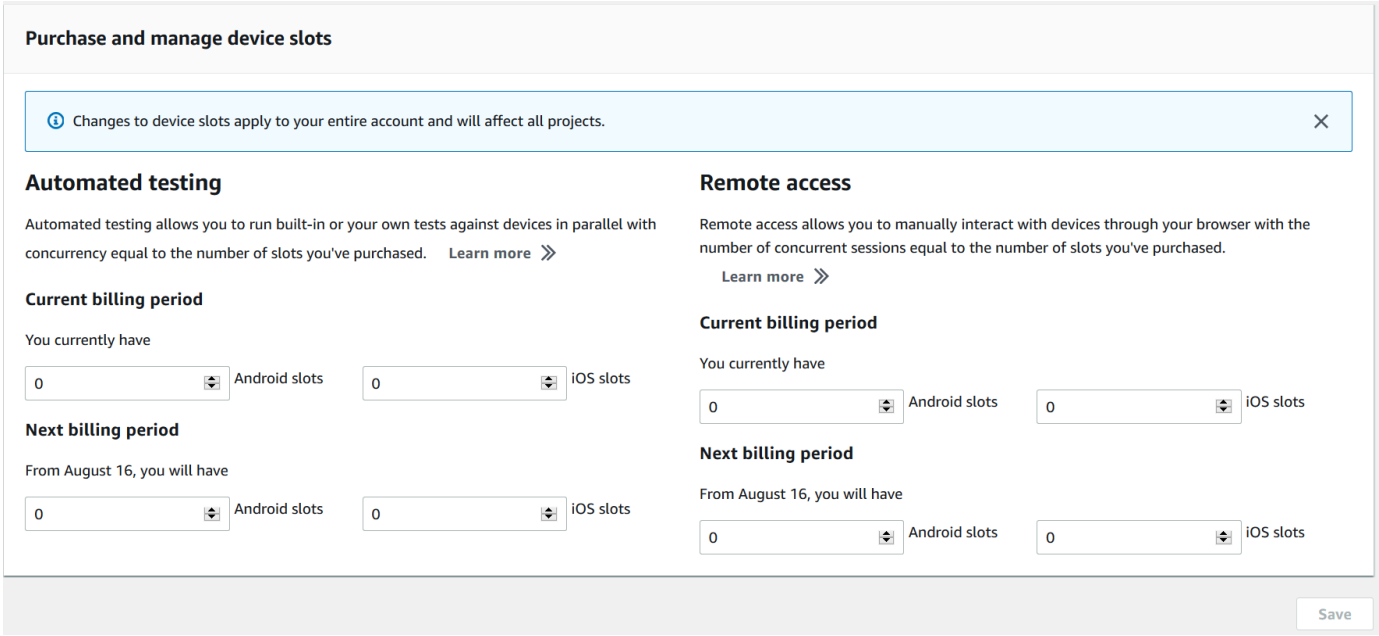
購買裝置插槽 (主機)

1. 登入 Device Farm 主控台，網址為 https://console.aws.amazon.com/devicefarm。
2. 在瀏覽窗格中，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，您可以選擇要購買的自動化測試和遠端存取裝置插槽數量，以建立自己的自訂套件。指定目前和下一個帳單週期的插槽金額。

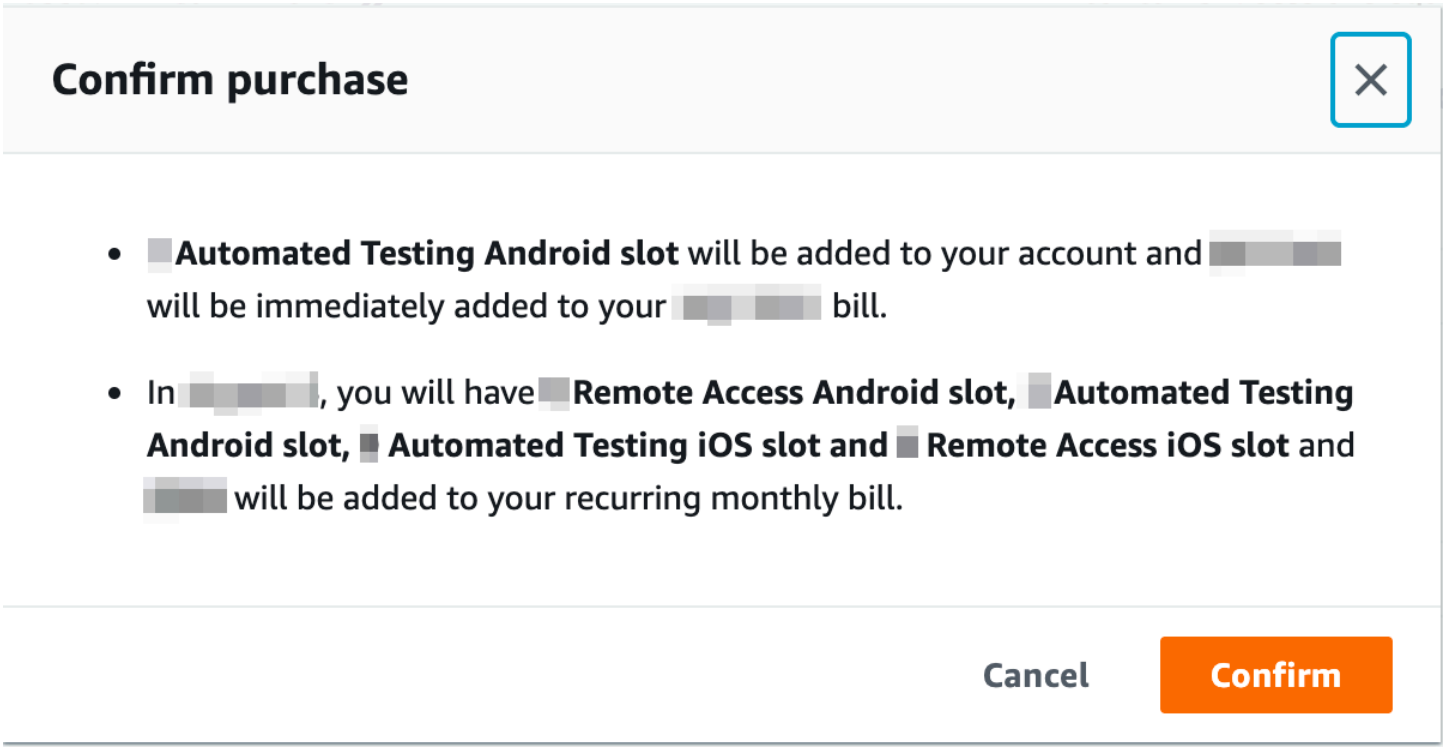
   當您變更插槽數量時，文字會隨帳單金額動態更新。如需詳細資訊，請參閱 AWS Device Farm 定價。

   > **Important**
   >
   > 如果您變更裝置插槽數量，但看到「與我們聯絡」或聯絡我們以購買訊息，表示您的 AWS 帳戶尚未獲准購買您要求的裝置插槽數量。
   > 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，指定您要購買的每種裝置類型的數量，以及計費週期的數量。

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



4. 選擇 Purchase (購買)。將出現「確認購買」窗口。複查資訊，然後選擇「確認」以完成異動。



在 [購買和管理裝置插槽] 頁面上，您可以看到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 Device Farm 列帳戶設定，包括可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"
    }
]
}
```

若要列出可用的提供項目促銷活動，請執行list-offering-promotions指令。

> **Note**
>
> 此命令只會傳回您尚未購買的促銷活動。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和指list-offering-transactions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「GetAccount設定」 操作以列出您的帳戶設定。

2. 呼叫ListOfferings作業以列出可供您使用的裝置插槽供應項目。
3. 呼叫「ListOffering促銷」作業以列出可用的提供項目促銷。

> **Note**
>
> 此命令只會傳回您尚未購買的促銷活動。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. 呼叫作PurchaseOffering業以購買提供項目。
5. 呼叫 GetOffering[狀態] 作業以取得提供項目狀態。

此功能也可以使用RenewOffering和「ListOffering交易」指令。

如需使用 Device Farm API 的相關資訊，請參閱自動化裝置農場。

取消裝置插槽 (主控台)

1. 登入 Device Farm 主控台，網址為 https://console.aws.amazon.com/devicefarm。
2. 在瀏覽窗格中，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，您可以減少 [下一個計費週期] 下方的值，以減少用於自動化測試和遠端存取的裝置插槽數量。在下一個帳單週期向您的帳戶收取的金額將列在帳單週期欄位下方。
4. 選擇儲存。此時會出現「確認變更」視窗。複查資訊，然後選擇「確認」以完成異動。

取消裝置插槽 (AWS CLI)

您可以執行renew-offering指令來變更下一個計費週期的裝置數量。

取消裝置插槽 (API)

調用RenewOffering操作以更改帳戶中設備的數量。

AWS Device Farm 概念

本節說明重要的 Device Farm 概念。

- AWS 裝置伺服器陣列的裝置支援
- 測試環境
- 執行
- 應用程式
- AWS Device Farm 中的報告
- 工作階段

如需裝置伺服器陣列中支援之測試類型的詳細資訊，請參閱在 AWS Device Farm 中使用測試類型。

AWS 裝置伺服器陣列的裝置支援

下列各節提供裝置 Device Farm 列中裝置支援的相關資訊。

主題

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

支援的裝置

Device Farm 提供數百種獨特、常用的 Android 和 iOS 裝置以及作業系統組合的支援。隨著新裝置在市場上推出，清單中的可用裝置也會增加。如果需要完整的裝置清單，請參閱裝置清單。

裝置集區

Device Farm 會將其裝置組織到可用於測試的裝置集區中。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，例如用於頂級裝置的集區。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。某些配置（例如根目錄的 Android 設備）可以作為私有設備進行支持。每個私有裝置都是裝置伺服器陣列代表您在 Amazon 資料中心部署的實體裝置。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 在 AWS 裝置伺服器陣列中使用私有裝置。

裝置品牌

Device Farm 會在各種 OEM 的實體行動裝置和平板電腦裝置上執行測試。

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。

預裝的裝置應用程式

裝 Device Farm 中的裝置包含少量製造商和電信業者已安裝的應用程式。

裝置功能

所有裝置皆可透過 Wi-Fi 與網際網路連線。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。

所有支援 Google Play 服務的裝置上都已安裝該服務，但這些裝置上沒有作用中的 Google 帳戶。

AWS Device Farm 中的測試環境

AWS Device Farm 提供自訂和標準測試環境來執行自動化測試。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，此環境會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

當您在標準環境中運行測試時，Device Farm 會為測試套件中的每個案例提供詳細的日誌和報告。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會比您在本機執行測試時長。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 應執行以執行測試的命令。這可確保 Device Farm 上的測試以類似於在本機電腦上執行測試的方式執行。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 使用自訂測試環境。

您可以選擇在使用 Device Farm 主控台時使用自訂測試環境 AWS CLI，或使用 Device Farm API 建立測試回合。

如需詳細資訊，請參閱使用 AWS CLI和上傳自訂測試規格在 Device Farm 中建立測試回合。

在 AWS Device Farm 中執行

下列各節包含在 Device Farm 列中執行的相關資訊。

在設 Device Farm 中運行代表應用程序的特定構建，具有一組特定的測試，將在一組特定的設備上運行。執行會產生包含執行結果相關資訊的報告。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。其中包括經緯度座標、地區設定、無線電狀態 (例如藍牙、GPS、NFC 和 Wi-Fi)、額外資料 (包含在 .zip 檔案中)，以及輔助應用程式 (在測試應用程式之前應該安裝的應用程式)。

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、記錄和螢幕擷取畫面 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，然後才能進行下一項工作。

平行運行

Device Farm 會在裝置可用時 parallel 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 中設定測試執行逾時。

放送中的廣告

建議您先從應用程式中移除廣告，然後再將廣告上傳到 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。

執行的一般工作

如需詳細資訊，請參閱 在 Device Farm 中建立測試回合 及 在 AWS Device Farm 中使用測試回合。

AWS Device Farm 中的應用程式

下列各節包含裝置伺服陣列中應用程式行為的相關資訊。

主題

- 檢測應用程式
- 執行中重新簽署應用程式
- 運行中混淆應用程序

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 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。

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 UUID 新增至佈建設定檔。Device Farm 以萬用字元設定檔取代內嵌的佈建設定檔，然後重新簽署應用程式。如果您提供輔助數據，則 Device Farm 會在設備農場安裝之前將其添加到應用程序的包中，以便輔助數據存在於應用程序的沙箱中。重新簽署應用程式會移除應用程式群組、關聯網域、遊戲中心、、無線配件設定 HealthKit HomeKit、應用程式內購買、應用程式間音訊、Apple Pay、推送通知和 VPN 設定與控制等權利。

對於 Android 應用程式，Device Farm 會重新簽署應用程式。這可能會破壞取決於應用程序簽名的任何功能，例如 Google Maps Android API，或者可能會觸發來自產品的反盜版或防篡改檢測。 DexGuard

運行中混淆應用程序

對於 Android 應用程序，如果應用程序被混淆，則仍然可以使用 Device Farm 進行測試（如果您使用）。 ProGuard不過，如果您 DexGuard 搭配反盜版措施使用，Device Farm 就無法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 列測試報告的相關資訊。

主題

- 報告保留
- 報表元件
- 登入報告
- 報表的一般工作

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。

報表元件

Device Farm 中的報告包含通過和失敗資訊、當機報告、測試和裝置記錄、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。

登入報告

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。

報表的一般工作

如需更多詳細資訊，請參閱 在 Device Farm 中使用測試報告。

AWS 裝置農場中的工作階段

您可以使用裝置伺服器陣列，透過網頁瀏覽器中的遠端存取工作階段，對 Android 和 iOS 應用程式執行互動式測試。這種互動式測試可協助支援工程師在客戶電話上逐步解說客戶的問題。開發人員可以在特定裝置重現問題，以隔離可能的問題來源。您可以使用遠端工作階段，與您的目標客戶進行可用性測試。

主題

- 支援遠端存取的裝置
- 保留工作階段檔
- 檢測應用程式
- 在工作階段中重新簽署應
- 工作階段中的模糊應用程式

支援遠端存取的裝置

裝置伺服器陣列提供了許多獨特且受歡迎的 Android 和 iOS 裝置的支援。隨著新裝置在市場上推出，清單中的可用裝置也會增加。裝置伺服器陣列主控台會顯示目前可用於遠端存取的 Android 和 iOS 裝置清單。如需詳細資訊，請參閱AWS 裝置伺服器陣列的裝置支援。

保留工作階段檔

裝置伺服器陣列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。

裝置伺服器陣列會儲存您的工作階段記錄和擷取的視訊 400 天，然後將其從系統中刪除。

檢測應用程式

您不需要檢測應用程式，也不需要為裝置伺服器陣列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

裝置農場會重新簽署安卓和 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。應用程序重新簽名還可以從諸如以下產品觸發反盜版或防篡改檢測DexGuard適用於安卓設備。

工作階段中的模糊應用程式

對於 Android 應用程序，如果應用程序被混淆，則仍然可以使用設備農場進行測試（如果您使用）ProGuard。但是，如果您使用DexGuard採用反盜版措施，裝置伺服器陣列無法重新簽署應用程式。

使用 AWS 裝置伺服器陣列中的專案

Device Farm 中的專案代表裝置伺服器陣列中的邏輯工作區，其中包含執行，針對一或多個裝置對單一應用程式的每個測試執行一次執行。專案可讓您以選擇的任何方式組織工作區。例如，每個應用程序標題可以有一個項目，或者每個平台可以有一個項目。您可以視需要建立任意數量的專案。

您可以使用 AWS 裝置農場主控台，AWS Command Line Interface(AWS CLI)，或使用 AWS 裝置農場 API 來處理專案。

主題

在 AWS 裝置伺服器陣列中建立專案

您可以使用 AWS 裝置農場主控台建立專案，AWS CLI或 AWS 裝置伺服器陣列 API。

先決條件

- 完成「設定」中的步驟。

建立專案 (主控台)

1. 登入裝置伺服器陣列主控台，位於：https://console.aws.amazon.com/devicefarm。
2. 在 [裝置伺服陣列] 導覽面板上，選擇行動裝置測試，然後選擇項目。
3. 選擇 New project (新專案)。
4. 輸入專案名稱，然後選擇提交。
5. 若要指定專案的設定，請選擇 Project settings (專案設定)。這些設定包括測試執行的預設逾時時間。套用設定後，系統會將這些設定用於專案的所有測試執行。如需詳細資訊，請參閱在 AWS Device Farm 中設定測試執行逾時。

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 回應會包含專案的 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。

創建一個項目（API）

- 呼叫 `CreateProject` API。

如需使用裝置伺服器陣列 API 的相關資訊，請參閱自動化裝置農場。

檢視 AWS 裝置伺服器陣列中的專案清單

您可以使用 AWS 裝置農場主控台，AWS CLI，或使用 AWS 裝置伺服器陣列 API 來檢視專案清單。

主題

- 先決條件
- 檢視專案清單 (主控台)
- 檢視專案清單 (AWS CLI)
- 檢視專案清單 (API)

先決條件

- 在裝置伺服器陣列中建立至少一個專案。請遵循在 AWS 裝置伺服器陣列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. 登入裝置伺服器陣列主控台，位於：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，請在裝置伺服器陣列導覽功能表上選擇行動裝置測試，然後選擇項目。
   - 對於桌面瀏覽器測試專案，請在裝置伺服器陣列導覽功能表上選擇桌面瀏覽器測試，然後選擇項目。

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。

如需使用裝置伺服器陣列搭配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 裝置伺服器陣列 API 的相關資訊，請參閱自動化裝置農場。

在 AWS Device Farm 中使用測試回合

在設 Device Farm 中運行代表應用程序的特定構建，具有一組特定的測試，將在特定的設備上運行。執行會產生包含執行結果相關資訊的報告。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 來處理執行。

主題

在 Device Farm 中建立測試回合

您可以使用 Device Farm 主控 AWS CLI台或 Device Farm API 建立測試回合。您還可以使用支持的插件，如詹金斯或搖籃插件設 Device Farm。如需外掛程式的詳細資訊，請參閱 工具和插件。如需執行的詳細資訊，請參閱執行。

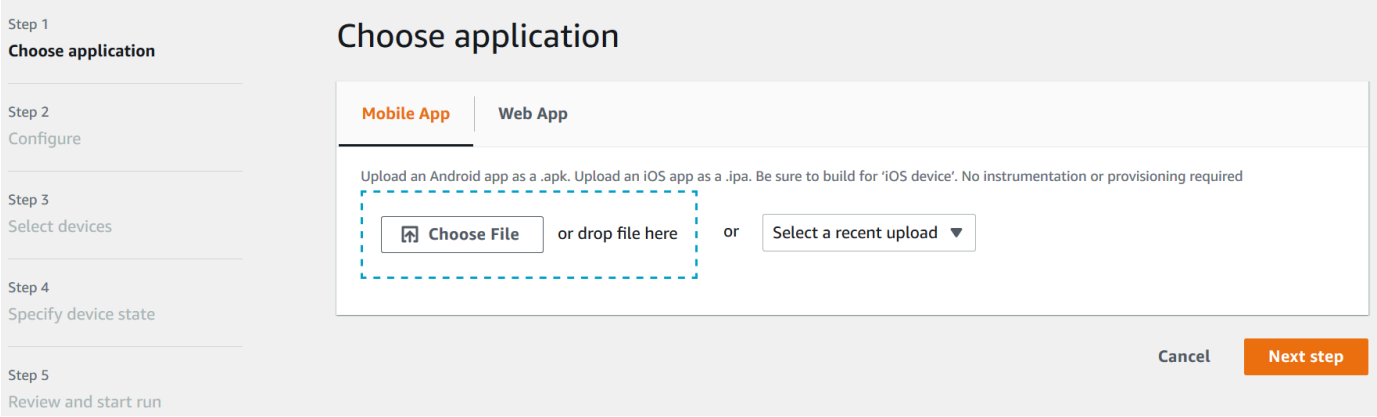
主題

必要條件

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

創建測試運行（控制台）

1. 登入 Device Farm 主控台，網址為 https://console.aws.amazon.com/devicefarm。
2. 在瀏覽窗格中，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 Create a new run (建立新執行)。
5. 在 [選擇應用程式] 頁面上，選擇 [行動應用程式] 或 [Web 應



6. 上傳您的應用程式檔案。您也可以拖放檔案或選擇最近上傳的項目。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。
7. (選用) 在 Run name (執行名稱) 中，輸入名稱。依預設，Device Farm 會使用應用程式檔案名稱。
8. 選擇下一步。
9. 在 Configure (設定) 頁面上，選擇其中一個可用的測試套件。

> **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，請參閱在 AWS Device Farm 中使用測試類型。

10. 如果您未選擇「內建:Fuzz」，請選取「選擇檔案」，然後瀏覽並選擇包含測試的檔案。
11. 對於您的測試環境，請選擇在我們的標準環境中運行測試或在自定義環境中運行測試。如需詳細資訊，請參閱 測試環境。

12. 如果您使用的是標準測試環境，請跳到步驟 13。如果您使用的是自訂測試環境，搭配預設測試規格 YAML 檔案，請跳到步驟 13。

    a. 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。

    b. 如果您變更了測試規格，請選擇「另存為新」來更新測試規格。

13. 如果您想要設定視訊錄製或效能資料擷取選項，請選擇 Advanced Configuration (進階組態)。

    a. 選擇啟用視頻錄製以在測試期間錄製視頻。

    b. 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**
    >
    > 如果您有私人裝置，也會顯示私人裝置的特定組態。

14. 選擇下一步。

15. 在 Select devices (選取裝置) 頁面上，執行以下其中一項操作：

    - 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。

    如需詳細資訊，請參閱 AWS 裝置伺服器陣列的裝置支援。

16. 選擇下一步。

17. 在 Specify device state (指定裝置狀態) 頁面上：

    - 若要提供其他資料供執行期間使用的 Device Farm，請選擇 [新增額外資料] 旁邊的 [選擇檔案]，然後瀏覽並選擇包含資料的 .zip 檔案。
    - 若要安裝其他應用程式以供執行期間使用的 Device Farm，請選擇 [安裝其他應用程式] 旁邊的 [選擇檔案]，然後瀏覽並選擇包含該應用程式的 .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。

18. 選擇下一步。
19. 在 [檢閱並開始執行] 頁面上，您可以指定測試回合的執行逾時。如果您是使用無限制的測試插槽，請確認已選取 Run on unmetered slots (在無限制插槽上執行)。
20. 輸入數值或使用捲軸來變更執行逾時。如需詳細資訊，請參閱 在 AWS Device Farm 中設定測試執行逾時。
21. 選擇 Confirm and start run (確認並開始執行)。

Device Farm 會在裝置可用時立即開始執行，通常會在幾分鐘內完成。在測試執行期間，Device Farm 主控台會在執行表

格 中

顯示擱置中的圖示。運行中的每個設備也將以待處理圖標啟動，然後在測試開

始 時

切換到正在運行的圖標。每次測試完成時，裝置名稱旁會顯示測試結果圖示。完成所有測試後，運行旁邊的待處理圖標將更改為測試結果圖標。

如果您想要停止測試回合，請參閱停止在 AWS Device Farm 中執行。

創建一個測試運行（AWS CLI）

您可以使用 AWS CLI 建立測試回合。

主題

步驟 1：選擇專案

您必須將測試回合與 Device Farm 專案相關聯。

1. 若要列出 Device Farm 列專案，請執行list-projects。如果您沒有專案，請參閱在 AWS 裝置伺服器陣列中建立專案。

   範例：

   ```
   aws devicefarm list-projects
   ```

   回應包含 Device Farm 列專案的清單。

   ```
   {
       "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
   ```

   回應包括內建的裝置伺服器陣列裝置集區，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：上傳您的申請文件

若要建立上傳請求並取得 Amazon Simple Storage Service (Amazon S3) 預先簽署的上傳 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 預先簽署的 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 簡單儲存服務使用者指南中的使用預先簽署的 URL 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. 若要建立上傳請求並取得 Amazon S3 預先簽署的上傳 URL，請create-upload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 預先簽署的 URL 上傳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：（可選）上傳您的自定義測試規格

如果您是在標準測試環境中執行測試，則可跳過此步驟。

設 Device Farm 為每個支持的測試類型維護一個默認的測試規格文件。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 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 預先簽署的 URL 上傳您的測試規格檔案。這個例子用curl來上傳一個 Appium JavaTest NG 測試規範：

```
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：安排測試運行

要使用 AWS CLI，請運行，指定以下內容來安排測試運行schedule-run：

- 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 的相關資訊 AWS CLI，請參閱AWS CLI 參考。

建立測試回合 (API)

這些步驟與本 AWS CLI 節中描述的步驟相同。請參閱創建一個測試運行（AWS CLI）。

您需要此資訊，才能呼叫 ScheduleRun API：

- 專案 ARN。請參閱 創建一個項目（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：（可選）上傳您的自定義測試規格 及 CreateUpload。

如需使用 Device Farm API 的相關資訊，請參閱自動化裝置農場。

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 中使用測試報告中的指示。

在 AWS Device Farm 中設定測試執行逾時

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 AWS CLI台或 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。

必要條件

1. 完成「設定」中的步驟。
2. 在 Device Farm 中建立專案。請遵循在 AWS 裝置伺服器陣列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://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. 如果您已經有一個項目，請從列表中選擇它。否則，請選擇「新建項目」，輸入項目的名稱，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。

設置測試運行的執行超時

1. 登入 Device Farm 主控台，網址為 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. 如果您已經有一個項目，請從列表中選擇它。否則，請選擇「新建項目」，輸入項目的名稱，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. 在檢閱並開始執行時，對於設定執行逾時，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

在 Device Farm 中測試 Android、iOS、FireOS 和 Web 應用程式時，您可以使用網路塑型模擬網路連線和狀況。例如，您可以在不是完美的網路狀況下測試您的應用程式。

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。使用網路控管時，您可以變更 Wi-Fi 連線以指定網路設定檔 (例如 3G 或 Lossy)， WiFi以控制輸入和輸出流量的輸送量、延遲、抖動和遺失。

主題

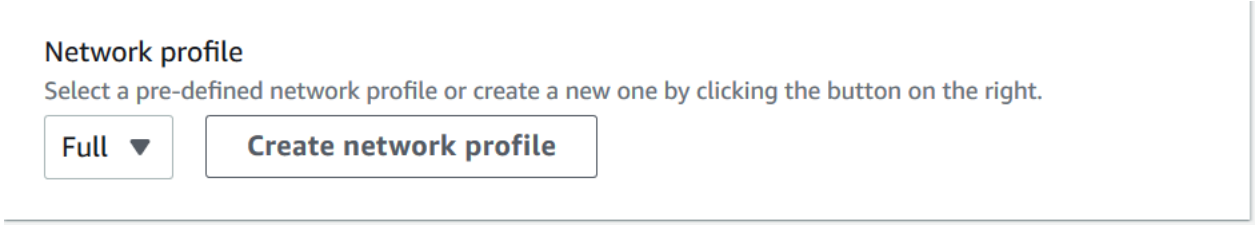
在排程測試執行時設定網路塑型

當您安排跑步時，您可以從任何裝置農場策劃的設定檔中進行選擇，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 裝置伺服器陣列中建立專案。

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. 請選取您的裝置，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」來建立您自己的設定檔。

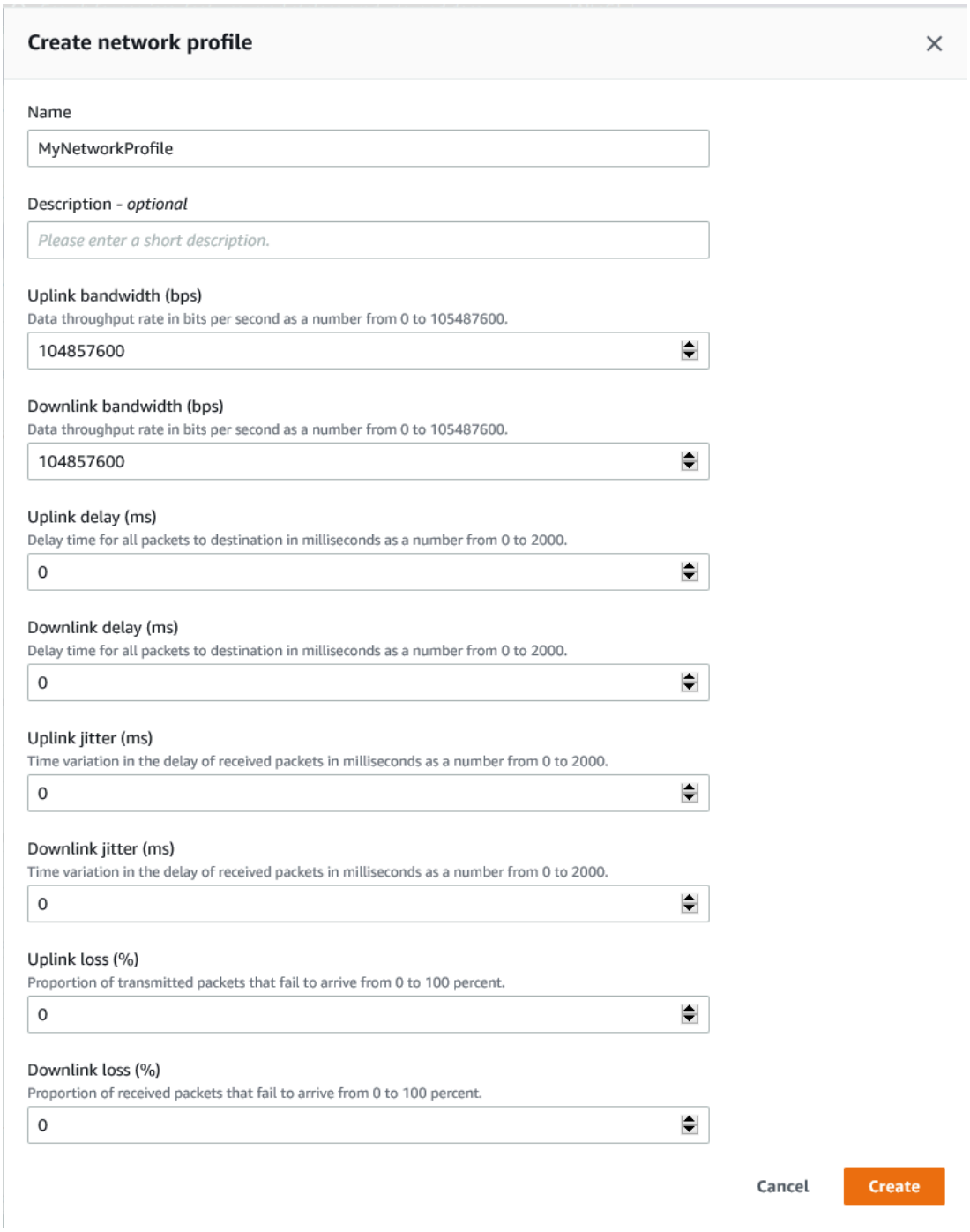


6. 選擇下一步。
7. 檢閱並開始您的執行測試。

建立網路設定檔

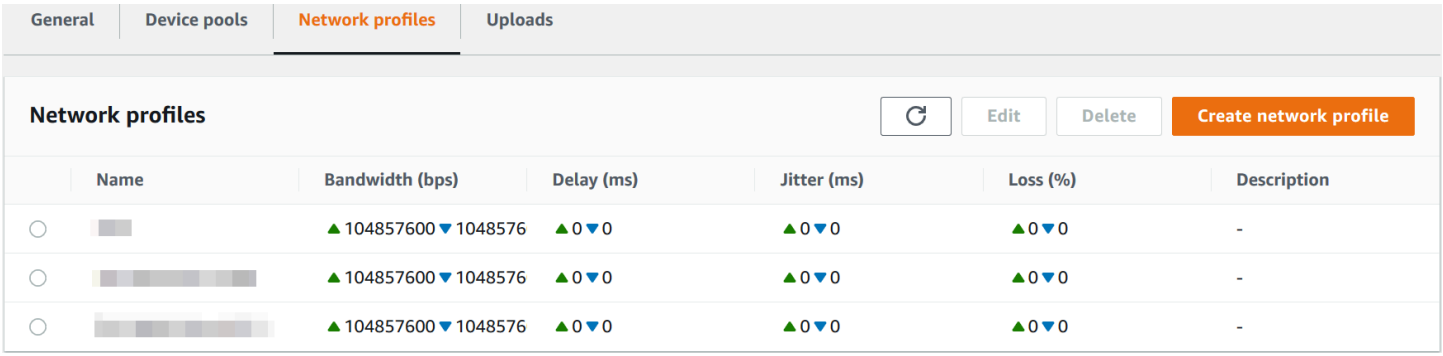
建立測試執行時，您可以建立網路設定檔。

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狀況

您可以使用 Appium 之類的框架從設備主機調用 API，以模擬動態網絡條件，例如在測試運行期間減少帶寬。如需詳細資訊，請參閱CreateNetwork設定檔。

停止在 AWS Device Farm 中執行

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 AWS CLI台或 API 來停止執行。

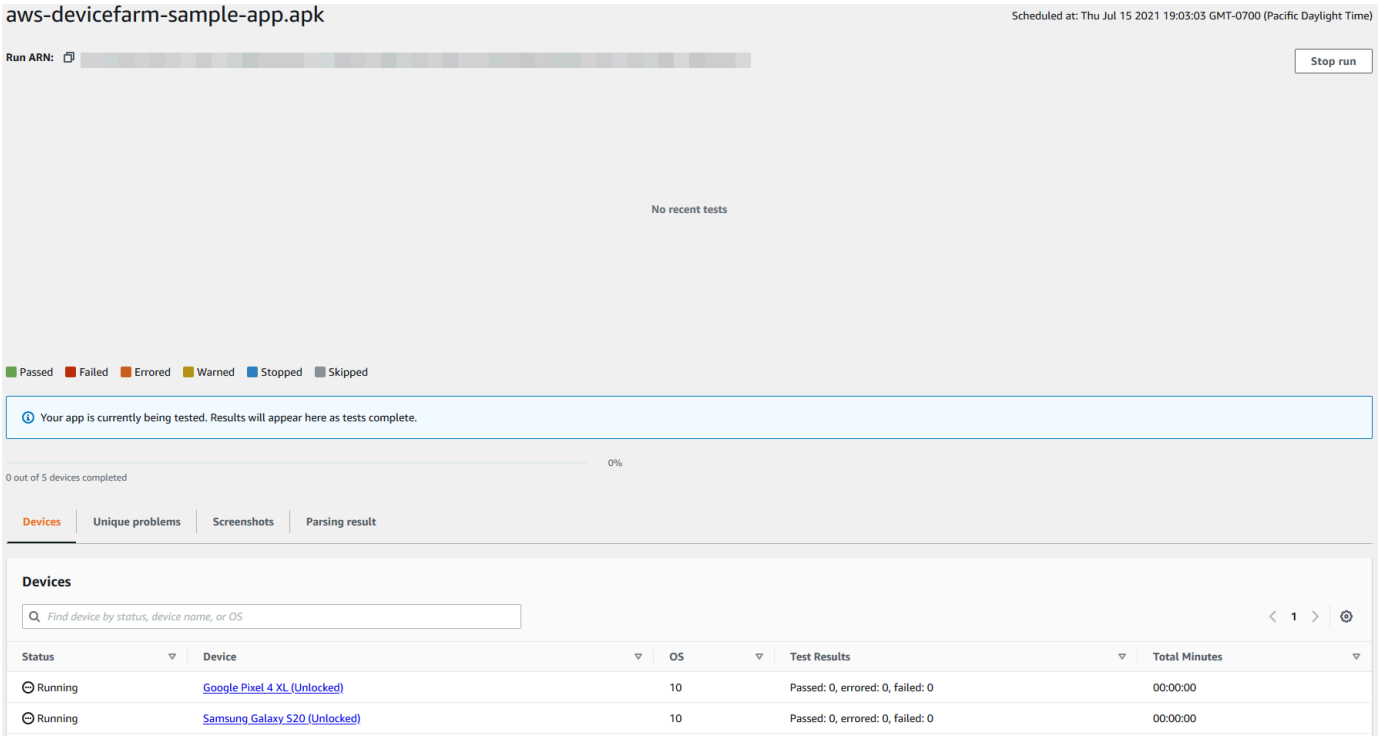
主題

- 停止運行（控制台）
- 停止運行（AWS CLI）
- 停止執行 (API)

停止運行（控制台）

1. 登入 Device Farm 主控台，網址為 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. 選擇您有一個活動測試運行的項目。
4. 在「自動測試」頁面上，選擇測試運行。

   待處理或正在運行的圖標應出現在設備名稱的左側。

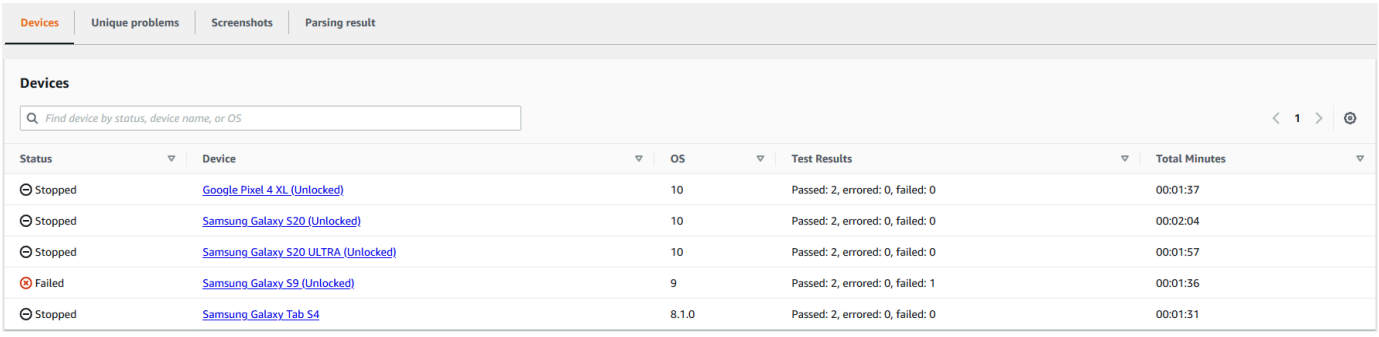


5. 選擇 Stop run (停止執行)。

短時間後，裝置名稱旁邊會出現一個帶有減號的紅色圓圈的圖示。當運行停止時，圖標顏色從紅色變為黑色。

> **Important**
>
> 如果測試已經執行，Device Farm 將無法停止測試。如果測試正在進行中，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，您還需要支付 Device Farm 執行安裝套件和拆卸套件所花費的總分鐘數計費。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。



停止運行（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 的相關資訊 AWS CLI，請參閱AWS CLI 參考。

停止執行 (API)

- 將StopRun操作調用到測試運行。

如需使用 Device Farm API 的相關資訊，請參閱自動化裝置農場。

檢視 AWS Device Farm 列中的執行清單

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

主題

- 檢視執行清單 (主控台)
- 檢視執行清單 (AWS CLI)
- 檢視執行清單 (API)

檢視執行清單 (主控台)

1. 登入 Device Farm 主控台，網址為 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**
> 您可以使用搜索欄按名稱過濾項目列表。

檢視執行清單 (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 的相關資訊 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，請參閱自動化裝置農場。

在 AWS 裝置伺服器陣列中建立裝置集區

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

主題

必要條件

- 在 Device Farm 主控台中建立執行。請遵循中的說明進行在 Device Farm 中建立測試回合 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 [選取裝置] 頁面上，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. 對於「欄位」(第一個下拉式清單)，請選擇下列其中一項：

      - 若要依製造商名稱加入裝置，請選擇「裝置製造商」。
      - 若要依類型值包含裝置，請選擇「外形規格」。

   d. 對於「運算子」(第二個下拉式清單)，選擇「等於」以包含「欄位」值等於「值」值的裝置。

   e. 對於「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。如果您選擇「欄位平台」，則唯一可用的選項是 Android 和 IOS。同樣地，如果您為欄位選擇「外形規格」，則只有「手機」和「平板電腦」可供選擇。

   f. 若要新增其他規則，請選擇 [新增規則]。

   g. 若要刪除規則，請選擇規則旁邊的 X 圖示。

   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。

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邊的核取方塊。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。

> **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)

- 執行 create-device-pool 命令。

如需搭配使用 Device Farm 的相關資訊 AWS CLI，請參閱AWS CLI 參考。

建立裝置集區 (API)

- 呼叫 `CreateDevicePool` API。

如需使用 Device Farm API 的相關資訊，請參閱自動化裝置農場。

分析 AWS Device Farm 中的結果

在標準測試環境中，您可以使用 Device Farm 控制台來查看測試運行中每個測試的報告。

Device Farm 還會收集其他成品，例如文件，日誌和圖像，您可以在測試運行完成時下載這些成品。

主題

- 在 Device Farm 中使用測試報告
- 在裝置伺服器陣列中使用成品

在 Device Farm 中使用測試報告

使用 [Device Farm] 主控台來檢視測試報告。如需詳細資訊，請參閱 AWS Device Farm 中的報告。

主題

必要條件

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 中建立測試回合，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  針對進行中的執行顯示擱置圖示。運行中的每個設備也將以待處理圖標啟動，然後在測試開始時切換到正在運行的  圖標。每次測試完成時，裝置名稱旁會顯示測試結果圖示。完成所有測試後，運行旁邊的待處理圖標將更改為測試結果圖標。如需詳細資訊，請參閱 了解測試結果。

了解測試結果

Device Farm 主控台會顯示圖示，協助您快速評估已完成測試回合的狀態。

主題

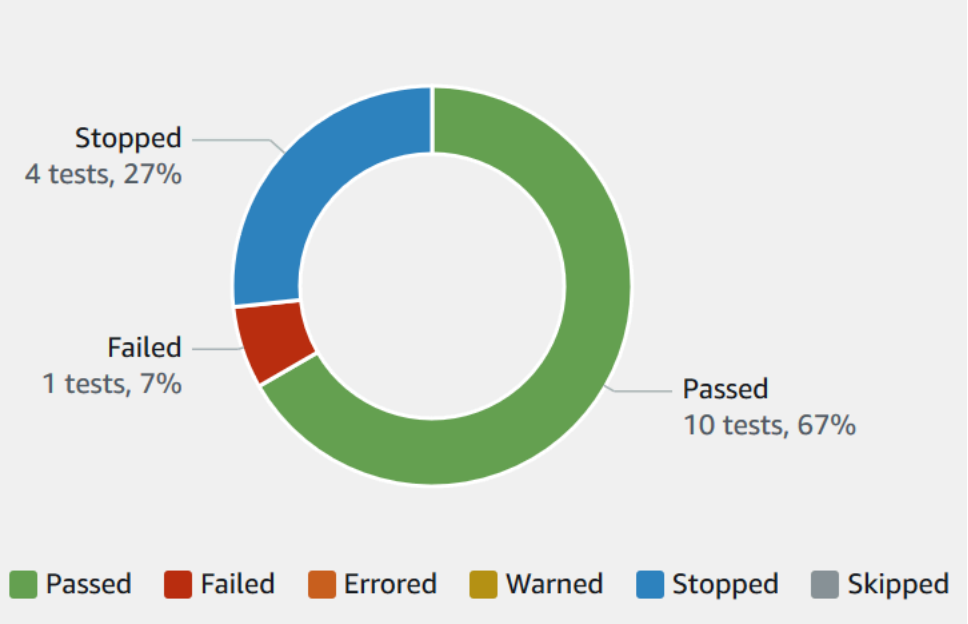
報告個別測試的結果

對於描述個別測試的報告，裝置伺服器陣列會顯示一個圖示：

描述	圖示
測試成功。	
測試失敗。	
Device Farm 已跳過測試。	
測試已停止。	
Device Farm 傳回警告。	
Device Farm 傳回錯誤。	

報告多個測試的結果

如果您選擇已完成的執行，裝置伺服器陣列會顯示測試結果摘要圖表。



例如，此測試運行結果圖表顯示運行有 4 個停止的測試，1 個失敗的測試和 10 個成功的測試。

圖表始終以顏色編碼和標記。

檢視報告

您可以在 [Device Farm] 主控台中檢視測試結果。

主題

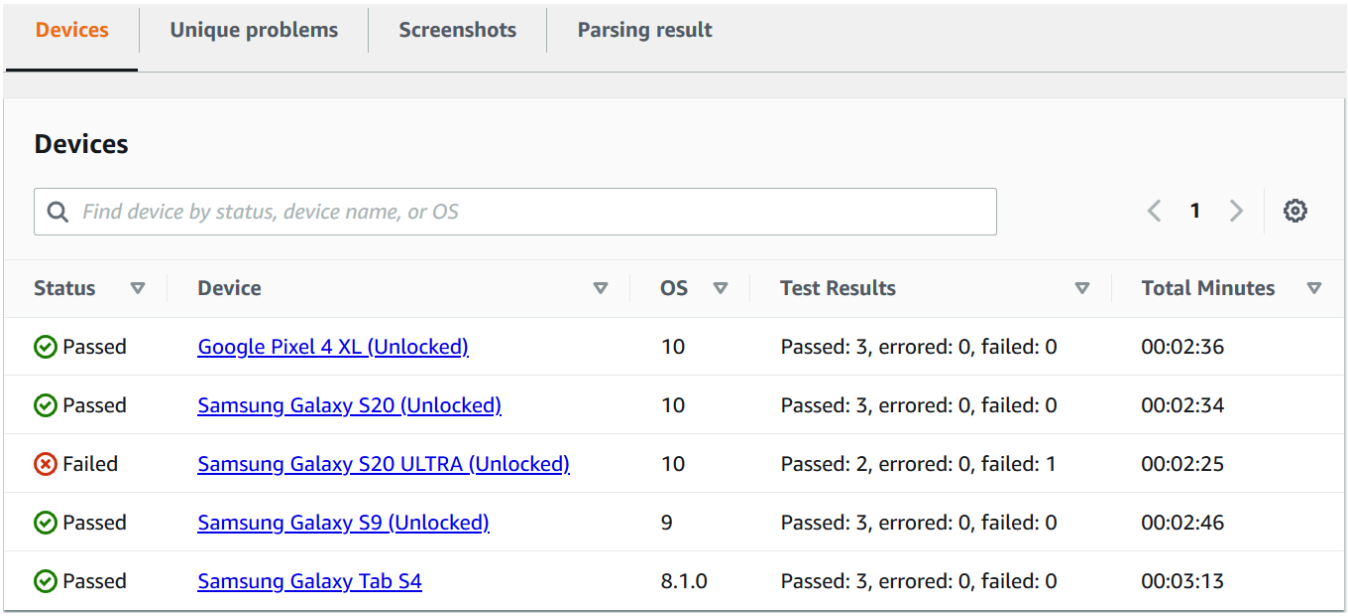
檢視測試回合摘要頁面

1. 登入 Device Farm 主控台，網址為 https://console.aws.amazon.com/devicefarm。
2. 在瀏覽窗格中，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。



在這個例子中，有幾個設備。在第一個表條目，在谷歌像素 4 XL 設備運行 Android 版本 10 報告了三個成功的測試，花了 02:36 分鐘來運行.

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在執行期間擷取的任何螢幕擷取畫面清單，依裝置分組。
- 在解析結果部分，您可以下載解析結果。

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」段落會顯示測試結果。狀態會以結果圖示表示。如需詳細資訊，請參閱 報告個別測試的結果。

   [記錄] 區段會顯示測試期間 Device Farm 所記錄的任何資訊。若要檢視此資訊，請按照在報告中檢視問題、裝置、套件或測試的記錄資訊中的指示。

   [效能] 索引標籤會顯示測試期間裝置伺服器陣列產生之任何效能資料的相關資 若要檢視此效能資料，請按照在報告中檢視問題、裝置、套件或測試的效能資料中的指示。

「檔案」標籤會顯示您可以下載的任何測試相關檔案 (例如記錄檔) 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] 索引標籤會顯示測試期間擷取 Device Farm 列的任何螢幕擷取

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。

  「組件」 段落顯示一個表格，其中包含裝置之組件的相關資訊。

  在此表格中，「測試結果」資料欄會依照在裝置上執行之每個測試套件的結果總結測試數目。該數據還具有圖形組件。如需詳細資訊，請參閱 報告多個測試的結果。

  若要依照套件檢視完整結果，請遵循中的指示檢視測試套件報告。

  [記錄] 區段會顯示裝置伺服器陣列在執行期間為裝置記錄的任何資訊。若要檢視此資訊，請按照在報告中檢視問題、裝置、套件或測試的記錄資訊中的指示。

  效能段落會顯示在執行期間為裝置產生之「Device Farm」之任何效能資料的相關資訊。若要檢視此效能資料，請按照在報告中檢視問題、裝置、套件或測試的效能資料中的指示。

  「檔案」段落會顯示裝置的組件清單，以及您可以下載的任何關聯檔案 (例如日誌檔)。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，從表格中選擇組件。

   Video (視訊) 區段顯示測試的可下載影片錄製。

   「測試」段落顯示一個表格，其中包含組件中測試的相關資訊。

   在表格中，「測試結果」欄會顯示結果。該數據還具有圖形組件。如需詳細資訊，請參閱 報告多個測試的結果。

若要依測試檢視完整結果，請遵循中的指示檢視測試報告。

「記錄」區段會顯示在執行套件期間「Device Farm」所記錄的任何資訊。若要檢視此資訊，請按照在報告中檢視問題、裝置、套件或測試的記錄資訊中的指示。

「效能」段落顯示在執行組件期間「Device Farm」所產生之任何效能資料的相關資訊。若要檢視此效能資料，請按照在報告中檢視問題、裝置、套件或測試的效能資料中的指示。

「檔案」段落會顯示組件的測試清單，以及您可以下載的任何相關檔案 (例如日誌檔)。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套件執行期間 Device Farm 列擷取的任何螢幕擷取畫面清單，並依測試分組。

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」段落會顯示測試結果。狀態會以結果圖示表示。如需詳細資訊，請參閱 報告個別測試的結果。

[記錄] 區段會顯示測試期間 Device Farm 所記錄的任何資訊。若要檢視此資訊，請按照在報告中檢視問題、裝置、套件或測試的記錄資訊中的指示。

[效能] 索引標籤會顯示測試期間裝置伺服器陣列產生之任何效能資料的相關資 若要檢視此效能資料，請按照在報告中檢視問題、裝置、套件或測試的效能資料中的指示。

「檔案」標籤會顯示您可以下載的任何測試相關檔案 (例如記錄檔) 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] 索引標籤會顯示測試期間擷取 Device Farm 列的任何螢幕擷取

在報告中檢視問題、裝置、套件或測試的效能資料

> **Note**
>
> Device Farm 目前僅會收集 Android 裝置的裝置效能資料。

「效能」標籤會顯示下列資訊：

- CPU 圖表會顯示應用程式在選取的問題、裝置、套件或測試 (沿垂直軸) 期間 (沿著垂直軸) 在單一核心上使用的 CPU 百分比 (沿著水平軸)。

  垂直軸是以百分比表示，從 0% 到最大的記錄百分比。

  如果應用程式已使用多個核心，則此百分比可能超過 100%。例如，如果三個核心各為 60% 用量，則此百分比會顯示為 180%。
- 記憶體圖表會顯示應用程式在選取的問題、裝置、套件或測試 (沿垂直軸) 一段時間 (沿著水平軸) 期間使用的 MB 數。

  垂直軸是以 MB 表示，從 0 MB 到所記錄之 MB 的數目上限。
- Threads (執行緒) 圖形顯示在選取的問題、裝置、套件或測試期間 (沿著垂直軸) 隨著時間 (沿著水平軸) 使用的執行緒數目。

  垂直軸以螺紋數表示，從零執行緒到記錄的最大執行緒數目。

在所有情況下，對於選取的問題、裝置、套件或測試，從開始執行到結束執行，水平軸都會以秒為單位表示。

若要顯示特定資料點的資訊，請沿著水平軸暫停在所需之秒上的所需圖形中。

在報告中檢視問題、裝置、套件或測試的記錄資訊

「記錄檔」段落會顯示下列資訊：

- Source (來源) 代表日誌項目的來源。可能的值包括：
  - 線束代表 Device Farm 所建立的記錄項目。這些日誌項目通常是在啟動和停止事件期間建立的。
  - 裝置代表裝置所建立的記錄項目。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，`Logger.debug("This is a message!")`記錄的層級Debug。以下是可能值：
  - 警示
  - 嚴重
  - 偵錯
  - Emergency (緊急)
  - 錯誤
  - Errored (錯誤)
  - 失敗
  - Info (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，`Logger.debug("Hello, World!")`記錄的訊息`"Hello, World!"`。

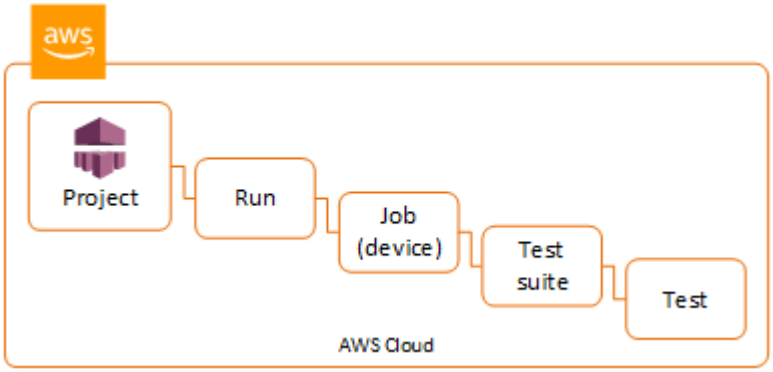
若要只顯示局部資訊：

- 若要顯示符合特定欄值的所有記錄項目，請在搜尋列中輸入值。例如，若要顯示「來源」值為的所有記錄項目`Harness`，請**`Harness`**在搜尋列中輸入。

- 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。從欄標題方塊中移除所有字元與在欄標題方塊*中輸入的作用相同。

若要下載裝置的所有記錄資訊，包括您執行的所有套件和測試，請選擇 [下載記錄]。

在裝置伺服器陣列中使用成品



裝置伺服器陣列會針對執行中的每個測試收集成品，例如報表、記錄檔和映像。

您可以下載測試執行期間所建立的成品：

檔案

測試運行期間生成的文件，包括 Device Farm 報告。如需詳細資訊，請參閱 在 Device Farm 中使用測試報告。

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。

使用成品 (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 在自訂測試環境中使用人工因素。

使用人工因素 (AWS CLI)

您可以使用列 AWS CLI 出測試回合成品。

主題

步驟 1：獲取您的 Amazon 資源名稱（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。下表顯示每個 AWS CLI 列表命令的輸入 ARN：

AWS CLI 清單指令	需要 ARN
list-projects	此命令會傳回所有專案，而且不需要 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：列出您的工件

列表 AWS CLI 工件命令返回成品的列表，如文件，屏幕截圖和日誌。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：下載您的文物

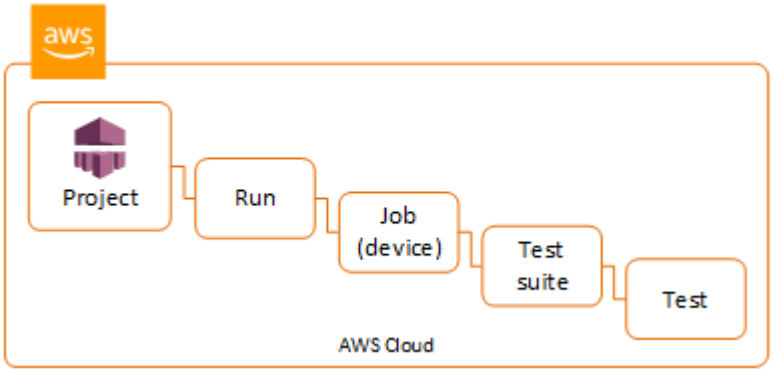
- 使用先前步驟的 URL 下載您的成品。此範例使用 curl 來下載 Android Appium Java 輸出檔：

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

使用成品 (API)

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

在自訂測試環境中使用人工因素



在自訂測試環境中，裝置伺服器陣列會收集自訂報表、記錄檔和映像等成品。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 在裝置伺服器陣列中使用成品。

標記 AWS 裝置農場資源

AWS 裝置伺服器陣列可與AWS資源群組標記 API。此 API 允許您管理中的資源AWS帳戶與標籤。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，請參閱成本分配和標記在關於 AWS 帳單和成本管理。
- 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
- 按標籤查找資源
- 移除資源的標籤

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TagResources從操作resourcegroupstaggingapi端點。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。

按標籤查找資源

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。

下列範例查詢裝置伺服器陣列桌面瀏覽器測試工作階段 (devicefarm:testgrid-session資源) 與標籤stack具有價值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"])
```

在 AWS Device Farm 中使用測試類型

本節說明測試架構和內建測試類型的裝置伺服器陣列支援。

測試框架

Device Farm 支援下列行動自動化測試架構：

安卓應用測試框架

- 使用 Appium 和 AWS Device Farm
- 使用適用於安卓和 AWS Device Farm 的儀器

iOS 應用程序測試框架

- 使用 Appium 和 AWS Device Farm
- 使用適用於 iOS 和 AWS Device Farm 的 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 使用 Appium 和 AWS Device Farm。

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 使用自訂測試環境。

Appium 版本支持

對於在自定義環境中運行的測試，Device Farm 支持 Appium 版本 1。如需詳細資訊，請參閱 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，而不必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：

- 內置：模糊（安卓和 iOS）

使用 Appium 和 AWS Device Farm

本節說明如何設定、封裝 Appium 測試並上傳至 Device Farm。Appium 是用於自動化本地和移動 Web 應用程序的開源工具。有關更多信息，請參閱 Appium 網站上的 Appium 簡介。

有關示例應用程序和工作測試的鏈接，請參閱適用於 Android 的 Device Farm 示例應用程序和適用於 iOS 的 Device Farm 示例應用程序 GitHub。

版本支援

對各種架構和程式設計語言的支援取決於所使用的語言。

Device Farm 支援所有 Appium 1.x 和 2.x 伺服器版本。對於 Android，您可以選擇任何主要的 Appium 版本。`devicefarm-cli`例如，要使用 Appium 服務器版本 2，請將以下命令添加到測試規格 YAML 文件中：

```
phases:
  install:
    commands:
      # To install a newer version of Appium such as version 2:
      - export APPIUM_VERSION=2
      - devicefarm-cli use appium $APPIUM_VERSION
```

對於 iOS，您可以使用avm或npm命令選擇特定的 Appium 版本。例如，要使用avm命令將 Appium 服務器版本設置為 2.1.2，請將以下命令添加到測試規格 YAML 文件中：

```
phases:
  install:
    commands:
      # To install a newer version of Appium such as version 2.1.2:
      - export APPIUM_VERSION=2.1.2
      - avm $APPIUM_VERSION
```

使用命npm令使用 Appium 2 的最新版本，將這些命令添加到測試規格 YAML 文件中：

```
phases:
  install:
    commands:
      - export APPIUM_VERSION=2
```

```
      - npm install -g appium@$APPIUM_VERSION
```

如需有關devicefarm-cli或任何其他 CLI 命令的詳細資訊，請參閱 AWS CLI 參考資料。

若要使用架構的所有功能 (例如註解)，請選擇自訂測試環境，然後使用 AWS CLI 或Device Farm主控台上傳自訂測試規格。

主題

- 配置您的 Appium 測試包
- 建立壓縮的測試套件檔案
- 將測試套件上傳至裝 Device Farm
- 拍攝測試的屏幕截圖（可選）

配置您的 Appium 測試包

使用以下指示來設定您的測試套件。

Java (JUnit)

1. 修改pom.xml以將打包設置為 JAR 文件：

   ```
   <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將測試構建到 JAR 文件中。

   以下插件將您的測試源代碼（src/test目錄中的任何內容）構建到 JAR 文件中：

   ```
   <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pom.xml以用maven-dependency-plugin來將依賴項構建為 JAR 文件。

   以下插件將您的依賴關係複製到dependency-jars目錄中：

```
<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，其中包含組建輸出目錄和目錄根目錄中的所有項目dependency-jars：

```
<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以用maven-assembly-plugin於將測試和所有依賴關係打包到單個 .zip 文件中。

   下列外掛程式會使用上述組件，在每次執行時mvn package建立zip-with-dependencies在 build 輸出目錄中名為的 .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以將打包設置為 JAR 文件：

   ```
   <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將測試構建到 JAR 文件中。

   以下插件將您的測試源代碼（src/test目錄中的任何內容）構建到 JAR 文件中：

   ```
   <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pom.xml以用maven-dependency-plugin來將依賴項構建為 JAR 文件。

   以下插件將您的依賴關係複製到dependency-jars目錄中：

```
<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，其中包含組建輸出目錄和目錄根目錄中的所有項目`dependency-jars`：

```
<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以用maven-assembly-plugin於將測試和所有依賴關係打包到單個 .zip 文件中。

   下列外掛程式會使用上述組件，在每次執行時mvn package建立zip-with-dependencies在 build 輸出目錄中名為的 .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

要打包 Appium Node.js 測試並將其上傳到 Device Farm，您必須在本地計算機上安裝以下內容：

- 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，請參閱上 GitHub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 會預期您的測試儲存在中`tests/`。您可以使用 `find` 來顯示資料夾內的所有檔案：

```
$ find tests/
```

確認這些文件包含您要在 Device Farm 上運行的測試套件

```
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/
```

確認輸出顯示您要在 Device Farm 上運行的測試。

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

要打包 Appium Ruby 測試並將其上傳到 Device Farm，您必須在本地計算機上安裝以下內容：

- 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. 配置捆綁器以編譯所需測試平台的軟件包：

```
bundle config specific_platform true
```

5. 更新您的 .lock 文件以添加運行測試所需的平台。

   - 如果您正在編譯測試以在 Android 設備上運行，請運行此命令以配置 Gemfile 以使用 Android 測試主機的依賴關係：

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

   - 如果您正在編譯測試以在 iOS 設備上運行，請運行此命令以配置 Gemfile 以使用 iOS 測試主機的依賴關係：

```
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，壓縮測試包中文件的文件夾結構很重要，並且某些存檔工具將隱式更改 ZIP 文件的結構。我們建議您遵循以下指定的命令列公用程式，而不要使用本機桌面 (例如 Finder 或 Windows 檔案總管) 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 列的`MyTests.zip`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 列的MyTests.zip檔案。

將測試套件上傳至裝 Device Farm

您可以使用 Device Farm 主控台來上傳測試。

1. 登入 Device Farm 主控台，網址為 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」，輸入專案名稱，然後選擇「提交」。

   如果您已經有一個項目，則可以選擇它將測試上傳到該項目。
4. 開啟您的專案，然後選擇 Create a new run (建立新執行)。
5. 若是原生 Android 和 iOS 測試

   在 [選擇應用程式] 頁面上，選擇 [行動應用程式]，然後選取 [選擇檔案] 以上傳應用程式的可散發套件

   > **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

   在選擇應用程式頁面上，選擇 [Web 應用程式]。
6. 為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。
7. 選擇下一步。
8. 在「配置」頁面的「設置測試框架」部分中，選擇 Appium **##**，然後選擇文件。
9. 瀏覽並選擇包含測試的 .zip 檔案。.zip 檔案必須遵循配置您的 Appium 測試包中所述的格式。
10. 選擇在自訂環境中執行測試。此執行環境允許完全控制測試設置，拆卸和調用，以及選擇特定版本的運行時和 Appium 服務器。您可以通過測試規格文件配置您的自定義環境。如需詳細資訊，請參閱在 AWS Device Farm 中使用自訂測試環境。
11. 選擇 [下一步]，然後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 中建立測試回合。

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

拍攝測試的屏幕截圖（可選）

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。

在 AWS Device Farm 中使用安卓測試

Device Farm 為 Android 設備的多種自動化測試類型和兩個內置測試提供支持。

安卓應用測試框架

下列測試適用於 Android 裝置。

- 使用 Appium 和 AWS Device Farm
- 使用適用於安卓和 AWS Device Farm 的儀器

安卓系統的內置測試類型

有一種內置的測試類型可用於 Android 設備。

- 內置：模糊（安卓和 iOS）

使用適用於安卓和 AWS Device Farm 的儀器

Device Farm 為儀器儀表提供支持 (JUnit, 咖啡, Robotium, 或任何基於儀器的測試) 為 Android.

Device Farm 還提供了一個示例 Android 應用程序以及指向三個 Android 自動化框架（包括儀器（Espresso）中的工作測試的鏈接。適用於 Android 的 Device Farm 範例應用程式可在上下載 GitHub。

主題

- 什麼是儀器？

什麼是儀器？

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。有關更多信息，請參閱 Android 開發人員工具文檔的測試類型和位置部分中的檢測測試。

上傳您的安卓儀器測試

使用 Device Farm 主控台上傳測試。

1. 登入 Device Farm 主控台，網址為 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，選擇您要上傳測試的專案。

> **Tip**
>
> 您可以使用搜索欄按名稱過濾項目列表。
> 若要建立專案，請按照 在 AWS 裝置伺服器陣列中建立專案 中的說明進行操作。

4. 如果系統顯示 Create a new run (建立新執行) 按鈕，請選擇此按鈕。
5. 在選擇應用程式頁面上，選取選擇檔案。
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. 選擇下一步。
8. 在 [設定] 頁面的 [安裝程式測試架構] 區段中，選擇 [檢測]，然後選取 [選擇檔案]。
9. 瀏覽並選擇包含測試的 .apk 檔案。
10. 選擇 [下一步]，然後完成剩餘指示以選取裝置並開始執行。

在 Android 儀器測試中截取屏幕截圖

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");
```

在測試運行期間，Device Farm 會從設備上的以下位置獲取屏幕截圖（如果存在），然後將它們添加到測試報告中：

- `/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`*

安卓儀器測試的其他注意事項

System Animations (動畫系統)

根據 Espresso 測試的 Android 文檔，建議在實際設備上進行測試時關閉系統動畫。設 Device Farm 在使用 Android 執行時會自動禁用窗口動畫縮放，過渡動畫縮放和動畫持續時間縮放設置。UnitRunner

Test Recorders (測試錄製器)

Device Farm 支持框架，如 Robotium，具有 record-and-playback 腳本工具。

標準模式測試剖析

在運行的標準模式下，Device Farm 解析您的測試套件，並識別將運行的唯一測試類和方法。這是通過一個名為 Dex 測試解析器的工具完成的。

當給定 Android 儀器 .apk 文件作為輸入時，解析器將返回與 JUnit 3 和 JUnit 4 約定匹配的測試的完全限定方法名稱。

要在本地環境中對其進行測試：

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

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

在 AWS Device Farm 中使用 iOS 測試

Device Farm 為 iOS 裝置提供多種自動化測試類型的支援，以及內建測試。

iOS 應用程序測試框架

下列測試適用於 iOS 裝置。

- 使用 Appium 和 AWS Device Farm
- 使用適用於 iOS 和 AWS Device Farm 的 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。

- 內置：模糊（安卓和 iOS）

使用適用於 iOS 和 AWS Device Farm 的 XCTest

使用 Device Farm，您可以使用 XCTest 框架在實際設備上測試您的應用程序。有關 XCTest 的更多信息，請參閱使用 Xcode 測試中的基礎知識。

要運行測試，請為測試運行創建軟件包，然後將這些軟件包上傳到 Device Farm。

主題

- 為您的 XCTest 運行創建軟件包
- 將 XCTest 運行的軟件包上傳到 Device Farm

為您的 XCTest 運行創建軟件包

要使用 XCTest 框架測試您的應用程序，Device Farm 需要以下內容：

- 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。完成下列步驟來建立套件，以便將套件上傳至 Device Farm 列。

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`。

   在稍後的步驟中，您將此檔案提供給 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。

   在稍後的步驟中，您將此檔案提供給 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 運行的軟件包上傳到 Device Farm

使用 Device Farm 主控台來上傳測試的套件。

1. 登入 Device Farm 主控台，網址為 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 裝置伺服器陣列中建立專案。

   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 Create a new run (建立新執行)。
5. 在選擇應用程式頁面上，選擇行動應用程式。
6. 選取 [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. 上傳完成後，選擇 [下一步]。
9. 在「配置」頁面的「安裝測試框架」部分中，選擇「XCT est」。然後，選擇選擇文件。
10. 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。
11. 上傳完成後，選擇 [下一步]。
12. 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。
13. 設定執行之後，在 [檢閱並開始執行] 頁面上，選擇 [確認並開始執行]。

    Device Farm 會執行測試，並在主控台中顯示結果。

使用適用於 iOS 和 AWS Device Farm 的 XCTest 使用者介面測試架構

Device Farm 為 iOS 的 XCTest UI 測試框架提供支持。具體來說，Device Farm 支持用目標 C 和斯威夫特編寫的 XCTest UI 測試。

主題

什麼是測試用戶界面測試框架？

XCTest UI 架構是由 Xcode 7 引入的新測試架構。這個架構可為 XCTest 擴展 UI 測試功能。如需詳細資訊，請參閱 iOS Developer Library 中的 使用者界面測試。

準備您的 iOS 測試使用者介面測試

您的 iOS XCTest UI 測試執行器 bundle 必須包含在格式正確的 .ipa 檔案中。

若要建立 .ipa 檔案，請將 my-project-name UITST 執行者 .app 套件組合放在空的有效負載目錄中。接著，將「承載」目錄封存為 .zip 檔案，然後將副檔名變更為 .ipa。當您建置測試專案時，Xcode 會產生 UITest-Runner.app bundle。您可以在專案的「產品」目錄中找到該 bundle。

上傳您的 iOS 測試使用者介面測試

使用 Device Farm 主控台上傳測試。

1. 登入 Device Farm 主控台，網址為 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，選擇您要上傳測試的專案。

> **Tip**
>
> 您可以使用搜索欄按名稱過濾項目列表。
> 若要建立專案，請遵循中的指示 在 AWS 裝置伺服器陣列中建立專案

4. 如果系統顯示 Create a new run (建立新執行) 按鈕，請選擇此按鈕。
5. 在選擇應用程式頁面上，選取選擇檔案。
6. 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。

7. 選擇下一步。
8. 在 [設定] 頁面的 [安裝程式測試架構] 區段中，選擇 [XCTest UI]，然後選取 [選擇檔案]。
9. 瀏覽並選擇包含 iOS XCTest UI 測試執行器的 .ipa 檔案。
10. 選擇 [下一步]，然後完成剩餘的指示以選取要在其上執行測試的裝置並開始執行。

在 iOS 測試用戶界面測試中獲取屏幕截圖

XCTest UI 測試會自動為測試的每個步驟擷取螢幕擷取畫面。這些螢幕擷取畫面會顯示在裝 Device Farm 測試報告 您無需設定其他程式碼。

在 AWS Device Farm 中使用 Web 應用程式測試

Device Farm 提供測試與 Appium 的 Web 應用程序。如需在 Device Farm 上設定 Appium 測試的詳細資訊，請參閱。the section called “Appium”

計量和未計量裝置的規則

計量是指裝置的計費。依預設，會計量 Device Farm 的計量費用，而且在免費試用分鐘數用完之後，您需支付每分鐘的費用。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。

在 AWS 裝置伺服器陣列中使用內建測試

Device Farm 提供適用於 Android 和 iOS 裝置的內建測試類型的支援。

內建測試類型

內建測試可讓您測試應用程式，而無需編寫指令碼。

- 內置：模糊（安卓和 iOS）

使用裝置伺服器陣列的內建模糊測試

Device Farm 提供內建的模糊測試類型。

什麼是內置的模糊測試？

內建模糊測試會將使用者界面事件隨機傳送到裝置，然後報告結果。

使用內建模糊測試類型

使用 Device Farm 主控台執行內建的模糊測試。

1. 登入 Device Farm 主控台，網址為 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. 在項目列表中，選擇要運行內置模糊測試的項目。

> **Tip**
>
> 您可以使用搜索欄按名稱過濾項目列表。
> 若要建立專案，請按照 在 AWS 裝置伺服器陣列中建立專案 中的說明進行操作。

4. 如果系統顯示 Create a new run (建立新執行) 按鈕，請選擇此按鈕。
5. 在選擇應用程式頁面上，選取選擇檔案。
6. 瀏覽並選擇您要執行內建模糊測試的應用程式檔案。
7. 選擇下一步。
8. 在「設定」頁面的「安裝程式測試架構」區段中，選擇「內建：Fuzz」。
9. 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。
- 事件節流：指定介於 0 到 1,000 之間的數字，代表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。

10. 選擇 [下一步]，然後完成剩餘指示以選取裝置並開始執行。

在 AWS Device Farm 中使用自訂測試環境

AWS Device Farm 可讓您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。若要深入了解 Device Farm 中的環境，請參閱測試環境。

與標準模式相比，「自訂模式」的優點包括：

- 更快的 end-to-end 測試執行：測試包不被解析為檢測套件中的每個測試，從而避免了預處理/後處理開銷。
- 即時記錄和影片串流：使用「自訂模式」時，用戶端測試記錄和影片會即時串流。標準模式無法使用此功能。
- 捕獲所有工件：在主機和設備上，「自定義模式」允許您捕獲所有測試成品。這在標準模式下可能無法實現。
- 更一致且可複製的本地環境：在標準模式下，將分別為每個單獨的測試提供構件，這在某些情況下可能會有所幫助。但是，由於 Device Farm 以不同方式處理每個執行的測試，因此您的本地測試環境可能與原始配置不同。

  相反地，自訂模式可讓您使 Device Farm 測試執行環境與本機測試環境保持一致。

使用 YAML 格式的測試規格 (測試規格) 檔案來設定自訂環境。Device Farm 為每種受支持的測試類型提供了一個默認的測試規格文件，這些文件可以原樣使用或自定義；可以將測試過濾器或配置文件等自定義添加到測試規格中。編輯的測試規格可以保存以備將 future 的測試運行。

如需詳細資訊，請參閱使用AWS CLI和上傳自訂測試規格在 Device Farm 中建立測試回合。

主題

測試規範語法

這是 YAML 測試規格檔案結構：

```
version: 0.1

phases:
  install:
    commands:
      - command
      - command
  pre_test:
    commands:
      - command
      - command
  test:
    commands:
      - command
      - command
  post_test:
    commands:
      - command
      - command

artifacts:
  - location
  - location
```

測試規格包含下列各項：

`version`

反映 Device Farm 支援的測試規格版本。目前版本編號為 0.1。

`phases`

此區段包含會在測試執行中執行的命令群組。

允許的測試階段名稱為：

`install`

選用。

已安裝裝 Device Farm 支援之測試架構的預設相依性。此階段包含該裝 Device Farm 在安裝期間執行的其他命令 (如果有的話)。

pre_test

選用。

在您進行自動測試執行之前會執行的命令 (若有)。

test

選用。

在您進行自動測試執行期間會執行的命令 (若有)。如果測試階段中有任何命令失敗，測試就會標示為失敗。

post_test

選用。

在您進行自動測試執行之後會執行的命令 (若有)。

artifacts

選用。

裝置伺服器陣列會從此處指定的位置收集成品，例如自訂報告、記錄檔和映像。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 在自訂測試環境中使用人工因素。

> **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

這是設定 Appium Java TestNG 測試回合的「Device Farm」YAML 測試規格的範例：

```
version: 0.1
```

```
# This flag enables your test to run using Device Farm's Amazon Linux 2 test host when
 scheduled on
# Android devices. By default, iOS device tests will always run on Device Farm's macOS
 test hosts.
# For Android, you can explicitly select your test host to use our Amazon Linux 2
 infrastructure.
# For more information, please see:
# https://docs.aws.amazon.com/devicefarm/latest/developerguide/amazon-linux-2.html
android_test_host: amazon_linux_2

# Phases represent collections of commands that are executed during your test run on
 the test host.
phases:

  # The install phase contains commands for installing dependencies to run your tests.
  # For your convenience, certain dependencies are preinstalled on the test host.

  # For Android tests running on the Amazon Linux 2 test host, many software libraries
 are available
  # from the test host using the devicefarm-cli tool. To learn more, please see:
  # https://docs.aws.amazon.com/devicefarm/latest/developerguide/amazon-linux-2-
devicefarm-cli.html

  # For iOS tests, you can use the Node.JS tools nvm, npm, and avm to setup your
 environment. By
  # default, Node.js versions 16.20.2 and 14.19.3 are available on the test host.
  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.
      - |-
        if [ $DEVICEFARM_DEVICE_PLATFORM_NAME = "Android" ];
        then
          devicefarm-cli use node 16;
        else
          # For iOS, use "nvm use" to switch between the two preinstalled NodeJS
 versions 14 and 16,
          # and use "nvm install" to download a new version of your choice.
          nvm use 16;
        fi;
      - node --version
```

```
    # Use the devicefarm-cli to select a preinstalled major version of Appium on
Android.
    # Use avm or npm to select Appium for iOS.
    - |-
      if [ $DEVICEFARM_DEVICE_PLATFORM_NAME = "Android" ];
      then
        # For Android, the Device Farm service automatically 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 instead use NPM to
install it:
        # npm install -g appium@2.1.3;
        devicefarm-cli use appium 2;
      else
        # For iOS, Appium versions 1.22.2 and 2.2.1 are preinstalled and selectable
through avm.
        # For all other versions, please use npm to install them. For example:
        # npm install -g appium@2.1.3;
        # Note that, for iOS devices, Appium 2 is only supported on iOS version 14
and above using
        # NodeJS version 16 and above.
        avm 2.2.1;
      fi;
    - appium --version

    # For Appium version 2, for Android tests, Device Farm automatically updates the
preinstalled
    # UIAutomator2 driver over time to incorporate the latest minor and patch
versions for its major
    # version 2. If you want to install a specific 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;

    # For Appium version 2, for iOS tests, the XCUITest driver is preinstalled using
version 5.7.0
```

```
     # 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@5.8.1;
     #   fi;

     # We recommend setting the Appium server's base path explicitly for accepting
commands.
     - export APPIUM_BASE_PATH=/wd/hub

     # Install the NodeJS dependencies.
     - cd $DEVICEFARM_TEST_PACKAGE_PATH
     # First, install dependencies which were packaged with the test package using
npm-bundle.
     - npm install *.tgz
     # Then, optionally, install any additional dependencies using npm install.
     # If you do run these commands, we strongly recommend that you include your
package-lock.json
     # file with your test package so that the dependencies installed on Device Farm
match
     # the dependencies you've installed locally.
     # - cd node_modules/*
     # - npm install

 # The pre-test phase contains commands for setting up your test environment.
 pre_test:
   commands:
     # Device farm provides different pre-built versions of WebDriverAgent, an
essential Appium
     # dependency for iOS devices, and each version is suggested for different
versions of Appium:
     # DEVICEFARM_WDA_DERIVED_DATA_PATH_V8: this version is suggested for Appium 2
     # DEVICEFARM_WDA_DERIVED_DATA_PATH_V7: this version is suggested for Appium 1
     # Additionally, for iOS versions 16 and below, the device unique identifier
(UDID) needs
     # to be slightly modified for Appium tests.
     - |-
       if [ $DEVICEFARM_DEVICE_PLATFORM_NAME = "iOS" ];
       then
         if [ $(appium --version | cut -d "." -f1) -ge 2 ];
```

```
          then
            DEVICEFARM_WDA_DERIVED_DATA_PATH=$DEVICEFARM_WDA_DERIVED_DATA_PATH_V8;
          else
            DEVICEFARM_WDA_DERIVED_DATA_PATH=$DEVICEFARM_WDA_DERIVED_DATA_PATH_V7;
          fi;

          if [ $(echo $DEVICEFARM_DEVICE_OS_VERSION | cut -d "." -f 1) -le 16 ];
          then
            DEVICEFARM_DEVICE_UDID_FOR_APPIUM=$(echo $DEVICEFARM_DEVICE_UDID | tr -d
 "-");
          else
            DEVICEFARM_DEVICE_UDID_FOR_APPIUM=$DEVICEFARM_DEVICE_UDID;
          fi;
        fi;

      # Appium downloads Chromedriver using a feature that is considered insecure for
 multitenant
      # environments. This is not a problem for Device Farm because each test host is
 allocated
      # exclusively for one customer, then terminated entirely. For more information,
 please see
      # https://github.com/appium/appium/blob/master/packages/appium/docs/en/guides/
security.md

      # 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:app\": \"$DEVICEFARM_APP_PATH\", \
            \"appium:udid\":\"$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:app\": \"$DEVICEFARM_APP_PATH\", \
            \"appium:udid\":\"$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 will 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.
      # When compiling with npm-bundle, the test folder can be found in the
node_modules/*/ subdirectory.
      - cd $DEVICEFARM_TEST_PACKAGE_PATH/node_modules/*
      - echo "Starting the Appium NodeJS test"

      # Enter your command below to start the tests. The command should be the same
command as the one
```

```
      # you use to run your tests locally from the command line. An example, "npm
 test", is given below:
      - npm test

  # 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.
# 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
```

使用 Amazon Linux 2 測試環境進行安卓測試

AWS Device Farm 利用 Amazon Elastic Compute Cloud (EC2) 主機執行 Amazon Linux 2 來執行安卓測試。當您排程測試回合時，Device Farm 會為每個裝置分配專用主機，以獨立執行測試。主機在測試運行後與任何生成的成品一起終止。

Amazon Linux 2 測試主機是最新的安卓測試環境，取代了以前的基於 Ubuntuu 的系統。使用您的測試規格文件，您可以選擇在 Amazon Linux 2 環境上運行 Android 測試。

Amazon Linux 2 主機提供了幾個優點：

- 測試速度更快，更可靠：與舊版主機相比，新的測試主機可大幅提高測試速度，尤其是縮短測試開始時間。Amazon Linux 2 主機在測試期間也展現出更高的穩定性和可靠性。
- 用於手動測試的增強遠程訪問：升級到最新的測試主機和改進可以為 Android 手動測試提供更低的延遲和更好的視頻性能。
- 標準軟體版本選擇：Device Farm 現在可將測試主機和 Appium 架構版本上的主要程式設計語言支援標準化。對於支持的語言（目前是 Java，Python，Node.js 和 Ruby）和 Appium，新的測試主機在啟動後不久提供長期穩定版本。透過此`devicefarm-cli`工具進行集中式版本管理，可讓測試規格檔案開發，並在不同架構中獲得一致

主題

支援的軟體

Amazon Linux 2 測試主機已預先安裝許多必要的軟體程式庫，以支援裝 Device Farm 測試架構，並在啟動時提供現成的測試環境。對於任何其他必要軟體，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 內的私有來源 (如需詳細資訊，請參閱 VPC ENI)。如需詳細資訊，請參閱測試規格檔案範例。

主機上目前提供下列軟體版本：

軟體程式庫	軟體版本	在測試規格文件中使用的命令
Python	3.8	`devicefarm-cli use python 3.8`
	3.9	`devicefarm-cli use python 3.9`
	3.10	`devicefarm-cli use python 3.10`
Java	8	`devicefarm-cli use java 8`
	11	`devicefarm-cli use java 11`
	17	`devicefarm-cli use java 17`

NodeJS	16	`devicefarm-cli use node 16`
	18	`devicefarm-cli use node 18`
Ruby	2.7	`devicefarm-cli use ruby 2.7`
	3.2	`devicefarm-cli use ruby 3.2`
Appium	1	`devicefarm-cli use appium 1`
	2	`devicefarm-cli use appium 2`

測試主機還包括用於每個軟件版本的常用支持工具，例如`pip`和軟件`npm`包管理器（分別包含在 Python 和 Node.js 中）以及 Appium 等工具的依賴關係（例如 Appium UiAutomator2 驅動程序）。這可確保您擁有與支持的測試框架一起工作所需的工具。

`devicefarm-cli` 工具

Amazon Linux 2 測試主機使用標準化版本管理工具`devicefarm-cli`來選取軟體版本。此工具與「裝置伺服器陣列測試主機」不同， AWS CLI 且僅在「Device Farm 測試 使用`devicefarm-cli`，您可以切換到測試主機上任何預先安裝的軟體版本。這提供了一個直接的方式來維護您的 Device Farm 測試規格檔案隨著時間的推移，並為您提供可預測的機制來在 future 升級軟體版本。

下面的代碼片段顯示了以下`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.
```

讓我們回顧一些使用devicefarm-cli. 要使用該工具在測試規格文件中將 Python 版本從 *3.10* 更改為 *3.9*，請運行以下命令：

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

Appium ### 1 ### 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，pip例如 Python 和 npm NodeJS。

安卓測試主機選擇

> **Warning**
>
> 舊版安卓測試主機將不再於 2024 年 10 月 21 日提供。請注意，棄用過程分為幾個日期：
>
> - 自 2024 年 4 月 22 日起，任何新帳戶的工作都會導向至已升級的測試主機。
> - 在 2024 年 9 月 2 日，所有新的或修改過的測試規格檔案都必須針對升級的測試主機。
> - 在 2024 年 10 月 21 日，工作將無法再在舊版測試主機上執行。
>
> 將測試規格文件設置為amazon_linux_2主機以防止兼容性問題。

對於 Android 測試，Device Farm 列需要測試規格檔案中的以下欄位，才能選擇 Amazon Linux 2 測試主機：

```
android_test_host: amazon_linux_2 | legacy
```

用amazon_linux_2於在 Amazon Linux 2 測試主機上執行測試：

```
android_test_host: amazon_linux_2
```

在這裡進一步了解 Amazon Linux 2 的好處。

Device Farm 建議使用 Amazon Linux 2 主機進行 Android 測試，而不是使用舊版主機環境。如果您希望使用舊版環境，請使legacy用在舊版測試主機上運行測試：

```
android_test_host: legacy
```

預設情況下，未選擇測試主機的測試規格檔案將在舊版測試主機上執行。

棄用的語法

以下是在測試規格文件中選擇 Amazon Linux 2 的棄用語法：

```
preview_features:
  android_amazon_linux_2_host: true
```

如果您使用這個標誌，您的測試將繼續在 Amazon Linux 2 上運行。不過，我們強烈建議您移除preview_features旗標區段，並將其取代為新android_test_host欄位，以避免 future 的維護額外負荷。

> **Warning**
>
> 在測試規範文件中使用android_test_host和android_amazon_linux_2_host標誌將返回錯誤。只能使用一個；我們建議使用android_test_host。

測試規格檔案範例

以下代碼片段是 Device Farm 測試規範文件的示例，該文件使用適用於 Android 的 Amazon Linux 2 測試主機配置 Appium NodeJS JS 測試運行：

```
version: 0.1

# This flag enables your test to run using Device Farm's Amazon Linux 2 test host. For
 more information,
# please see https://docs.aws.amazon.com/devicefarm/latest/developerguide/amazon-
linux-2.html
android_test_host: amazon_linux_2

# Phases represent collections of commands that are executed during your test run on
 the test host.
phases:

  # The install phase contains commands for installing dependencies to run your tests.
  # For your convenience, certain dependencies are preinstalled on the test host. To
 lean about which
  # software is included with the host, and how to install additional software, please
 see:
  # https://docs.aws.amazon.com/devicefarm/latest/developerguide/amazon-linux-2-
supported-software.html

  # Many software libraries you may need are available from the test host using the
 devicefarm-cli tool.
  # To learn more about what software is available from it and how to use it, please
 see:
  # https://docs.aws.amazon.com/devicefarm/latest/developerguide/amazon-linux-2-
devicefarm-cli.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 18
      - 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 automatically 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@2.1.3

     # For Appium version 2, Device Farm automatically updates the preinstalled
UIAutomator2 driver
     # over time to incorporate the latest minor and patch versions for its major
version 2. If you
     # want to install a specific version of the driver, you can use the Appium
extension CLI to
     # uninstall the existing UIAutomator2 driver and install your desired version:
     # - appium driver uninstall uiautomator2
     # - appium driver install uiautomator2@2.34.0

     # We recommend setting the Appium server's base path explicitly for accepting
commands.
     - export APPIUM_BASE_PATH=/wd/hub

     # Install the NodeJS dependencies.
     - cd $DEVICEFARM_TEST_PACKAGE_PATH
     # First, install dependencies which were packaged with the test package using
npm-bundle.
     - npm install *.tgz
     # Then, optionally, install any additional dependencies using npm install.
     # If you do run these commands, we strongly recommend that you include your
package-lock.json
     # file with your test package so that the dependencies installed on Device Farm
match
     # the dependencies you've installed locally.
     # - cd node_modules/*
     # - npm install

 # The pre-test phase contains commands for setting up your test environment.
 pre_test:
   commands:

     # Appium downloads Chromedriver using a feature that is considered insecure for
multitenant
     # environments. This is not a problem for Device Farm because each test host is
allocated
     # exclusively for one customer, then terminated entirely. For more information,
please see
```

```
      # https://github.com/appium/appium/blob/master/packages/appium/docs/en/guides/
security.md

      # 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\", \
          \"appium:platformVersion\": \"$DEVICEFARM_DEVICE_OS_VERSION\", \
          \"appium:chromedriverExecutableDir\":
 \"$DEVICEFARM_CHROMEDRIVER_EXECUTABLE_DIR\", \
          \"appium:automationName\": \"UiAutomator2\"}" \
          >> $DEVICEFARM_LOG_DIR/appium.log 2>&amp;1 &amp;

      # This code will 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.
```

```
      # When compiling with npm-bundle, the test folder can be found in the
 node_modules/*/ subdirectory.
      - cd $DEVICEFARM_TEST_PACKAGE_PATH/node_modules/*
      - echo "Starting the Appium NodeJS test"

      # Enter your command below to start the tests. The command should be the same
 command as the one
      # you use to run your tests locally from the command line. An example, "npm
 test", is given below:
      - npm test

  # 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.
# 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
```

遷移到 Amazon Linux 2 測試主機

> **Warning**
>
> 舊版安卓測試主機將不再於 2024 年 10 月 21 日提供。請注意，棄用過程分為幾個日期：
>
> - 自 2024 年 4 月 22 日起，任何新帳戶的工作都會導向至已升級的測試主機。
> - 在 2024 年 9 月 2 日，所有新的或修改過的測試規格檔案都必須針對升級的測試主機。
> - 在 2024 年 10 月 21 日，工作將無法再在舊版測試主機上執行。
>
> 將測試規格文件設置為`amazon_linux_2`主機以防止兼容性問題。

若要將現有測試從舊版主機遷移到新的 Amazon Linux 2 主機，請根據預先存在的測試規格檔案開發新的測試規格檔案。建議的方法是從測試類型的新預設測試規格檔案開始。然後，將相關命令從舊的測試規格文件遷移到新文件，將舊文件保存為備份。這可讓您在重複使用現有程式碼的同時，利用新主機的最佳化預設規格。它可確保您獲得針對測試進行最佳配置的新主機的全部優勢，同時保留舊版測試規格以供參考，以供您根據新環境調整指令。

您可以使用下列步驟建立新的 Amazon Linux 2 測試規格檔案，同時重複使用舊測試規格檔案中的命令：

1. 登入 Device Farm 主控台，網址為 https://console.aws.amazon.com/devicefarm。
2. 導覽至包含自動化測試的 Device Farm 專案。
3. 選擇在項目中創建一個新的測試運行。
4. 為您的測試框架選擇以前使用的應用程序和測試包。
5. 選擇在自訂環境中執行測試。
6. 從「測試規格」下拉式功能表中，選擇您目前用於舊版測試主機上測試的測試規格檔案。
7. 複製此文件的內容並將其粘貼到本地文本地編輯器中以供以後參考。
8. 在「測試規格」下拉式功能表中，將您的測試規格選取項變更為最新的預設測試規格檔案。
9. 選擇編輯，您將進入測試規格編輯界面。您會注意到，在測試規範文件的前幾行中，它已經選擇加入新的測試主機：

   ```
   android_test_host: amazon_linux_2
   ```

10. 在此處查看選擇測試主機的語法以及此處測試主機之間的主要差異。
11. 從步驟 6 中選擇性地將本機儲存的測試規格檔案中的指令新增並編輯到新的預設測試規格檔案中。然後，選擇「另存新檔」以儲存新的規格檔案。您現在可以在 Amazon Linux 2 測試主機上安排測試運行。

新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 Amazon Linux 2 測試主機，並從舊版測試主機轉換測試時，請注意以下主要環境差異：

- 選取軟體版本：在許多情況下，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，則可能需要立即在 Amazon Linux 2 測試主機中使用`devicefarm-cli`。在絕大多數的使用案例中，我們建議客戶明確選取他們使用的軟體版本。透過選取的軟體版本`devicefarm-`

cli，您將擁有可預測且一致的使用經驗，如果 Device Farm 計劃從測試主機移除該版本，則會收到大量警告。

此外，軟件選擇工具（例如 nvmpyenv，avm，和）rvm已被刪除，以支持新的devicefarm-cli軟件選擇系統。

- 可用的軟體版本：已移除許多先前預先安裝的軟體版本，並新增了許多新版本。因此，請確定在使用devicefarm-cli來選取您的軟體版本時，請選取支援的版本清單中的版本。
- 在舊版主機測試規格檔案中以絕對路徑進行硬式編碼的任何檔案路徑在 Amazon Linux 2 測試主機中很可能無法如預期般運作；通常不建議將其用於測試規格檔案。我們建議您針對所有測試規格檔案程式碼使用相對路徑和環境變數。此外，請注意，您測試所需的大多數二進製文件都可以在主機的 PATH 中找到，以便它們立即從 spec 文件中運行只使用其名稱（例如 appium）。
- 新的測試主機目前不支援效能資料收集。
- 作業系統版本：舊版測試主機是以 Ubuntu 作業系統為基礎，而新主機則以 Amazon Linux 2 為基礎。因此，用戶可能會注意到可用的系統庫和系統庫版本存在一些差異。
- 對於 Appium Java 用戶，新的測試主機在其類路徑中不包含任何預先安裝的 JAR 文件，而先前的主機包含 TestNG 框架的一個文件（通過環境變量）。$DEVICEFARM_TESTNG_JAR我們建議客戶將測試框架的必要 JAR 文件包裝在其測試包中，並從其測試規範文件中刪除$DEVICEFARM_TESTNG_JAR變量的實例。如需詳細資訊，請參閱使用 Appium 和 AWS Device Farm。
- 對於 Appium 用戶，$DEVICEFARM_CHROMEDRIVER_EXECUTABLE環境變量已被刪除，以支持使客戶能夠訪問 Android 版 Chromedriver 的新方法。有關使用新環境變量的示例，請參閱我們的默認測試規範文件$DEVICEFARM_CHROMEDRIVER_EXECUTABLE_DIR。

> **Note**
>
> 我們強烈建議保留現有的 Appium 服務器命令從默認的測試規範文件原樣。

如果您對測試主機之間的差異有任何反饋或疑問，我們建議您通過支持案例與服務團隊聯繫。

環境變數

環境變數代表自動化測試所使用的值。您可以在 YAML 檔案和測試程式碼中使用這些環境變數。在自訂測試環境中，Device Farm 會在執行階段動態填入環境變數。

主題

一般環境變數

安卓測試

本節介紹 Device Farm 支持的 Android 平台測試常見的自定義環境變量。

$DEVICEFARM_DEVICE_NAME

您會執行測試的裝置名稱。其代表裝置的唯一裝置識別碼 (UDID)。

$DEVICEFARM_DEVICE_PLATFORM_NAME

裝置平台名稱。這是 Android 或 iOS。

$DEVICEFARM_DEVICE_OS_VERSION

裝置作業系統版本。

$DEVICEFARM_APP_PATH

主機上行動應用程式的路徑，測試會在此處執行。應用程式路徑僅適用於行動應用程式。

$DEVICEFARM_DEVICE_UDID

執行自動測試之行動裝置的唯一識別碼。

$DEVICEFARM_LOG_DIR

測試期間產生之日誌檔的路徑。默認情況下，此目錄中的所有文件都歸檔為 ZIP 文件，並在測試運行後作為成品提供。

$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_CHROMEDRIVER_EXECUTABLE_DIR

包含在 Appium 網絡和混合測試中使用必要的 Chromedriver 可執行文件的目錄的位置。

$ANDROID_HOME

安卓 SDK 安裝目錄的路徑。

> **Note**
>
> ANDROID_HOME環境變數僅適用於安卓系統的 Amazon Linux 2 測試主機上。

iOS 測試

本節說明 Device Farm 支援的 iOS 平台測試通用的自訂環境變數。

$DEVICEFARM_DEVICE_NAME

您會執行測試的裝置名稱。其代表裝置的唯一裝置識別碼 (UDID)。

$DEVICEFARM_DEVICE_PLATFORM_NAME

裝置平台名稱。這是 Android 或 iOS。

$DEVICEFARM_APP_PATH

主機上行動應用程式的路徑，測試會在此處執行。應用程式路徑僅適用於行動應用程式。

$DEVICEFARM_DEVICE_UDID

執行自動測試之行動裝置的唯一識別碼。

$DEVICEFARM_LOG_DIR

測試期間產生之日誌檔的路徑。

$DEVICEFARM_SCREENSHOT_PATH

在測試執行期間擷取之螢幕擷取畫面的路徑 (若有)。

Java 的 JUnit 環境變量

本節說明在自訂測試環境中 Appium Java JUnit 測試會使用的環境變數。

$DEVICEFARM_TESTNG_JAR

TestNG .jar 檔案的路徑。

$DEVICEFARM_TEST_PACKAGE_PATH

測試套件檔案之未解壓縮內容的路徑。

Java TestNG 中的環境變量

本節說明在自訂測試環境中 Appium Java TestNG 測試會使用的環境變數。

$DEVICEFARM_TESTNG_JAR

TestNG .jar 檔案的路徑。

$DEVICEFARM_TEST_PACKAGE_PATH

測試套件檔案之未解壓縮內容的路徑。

特殊環境變量

$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。

將測試從標準測試環境遷移到自定義測試環境

下列指南說明如何從標準測試執行模式切換至自訂執行模式。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提供客戶精細的報告和完全受管理的環境而建置。
2. 自訂模式：此測試執行模式是針對需要更快的測試執行速度、提升和轉移和實現與本機環境的同位檢查以及即時視訊串流的不同使用案例而建置。

移轉時的考量

本節列出移轉至自訂模式時要考慮的一些重要使用案例：

1. 速度：在標準執行模式下，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目。之後，Device Farm 會分別執行每個測試，並針對每個測試個別顯示記錄檔、影片和其他結果成品。但是，這會穩步增加總 end-to-end 測試執行時間，因為服務端上有測試和結果工件的預處理和後處理。

相比之下，執行的自定義模式不會解析測試包; 這意味著沒有預處理和最小的後處理測試或結果工件。這會導致接近本地設置的總 end-to-end 執行時間。測試的執行格式與在本地計算機上運行時的格式相同。測試的結果與您在本地獲得的結果相同，並且可以在作業執行結束時下載。

2. 自定義或靈活性：執行的標準模式解析您的測試包以檢測測試的數量，然後單獨運行每個測試。請注意，不能保證測試將按照您指定的順序運行。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，沒有辦法自定義主機環境或傳遞以特定方式運行測試所需的配置文件。

   相比之下，自訂模式可讓您設定主機環境，包括安裝其他軟體、將篩選器傳遞至測試、傳遞組態檔案，以及控制測試執行設定的功能。它通過一個 yaml 文件（也稱為 testspec 文件）來實現這一點，您可以通過向其添加 shell 命令來修改該文件。這個 yaml 文件被轉換為在測試主機上執行的 shell 腳本。您可以保存多個 yaml 文件，並在計劃運行時根據您的要求動態選擇一個文件。

3. 實時視頻和日誌記錄：標準和自定義執行模式都為您提供測試的視頻和日誌。但是，在標準模式下，只有在測試完成後，您才能獲得測試的視頻和預定義日誌。

   相比之下，自定義模式為您提供了測試的視頻和客戶端日誌的實時流。此外，您還可以在測試結束時下載視頻和其他工件。

4. 棄用：以下測試類型將在 2023 年 12 月底以標準執行模式棄用：

   - 鴉片（所有語言）
   - Calabash
   - XCTest
   - UI Automation
   - UI Automator
   - 網頁測試
   - 內建總管

   一旦棄用，您將無法在標準模式下使用這些框架。您可以改為上面列出的測試類型使用自定義模式。

> **Tip**
>
> 如果您的使用案例涉及至少上述其中一個因素，我們強烈建議您切換到自訂執行模式。

遷移步驟

若要從標準模式移轉至自訂模式，請執行下列操作：

1. 登入AWS Management Console並開啟 Device Farm 主控台，網址為 https://console.aws.amazon.com/devicefarm/。
2. 選擇您的專案，然後開始新的自動化執行。
3. 上傳您的應用程序（或選擇web app），選擇您的測試框架類型，上傳測試包，然後在Choose your execution environment參數下，選擇選項Run your test in a custom environment。
4. 依預設，裝置伺服器陣列的範例測試規格檔案將會出現，供您檢視和編輯。此示例文件可以用作在自定義環境模式下嘗試測試的起始位置。然後，一旦您確認測試是否從主控台正常運作，您就可以變更任何與 Device Farm 的 API、CLI 和管道整合，以便在排程測試執行時使用此測試規格檔做為參數。有關如何將測試規格文件添加為運行的參數的信息，請testSpecArn參閱我們的 API 指南中 ScheduleRun API 的參數部分。

APPIUM 框架

在自定義測試環境中，Device Farm 不會在 Appium 框架測試中插入或覆蓋任何 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。

安卓儀器

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 中擴充自訂測試

Device Farm 自訂模式可讓您執行的不僅僅是測試套件。在本節中，您將學習如何擴展測試套件並最佳化測試。

設定 PIN 碼

某些應用程式會要求您在裝置上設定 PIN 碼。Device Farm 不支援以原生方式在裝置上設定 PIN 碼。但是，這可能需要以下警告：

- 該設備必須運行安卓 8 或更高版本。
- 測試完成後，必須移除 PIN 碼。

若要在測試中設定 PIN 碼，請使用`pre_test`和`post_test`階段來設定和移除 PIN 碼，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 碼 1234。測試套件退出後，PIN 碼將被刪除。

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

通過所需功能加快基於 Appium 的測試

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，而且不會對您要如何使用 Appium 環境做任何假設。雖然這些預設值是根據業界最佳實務建立的，但它們可能不適用於您的情況。要微調 Appium 服務器的參數，您可以在測試規範中調整默認 Appium 功能。例如，以下內容將 iOS 測試套件`true`中的`usePrebuildWDA`功能設置為以加快初始啟動時間：

```
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、引用的 JSON 結構。

以下 Appium 功能是性能改進的常見來源：

`noReset` 和 `fullReset`

這兩個功能是互斥的，描述了 Appium 在每個會話完成後的行為。設置`noReset`為時`true`，Appium 服務器不會在 Appium 會話結束時從應用程序中刪除數據，從而實際上不進行任何清理。 `fullReset`工作階段關閉後，從裝置解除安裝並清除所有應用程式資料。有關更多信息，請參閱 Appium 文檔中的重置策略。

`ignoreUnimportantViews`(僅限安卓系統)

指示 Appium 將 Android UI 層次結構僅壓縮到用於測試的相關視圖，從而加快某些元素查找速度。但是，這可能會破壞一些基於 XPath 的測試套件，因為 UI 佈局的層次結構已更改。

`skipUnlock`(僅限安卓系統)

通知 Appium 目前沒有設置 PIN 碼，這可以加快屏幕關閉事件或其他鎖定事件後的測試速度。

`webDriverAgentUrl`(僅適用於 iOS 版本)

指示 Appium 假設基本的 iOS 依賴項已經在運行`webDriverAgent`，並且可以在指定的 URL 接受 HTTP 請求。如果尚`webDriverAgent`未啟動並運行，則在測試套件開始時可能需要 Appium 一些時間才能啟動. `webDriverAgent` 如果您啟動`webDriverAgent`自己並設置`webDriverAgentUrl`為啟動 Appium `http://localhost:8100` 時，則可以更快地啟動測試套件。請注意，此功能絕對不應與該`useNewWDA`功能一起使用。

您可以使用以下代碼`webDriverAgent`從設備本地端口上的測試規格文件開始`8100`，然後將其轉發到測試主機的本地端口`8100`（這允許您將值設`webDriverAgentUrl`置為`http://localhost:8100`）。在定義了用於設置 Appium 和`webDriverAgent`環境變量的任何代碼之後，此代碼應該在安裝階段運行：

```
      # 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，您可以將以下代碼添加到測試規範文件中，以確保成功webDriverAgent啟動。在確保 Appium 成功啟動後，應在測試前階段結束時運行此代碼：

```
      # 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;
```

有關 Appium 支持的功能的更多信息，請參閱 Appium 文檔中的 Appium 所需功能。

測試運行後使用網絡掛鉤和其他 API

在每個測試套件完成使用後，您可以讓 Device Farm 調用 curl webhook。執行此操作的程序會因目標和格式而異。對於您的特定網絡掛鉤，請參閱該網絡掛鉤的文檔。下列範例會在每次測試套件完成 Slack Webhook 時張貼訊息：

```
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 使用網路掛接的詳細資訊，請參閱 Slack API 參考資料中的使用 Webhook 傳送您的第一則 Slack 訊息。

您不僅限於使用調curl用網絡掛鉤。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 API 發出要求的輔助指令碼。確保所有必需的軟件包都與測試套件的要求一起安裝。要添加在測試套件完成後運行的腳本，請在測試包中包含該腳本，並將以下內容添加到測試規範中：

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

> **Note**
>
> 維護測試包中使用的任何 API 密鑰或其他身份驗證令牌是您的責任。我們建議您將任何形式的安全憑證保留在原始檔控制之外、使用具有最少權限的認證，並儘可能使用可撤銷的短期權杖。若要驗證安全性需求，請參閱您所使用之協力廠商 API 的文件。

如果您計劃將 AWS 服務用作測試執行套件的一部分，則應使用在測試套件外部生成並包含在測試包中的 IAM 臨時登入資料。這些認證應具有最少授予的權限和最短的壽命。如需建立臨時登入資料的詳細資訊，請參閱 IAM 使用者指南中的要求臨時安全登入資料。

將額外的文件添加到您的測試包

您可能希望使用其他文件作為測試的一部分，無論是額外的配置文件或其他測試數據。您可以先將這些額外檔案新增至測試套件，然後再將其上傳至 AWS Device Farm，然後從自訂環境模式存取這些檔

案。從根本上講，所有測試包上傳格式（ZIP，IPA，APK，JAR 等）都是支持標準 ZIP 操作的包存檔格式。

您可以使用以下命令將文件添加到 AWS Device Farm 測試存檔中，然後再將其上傳到中：

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

對於額外文件的目錄：

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

除了 IPA 檔案以外的所有測試套件上傳格式，這些指令都能如預期般運作。對於 IPA 文件，尤其是與 XCuitests 一起使用時，由於 AWS Device Farm 退出 iOS 測試包的方式，我們建議您將任何額外的文件放在略有不同的位置。構建 iOS 測試時，測試應用程序目錄將位於名為####的另一個目錄內。

例如，這就是一個這樣的 iOS 測試目錄可能的外觀：

```
$ 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
```

*#### XCuitest ##################### .app ######*例如，以下命令顯示如何將文件添加到此測試包中：

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

當您將文件添加到測試包時， AWS Device Farm 根據其上傳格式，您可以期望在其中略有不同的交互行為。如果上傳使用 ZIP 文件擴展名，則會在測試之前自動解壓縮上傳，並 AWS Device Farm 將解壓縮的文件保留在具有 *$DEVIC* EAM_TEST_PACKAGE_PATH 環境變量的位置。（這意味著，如果您像第一個示例中一樣將一個名為 *Extra_file* 的文件添加到歸檔的根目錄中，它將位於測試期間將位於 *$ DeviceFarm* _TEST_PACKAGE _ 路徑/外部文件）。

要使用更實際的例子，如果您是 Appium TestNG 用戶，想要在測試中包含 *testng.xml* 文件，則可以使用以下命令將其包含在存檔中：

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

然後，您可以將自訂環境模式中的 test 命令變更為以下內容：

```
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
```

*############## ZIP ##### APK#IPA # JAR ######################*因為這些仍然是歸檔格式文件，因此您可以解壓縮文件以便從內部訪問其他文件。例如，下列指令會將測試套件的內容 (針對 APK、IPA 或 JAR 檔案) 解壓縮至*/*tmp 目錄：

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

*# APK # JAR ##################### /tmp ######/t mp/extra_file##*在 IPA 檔案的情況下，如前所述，額外的檔案會位於以 *.app* 結尾的資料夾內部稍有不同的位置，該資料夾位於 *Payload* 目錄內。*######## IPA ######### /TMP/ ####/ADFIOS UITSTS ### .####/ Extra_## (#### /TMP/ ####/*.app/ ReferenceApp extra_file)#*

在 AWS 裝置伺服器陣列中使用遠端存取

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。

裝置伺服器陣列中的工作階段是與 Web 瀏覽器中裝載的實際實體裝置的即時互動。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。

裝置伺服器陣列目前提供用於遠端存取測試的裝置子集。新的裝置始終都會新增到裝置集區。

裝置陣列會擷取每個遠端存取工作階段的視訊，並在工作階段期間產生活動記錄。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。

主題

在 AWS 裝置農場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

- 在裝置伺服器陣列中建立專案。請遵循在 AWS 裝置伺服器陣列中建立專案中的指示，然後返回此頁面。

使用裝置陣列主控台建立工作階段

1. 登入裝置伺服器陣列主控台，位於：https://console.aws.amazon.com/devicefarm。
2. 在 [裝置伺服陣列] 導覽面板上，選擇行動裝置測試，然後選擇項目。
3. 如果您已經有一個項目，請從列表中選擇它。否則，請依照中的指示建立專案在 AWS 裝置伺服器陣列中建立專案。
4. 在 Remote access (遠端存取) 標籤上，選擇 Start a new session (啟動新的工作階段)。
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。您可以依據下列項目進行搜尋：
   - 名稱
   - 平台
   - 機型
   - 艦隊類型
6. 在 Session name (工作階段名稱) 中，輸入工作階段的名稱。
7. 選擇 Confirm and start session (確認並啟動工作階段)。

後續步驟

裝置伺服器陣列會在要求的裝置可用時立即啟動工作階段，通常會在幾分鐘內完成。該要求的裝置對話方塊會出現，直到作業階段開始。若要取消工作階段請求，請選擇 Cancel request (取消請求)。

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，裝置伺服器陣列會結束工作階段。您的帳戶需要為閒置時間付費。

在工作階段開始之後，您可以在 Web 瀏覽器中與裝置互動。

在 AWS 裝置伺服器陣列中使用遠端存取會話

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

- 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。

使用裝置陣列主控台中的工作階段

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顯示在剩餘時間設備名稱附近的字段。

安裝應用程式

若要在工作階段裝置上安裝應用程式，請安裝應用，選取選擇檔案，然後選擇您要安裝的 .apk 文件（安卓）或 .ipa 文件（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**
>
> 安裝應用程式後，AWS 裝置伺服器陣列不會顯示確認訊息。嘗試與應用程式圖示互動，看看應用程式是否可以開始使用。
>
> 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。請查看系統匣，判斷應用程式是否可用。

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。您也可以選擇在裝置上執行的應用程式之間切換最近的應用。

在縱向和橫向模式之間切換

您也可以在使用的裝置之間切換縱向 (垂直) 和橫向 (水平) 模式。

後續步驟

裝置伺服器陣列會繼續工作階段，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，請參閱獲取會話結果。

提示和技巧

在某些 AWS 區域中，您可能會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 裝置伺服器陣列中取得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
- 檢視階段作業詳
- 下載工作階段影片或記錄

先決條件

- 完成工作階段。請遵循在 AWS 裝置伺服器陣列中使用遠端存取會話中的指示，然後返回此頁面。

檢視階段作業詳

遠端存取工作階段結束時，裝置伺服器陣列主控台會顯示一個表格，其中包含工作階段期間活動的詳細資料 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 [裝置伺服陣列] 導覽面板上，選擇行動裝置測試，然後選擇項目。
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記錄

當遠端存取工作階段結束時，Device Farm 主控台可讓您存取工作階段和活動記錄的視訊擷取。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。

在 AWS 裝置伺服器陣列中使用私有裝置

私有裝置是 AWS Device Farm 代表您在 Amazon 資料中心部署的實體行動裝置。此裝置僅限於您的 AWS 帳戶。

> **Note**
>
> 目前，私人裝置僅在 AWS 美國西部 (奧勒岡) 區域 (us-west-2) 提供。

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。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 裝置伺服器陣列管理私有裝置。或者，您可以要求將某些 Android 私人設備部署為 root 設備。

您也可以建立 Amazon Virtual Private Cloud 端端點服務，以測試貴公司可存取但無法透過網際網路存取的私有應用程式。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 搭配裝置伺服器陣列使用 Amazon VPC 端點服務-舊版 (不建議使用)。

如果您有興趣使用一系列私有裝置，請聯絡我們。Device Farm 團隊必須與您合作，為您的 AWS 帳戶設定和部署私人裝置叢集。

主題

在 AWS 裝置伺服器陣列管理私有裝置

私有裝置是 AWS 裝置農場代表您在 Amazon 資料中心部署的實體行動裝置。這個裝置專屬於您的 AWS 帳戶。

> **Note**
>
> 目前，私人裝置可在AWS美國西部 (奧勒岡) 區域 (us-west-2) 只。

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。

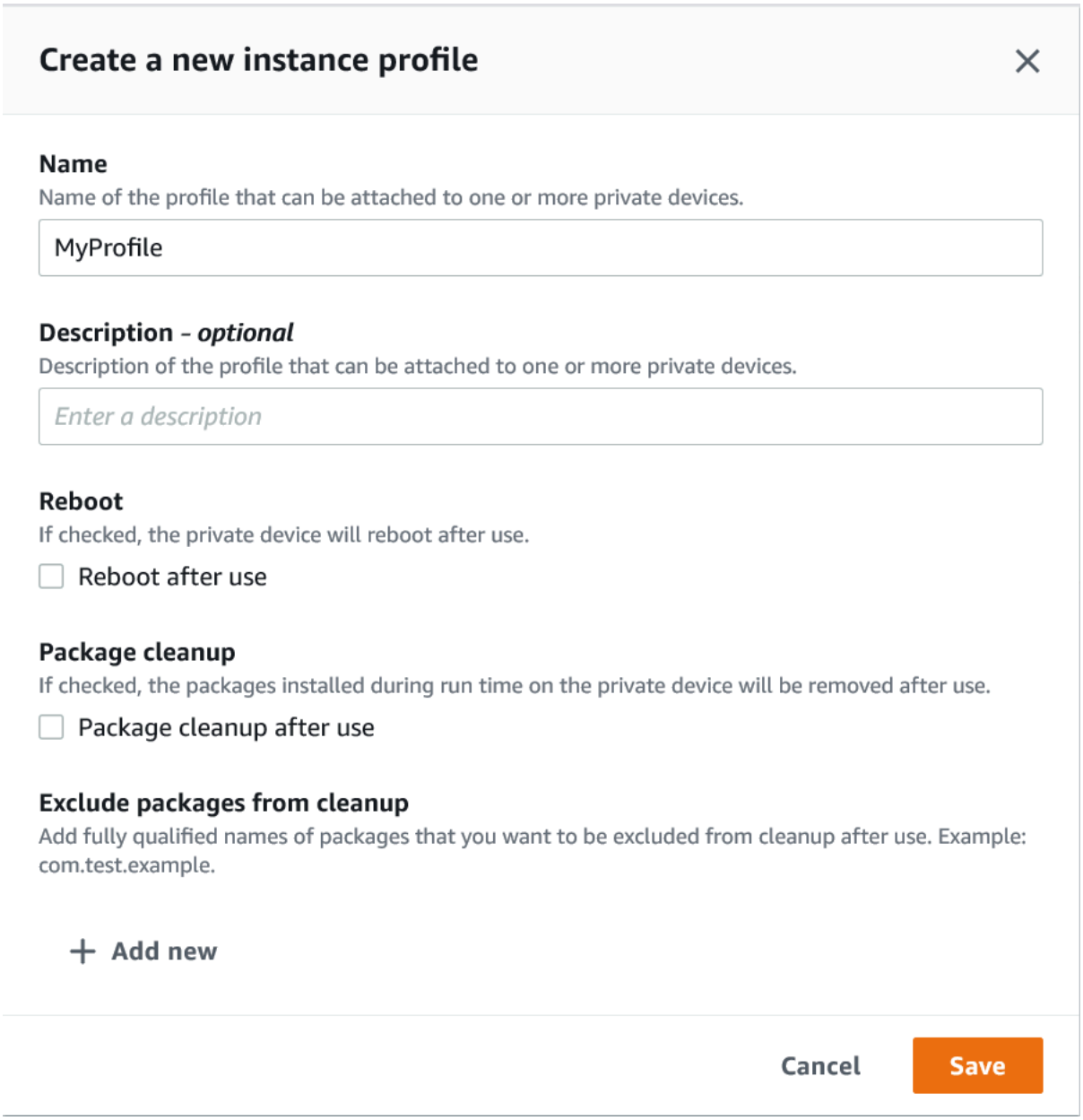
此主題說明如何建立執行個體描述檔，並執行其他常用的裝置管理任務。

主題

建立執行個體描述檔

若要控制私人裝置在測試執行或遠端存取工作階段期間的行為，您可以在裝置伺服器陣列中建立或修改執行個體設定檔。您不需要執行個體描述檔，即可開始使用您的私有裝置。

1. 開啟裝置伺服器陣列主控台，https://console.aws.amazon.com/devicefarm/。
2. 在 [裝置伺服陣列] 導覽面板上，選擇行動裝置測試，然後選擇私人裝置。
3. 選擇 Instance profiles (啟動設定檔)。
4. 選擇建立例項設定檔。
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. （可選）變更下列任何設定，以指定您希望裝置伺服器陣列在每次測試執行或工作階段結束後對裝置執行的動作：

   - 使用後重新開機— 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`)。
   - 軟件包清理— 要刪除設備上安裝的所有應用程序包，請選擇此複選框。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。

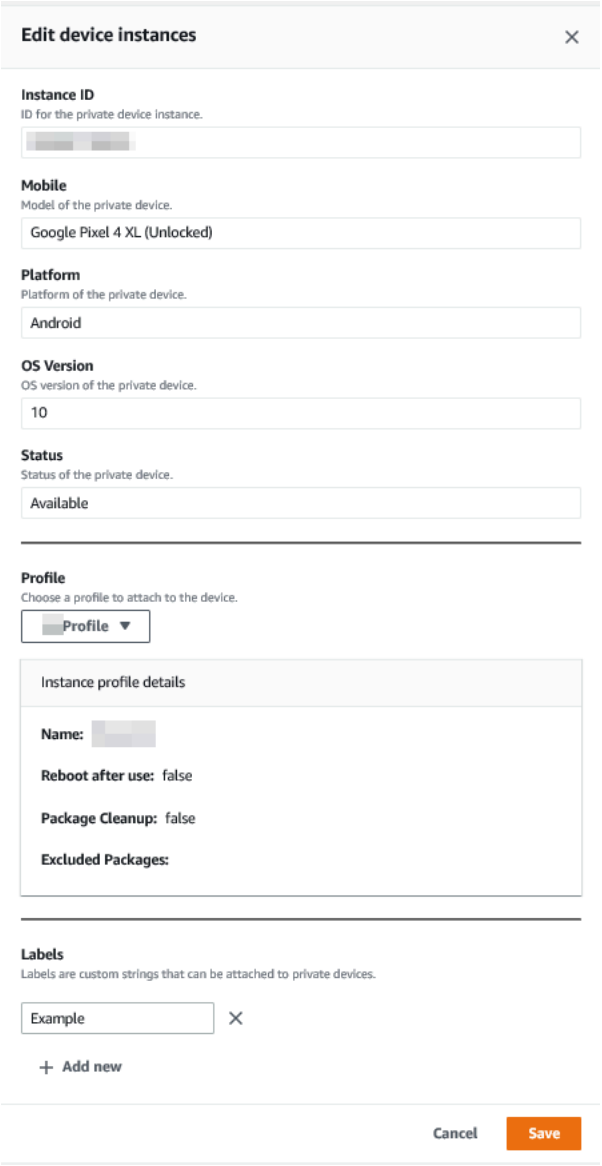
- 從清除中排除套件— 要僅在設備上保留選定的應用程序包，請選擇套件清理」核取方塊，然後選擇添加新的。對於套件名稱，輸入您要保留在裝置上之應用程式套件的完整名稱 (例如，`com.test.example`)。若要在裝置上保留更多應用程式套件，請選擇 Add new (新增)，然後輸入每個套件的完整名稱。

8. 選擇 儲存 。

管理私有裝置執行個體

如果在您的機群中已經有一或多個私有裝置，您可以檢視每個裝置執行個體的相關資訊並管理特定設定。您也可以請求額外的私有裝置執行個體。

1. 開啟裝置伺服器陣列主控台，https://console.aws.amazon.com/devicefarm/。
2. 在 [裝置伺服陣列] 導覽面板上，選擇行動裝置測試，然後選擇私人裝置。
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表格，請在欄上方的搜尋列中輸入搜尋字詞。
4. (選擇性) 若要要求新的私人裝置執行個體，請選擇請求設備實例或者聯絡我們。私人裝置需要在裝置伺服器陣列團隊的協助下進行其他設定。
5. 在裝置執行個體表格中，選擇您要檢視或管理相關資訊的執行個體旁的切換選項，然後選擇編輯。

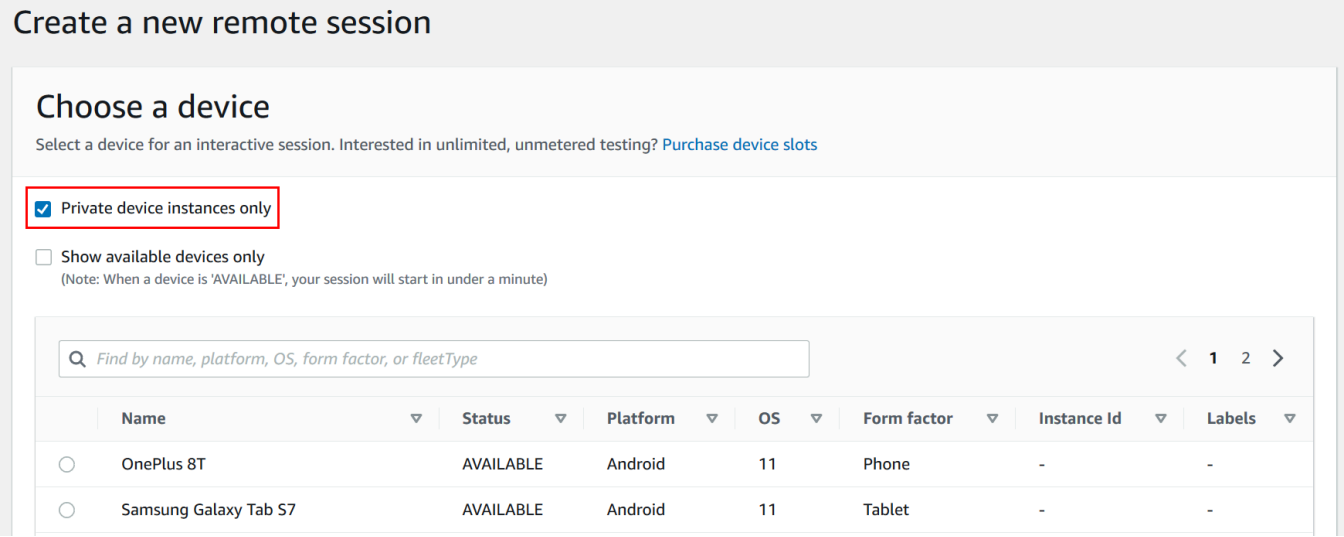


6. (選用) 對於 Profile (描述檔)，請選擇一個執行個體描述檔來連接至裝置執行個體。如果您想要一律從清除任務 (舉例而言) 排除特定應用程式套件，則這將非常有用。
7. (選用) 在 Labels (標籤) 中，選擇 Add new) 來新增標籤至裝置執行個體。標籤可協助您分類裝置並更輕鬆地找到特定裝置。
8. 選擇 儲存 。

創建測試運行或啟動遠程訪問會話

設定私有裝置叢機群之後，您可以使用一或多個私有裝置，在您的機群中建立測試執行或啟動遠端存取工作階段。

1. 開啟裝置伺服器陣列主控台，https://console.aws.amazon.com/devicefarm/。
2. 在 [裝置伺服陣列] 導覽面板上，選擇行動裝置測試，然後選擇項目。
3. 從清單中選擇現有專案或建立新專案。若要建立新專案，請選擇新項目，輸入專案的名稱，然後選擇提交。
4. 執行下列任意一項：
   - 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。對於選擇裝置步驟中，您可以編輯現有的裝置集區或建立新的裝置集區，其中僅包含 Device Farm 團隊設定並與您關聯的私人裝置AWS帳戶。如需詳細資訊，請參閱the section called “建立私人裝置集區”。
   - 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，選取僅限私有設備實例，將清單限制為裝置伺服器陣列小組設定並與您關聯的私人裝置AWS帳戶。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)。



後續步驟

設定您的私有裝置後，您也可以以下列方式管理您的私有裝置：

- 在私有裝置上略過應用程式重新簽署
- 搭配裝置伺服器陣列使用 Amazon 虛擬私有雲端點服務

若要刪除執行個體設定檔，請在例項設定檔」選單中，選擇您要刪除之執行個體旁的切換選項，然後選擇刪除。

選取裝置集區中的私人裝置

要在測試運行中使用私人設備，您可以創建一個選擇私有設備的設備池。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人裝置：

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. 以裝置實例 ARN 為基礎的規則

在以下各節中，將深入描述每個規則類型及其使用案例。您可以使用裝置伺服器陣列主控台，AWS指令行介面 (AWSCLI)，或使用這些規則建立或修改具有私人裝置的裝置集區的裝置伺服器陣列 API。

主題

設備 ARN

設備 ARN 是代表設備類型而不是任何特定物理設備實例的標識符。裝置類型由下列屬性定義：

- 裝置的叢集 ID
- 該設備的 OEM
- 設備的型號
- 裝置的作業系統版本
- 指示裝置是否已植根的狀態

許多實體裝置執行個體可以透過單一裝置類型來表示，其中該類型的每個執行個體都具有相同的這些屬性值。例如，如果你有三個*##*iOS 版本上的設備*16.1.0*在您的私人機群中，每個裝置都會共用相同的裝置 ARN。如果使用這些相同屬性從叢集新增或移除任何裝置，裝置 ARN 將繼續代表您在叢集中針對該裝置類型的任何可用裝置。

裝置 ARN 是為裝置集區選取私人裝置的最可靠方式，因為無論您在任何指定時間部署的特定裝置執行個體為何，裝置集區都能繼續選取裝置。個別私有裝置執行個體可能會遇到硬體故障，並提示 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硬體故障時繼續選取裝置。

當您針對裝置集區中的私人裝置使用裝置 ARN 規則，並使用該集區排程測試執行時，Device Farm 會自動檢查該裝置 ARN 代表哪些私人裝置執行個體。當前可用的實例中，其中之一將被分配運行您的測試。如果目前沒有可用的執行個體，裝置伺服器陣列會等待該裝置 ARN 的第一個可用執行個體可用，並將其指派給執行測試。

裝置實例標籤

設備實例標籤是文本標識符，您可以將其作為設備實例的元數據附加。您可以將多個標籤附加到每個設備實例，並將相同的標籤附加到多個設備實例。如需有關從裝置執行個體新增、修改或移除裝置標籤的詳細資訊，請參閱管理私人裝置。

設備實例標籤可以是為設備池選擇私有設備的一種可靠方法，因為如果您有多個具有相同標籤的設備實例，則它允許設備池中的任何一個進行測試。如果裝置 ARN 不是適合您使用案例的規則 (例如，如果您想要從多種裝置類型的裝置中進行選取，或想要從某種裝置類型的所有裝置的子集中選取)，則裝置執行個體標籤可讓您從多個裝置中進行選取，具有更大的細微度。個別私有裝置執行個體可能會遇到硬體故障，並提示 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不會保留已取代裝置的任何執行個體標籤中繼資料。因此，如果將相同的設備實例標籤應用於多個設備實例，則設備實例標籤規則可確保在硬件故障時，設備池可以繼續選擇設備實例。

當您對設備池中的私有設備使用設備實例標籤規則並使用該池安排測試運行時，Device FARM 將自動檢查該設備實例標籤表示的私有設備實例，以及這些實例中，隨機選擇一個可用於運行測試的實例。如果沒有可用，則 Device Farm 將隨機選擇具有設備實例標籤的任何設備實例以運行測試，並在可用時將測試排入隊列以在設備上運行。

執行個體 ARN

裝置執行個體 ARN 是代表私人叢集中部署的實體裸機裝置執行個體的識別碼。例如，如果你有三個*##*OS 上的裝置*15.0.0*在您的私有機隊中，雖然每個設備都會共享相同的設備 ARN，但每個設備也將具有自己的實例 ARN 代表該實例。

設備實例 ARN 是為設備池選擇私有設備的最不可靠方法，只有在設備 ARN 和設備實例標籤不適合您的用例時才建議使用。當以獨特且特定的方式配置特定的設備實例作為測試的先決條件，以及在對其運行測試之前是否需要了解和驗證該配置，則設備實例 ARN 通常用作設備池的規則。個別私有裝置執行個體可能會遇到硬體故障，並提示 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，替換設備實例將具有與替換設備不同的設備實例 ARN。因此，如果您依賴裝置集區的裝置執行個體 ARN，則需要手動將裝置集區的規則定義從使用舊 ARN 變更為使用新的 ARN。如果您確實需要手動預先配置設備進行測試，則這可能是一個有效的工作流程（與設備 ARN 相比）。對於大規模測試，建議嘗試調整這些用例以使用設備實例標籤，並在可能的情況下預先配置多個設備實例以進行測試。

當您對設備池中的私人設備使用設備實例 ARN 規則並使用該池安排測試運行時，Device Farm 將自動將該測試分配給該設備實例。如果該設備實例不可用，則 Device Farm 將在可用時將測試排入隊列。

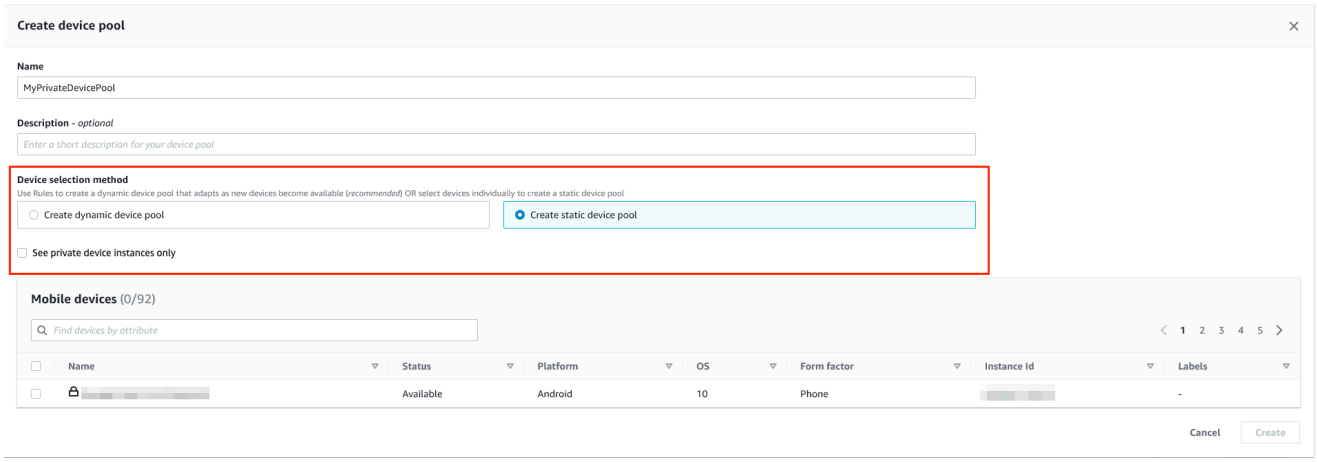
使用私人裝置建立私人裝置集區 (主控台)

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。

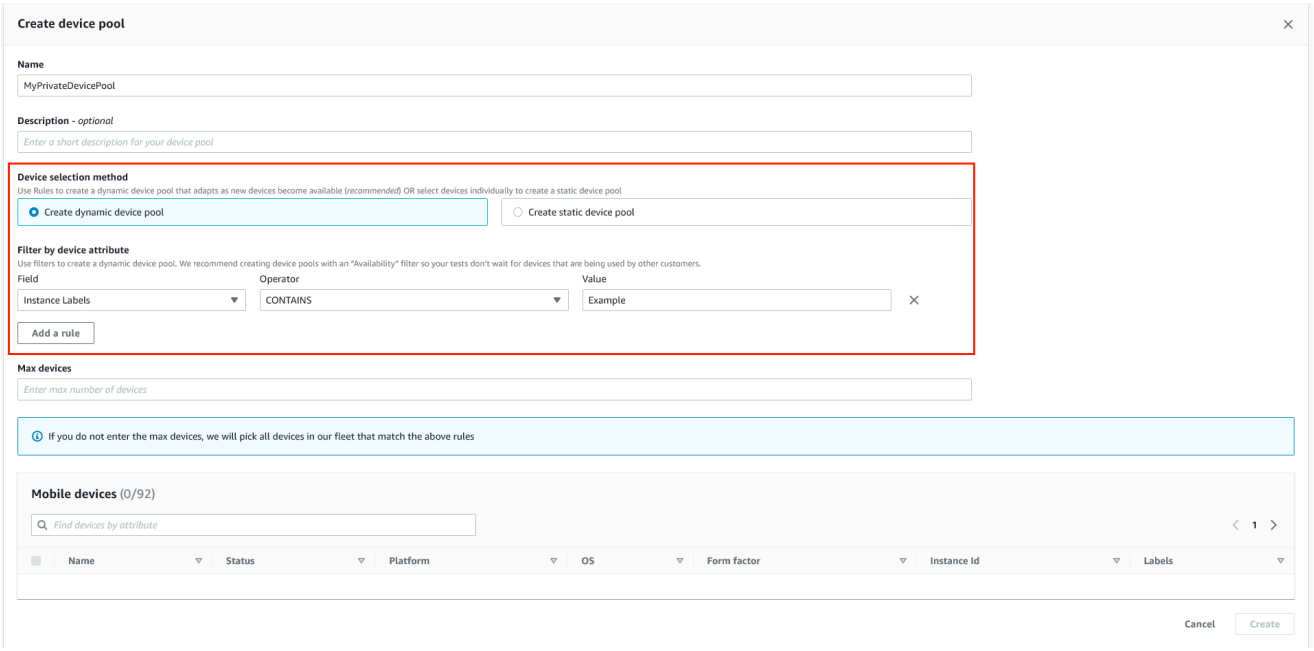
> **Note**
>
> 在主控台中使用私人裝置建立裝置集區時，您只能使用三個可用規則中的任何一個來選取私人裝置。如果您想要建立包含針對私人裝置的多種規則類型的裝置集區 (例如，包含裝置 ARN 和裝置執行個體 ARN 規則的裝置集區)，則需要透過 CLI 或 API 建立集區。

1. 開啟裝置伺服器陣列主控台，https://console.aws.amazon.com/devicefarm/。
2. 在 [裝置伺服陣列] 導覽面板上，選擇行動裝置測試，然後選擇項目。
3. 從清單中選擇現有專案或建立新專案。若要建立新專案，請選擇新專案，輸入專案的名稱，然後選擇提交。
4. 選擇 Automated tests (自動測試)，然後選擇 Create a new run (建立新執行)。精靈會逐步引導您選擇您的應用程式和設定要執行的測試的步驟。
5. 對於選擇裝置步驟，選擇建立裝置集區，然後輸入裝置集區的名稱和選用說明。

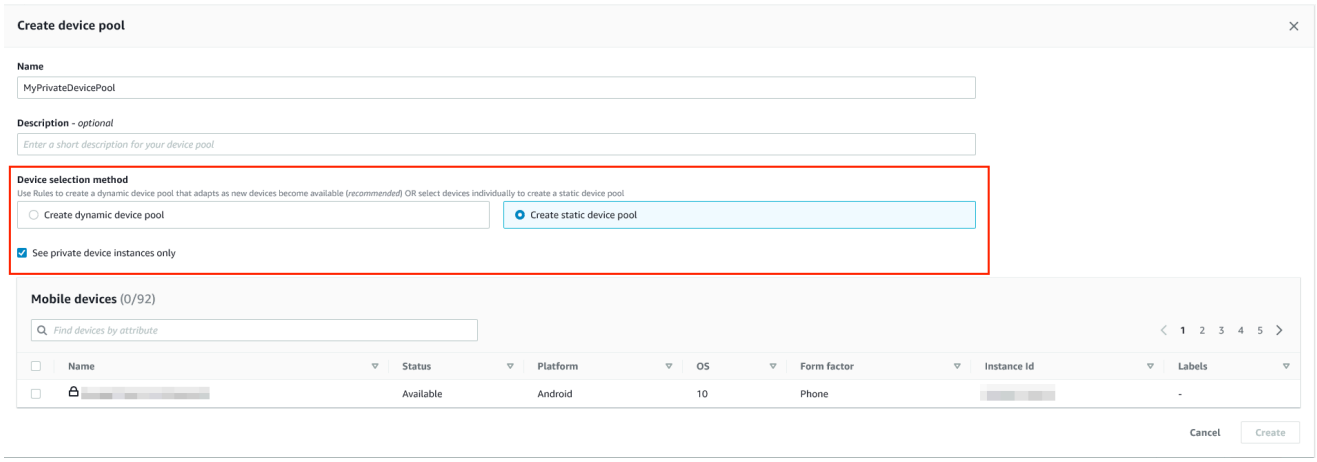
    a. 若要為您的裝置集區使用裝置 ARN 規則，請選擇建立靜態裝置集區，然後從清單中選取您要在裝置集區中使用的特定裝置類型。不要選擇僅限私有設備實例因為此選項會使用設備實例 ARN 規則（而不是設備 ARN 規則）創建設備池。



b. 要為您的設備池使用設備實例標籤規則，請選擇建立動態裝置集區。然後，對於您要在設備池中使用的每個標籤，選擇新增規則。針對每個規則，選擇實例標籤作為`Field`，選擇包含作為`Operator`，並將所需的設備實例標籤指定為`Value`。



c. 要為您的設備池使用設備實例 ARN 規則，請選擇建立靜態裝置集區」，然後選取僅限私有設備實例將設備列表限制為僅設備場與您關聯的私有設備實例AWS帳戶。



6. 選擇 建立 。

使用私人裝置建立私人裝置集區 (AWS CLI)

- 執行 create-device-pool 命令。

如需使用裝置伺服器陣列搭配AWS CLI，請參閱AWS CLI 參考。

使用私人裝置 (API) 建立私人裝置集區

- 呼叫 `CreateDevicePool` API。

如需使用裝置伺服器陣列 API 的相關資訊，請參閱自動化裝置農場。

略過 AWS Device Farm 中私有裝置上的應用程式重新簽署

應用程序簽名是一個過程，涉及使用私鑰對應用程序包（例如 APK，IPA）進行數字簽名，然後才能將其安裝在設備上或發佈到 Google Play 商店或 Apple App Store 等應用程序商店。為了透過減少所需的簽名和設定檔數量來簡化測試，並提高遠端裝置的資料安全性，AWS Device Farm 會在您的應用程式上傳到服務後重新簽署應用程式。

將應用程式上傳到 AWS Device Farm 後，服務會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。此程序會以 AWS 裝置伺服器陣列的簽章取代原始應用程式簽章。然後，重新簽署的應用程式會安裝在 AWS 裝置伺服器陣列提供的測試裝置上。新的簽名允許在這些設備上安裝和運行該應用程序，而無需原始開發人員的證書。

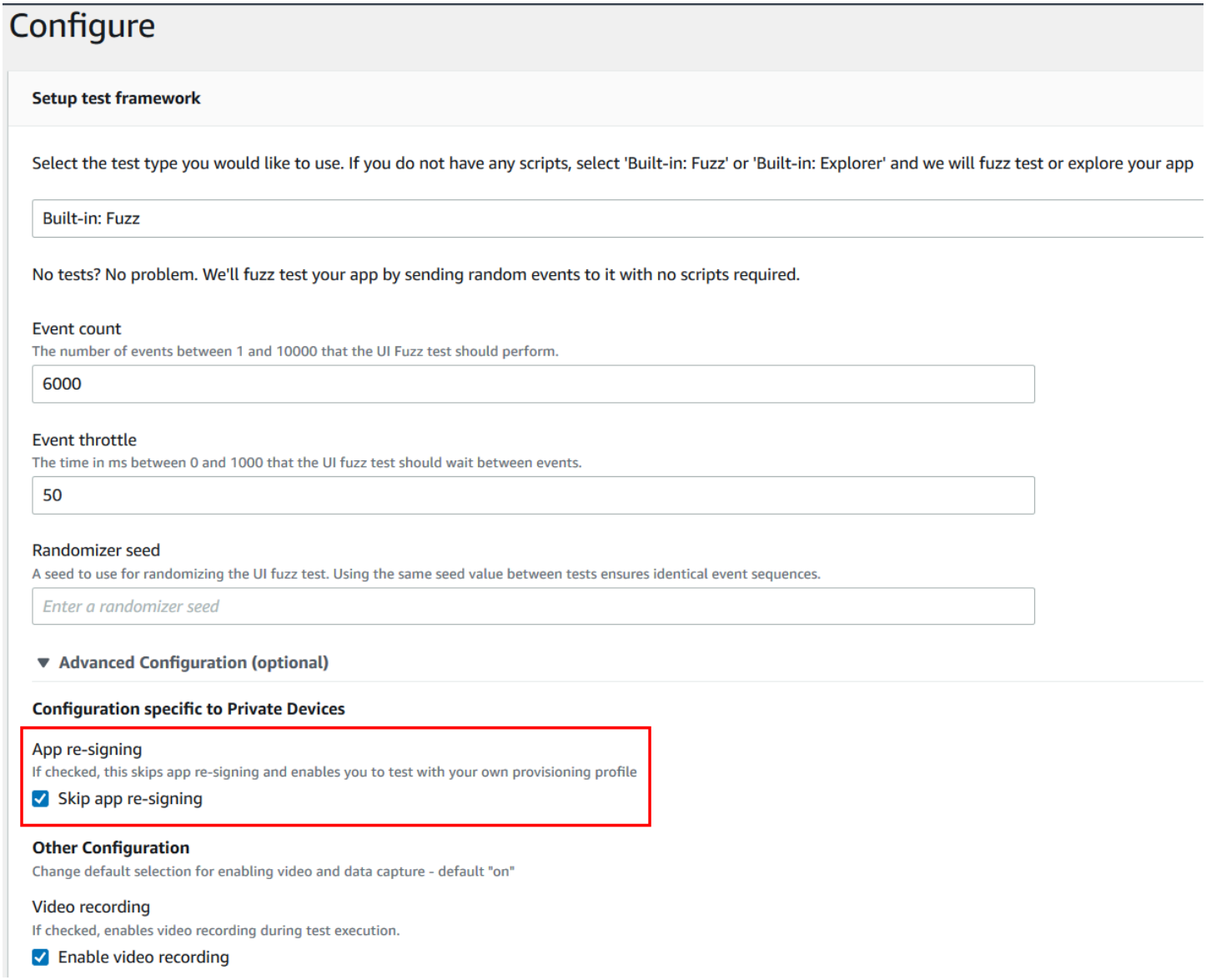
在 iOS 上，我們將內嵌的佈建描述檔取代為萬用字元設定檔，然後辭去應用程式。如果您提供它，我們將在安裝之前將輔助數據添加到應用程序包中，以便數據將出現在應用程序的沙箱中。退出 iOS 應用程序會導致刪除某些權利。這包括應用程式群組、關聯網域、遊戲中心、 HealthKit、 HomeKit、無線配件設定、應用程式內購買、應用程式間音訊、Apple Pay、推送通知，以及 VPN 設定與控制。

在安卓系統上，我們會辭去應用程式。這可能會破壞取決於應用程序簽名的功能，例如谷歌地圖 Android API。它還可能會觸發從諸如產品提供的反盜版和防篡改檢測。 DexGuard對於內置測試，我們可能會修改清單以包含捕獲和保存屏幕截圖所需的權限。

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。這與公共設備不同，其中 Device Farm 始終在 Android 和 iOS 平台上重新簽署您的應用程序。

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。如果您的應用程序具有在 Device Farm 重新簽名您的應用程序時中斷的功能，這將很有幫助。例如，在重新簽署之後，推送通知可能無法運作。如需 Device Farm 在測試應用程式時所做變更的詳細資訊，請參閱 AWS Device Farm 常見問答集或應用程式頁面。

要跳過應用程序重新簽名以進行測試運行，請在創建測試運行時在「配置」頁面上選擇「跳過應用程序重新簽名」。



> **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 使用適用於 iOS 和 AWS Device Farm 的 XCTest。

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 設備上跳過應用程序重新簽名

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 設備上跳過應用重新簽名

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 “創建遠程訪問會話以信任您的應用”。

- 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 中運行重新簽名的應用程序。

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，請使用裝 Device Farm API 的`ListDeviceInstances`作業。
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)。

創建遠程訪問會話以信任您的 iOS 應用

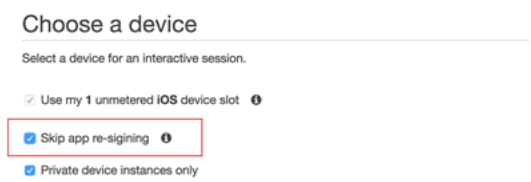
如果您是使用內部 (Enterprise) 開發人員佈建描述檔，則必須執行一次性程序，來信任每個私有裝置上的內部應用程式開發人員憑證。

若要這樣做，您可以在私有裝置上安裝您想要測試的應用程式，或是可以安裝虛擬應用程式，而其簽署使用的憑證與您想要測試的應用程式憑證相同。這是安裝使用相同憑證簽署的虛擬應用程式的一個優點。在您信任組態描述檔或企業應用程式開發人員之後，私有裝置會信任來自該開發人員的所有應用程式，直到您將其刪除為止。因此，當上傳您要測試之應用程式的新版本時，您不必重新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。

開始遠端存取工作階段之前，請按照中的步驟在建立執行個體描述檔裝置伺服器陣列中建立或修改執行個體設定檔。在實例配置文件中，將測試應用程序或虛擬應用程序的捆綁包 ID 添加到「從清理中排除包」 設置。然後，將實例配置文件附加到私有設備實例，以確保 Device Farm 在開始新的測試運行之前不會從設備中刪除此應用程序。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虛擬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。

   請務必同時將虛擬應用程式或您要測試的應用程式新增至附加至此執行個體的執行個體設定檔的 [從清理排除套件] 設定中。

2. 當遠端工作階段啟動時，請選擇 [選擇檔案] 以安裝使用內部佈建描述檔的應用程式。
3. 啟動您剛上傳的應用程式。
4. 按照指示來信任開發人員憑證。

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。

與亞馬遜 VPC 跨越工作AWS地區

裝置伺服器陣列服務僅位於美國西部 (奧勒岡) (us-west-2) 地區。您可以使用亞馬遜虛擬私有雲（亞馬遜 VPC）在另一個亞馬遜虛擬私有雲中達到服務AWS使用裝置陣列的區域。如果裝置伺服器陣列和您的服務位於相同的區域，請參閱搭配裝置伺服器陣列使用 Amazon VPC 端點服務-舊版 (不建議使用)。

有兩種方式可以訪問位於不同地區的私人服務。如果您的服務位於另一個區域，但不是us-west-2，您可以使用 VPC 對等，將該區域的 VPC 對等至與裝置伺服器陣列連接的另一個 VPCus-west-2。不過，如果您在多個區域提供服務，Transit Gateway 將允許您使用更簡單的網路設定來存取這些服務。

如需詳細資訊，請參閱VPC 對等互連案例在亞馬遜 VPC 對等互連指南。

VPC 對等互連

您可以對等不同區域中的任何兩個 VPC，只要它們具有不同、非重疊的 CIDR 區塊即可。這樣可確保所有私有 IP 位址都是唯一的，並允許 VPC 中的所有資源互相解決，而不需要任何形式的網路位址轉譯 (NAT)。如需 CIDR 表示法的詳細資訊，請參閱 RFC 4632。

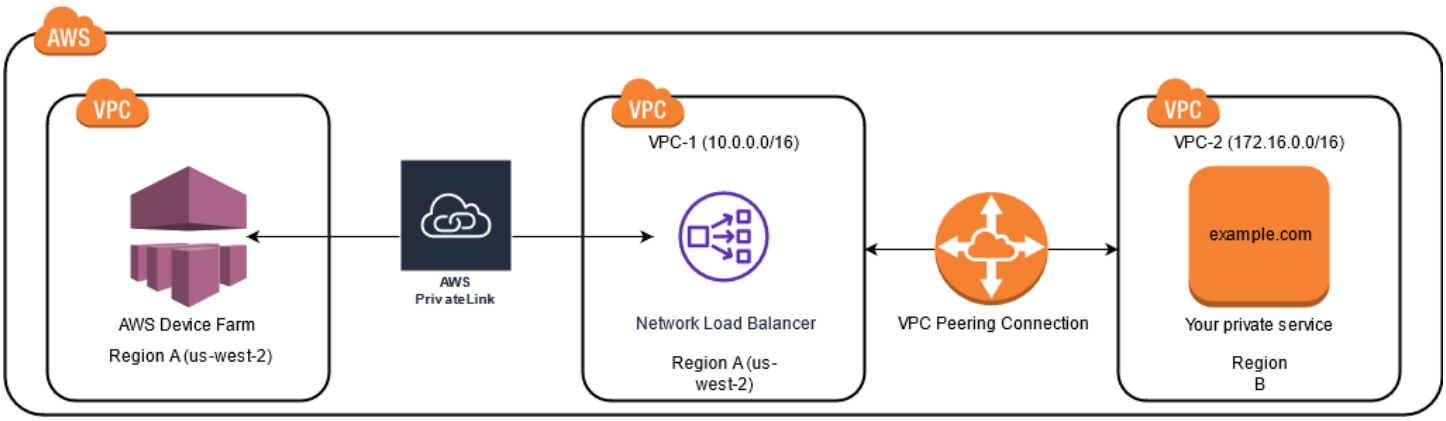
本主題包含跨區域範例案例，其中裝置伺服器陣列 (稱為VPC-1) 位於美國西部 (奧勒岡) (us-west-2) 地區。此範例中的第二個 VPC (稱為VPC-2) 位於另一個區域。

裝置伺服器陣列 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 VPC 之間建立對等連線可能會變更 VPC 的安全性狀態。此外，將新項目新增至其路由表格可變更 VPC 內資源的安全狀況。您有責任以符合組織安全性需求的方式來實作這些組態。如需詳細資訊，請參閱共同責任模式。

下圖顯示範例中的元件，以及這些元件之間的互動。



主題

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 VPC。
- VPC-1必須在`us-west-2`區域和包含可用區域的子網路`us-west-2a`,`us-west-2b`，以及`us-west-2c`。

如需建立 VPC 和設定子網路的詳細資訊，請參閱使用 VPC 和子網路在亞馬遜 VPC 對等互連指南。

步驟一：在 VPC-1 和 VPC-2 之間設定對等連線

在包含非重疊 CIDR 區塊的兩個 VPC 之間建立對等連線。要做到這一點，請參閱建立並接受 VPC 對等連線在亞馬遜 VPC 對等互連指南。使用本主題的跨區域案例和亞馬遜 VPC 對等互連指南時，會建立下列對等連線組態範例：

名稱

```
Device-Farm-Peering-Connection-1
```

VPC 識別碼 (請求者)

```
vpc-0987654321gfedcba (VPC-2)
```

帳戶

```
My account
```

區域

```
US West (Oregon) (us-west-2)
```

虛擬私人雲端識別碼 (接受程式)

```
vpc-1234567890abcdefg (VPC-1)
```

> **Note**
>
> 建立任何新的對等連線時，請務必查閱 VPC 對等連線配額。如需詳細資訊，請參閱亞馬遜 VPC 配額在亞馬遜 VPC 對等互連指南。

第二步：在 VPC-1 和 VPC-2 中更新路由表

設定對等連線後，您必須在兩個 VPC 之間建立目標路由，才能在這兩個 VPC 之間傳輸資料。要建立此路由，您可以手動更新路由表VPC-1以指向的子網路VPC-2反之亦然。要做到這一點，請參閱更新 VPC 對等連線的路由表在亞馬遜 VPC 對等互連指南。使用本主題的跨區域案例和亞馬遜 VPC 對等互連指南，下列範例會建立路由表組態：

裝置伺服器陣列 VPC 路由表範例

VPC 元件	VPC-1	VPC-2
路由表 ID	rtb-1234567890 安全定義	RTB-0987654321 服务中心
本機位址範圍	10.0.0.0/16	172.16.0.0/16
目的地位址範圍	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，您可以在中設定網路負載平衡器VPC-1將請求路由到VPC-2。

網路負載平衡器必須先包含目標群組，其中包含要求傳送至的 IP 位址。

若要建立目標群組

1. 識別您要鎖定目標之服務的 IP 位址VPC-2。
   - 這些 IP 位址必須是對等連線中使用之子網路的成員。
   - 目標 IP 位址必須是靜態且不可變的。如果您的服務具有動態 IP 地址，請考慮定位靜態資源（例如網絡負載平衡器），並將該靜態資源路由請求到您的真實目標。

   > **Note**
   >
   > - 如果您的目標是一個或多個獨立的亞馬遜彈性運算雲端 (Amazon EC2) 執行個體，請在以下位置開啟 Amazon EC2 主控台：https://console.aws.amazon.com/ec2/，然後選擇實例。
   > - 如果您的目標是 Amazon EC2 自動擴展執行個體群組，則必須將 Amazon EC2 自動擴展群組與網路負載平衡器建立關聯。如需詳細資訊，請參閱將負載平衡器連接至自動調整資源調整群組在亞馬遜 EC2 自動擴展用戶指南。
   >
   >   然後，您可以在以下位置打開亞馬遜 EC2 控制台https://console.aws.amazon.com/ec2/，然後選擇網路介面。從那裡，您可以檢視每個網路負載平衡器網路介面中每個網路介面的 IP 位址可用區域。

2. 在中建立目標群組VPC-1。要做到這一點，請參閱為您的網路負載平衡器建立目標群組在網路負載平衡器使用者指南。

不同 VPC 中服務的目標群組需要下列組態：

- 對於選擇目標類型，選擇IP 位址。
- 對於VPC」中，選擇將託管負載平衡器的 VPC。對於主題示例，這將是VPC-1。
- 在「」註冊目標頁面上，為每個 IP 位址註冊一個目標VPC-2。

  對於網絡，選擇其他私有 IP 位址。

  對於可用區域，選擇您想要的區域VPC-1。

  對於地址，選擇VPC-2IP 位址。

  對於連接埠，選擇您的連接埠。
- 選擇包括下面的待處理。完成指定地址後，請選擇註冊擱置目標。

使用本主題的跨區域案例和網路負載平衡器使用者指南，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
```

地址

```
172.16.100.60
```

連接埠

80

步驟 4：建立網路負載平衡器

使用中所述的目標群組建立網路負載平衡器步驟三。要做到這一點，請參閱建立網路負載平衡器。

使用本主題的跨區域案例時，網路負載平衡器組態範例會使用下列值：

Load balancer name (負載平衡器名稱)

my-nlb

計劃

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

您可以使用網路負載平衡器建立 VPC 端點服務。透過此 VPC 端點服務，裝置伺服器陣列可以連線到中的服務VPC-2不需要任何額外的基礎架構，例如網際網路閘道、NAT 執行個體或 VPN 連線。

要做到這一點，請參閱建立亞馬遜 VPC 端點服務。

步驟 6：在裝置伺服器陣列中建立 VPC 端點組態

現在，您可以在 VPC 和裝置伺服器陣列之間建立私人連線。您可以使用裝置伺服器陣列來測試私人服務，而不會透過公用網際網路公開它們。要做到這一點，請參閱在裝置伺服器陣列中建立 VPC 端點組態。

根據本主題的跨區域案例，VPC 端點組態範例會使用下列值：

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：建立測試回合

您可以使用中所述的 VPC 端點配置來建立測試回合步驟六。如需詳細資訊，請參閱在 Device Farm 中建立測試回合或建立工作階段。

使用傳輸閘道建立可擴充的網路

若要使用兩個以上的 VPC 建立可擴充的網路，您可以使用 Transit Gateway 做為網路傳輸中樞，將 VPC 與內部部署網路互連。若要在與裝置伺服器陣列相同的區域中設定 VPC 以使用傳輸閘道，您可以遵循搭配裝置伺服器陣列的 Amazon VPC 端點服務指導根據其私有 IP 地址定位另一個地區的資源。

如需傳輸閘道的詳細資訊，請參閱什麼是交通閘道？在亞馬遜 VPC 交通閘道指南。

終止私人裝置

> **Important**
>
> 這些指示僅適用於終止私人裝置合約。對於所有其他 AWS 服務和帳單問題，請參閱這些產品的相應文件或聯絡 AWS 支援部門。

若要在您最初同意的期限後終止私人裝置，您必須透過我們的電子郵件提供 30 天的不續約通知<#aws-devicefarm-support@amazon.com>。

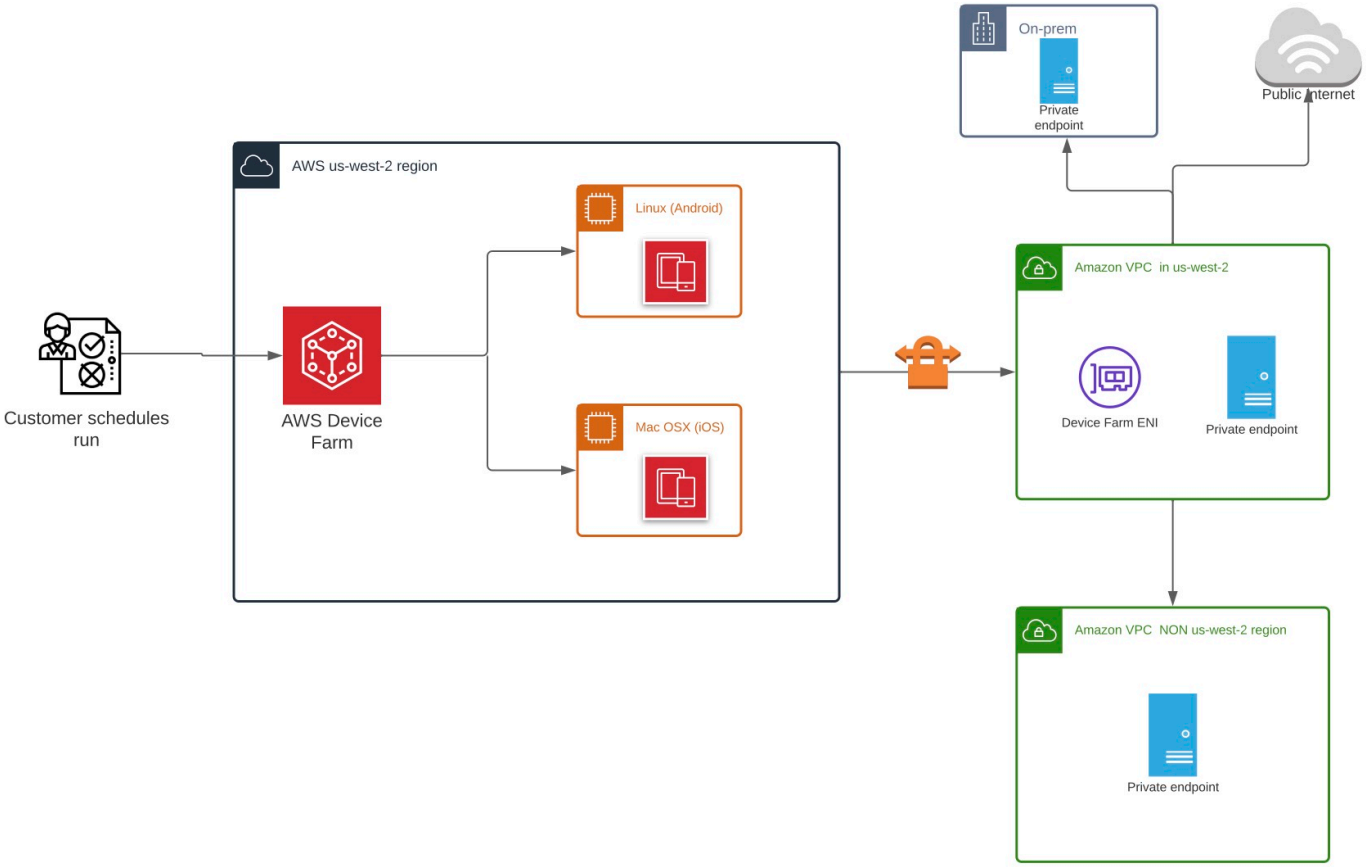
AWS Device Farm 中的 VPC-ENI

> **Warning**
>
> 此功能僅適用於私人裝置。要申請在您的 AWS 帳戶上使用私人設備，請與我們聯繫。如果您已將私人裝置新增至您的 AWS 帳戶，我們強烈建議您使用這種 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到託管在內部部署軟體或其他雲端供應商上 AWS的私有端點。

您可以將 Device Farm 行動裝置及其主機器連接到us-west-2區域中的 Amazon Virtual Private Cloud (Amazon VPC) 環境，以便透過 elastic network interface 存取隔離、 non-internet-facing 服務和應用程式。如需 VPC 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 不在該us-west-2地區，您可以使用諸如 Transit Gateway 或 VPC 對等解決方案將其與該us-west-2地區中的 V PC 連結。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 VPC 的資訊，請參閱的範本存放庫中的虛擬私人雲端連線範本 AWS CloudFormation 。 GitHub

> **Note**
>
> 在客戶的 VPC 中建立 ENI 時，Device Farm 不會收取任何費用。`us-west-2`此功能不包括跨區域或外部 VPC 間連線的成本。

設定 VPC 存取後，除非您在 VPC 中指定了 NAT 閘道，否則用於測試的裝置和主機將無法連線至 VPC 以外的資源 (例如公用 CDN)。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，授予或限制裝置伺服器陣列功能的存取權。若要將 VPC 連線功能與 AWS Device Farm 列搭配使用，您用來存取 AWS 裝 Device Farm 列的使用者帳戶或角色需要下列 IAM 政策：

```
{
  "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 列專案，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 政策還必須允許建立服務連結角色：

```
"iam:CreateServiceLinkedRole"
```

> **Note**
>
> 對於未在其專案中使用 VPC 組態的使用者，則不需要任何這些權限。

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。服務連結角色由 Device Farm 預先定義，並包含服務代表您呼叫其他服 AWS 務所需的所有權限。

服務連結角色可讓您更輕鬆地設定 Device Farm，因為您不需要手動新增必要的權限。Device Farm 會定義其服務連結角色的權限，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。這樣可以保護您的 Device Farm 資源，因為您無法意外移除存取資源的權限。

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。

裝置伺服陣列的服務連結角色權限

Device Farm 使用名為的服務連結角色 AWSServiceRoleForDeviceFarm— 允許 Device Farm 代表您存取 AWS 資源。

服 AWSServiceRoleForDeviceFarm 務連結角色會信任下列服務擔任該角色：

- `devicefarm.amazonaws.com`

角色權限原則允許 Device Farm 列完成下列動作：

- 對於您的帳戶
  - 建立網路介面
  - 描述網路介面
  - 描述 VPC
  - 描述子網路
  - 描述安全群組
  - 刪除介面
  - 修改網路介面
- 適用於網路介面
  - 建立標籤
- 對於 Device Farm 管理的 EC2 網路界面
  - 建立網路介面權限

完整的 IAM 政策內容如下：

```
{
"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。

為裝置伺服器陣列建立服務連結角色

當您為行動測試專案提供 VPC 設定時，不需要手動建立服務連結角色。當您在 AWS Management Console、或 AWS API 中建立第一個 Device Farm 資源時 AWS CLI，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，裝置伺服器陣列會再次為您建立服務連結角色。

您也可以使用 IAM 主控台，透過裝置伺服器陣列使用案例建立服務連結角色。在 AWS CLI 或 AWS API 中，使用`devicefarm.amazonaws.com`服務名稱建立服務連結角色。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。

編輯裝置伺服器陣列的服務連結角色

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。如需更多資訊，請參閱 IAM 使用者指南中的編輯服務連結角色。

刪除裝置伺服器陣列的服務連結角色

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**
>
> 當您嘗試刪除資源時，如果裝置伺服器陣列服務正在使用此角色，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 主控台或 AWS API 刪除 AWSServiceRoleForDeviceFarm 服務連結角色。 AWS CLI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。

裝置伺服陣列服務連結角色的支援區域

裝置伺服器陣列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 不支援在每個提供服務的區域中使用服務連結角色。您可以在下列區域中使用此 AWSServiceRoleForDeviceFarm 角色。

區域名稱	區域身分	Device Farm 中的 Support
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
亞太區域 (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
亞太區域 (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
歐洲 (巴黎)	eu-west-3	否
南美洲 (聖保羅)	sa-east-1	否
AWS GovCloud (US)	us-gov-west-1	否

必要條件

下列清單說明建立 VPC-ENI 組態時要檢閱的一些需求和建議：

- 私人裝置必須指派至您的 AWS 帳戶。
- 您必須擁有具有建立服務連結角色之權限的 AWS 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 只能連線至us-west-2區域中的 VPC。如果您的us-west-2區域中沒有 VPC，則需要建立一個 VPC。然後，若要存取其他區域中 VPC 中的資源，您必須在該地區的 VPC 與其他us-west-2區域中的 VPC 之間建立對等連線。如需對等 VPC 的相關資訊，請參閱 Amazon VPC 對等互連指南。

  設定連線時，應確認您是否具有指定 VPC 的存取權。您必須為 Device Farm 設定特定的 Amazon 彈性運算雲端 (Amazon EC2) 許可。
- 您使用的 VPC 需要 DNS 解析。
- 建立 VPC 之後，您將需要有關該地區中 VPC 的下列資訊：us-west-2
  - VPC ID
  - 子網路 ID
  - 安全群組 ID
- 您必須以每個專案為基礎設定 Amazon VPC 人雲端連線。目前，每個專案只能設定一個 VPC 組態。設定 VPC 時，Amazon VPC 會在您的 VPC 中建立一個介面，並將其指派給指定的子網路和安全群組。與專案相關聯的所有 future 工作階段都將使用已設定的 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 配置更新現有項目，因為現有項目可能具有持續在運行級別的 VPCE 設置。相反地，如果您已經使用現有的 VPCE 功能，請針對所有新專案使用 VPC-ENI。

連接到 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。一個專案在任何給定時間只能有一個 VPC-ENI 端點。若要設定專案的 VPC 存取權限，您必須瞭解下列詳細資訊：

- VPC ID (us-west-2如果您的應用程式託管於該處)，或連線至不同區域中其他 VPC 的 VPC ID。us-west-2
- 要套用至連線的適用安全性群組。
- 將與連線相關聯的子網路。工作階段啟動時，會使用最大的可用子網路。我們建議您將多個子網路與不同的可用區域相關聯，以改善 VPC 連線的可用性狀態。

建立 VPC-ENI 組態後，您可以使用主控台或 CLI 使用以下步驟更新其詳細資料。

Console

1. 登入 Device Farm 主控台，網址為 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. 在移動測試項目下，從列表中選擇項目的名稱。
4. 選擇 Project settings (專案設定)。
5. 在「V irtual Private Cloud (VPC) 設定」區段中，您可以變更VPCSubnets、和Security Groups。
6. 選擇儲存。

CLI

使用以下 AWS CLI 命令更新 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：

- 您最多可以在裝置伺服器陣列專案的 VPC 組態中提供五個安全群組。

- 您最多可以在 Device Farm 專案的 VPC 組態中提供八個子網路。
- 將 Device Farm 專案設定為與您的 VPC 搭配使用時，您可以提供的最小子網路必須至少有五個可用的 IPv4 位址。
- 目前不支援公用 IP 位址。相反地，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用網際網路存取，請使用網路位址轉譯 (NAT) 閘道。使用公用子網路設定 Device Farm 專案並不會提供測試網際網路存取權或公用 IP 位址。
- 僅支援來自服務管理 ENI 的傳出流量。這表示 ENI 無法接收來自 VPC 的未經請求的輸入要求。

搭配裝置伺服器陣列使用 Amazon VPC 端點服務-舊版 (不建議使用)

Warning

我們強烈建議您針對私有端點連線使用本頁所述的 VPC-ENI 連線，因為 VPCE 現在被視為舊版功能。與 VPCE 連接方法相比，VPC-ENI 提供了更大的靈活性，更簡單的配置，更具成本效益，並且需要大幅減少維護開銷。

Note

只有擁有已設定私有裝置的客戶才支援搭配裝置伺服器陣列使用 Amazon VPC 端點服務。若要讓您的 AWS 帳戶能夠在私有裝置上使用此功能，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。透過 VPC，您可以控制網路設定，例如 IP 位址範圍、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，則可以在 VPC 和 Device Farm 之間建立私有連接。透過此連線，您可以使用 Device Farm 來測試私有應用程式，而不會透過公用網際網路公開它們。若要讓您的 AWS 帳戶能夠在私人裝置上使用此功能，請聯絡我們。

若要將 VPC 中的資源連接到 Device Farm，您可以使用 Amazon VPC 主控台建立 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、可擴充的連線，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連線，將 VPC 中的私人內部服務安全地連線至 Device Farm 公用 VPC。 AWS PrivateLink 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 憑證遭到入侵，攻擊者可以存取您的服務資料，或將您的服務資料公開給外界。

在 Amazon VPC 中建立 VPC 端點服務後，您可以使用裝置伺服器陣列主控台在裝置伺服器陣 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

以下資訊適用於美國西部 (奧勒岡) (`us-west-2`) 區域的 Amazon VPC 使用者，且每個可用區域都有子網路：美國西部 2a、美國西部 2B 和美國西部 2c。

裝置伺服器陣列對可與 VPC 端點服務搭配使用的其他需求。當您建立並設定 VPC 端點服務以與 Device Farm 列搭配使用時，請務必選擇符合下列需求的選項：

- 服務的可用區域必須包括美國西部 2a、美國西部 2b 和美國西部 2c。與 VPC 端點服務關聯的 Network Load Balancer 會決定該 VPC 端點服務的可用區域。如果您的 VPC 端點服務未顯示這三個可用區域，則必須重新建立 Network Load Balancer 以啟用這三個區域，然後將 Network Load Balancer 與端點服務重新關聯。
- 端點服務允許的主體必須包含裝置伺服器陣列 VPC 端點 (服務 ARN) 的 Amazon 資源名稱 (ARN)。建立端點服務後，請將 Device Farm 列 VPC 端點服務 ARN 新增至允許清單，以授與 Device Farm 列存取 VPC 端點服務的權限。若要取得 Device Farm VPC 端點服務 ARN，請連絡我們。

此外，如果您在建立 VPC 端點服務時保持開啟「需要接受」設定，則必須手動接受裝置伺服器陣列傳送至端點服務的每個連線要求。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 「修改端點接受度設定」。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程度設定。

下一節說明如何建立符合這些需求的 Amazon VPC 端點服務。

步驟 1：建立 Network Load Balancer

在 VPC 和 Device Farm 之間建立私人連線的第一個步驟是建立 Network Load Balancer，將要求路由到目標群組。

New console

使用新主控台建立 Network Load Balancer

1. 在 https://console.aws.amazon.com/ec2/ 開啟 Amazon Elastic Compute Cloud (Amazon EC2) 主控台。
2. 在瀏覽窗格的 [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「配置」，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. 選取下列對應：
      - 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」選項選擇目標群組。

> **Note**
> 依預設，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-2aus-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. 在 https://console.aws.amazon.com/ec2/ 開啟 Amazon Elastic Compute Cloud (Amazon EC2) 主控台。
2. 在瀏覽窗格的 [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「配置」，選擇「內部」
6. 在「監聽器」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：設定安全性設定。
8. (選擇性) 設定您的安全性設定，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. 在 [名稱] 中，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，複查您的目標，然後選擇下一步：複查。

> **Note**
>
> 依預設，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-2aus-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 後，請使用 Amazon VPC 主控台在 VPC 中建立端點服務。

1. 前往 https://console.aws.amazon.com/vpc/ 開啟 Amazon VPC 主控台。
2. 在 [依地區分類的資源] 下方，選擇 [端點
3. 選擇 Create Endpoint Service (建立端點服務)。
4. 執行以下任意一項：

   - 如果您已經有想要端點服務使用的 Network Load Balancer，請在可用負載平衡器下選擇它，然後繼續執行步驟 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。從步驟 3 開始依照建立 Network Load Balancer 中的步驟進行，然後在 Amazon VPC 主控台中繼續執行這些步驟。

5. 對於包含的可用區域`us-west-2a`，請確認`us-west-2b`、並`us-west-2c`顯示在清單中。
6. 如果您不想手動接受或拒絕傳送至端點服務的每個連線要求，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 [允許主參與者]。
9. 請連絡我們以取得 Device Farm 列 VPC 端點 (服務 ARN) 的 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 端點服務現在已準備好可與 Device Farm 搭配使用。

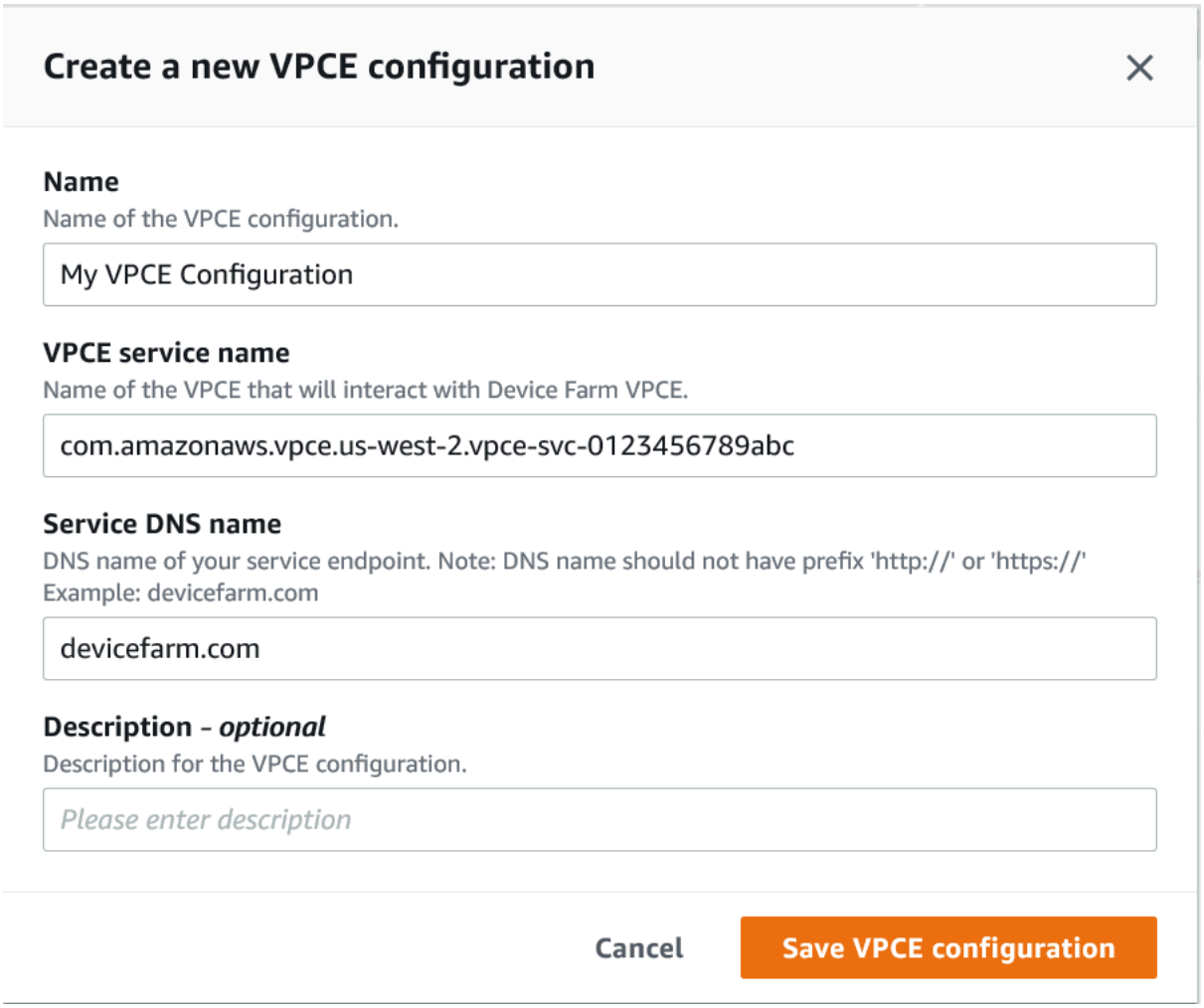
步驟 3：在 Device Farm 中建立 VPC 端點組態

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://console.aws.amazon.com/devicefarm。
2. 在導覽窗格中，選擇 [行動裝置測試]，然後選取 [私人裝置]。
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. 對於 VPCE 服務名稱，請輸入您在 Amazon VPC 主控台中記下的 Amazon VPC 端點服務名稱 (服務名稱)。名稱的形式类似于 `com.amazonaws.vpce.us-west-2.vpce-svc-id`。
7. 針對服務 DNS 名稱，輸入您要測試之應用程式的服務 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，這個對應至您的 VPC 端點服務的新網域名稱是由 Amazon Route 53 產生的，並可在您的 Device Farm 工作階段中專用。
8. 選擇儲存。



步驟 4：建立測試回合

儲存 VPC 端點組態後，您可以使用組態建立測試回合或遠端存取工作階段。如需詳細資訊，請參閱 [在 Device Farm 中建立測試回合] 或 [建立工作階段] 。

使用記錄 AWS 裝置伺服器陣列 API 呼叫AWS CloudTrail

AWS 裝置伺服器陣列已與AWS CloudTrail，提供使用者、角色或使用者所採取之動作記錄的服務AWSAWS 裝置伺服器陣列中的服務。CloudTrail以事件形式擷取 AWS 裝置伺服器陣列的所有 API 呼叫。擷取的呼叫包括來自 AWS 裝置伺服器陣列主控台的呼叫，以及對 AWS 裝置伺服器陣列 API 操作的程式碼呼叫。如果您建立追蹤，您可以啟用持續傳遞CloudTrailAmazon S3 儲存貯體的事件，包括 AWS 裝置伺服器陣列的事件。如果您不設定追蹤記錄，仍然可以透過 CloudTrail 主控台中的 Event history (事件歷史記錄) 檢視最新的事件。使用所收集的資訊CloudTrail，您可以決定向 AWS 裝置伺服器陣列發出的請求、提出請求的 IP 位址、提出請求的人員、提出要求的時間以及其他詳細資訊。

若要進一步了解 CloudTrail，請參閱 AWS CloudTrail 使用者指南。

AWS 裝置伺服器陣列資訊CloudTrail

當您建立帳戶時，系統會在您的 AWS 帳戶中啟用 CloudTrail。當 AWS 裝置伺服器陣列中發生活動時，該活動會記錄在CloudTrail與其他一起事件AWS服務事件事件歷史。您可以檢視、搜尋和下載 AWS 帳戶的最新事件。如需詳細資訊，請參閱使用 CloudTrail 事件歷程記錄檢視事件。

在您的事件的持續記錄AWS帳戶 (包括 AWS 裝置伺服器陣列的事件) 建立追蹤。一個線索啟用CloudTrail將日誌文件交付到亞馬遜 S3 存儲桶。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。該追蹤會記錄來自 AWS 分割區中所有區域的事件，並將日誌檔案交付到您指定的 Amazon S3 儲存貯體。此外，您可以設定其他 AWS 服務，以進一步分析和處理 CloudTrail 日誌中所收集的事件資料。如需詳細資訊，請參閱下列內容：

- 建立追蹤的概觀
- CloudTrail 支援的服務和整合
- 設定 CloudTrail 的 Amazon SNS 通知
- 接收多個區域的 CloudTrail 日誌檔案及接收多個帳戶的 CloudTrail 日誌檔案

何時CloudTrail日誌記錄已啟用AWS帳戶，對設備農場操作進行的 API 調用在日誌文件中進行跟踪。裝置伺服器陣列記錄與其他記錄一起寫AWS記錄檔中的服務記錄。CloudTrail 會根據期間與檔案大小，決定何時建立與寫入新檔案。

所有裝置伺服器陣列動作都會記錄並記錄在AWS CLI 參考和自動化裝置農場。例如，建立新專案或在裝置伺服器陣列中執行的呼叫會產生CloudTrail日誌文件。

每一筆事件或日誌項目都會包含產生請求者的資訊。身分資訊可協助您判斷下列事項：

- 該請求是否透過根或 AWS Identity and Access Management (IAM) 使用者憑證來提出。
- 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。
- 該請求是否由另一項 AWS 服務提出。

如需詳細資訊，請參閱 CloudTrail 使用者身分元素。

了解 AWS 裝置農場日誌檔項目

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 日誌檔案包含一個或多個日誌項目。一個事件為任何來源提出的單一請求，並包含請求動作、請求的日期和時間、請求參數等資訊。CloudTrail 日誌檔案並非依公有 API 呼叫追蹤記錄的堆疊排序，因此不會以任何特定順序出現。

下面的例子顯示了CloudTrail示範裝置伺服器陣列的記錄項目`ListRuns`動作：

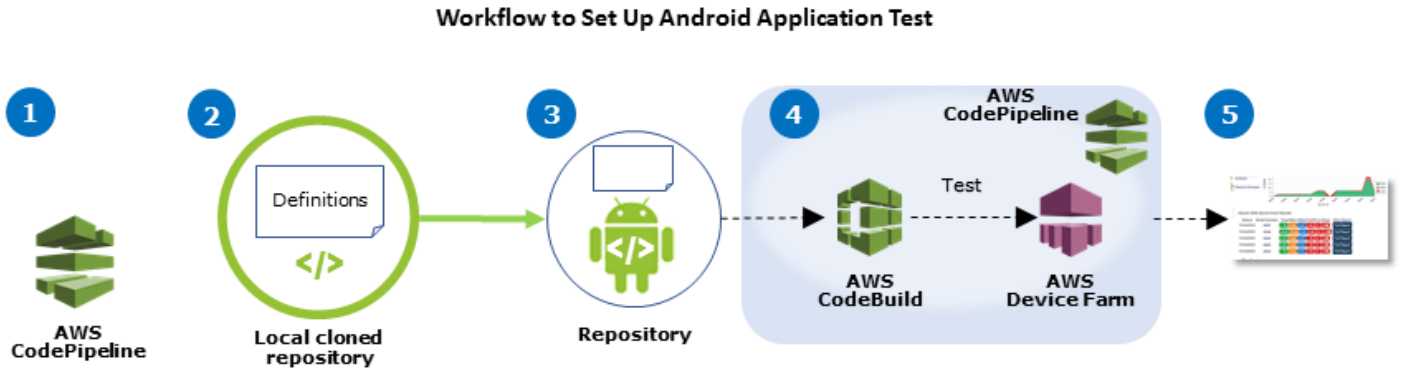
```
{
  "Records": [
    {
      "eventVersion": "1.03",
      "userIdentity": {
        "type": "Root",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "arn:aws:iam::123456789012:root",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
          "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 ...
          ]
        }
      }
    }
  ]
}
```

在中使用 AWS 裝置伺服器陣列CodePipeline測試階段

您可以使用AWS CodePipeline將裝置伺服器陣列中設定的行動應用程式測試併入 AWS 管理的自動化發行管道中。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。若要建立此配管組態，請參閱教程：在推送到 Android 應用程序時構建和測試GitHub。



1。設定	2. 新增定義	3. 推送	4. 建置與測試	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。

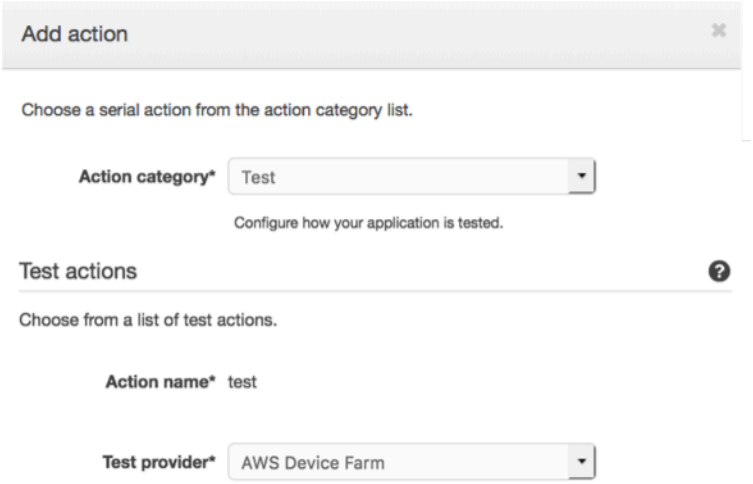
配置CodePipeline使用您的裝置伺服器陣列測試

在這些步驟中，我們假設您有設定裝置伺服器陣列專案和創建了一個管道。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。

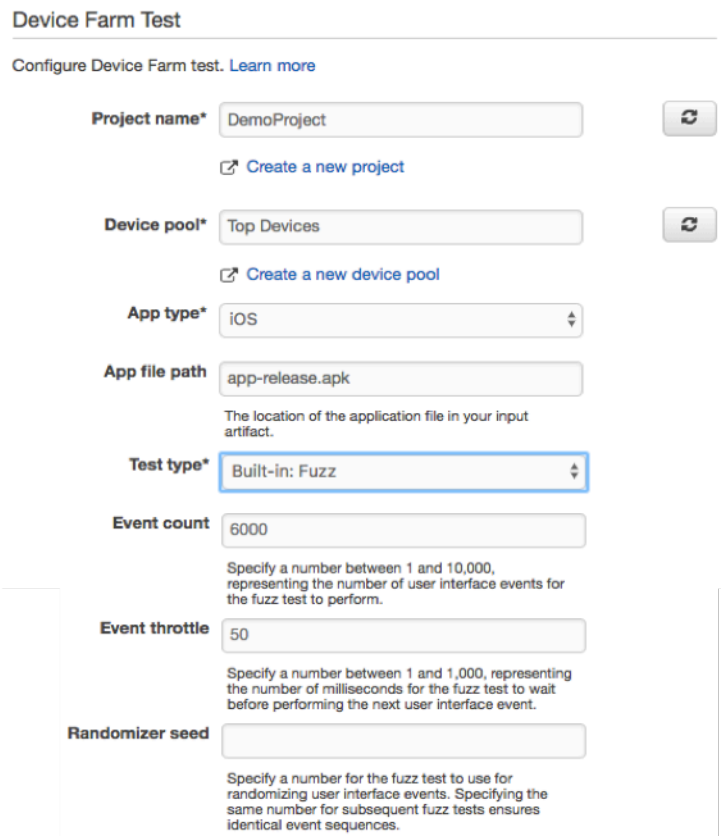
將裝置伺服器陣列測試執行設定為CodePipeline測試動作

1. 登入AWS Management Console並打開CodePipeline控制台https://console.aws.amazon.com/codepipeline/。

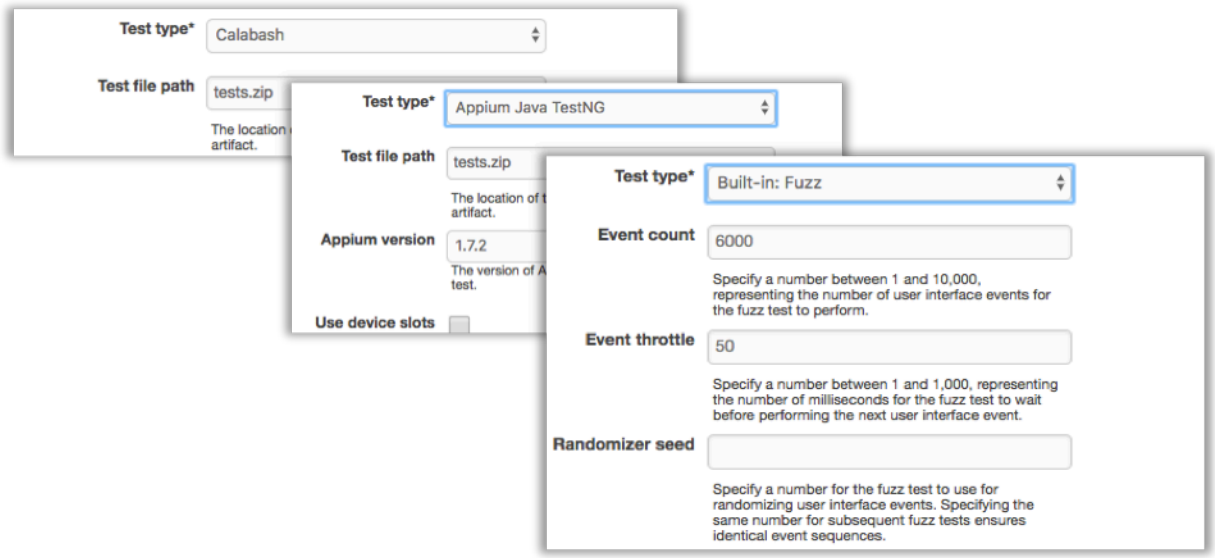
2. 選擇應用程式的發行管道。
3. 在測試階段面板中，選擇鉛筆圖示，然後選擇 Action (動作)。
4. 在 Add action (新增動作) 面板中，針對 Action category (動作類別)，選擇 Test (測試)。
5. 在 Action name (動作名稱) 中，輸入名稱。
6. 在 Test provider (測試提供者) 中，選擇 AWS Device Farm。



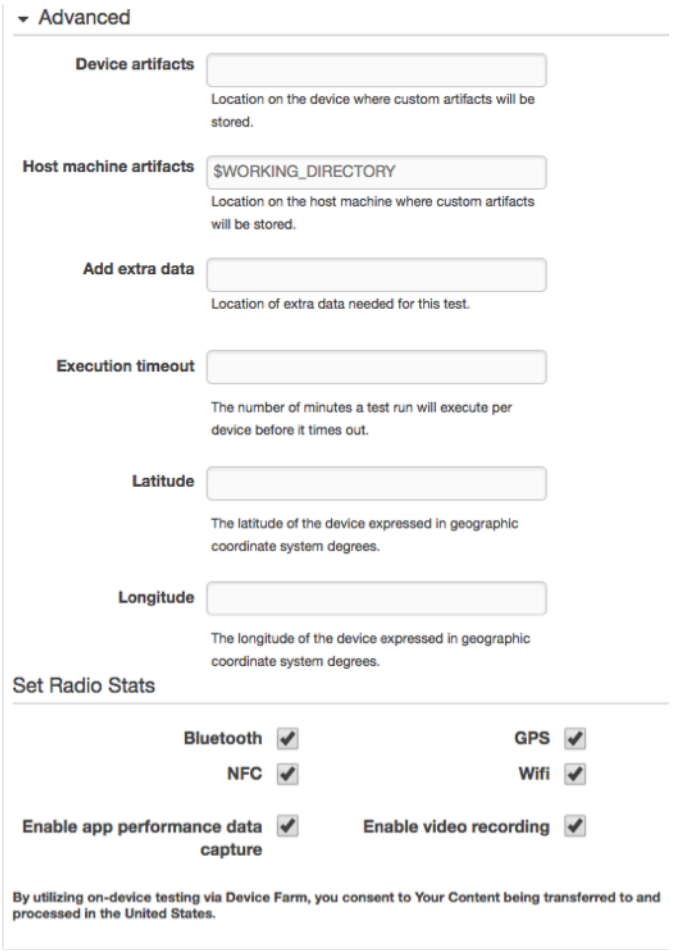
7. 在專案名稱，選擇您現有的裝置伺服器陣列專案或選擇創建一個新項目。
8. 在 Device pool (裝置集區) 中，選擇您現有的裝置集區，或是選擇 Create a new device pool (新增新裝置集區)。若您建立裝置集區，您需要選取一組測試裝置。
9. 在 App type (應用程式類型) 中，選擇應用程式的平台。



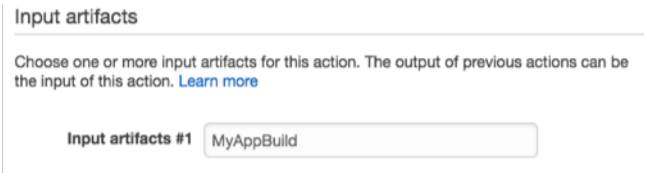
10. 在 App file path (應用程式檔案路徑) 中，輸入已編譯的應用程式套件路徑。路徑為相對於您測試輸入成品根的相對路徑。
11. 在 Test type (測試類型) 中，執行下列其中一項作業：
    - 如果您使用的是其中一個內建的 [裝置伺服器陣列] 測試，請選擇 [裝置伺服器陣列] 專案中設定的測試類型。
    - 如果您沒有使用其中一個「裝置伺服器陣列」內建測試，請在測試檔案路徑，輸入測試定義檔案的路徑。路徑為相對於您測試輸入成品根的相對路徑。



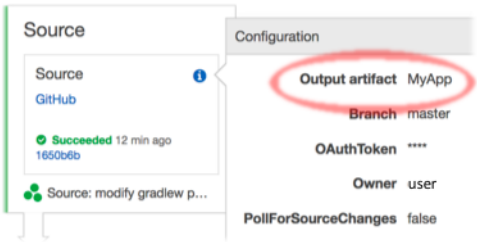
12. 在剩餘欄位中，提供適用於您測試及應用程式類型的組態。
13. (選用) 在 Advanced (進階) 中，提供測試執行的詳細組態。



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，您可以透過在管道圖表中資訊圖示的上方暫留，找到每個階段的輸出成品名稱。如果您的管道直接從來源階段，選擇MyApp。如果您的管道包含建置階段，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 Save pipeline change (儲存管道變更)，然後選擇 Save change (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 CLIAWS 裝置伺服器陣列的參考

若要使用AWS Command Line Interface(AWS CLI) 若要執行裝置伺服器陣列命令，請參閱AWS CLIAWS 裝置伺服器陣列的參考。

有關的一般信息AWS CLI，請參閱AWS Command Line Interface使用者指南和AWS CLI指令參考。

視窗PowerShellAWS 裝置伺服器陣列的參考

使用視窗PowerShell若要執行裝置伺服器陣列命令，請參閱裝置陣列指令程式參考在AWS Tools for Windows PowerShell指令程式參考。如需詳細資訊，請參閱設定適用於視窗的 AWS 工具PowerShell在AWS Tools for Windows PowerShell使用者指南。

自動化 AWS 裝置伺服器陣列

以程式設計方式存取裝置伺服器陣列是一種強大的方式，可自動化您需要完成的一般工作，例如排程執行或下載執行、套件或測試的成品。AWS SDK 和 AWS CLI 提供方法來執行此操作。

該AWSSDK 可讓您存取每個AWS服務，包括裝置伺服器陣列、Amazon S3 等。如需詳細資訊，請參閱

- AWS 工具和軟體開發套件
- 該AWS 裝置伺服器陣列 API 參考

範例：使用AWS開始執行裝置伺服器陣列並收集成品的 SDK

下列範例提供如何使用AWSSDK 可與裝置伺服器陣列搭配使用。此範例會執行下列操作：

- 上傳測試和應用程式套件至裝置伺服器陣列
- 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")
```

Device Farm 錯誤

在本節中，您將找到錯誤訊息和程序，以協助您修正 Device Farm 的常見問題。

主題

AWS 裝置伺服器陣列中的 Android 應用程式測試

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，請參閱在 AWS Device Farm 中使用安卓測試。

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 裝置伺服器陣列會剖析輸出的資訊`aapt debug badging <path to your package>`指令。

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，請參閱在 AWS Device Farm 中使用安卓測試。

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 裝置伺服器陣列會從輸出中剖析套件名稱值`aapt debug badging` *`<path to your package>`*指令。

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，請參閱在 AWS Device Farm 中使用安卓測試。

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 裝置伺服器陣列會從輸出中剖析 SDK 版本值`aapt debug badging` *`<path to your package>`*指令。

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，請參閱在 AWS Device Farm 中使用安卓測試。

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**
>
> 我們找不到有效的AndroidManifest.xml 在您的應用程序中。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 裝置伺服器陣列會使用命令從 XML 剖析樹狀結構剖析資訊，找出套件中包含的 XML 檔案`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，請參閱在 AWS Device Farm 中使用安卓測試。

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 裝置農場會使用命令從 xml 剖析樹中剖析套件中包含的 xml 檔案的許可資訊`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，請參閱在 AWS Device Farm 中使用安卓測試。

我的 Android 應用程序中的某些窗口顯示空白或黑屏

如果您正在測試 Android 應用程序，並發現應用程序中的某些窗口在設備農場的測試視頻錄製中出現黑屏，則您的應用程序可能正在使用 AndroidFLAG_SECURE功能。此旗標 (如中所述)安卓官方文件) 用於防止螢幕錄製工具記錄應用程式的某些視窗。因此，如果窗口使用此標誌，Device Farm 的屏幕錄製功能（用於自動化和遠程訪問測試）可能會顯示黑屏代替應用程序的窗口。

開發人員通常會在應用程式中包含敏感資訊 (例如登入頁面) 的網頁上使用此旗標。如果您看到某些頁面（例如其登錄頁面）的應用程序屏幕黑屏，請與您的開發人員合作以獲取不使用此標誌進行測試的應用程序構建。

此外，請注意，裝置伺服器陣列仍可與具有此旗標的應用程式視窗互動。因此，如果您的應用程序的登錄頁面顯示為黑屏，則您仍然可以輸入憑據以登錄應用程序（從而查看未被阻止的頁面FLAG_SECURE標誌）。

AWS 裝置伺服器陣列中的 Java JUnit 測試進行疑難排解

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_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。在下面的例子中，包的名稱是zip-with-dependencies. 拉鍊。

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，請參閱使用 Appium 和 AWS Device Farm。

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。

在下面的例子中，包的名稱是zip-with-dependencies. 拉鍊。

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，請參閱使用 Appium 和 AWS Device Farm。

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。

在下面的例子中，包的名稱是zip-with-dependencies. 拉鍊。

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 .
```

如果蘋果 Java JUnit 包是有效的，你會發現至少一個##文件裡面###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，請參閱使用 Appium 和 AWS Device Farm。

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。

在下面的例子中，包的名稱是zip-with-dependencies. 拉鍊。

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 .
```

如果蘋果 Java JUnit 包是有效的，你會發現至少一個*##像文件acme-android-appium-1.0 ##### *在我們的例子中。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，請參閱使用 Appium 和 AWS Device Farm。

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。

在下面的例子中，包的名稱是zip-with-dependencies. 拉鍊。

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 .
```

你應該找到至少一個 jar 文件*acme-android-appium-1.0 ######*在我們的例子中。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，請參閱使用 Appium 和 AWS Device Farm。

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。

在下面的例子中，包的名稱是zip-with-dependencies. 拉鍊。

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，請參閱使用 Appium 和 AWS Device Farm。

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。

在下面的例子中，包的名稱是zip-with-dependencies. 拉鍊。

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 使用 Appium 和 AWS Device Farm。

AWS 裝置伺服器陣列中的 Java JUnit Web 應用程式測試進行疑難排解

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。如需將 Appium 與裝置伺服器陣列搭配使用的詳細資訊，請參閱the section called “Appium”。

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。在下面的例子中，包的名稱是zip-with-dependencies. 拉鍊。

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。

在下面的例子中，包的名稱是zip-with-dependencies. 拉鍊。

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。

在下面的例子中，包的名稱是zip-with-dependencies. 拉鍊。

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 .
```

如果蘋果 Java JUnit 包是有效的，你會發現至少一個##文件裡面###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_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。

在下面的例子中，包的名稱是zip-with-dependencies. 拉鍊。

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 .
```

如果蘋果 Java JUnit 包是有效的，你會發現至少一個*##像文件acme-android-appium-1.0 ###*在我們的例子中。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。

在下面的例子中，包的名稱是zip-with-dependencies. 拉鍊。

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 .
   ```

   你應該找到至少一個 jar 文件*acme-android-appium-1.0 ####*在我們的例子中。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。

在下面的例子中，包的名稱是zip-with-dependencies. 拉鍊。

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。

在下面的例子中，包的名稱是zip-with-dependencies. 拉鍊。

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 使用 Appium 和 AWS Device Farm。

在 AWS 裝置伺服器陣列中進行 Java 測試的疑難排解

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。在下面的例子中，包的名稱是zip-with-dependencies. 拉鍊。

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，請參閱使用 Appium 和 AWS Device Farm。

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。

在下面的例子中，包的名稱是zip-with-dependencies. 拉鍊。

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，請參閱使用 Appium 和 AWS Device Farm。

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。

在下面的例子中，包的名稱是zip-with-dependencies. 拉鍊。

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 .
```

如果蘋果 Java JUnit 包是有效的，你會發現至少一個##文件裡面###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，請參閱使用 Appium 和 AWS Device Farm。

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。

在下面的例子中，包的名稱是zip-with-dependencies. 拉鍊。

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 .
```

如果蘋果 Java JUnit 包是有效的，你會發現至少一個*##像文件acme-android-appium-1.0 ##### *在我們的例子中。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，請參閱使用 Appium 和 AWS Device Farm。

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。

在下面的例子中，包的名稱是zip-with-dependencies. 拉鍊。

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 .
```

你應該找到至少一個 jar 文件*acme-android-appium-1.0 ######*在我們的例子中。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 使用 Appium 和 AWS Device Farm。

AWS 裝置伺服器陣列中 Web 應用程式的 Appium Java 測試疑難排解

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。在下面的例子中，包的名稱是zip-with-dependencies. 拉鍊。

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，請參閱使用 Appium 和 AWS Device Farm。

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。

在下面的例子中，包的名稱是zip-with-dependencies. 拉鍊。

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，請參閱使用 Appium 和 AWS Device Farm。

APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_JAR_MISSING_IN_DEPENDE

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。

在下面的例子中，包的名稱是zip-with-dependencies. 拉鍊。

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 .
```

如果蘋果 Java JUnit 包是有效的，你會發現至少一個##文件裡面###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，請參閱使用 Appium 和 AWS Device Farm。

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。

在下面的例子中，包的名稱是zip-with-dependencies. 拉鍊。

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 .
```

如果蘋果 Java JUnit 包是有效的，你會發現至少一個*##像文件acme-android-appium-1.0 # #####*在我們的例子中。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，請參閱使用 Appium 和 AWS Device Farm。

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。

在下面的例子中，包的名稱是zip-with-dependencies. 拉鍊。

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 .
```

你應該找到至少一個 jar 文件*acme-android-appium-1.0 ######*在我們的例子中。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 使用 Appium 和 AWS Device Farm。

AWS 裝置伺服器陣列中的 Appium Python 測試進行疑難排解

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，請參閱使用 Appium 和 AWS Device Farm。

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，請參閱使用 Appium 和 AWS Device Farm。

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，請參閱使用 Appium 和 AWS Device Farm。

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，請參閱使用 Appium 和 AWS Device Farm。

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，請參閱使用 Appium 和 AWS Device Farm。

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，請參閱使用 Appium 和 AWS Device Farm。

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 .
```

你應該找到*requirements.txt*工作目錄中的文件。

```
.
|-- 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，請參閱使用 Appium 和 AWS Device Farm。

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，請參閱使用 Appium 和 AWS Device Farm。

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，請參閱使用 Appium 和 AWS Device Farm。

APPIUM_ 蟒蛇 _ 測試封裝 _ 依賴性輪子 _ 不足

如果您看到下列訊息，請依照以下步驟修復問題。

> **Warning**
>
> 我們無法在駕駛室目錄中找到足夠的車輪依賴關係。請解壓縮您的測試包，然後打開駕駛室目錄。請確認您已在 requirements.txt 檔案中指定了所有車輪相依性。

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. 檢查長度*requirements.txt*文件以及的數量*. ##*駕駛室目錄中的依賴文件：

```
$ cat requirements.txt | egrep "." |wc -l
    12
$ ls wheelhouse/ | egrep ".+\.whl" | wc -l
    11
```

如果數*. ##*從屬文件小於您的非空行的數量*requirements.txt*文件，那麼你需要確保以下內容：

- 有一個*. ##*與中的每一行相對應的從屬文件*requirements.txt*文件。
- 中沒有其他線*requirements.txt*包含相依性套件名稱以外資訊的檔案。
- 在中的多行中不會重複相依性名稱*requirements.txt*文件，使文件中的兩行可以對應於一行*. ##*從屬檔案。

AWS 裝置伺服器陣列不支援*requirements.txt*不直接對應至相依套件的檔案，例如指定全域選項的行`pip install`指令。請參閱需求檔案格式以取得全域選項清單。

如需詳細資訊，請參閱 使用 Appium 和 AWS Device Farm。

AWS 裝置伺服器陣列中 Appium Python 網路應用程式測試的疑難排解

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，請參閱使用 Appium 和 AWS Device Farm。

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，請參閱使用 Appium 和 AWS Device Farm。

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，請參閱使用 Appium 和 AWS Device Farm。

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，請參閱使用 Appium 和 AWS Device Farm。

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，請參閱使用 Appium 和 AWS Device Farm。

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，請參閱使用 Appium 和 AWS Device Farm。

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，請參閱使用 Appium 和 AWS Device Farm。

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，請參閱使用 Appium 和 AWS Device Farm。

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 使用 Appium 和 AWS Device Farm。

疑難排解 AWS 裝置伺服器陣列中的儀

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。

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**
>
> 無法開啟您的測試 APK 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。在下面的例子中，包的名稱是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，請參閱使用適用於安卓和 AWS Device Farm 的儀器。

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。

> **Warning**
>
> 無法擷取測試套件的相關資訊。請執行命令「aapt debug badging <path to your test package>」，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，裝置伺服器陣列會剖析輸出`aapt debug badging <path to your package>`指令。

請確認您可以在檢測測試套件上成功執行此命令。

在下面的例子中，包的名稱是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，請參閱使用適用於安卓和 AWS Device Farm 的儀器。

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。

> **Warning**
>
> 我們找不到儀器運行器的值AndroidManifest.xml。請執行指令「aapt 傾印 xmltree」來確認測試套件是否有效 <path to your test package>AndroidManifest.xml」，並在找到關鍵字「儀器」後面的儀器運行器值後再次嘗試。

在上傳驗證程序期間，裝置伺服器陣列會針對封裝中包含的 XML 檔案，剖析 XML 剖析樹狀結構中的檢測執行程式值。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。

在下面的例子中，包的名稱是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，請參閱使用適用於安卓和 AWS Device Farm 的儀器。

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。

> **Warning**
>
> 我們找不到有效的AndroidManifest.xml 在你的測試包中。請執行指令「aapt 傾印 xmltree」來確認測試套件是否有效 <path to your test package>AndroidManifest.xml「，並在命令不打印任何錯誤後再試一次。

在上傳驗證程序期間，裝置伺服器陣列會使用下列命令，從 XML 剖析樹狀結構剖析封裝中包含的 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。

在下面的例子中，包的名稱是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，請參閱使用適用於安卓和 AWS Device Farm 的儀器。

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。

> **Warning**
>
> 在測試套件中找不到套件名稱。請執行命令「aapt debug badging <path to your test package>」，確認測試套件是否有效，然後在以關鍵字「package: name.」找到套件名稱值後再試一次。

在上傳驗證程序期間，裝置伺服器陣列會從下列命令的輸出剖析套件名稱值：`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。

在下面的例子中，包的名稱是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 使用適用於安卓和 AWS Device Farm 的儀器。

AWS 裝置伺服器陣列中的 iOS 應用程式測試

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。在下面的例子中，包的名稱是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，請參閱在 AWS Device Farm 中使用 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。

在下面的例子中，包的名稱是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，請參閱在 AWS Device Farm 中使用 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。

在下面的例子中，包的名稱是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 應用程序包有效，您會發現. ####目錄像*AWSDeviceFarmiOSReferenceApp. ### #*在我們的例子裡面##目錄。

   ```
   .
   `-- Payload (directory)
           `-- AWSDeviceFarmiOSReferenceApp.app (directory)
                       |-- Info.plist
                       `-- (any other files)
   ```

   如需詳細資訊，請參閱在 AWS Device Farm 中使用 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。

在下面的例子中，包的名稱是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 應用程序包有效，您將找到*##. P ##*文件裡面*. ####*目錄像*AWSDeviceFarmiOSReferenceApp. ####*在我們的例子中。

```
.
`-- Payload (directory)
        `-- AWSDeviceFarmiOSReferenceApp.app (directory)
                        |-- Info.plist
                        `-- (any other files)
```

如需詳細資訊，請參閱在 AWS Device Farm 中使用 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序，然後在 .app 目錄中打開 Info.plist 文件，驗證密鑰是否為「用戶界面」RequiredDeviceCapabilities已指定「，然後再試一次。

在下面的例子中，包的名稱是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 .
```

你應該找到*##. P ##*文件裡面*. ####*目錄像*AWSDeviceFarmiOSReferenceApp. ####*在我們的例子中：

```
.
`-- 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，請參閱在 AWS Device Farm 中使用 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序，然後在 .app 目錄中打開 Info.plist 文件，驗證密鑰「CFBundleSupportedPlatforms已指定「，然後再試一次。

在下面的例子中，包的名稱是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 .
```

你應該找到*##. P ##*文件裡面*. ####*目錄像*AWSDeviceFarmiOSReferenceApp. ####*在我們的例子中：

```
.
`-- 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，請參閱在 AWS Device Farm 中使用 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序，然後在 .app 目錄中打開 Info.plist 文件，驗證鍵值「CFBundleSupportedPlatforms「不包含關鍵字「模擬器」，然後再試一次。

在下面的例子中，包的名稱是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 .
```

你應該找到*##. P ##*文件裡面*. ####*目錄像*AWSDeviceFarmiOSReferenceApp. ####*在我們的例子中：

```
.
`-- 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，請參閱在 AWS Device Farm 中使用 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序，然後在 .app 目錄中打開 Info.plist 文件，驗證密鑰是否為「用戶界面」DeviceFamily已指定「，然後再試一次。

在下面的例子中，包的名稱是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 .
```

你應該找到*##. P ##*文件裡面*. ####*目錄像*AWSDeviceFarmiOSReferenceApp. ####*在我們的例子中：

```
.
```

```
`-- 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，請參閱在 AWS Device Farm 中使用 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序，然後在 .app 目錄中打開 Info.plist 文件，驗證密鑰「CFBundleIdentifier已指定「，然後再試一次。

在下面的例子中，包的名稱是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 .
```

你應該找到*##. P ##*文件裡面*. ####*目錄像*AWSDeviceFarmiOSReferenceApp. ####*在我們的例子中：

```
.
`-- 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，請參閱在 AWS Device Farm 中使用 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序，然後在 .app 目錄中打開 Info.plist 文件，驗證密鑰「CFBundleExecutable已指定「，然後再試一次。

在下面的例子中，包的名稱是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 .
```

你應該找到*##. P ##*文件裡面*. ####*目錄像*AWSDeviceFarmiOSReferenceApp. ####*在我們的例子中：

```
.
`-- 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 在 AWS Device Farm 中使用 iOS 測試。

AWS 裝置伺服器陣列中的 XCTest 測試疑難排解

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。在下面的例子中，包的名稱是swiftExampleTests.c-1. 拉鍊。

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，請參閱使用適用於 iOS 和 AWS Device Farm 的 XCTest。

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。

在下面的例子中，包的名稱是swiftExampleTests.c-1. 拉鍊。

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 軟件包是有效的，你會發現一個名稱類似於*swiftExampleTests.xctest*在工作目錄內。該名稱應以 *.xctest* 結尾。

```
.
`-- swiftExampleTests.xctest (directory)
              |-- Info.plist
              `-- (any other files)
```

如需詳細資訊，請參閱使用適用於 iOS 和 AWS Device Farm 的 XCTest。

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。

在下面的例子中，包的名稱是swiftExampleTests.c-1. 拉鍊。

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 軟件包是有效的，你會發現*##. P ##*文件裡面*.xctest*目錄。在下面的例子中，該目錄被調用*swiftExampleTests.xctest*。

```
.
`-- swiftExampleTests.xctest (directory)
              |-- Info.plist
              `-- (any other files)
```

如需詳細資訊，請參閱使用適用於 iOS 和 AWS Device Farm 的 XCTest。

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。請解壓縮您的測試包，然後打開 Info.plist 文件，驗證密鑰是否為「CFBundleIdentifier已指定「，然後再試一次。

在下面的例子中，包的名稱是swiftExampleTests.c-1. 拉鍊。

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 .
```

你應該找到*##. P ##*文件裡面*.xctest*目錄像*swiftExampleTests.xctest*在我們的例子中：

```
.
`-- 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，請參閱使用適用於 iOS 和 AWS Device Farm 的 XCTest。

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。請解壓縮您的測試包，然後打開 Info.plist 文件，驗證密鑰是否為「CFBundleExecutable已指定「，然後再試一次。

在下面的例子中，包的名稱是swiftExampleTests.c-1. 拉鍊。

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 .
```

你應該找到*##. P ##*文件裡面*.xctest*目錄像*swiftExampleTests.xctest*在我們的例子中：

```
.
`-- 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 使用適用於 iOS 和 AWS Device Farm 的 XCTest。

AWS 裝置伺服器陣列中的 XCTest 使用者介面測試的

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。

> **Warning**
>
> 無法開啟您的測試 IPA 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 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XCTest UI。

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。

> **Warning**
>
> 在測試套件中找不到承載目錄。請解壓縮您的測試套件，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 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*Payload*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XCTest UI。

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。

> **Warning**
>
> 在承載目錄中找不到 .app 目錄。請解壓縮您的測試套件，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 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 使用者介面套件是有效的，您會發現. ####目錄像####-#####. ##在我們的例子裡###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XCTest UI。

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。

> **Warning**
>
> 在 .app 目錄中找不到外掛程式目錄。請解壓縮您的測試套件，開啟 .app 目錄，確認目錄中有外掛程式目錄，然後再試一次。

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 使用者介面套件是有效的，您會發現##內的目錄. ####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XCTest UI。

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。

> **Warning**
>
> 在外掛程式目錄中找不到 .xctest 目錄。請解壓縮您的測試套件，開啟外掛程式目錄，確認目錄中有 .xctest 目錄，然後再試一次。

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 使用者介面套件是有效的，您會發現*.xctest*裡面的目錄*##*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XCTest UI。

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。

> **Warning**
>
> 在 .app 目錄中找不到 Info.plist 檔案。請解壓縮您的測試套件，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 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 使用者介面套件是有效的，您會發現*##. P ##*文件裡面*. ####*目錄。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XCTest UI。

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。

> **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 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 使用者介面套件是有效的，您會發現*##. P ##*文件裡面*.xctest*目錄。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XCTest UI。

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。

> **Warning**
>
> 在 Info.plist 檔案中找不到 CPU 架構值。請解壓縮您的測試包，然後打開 .app 目錄中的 Info.plist 文件，驗證密鑰是否為「用戶界面」RequiredDeviceCapabilities已指定「，然後再試一次。

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 .
```

你應該找到*##. P ##*文件裡面*. ####*目錄像*####-#####. ##*在我們的例子中：

```
.
`-- 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XCTest UI。

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。

> **Warning**
>
> 在 Info.plist 中找不到平台值。請解壓縮您的測試包，然後打開 .app 目錄中的 Info.plist 文件，驗證密鑰是否為「CFBundleSupportedPlatforms已指定「，然後再試一次。

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 .
```

你應該找到*##. P ##*文件裡面*. ####*目錄像*####-#####. ##*在我們的例子中：

```
.
`-- 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XCTest UI。

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。

> **Warning**
>
> Info.plist 檔案中的平台裝置值錯誤。請解壓縮您的測試包，然後打開 .app 目錄中的 Info.plist 文件，驗證密鑰的值是否為「CFBundleSupportedPlatforms「不包含關鍵字「模擬器」，然後再試一次。

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 .
   ```

你應該找到*##. P ##*文件裡面*. ####*目錄像*####-#####. ##*在我們的例子中：

```
.
`-- 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XCTest UI。

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。

> **Warning**
>
> 在 Info.plist 檔案中找不到表單係數值。請解壓縮您的測試包，然後打開 .app 目錄中的 Info.plist 文件，驗證密鑰是否為「用戶界面」DeviceFamily已指定「，然後再試一次。

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 .
```

你應該找到*##. P ##*文件裡面*. ####*目錄像*####-#####. ##*在我們的例子中：

```
.
`-- 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XCTest UI。

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。

> **Warning**
>
> 在 Info.plist 檔案中找不到套件名稱值。請解壓縮您的測試包，然後打開 .app 目錄中的 Info.plist 文件，驗證密鑰是否為「CFBundleIdentifier已指定「，然後再試一次。

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 .
```

你應該找到*##. P ##*文件裡面*. ####*目錄像*####-#####. ##*在我們的例子中：

```
.
`-- 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XCTest UI。

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。

> **Warning**
>
> 在 Info.plist 檔案中找不到可執行的值。請解壓縮您的測試包，然後打開 .app 目錄中的 Info.plist 文件，驗證密鑰是否為「CFBundleExecutable已指定「，然後再試一次。

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 .
```

你應該找到*##. P ##*文件裡面*. ####*目錄像*####-#####. ##*在我們的例子中：

```
.
`-- 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XCTest UI。

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。

> **Warning**
>
> 在 .xctest 目錄的 Info.plist 檔案中找不到套件名稱值。請解壓縮您的測試包，然後打開 .xctest 目錄中的 Info.plist 文件，驗證密鑰是否為「CFBundleIdentifier已指定「，然後再試一次。

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 .
```

你應該找到*##. P ##*文件裡面*. ####*目錄像*####-#####. ##*在我們的例子中：

```
.
`-- 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XCTest UI。

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。

> **Warning**
>
> 在 .xctest 目錄的 Info.plist 檔案中找不到可執行的值。請解壓縮您的測試包，然後打開 .xctest 目錄中的 Info.plist 文件，驗證密鑰是否為「CFBundleExecutable已指定「，然後再試一次。

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 .
```

你應該找到*##. P ##*文件裡面*. ####*目錄像*####-#####. ##*在我們的例子中：

```
.
`-- 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 XCTest UI。

AWS Device Farm 中的安全性

雲端安全是 AWS 最重視的一環。身為 AWS 客戶的您，將能從資料中心和網路架構的建置中獲益，以滿足組織最為敏感的安全要求。

安全是 AWS 與您共同肩負的責任。共同責任模型將其描述為雲端的安全性和雲端中的安全性：

- 雲端本身的安全 – AWS 負責保護在 AWS Cloud 中執行 AWS 服務的基礎設施。AWS 也提供您可安全使用的服務。第三方稽核人員會定期測試和驗證我們安全性的有效性，作為 AWS 合規計劃的一部分。若要了解適用於 AWS Device Farm 的合規計畫，請參閱合規計畫的 AWS 服務範圍。
- 雲端內部的安全：您的責任取決於所使用的 AWS 服務。您也必須對其他因素負責，包括資料的機密性、您公司的要求和適用法律和法規。

本文件可協助您瞭解如何在使用裝置伺服器陣列時套用共同的責任模型。下列主題說明如何設定裝置伺服器陣列以符合安全性和合規性目標。您也會學到如何使用其他可協助您監控和保護裝置伺服器陣列資源的 AWS 服務。

主題

AWS Device Farm 中的身分和存取管理

物件

使用方式 AWS Identity and Access Management (IAM) 會因您在 Device Farm 中執行的工作而有所不同。

服務使用者 — 如果您使用裝置伺服器陣列服務來執行工作，則系統管理員會為您提供所需的認證和權限。當您使用更多 Device Farm 功能來完成工作時，您可能需要其他權限。瞭解存取許可的管理方式可協助您向管理員請求正確的許可。如果您無法存取 Device Farm 中的功能，請參閱疑難排解 AWS Device Farm 身分和存取。

服務管理員 — 如果您負責公司的 Device Farm 資源，您可能擁有 Device Farm 的完整存取權。決定服務使用者應存取哪些裝置伺服器陣列功能和資源是您的工作。接著，您必須將請求提交給您的 IAM 管理員，來變更您服務使用者的許可。檢閱此頁面上的資訊，了解 IAM 的基本概念。若要深入瞭解貴公司如何搭配 Device Farm 使用 IAM，請參閱AWS Device Farm 如何與 IAM 搭配使用。

IAM 管理員 — 如果您是 IAM 管理員，您可能想要瞭解如何撰寫政策以管理 Device Farm 的存取權限的詳細資訊。若要檢視可在 IAM 中使用的 Device Farm 身分型政策範例，請參閱。AWS Device Farm 身分型政策範例

使用身分驗證

身分驗證是使用身分憑證登入 AWS 的方式。您必須以 AWS 帳戶根使用者、IAM 使用者身分，或擔任 IAM 角色進行 驗證 (登入至 AWS)。

您可以使用透過身分來源 AWS IAM Identity Center 提供的憑證，以聯合身分登入 AWS。(IAM Identity Center) 使用者、貴公司的單一登入身分驗證和您的 Google 或 Facebook 憑證都是聯合身分的範例。您以聯合身分登入時，您的管理員先前已設定使用 IAM 角色的聯合身分。您 AWS 藉由使用聯合進行存取時，您會間接擔任角色。

根據您的使用者類型，您可以登入 AWS Management Console 或 AWS 存取入口網站。如需登入至 AWS 的相關資訊，請參閱《AWS 登入 使用者指南》中的如何登入您的 AWS 帳戶。

如果您是以程式設計的方式存取 AWS，AWS 提供軟體開發套件 (SDK) 和命令列介面 (CLI)，以便使用您的憑證透過密碼編譯方式簽署您的請求。如果您不使用 AWS 工具，您必須自行簽署請求。如需使用建議的方法自行簽署請求的相關資訊，請參閱《IAM 使用者指南》中的簽署 AWS API 請求。

無論您使用何種身分驗證方法，您可能都需要提供額外的安全性資訊。例如，AWS 建議您使用多重要素驗證 (MFA) 以提高帳戶的安全。如需更多資訊，請參閱《AWS IAM Identity Center 使用者指南》中的多重要素驗證和《IAM 使用者指南》中的在 AWS 中使用多重要素驗證 (MFA)。

AWS 帳戶 根使用者

如果是建立 AWS 帳戶，您會先有一個登入身分，可以完整存取帳戶中所有 AWS 服務 與資源。此身分稱為 AWS 帳戶 根使用者，使用建立帳戶時所使用的電子郵件地址和密碼即可登入並存取。強烈建議您不要以根使用者處理日常作業。保護您的根使用者憑證，並將其用來執行只能由根使用者執行的任

務。如需這些任務的完整清單，了解需以根使用者登入的任務，請參閱《IAM 使用者指南》中的需要根使用者憑證的任務。

IAM 使用者和群組

IAM 使用者是您 AWS 帳戶 中的一種身分，具備單一人員或應用程式的特定許可。建議您盡可能依賴暫時憑證，而不是擁有建立長期憑證 (例如密碼和存取金鑰)的 IAM 使用者。但是如果特定使用案例需要擁有長期憑證的 IAM 使用者，建議您輪換存取金鑰。如需詳細資訊，請參閱《IAM 使用者指南》中的為需要長期憑證的使用案例定期輪換存取金鑰。

IAM 群組是一種指定 IAM 使用者集合的身分。您無法以群組身分登入。您可以使用群組來一次為多名使用者指定許可。群組可讓管理大量使用者許可的過程變得更為容易。例如，您可以擁有一個名為 IAMAdmins 的群組，並給予該群組管理 IAM 資源的許可。

使用者與角色不同。使用者只會與單一人員或應用程式建立關聯，但角色的目的是在由任何需要它的人員取得。使用者擁有永久的長期憑證，但角色僅提供暫時憑證。若要進一步了解，請參閱《IAM 使用者指南》中的建立 IAM 使用者 (而非角色) 的時機。

IAM 角色

IAM 角色是您 AWS 帳戶 中的一種身分，具備特定許可。它類似 IAM 使用者，但不與特定的人員相關聯。您可以在 AWS Management Console 中透過切換角色來暫時取得 IAM 角色。您可以透過呼叫 AWS CLI 或 AWS API 操作，或是使用自訂 URL 來取得角色。如需使用角色的方法的相關資訊，請參閱《IAM 使用者指南》中的使用 IAM 角色。

使用暫時憑證的 IAM 角色在下列情況中非常有用：

- 聯合身分使用者存取 — 如需向聯合身分指派許可，請建立角色，並為角色定義許可。當聯合身分進行身分驗證時，該身分會與角色建立關聯，並取得由角色定義的許可。如需有關聯合角色的詳細資訊，請參閱《IAM 使用者指南》https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_create_for-idp.html中的為第三方身分提供者建立角色。如果您使用 IAM Identity Center，則需要設定許可集。為控制身分驗證後可以存取的內容，IAM Identity Center 將許可集與 IAM 中的角色相關聯。如需有關許可集的資訊，請參閱《AWS IAM Identity Center 使用者指南》中的許可集。
- 暫時 IAM 使用者許可 – IAM 使用者或角色可以擔任 IAM 角色來暫時針對特定任務採用不同的許可。
- 跨帳戶存取權 – 您可以使用 IAM 角色，允許不同帳戶中的某人 (信任的委託人)存取您帳戶中的資源。角色是授予跨帳戶存取權的主要方式。但是，針對某些 AWS 服務，您可以將政策直接連接到資源 (而非使用角色作為代理)。若要了解跨帳戶存取角色和資源型政策間的差異，請參閱《IAM 使用者指南》中的 IAM 角色與資源類型政策的差異。

- 跨服務存取 – 有些 AWS 服務 會使用其他 AWS 服務 中的功能。例如，當您在服務中進行呼叫時，該服務通常會在 Amazon EC2 中執行應用程式或將物件儲存在 Amazon Simple Storage Service (Amazon S3) 中。服務可能會使用呼叫主體的許可、使用服務角色或使用服務連結角色來執行此作業。
  - 轉發存取工作階段 (FAS)：當您使用 IAM 使用者或角色在 AWS 中執行動作時，系統會將您視為主體。當您使用某些服務時，您可能會執行一個動作，而該動作之後會在不同的服務中啟動另一個動作。FAS 使用主體的許可呼叫 AWS 服務，搭配請求 AWS 服務 以向下游服務發出請求。只有在服務收到需要與其他 AWS 服務 或資源互動才能完成的請求之後，才會提出 FAS 請求。在此情況下，您必須具有執行這兩個動作的許可。如需提出 FAS 請求時的政策詳細資訊，請參閱《轉發存取工作階段》。
  - 服務角色：服務角色是服務擔任的 IAM 角色，可代表您執行動作。IAM 管理員可以從 IAM 內建立、修改和刪除服務角色。如需詳細資訊，請參閱《IAM 使用者指南》中的建立角色以委派許可給 AWS 服務 服務。
  - 服務連結角色 – 服務連結角色是一種連結到 AWS 服務 的服務角色類型。服務可以擔任代表您執行動作的角色。服務連結角色會顯示在您的 AWS 帳戶 中，並由該服務所擁有。IAM 管理員可以檢視，但不能編輯服務連結角色的許可。
- 在 Amazon EC2 上執行的應用程式 – 針對在 EC2 執行個體上執行並提出 AWS CLI 和 AWS API 請求的應用程式，您可以使用 IAM 角色來管理暫時憑證。這是在 EC2 執行個體內儲存存取金鑰的較好方式。如需指派 AWS 角色給 EC2 執行個體並提供其所有應用程式使用，您可以建立連接到執行個體的執行個體設定檔。執行個體設定檔包含該角色，並且可讓 EC2 執行個體上執行的程式取得暫時憑證。如需詳細資訊，請參閱《IAM 使用者指南》中的利用 IAM 角色來授予許可給 Amazon EC2 執行個體上執行的應用程式。

如需了解是否要使用 IAM 角色或 IAM 使用者，請參閱《IAM 使用者指南》中的建立 IAM 角色 (而非使用者) 的時機。

AWS Device Farm 如何與 IAM 搭配使用

在您使用 IAM 管理 Device Farm 的存取權限之前，您應該瞭解哪些 IAM 功能可用於 Device Farm。若要取得 Device Farm 和其他AWS服務如何與 IAM 搭配運作的高階檢視，請參閱 IAM 使用者指南中的與 IAM 搭配使用的AWS服務。

主題

- Device Farm 身分型原則
- Device Farm 資源型原則
- 存取控制清單

- 根據 Device Farm 標籤授權
- Device Farm IAM 角色

Device Farm 身分型原則

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、資源和條件金鑰。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哪些內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

JSON 政策的 `Action` 元素描述您可以用來允許或拒絕政策中存取的動作。政策動作的名稱通常會和相關聯的 AWS API 操作相同。有一些例外狀況，例如沒有相符的 API 操作的僅限許可動作。也有一些操作需要政策中的多個動作。這些額外的動作稱為相依動作。

政策會使用動作來授與執行相關聯操作的許可。

Device Farm 列中的原則動作會在動作之前使用下列前置詞：`devicefarm:`例如，要授予某人使用「Device Farm」桌面瀏覽器測試 `CreateTestGridUrl` API 操作啟`devicefarm:CreateTestGridUrl`動 Selenium 會話的權限，您可以在策略中包含該操作。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，說明您可以使用此服務執行的工作。

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 列動作清單，請參閱 IAM 服務授權參考AWS Device Farm中定義的動作。

資源

管理員可以使用 AWS JSON 政策來指定誰可以存取哪些內容。也就是說，哪個主體在什麼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。陳述式必須包含 Resource 或 NotResource 元素。最佳實務是使用其 Amazon Resource Name (ARN) 來指定資源。您可以針對支援特定資源類型的動作 (稱為資源層級許可) 來這麼做。

對於不支援資源層級許可的動作 (例如列出作業)，請使用萬用字元 (*) 來表示陳述式適用於所有資源。

```
"Resource": "*"
```

Amazon EC2 實例資源具有以下 ARN：

```
arn:${Partition}:ec2:${Region}:${Account}:instance/${InstanceId}
```

如需 ARN 格式的詳細資訊，請參閱 Amazon Resource Name (ARN) 和 AWS 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/*"
```

某些 Device Farm 動作 (例如用來建立資源的動作) 無法在資源上執行。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 列資源類型及其 ARN 的清單，請參閱 IAM 服務授權參考AWS Device Farm中所定義的資源類型。若要了解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AWS Device Farm中定義的動作。

條件索引鍵

管理員可以使用 AWS JSON 政策來指定誰可以存取哪些內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

Condition 元素 (或 Condition 區塊)可讓您指定使陳述式生效的條件。Condition 元素是選用項目。您可以建立使用條件運算子的條件運算式 (例如等於或小於)，來比對政策中的條件和請求中的值。

若您在陳述式中指定多個 Condition 元素，或是在單一 Condition 元素中指定多個索引鍵，AWS 會使用邏輯 AND 操作評估他們。若您為單一條件索引鍵指定多個值，AWS 會使用邏輯 OR 操作評估條件。必須符合所有條件，才會授與陳述式的許可。

您也可以在指定條件時使用預留位置變數。例如，您可以只在使用者使用其 IAM 使用者名稱標記時，將存取資源的許可授與該 IAM 使用者。如需更多資訊，請參閱《IAM 使用者指南》中的 IAM 政策元素：變數和標籤。

AWS 支援全域條件索引鍵和服務特定的條件索引鍵。若要查看 AWS 全域條件索引鍵，請參閱《IAM 使用者指南》中的 AWS 全域條件內容索引鍵。

Device Farm 會定義自己的一組條件金鑰，也支援使用某些全域條件金鑰。若要查看 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 列條件金鑰清單，請參閱 IAM 服務授權參考AWS Device Farm中的條件金鑰。若要了解可以使用條件金鑰的動作和資源，請參閱 IAM 服務授權參考AWS Device Farm中由定義的動作。

範例

若要檢視 Device Farm 識別型原則的範例，請參閱。AWS Device Farm 身分型政策範例

Device Farm 資源型原則

Device Farm 不支援資源型政策。

存取控制清單

Device Farm 列不支援存取控制清單 (ACL)。

根據 Device Farm 標籤授權

您可以將標籤附加至 Device Farm 資源，或將要求中的標籤傳遞至 Device Farm 若要根據標籤控制存取，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，請參閱裝置陣列中的標記。

若要檢視身分型原則範例，以根據該資源上的標籤來限制存取資源，請參閱根據標籤檢視 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

Device Farm 支援使用臨時認證。

您可以使用臨時登入資料以同盟登入，以擔任 IAM 角色或跨帳戶角色。您可以透過呼叫AssumeRole或等 AWS STS API 作業來取得臨時安全登入資料GetFederationToken。

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 會在裝置伺服器 Device Farm 桌面瀏覽器測試功能中使用服務連結 如需這些角色的相關資訊，請參閱開發人員指南中的在裝置伺服器陣列桌面瀏覽器測試中使用服務連結

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附加到 AWS 身分或資源，在 AWS 中控制存取。政策是 AWS 中的一個物件，當其和身分或資源建立關聯時，便可定義其許可。AWS 會在主體 (使用者、根使用者或角色工作階段)發出請求時評估這些政策。政策中的許可，決定是否允許或拒絕請求。大部分政策以 JSON 文件形式儲存在 AWS 中。如需 JSON 政策文件結構和內容的詳細資訊，請參閱《IAM 使用者指南》中的 JSON 政策概觀。

管理員可以使用 AWS JSON 政策來指定誰可以存取哪些內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

預設情況下，使用者和角色沒有許可。若要授與使用者對其所需資源執行動作的許可，IAM 管理員可以建立 IAM 政策。然後，管理員可以將 IAM 政策新增至角色，使用者便能擔任這些角色。

IAM 政策定義該動作的許可，無論您使用何種方法來執行操作。例如，假設您有一個允許 `iam:GetRole` 動作的政策。具備該政策的使用者便可以從 AWS Management Console、AWS CLI 或 AWS API 取得角色資訊。

身分型政策

身分型政策是可以附加到身分 (例如 IAM 使用者、使用者群組或角色)的 JSON 許可政策文件。這些政策可控制身分在何種條件下能對哪些資源執行哪些動作。若要了解如何建立身分類型政策，請參閱《IAM 使用者指南》中的建立 IAM 政策。

身分型政策可進一步分類成內嵌政策或受管政策。內嵌政策會直接內嵌到單一使用者、群組或角色。受管政策則是獨立的政策，您可以將這些政策附加到 AWS 帳戶 中的多個使用者、群組和角色。受管政策包含 AWS 管理政策和客戶管理政策。如需瞭解如何在受管政策及內嵌政策間選擇，請參閱 IAM 使用者指南中的在受管政策和內嵌政策間選擇。

下表概述 Device Farm 列 AWS 受管政策。

變更	描述	日期
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 支援其他較少見的政策類型。這些政策類型可設定較常見政策類型授與您的最大許可。

- 許可界限 – 許可範圍是一種進階功能，可供您設定身分型政策能授予 IAM 實體 (IAM 使用者或角色) 的最大許可。您可以為實體設定許可界限。所產生的許可會是實體的身分型政策和其許可界限的交集。會在 `Principal` 欄位中指定使用者或角色的資源型政策則不會受到許可界限限制。所有這類政策中的明確拒絕都會覆寫該允許。如需許可範圍的更多相關資訊，請參閱《IAM 使用者指南》中的 IAM 實體許可範圍。
- 服務控制政策 (SCP) – SCP 是 JSON 政策，可指定 AWS Organizations 中組織或組織單位 (OU) 的最大許可。AWS Organizations 服務可用來分組和集中管理您企業所擁有的多個 AWS 帳戶。若您啟用組織中的所有功能，您可以將服務控制政策 (SCP) 套用到任何或所有帳戶。SCP 會限制成員帳戶中實體的許可，包括每個 AWS 帳戶根使用者。如需組織和 SCP 的更多相關資訊，請參閱《AWS Organizations 使用者指南》中的 SCP 運作方式。
- 工作階段政策 – 工作階段政策是一種進階政策，您可以在透過編寫程式的方式建立角色或聯合使用者的暫時工作階段時，作為參數傳遞。所產生工作階段的許可會是使用者或角色的身分型政策和工作階段政策的交集。許可也可以來自資源型政策。所有這類政策中的明確拒絕都會覆寫該允許。如需更多資訊，請參閱《IAM 使用者指南》中的工作階段政策。

多種政策類型

將多種政策類型套用到請求時，其結果形成的許可會更為複雜、更加難以理解。如需瞭解 AWS 在涉及多種政策類型時如何判斷是否允許一項請求，請參閱 IAM 使用者指南中的政策評估邏輯。

AWS Device Farm 身分型政策範例

依預設，IAM 使用者和角色沒有建立或修改 Device Farm 資源的權限。他們也無法使用 AWS Management Console、AWS CLI 或 AWS API 執行任務。IAM 管理員必須建立 IAM 政策，授予使用者和角色在指定資源上執行特定 API 操作的所需許可。管理員接著必須將這些政策連接至需要這些許可的 IAM 使用者或群組。

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

以身分識別為基礎的原則會決定某人是否可以建立、存取或刪除您帳戶中的 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 受管政策並朝向最低權限許可的目標邁進：如需開始授予許可給使用者和工作負載，請使用 AWS 受管政策，這些政策會授予許可給許多常用案例。它們可在您的 AWS 帳戶 中使用。我們建議您定義特定於使用案例的 AWS 客戶管理政策，以便進一步減少許可。如需更多資訊，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。
- 套用最低許可許可 – 設定 IAM 政策的許可時，請僅授予執行任務所需的權限。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。您也可以使用條件來授予對服務動作的存取權，前提是透過特定 AWS 服務 (例如 AWS CloudFormation)使用條件。如需更多資訊，請參閱《IAM 使用者指南》中的 IAM JSON 政策元素：條件。
- 使用 IAM Access Analyzer 驗證 IAM 政策，確保許可安全且可正常運作 – IAM Access Analyzer 驗證新政策和現有政策，確保這些政策遵從 IAM 政策語言 (JSON) 和 IAM 最佳實務。IAM Access Analyzer 提供 100 多項政策檢查及切實可行的建議，可協助您編寫安全且實用的政策。如需更多資訊，請參閱 IAM 使用者指南中的 IAM Access Analyzer 政策驗證。
- 需要多重要素驗證 (MFA)：如果存在需要 AWS 帳戶中 IAM 使用者或根使用者的情況，請開啟 MFA 提供額外的安全性。如需在呼叫 API 操作時請求 MFA，請將 MFA 條件新增至您的政策。如需更多資訊，請參閱 IAM 使用者指南中的設定 MFA 保護的 API 存取。

有關 IAM 中最佳實務的更多相關資訊，請參閱 IAM 使用者指南中的 IAM 最佳安全實務。

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上，或是使用 AWS CLI 或 AWS API 透過編寫程式的方式完成此動作的許可。

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

存取一個 Device Farm 桌面瀏覽器測試專案

在此範例中，您想要授與AWS帳戶中的 IAM 使用者存取其中一個 Device Farm Destktop 瀏覽器測試專案的存取權。`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。

```
{
```

```
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"GetTestGridProject",
         "Effect":"Allow",
         "Action":[
            "devicefarm:GetTestGridProject"
         ],
         "Resource":"arn:aws:devicefarm:us-west-2:111122223333:testgrid-
project:123e4567-e89b-12d3-a456-426655441111"
      },
      {
         "Sid":"ViewProjectInfo",
         "Effect":"Allow",
         "Action":[
            "devicefarm:ListTestGridSessions",
            "devicefarm:ListTestGridSessionActions",
            "devicefarm:ListTestGridSessionArtifacts"
         ],
         "Resource":"arn:aws:devicefarm:us-west-2:111122223333:testgrid-*:123e4567-
e89b-12d3-a456-426655441111/*"
      }
   ]
}
```

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。以下 IAM 策略概述了一個最小策略，允許 CI 運行者通過創建上傳並使用它來安排測試運行來啟動新的 Device Farm 本機應用程序測試的測試：

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "$id":"scheduleTestRuns",
         "effect":"Allow",
         "Action": [ "devicefarm:CreateUpload","devicefarm:ScheduleRun" ],
         "Resource": [
            "arn:aws:devicefarm:us-west-2:111122223333:project:123e4567-e89b-12d3-
a456-426655440000",
            "arn:aws:devicefarm:us-west-2:111122223333:*:123e4567-e89b-12d3-
a456-426655440000/*",
            ]
```

```
          }
    ]
}
```

根據標籤檢視 Device Farm 桌面瀏覽器測試專案

您可以使用身分型原則中的條件，根據標籤來控制 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。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ListTestGridProjectSessions",
            "Effect": "Allow",
            "Action": [
              "devicefarm:ListTestGridSession*",
              "devicefarm:GetTestGridSession",
              "devicefarm:ListTestGridProjects"
              ],
            "Resource": [
               "arn:aws:devicefarm:us-west-2:testgrid-project:*/*"
               "arn:aws:devicefarm:us-west-2:testgrid-session:*/*"
               ],
            "Condition": {
               "StringEquals": {"aws:TagKey/Owner":"${aws:username}"}
            }
        }
    ]
}
```

您可以將此政策連接到您帳戶中的 IAM 使用者。如果命名為的使用者`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。

疑難排解 AWS Device Farm 身分和存取

使用下列資訊可協助您診斷並修正使用 Device Farm 列和 IAM 時可能會遇到的常見問題。

我沒有在 Device Farm 中執行動作的授權

如果在 AWS Management Console中收到錯誤，告知您未獲授權不得執行某動作，您必須聯絡管理員以取得協助。您的管理員是提供您使用者名稱和密碼的人員。

當 IAM 使用者嘗試使用主控台檢視執行的詳細資料mateojackson，但沒有devicefarm:GetRun權限時，就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。

我沒有授權執行 iam：PassRole

如果您收到未獲授權執行iam:PassRole動作的錯誤訊息，則必須更新原則，以允許您將角色傳遞至 Device Farm。

有些 AWS 服務 允許您傳遞現有的角色至該服務，而無須建立新的服務角色或服務連結角色。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 使用者marymajor嘗試使用主控台在 Device Farm 列中執行動作時，就會發生下列範例錯誤。但是，該動作要求服務具備服務角色授與的許可。Mary 沒有將角色傳遞至該服務的許可。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

在這種情況下，Mary 的政策必須更新，允許她執行 iam:PassRole 動作。

如需任何協助，請聯絡您的 AWS 管理員。您的管理員提供您的登入憑證。

我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。透過執行此操作，可能會讓他人永久存取您的 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。

我是系統管理員，想要允許其他人存取 Device Farm

若要允許其他人存取 Device Farm，您必須為需要存取的人員或應用程式建立 IAM 實體 (使用者或角色)。他們將使用該實體的憑證來存取 AWS。然後，您必須將原則附加至實體，以便在 Device Farm 中授與其正確權限。

若要立即開始使用，請參閱《IAM 使用者指南》中的建立您的第一個 IAM 委派使用者及群組。

我想要允許AWS帳戶以外的人員存取我的 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的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：

- 若要瞭解 Device Farm 是否支援這些功能，請參閱AWS Device Farm 如何與 IAM 搭配使用。
- 如需了解如何存取您擁有的所有 AWS 帳戶 所提供的資源，請參閱《IAM 使用者指南》中的將存取權提供給您所擁有的另一個 AWS 帳戶 中的 IAM 使用者。
- 如需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擁有的 AWS 帳戶。
- 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)。
- 若要了解使用角色和資源型政策進行跨帳戶存取之間的差異，請參閱《IAM 使用者指南》中的 IAM 角色與資源型政策的差異。

AWS Device Farm 的合規驗證

在多個 AWS 合規計劃中，第三方稽核人員會評估 AWS Device Farm 的安全與合規。這些包括 SOC、PCI、FedRAMP、HIPAA 和其他。任何 AWS 合規計劃皆不包含 AWS Device Farm。

如需特定合規計劃範圍內的 AWS 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 合規計畫。

您可使用 AWS Artifact 下載第三方稽核報告。如需詳細資訊，請參閱下載 AWS Artifact 中的報告。

使用 Device Farm 時，您的合規責任取決於資料的敏感度、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遵循法規：

- 安全與合規 Quick Start 指南：這些部署指南就在 AWS 上部署以安全及合規為重心之基準環境，討論架構考量並提供相關步驟。
- AWS 合規資源 – 這組手冊和指南可能適用於您的產業和位置。
- 使用規則評估資源在AWS Config開發者指南—AWS Config評估您的資源組態是否符合內部實務、產業準則和法規。
- AWS Security Hub – 此 AWS 服務可供您檢視 AWS 中的安全狀態，可助您檢查是否符合安全產業標準和最佳實務。

AWS Device Farm 中的資料保護

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 AWS 雲端 的全球基礎設施。您負責維護在此基礎設施上託管內容的控制權。您也必須負責您所使用 AWS 服務 的安全組態和管理任務。如需資料隱私權的相關資訊，請參閱資料隱私權常見問答集。如需有關歐洲資料保護的相關資訊，請參閱 AWS 安全性部落格上的 AWS 共同的責任模型和 GDPR 部落格文章。

基於資料保護目的，建議您使用 AWS IAM Identity Center 或 AWS Identity and Access Management (IAM) 保護 AWS 帳戶 憑證，並設定個人使用者。如此一來，每個使用者都只會獲得授與完成其任務所必須的許可。我們也建議您採用下列方式保護資料：

- 每個帳戶均要使用多重要素驗證 (MFA)。
- 使用 SSL/TLS 與 AWS 資源通訊。我們需要 TLS 1.2 並建議使用 TLS 1.3。
- 使用 AWS CloudTrail 設定 API 和使用者活動日誌記錄。
- 使用 AWS 加密解決方案，以及 AWS 服務 內的所有預設安全控制項。

- 使用進階的受管安全服務 (例如 Amazon Macie)，協助探索和保護儲存在 Amazon S3 的敏感資料。
- 如果您在透過命令列介面或 API 存取 AWS 時，需要 FIPS 140-2 驗證的加密模組，請使用 FIPS 端點。如需 FIPS 和 FIPS 端點的相關資訊，請參閱聯邦資訊處理標準 (FIPS) 140-2 概觀。

我們強烈建議您絕對不要將客戶的電子郵件地址等機密或敏感資訊，放在標籤或自由格式的文字欄位中，例如名稱欄位。這包括當您使用主控台、API 或 AWS SDK AWS 服務 使用 Device Farm 或其他裝置時。AWS CLI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) 要求，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。如需有關如何登入 HTTPS 要求的詳細資訊AWS，請參閱AWS一般參考中的簽署 AWS API 要求。

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

裝置伺服器陣列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。

裝置伺服器陣列的實體行動裝置測試資料在靜態時未加密。

資料保留

Device Farm 中的資料會保留有限的時間。保留期過期後，資料會從 Device Farm 支援儲存體中移除，但會保留所有中繼資料 (ARN、上傳日期、檔案名稱等) 以供 future 使用。下表列出各種內容類型的保留期間。

內容類型	保留期間 (天)
上傳的應用程式	30
上傳的測試套件	30
Logs (日誌)	400
影片和其他成品	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 中的資料會根據使用的功能而有不同的管理方式。本節說明在您使用 Device Farm 時和之後如何管理資料。

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。

此功能目前支援測試期間產生的成品靜態加密。

物理設備測試

下列各節提供使用裝置伺服器陣列之後清理或銷毀裝置所AWS需步驟的相關資訊。

裝置伺服器陣列的實體行動裝置測試資料在靜態時未加密。

公用裝置叢集

測試執行完成後，Device Farm 會在公用裝置叢集中的每個裝置上執行一系列清理工作，包括解除安裝應用程式。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，數據可能會在會話之間持續存在，特別是如果您在應用程序的上下文之外使用設備系統。因此，由於 Device Farm 會擷取您使用每部裝置期間發生的影片和活動記錄，因此建議您不要在自動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料。

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 在 AWS 裝置伺服器陣列中使用私有裝置。

金鑰管理

目前，Device Farm 不提供任何外部金鑰管理來加密資料 (無論是靜態或傳輸中)。

網際網路流量隱私權

Device Farm 只能針對私有裝置設定，以使用 Amazon VPC 端點連線到中AWS的資源。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 都會將您的流量與裝 Device Farm 網路中的其他使用者隔離。

因為您在 AWS 網路外的連線無法保證受到保護或是安全的，所以您必須負責保護您應用程式所建立的所有網際網路連線。

AWS Device Farm 中的恢復能力

AWS 全球基礎設施是以 AWS 區域與可用區域為中心建置的。AWS區域提供多個分開且隔離的實際可用區域，並以低延遲、高輸送量和高度備援聯網功能相互連結。透過可用區域，您可以設計與操作的應用程式和資料庫，在可用區域之間自動容錯移轉而不會發生中斷。可用區域的可用性、容錯能力和擴充能力，均較單一或多個資料中心的傳統基礎設施還高。

如需有關 AWS 區域與可用區域的詳細資訊，請參閱 AWS 全球基礎設施。

因為裝置伺服器陣列在us-west-2僅限區域，我們強烈建議您實作備份和復原程序。裝置伺服器陣列不應該是任何上傳內容的唯一來源。

裝置伺服器陣列不保證公用裝置的可用性。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。

AWS Device Farm 中的基礎設施安全

作為一種受管服務，AWS Device Farm 受 AWS 全球網路安全保護。如需有關 AWS 安全服務以及 AWS 如何保護基礎設施的詳細資訊，請參閱 AWS 雲端安全。若要使用基礎設施安全性的最佳實務來設計您的 AWS 環境，請參閱安全性支柱 AWS 架構良好的框架中的基礎設施保護。

您使用AWS透過網路存取裝置伺服器陣列的已發佈 API 呼叫。用戶端必須支援下列項目：

- 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，請求必須使用存取索引鍵 ID 和與 IAM 主體相關聯的私密存取索引鍵來簽署。或者，您可以使用 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架構安全

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。

公用裝置是共用的，而裝置伺服器陣列會盡最大努力保護裝置的安全。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。

私人裝置只能由人存取AWS明確授權這樣做的帳戶。裝置伺服器陣列會將這些裝置與其他裝置實際隔離，並將它們保留在不同的網路上。

在私有受管裝置上，可將測試設定為使用 Amazon VPC 端點來保護進出連線的安全AWS帳戶。

桌上型瀏覽器測試的基礎架構

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。Selenium 執行個體不能在沒有中間第三方的情況下，在 AWS 外部進行交叉通訊。

硒的所有流量WebDriver控制器必須透過以下方式產生的 HTTPS 端點建立`createTestGridUrl`。

桌面瀏覽器測試功能目前不支援 Amazon VPC 端點組態。您必須負責確定每個裝置伺服器陣列測試執行個體都能安全存取其測試的資源。

設備場中的配置漏洞分析和管理

Device Farm 可讓您執行未由廠商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軟體，但不能保證實體裝置上的任何特定軟體版本為最新版本，其設計允許使用潛在易受攻擊的軟體。

例如，如果在運行 Android 4.4.2 的設備上執行測試，則設備場不保證該設備已根據安卓系統中的漏洞被稱為StageFright。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。

私有裝置會依照您的 AWS 合約進行維護。

裝置伺服器陣列盡最大努力，以防止客戶應用程式遭受諸如此類動作生根或者越獄。裝置伺服器陣列會移除已從公用集區隔離的裝置，直到手動檢閱這些裝置為止。

您有責任保持您在測試中使用的任何庫或軟件版本，例如 Python 輪子和 Ruby 寶石，是最新的。裝置伺服器陣列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：

- 有關如何保護 Ruby 寶石的資訊，請參閱安全性做法上的RubyGems網站。
- 如需有關 Pipenv 使用並由 Python 封裝授權單位認可用來掃描相依性圖表中是否有已知弱點的安全套件的資訊，請參閱安全漏洞的檢測上GitHub。
- 如需開放 Web 應用程式安全性專案 (OWASP) Maven 相依性檢查程式的相關資訊，請參閱OWASPDependencyCheck在 OWASP 網站上。

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。

裝置陣列中的事件回應

裝置伺服器陣列會持續監控裝置中是否有可能指出安全性問題 如果 AWS 發現客戶資料 (例如測試結果或寫入公有裝置的檔案) 可供其他客戶存取，AWS 會根據 AWS 服務通用的標準事件警示和報告政策，聯絡受影響的客戶。

裝置伺服陣列中的記錄和監視

此服務支持AWS CloudTrail，這是記錄的服務AWS呼叫您的AWS 帳戶並將日誌檔交付到 Amazon S3 儲存貯體。通過使用收集的信息CloudTrail，您可以確定哪些請求已成功發出AWS 服務，誰提出了請求，當它被提出，等等。若要深入瞭解CloudTrail，包括如何開啟和尋找您的記錄檔，請參閱AWS CloudTrail使用者指南。

有關使用的信息CloudTrail使用裝置農場，請參閱使用記錄 AWS 裝置伺服器陣列 API 呼叫AWS CloudTrail。

裝置伺服器陣列的安全性最佳

Device Farm 提供許多安全性功能，可在您開發和實作自己的安全性原則時考量。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。如需有關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，請參閱使用自訂測試環境。

AWS Device Farm 的限制

下列清單說明目前的 AWS Device Farm 列限制：

- 您可以上傳的應用程式檔案大小上限為 4 GB。
- 測試執行中可以包含的裝置數量沒有限制。不過，在測試回合期間，Device Farm 將同時測試的裝置數目上限為五個。(可依請求提高此數量。)
- 您可以排定的執行次數沒有限制。
- 遠端存取工作階段持續時間有 150 分鐘的限制。
- 自動化測試執行持續時間有 150 分鐘的限制。
- 執行中工作 (包括帳戶中擱置的佇列工作) 數目上限為 250 個。這是一個軟限制。
- 您可以在測試執行中包含的裝置數目沒有限制。您可以在任何給定時間並行執 parallel 測試的裝置或作業數量等於您的帳戶層級並行運作。AWS Device Farm 上計量使用的預設帳戶層級並行為 5。您可以根據使用案例要求將此數字提高到特定臨界值。未計量使用的預設帳戶層級並行性等於您為該平台訂閱的插槽數量。

AWS 裝置農場的工具和外掛程式

本節包含使用 AWS 裝置伺服器陣列工具和外掛程式的連結和資訊。你可以找到設備農場插件AWS 實驗室GitHub。

如果您是 Android 開發人員，我們也有AWS 裝置農場範例應用程式GitHub。您可以使用應用程式和範例測試作為自己裝置伺服器陣列測試指令碼的參考。

主題

- AWS 設備農場與詹金斯 CI 插件集成
- AWS 設備農場搖籃插件

AWS 設備農場與詹金斯 CI 插件集成

此外掛程式從您自己的 Jenkins 持續整合 (CI) 伺服器提供 AWS 裝置農場功能。如需詳細資訊，請參閱 Jenkins (軟體)。

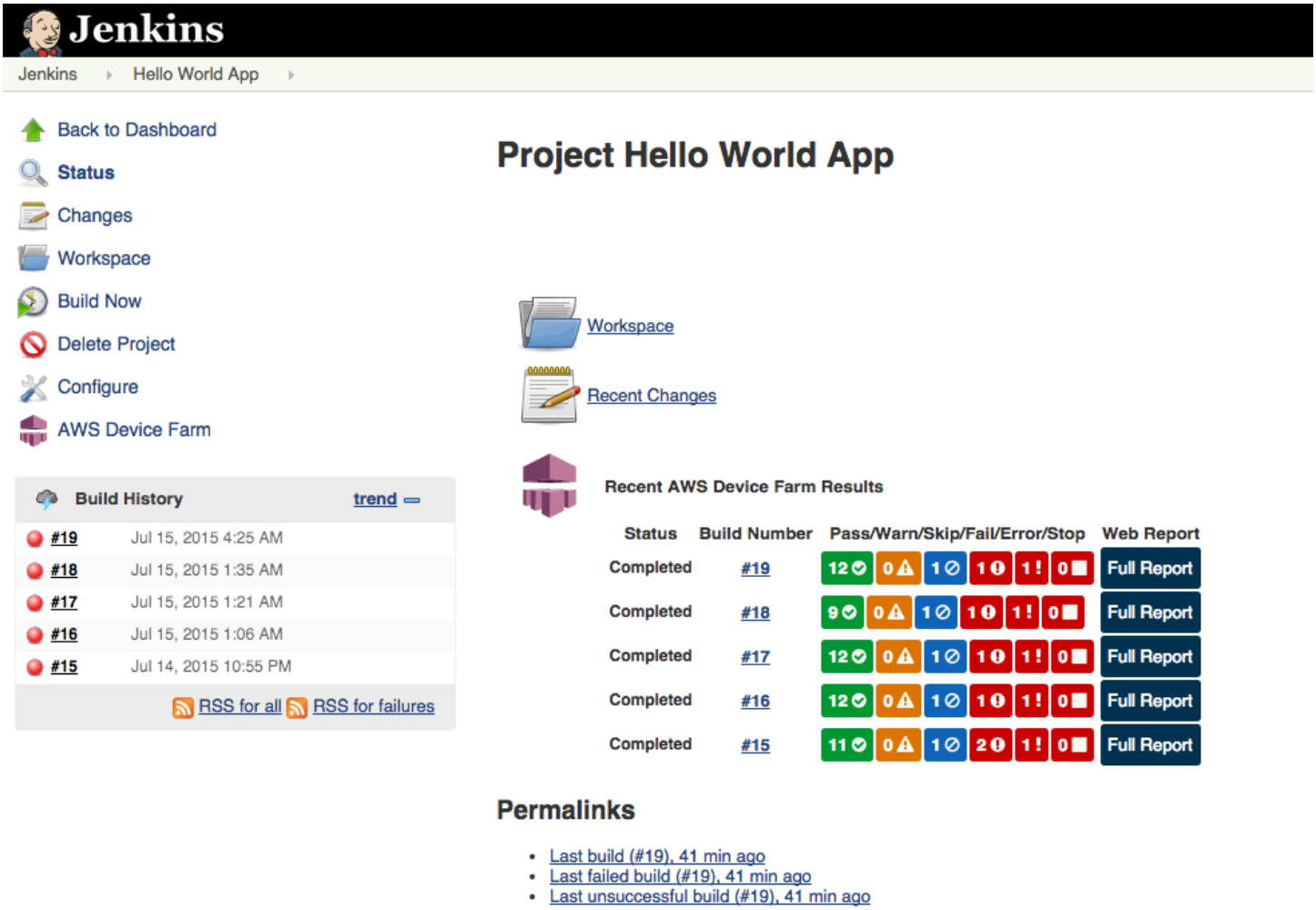
> **Note**
>
> 要下載詹金斯插件，請轉到GitHub並按照中的說明進行操作步驟 1：安裝插件。

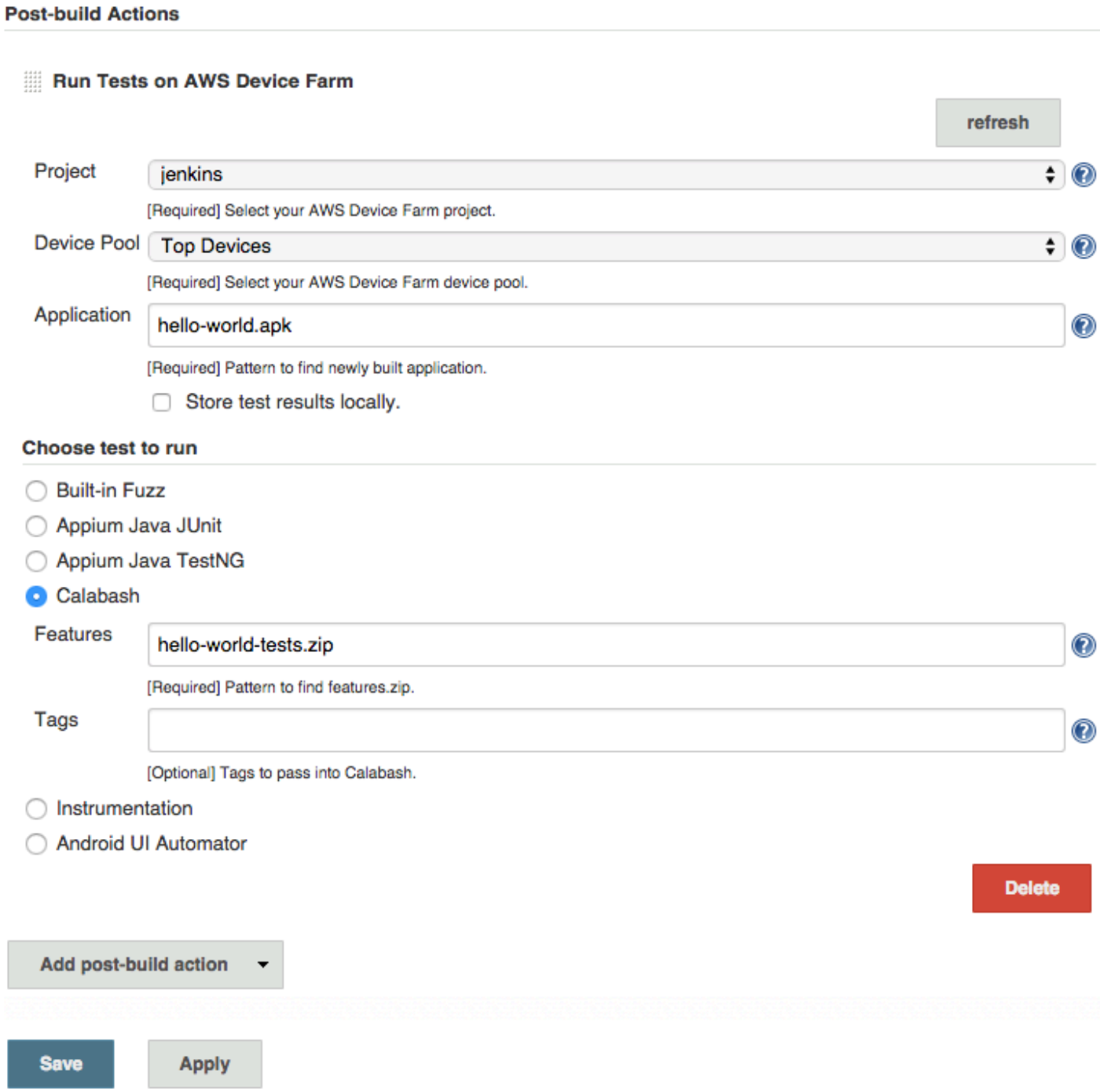
本節包含一系列程序，以便在 AWS 裝置伺服器陣列中設定和使用 Jenkins CI 外掛程式。

主題

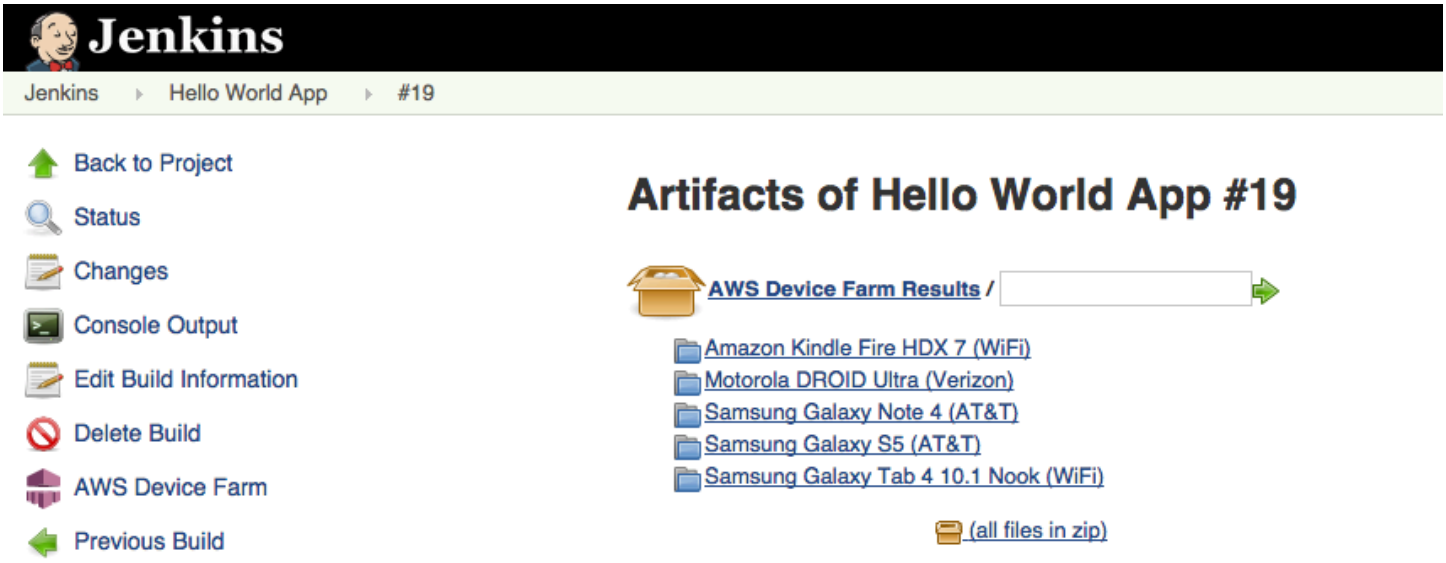
- 步驟 1：安裝插件
- 步驟 2：建立AWS Identity and Access Management用戶為您詹金斯 CI 插件
- 步驟 3：第一次設定指示
- 第 4 步：在詹金斯工作中使用插件
- 相依性

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)：



步驟 1：安裝插件

安裝適用於 AWS 裝置農場的 Jenkins 持續整合 (CI) 外掛程式有兩個選項。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 裝置伺服器陣列外掛程式。
4. 確保外掛程式是由 `Jenkins` 使用者擁有。
5. 重新啟動 Jenkins。

下載插件

1. 下載hpi直接從檔案http://updates.jenkins-ci.org/latest/aws-device-farm.hpi。
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：建立AWS Identity and Access Management用戶為您詹金斯 CI 插件

我們建議您不要使用您的AWSroot 帳戶以存取裝置陣列。相反，請創建一個新的AWS Identity and Access Management(IAM) 使用者 (或使用現有的 IAM 使用者)AWS帳戶，然後使用該 IAM 使用者存取裝置伺服器陣列。

若要建立新的 IAM 使用者，請參閱建立 IAM 使用者 (AWS Management Console)。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。

授與 IAM 使用者存取裝置伺服器陣列的權限

若要授與 IAM 使用者存取裝置伺服器陣列的權限，請在 IAM 中建立新的存取政策，然後按如下方式將存取政策指派給 IAM 使用者。

> **Note**
>
> 該AWS您用來完成下列步驟的根帳戶或 IAM 使用者必須具有建立下列 IAM 政策的權限，並將其附加至 IAM 使用者。如需詳細資訊，請參閱使用政策

在 IAM 中建立存取政策

1. 前往網址 https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。
2. 選擇 Policies (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 Create Your Own Policy (建立您自己的政策) 旁邊，選擇 Select (選取)。
5. 針對 Policy Name (政策名稱)，輸入政策的名稱 (例如，**`AWSDeviceFarmAccessPolicy`**)。
6. 對於描述」中，輸入可協助您將此 IAM 使用者與 Jenkins 專案建立關聯的說明。
7. 針對 Policy Document (政策文件)，輸入下列聲明：

```
{
    "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. 前往網址 https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。
2. 選擇 Users (使用者)。
3. 選擇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：第一次設定指示

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 裝置農場頭。
5. 從 步驟 2：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 Save (儲存)。

第 4 步：在詹金斯工作中使用插件

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並選擇在 AWS 裝置伺服器陣列執行測試。
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 Save (儲存)。

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。

AWS 設備農場搖籃插件

此外掛程式提供 AWS 裝置農場與安卓工作室中的搖籃建置系統整合。如需詳細資訊，請參閱 Gradle。

> **Note**
>
> 要下載搖籃插件，請轉到GitHub並按照中的說明進行操作構建設備農場搖籃插件。

設備農場搖籃插件提供設備農場功能從您的 Android 工作室環境。您可以在設備農場託管的真實 Android 手機和平板電腦上開始測試。

本節包含一系列設置和使用設備農場 Gradle 插件的程序。

主題

步驟 1：建置 AWS 裝置農場搖籃外掛程式

此外掛程式提供 AWS 裝置農場與安卓工作室中的搖籃建置系統整合。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 裝置農場搖籃外掛程式。

建置外掛程式

1. 前往GitHub並克隆存儲庫。
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 裝置農場搖籃外掛程式

步驟 2：設定 AWS 裝置農場搖籃外掛程式

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：構建設備農場搖籃插件。

設定 AWS 裝置農場搖籃外掛程式

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. 使用下列工作執行裝置伺服器陣列測試：`gradle devicefarmUpload`。

   構建輸出將打印出指向設備農場控制台的鏈接，您可以在其中監視測試執行。

後續步驟：[產生 IAM 使用者]

步驟 3：產生 IAM 使用者

AWS Identity and Access Management(IAM) 可協助您管理使用的許可和政策AWS資源。本主題將逐步引導您產生具有存取 AWS 裝置伺服器陣列資源許可的 IAM 使用者。

如果您尚未這樣做，請先完成步驟 1 和 2，然後再產生 IAM 使用者。

我們建議您不要使用您的AWSroot 帳戶以存取裝置伺服器陣列。而是建立新的 IAM 使用者 (或使用現有的 IAM 使用者)AWS帳戶，然後使用該 IAM 使用者存取裝置伺服器陣列。

> **Note**
>
> 該AWS您用來完成下列步驟的根帳戶或 IAM 使用者必須具有建立下列 IAM 政策的權限，並將其附加至 IAM 使用者。如需詳細資訊，請參閱使用政策。

使用 IAM 中的適當存取政策建立新使用者

1. 前往網址 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. 選擇點擊這裡它說，沒有內嵌政策可顯示。要創建一個，請單擊此處。
11. 在 Set Permissions (設定許可) 畫面上，選擇 Custom Policy (自訂政策)。
12. 選擇 Select (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。

```
{
    "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 Management Console)或者設定。

步驟 4：設定測試類型

根據預設，AWS Device Farm Gradle 外掛程式會執行使用適用於安卓和 AWS Device Farm 的儀器測試。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，請參閱在 AWS Device Farm 中使用測試類型。

如果您尚未這樣做，請在設定測試類型之前完成步驟 1 — 3。

> **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 提供了蘋果 Java JUnit 和 TestNG 為安卓系統的支持。

- 蘋果（在爪哇（JUnit）下）
- 蘋果（在爪哇下（TestNG））

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()
    }
```

內置：模糊

Device Farm 提供內建的模糊測試類型，可將使用者介面事件隨機傳送至裝置，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 內置：模糊（安卓和 iOS）。

檢測

Device Farm 為 Android 的儀器儀表提供支持 (JUnit, 咖啡, 機器人, 或任何基於儀器的測試). 如需詳細資訊，請參閱 使用適用於安卓和 AWS Device Farm 的儀器。

在 Gradle 中運行儀器測試時，Device Farm 使用從 AndroidTest 目錄生成的**.apk**文件作為測試的源。

```
    instrumentation {

        filter "test filter per developer docs" // optional

    }
```

相依性

執行時間

- 設備農場搖籃插件需要AWS行動 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

文件歷史紀錄

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
AL2 支援	Device Farm 現在支援安卓系統的 AL2 測試環境。了解有關 AL2 的更多信息。	2023 年 11 月 6 日
從標準移轉至自訂測試環境	更新了 2023 年 12 月標準模式測試的文檔棄用的遷移指南。	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到在 AWS、內部部署軟體或其他雲端供應商上託管的私有端點。了解更多關於 V PC-埃尼.	2023 年 5 月 15 日
北極星 UI 更新	Device Farm 主控台現在支援 Polaris 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
新的安全資訊和標記 AWS 資源的資訊。	為了確保 AWS 服務更容易且更全面，我們建立了新的安全部門。若要深入了解，請參閱AWS Device Farm 中的安全性 已新增有關 Device Farm 中標記的新章節。如需進一步了解標記的資訊，請參閱 裝置陣列中的標記。	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 設置設備農場搖籃插件。	2019 年 8 月 16 日

變更	描述	變更日期
使用 XCTest 測試執行的新需求	對於使用 XCTest 框架的測試運行，Device Farm 現在需要為測試構建的應用程序包。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 在 AWS Device Farm 中使用測試類型。	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 在 AWS Device Farm 中使用測試類型。	2018 年 10 月 2 日
自訂測試環境	透過自訂測試環境，您可以確保測試在本機環境中執行。Device Farm 現在提供即時記錄和視訊串流的支援，因此您可以立即取得在自訂測試環境中執行的測試的意見反應。進一步了解 使用自訂測試環境。	2018 年 8 月 16 日
Support 使用 Device Farm 做為 AWS CodePipeline 測試提供者	您現在可以在中AWS CodePipeline設定管道，在發行程序中使用 AWS Device Farm 執行作為測試動作。 CodePipeline 使您能夠快速鏈接存儲庫以構建和測試階段，以實現根據您的需求定制的持續集成系統。進一步了解 在中使用 AWS 裝置伺服器陣列CodePipeline測試階段。	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 在 AWS 裝置伺服器陣列中使用私有裝置。	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

變更	描述	變更日期
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專為解決您在 Device Farm 主控台中可能遇到的錯誤訊息而設計的程序，對測試套件上傳進行疑難排解。進一步了解 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 使用遠端存取。	2016 年 4 月 19 日
裝置插槽自助服務	您現在可以使用 AWS Management Console、AWS Command Line Interface 或 API 購買裝置插槽。進一步了解如何 在裝置伺服器陣列中購買裝置插槽。	2016 年 3 月 22 日
如何停止測試執行	您現在可以使用 AWS Management Console、AWS Command Line Interface 或 API 來停止測試執行。進一步了解如何 停止在 AWS Device Farm 中執行。	2016 年 3 月 22 日
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 XCTest UI 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 在 AWS Device Farm 中使用測試類型。	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 在 AWS Device Farm 中使用 Web 應用程式測試。	2015 年 11 月 19 日
AWS Device Farm 搖籃插件	進一步了解如何安裝和使用 設備農場搖籃插件。	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

變更	描述	變更日期
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 在 AWS Device Farm 中使用測試類型。	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發行版本。	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。