aws

Panduan Developerr

Device Farm AWS

Versi API 2015-06-23

Device Farm AWS: Panduan Developerr

Table of Contents

Apa itu AWS Device Farm?

Device Farm adalah layanan pengujian aplikasi yang dapat Anda gunakan untuk menguji dan berinteraksi dengan aplikasi Android, iOS, dan web di ponsel dan tablet fisik nyata yang di-host oleh Amazon Web Services (AWS).

Ada dua cara utama untuk menggunakan Device Farm:

- Akses perangkat dari jarak jauh dari komputer lokal Anda, baik secara interaktif di browser web Anda atau secara otomatis mengujinya menggunakan Appium dari klien lokal.
- Jalankan pengujian aplikasi secara otomatis menggunakan lingkungan eksekusi pengujian terkelola Device Farm.

> **Note**
>
> Device Farm hanya tersedia di wilayah `us-west-2` (Oregon).

Akses jarak jauh

Akses jarak jauh memungkinkan Anda berinteraksi dengan perangkat melalui browser web Anda secara real time. Akses jarak jauh juga memungkinkan Anda menjalankan pengujian Appium dari klien lokal Anda terhadap perangkat Device Farm jarak jauh menggunakan titik akhir Appium yang dikelola.

Interaksi real-time dengan perangkat dapat berguna untuk sejumlah skenario, seperti pengujian aplikasi manual, mereproduksi bug pada perangkat tertentu, memeriksa rendering visual aplikasi Anda pada berbagai jenis layar, serta urutan penginstalan dan peningkatan aplikasi. Titik akhir Appium yang dikelola sepenuhnya oleh Device Farm memungkinkan Anda mengembangkan, menguji, dan men-debug pengujian Appium Anda, memberikan umpan balik yang cepat.

Endpoint Appium mendukung bahasa apa pun pilihan Anda, IDE lokal apa pun, debugging langsung dengan breakpoint, video langsung dan log, dan alat seperti Appium Inspector. Anda dapat menjalankan tes sebanyak yang Anda suka pada perangkat yang sama selama sesi akses jarak jauh Anda dengan batas 150 menit.

Selama sesi akses jarak jauh, Device Farm mencatat detail tentang tindakan yang terjadi saat Anda berinteraksi dengan perangkat. Log dengan detail ini dan pengambilan video sesi diproduksi di akhir sesi.

Pengujian aplikasi otomatis

Device Farm memungkinkan Anda menjalankan pengujian otomatis pada beberapa perangkat secara paralel dengan mengunggah aplikasi dan pengujian. Pengujian dijalankan secara otomatis di lingkungan yang dikelola sepenuhnya pada host pengujian sehingga Anda dapat mengonfigurasi file spesifikasi pengujian. Lingkungan menggunakan host uji Device Farm, jadi Anda tidak perlu khawatir menyediakan infrastruktur Anda sendiri untuk menjalankan pengujian. Host dan perangkat pengujian dapat terhubung dengan aman ke VPC Anda untuk mengakses titik akhir pribadi Anda.

Saat pengujian selesai, laporan pengujian dibuat yang berisi hasil tingkat tinggi, log tingkat rendah, tangkapan layar, dan artefak pengujian Anda.

Device Farm mendukung pengujian aplikasi Android dan iOS native dan hybrid. Untuk informasi selengkapnya tentang jenis pengujian yang didukung, lihatUji kerangka kerja dan pengujian bawaan di AWS Device Farm.

Terminologi

Device Farm memperkenalkan istilah berikut yang menentukan cara informasi diatur:

perangkat kolam

> Kumpulan perangkat yang biasanya memiliki karakteristik serupa, seperti platform, pabrikan, atau model.

pekerjaan

> Permintaan Device Farm untuk menguji satu aplikasi terhadap satu perangkat. Pekerjaan berisi satu atau lebih suite.

pengukuran

> Mengacu pada penagihan untuk perangkat. Anda mungkin melihat referensi ke perangkat terukur atau perangkat yang tidak diukur dalam dokumentasi dan referensi API. Untuk informasi selengkapnya tentang harga, lihat Harga AWS Device Farm.

proyek

Ruang kerja logis yang berisi run, satu run untuk setiap pengujian satu aplikasi terhadap satu perangkat atau beberapa. Anda dapat menggunakan proyek untuk mengatur ruang kerja dengan cara apa pun yang Anda pilih. Misalnya, Anda dapat memiliki satu proyek per judul aplikasi atau satu proyek per platform. Anda dapat membuat proyek sebanyak yang Anda butuhkan.

laporan

Berisi informasi tentang proses, yang merupakan permintaan Device Farm untuk menguji satu aplikasi terhadap satu atau beberapa perangkat. Untuk informasi selengkapnya, lihat Laporan di AWS Device Farm.

run

Build spesifik aplikasi Anda, dengan serangkaian pengujian tertentu, akan dijalankan pada satu set perangkat tertentu. Lari menghasilkan laporan hasil. Lari berisi satu atau lebih pekerjaan. Untuk informasi selengkapnya, lihat Berjalan.

sesi

Interaksi real-time dengan perangkat fisik aktual melalui browser web Anda. Untuk informasi selengkapnya, lihat Sesi.

suite

Organisasi hierarkis tes dalam paket uji. Suite berisi satu atau lebih tes.

pengujian

Kasus uji individu dalam paket uji.

Untuk informasi selengkapnya tentang Device Farm, lihatKonsep.

Menyiapkan

Untuk menggunakan Device Farm, lihatMenyiapkan.

Menyiapkan AWS Device Farm

Sebelum Anda menggunakan Device Farm untuk pertama kalinya, Anda harus menyelesaikan tugas-tugas berikut:

Topik

Langkah 1: Mendaftar untuk AWS

Mendaftar untuk Amazon Web Services (AWS).

Jika Anda tidak memiliki Akun AWS, selesaikan langkah-langkah berikut untuk membuatnya.

Untuk mendaftar untuk Akun AWS

1. Buka https://portal.aws.amazon.com/billing/pendaftaran.
2. Ikuti petunjuk online.

   Bagian dari prosedur pendaftaran melibatkan menerima panggilan telepon atau pesan teks dan memasukkan kode verifikasi pada keypad telepon.

   Saat Anda mendaftar untuk sebuah Akun AWS, sebuah Pengguna root akun AWSdibuat. Pengguna root memiliki akses ke semua Layanan AWS dan sumber daya di akun. Sebagai praktik keamanan terbaik, tetapkan akses administratif ke pengguna, dan gunakan hanya pengguna root untuk melakukan tugas yang memerlukan akses pengguna root.

Langkah 2: Buat atau gunakan pengguna IAM di akun Anda AWS

Kami menyarankan Anda untuk tidak menggunakan akun AWS root Anda untuk mengakses Device Farm. Sebagai gantinya, buat pengguna AWS Identity and Access Management (IAM) (atau gunakan yang sudah ada) di AWS akun Anda, lalu akses Device Farm dengan pengguna IAM tersebut.

Untuk informasi selengkapnya, lihat Membuat Pengguna IAM (Konsol Manajemen AWS).

Langkah 3: Berikan izin kepada pengguna IAM untuk mengakses Device Farm

Berikan izin kepada pengguna IAM untuk mengakses Device Farm. Untuk melakukannya, buat kebijakan akses di IAM, lalu tetapkan kebijakan akses ke pengguna IAM, sebagai berikut.

> **Note**
>
> Akun AWS root atau pengguna IAM yang Anda gunakan untuk menyelesaikan langkah-langkah berikut harus memiliki izin untuk membuat kebijakan IAM berikut dan melampirkannya ke pengguna IAM. Untuk informasi selengkapnya, lihat Bekerja dengan Kebijakan.

1. Buat kebijakan dengan badan JSON berikut. Berikan judul deskriptif, seperti*`DeviceFarmAdmin`*.

   Untuk informasi selengkapnya tentang membuat kebijakan IAM, lihat Membuat Kebijakan IAM di Panduan Pengguna IAM.
2. Lampirkan kebijakan IAM yang Anda buat ke pengguna baru Anda. Untuk informasi selengkapnya tentang melampirkan kebijakan IAM ke pengguna, lihat Menambahkan dan Menghapus Kebijakan IAM di Panduan Pengguna IAM.

Melampirkan kebijakan memberi pengguna IAM akses ke semua tindakan dan sumber daya Device Farm yang terkait dengan pengguna IAM tersebut. Untuk informasi tentang cara membatasi pengguna IAM pada serangkaian tindakan dan sumber daya Device Farm terbatas, lihat. Manajemen identitas dan akses di AWS Device Farm

Langkah berikutnya

Anda sekarang siap untuk mulai menggunakan Device Farm. Lihat Memulai Device Farm.

Memulai Device Farm

Panduan ini menunjukkan cara menggunakan Device Farm untuk menguji aplikasi Android atau iOS asli. Anda menggunakan konsol Device Farm untuk membuat project, mengunggah file.apk atau .ipa, menjalankan rangkaian pengujian standar, dan kemudian melihat hasilnya.

> **Note**
>
> Device Farm hanya tersedia di AWS Wilayah `us-west-2` (Oregon).

Topik

Prasyarat

Sebelum Anda mulai, pastikan Anda telah menyelesaikan persyaratan berikut:

- Selesaikan langkah-langkah dalam Menyiapkan. Anda memerlukan AWS akun dan pengguna AWS Identity and Access Management (IAM) dengan izin untuk mengakses Device Farm.

- Untuk Android, Anda dapat membawa file.apk (paket aplikasi Android), atau menggunakan contoh aplikasi yang kami sediakan. Untuk iOS, Anda memerlukan file.ipa (arsip aplikasi iOS). Anda mengunggah file ke Device Farm nanti dalam panduan ini.

  > **Note**
  >
  > Pastikan file.ipa Anda dibuat untuk perangkat iOS dan bukan untuk simulator.

- (Opsional) Anda memerlukan pengujian dari salah satu kerangka pengujian yang didukung Device Farm. Anda mengunggah paket pengujian ini ke Device Farm, lalu jalankan pengujian nanti dalam panduan ini. Jika Anda tidak memiliki paket pengujian yang tersedia, Anda dapat menentukan dan menjalankan rangkaian pengujian bawaan standar. Untuk informasi selengkapnya, lihat Uji kerangka kerja dan pengujian bawaan di AWS Device Farm.

Langkah 1: Masuk ke konsol

Anda dapat menggunakan konsol Device Farm untuk membuat dan mengelola proyek dan menjalankan pengujian. Anda belajar tentang proyek dan berjalan nanti dalam panduan ini.

- Masuk ke konsol Device Farm di https://console.aws.amazon.com/devicefarm.

Langkah 2: Buat proyek

Untuk menguji aplikasi di Device Farm, Anda harus terlebih dahulu membuat proyek.

1. Di panel navigasi, pilih Pengujian Perangkat Seluler, lalu pilih Proyek.
2. Di bawah Proyek Pengujian Perangkat Seluler, pilih Buat proyek.
3. Di bawah Buat proyek, masukkan Nama Proyek (misalnya,**`MyDemoProject`**).
4. Pilih Buat.

   Konsol membuka halaman pengujian otomatis dari proyek yang baru Anda buat.

Langkah 3: Buat dan mulai lari

Sekarang setelah Anda memiliki proyek, Anda dapat membuat dan kemudian mulai menjalankan. Untuk informasi selengkapnya, lihat Berjalan.

1. Pada tab Tes otomatis, pilih Buat jalankan. Atau, Anda dapat mengikuti tutorial di konsol dengan memilih Buat run dengan tutorial.
2. (Opsional) Di bawah pengaturan Jalankan, di bagian Jalankan nama, masukkan nama untuk menjalankan Anda. Jika tidak ada nama yang diberikan, konsol Device Farm akan menamai run Anda 'My Device Farm run' secara default.

3. Di bawah Run settings, di bagian Run type, pilih tipe run Anda. Pilih aplikasi Android jika Anda tidak memiliki aplikasi yang siap untuk diuji, atau jika Anda sedang menguji aplikasi android (.apk). Pilih aplikasi iOS jika Anda menguji aplikasi iOS (.ipa).
4. Di bagian Pilih aplikasi, di bagian Opsi pemilihan aplikasi, pilih Pilih contoh aplikasi yang disediakan oleh Device Farm jika Anda tidak memiliki aplikasi yang tersedia untuk pengujian. Jika Anda membawa aplikasi sendiri, pilih Unggah aplikasi sendiri, dan pilih file aplikasi Anda. Jika Anda mengunggah aplikasi iOS, pastikan untuk memilih perangkat iOS, bukan simulator.
5. Di bawah Configure test, di bagian Select test framework, pilih salah satu framework pengujian atau suite pengujian bawaan. Lihat informasi tentang setiap opsi di Kerangka kerja uji dan pengujian bawaan.
   - Jika Anda belum mengemas pengujian untuk Device Farm, pilih Built-in: Fuzz untuk menjalankan rangkaian pengujian bawaan standar. Anda dapat menyimpan nilai default untuk jumlah Event, throttle Event, dan Randomizer seed. Untuk informasi selengkapnya, lihat the section called “Bawaan: fuzz (Android dan iOS)”.
   - Jika Anda memiliki paket pengujian dari salah satu kerangka pengujian yang didukung, pilih kerangka pengujian yang sesuai, lalu unggah file yang berisi pengujian Anda.
6. Di bawah Pilih perangkat, pilih Gunakan Pool Perangkat dan Perangkat Teratas.
7. (Opsional) Untuk menambahkan konfigurasi tambahan, buka tarik-turun konfigurasi Tambahan. Di bagian ini, Anda dapat melakukan salah satu hal berikut:
   - Untuk menyediakan data lain bagi Device Farm untuk digunakan selama proses, di samping Tambahkan data tambahan, pilih Pilih File, lalu telusuri ke dan pilih file.zip yang berisi data.
   - Untuk menginstal aplikasi tambahan untuk Device Farm untuk digunakan selama menjalankan, di samping Instal aplikasi lain, pilih Pilih File, lalu telusuri ke dan pilih file.apk atau.ipa yang berisi aplikasi. Ulangi ini untuk aplikasi lain yang ingin Anda instal. Anda dapat mengubah urutan instalasi dengan menyeret dan menjatuhkan aplikasi setelah Anda mengunggahnya.
   - Untuk menentukan apakah Wi-Fi, Bluetooth, GPS, atau NFC diaktifkan selama proses, di samping Setel status radio, pilih kotak yang sesuai.
   - Untuk mengatur lintang dan bujur perangkat untuk menjalankan, di samping Lokasi perangkat, masukkan koordinat.
   - Untuk mengatur lokal perangkat untuk dijalankan, di lokal Perangkat, pilih lokal.
   - Pilih Aktifkan perekaman video untuk merekam video selama pengujian.
   - Pilih Aktifkan pengambilan data kinerja aplikasi untuk menangkap data kinerja dari perangkat.

> **Note**
>
> Menyetel status radio perangkat dan lokal adalah opsi yang hanya tersedia untuk pengujian asli Android saat ini.

> **Note**
>
> Jika Anda memiliki perangkat pribadi, konfigurasi khusus untuk perangkat pribadi juga ditampilkan.

8. Di bagian bawah halaman, pilih Buat jalankan untuk menjadwalkan proses.

Device Farm mulai dijalankan segera setelah perangkat tersedia, biasanya dalam beberapa menit. Untuk melihat status run, pada halaman Pengujian otomatis proyek Anda, pilih nama run Anda. Satu halaman jalankan, di bawah Perangkat, setiap perangkat dimulai dengan ikon yang tertunda

di tabel perangkat, lalu beralih ke ikon yang sedang berjalan

saat pengujian dimulai. Saat setiap pengujian selesai, konsol menampilkan ikon hasil pengujian di sebelah nama perangkat. Ketika semua pengujian selesai, ikon tertunda di sebelah run berubah menjadi ikon hasil pengujian.

Langkah 4: Lihat hasil run

Untuk melihat hasil pengujian dari proses, pada halaman Pengujian otomatis proyek Anda, pilih nama proses Anda. Halaman ringkasan menampilkan:

- Jumlah total tes, berdasarkan hasil.
- Daftar tes dengan peringatan atau kegagalan unik.
- Daftar perangkat dengan hasil tes untuk masing-masing.
- Setiap tangkapan layar yang diambil selama proses, dikelompokkan berdasarkan perangkat.
- Bagian untuk mengunduh hasil parsing.

Untuk informasi selengkapnya, lihat Melihat laporan pengujian di Device Farm.

Langkah selanjutnya

Untuk informasi selengkapnya tentang Device Farm, lihatKonsep.

Membeli slot perangkat di Device Farm

Anda dapat menggunakan konsol Device Farm, AWS Command Line Interface (AWS CLI), atau Device Farm API untuk membeli slot perangkat.

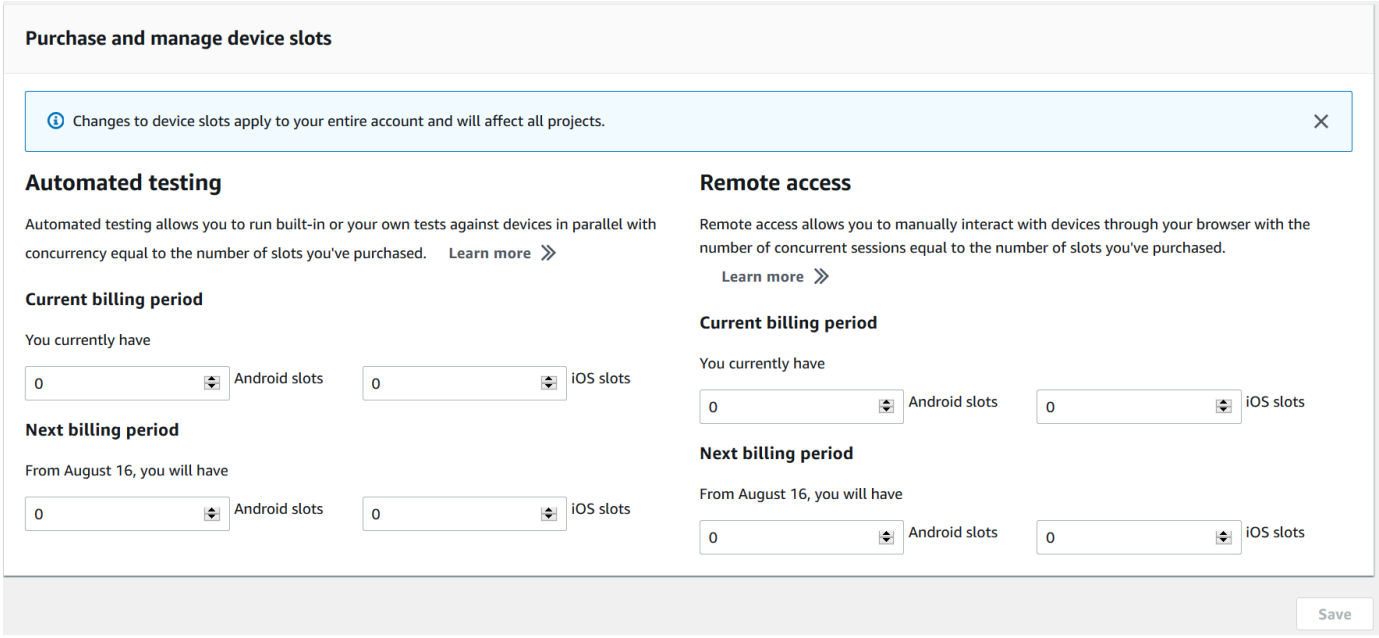
Beli slot perangkat (konsol)

1. Masuk ke konsol Device Farm di https://console.aws.amazon.com/devicefarm.
2. Di panel navigasi, pilih Pengujian Perangkat Seluler, lalu pilih Slot perangkat.
3. Pada halaman Pembelian dan kelola slot perangkat, Anda dapat membuat paket kustom Anda sendiri dengan memilih jumlah slot pengujian Otomatis dan perangkat akses jarak jauh yang ingin Anda beli. Tentukan jumlah slot untuk periode penagihan saat ini dan berikutnya.

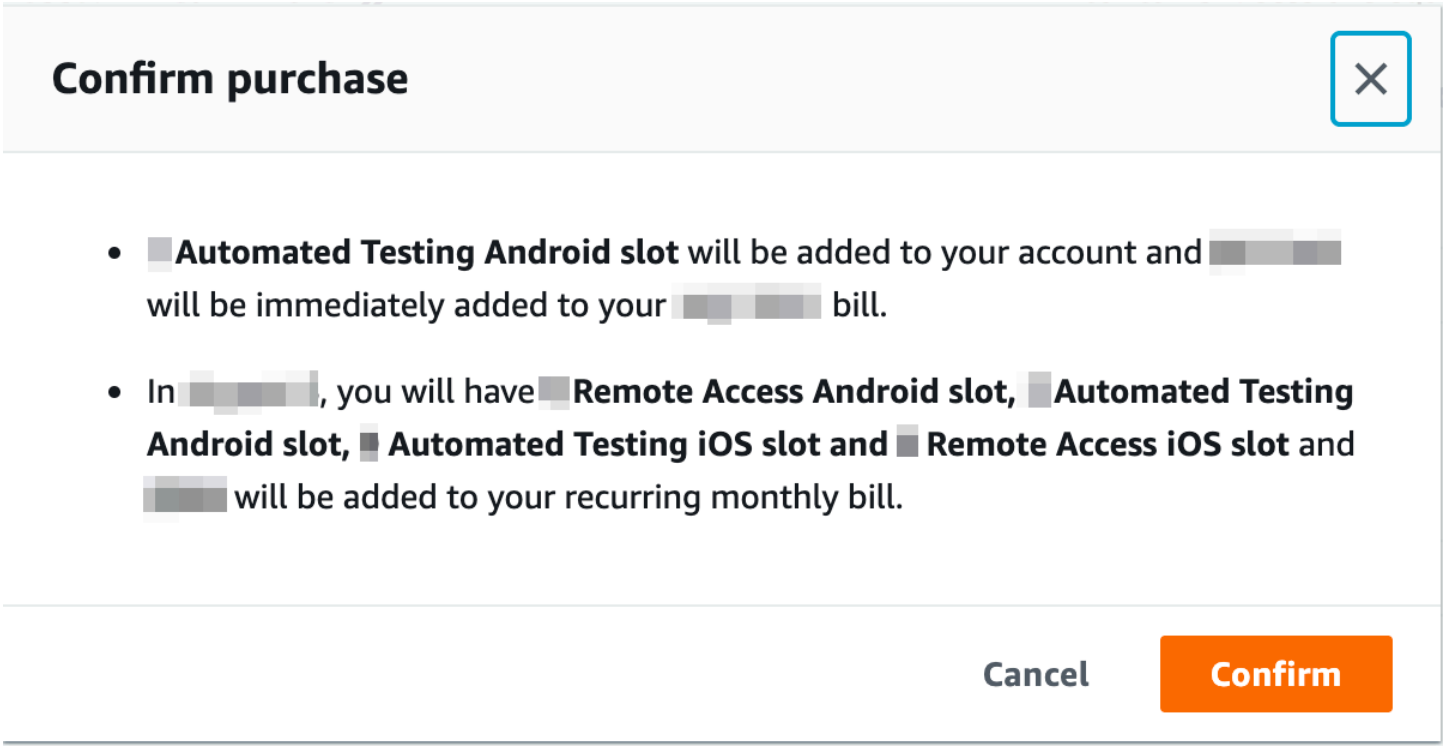
   Saat Anda mengubah jumlah slot, teks diperbarui secara dinamis dengan jumlah penagihan. Untuk informasi selengkapnya, lihat harga AWS Device Farm.

   > **Important**
   >
   > Jika Anda mengubah jumlah slot perangkat tetapi melihat kontak kami atau menghubungi kami untuk membeli pesan, AWS akun Anda belum disetujui untuk membeli jumlah slot perangkat yang Anda minta.
   > Opsi ini meminta Anda untuk mengirim email ke tim dukungan Device Farm. Di email, tentukan jumlah setiap jenis perangkat yang ingin Anda beli dan siklus penagihan mana.

   > **Note**
   >
   > Perubahan pada slot perangkat berlaku untuk seluruh akun Anda dan memengaruhi semua proyek.



4. Pilih Beli. Jendela Konfirmasi pembelian akan muncul. Tinjau informasi dan kemudian pilih Konfirmasi untuk menyelesaikan transaksi.



Pada halaman Pembelian dan kelola slot perangkat, Anda dapat melihat jumlah slot perangkat yang Anda miliki saat ini. Jika Anda menambah atau mengurangi jumlah slot, Anda akan melihat jumlah slot yang akan Anda miliki satu bulan setelah tanggal Anda melakukan perubahan.

Beli slot perangkat (AWS CLI)

Anda dapat menjalankan purchase-offering perintah untuk membeli penawaran.

Untuk mencantumkan pengaturan akun Device Farm Anda, termasuk jumlah maksimum slot perangkat yang dapat Anda beli dan jumlah menit uji coba gratis yang tersisa, jalankan get-account-settings perintah. Anda akan melihat output yang mirip dengan berikut ini:

```
{
    "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
        }
    }
}
```

Untuk membuat daftar penawaran slot perangkat yang tersedia untuk Anda, jalankan perintah. list-offerings Anda akan melihat output yang serupa dengan yang berikut:

```
{
```

```
    "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"
        }
    ]
}
```

Untuk membuat daftar promosi penawaran yang tersedia, jalankan list-offering-promotions perintah.

> **Note**
>
> Perintah ini hanya mengembalikan promosi yang belum Anda beli. Segera setelah Anda membeli satu atau lebih slot di penawaran apa pun menggunakan promosi, promosi itu tidak lagi muncul dalam hasil.

Anda akan melihat output yang serupa dengan yang berikut:

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

Untuk mendapatkan status penawaran, jalankan get-offering-status perintah. Anda akan melihat output yang serupa dengan yang berikut:

```
{
    "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
        }
    }
```

```
}
```

list-offering-transactionsPerintah renew-offering dan juga tersedia untuk fitur ini. Untuk informasi selengkapnya, lihat AWS CLI referensi.

Beli slot perangkat (API)

1. Panggil GetAccountSettingsoperasi untuk membuat daftar pengaturan akun Anda.
2. Panggil ListOfferingsoperasi untuk mencantumkan penawaran slot perangkat yang tersedia untuk Anda.
3. Hubungi ListOfferingPromotionsoperasi untuk membuat daftar promosi penawaran yang tersedia.

   > **Note**
   >
   > Perintah ini hanya mengembalikan promosi yang belum Anda beli. Segera setelah Anda membeli satu atau lebih slot menggunakan promosi penawaran, promosi itu tidak lagi muncul dalam hasil.

4. Hubungi PurchaseOfferingoperasi untuk membeli penawaran.
5. Hubungi GetOfferingStatusoperasi untuk mendapatkan status penawaran.

ListOfferingTransactionsPerintah RenewOfferingdan juga tersedia untuk fitur ini.

Untuk informasi tentang menggunakan Device Farm API, lihatMengotomatisasi Device Farm.

Membatalkan slot perangkat di Device Farm

Anda dapat membatalkan jumlah slot perangkat untuk pengujian otomatis dan akses jarak jauh. Untuk instruksi, lihat salah satu bagian berikut. Jumlah yang dibebankan ke akun Anda untuk siklus penagihan berikutnya akan dicantumkan di bawah bidang periode penagihan.

Untuk informasi selengkapnya tentang slot perangkat, lihatMembeli slot perangkat di Device Farm.

Batalkan slot perangkat (konsol)

1. Masuk ke konsol Device Farm di https://console.aws.amazon.com/devicefarm.

2. Di panel navigasi, pilih Pengujian Perangkat Seluler, lalu pilih Slot perangkat.
3. Pada halaman Pembelian dan kelola slot perangkat, Anda dapat mengurangi jumlah slot perangkat untuk pengujian otomatis dan akses jarak jauh dengan mengurangi nilai pada periode penagihan berikutnya. Jumlah yang dibebankan ke akun Anda untuk siklus penagihan berikutnya akan dicantumkan di bawah bidang periode penagihan.
4. Pilih Simpan. Jendela Konfirmasi Perubahan akan muncul. Tinjau informasi dan kemudian pilih Konfirmasi untuk menyelesaikan transaksi.

Batalkan slot perangkat (AWS CLI)

Anda dapat menjalankan renew-offering perintah untuk mengubah jumlah perangkat untuk siklus penagihan berikutnya.

Membatalkan slot perangkat (API)

Panggil RenewOfferingoperasi untuk mengubah jumlah perangkat di akun Anda.

Konsep AWS Device Farm

Device Farm adalah layanan pengujian aplikasi yang dapat Anda gunakan untuk menguji dan berinteraksi dengan aplikasi Android, iOS, dan web di ponsel dan tablet fisik nyata yang di-host oleh Amazon Web Services (AWS).

Bagian ini menjelaskan konsep-konsep penting Device Farm.

- Dukungan perangkat di AWS Device Farm
- Uji lingkungan di AWS Device Farm
- Berjalan
- Aplikasi
- Laporan di AWS Device Farm
- Sesi

Untuk informasi selengkapnya tentang jenis pengujian yang didukung di Device Farm, lihatUji kerangka kerja dan pengujian bawaan di AWS Device Farm.

Dukungan perangkat di AWS Device Farm

Bagian berikut memberikan informasi tentang dukungan perangkat di Device Farm.

Topik

- Perangkat yang didukung
- Kolam perangkat
- Perangkat pribadi
- Pencitraan merek perangkat
- Slot perangkat
- Aplikasi perangkat yang sudah diinstal sebelumnya
- Kemampuan perangkat

Perangkat yang didukung

Device Farm menyediakan dukungan untuk ratusan perangkat Android dan iOS yang unik dan populer serta kombinasi sistem operasi. Daftar perangkat yang tersedia bertambah saat perangkat baru memasuki pasar. Untuk daftar lengkap perangkat, lihat daftar perangkat interaktif di AWS konsol Anda.

Kolam perangkat

Device Farm mengatur perangkatnya ke dalam kumpulan perangkat yang dapat Anda gunakan untuk pengujian. Kumpulan perangkat ini berisi perangkat terkait, seperti perangkat yang hanya berjalan di Android atau hanya di iOS. Device Farm menyediakan kumpulan perangkat yang dikuratori, seperti untuk perangkat teratas. Anda juga dapat membuat kumpulan perangkat yang memadukan perangkat publik dan pribadi.

Perangkat pribadi

Perangkat pribadi memungkinkan Anda menentukan konfigurasi perangkat keras dan perangkat lunak yang tepat untuk kebutuhan pengujian Anda. Konfigurasi tertentu, seperti perangkat Android yang di-rooting, dapat didukung sebagai perangkat pribadi. Setiap perangkat pribadi adalah perangkat fisik yang digunakan Device Farm atas nama Anda di pusat data Amazon. Perangkat pribadi Anda tersedia secara eksklusif untuk Anda untuk pengujian otomatis dan manual. Setelah Anda memilih untuk mengakhiri langganan Anda, perangkat keras dihapus dari lingkungan kami. Untuk informasi selengkapnya, lihat Perangkat Pribadi danPerangkat pribadi di AWS Device Farm.

Pencitraan merek perangkat

Device Farm menjalankan pengujian pada perangkat seluler dan tablet fisik dari berbagai perangkat OEMs.

Slot perangkat

Slot perangkat sesuai dengan konkurensi di mana jumlah slot perangkat yang telah Anda beli menentukan berapa banyak perangkat yang dapat Anda jalankan dalam pengujian atau sesi akses jarak jauh.

Ada dua jenis slot perangkat:

- Slot perangkat akses jarak jauh adalah slot yang dapat Anda jalankan dalam sesi akses jarak jauh secara bersamaan.

Jika Anda memiliki satu slot perangkat akses jarak jauh, Anda hanya dapat menjalankan satu sesi akses jarak jauh pada satu waktu. Jika Anda membeli slot perangkat pengujian jarak jauh tambahan, Anda dapat menjalankan beberapa sesi secara bersamaan.

- Slot perangkat pengujian otomatis adalah slot di mana Anda dapat menjalankan pengujian secara bersamaan.

  Jika Anda memiliki satu slot perangkat pengujian otomatis, Anda hanya dapat menjalankan tes pada satu perangkat pada satu waktu. Jika Anda membeli slot perangkat pengujian otomatis tambahan, Anda dapat menjalankan beberapa pengujian secara bersamaan, di beberapa perangkat, untuk mendapatkan hasil pengujian lebih cepat.

Anda dapat membeli slot perangkat berdasarkan keluarga perangkat (perangkat Android atau iOS untuk pengujian otomatis dan perangkat Android atau iOS untuk akses jarak jauh). Untuk informasi selengkapnya, lihat Harga Device Farm.

Aplikasi perangkat yang sudah diinstal sebelumnya

Perangkat di Device Farm menyertakan sejumlah kecil aplikasi yang sudah diinstal oleh produsen dan operator.

Kemampuan perangkat

Semua perangkat memiliki konektivitas internet. Mereka tidak memiliki koneksi operator dan tidak dapat melakukan panggilan telepon atau mengirim pesan SMS.

Anda dapat mengambil foto dengan perangkat apa pun yang mendukung kamera depan atau belakang. Karena cara perangkat dipasang, foto mungkin terlihat gelap dan buram.

Layanan Google Play dan Google Chrome diinstal pada perangkat Android.

Uji lingkungan di AWS Device Farm

AWS Device Farm menyediakan lingkungan pengujian kustom dan standar untuk menjalankan pengujian otomatis Anda. Anda dapat memilih lingkungan pengujian khusus untuk kontrol penuh atas pengujian otomatis Anda. Atau, Anda dapat memilih lingkungan pengujian standar default Device Farm, yang menawarkan pelaporan terperinci dari setiap pengujian dalam rangkaian pengujian otomatis Anda.

Topik

Lingkungan uji standar

Saat Anda menjalankan pengujian di lingkungan standar, Device Farm menyediakan log dan pelaporan terperinci untuk setiap kasus dalam rangkaian pengujian Anda. Anda dapat melihat data performa, video, tangkapan layar, dan log untuk setiap pengujian untuk menentukan dan memperbaiki masalah di aplikasi Anda.

> **Note**
>
> Karena Device Farm menyediakan pelaporan terperinci di lingkungan standar, waktu eksekusi pengujian bisa lebih lama daripada saat Anda menjalankan pengujian secara lokal. Jika Anda ingin waktu eksekusi lebih cepat, jalankan pengujian Anda di lingkungan pengujian khusus.

Lingkungan uji kustom

Saat Anda menyesuaikan lingkungan pengujian, Anda dapat menentukan perintah yang harus dijalankan Device Farm untuk menjalankan pengujian. Ini memastikan bahwa pengujian di Device Farm berjalan dengan cara yang mirip dengan pengujian yang dijalankan di mesin lokal Anda. Menjalankan pengujian Anda dalam mode ini juga memungkinkan live log dan streaming video pengujian Anda. Saat Anda menjalankan pengujian di lingkungan pengujian yang disesuaikan, Anda tidak mendapatkan laporan terperinci untuk setiap kasus uji. Untuk informasi selengkapnya, lihat Lingkungan pengujian khusus di AWS Device Farm.

Anda memiliki opsi untuk menggunakan lingkungan pengujian khusus saat menggunakan konsol Device Farm AWS CLI, atau Device Farm API untuk membuat uji coba.

Untuk informasi selengkapnya, lihat Mengunggah Spesifikasi Uji Kustom Menggunakan dan. AWS CLI Membuat uji coba di Device Farm

Berjalan di AWS Device Farm

Bagian berikut berisi informasi tentang berjalan di Device Farm.

Jalankan di Device Farm mewakili build spesifik aplikasi Anda, dengan serangkaian pengujian tertentu, untuk dijalankan pada set perangkat tertentu. Run menghasilkan laporan yang berisi informasi tentang hasil run. Lari berisi satu atau lebih pekerjaan.

Topik

Jalankan konfigurasi

Sebagai bagian dari proses, Anda dapat menyediakan pengaturan yang dapat digunakan Device Farm untuk mengganti pengaturan perangkat saat ini. Ini termasuk koordinat lintang dan bujur, data tambahan (terkandung dalam file.zip), dan aplikasi tambahan (aplikasi yang harus diinstal sebelum aplikasi diuji). Di Android, beberapa pengaturan tambahan dapat diubah, seperti status lokal dan radio (Bluetooth, GPS, NFC, dan Wi-Fi).

Jalankan retensi file

Device Farm menyimpan aplikasi dan file Anda selama 30 hari dan kemudian menghapusnya dari sistemnya. Namun, Anda dapat menghapus file Anda kapan saja.

Device Farm menyimpan hasil run, log, dan tangkapan layar Anda selama 400 hari dan kemudian menghapusnya dari sistemnya.

Jalankan status perangkat

Device Farm selalu me-reboot perangkat sebelum membuatnya tersedia untuk pekerjaan berikutnya.

Berjalan paralel

Device Farm menjalankan pengujian secara paralel saat perangkat tersedia.

Mengatur batas waktu eksekusi

Anda dapat menetapkan nilai untuk berapa lama uji coba harus dijalankan sebelum Anda menghentikan setiap perangkat menjalankan pengujian. Misalnya, jika pengujian Anda membutuhkan waktu 20 menit per perangkat untuk diselesaikan, Anda harus memilih batas waktu 30 menit per perangkat.

Untuk informasi selengkapnya, lihat Menyetel batas waktu eksekusi untuk pengujian berjalan di AWS Device Farm.

Iklan berjalan

Sebaiknya hapus iklan dari aplikasi sebelum mengunggahnya ke Device Farm. Kami tidak dapat menjamin bahwa iklan ditampilkan selama berjalan.

Media sedang berjalan

Anda dapat menyediakan media atau data lain untuk menemani aplikasi Anda. Data tambahan harus disediakan dalam file.zip berukuran tidak lebih dari 4 GB.

Tugas umum untuk berjalan

Untuk informasi selengkapnya, lihat Membuat uji coba di Device Farm dan Pengujian berjalan di AWS Device Farm.

Aplikasi di AWS Device Farm

Bagian berikut berisi informasi tentang perilaku aplikasi di Device Farm.

Topik

- Aplikasi instrumentasi
- Menandatangani ulang aplikasi dalam proses
- Aplikasi yang dikaburkan sedang berjalan

Aplikasi instrumentasi

Anda tidak perlu menginstruksikan aplikasi Anda atau menyediakan Device Farm dengan kode sumber untuk aplikasi Anda. Aplikasi Android dapat dikirimkan tanpa dimodifikasi. Aplikasi iOS harus dibangun dengan target Perangkat iOS, bukan dengan simulator.

Menandatangani ulang aplikasi dalam proses

Untuk aplikasi iOS, Anda tidak perlu menambahkan Device Farm apa pun UUIDs ke profil penyediaan Anda. Device Farm mengganti profil penyediaan yang disematkan dengan profil wildcard dan kemudian menandatangani ulang aplikasi. Jika Anda memberikan data tambahan, Device Farm menambahkannya ke paket aplikasi sebelum Device Farm menginstalnya, sehingga tambahan tersebut ada di kotak pasir aplikasi Anda. Penandatanganan ulang aplikasi menghapus hak seperti Grup Aplikasi, Domain Terkait, Game Center,,, Konfigurasi Aksesori Nirkabel HealthKit HomeKit, Pembelian Dalam Aplikasi, Audio Antar-Aplikasi, Apple Pay, Pemberitahuan Push, dan Konfigurasi & Kontrol VPN.

Untuk aplikasi Android, Device Farm menandatangani ulang aplikasi. Ini dapat merusak fungsionalitas apa pun yang bergantung pada tanda tangan aplikasi, seperti Google Maps Android API, atau mungkin memicu antipiracy atau deteksi antitamper dari produk seperti. DexGuard

Aplikasi yang dikaburkan sedang berjalan

Untuk aplikasi Android, jika aplikasi dikaburkan, Anda masih dapat mengujinya dengan Device Farm jika Anda menggunakannya. ProGuard Namun, jika Anda menggunakan DexGuard dengan tindakan antipembajakan, Device Farm tidak dapat menandatangani ulang dan menjalankan pengujian terhadap aplikasi.

Laporan di AWS Device Farm

Bagian berikut memberikan informasi tentang laporan pengujian Device Farm.

Topik

Laporkan retensi

Device Farm menyimpan laporan Anda selama 400 hari. Laporan ini mencakup metadata, log, tangkapan layar, dan data kinerja.

Laporkan komponen

Laporan di Device Farm berisi informasi lulus dan gagal, laporan kerusakan, log pengujian dan perangkat, tangkapan layar, dan data kinerja.

Laporan mencakup data per-perangkat rinci dan hasil tingkat tinggi, seperti jumlah kemunculan masalah yang diberikan.

Log dalam laporan

Laporan mencakup tangkapan logcat lengkap untuk pengujian Android dan log Konsol Perangkat lengkap untuk pengujian iOS.

Tugas umum untuk laporan

Lihat informasi yang lebih lengkap di Melihat laporan pengujian di Device Farm.

Sesi di AWS Device Farm

Anda dapat menggunakan Device Farm untuk melakukan pengujian interaktif aplikasi Android dan iOS melalui sesi akses jarak jauh. Ini termasuk interaksi manual di browser web dan menjalankan tes Appium dari klien lokal terhadap perangkat jarak jauh. Pengembang dapat mereproduksi masalah dengan aplikasi mereka atau dengan pengujian Appium mereka pada perangkat tertentu untuk mengisolasi dan menyelesaikan masalah.

Topik

- Perangkat yang didukung untuk akses jarak jauh
- Retensi file sesi
- Aplikasi instrumentasi
- Menandatangani ulang aplikasi dalam sesi
- Aplikasi yang dikaburkan dalam sesi

Perangkat yang didukung untuk akses jarak jauh

Device Farm menyediakan dukungan untuk sejumlah perangkat Android dan iOS yang unik dan populer. Daftar perangkat yang tersedia bertambah saat perangkat baru memasuki pasar. Konsol

Device Farm menampilkan daftar perangkat Android dan iOS saat ini yang tersedia untuk akses jarak jauh. Untuk informasi selengkapnya, lihat Dukungan perangkat di AWS Device Farm.

Retensi file sesi

Device Farm menyimpan aplikasi dan file Anda selama 30 hari dan kemudian menghapusnya dari sistemnya. Namun, Anda dapat menghapus file Anda kapan saja.

Device Farm menyimpan log sesi Anda dan merekam video selama 400 hari dan kemudian menghapusnya dari sistemnya.

Aplikasi instrumentasi

Anda tidak perlu menginstruksikan aplikasi Anda atau menyediakan Device Farm dengan kode sumber untuk aplikasi Anda. Aplikasi Android dan iOS dapat dikirimkan tanpa dimodifikasi.

Menandatangani ulang aplikasi dalam sesi

Device Farm menandatangani ulang aplikasi Android dan iOS. Ini dapat merusak fungsionalitas yang bergantung pada tanda tangan aplikasi. Misalnya, Google Maps API untuk Android bergantung pada tanda tangan aplikasi Anda. Penandatanganan ulang aplikasi juga dapat memicu deteksi antipembajakan atau antitamper dari produk seperti untuk perangkat Android. DexGuard

Aplikasi yang dikaburkan dalam sesi

Untuk aplikasi Android, jika aplikasi dikaburkan, Anda masih dapat mengujinya dengan Device Farm jika Anda menggunakannya. ProGuard Namun, jika Anda menggunakan DexGuard dengan tindakan antipembajakan, Device Farm tidak dapat menandatangani ulang aplikasi.

Proyek di AWS Device Farm

Project di Device Farm mewakili ruang kerja logis di Device Farm yang berisi run, satu run untuk setiap pengujian satu aplikasi terhadap satu perangkat atau beberapa. Proyek memungkinkan Anda untuk mengatur ruang kerja dengan cara apa pun yang Anda pilih. Misalnya, mungkin ada satu proyek per judul aplikasi, atau mungkin ada satu proyek per platform. Anda dapat membuat proyek sebanyak yang Anda butuhkan.

Anda dapat menggunakan konsol AWS Device Farm, AWS Command Line Interface (AWS CLI), atau AWS Device Farm API untuk bekerja dengan proyek.

Topik

Membuat proyek di AWS Device Farm

Anda dapat membuat project dengan menggunakan konsol AWS Device Farm AWS CLI, atau AWS Device Farm API.

Prasyarat

- Selesaikan langkah-langkah dalam Menyiapkan.

Buat proyek (konsol)

1. Masuk ke konsol Device Farm di https://console.aws.amazon.com/devicefarm.
2. Pada panel navigasi Device Farm, pilih Pengujian Perangkat Seluler, lalu pilih Proyek.
3. Pilih Proyek baru.
4. Masukkan nama untuk proyek Anda. Secara opsional, Anda dapat memberikan satu atau beberapa parameter di bawah ini, lalu pilih Kirim.

Pengaturan Virtual Private Cloud (VPC)

Pilih VPC, subnet, dan grup keamanan yang akan diterapkan ke perangkat yang sedang diuji dan host uji pasangannya. Fitur ini hanya didukung dengan perangkat pribadi. Untuk informasi selengkapnya, lihat VPC-ENI di AWS Device Farm.

Peran Eksekusi ARN

Peran IAM yang akan diasumsikan oleh pelari uji di lingkungan pengujian khusus. Untuk informasi selengkapnya, lihat Akses sumber daya AWS menggunakan Peran Eksekusi IAM.

Variabel Lingkungan

Satu atau lebih variabel yang akan dimasukkan ke dalam lingkungan proses runner eksekusi uji. Nama variabel yang dimulai dengan “DEVICEFARM_” dicadangkan untuk penggunaan layanan. Sebaiknya jangan menyimpan nilai sensitif dalam variabel lingkungan ini, dan sebagai gantinya menyarankan untuk menggunakan peran eksekusi IAM untuk mengambil nilai tersebut dari AWS Secrets Manager selama pengujian Anda.

Buat proyek (AWS CLI)

- Jalankancreate-project, tentukan nama proyek.

  Contoh:

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

  AWS CLI Tanggapan tersebut mencakup Nama Sumber Daya Amazon (ARN) dari proyek tersebut.

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

  Untuk informasi selengkapnya, lihat create-project dan AWS CLI referensi.

Buat proyek (API)

- Panggil `CreateProject`API.

Untuk informasi tentang menggunakan Device Farm API, lihatMengotomatisasi Device Farm.

Melihat daftar proyek di AWS Device Farm

Anda dapat menggunakan konsol AWS Device Farm AWS CLI,, atau AWS Device Farm API untuk melihat daftar proyek.

Topik

Prasyarat

- Buat setidaknya satu proyek di Device Farm. Ikuti instruksi diMembuat proyek di AWS Device Farm, dan kemudian kembali ke halaman ini.

Lihat daftar proyek (konsol)

1. Masuk ke konsol Device Farm di https://console.aws.amazon.com/devicefarm.
2. Untuk menemukan daftar proyek yang tersedia, lakukan hal berikut:
   - Untuk proyek pengujian perangkat seluler, pada menu navigasi Device Farm, pilih Pengujian Perangkat Seluler, lalu pilih Proyek.
   - Untuk proyek pengujian browser desktop, pada menu navigasi Device Farm, pilih Pengujian Browser Desktop, lalu pilih Proyek.

Lihat daftar proyek (AWS CLI)

- Untuk melihat daftar proyek, jalankan list-projectsperintah.

Untuk melihat informasi tentang satu proyek, jalankan get-projectperintah.

Untuk informasi tentang menggunakan Device Farm dengan AWS CLI, lihatAWS CLI referensi.

Lihat daftar proyek (API)

- Untuk melihat daftar proyek, panggil `ListProjects`API.

  Untuk melihat informasi tentang satu proyek, hubungi `GetProject`API.

Untuk informasi tentang AWS Device Farm API, lihatMengotomatisasi Device Farm.

Pengujian berjalan di AWS Device Farm

Jalankan di Device Farm mewakili build spesifik aplikasi Anda, dengan serangkaian pengujian tertentu, untuk dijalankan pada set perangkat tertentu. Run menghasilkan laporan yang berisi informasi tentang hasil run. Lari berisi satu atau lebih pekerjaan. Untuk informasi selengkapnya, lihat Berjalan.

Anda dapat menggunakan konsol AWS Device Farm, AWS Command Line Interface (AWS CLI), atau AWS Device Farm API untuk bekerja dengan pengujian.

Topik

Membuat uji coba di Device Farm

Anda dapat menggunakan konsol Device Farm AWS CLI, atau Device Farm API untuk membuat uji coba. Anda juga dapat menggunakan plugin yang didukung, seperti plugin Jenkins atau Gradle untuk Device Farm. Untuk informasi selengkapnya tentang plugin, lihat Alat dan plugin. Untuk informasi tentang lari, lihatBerjalan.

Topik

Prasyarat

Anda harus memiliki proyek di Device Farm. Ikuti instruksi diMembuat proyek di AWS Device Farm, dan kemudian kembali ke halaman ini.

Buat uji coba (konsol)

1. Masuk ke konsol Device Farm di https://console.aws.amazon.com/devicefarm.
2. Di panel navigasi, pilih Pengujian Perangkat Seluler, lalu pilih Proyek.
3. Jika Anda sudah memiliki proyek, Anda dapat mengunggah tes Anda ke sana. Jika tidak, pilih Proyek baru, masukkan Nama Proyek, lalu pilih Buat.
4. Buka proyek Anda, lalu pilih Create run.
5. (Opsional) Di bawah pengaturan Jalankan, di bagian Jalankan nama, masukkan nama untuk menjalankan Anda. Jika tidak ada nama yang diberikan, konsol Device Farm akan menamai run Anda 'My Device Farm run' secara default.
6. (Opsional) Di bawah pengaturan Jalankan, di bagian Batas waktu Job, Anda dapat menentukan batas waktu eksekusi untuk uji coba Anda. Jika Anda menggunakan slot pengujian tak terbatas, konfirmasikan bahwa Unmetered dipilih di bawah Metode penagihan.
7. Di bawah Run settings, di bagian Run type, pilih tipe run Anda. Pilih aplikasi Android jika Anda tidak memiliki aplikasi yang siap untuk diuji, atau jika Anda sedang menguji aplikasi android (.apk). Pilih aplikasi iOS jika Anda menguji aplikasi iOS (.ipa). Pilih aplikasi Web jika Anda ingin menguji aplikasi web.
8. Di bagian Pilih aplikasi, di bagian Opsi pemilihan aplikasi, pilih Pilih contoh aplikasi yang disediakan oleh Device Farm jika Anda tidak memiliki aplikasi yang tersedia untuk pengujian. Jika Anda membawa aplikasi sendiri, pilih Unggah aplikasi sendiri, dan pilih file aplikasi Anda. Jika Anda mengunggah aplikasi iOS, pastikan untuk memilih perangkat iOS, bukan simulator.
9. Di bawah Konfigurasi pengujian, pilih salah satu kerangka kerja pengujian yang tersedia.

> **Note**
>
> Jika Anda tidak memiliki pengujian yang tersedia, pilih Built-in: Fuzz untuk menjalankan rangkaian pengujian bawaan standar. Jika Anda memilih Built-in: Fuzz, dan kotak benih Event count, Event throttle, dan Randomizer muncul, Anda dapat mengubah atau menyimpan nilainya.

Untuk informasi tentang rangkaian pengujian yang tersedia, lihatUji kerangka kerja dan pengujian bawaan di AWS Device Farm.

10. Jika Anda tidak memilih Built-in: Fuzz, pilih Pilih File di bawah Pilih paket uji. Jelajahi dan pilih file yang berisi pengujian Anda.
11. Untuk lingkungan pengujian Anda, pilih Jalankan pengujian Anda di lingkungan standar kami atau Jalankan pengujian Anda di lingkungan khusus. Untuk informasi selengkapnya, lihat Uji lingkungan di AWS Device Farm.
12. Jika Anda menggunakan lingkungan pengujian kustom, Anda dapat melakukan hal berikut secara opsional:

    - Jika Anda ingin mengedit spesifikasi pengujian default di lingkungan pengujian kustom, pilih Edit untuk memperbarui spesifikasi YAMAL default.
    - Jika Anda mengubah spesifikasi pengujian, pilih Simpan sebagai Baru untuk memperbaruinya.
    - Anda dapat mengonfigurasi variabel lingkungan. Variabel yang disediakan di sini akan diutamakan daripada variabel apa pun yang dapat dikonfigurasi pada proyek induk.
13. Di bawah Pilih perangkat, lakukan salah satu hal berikut:

    - Untuk memilih kumpulan perangkat bawaan untuk menjalankan pengujian, untuk kumpulan Perangkat, pilih Perangkat Teratas.
    - Untuk membuat kumpulan perangkat Anda sendiri untuk menjalankan pengujian, ikuti petunjuk diMembuat kumpulan perangkat, lalu kembali ke halaman ini.
    - Jika Anda membuat kumpulan perangkat sendiri sebelumnya, untuk kumpulan Perangkat, pilih kumpulan perangkat Anda.
    - Pilih Pilih perangkat secara manual dan pilih perangkat yang diinginkan yang ingin Anda jalankan. Konfigurasi ini tidak akan disimpan.

    Untuk informasi selengkapnya, lihat Dukungan perangkat di AWS Device Farm.
14. (Opsional) Untuk menambahkan konfigurasi tambahan, buka tarik-turun konfigurasi Tambahan. Di bagian ini, Anda dapat melakukan salah satu hal berikut:

    - Untuk memberikan peran eksekusi ARN, atau mengganti yang dikonfigurasi pada proyek induk, gunakan bidang ARN peran Exectuion.
    - Untuk menyediakan data lain bagi Device Farm untuk digunakan selama proses, di samping Tambahkan data tambahan, pilih Pilih File, lalu telusuri ke dan pilih file.zip yang berisi data.

- Untuk menginstal aplikasi tambahan untuk Device Farm untuk digunakan selama menjalankan, di samping Instal aplikasi lain, pilih Pilih File, lalu telusuri ke dan pilih file.apk atau.ipa yang berisi aplikasi. Ulangi ini untuk aplikasi lain yang ingin Anda instal. Anda dapat mengubah urutan instalasi dengan menyeret dan menjatuhkan aplikasi setelah Anda mengunggahnya.
- Untuk menentukan apakah Wi-Fi, Bluetooth, GPS, atau NFC diaktifkan selama proses, di samping Setel status radio, pilih kotak yang sesuai.
- Untuk mengatur lintang dan bujur perangkat untuk menjalankan, di samping Lokasi perangkat, masukkan koordinat.
- Untuk mengatur lokal perangkat untuk dijalankan, di lokal Perangkat, pilih lokal.
- Pilih Aktifkan perekaman video untuk merekam video selama pengujian.
- Pilih Aktifkan pengambilan data kinerja aplikasi untuk menangkap data kinerja dari perangkat.

> **Note**
>
> Menyetel status radio perangkat dan lokal adalah opsi yang hanya tersedia untuk pengujian asli Android saat ini.

> **Note**
>
> Jika Anda memiliki perangkat pribadi, konfigurasi khusus untuk perangkat pribadi juga ditampilkan.

15. Di bagian bawah halaman, pilih Buat jalankan untuk menjadwalkan proses.

Device Farm mulai dijalankan segera setelah perangkat tersedia, biasanya dalam beberapa menit. Selama uji coba, konsol Device Farm menampilkan ikon yang tertunda  di tabel run. Setiap perangkat yang sedang dijalankan juga akan dimulai dengan ikon yang tertunda, lalu beralih ke ikon yang sedang berjalan  saat pengujian dimulai. Saat setiap pengujian selesai, ikon hasil pengujian ditampilkan di sebelah

nama perangkat. Ketika semua pengujian telah selesai, ikon yang tertunda di sebelah run berubah menjadi ikon hasil pengujian.

Jika Anda ingin menghentikan uji coba, lihatMenghentikan proses di AWS Device Farm.

Buat test run (AWS CLI)

Anda dapat menggunakan AWS CLI untuk membuat uji coba.

Topik

- Langkah 1: Pilih proyek
- Langkah 2: Pilih kumpulan perangkat
- Langkah 3: Unggah file aplikasi Anda
- Langkah 4: Unggah paket skrip pengujian Anda
- Langkah 5: (Opsional) Unggah spesifikasi pengujian khusus Anda
- Langkah 6: Jadwalkan uji coba

Langkah 1: Pilih proyek

Anda harus mengaitkan uji coba Anda dengan proyek Device Farm.

1. Untuk membuat daftar proyek Device Farm, jalankanlist-projects. Jika Anda tidak memiliki proyek, lihatMembuat proyek di AWS Device Farm.

   Contoh:

   ```
   aws devicefarm list-projects
   ```

   Responsnya mencakup daftar proyek Device Farm Anda.

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

```
    ]
}
```

2. Pilih proyek yang akan dikaitkan dengan uji coba Anda, dan catat Nama Sumber Daya Amazon (ARN).

Langkah 2: Pilih kumpulan perangkat

Anda harus memilih kumpulan perangkat untuk dikaitkan dengan uji coba Anda.

1. Untuk melihat kumpulan perangkat Anda, jalankanlist-device-pools, tentukan ARN proyek Anda.

   Contoh:

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

   Responsnya mencakup kumpulan perangkat Device Farm bawaan, sepertiTop Devices, dan kumpulan perangkat apa pun yang sebelumnya dibuat untuk proyek ini:

   ```
   {
       "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. Pilih kumpulan perangkat, dan catat ARN-nya.

   Anda juga dapat membuat kumpulan perangkat, dan kemudian kembali ke langkah ini. Untuk informasi selengkapnya, lihat Buat kumpulan perangkat (AWS CLI).

Langkah 3: Unggah file aplikasi Anda

Untuk membuat permintaan unggahan dan mendapatkan URL unggahan yang telah ditetapkan sebelumnya dari Amazon Simple Storage Service (Amazon S3), Anda memerlukan:

- Proyek Anda ARN.
- Nama file aplikasi Anda.
- Jenis unggahan.

Untuk informasi selengkapnya, lihat create-upload.

1. Untuk mengunggah file, jalankan create-upload dengan`--project-arn`,`--name`, dan `--type` parameter.

   Contoh ini membuat unggahan untuk aplikasi Android:

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

   Responsnya mencakup ARN unggahan aplikasi Anda dan URL yang telah ditetapkan sebelumnya.

   ```
   {
       "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. Catat ARN unggahan aplikasi dan URL yang telah ditentukan sebelumnya.
3. Unggah file aplikasi Anda menggunakan URL presigned Amazon S3. Contoh ini digunakan curl untuk mengunggah file Android .apk:

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

Untuk informasi selengkapnya, lihat Mengunggah objek menggunakan presigned URLs di Panduan Pengguna Layanan Penyimpanan Sederhana Amazon.

4. Untuk memeriksa status upload aplikasi Anda, jalankan get-upload dan tentukan ARN upload aplikasi.

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

Tunggu hingga status dalam respons SUCCEEDED sebelum Anda mengunggah paket skrip pengujian Anda.

```
{
    "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}"
    }
}
```

Langkah 4: Unggah paket skrip pengujian Anda

Selanjutnya, Anda mengunggah paket skrip pengujian Anda.

1. Untuk membuat permintaan unggahan dan mendapatkan URL unggahan Amazon S3 yang telah ditetapkan sebelumnya, jalankan create-upload dengan parameter `--project-arn--name`, dan parameter. `--type`

   Contoh ini membuat upload paket uji Appium Java TestNG:

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

   Respons termasuk paket pengujian Anda mengunggah ARN dan URL yang telah ditetapkan sebelumnya.

   ```
   {
       "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. Catat ARN dari unggahan paket uji dan URL yang telah ditentukan sebelumnya.
3. Unggah file paket skrip pengujian Anda menggunakan URL presigned Amazon S3. Contoh ini digunakan curl untuk mengunggah file skrip Appium TestNG yang di-zip:

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

4. Untuk memeriksa status unggahan paket skrip pengujian Anda, jalankan get-upload dan tentukan ARN dari unggahan paket pengujian dari langkah 1.

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

Tunggu hingga status dalam respons SUCCEEDED sebelum Anda melanjutkan ke langkah opsional berikutnya.

```
{
    "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}"
    }
}
```

Langkah 5: (Opsional) Unggah spesifikasi pengujian khusus Anda

Jika Anda menjalankan pengujian di lingkungan pengujian standar, lewati langkah ini.

Device Farm mempertahankan file spesifikasi pengujian default untuk setiap jenis pengujian yang didukung. Selanjutnya, Anda mengunduh spesifikasi pengujian default dan menggunakannya untuk membuat unggahan spesifikasi pengujian khusus untuk menjalankan pengujian Anda di lingkungan pengujian khusus. Untuk informasi selengkapnya, lihat Uji lingkungan di AWS Device Farm.

1. Untuk menemukan ARN unggahan untuk spesifikasi pengujian default Anda, jalankan list-uploads dan tentukan ARN proyek Anda.

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

Respons berisi entri untuk setiap spesifikasi pengujian default:

```
{
    "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. Pilih spesifikasi pengujian default Anda dari daftar. Catat unggahannya ARN.
3. Untuk mengunduh spesifikasi pengujian default Anda, jalankan get-upload dan tentukan ARN unggahan.

   Contoh:

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

   Respons berisi URL yang telah ditetapkan sebelumnya tempat Anda dapat mengunduh spesifikasi pengujian default.
4. Contoh ini digunakan curl untuk mengunduh spesifikasi pengujian default dan menyimpannya sebagai`MyTestSpec.yml`:

   ```
   curl "https://prod-us-west-2-uploads.s3-us-west-2.amazonaws.com/ExampleURL" >
    MyTestSpec.yml
   ```
5. Anda dapat mengedit spesifikasi pengujian default untuk memenuhi persyaratan pengujian Anda, dan kemudian menggunakan spesifikasi pengujian yang dimodifikasi dalam uji coba di masa mendatang. Lewati langkah ini untuk menggunakan spesifikasi pengujian default apa adanya di lingkungan pengujian khusus.
6. Untuk membuat unggahan spesifikasi pengujian kustom Anda, jalankancreate-upload, tentukan nama spesifikasi pengujian Anda, jenis spesifikasi pengujian, dan ARN proyek.

   Contoh ini membuat unggahan untuk spesifikasi pengujian kustom Appium Java TestNG:

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

Tanggapan tersebut mencakup ARN unggahan spesifikasi pengujian dan URL yang telah ditentukan sebelumnya:

```
{
    "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. Catat ARN untuk unggahan spesifikasi pengujian dan URL yang telah ditentukan sebelumnya.
8. Unggah file spesifikasi pengujian Anda menggunakan URL presigned Amazon S3. Contoh ini digunakan curl untuk mengunggah spesifikasi pengujian Appium JavaTest NG:

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

9. Untuk memeriksa status unggahan spesifikasi pengujian Anda, jalankan get-upload dan tentukan ARN unggahan.

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

Tunggu hingga status dalam respons SUCCEEDED sebelum Anda menjadwalkan uji coba Anda.

```
{
    "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}"
    }
}
```

Untuk memperbarui spesifikasi pengujian kustom Anda, jalankanupdate-upload, tentukan ARN unggahan untuk spesifikasi pengujian. Untuk informasi selengkapnya, lihat update-upload.

Langkah 6: Jadwalkan uji coba

Untuk menjadwalkan uji coba dengan AWS CLI, jalankanschedule-run, tentukan:

- Proyek ARN dari langkah 1.
- Perangkat mengumpulkan ARN dari langkah 2.
- Aplikasi mengunggah ARN dari langkah 3.
- Paket uji mengunggah ARN dari langkah 4.

Jika Anda menjalankan pengujian di lingkungan pengujian khusus, Anda juga memerlukan spesifikasi pengujian ARN dari langkah 5.

Untuk menjadwalkan lari di lingkungan pengujian standar

- Jalankanschedule-run, tentukan ARN proyek Anda, ARN kumpulan perangkat, ARN unggahan aplikasi, dan informasi paket uji.

  Contoh:

  ```
  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
  ```

  Respons berisi ARN run yang dapat Anda gunakan untuk memeriksa status uji coba Anda.

  ```
  {
      "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
        }
    }
}
```


Untuk informasi selengkapnya, lihat schedule-run.

Untuk menjadwalkan proses di lingkungan pengujian kustom

- Langkah-langkahnya hampir sama dengan langkah-langkahnya untuk lingkungan pengujian standar, dengan `testSpecArn` atribut tambahan dalam `--test` parameter.

  Contoh:

```
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
```

Untuk memeriksa status uji coba Anda

- Gunakan get-run perintah dan tentukan run ARN:

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

Untuk informasi selengkapnya, lihat get-run. Untuk informasi tentang menggunakan Device Farm dengan AWS CLI, lihatAWS CLI referensi.

Buat uji coba (API)

Langkah-langkahnya sama dengan yang dijelaskan di AWS CLI bagian ini. Lihat Buat test run (AWS CLI).

Anda memerlukan informasi ini untuk memanggil `ScheduleRun`API:

- Sebuah proyek ARN. Lihat Buat proyek (API) dan `CreateProject`.
- Aplikasi mengunggah ARN. Lihat `CreateUpload`.
- Paket uji mengunggah ARN. Lihat `CreateUpload`.
- Sebuah perangkat kolam ARN. Lihat Membuat kumpulan perangkat dan `CreateDevicePool`.

> **Note**
>
> Jika Anda menjalankan pengujian di lingkungan pengujian khusus, Anda juga memerlukan ARN unggahan spesifikasi pengujian. Untuk informasi selengkapnya, lihat Langkah 5: (Opsional) Unggah spesifikasi pengujian khusus Anda dan `CreateUpload`.

Untuk informasi tentang menggunakan Device Farm API, lihatMengotomatisasi Device Farm.

Langkah selanjutnya

Di konsol Device Farm, ikon jam

berubah menjadi ikon hasil seperti sukses

saat proses selesai. Laporan untuk proses muncul segera setelah pengujian selesai. Untuk informasi selengkapnya, lihat Laporan di AWS Device Farm.

Untuk menggunakan laporan, ikuti instruksi diMelihat laporan pengujian di Device Farm.

Menyetel batas waktu eksekusi untuk pengujian berjalan di AWS Device Farm

Anda dapat menetapkan nilai untuk berapa lama uji coba harus dijalankan sebelum Anda menghentikan setiap perangkat menjalankan pengujian. Batas waktu eksekusi default adalah 150 menit per perangkat, tetapi Anda dapat menetapkan nilai serendah 5 menit. Anda dapat menggunakan konsol AWS Device Farm, AWS CLI, atau AWS Device Farm API untuk mengatur batas waktu eksekusi.

> **Important**
>
> Opsi batas waktu eksekusi harus diatur ke durasi maksimum untuk uji coba, bersama dengan beberapa buffer. Misalnya, jika pengujian Anda memakan waktu 20 menit per perangkat, Anda harus memilih batas waktu 30 menit per perangkat.

Jika eksekusi melebihi batas waktu Anda, eksekusi pada perangkat tersebut dihentikan secara paksa. Hasil sebagian tersedia, jika memungkinkan. Anda ditagih untuk eksekusi hingga saat itu, jika Anda menggunakan opsi penagihan terukur. Untuk informasi selengkapnya tentang harga, lihat Harga Device Farm.

Anda mungkin ingin menggunakan fitur ini jika Anda tahu berapa lama waktu yang diperlukan untuk menjalankan uji coba di setiap perangkat. Saat menentukan batas waktu eksekusi untuk uji coba, Anda dapat menghindari situasi di mana uji coba macet karena alasan tertentu dan Anda ditagih untuk menit perangkat saat tidak ada pengujian yang dijalankan. Dengan kata lain, menggunakan

fitur batas waktu eksekusi memungkinkan Anda menghentikan proses itu jika memakan waktu lebih lama dari yang diharapkan.

Anda dapat mengatur batas waktu eksekusi di dua tempat, di tingkat proyek dan tingkat uji coba.

Prasyarat

1. Selesaikan langkah-langkah dalam Menyiapkan.
2. Buat proyek di Device Farm. Ikuti instruksi diMembuat proyek di AWS Device Farm, dan kemudian kembali ke halaman ini.

Mengatur batas waktu eksekusi untuk sebuah proyek

1. Masuk ke konsol Device Farm di https://console.aws.amazon.com/devicefarm.
2. Pada panel navigasi Device Farm, pilih Pengujian Perangkat Seluler, lalu pilih Proyek.
3. Jika Anda sudah memiliki proyek, pilih dari daftar. Jika tidak, pilih Proyek baru, masukkan nama untuk proyek Anda, lalu pilih Kirim.
4. Pilih Pengaturan proyek.
5. Pada tab Umum, untuk batas waktu Eksekusi, masukkan nilai atau gunakan bilah geser.
6. Pilih Simpan.

   Semua pengujian yang berjalan di proyek Anda sekarang menggunakan nilai batas waktu eksekusi yang Anda tentukan, kecuali jika Anda mengganti nilai batas waktu saat Anda menjadwalkan proses.

Mengatur batas waktu eksekusi untuk uji coba

1. Masuk ke konsol Device Farm di https://console.aws.amazon.com/devicefarm.
2. Pada panel navigasi Device Farm, pilih Pengujian Perangkat Seluler, lalu pilih Proyek.
3. Jika Anda sudah memiliki proyek, pilih dari daftar. Jika tidak, pilih Proyek baru, masukkan nama untuk proyek Anda, lalu pilih Kirim.
4. Pilih Buat proses baru.
5. Ikuti langkah-langkah untuk memilih aplikasi, mengonfigurasi pengujian, memilih perangkat, dan menentukan status perangkat.

6. Pada Review dan mulai jalankan, untuk Setel batas waktu eksekusi, masukkan nilai atau gunakan bilah geser.
7. Pilih Konfirmasi dan mulai jalankan.

Mensimulasikan koneksi dan kondisi jaringan untuk AWS Device Farm Anda berjalan

Anda dapat menggunakan pembentukan jaringan untuk mensimulasikan koneksi dan kondisi jaringan saat menguji aplikasi Android, iOS, dan web Anda di Device Farm. Misalnya, Anda dapat mensimulasikan konektivitas internet lossy atau intermiten.

Saat Anda membuat proses menggunakan pengaturan jaringan default, setiap perangkat memiliki koneksi Wi-Fi penuh tanpa hambatan dengan konektivitas internet. Saat Anda menggunakan pembentukan jaringan, Anda dapat mengubah koneksi Wi-Fi untuk menentukan profil jaringan seperti 3G atau Lossy WiFi yang mengontrol throughput, delay, jitter, dan loss untuk lalu lintas masuk dan keluar.

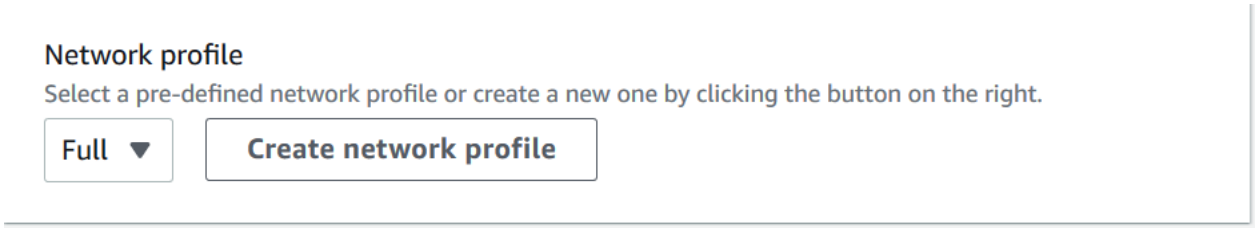
Topik

Siapkan pembentukan jaringan saat menjadwalkan uji coba

Saat Anda menjadwalkan proses, Anda dapat memilih dari salah satu profil Device Farm-curated, atau Anda dapat membuat dan mengelola profil Anda sendiri.

1. Dari proyek Device Farm apa pun, pilih Buat proses baru.

   Jika Anda belum memiliki proyek, lihatMembuat proyek di AWS Device Farm.
2. Pilih aplikasi Anda, lalu pilih Berikutnya.
3. Konfigurasikan pengujian Anda, lalu pilih Berikutnya.
4. Pilih perangkat Anda, lalu pilih Berikutnya.
5. Di bagian Pengaturan lokasi dan jaringan, pilih profil jaringan atau pilih Buat profil jaringan untuk membuat profil Anda sendiri.

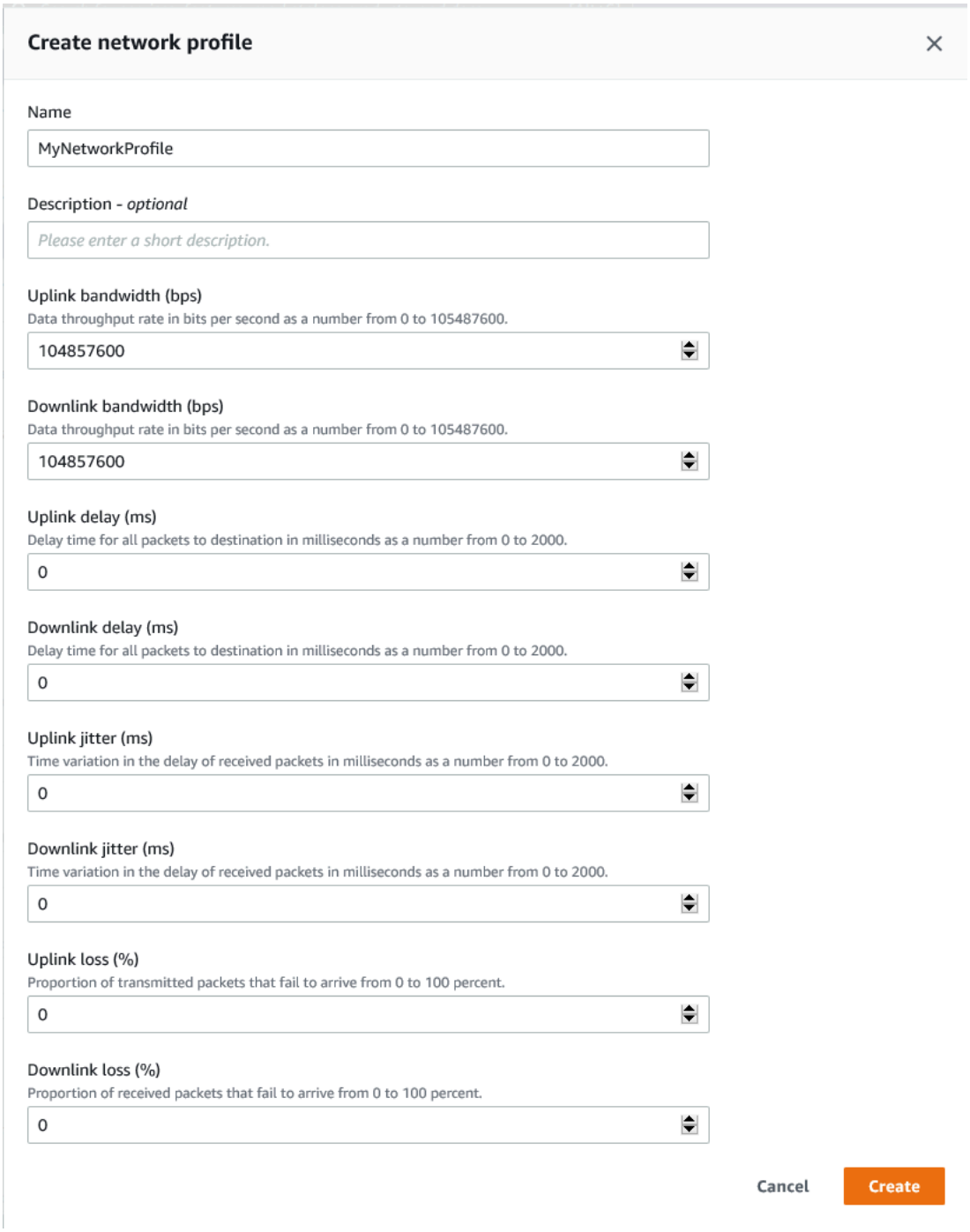


6. Pilih Berikutnya.
7. Tinjau dan mulai uji coba Anda.

Buat profil jaringan

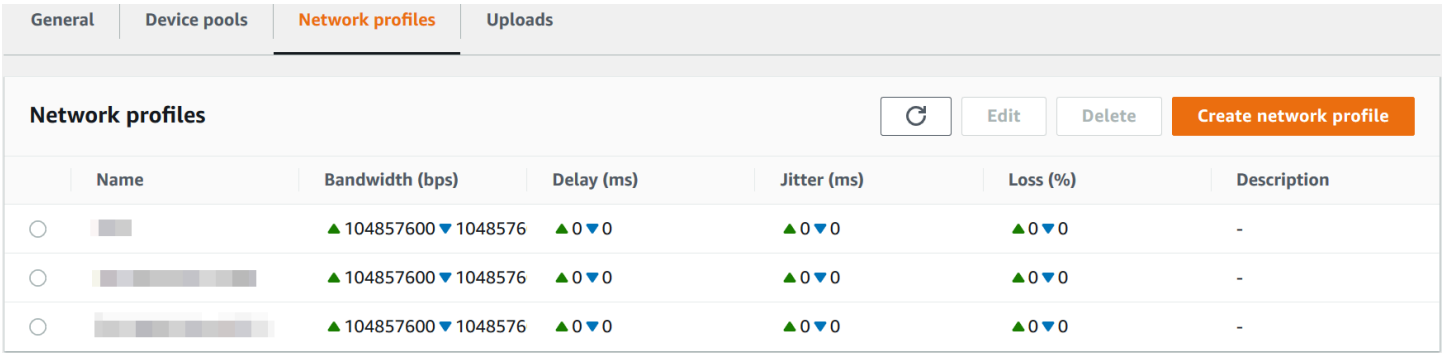
Saat Anda membuat uji coba, Anda dapat membuat profil jaringan.

1. Pilih Buat profil jaringan.



2. Masukkan nama dan pengaturan untuk profil jaringan Anda.
3. Pilih Buat.
4. Selesai membuat test run Anda dan mulai lari.

Setelah Anda membuat profil jaringan, Anda akan dapat melihat dan mengelolanya di halaman pengaturan Proyek.



Ubah kondisi jaringan selama pengujian

Anda dapat memanggil API dari host perangkat Anda menggunakan kerangka kerja seperti Appium untuk mensimulasikan kondisi jaringan dinamis seperti pengurangan bandwidth selama pengujian dijalankan. Untuk informasi selengkapnya, lihat CreateNetworkProfile.

Menghentikan proses di AWS Device Farm

Anda mungkin ingin berhenti berlari setelah Anda memulainya. Misalnya, jika Anda melihat masalah saat pengujian sedang berjalan, Anda mungkin ingin memulai ulang proses dengan skrip pengujian yang diperbarui.

Anda dapat menggunakan konsol Device Farm AWS CLI, atau API untuk menghentikan proses.

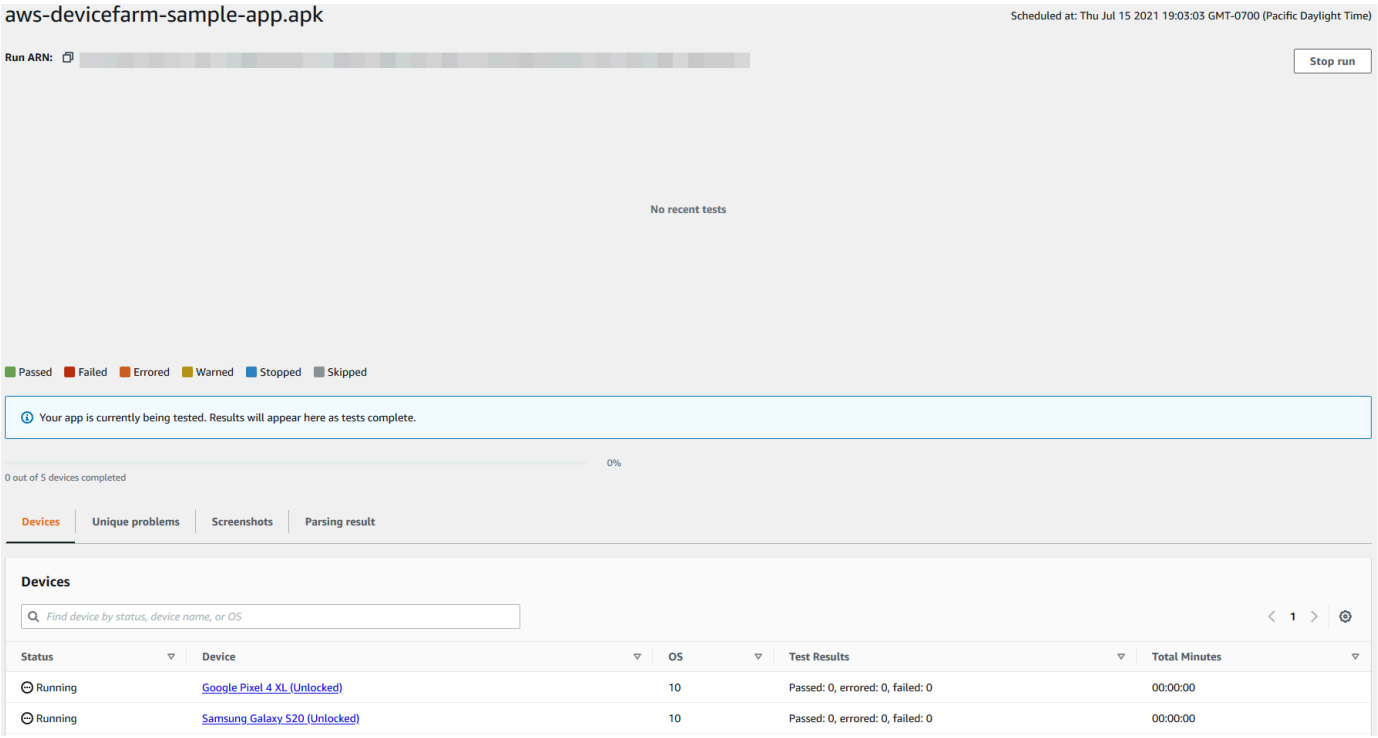
Topik

- Hentikan lari (konsol)
- Hentikan lari (AWS CLI)
- Hentikan lari (API)

Hentikan lari (konsol)

1. Masuk ke konsol Device Farm di https://console.aws.amazon.com/devicefarm.
2. Pada panel navigasi Device Farm, pilih Pengujian Perangkat Seluler, lalu pilih Proyek.
3. Pilih proyek tempat Anda menjalankan uji coba aktif.
4. Pada halaman Pengujian otomatis, pilih uji coba.

   Ikon yang tertunda atau berjalan akan muncul di sebelah kiri nama perangkat.

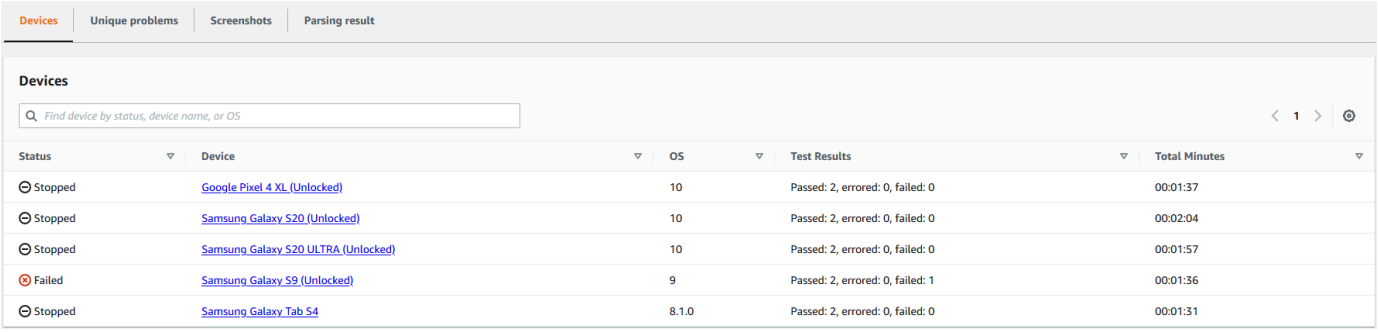


5. Pilih Stop run.

Setelah waktu yang singkat, ikon dengan lingkaran merah dengan minus di dalamnya muncul di sebelah nama perangkat. Ketika proses telah dihentikan, warna ikon berubah dari merah menjadi hitam.

> **Important**
>
> Jika pengujian telah dijalankan, Device Farm tidak dapat menghentikannya. Jika pengujian sedang berlangsung, Device Farm menghentikan pengujian. Total menit di mana Anda akan ditagih muncul di bagian Perangkat. Selain itu, Anda juga akan ditagih untuk total menit yang dibutuhkan Device Farm untuk menjalankan setup suite dan teardown suite. Untuk informasi selengkapnya, lihat Harga Device Farm.

Gambar berikut menunjukkan contoh bagian Perangkat setelah uji coba berhasil dihentikan.



Hentikan lari (AWS CLI)

Anda dapat menjalankan perintah berikut untuk menghentikan uji coba yang ditentukan, di mana *myARN* Amazon Resource Name (ARN) dari uji coba.

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

Anda akan melihat output yang serupa dengan yang berikut:

```
{
    "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
        }
    }
}
```

Untuk mendapatkan ARN dari proses Anda, gunakan perintah. `list-runs` Output harus serupa dengan yang berikut ini:

```
{
    "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
            }
        }
    ]
}
```

Untuk informasi tentang menggunakan Device Farm dengan AWS CLI, lihatAWS CLI referensi.

Hentikan lari (API)

- Panggil StopRunoperasi ke uji coba.

Untuk informasi tentang menggunakan Device Farm API, lihatMengotomatisasi Device Farm.

Melihat daftar proses di AWS Device Farm

Anda dapat menggunakan konsol Device Farm AWS CLI, atau API untuk melihat daftar proses proyek.

Topik

- Lihat daftar proses (konsol)
- Lihat daftar run (AWS CLI)
- Melihat daftar run (API)

Lihat daftar proses (konsol)

1. Masuk ke konsol Device Farm di https://console.aws.amazon.com/devicefarm.
2. Pada panel navigasi Device Farm, pilih Pengujian Perangkat Seluler, lalu pilih Proyek.
3. Dalam daftar proyek, pilih proyek yang sesuai dengan daftar yang ingin Anda lihat.

> **Tip**
>
> Anda dapat menggunakan bilah pencarian untuk memfilter daftar proyek berdasarkan nama.

Lihat daftar run (AWS CLI)

- Jalankan perintah list-runs.

  Untuk melihat informasi tentang satu proses, jalankan get-runperintah.

Untuk informasi tentang menggunakan Device Farm dengan AWS CLI, lihatAWS CLI referensi.

Melihat daftar run (API)

- Panggil `ListRuns`API.

  Untuk melihat informasi tentang satu proses, panggil `GetRun`API.

Untuk informasi tentang Device Farm API, lihatMengotomatisasi Device Farm.

Membuat kumpulan perangkat di AWS Device Farm

Anda dapat menggunakan konsol Device Farm AWS CLI, atau API untuk membuat kumpulan perangkat.

Topik

Prasyarat

- Buat run di konsol Device Farm. Ikuti petunjuk dalam Membuat uji coba di Device Farm. Saat Anda masuk ke halaman Pilih perangkat, lanjutkan dengan instruksi di bagian ini.

Buat kumpulan perangkat (konsol)

1. Pada halaman Proyek, pilih proyek Anda. Di halaman Detail proyek, pilih Pengaturan proyek. Di tab Device pool, pilih Create device pool.
2. Untuk Nama, masukkan nama yang membuat kumpulan perangkat ini mudah diidentifikasi.
3. Untuk Deskripsi, masukkan deskripsi yang membuat kumpulan perangkat ini mudah diidentifikasi.
4. Jika Anda ingin menggunakan satu atau beberapa kriteria pemilihan untuk perangkat di kumpulan perangkat ini, lakukan hal berikut:

   a. Pilih Buat kumpulan perangkat dinamis.

b. Pilih Tambahkan aturan.

c. Untuk Field (daftar drop-down pertama), pilih salah satu dari berikut ini:

- Untuk menyertakan perangkat berdasarkan nama pabrikannya, pilih Produsen Perangkat.
- Untuk memasukkan perangkat berdasarkan faktor bentuknya (tablet atau ponsel), pilih Faktor Formulir.
- Untuk menyertakan perangkat berdasarkan status ketersediaannya berdasarkan pemuatan, pilih Ketersediaan.
- Untuk hanya menyertakan perangkat publik atau pribadi, pilih Jenis Armada.
- Untuk menyertakan perangkat dengan sistem operasinya, pilih Platform.
- Beberapa perangkat memiliki tag label atau deskripsi tambahan tentang perangkat. Anda dapat menemukan perangkat berdasarkan konten labelnya dengan memilih label Instance.
- Untuk menyertakan perangkat berdasarkan versi sistem operasinya, pilih Versi OS.
- Untuk memasukkan perangkat berdasarkan modelnya, pilih Model.

d. Untuk Operator (daftar drop-down kedua), pilih operasi logis (EQUALS, CONTAINS, dll.) Untuk menyertakan perangkat berdasarkan kueri. Misalnya, Anda dapat memilih *`Availability EQUALS AVAILABLE`* untuk menyertakan perangkat yang saat ini memiliki `Available` status.

e. Untuk Nilai (daftar drop-down ketiga), masukkan atau pilih nilai yang ingin Anda tentukan untuk nilai Bidang dan Operator. Nilai terbatas berdasarkan pilihan Bidang Anda. Misalnya, jika Anda memilih Platform for Field, satu-satunya pilihan yang tersedia adalah ANDROID dan IOS. Demikian pula, jika Anda memilih Faktor Formulir untuk Bidang, satu-satunya pilihan yang tersedia adalah PHONE dan TABLET.

f. Untuk menambahkan aturan lain, pilih Tambahkan aturan.

Setelah Anda membuat aturan pertama, dalam daftar perangkat, kotak di samping setiap perangkat yang cocok dengan aturan dipilih. Setelah Anda membuat atau mengubah aturan, dalam daftar perangkat, kotak di samping setiap perangkat yang cocok dengan aturan gabungan tersebut akan dipilih. Perangkat dengan kotak yang dipilih disertakan dalam kumpulan perangkat. Perangkat dengan kotak yang dibersihkan tidak termasuk.

g. Di bawah perangkat Max, masukkan jumlah perangkat yang ingin Anda gunakan di kumpulan perangkat Anda. Jika Anda tidak memasukkan jumlah maksimal perangkat, Device Farm akan memilih semua perangkat dalam armada yang cocok dengan aturan

yang Anda buat. Untuk menghindari biaya tambahan, atur nomor ini ke jumlah yang sesuai dengan eksekusi paralel Anda yang sebenarnya dan persyaratan variasi perangkat.

h. Untuk menghapus aturan, pilih Hapus aturan.

5. Jika Anda ingin menyertakan atau mengecualikan perangkat individual secara manual, lakukan hal berikut:

   a. Pilih Buat kumpulan perangkat statis.

   b. Pilih atau kosongkan kotak di samping setiap perangkat. Anda dapat memilih atau menghapus kotak hanya jika Anda tidak memiliki aturan yang ditentukan.

6. Jika Anda ingin menyertakan atau mengecualikan semua perangkat yang ditampilkan, pilih atau kosongkan kotak di baris header kolom daftar. Jika Anda hanya ingin melihat instance perangkat pribadi, pilih Lihat instans perangkat pribadi saja.

   > **Important**
   >
   > Meskipun Anda dapat menggunakan kotak di baris header kolom untuk mengubah daftar perangkat yang ditampilkan, ini tidak berarti bahwa perangkat yang ditampilkan yang tersisa adalah satu-satunya yang disertakan atau dikecualikan. Untuk mengonfirmasi perangkat mana yang disertakan atau dikecualikan, pastikan untuk menghapus konten semua kotak di baris header kolom, lalu telusuri kotaknya.

7. Pilih Buat.

Buat kumpulan perangkat (AWS CLI)

> **Tip**
>
> Jika Anda tidak memasukkan jumlah maksimal perangkat, Device Farm akan memilih semua perangkat dalam armada yang cocok dengan aturan yang Anda buat. Untuk menghindari biaya tambahan, atur nomor ini ke jumlah yang sesuai dengan eksekusi paralel Anda yang sebenarnya dan persyaratan variasi perangkat.

- Jalankan perintah create-device-pool.

Untuk informasi tentang menggunakan Device Farm dengan AWS CLI, lihatAWS CLI referensi.

Membuat kumpulan perangkat (API)

> **Tip**
>
> Jika Anda tidak memasukkan jumlah maksimal perangkat, Device Farm akan memilih semua perangkat dalam armada yang cocok dengan aturan yang Anda buat. Untuk menghindari biaya tambahan, atur nomor ini ke jumlah yang sesuai dengan eksekusi paralel Anda yang sebenarnya dan persyaratan variasi perangkat.

- Panggil `CreateDevicePool`API.

Untuk informasi tentang menggunakan Device Farm API, lihatMengotomatisasi Device Farm.

Menganalisis hasil pengujian di AWS Device Farm

Di lingkungan pengujian standar, Anda dapat menggunakan konsol Device Farm untuk melihat laporan untuk setiap pengujian dalam proses pengujian. Mempelajari laporan membantu Anda memahami pengujian mana yang lulus atau gagal, dan memberi Anda detail tentang kinerja dan perilaku aplikasi Anda di berbagai konfigurasi perangkat.

Device Farm juga mengumpulkan artefak lain seperti file, log, dan gambar yang dapat Anda unduh saat uji coba selesai. Informasi ini dapat membantu Anda menganalisis bagaimana aplikasi Anda berperilaku pada perangkat nyata, mengidentifikasi masalah atau bug, dan mendiagnosis masalah.

Topik

- Melihat laporan pengujian di Device Farm
- Mengunduh artefak di Device Farm

Melihat laporan pengujian di Device Farm

Gunakan konsol Device Farm untuk melihat laporan pengujian Anda. Untuk informasi selengkapnya, lihat Laporan di AWS Device Farm.

Topik

- Prasyarat

- Lihat laporan
- Status hasil tes Device Farm

Prasyarat

Siapkan uji coba dan verifikasi bahwa itu sudah selesai.

1. Untuk membuat run, lihatMembuat uji coba di Device Farm, dan kemudian kembali ke halaman ini.
2. Verifikasi bahwa proses selesai. Selama uji coba, konsol Device Farm menampilkan ikon tertunda

   untuk proses yang sedang berlangsung. Setiap perangkat yang sedang dijalankan juga akan dimulai dengan ikon yang tertunda, lalu beralih ke

   ikon yang sedang berjalan saat pengujian dimulai. Saat setiap pengujian selesai, ikon hasil pengujian ditampilkan di sebelah nama perangkat. Ketika semua pengujian telah selesai, ikon yang tertunda di sebelah run berubah menjadi ikon hasil pengujian. Untuk informasi selengkapnya, lihat Status hasil tes Device Farm.

Lihat laporan

Anda dapat melihat hasil pengujian di konsol Device Farm.

Topik

- Lihat halaman ringkasan uji coba
- Lihat laporan masalah unik
- Lihat laporan perangkat
- Lihat laporan rangkaian pengujian
- Lihat laporan pengujian
- Melihat informasi log untuk masalah, perangkat, rangkaian, atau pengujian dalam laporan

Lihat halaman ringkasan uji coba

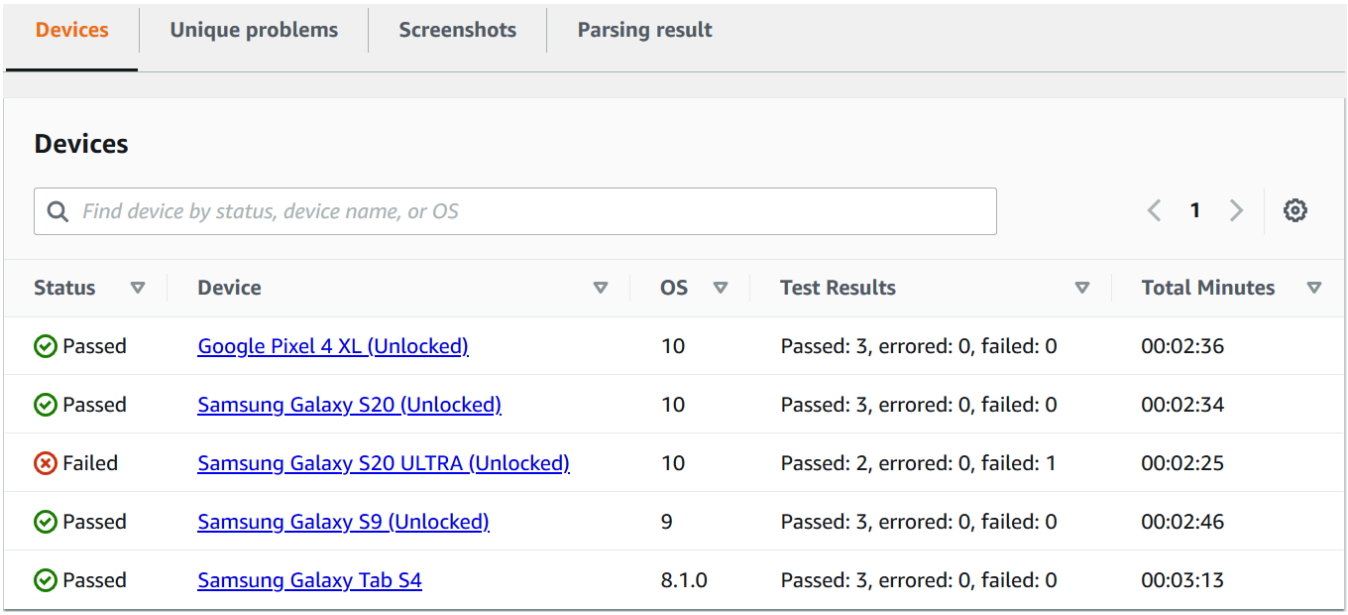
1. Masuk ke konsol Device Farm di https://console.aws.amazon.com/devicefarm.

2. Di panel navigasi, pilih Pengujian Perangkat Seluler, lalu pilih Proyek.
3. Dalam daftar proyek, pilih proyek untuk dijalankan.

> **Tip**
>
> Untuk memfilter daftar proyek berdasarkan nama, gunakan bilah pencarian.

4. Pilih proses yang sudah selesai untuk melihat halaman laporan ringkasannya.
5. Halaman ringkasan uji coba menampilkan ikhtisar hasil pengujian Anda.

- Bagian Masalah unik mencantumkan peringatan dan kegagalan unik. Untuk melihat masalah unik, ikuti instruksi diLihat laporan masalah unik.
- Bagian Perangkat menampilkan jumlah total pengujian, berdasarkan hasil, untuk setiap perangkat.



Dalam contoh ini, ada beberapa perangkat. Pada entri tabel pertama, perangkat Google Pixel 4 XL yang menjalankan Android versi 10 melaporkan tiga pengujian yang berhasil yang membutuhkan waktu 02:36 menit untuk dijalankan.

Untuk melihat hasil berdasarkan perangkat, ikuti petunjuk diLihat laporan perangkat.

- Bagian Screenshots menampilkan daftar tangkapan layar apa pun yang ditangkap Device Farm selama proses dijalankan, dikelompokkan berdasarkan perangkat.
- Di bagian Hasil parsing, Anda dapat mengunduh hasil parsing.

Lihat laporan masalah unik

1. Dalam Masalah unik, pilih masalah yang ingin Anda lihat.
2. Pilih perangkat. Laporan menampilkan informasi tentang masalah tersebut.

   Bagian Video menampilkan rekaman video pengujian yang dapat diunduh.

   Bagian Hasil menampilkan hasil tes. Status direpresentasikan sebagai ikon hasil. Untuk informasi selengkapnya, lihat Status tes individu.

   Bagian Log menampilkan informasi apa pun yang dicatat oleh Device Farm selama pengujian. Untuk melihat informasi ini, ikuti instruksi diMelihat informasi log untuk masalah, perangkat, rangkaian, atau pengujian dalam laporan.

   Tab File menampilkan daftar file terkait pengujian (seperti file log) yang dapat Anda unduh. Untuk mengunduh file, pilih tautan file dalam daftar.

   Tab Screenshots menampilkan daftar tangkapan layar apa pun yang ditangkap Device Farm selama pengujian.

Lihat laporan perangkat

- Di bagian Perangkat, pilih perangkat.

  Bagian Video menampilkan rekaman video pengujian yang dapat diunduh.

  Bagian Suites menampilkan tabel yang berisi informasi tentang suite untuk perangkat.

  Dalam tabel ini, kolom Hasil pengujian merangkum jumlah pengujian berdasarkan hasil untuk setiap rangkaian pengujian yang telah berjalan di perangkat. Data ini juga memiliki komponen grafis. Untuk informasi selengkapnya, lihat Status untuk beberapa tes.

  Untuk melihat hasil lengkap berdasarkan suite, ikuti petunjuk diLihat laporan rangkaian pengujian.

  Bagian Log menampilkan informasi apa pun yang dicatat oleh Device Farm untuk perangkat selama dijalankan. Untuk melihat informasi ini, ikuti instruksi diMelihat informasi log untuk masalah, perangkat, rangkaian, atau pengujian dalam laporan.

Bagian File menampilkan daftar suite untuk perangkat dan file terkait (seperti file log) yang dapat Anda unduh. Untuk mengunduh file, pilih tautan file dalam daftar.

Bagian Screenshots menampilkan daftar tangkapan layar apa pun yang ditangkap Device Farm selama menjalankan perangkat, dikelompokkan berdasarkan suite.

Lihat laporan rangkaian pengujian

1. Di bagian Perangkat, pilih perangkat.
2. Di bagian Suites, pilih suite dari tabel.

Bagian Video menampilkan rekaman video pengujian yang dapat diunduh.

Bagian Tes menampilkan tabel yang berisi informasi tentang pengujian di suite.

Dalam tabel, kolom Hasil tes menampilkan hasilnya. Data ini juga memiliki komponen grafis. Untuk informasi selengkapnya, lihat Status untuk beberapa tes.

Untuk melihat hasil lengkap dengan tes, ikuti instruksi diLihat laporan pengujian.

Bagian Log menampilkan informasi apa pun yang dicatat oleh Device Farm selama menjalankan suite. Untuk melihat informasi ini, ikuti instruksi diMelihat informasi log untuk masalah, perangkat, rangkaian, atau pengujian dalam laporan.

Bagian File menampilkan daftar pengujian untuk suite dan file terkait apa pun (seperti file log) yang dapat Anda unduh. Untuk mengunduh file, pilih tautan file dalam daftar.

Bagian Screenshots menampilkan daftar tangkapan layar apa pun yang ditangkap Device Farm selama menjalankan suite, dikelompokkan berdasarkan pengujian.

Lihat laporan pengujian

1. Di bagian Perangkat, pilih perangkat.
2. Di bagian Suites, pilih suite.
3. Di bagian Tes, pilih tes.
4. Bagian Video menampilkan rekaman video pengujian yang dapat diunduh.

Bagian Hasil menampilkan hasil tes. Status direpresentasikan sebagai ikon hasil. Untuk informasi selengkapnya, lihat Status tes individu.

Bagian Log menampilkan informasi apa pun yang dicatat oleh Device Farm selama pengujian. Untuk melihat informasi ini, ikuti instruksi diMelihat informasi log untuk masalah, perangkat, rangkaian, atau pengujian dalam laporan.

Tab File menampilkan daftar file terkait pengujian (seperti file log) yang dapat Anda unduh. Untuk mengunduh file, pilih tautan file dalam daftar.

Tab Screenshots menampilkan daftar tangkapan layar apa pun yang ditangkap Device Farm selama pengujian.

Melihat informasi log untuk masalah, perangkat, rangkaian, atau pengujian dalam laporan

Bagian Log menampilkan informasi berikut:

- Sumber mewakili sumber entri log. Nilai yang mungkin termasuk:
  - Harness mewakili entri log yang dibuat Device Farm. Entri log ini biasanya dibuat selama acara start dan stop.
  - Perangkat mewakili entri log yang dibuat perangkat. Untuk Android, entri log ini kompatibel dengan logcat-. Untuk iOS, entri log ini kompatibel dengan syslog.
  - Tes merupakan entri log yang dibuat oleh pengujian atau kerangka pengujian.
- Waktu mewakili waktu yang telah berlalu antara entri log pertama dan entri log ini. Waktu dinyatakan dalam *MM:SS.SSS* format, di mana *M* mewakili menit dan *S* mewakili detik.
- PID merupakan pengidentifikasi proses (PID) yang menciptakan entri log. Semua entri log yang dibuat oleh aplikasi pada perangkat memiliki PID yang sama.
- Level mewakili tingkat logging untuk entri log. Misalnya, `Logger.debug("This is a message!")` mencatat LevelDebug. Ini adalah nilai yang mungkin:
  - Waspada
  - Kritis
  - Debug
  - Darurat
  - Kesalahan
  - Errored

- Gagal
- Info
- Internal
- Pemberitahuan
- Lulus
- Dilewati
- Stopped
- Verbose
- Diperingatkan
- Peringatan

- Tag mewakili metadata arbitrer untuk entri log. Misalnya, Android logcat dapat menggunakan ini untuk menjelaskan bagian mana dari sistem yang membuat entri log (misalnya,`ActivityManager`).
- Pesan mewakili pesan atau data untuk entri log. Misalnya, `Logger.debug("Hello, World!")` mencatat Pesan dari`"Hello, World!"`.

Untuk menampilkan hanya sebagian dari informasi:

- Untuk menampilkan semua entri log yang cocok dengan nilai untuk kolom tertentu, masukkan nilai ke dalam bilah pencarian. Misalnya, untuk menampilkan semua entri log dengan nilai Sumber`Harness`, masukkan **`Harness`** di bilah pencarian.
- Untuk menghapus semua karakter dari kotak header kolom, pilih X di kotak header kolom itu. Menghapus semua karakter dari kotak header kolom sama dengan memasukkan ***** kotak header kolom itu.

Untuk mengunduh semua informasi log untuk perangkat, termasuk semua suite dan pengujian yang Anda jalankan, pilih Unduh log.

Status hasil tes Device Farm

Konsol Device Farm menampilkan ikon yang membantu Anda menilai status uji coba selesai dengan cepat. Untuk informasi selengkapnya tentang pengujian di Device Farm, lihatLaporan di AWS Device Farm.

Topik

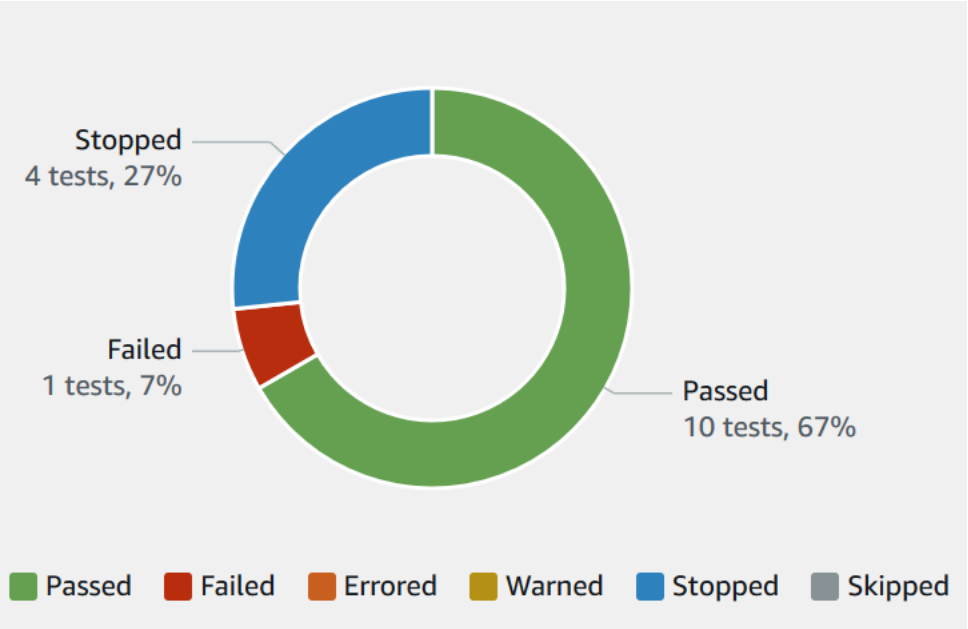
Status tes individu

Untuk laporan yang menjelaskan pengujian individual, Device Farm menampilkan ikon yang mewakili status hasil pengujian:

Deskripsi	Ikon
Tes berhasil.	
Tes gagal.	
Device Farm melewatkan tes.	
Tes berhenti.	
Device Farm mengembalikan peringatan.	
Device Farm mengembalikan kesalahan.	

Status untuk beberapa tes

Jika Anda memilih proses selesai, Device Farm menampilkan grafik ringkasan yang menunjukkan persentase pengujian di berbagai status.



Misalnya, grafik hasil uji coba ini menunjukkan bahwa run memiliki 4 tes berhenti, 1 tes gagal, dan 10 tes yang berhasil.

Grafik selalu diberi kode warna dan diberi label.

Mengunduh artefak di Device Farm

Device Farm mengumpulkan artefak seperti laporan, file log, dan gambar untuk setiap pengujian yang dijalankan.

Anda dapat mengunduh artefak yang dibuat selama uji coba:

Berkas

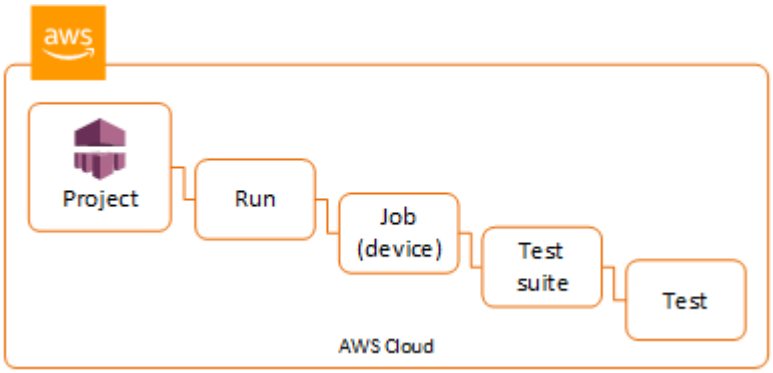
File yang dihasilkan selama uji coba termasuk laporan Device Farm. Untuk informasi selengkapnya, lihat Melihat laporan pengujian di Device Farm.

Beberapa catatan

Output dari setiap tes dalam uji coba.

Tangkapan layar

Gambar layar direkam untuk setiap pengujian dalam uji coba.



Unduh artefak (konsol)

1. Pada halaman laporan uji coba, dari Perangkat, pilih perangkat seluler.
2. Untuk mengunduh file, pilih salah satu dari File.
3. Untuk mengunduh log dari uji coba Anda, dari Log, pilih Unduh log.
4. Untuk mengunduh tangkapan layar, pilih tangkapan layar dari Screenshots.

Untuk informasi selengkapnya tentang mengunduh artefak di lingkungan pengujian khusus, lihatMengunduh artefak di lingkungan pengujian khusus.

Unduh artefak ()AWS CLI

Anda dapat menggunakan AWS CLI untuk membuat daftar artefak uji coba Anda.

Topik

- Langkah 1: Dapatkan Nama Sumber Daya Amazon Anda (ARN)
- Langkah 2: Daftar artefak Anda
- Langkah 3: Unduh artefak Anda

Langkah 1: Dapatkan Nama Sumber Daya Amazon Anda (ARN)

Anda dapat membuat daftar artefak Anda berdasarkan run, job, test suite, atau test. Anda membutuhkan ARN yang sesuai. Tabel ini menunjukkan input ARN untuk masing-masing perintah AWS CLI daftar:

AWS CLI Daftar Perintah	ARN yang dibutuhkan
list-projects	Perintah ini mengembalikan semua proyek dan tidak memerlukan ARN.
list-runs	`project`
list-jobs	`run`
list-suites	`job`
list-tests	`suite`

Misalnya, untuk menemukan ARN pengujian, jalankan list-tests menggunakan rangkaian pengujian ARN Anda sebagai parameter input.

Contoh:

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

Respons termasuk ARN tes untuk setiap tes dalam rangkaian pengujian.

```
{
    "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
            }
        }
    ]
}
```

Langkah 2: Daftar artefak Anda

Perintah AWS CLI daftar-artefak mengembalikan daftar artefak, seperti file, tangkapan layar, dan log. Setiap artefak memiliki URL sehingga Anda dapat mengunduh file.

- Panggilan list-artifacts yang menentukan ARN run, job, test suite, atau test. Tentukan jenis FILE, LOG, atau SCREENSHOT.

  Contoh ini mengembalikan URL unduhan untuk setiap artefak yang tersedia untuk pengujian individual:

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

  Respons berisi URL unduhan untuk setiap artefak.

  ```
  {
      "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",
          }
      ]
  }
  ```

Langkah 3: Unduh artefak Anda

- Unduh artefak Anda menggunakan URL dari langkah sebelumnya. Contoh ini digunakan curl untuk mengunduh file keluaran Android Appium Java:

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

Unduh artefak (API)

ListArtifactsMetode Device Farm API menampilkan daftar artefak, seperti file, tangkapan layar, dan log. Setiap artefak memiliki URL sehingga Anda dapat mengunduh file.

Mengunduh artefak di lingkungan pengujian khusus

Dalam lingkungan pengujian khusus, Device Farm mengumpulkan artefak seperti laporan kustom, file log, dan gambar. Artefak ini tersedia untuk setiap perangkat dalam uji coba.

Anda dapat mengunduh artefak ini yang dibuat selama uji coba:

Uji keluaran spesifikasi

Output dari menjalankan perintah dalam file YAMM spesifikasi pengujian.

Artefak pelanggan

File zip yang berisi artefak dari uji coba. Ini dikonfigurasi di bagian artefak: dari file YAMM spesifikasi pengujian Anda.

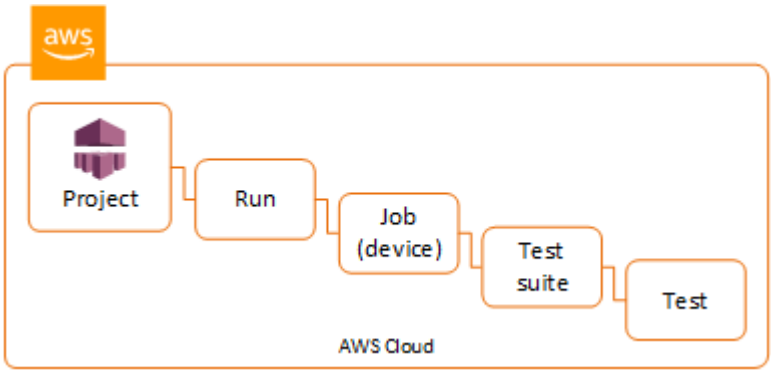
Uji skrip shell spesifikasi

File skrip shell perantara yang dibuat dari file YAMM Anda. Karena digunakan dalam uji coba, file skrip shell dapat digunakan untuk men-debug file YAMM.

Uji file spesifikasi

File YAMM yang digunakan dalam uji coba.

Untuk informasi selengkapnya, lihat Mengunduh artefak di Device Farm.


aws
Project
Run
Job
(device)
Test
suite
Test
AWS Cloud

Menandai sumber daya AWS Device Farm

AWS Device Farm bekerja dengan API Penandaan AWS Resource Groups. API ini memungkinkan Anda mengelola sumber daya di AWS akun Anda dengan tag. Anda dapat menambahkan tag ke sumber daya, seperti proyek dan uji coba.

Anda dapat menggunakan tag untuk:

- Atur tagihan AWS Anda untuk mencerminkan struktur biaya Anda sendiri. Untuk melakukannya, daftar untuk mendapatkan tagihan akun AWS Anda dengan menyertakan nilai kunci tag. Lalu, untuk melihat biaya sumber daya gabungan, kelola informasi penagihan Anda sesuai dengan sumber daya Anda dengan nilai kunci tag yang sama. Misalnya, Anda dapat menandai beberapa sumber daya dengan nama aplikasi, dan kemudian mengatur informasi penagihan Anda untuk melihat total biaya aplikasi tersebut di beberapa layanan. Untuk informasi selengkapnya, lihat Alokasi Biaya dan Pemberian Tanda di Tentang Manajemen Penagihan & Biaya AWS.
- Kontrol akses melalui kebijakan IAM. Untuk melakukannya, buat kebijakan yang memungkinkan akses ke sumber daya atau kumpulan sumber daya menggunakan kondisi nilai tag.
- Identifikasi dan kelola run yang memiliki properti tertentu sebagai tag, seperti cabang yang digunakan untuk pengujian.

Untuk informasi selengkapnya tentang tagging resource, lihat whitepaper Tagging Best Practices.

Topik

- Menandai sumber daya
- Mencari sumber daya berdasarkan tag
- Menghapus tag dari sumber daya

Menandai sumber daya

AWS Resource Group Tagging API memungkinkan Anda menambahkan, menghapus, atau memodifikasi tag pada sumber daya. Untuk informasi selengkapnya, lihat Referensi API Penandaan AWS Resource Group.

Untuk menandai sumber daya, gunakan `TagResources`operasi dari `resourcegroupstaggingapi` titik akhir. Operasi ini mengambil daftar ARNs dari layanan yang

didukung dan daftar pasangan kunci-nilai. Nilai ini bersifat opsional. String kosong menunjukkan bahwa seharusnya tidak ada nilai untuk tag itu. Misalnya, contoh Python berikut menandai serangkaian proyek ARNs dengan tag `build-config` dengan nilai: `release`

```
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"})
```

Nilai tag tidak diperlukan. Untuk menetapkan tag tanpa nilai, gunakan string kosong ("") saat menentukan nilai. Sebuah tag hanya dapat memiliki satu nilai. Nilai sebelumnya yang dimiliki tag untuk sumber daya akan ditimpa dengan nilai baru.

Mencari sumber daya berdasarkan tag

Untuk mencari sumber daya berdasarkan tag mereka, gunakan `GetResources` operasi dari `resourcegrouptaggingapi` titik akhir. Operasi ini mengambil serangkaian filter, tidak ada yang diperlukan, dan mengembalikan sumber daya yang sesuai dengan kriteria yang diberikan. Tanpa filter, semua sumber daya yang ditandai dikembalikan. `GetResources`Operasi ini memungkinkan Anda untuk memfilter sumber daya berdasarkan

- Nilai tanda
- Jenis sumber daya (misalnya,`devicefarm:run`)

Untuk informasi selengkapnya, lihat Referensi API Penandaan AWS Resource Group.

Contoh berikut mencari sesi pengujian browser desktop Device Farm (`devicefarm:testgrid-session`resource) dengan tag `stack` yang memiliki nilai`production`:

```
import boto3
client = boto3.client('resourcegroupstaggingapi')
sessions = client.get_resources(ResourceTypeFilters=['devicefarm:testgrid-session'],
```

```
                                TagFilters=[
                                  {"Key":"stack","Values":["production"]}
                                ])
```

Menghapus tag dari sumber daya

Untuk menghapus tag, gunakan `UntagResources` operasi, tentukan daftar sumber daya dan tag yang akan dihapus:

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

Uji kerangka kerja dan pengujian bawaan di AWS Device Farm

Bagian ini menjelaskan dukungan Device Farm untuk kerangka kerja pengujian dan tipe pengujian bawaan.

Device Farm menjalankan pengujian otomatis dengan mengunggah aplikasi dan pengujian ke bucket Amazon S3 aman yang dikelola oleh layanan. Setelah diunggah, itu memutar infrastruktur yang mendasarinya, termasuk host pengujian yang dikelola layanan, dan menjalankan pengujian secara paralel pada beberapa perangkat. Hasil pengujian disimpan dalam bucket S3 yang dikelola layanan. Arsitektur ini disebut eksekusi sisi layanan, dan merupakan cara cepat dan efisien untuk menjalankan pengujian pada host yang secara fisik dekat dengan perangkat, tanpa perlu mengelola sendiri infrastruktur host pengujian. Pendekatan ini berskala baik untuk pengujian pada banyak perangkat secara independen, serta pengujian dari konteks CI/CD pipa.

Untuk informasi selengkapnya tentang cara Device Farm menjalankan pengujian, lihatUji lingkungan di AWS Device Farm.

> **Note**
>
> Untuk penguji Appium, Anda mungkin lebih suka menjalankan tes Appium dari lingkungan lokal Anda. Dengan sesi akses jarak jauh, Anda dapat menjalankan pengujian Appium sisi klien. Untuk informasi lebih lanjut, silakan lihat pengujian Appium sisi klien.

Kerangka pengujian

Device Farm mendukung kerangka kerja pengujian otomatisasi seluler ini:

Kerangka kerja pengujian aplikasi Android

- Tes Appium otomatis
- Instrumentasi

Kerangka kerja pengujian aplikasi iOS

- Tes Appium otomatis

- XCTest
- XCTest UI

Kerangka kerja pengujian aplikasi web

Aplikasi web didukung menggunakan Appium. Untuk informasi lebih lanjut tentang membawa tes Anda ke Appium, lihat. Jalankan pengujian Appium secara otomatis di Device Farm

Kerangka kerja di lingkungan pengujian khusus

Device Farm tidak memberikan dukungan untuk menyesuaikan lingkungan pengujian untuk XCTest kerangka kerja. Untuk informasi selengkapnya, lihat Lingkungan pengujian khusus di AWS Device Farm.

Dukungan versi Appium

Untuk pengujian yang berjalan di lingkungan khusus, Device Farm mendukung Appium versi 1. Untuk informasi selengkapnya, lihat Uji lingkungan di AWS Device Farm.

Jenis pengujian bawaan

Dengan pengujian bawaan, Anda dapat menguji aplikasi di beberapa perangkat tanpa harus menulis dan memelihara skrip otomatisasi pengujian. Device Farm menawarkan satu jenis pengujian bawaan:

- Bawaan: fuzz (Android dan iOS)

Jalankan pengujian Appium secara otomatis di Device Farm

> **Note**
>
> Halaman ini mencakup menjalankan pengujian Appium di lingkungan eksekusi sisi server terkelola Device Farm. Untuk menjalankan pengujian Appium dari lingkungan sisi klien lokal Anda selama sesi akses jarak jauh, lihat pengujian Appium sisi klien.

Bagian ini menjelaskan cara mengonfigurasi, mengemas, dan mengunggah pengujian Appium agar berjalan di lingkungan sisi server terkelola Device Farm. Appium adalah alat open source untuk

mengotomatiskan aplikasi web asli dan seluler. Untuk informasi lebih lanjut, lihat Pengantar Appium di situs web Appium.

Untuk contoh aplikasi dan tautan ke pengujian yang berfungsi, lihat Aplikasi Sampel Device Farm untuk Android dan Aplikasi Sampel Device Farm untuk iOS aktif GitHub.

Untuk informasi selengkapnya tentang pengujian di Device Farm dan cara kerja sisi server, lihat. Uji kerangka kerja dan pengujian bawaan di AWS Device Farm

Memilih versi Appium

> **Note**
>
> Support untuk versi Appium tertentu, driver Appium, atau pemrograman SDKs akan bergantung pada perangkat dan host uji yang dipilih untuk uji coba.

Host pengujian Device Farm telah diinstal sebelumnya dengan Appium untuk mengaktifkan penyiapan pengujian yang lebih cepat untuk kasus penggunaan yang lebih mudah. Namun, penggunaan file spesifikasi pengujian memungkinkan Anda menginstal versi Appium yang berbeda jika diperlukan.

Skenario 1: Versi Appium yang telah dikonfigurasi sebelumnya

Device Farm hadir pra-konfigurasi dengan versi server Appium yang berbeda berdasarkan host uji. Host dilengkapi dengan perkakas yang memungkinkan versi pra-konfigurasi dengan driver default platform perangkat (UiAutomator2 untuk Android, dan untuk XCUITest iOS).

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

Untuk melihat daftar perangkat lunak yang didukung, lihat topik diPerangkat lunak yang didukung dalam lingkungan pengujian khusus.

Skenario 2: Versi Appium Kustom

Untuk memilih versi kustom Appium, gunakan `npm` perintah untuk menginstalnya. Contoh berikut menunjukkan cara menginstal versi terbaru Appium 2.

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

Skenario 3: Appium di host iOS Legacy

PadaHost uji iOS lama, Anda dapat memilih versi Appium tertentu dengan. `avm` Misalnya, untuk menggunakan `avm` perintah untuk menyetel versi server Appium`2.1.2`, tambahkan perintah ini ke file YAMM spesifikasi pengujian Anda.

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

Memilih WebDriverAgent versi untuk pengujian iOS

Untuk menjalankan tes Appium pada perangkat iOS, penggunaan WebDriverAgent diperlukan. Aplikasi ini harus ditandatangani agar dapat diinstal pada perangkat iOS. Device Farm menyediakan versi yang telah ditandatangani sebelumnya WebDriverAgent yang tersedia selama lingkungan pengujian kustom dijalankan.

Cuplikan kode berikut dapat digunakan untuk memilih WebDriverAgent versi di Device Farm dalam file spesifikasi pengujian Anda yang kompatibel dengan versi Driver XCTest UI Anda..

```
phases:
  pre_test:
    commands:
      - |-
        APPIUM_DRIVER_VERSION=$(appium driver list --installed --json | jq -r
 ".xcuitest.version" | cut -d "." -f 1);
        CORRESPONDING_APPIUM_WDA=$(env | grep
 "DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V${APPIUM_DRIVER_VERSION}")
        if [[ ! -z "$APPIUM_DRIVER_VERSION" ]] && [[ ! -z
 "$CORRESPONDING_APPIUM_WDA" ]]; then
          echo "Using Device Farm's prebuilt WDA version ${APPIUM_DRIVER_VERSION}.x,
 which corresponds with your driver";
```

```
        DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH=$(echo $CORRESPONDING_APPIUM_WDA |
cut -d "=" -f2)
      else
        LATEST_SUPPORTED_WDA_VERSION=$(env | grep
"DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V" | sort -V -r | head -n 1)
        echo "Unknown driver version $APPIUM_DRIVER_VERSION; falling back to the
Device Farm default version of $LATEST_SUPPORTED_WDA_VERSION";
        DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH=$(echo $LATEST_SUPPORTED_WDA_VERSION
| cut -d "=" -f2)
      fi;
```

Untuk informasi selengkapnya tentang WebDriverAgent, lihat dokumentasi Appium.

Mengintegrasikan tes Appium dengan Device Farm

Gunakan petunjuk berikut untuk mengintegrasikan pengujian Appium dengan AWS Device Farm. Untuk informasi selengkapnya tentang menggunakan pengujian Appium di Device Farm, lihat. Jalankan pengujian Appium secara otomatis di Device Farm

Konfigurasikan paket uji Appium Anda

Gunakan petunjuk berikut untuk mengonfigurasi paket pengujian Anda.

Java (JUnit)

1. Ubah `pom.xml` untuk mengatur kemasan ke file JAR:

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

2. Ubah `pom.xml` `maven-jar-plugin` untuk digunakan untuk membangun pengujian Anda menjadi file JAR.

   Plugin berikut membangun kode sumber pengujian Anda (apa pun di `src/test` direktori) ke dalam file JAR:

   ```
   <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. Ubah `pom.xml` untuk digunakan `maven-dependency-plugin` untuk membangun dependensi sebagai file JAR.

   Plugin berikut menyalin dependensi Anda ke direktori: `dependency-jars`

```
<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. Simpan rakitan XHTML berikut ke`src/main/assembly/zip.xml`.

   XHTML berikut adalah definisi perakitan yang, ketika dikonfigurasi, menginstruksikan Maven untuk membuat file.zip yang berisi segala sesuatu di root direktori keluaran build Anda dan direktori: `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. Ubah `pom.xml` untuk digunakan `maven-assembly-plugin` untuk mengemas tes dan semua dependensi menjadi satu file.zip.

   Plugin berikut menggunakan rakitan sebelumnya untuk membuat file.zip bernama `zip-with-dependencies` di direktori keluaran build setiap kali mvn package dijalankan:

```
<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**
>
> Jika Anda menerima kesalahan yang mengatakan anotasi tidak didukung di 1.3, tambahkan yang berikut ini kepom.xml:
>
> ```
> <plugin>
>   <artifactId>maven-compiler-plugin</artifactId>
>   <configuration>
>     <source>1.7</source>
>     <target>1.7</target>
>   </configuration>
> </plugin>
> ```

Java (TestNG)

1. Ubah pom.xml untuk mengatur kemasan ke file JAR:

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

2. Ubah pom.xml maven-jar-plugin untuk digunakan untuk membangun pengujian Anda menjadi file JAR.

   Plugin berikut membangun kode sumber pengujian Anda (apa pun di src/test direktori) ke dalam file JAR:

   ```
   <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. Ubah `pom.xml` untuk digunakan `maven-dependency-plugin` untuk membangun dependensi sebagai file JAR.

   Plugin berikut menyalin dependensi Anda ke direktori: `dependency-jars`

```
<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. Simpan rakitan XHTML berikut ke`src/main/assembly/zip.xml`.

   XHTML berikut adalah definisi perakitan yang, ketika dikonfigurasi, menginstruksikan Maven untuk membuat file.zip yang berisi segala sesuatu di root direktori keluaran build Anda dan direktori: `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. Ubah `pom.xml` untuk digunakan `maven-assembly-plugin` untuk mengemas tes dan semua dependensi menjadi satu file.zip.

   Plugin berikut menggunakan rakitan sebelumnya untuk membuat file.zip bernama `zip-with-dependencies` di direktori keluaran build setiap kali mvn package dijalankan:

```
<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**
>
> Jika Anda menerima kesalahan yang mengatakan anotasi tidak didukung di 1.3, tambahkan yang berikut ini kepom.xml:
>
> ```
> <plugin>
>   <artifactId>maven-compiler-plugin</artifactId>
>   <configuration>
>     <source>1.7</source>
>     <target>1.7</target>
>   </configuration>
> </plugin>
> ```

Node.JS

Untuk mengemas pengujian Appium Node.js dan mengunggahnya ke Device Farm, Anda harus menginstal yang berikut ini di mesin lokal Anda:

- Manajer Versi Node (nvm)

  Gunakan alat ini saat Anda mengembangkan dan mengemas pengujian Anda sehingga dependensi yang tidak perlu tidak disertakan dalam paket pengujian Anda.
- Node.js
- npm-bundle (diinstal secara global)

1. Verifikasi bahwa nvm ada

   ```
   command -v nvm
   ```

Anda harus melihat `nvm` sebagai output.

Untuk informasi lebih lanjut, lihat nvm di. GitHub

2. Jalankan perintah ini untuk menginstal Node.js:

```
nvm install node
```

Anda dapat menentukan versi tertentu dari Node.js:

```
nvm install 11.4.0
```

3. Verifikasi bahwa versi Node yang benar sedang digunakan:

```
node -v
```

4. Instal npm-bundle secara global:

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

Python

1. Kami sangat menyarankan Anda menyiapkan virtualenv Python untuk mengembangkan dan mengemas pengujian sehingga dependensi yang tidak perlu tidak disertakan dalam paket aplikasi Anda.

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

> **Tip**
>
> - Jangan membuat virtualenv Python dengan `--system-site-packages` opsi, karena itu mewarisi paket dari direktori paket situs global Anda. Ini dapat mengakibatkan termasuk dependensi di lingkungan virtual Anda yang tidak diperlukan oleh pengujian Anda.

- Anda juga harus memverifikasi bahwa pengujian Anda tidak menggunakan dependensi yang bergantung pada pustaka asli, karena pustaka asli ini mungkin tidak ada pada instance tempat pengujian ini dijalankan.

2. Instal py.test di lingkungan virtual Anda.

```
$ pip install pytest
```

3. Instal klien Appium Python di lingkungan virtual Anda.

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

4. Kecuali Anda menentukan jalur yang berbeda dalam mode kustom, Device Farm mengharapkan pengujian Anda disimpan`tests/`. Anda dapat menggunakan `find` untuk menampilkan semua file di dalam folder:

```
$ find tests/
```

Konfirmasikan bahwa file-file ini berisi rangkaian pengujian yang ingin Anda jalankan di Device Farm

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

5. Jalankan perintah ini dari folder ruang kerja lingkungan virtual Anda untuk menampilkan daftar pengujian Anda tanpa menjalankannya.

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

Konfirmasikan bahwa output menunjukkan pengujian yang ingin Anda jalankan di Device Farm.

6. Bersihkan semua file yang di-cache di bawah folder tests/Anda:

```
$ 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. Jalankan perintah berikut di ruang kerja Anda untuk menghasilkan file requirements.txt:

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

Ruby

Untuk mengemas tes Appium Ruby Anda dan mengunggahnya ke Device Farm, Anda harus menginstal yang berikut ini di mesin lokal Anda:

- Manajer Versi Ruby (RVM)

  Gunakan alat baris perintah ini saat Anda mengembangkan dan mengemas pengujian Anda sehingga dependensi yang tidak perlu tidak disertakan dalam paket pengujian Anda.
- Ruby
- Bundler (Permata ini biasanya dipasang dengan Ruby.)

1. Instal kunci yang diperlukan, RVM, dan Ruby. Untuk petunjuk, lihat Menginstal RVM di situs web RVM.

   Setelah instalasi selesai, muat ulang terminal Anda dengan keluar dan kemudian masuk lagi.

   > **Note**
   >
   > RVM dimuat sebagai fungsi untuk shell bash saja.

2. Verifikasi bahwa rvm sudah terpasang dengan benar

```
command -v rvm
```

   Anda harus melihat `rvm` sebagai output.

3. Jika Anda ingin menginstal versi Ruby tertentu, seperti*2.5.3*, jalankan perintah berikut:

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

   Verifikasi bahwa Anda menggunakan versi Ruby yang diminta:

```
ruby -v
```

4. Konfigurasikan bundler untuk mengkompilasi paket untuk platform pengujian yang Anda inginkan:

```
bundle config specific_platform true
```

5. Perbarui file.lock Anda untuk menambahkan platform yang diperlukan untuk menjalankan pengujian.

   - Jika Anda mengompilasi pengujian untuk dijalankan di perangkat Android, jalankan perintah ini untuk mengonfigurasi Gemfile agar menggunakan dependensi untuk host pengujian Android:

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

   - Jika Anda mengompilasi pengujian untuk dijalankan di perangkat iOS, jalankan perintah ini untuk mengonfigurasi Gemfile agar menggunakan dependensi untuk host uji iOS:

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

6. bundlerPermata biasanya dipasang secara default. Jika tidak, instal:

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

Buat file paket uji zip

> **Warning**
>
> Di Device Farm, struktur folder file dalam paket pengujian zip Anda penting, dan beberapa alat arsip akan mengubah struktur file ZIP Anda secara implisit. Kami menyarankan Anda mengikuti utilitas baris perintah yang ditentukan di bawah ini daripada menggunakan utilitas arsip yang dibangun ke dalam pengelola file desktop lokal Anda (seperti Finder atau Windows Explorer).

Sekarang, bundel pengujian Anda untuk Device Farm.

Java (JUnit)

Bangun dan kemas pengujian Anda:

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

File `zip-with-dependencies.zip` akan dibuat sebagai hasilnya. Ini adalah paket tes Anda.

Java (TestNG)

Bangun dan kemas pengujian Anda:

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

File `zip-with-dependencies.zip` akan dibuat sebagai hasilnya. Ini adalah paket tes Anda.

Node.JS

1. Periksa proyek Anda.

   Pastikan Anda berada di direktori root proyek Anda. Anda dapat `package.json` melihat di direktori root.

2. Jalankan perintah ini untuk menginstal dependensi lokal Anda.

   ```
   npm install
   ```

   Perintah ini juga membuat `node_modules` folder di dalam direktori Anda saat ini.

   > **Note**
   >
   > Pada titik ini, Anda harus dapat menjalankan pengujian Anda secara lokal.

3. Jalankan perintah ini untuk mengemas file di folder Anda saat ini ke dalam file*.tgz. File diberi nama menggunakan `name` properti di `package.json` file Anda.

   ```
   npm-bundle
   ```

   File tarball (.tgz) ini berisi semua kode dan dependensi Anda.

4. Jalankan perintah ini untuk menggabungkan tarball (*.tgz file) yang dihasilkan pada langkah sebelumnya ke dalam satu arsip zip:

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

Ini adalah `MyTests.zip` file yang Anda unggah ke Device Farm dalam prosedur berikut.

Python

Python 2

Buat arsip paket Python yang diperlukan (disebut “wheelhouse”) menggunakan pip:

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

Package ruang kemudi, pengujian, dan persyaratan pip Anda ke dalam arsip zip untuk Device Farm:

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

Python 3

Package tes dan persyaratan pip Anda ke dalam file zip:

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

Ruby

1. Jalankan perintah ini untuk membuat lingkungan Ruby virtual:

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

2. Jalankan perintah ini untuk menggunakan lingkungan yang baru saja Anda buat:

   ```
   rvm gemset use myGemset
   ```

3. Periksa kode sumber Anda.

   Pastikan Anda berada di direktori root proyek Anda. Anda dapat `Gemfile` melihat di direktori root.

4. Jalankan perintah ini untuk menginstal dependensi lokal Anda dan semua permata dari: `Gemfile`

```
bundle install
```

> **Note**
>
> Pada titik ini, Anda harus dapat menjalankan pengujian Anda secara lokal. Gunakan perintah ini untuk menjalankan pengujian secara lokal:
>
> ```
> bundle exec $test_command
> ```

5. Package permata Anda di `vendor/cache` folder.

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

6. Jalankan perintah berikut untuk menggabungkan kode sumber Anda, bersama dengan semua dependensi Anda, ke dalam satu arsip zip:

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

Ini adalah `MyTests.zip` file yang Anda unggah ke Device Farm dalam prosedur berikut.

Unggah paket pengujian Anda ke Device Farm

Anda dapat menggunakan konsol Device Farm untuk mengunggah pengujian.

1. Masuk ke konsol Device Farm di https://console.aws.amazon.com/devicefarm.
2. Pada panel navigasi Device Farm, pilih Pengujian Perangkat Seluler, lalu pilih Proyek.
3. Jika Anda adalah pengguna baru, pilih Proyek baru, masukkan nama untuk proyek, lalu pilih Kirim.

   Jika Anda sudah memiliki proyek, Anda dapat memilihnya untuk mengunggah tes Anda ke sana.
4. Buka proyek Anda, lalu pilih Create run.
5. Di bawah pengaturan Jalankan, berikan nama yang sesuai pada pengujian Anda. Ini mungkin berisi kombinasi spasi atau tanda baca.

6. Untuk pengujian Android dan iOS asli

   Di bawah Pengaturan Jalankan, pilih aplikasi Android jika Anda menguji aplikasi Android (.apk), atau pilih aplikasi iOS jika Anda menguji aplikasi iOS (.ipa). Kemudian, di bawah Pilih aplikasi, pilih Unggah aplikasi sendiri untuk mengunggah paket yang dapat didistribusikan aplikasi Anda.

   > **Note**
   >
   > File harus berupa Android `.apk` atau iOS `.ipa`. Aplikasi iOS harus dibangun untuk perangkat nyata, bukan Simulator.

   Untuk pengujian aplikasi Web Seluler

   Di bawah Setelan Jalankan, pilih Aplikasi Web.

7. Di bawah Configure test, di bagian Select test framework, pilih framework Appium yang Anda uji, lalu Upload paket pengujian Anda sendiri.
8. Jelajahi dan pilih file.zip yang berisi pengujian Anda. File.zip harus mengikuti format yang dijelaskan dalamKonfigurasikan paket uji Appium Anda.
9. Ikuti petunjuk untuk memilih perangkat dan mulai menjalankan. Untuk informasi selengkapnya, lihat Membuat uji coba di Device Farm.

> **Note**
>
> Device Farm tidak mengubah pengujian Appium.

Ambil tangkapan layar pengujian Anda (Opsional)

Anda dapat mengambil tangkapan layar sebagai bagian dari pengujian Anda.

Device Farm menyetel `DEVICEFARM_SCREENSHOT_PATH` properti ke jalur yang sepenuhnya memenuhi syarat pada sistem file lokal tempat Device Farm mengharapkan tangkapan layar Appium disimpan. Direktori khusus uji tempat tangkapan layar disimpan ditentukan saat runtime. Tangkapan layar ditarik ke laporan Device Farm Anda secara otomatis. Untuk melihat tangkapan layar, di konsol Device Farm, pilih bagian Screenshots.

Untuk informasi selengkapnya tentang pengambilan tangkapan layar dalam pengujian Appium, lihat Mengambil Screenshot di dokumentasi Appium API.

Pengujian Android di AWS Device Farm

Device Farm menyediakan dukungan untuk beberapa jenis pengujian otomatisasi untuk perangkat Android, dan dua pengujian bawaan.

Untuk informasi selengkapnya tentang pengujian di Device Farm, lihatUji kerangka kerja dan pengujian bawaan di AWS Device Farm.

Kerangka kerja pengujian aplikasi Android

Tes berikut tersedia untuk perangkat Android.

- Tes Appium otomatis
- Instrumentasi

Jenis pengujian bawaan untuk Android

Ada satu jenis pengujian bawaan yang tersedia untuk perangkat Android:

- Bawaan: fuzz (Android dan iOS)

Instrumentasi untuk Android dan AWS Device Farm

Device Farm menyediakan dukungan untuk Instrumentasi (JUnit, Espresso, Robotium, atau pengujian berbasis Instrumentasi) untuk Android.

Device Farm juga menyediakan contoh aplikasi Android dan tautan ke pengujian yang berfungsi di tiga kerangka kerja otomatisasi Android, termasuk Instrumentation (Espresso). Aplikasi contoh Device Farm untuk Android tersedia untuk diunduh GitHub.

Untuk informasi selengkapnya tentang pengujian di Device Farm, lihatUji kerangka kerja dan pengujian bawaan di AWS Device Farm.

Topik

- Apa itu instrumentasi?
- Pertimbangan untuk pengujian instrumentasi Android

- Penguraian uji mode standar
- Mengintegrasikan Instrumentasi Android dengan Device Farm

Apa itu instrumentasi?

Instrumentasi Android memungkinkan Anda untuk memanggil metode callback dalam kode pengujian sehingga Anda dapat menjalankan siklus hidup komponen selangkah demi selangkah, seolah-olah Anda sedang men-debug komponen. Untuk informasi selengkapnya, lihat Pengujian instrumen di bagian Jenis dan lokasi pengujian pada dokumentasi Alat Developer Android.

Pertimbangan untuk pengujian instrumentasi Android

Saat menggunakan instrumentasi Android, pertimbangkan rekomendasi dan catatan berikut.

Periksa Kompatibilitas OS Android

Periksa dokumentasi Android, untuk memastikan Instrumentasi kompatibel dengan versi OS Android Anda.

Berjalan dari Command Line

Untuk menjalankan pengujian Instrumentasi dari baris perintah, ikuti dokumentasi Android.

Animasi Sistem

Sesuai dokumentasi Android untuk pengujian Espresso, disarankan agar animasi sistem dimatikan saat menguji pada perangkat nyata. Device Farm secara otomatis menonaktifkan pengaturan Skala Animasi Jendela, Skala Animasi Transisi, dan Skala Durasi Animator saat dijalankan dengan runner pengujian instrumentasi Runner JUnitandroid.support.test.Runner.Android.

Perekam Uji

Device Farm mendukung framework, seperti Robotium, yang memiliki alat record-and-playback scripting.

Penguraian uji mode standar

Dalam mode standar run, Device Farm mem-parsing rangkaian pengujian Anda dan mengidentifikasi kelas pengujian unik dan metode yang akan dijalankan. Ini dilakukan melalui alat yang disebut Dex Test Parser.

Saat diberi file .apk instrumentasi Android sebagai input, parser mengembalikan nama metode pengujian yang memenuhi syarat sepenuhnya yang cocok dengan konvensi JUnit 3 dan JUnit 4.

Untuk menguji ini di lingkungan lokal:

1. Unduh `dex-test-parser`biner.
2. Jalankan perintah berikut untuk mendapatkan daftar metode pengujian yang akan berjalan di Device Farm:

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

Mengintegrasikan Instrumentasi Android dengan Device Farm

> **Note**
>
> Gunakan petunjuk berikut untuk mengintegrasikan pengujian instrumentasi Android dengan AWS Device Farm. Untuk informasi selengkapnya tentang menggunakan pengujian instrumentasi di Device Farm, lihatInstrumentasi untuk Android dan AWS Device Farm.

Unggah pengujian instrumentasi Android Anda

Gunakan konsol Device Farm untuk mengunggah pengujian Anda.

1. Masuk ke konsol Device Farm di https://console.aws.amazon.com/devicefarm.
2. Pada panel navigasi Device Farm, pilih Pengujian Perangkat Seluler, lalu pilih Proyek.
3. Dalam daftar proyek, pilih proyek yang ingin Anda unggah pengujian.

   > **Tip**
   >
   > Anda dapat menggunakan bilah pencarian untuk memfilter daftar proyek berdasarkan nama.
   > Untuk membuat proyek, ikuti instruksi diMembuat proyek di AWS Device Farm.

4. Pilih Buat jalankan.
5. Di bawah Pilih aplikasi, di bagian Opsi pemilihan aplikasi, pilih Unggah aplikasi sendiri.
6. Jelajahi dan pilih file aplikasi Android Anda. File harus berupa file.apk.

7. Di bawah Configure test, di bagian Select test framework, pilih Instrumentation, lalu pilih Choose File.
8. Jelajahi dan pilih file.apk yang berisi pengujian Anda.
9. Lengkapi instruksi yang tersisa untuk memilih perangkat dan mulai menjalankan.

(Opsional) Ambil tangkapan layar dalam pengujian instrumentasi Android

Anda dapat mengambil tangkapan layar sebagai bagian dari pengujian Instrumentasi Android Anda.

Untuk mengambil tangkapan layar, hubungi salah satu metode berikut:

- Untuk Robotium, panggil `takeScreenShot` metode (misalnya,`solo.takeScreenShot();`).
- Untuk Spoon, panggil `screenshot` metode, misalnya:

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

Selama uji coba, Device Farm mendapatkan tangkapan layar dari lokasi berikut di perangkat, jika ada, lalu menambahkannya ke laporan pengujian:

- `/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`*

Pengujian iOS di AWS Device Farm

Device Farm menyediakan dukungan untuk beberapa jenis pengujian otomatisasi untuk perangkat iOS, dan pengujian bawaan.

Untuk informasi selengkapnya tentang pengujian di Device Farm, lihatUji kerangka kerja dan pengujian bawaan di AWS Device Farm.

Kerangka kerja pengujian aplikasi iOS

Tes berikut tersedia untuk perangkat iOS.

- Tes Appium otomatis
- XCTest
- XCTest UI

Jenis pengujian bawaan untuk iOS

Saat ini ada satu jenis pengujian bawaan yang tersedia untuk perangkat iOS.

- Bawaan: fuzz (Android dan iOS)

Mengintegrasikan Device Farm dengan XCTest iOS

Dengan Device Farm, Anda dapat menggunakan XCTest framework untuk menguji aplikasi di perangkat nyata. Untuk informasi selengkapnya XCTest, lihat Dasar-dasar Pengujian dalam Pengujian dengan Xcode.

Untuk menjalankan pengujian, Anda membuat paket untuk uji coba, dan Anda mengunggah paket ini ke Device Farm.

Untuk informasi selengkapnya tentang pengujian di Device Farm, lihatUji kerangka kerja dan pengujian bawaan di AWS Device Farm.

Topik

- Buat paket untuk Anda XCTest jalankan
- Unggah paket untuk Anda XCTest jalankan ke Device Farm

Buat paket untuk Anda XCTest jalankan

Untuk menguji aplikasi Anda dengan menggunakan XCTest framework, Device Farm memerlukan hal berikut:

- Paket aplikasi Anda sebagai `.ipa` file.
- XCTest Paket Anda sebagai `.zip` file.

Anda membuat paket-paket ini dengan menggunakan output build yang dihasilkan Xcode. Selesaikan langkah-langkah berikut untuk membuat paket sehingga Anda dapat mengunggahnya ke Device Farm.

Untuk menghasilkan output build untuk aplikasi Anda

1. Buka project aplikasi Anda di Xcode.
2. Di menu tarik-turun skema di toolbar Xcode, pilih Perangkat iOS Generik sebagai tujuan.
3. Di menu Produk, pilih Build For, lalu pilih Testing.

Untuk membuat paket aplikasi

1. Di navigator proyek di Xcode, di bawah Produk, buka menu kontekstual untuk file bernama. *app-project-name*`.app` Kemudian, pilih Tampilkan di Finder. Finder membuka folder bernama`Debug-iphoneos`, yang berisi output yang dihasilkan Xcode untuk build pengujian Anda. Folder ini termasuk `.app` file Anda.
2. Di Finder, buat folder baru, dan beri nama`Payload`.
3. Salin *app-project-name*`.app` file, dan tempel di `Payload` folder.
4. Buka menu kontekstual untuk `Payload` folder dan pilih Kompres “Payload”. Sebuah file bernama `Payload.zip` dibuat.
5. Ubah nama file dan ekstensi `Payload.zip` ke*app-project-name*`.ipa`.

   Pada langkah selanjutnya, Anda memberikan file ini ke Device Farm. Untuk membuat file lebih mudah ditemukan, Anda mungkin ingin memindahkannya ke lokasi lain, seperti desktop Anda.
6. Secara opsional, Anda dapat menghapus `Payload` folder dan `.app` file di dalamnya.

Untuk membuat XCTest paket

1. Di Finder, di `Debug-iphoneos` direktori, buka menu kontekstual untuk file tersebut. *app-project-name*`.app` Kemudian, pilih Show Package Contents.
2. Dalam isi paket, buka `Plugins` folder. Folder ini berisi file bernama*app-project-name*`.xctest`.
3. Buka menu kontekstual untuk file ini dan pilih Kompres "" ***app-project-name*`.xctest`**. Sebuah file bernama *app-project-name*`.xctest.zip` dibuat.

Pada langkah selanjutnya, Anda memberikan file ini ke Device Farm. Untuk membuat file lebih mudah ditemukan, Anda mungkin ingin memindahkannya ke lokasi lain, seperti desktop Anda.

Unggah paket untuk Anda XCTest jalankan ke Device Farm

Gunakan konsol Device Farm untuk mengunggah paket untuk pengujian Anda.

1. Masuk ke konsol Device Farm di https://console.aws.amazon.com/devicefarm.
2. Jika Anda belum memiliki proyek, buat satu. Untuk langkah-langkah membuat proyek, lihatMembuat proyek di AWS Device Farm.

   Jika tidak, pada panel navigasi Device Farm, pilih Pengujian Perangkat Seluler, lalu pilih Proyek.
3. Pilih proyek yang ingin Anda gunakan untuk menjalankan pengujian.
4. Pilih Buat jalankan.
5. Di bawah Run settings, di bagian Run type, pilih iOS app.
6. Di bawah Pilih aplikasi, di bagian Opsi pemilihan aplikasi, pilih Unggah aplikasi sendiri. Kemudian, pilih Pilih file di bawah Unggah aplikasi.
7. Jelajahi `.ipa` file untuk aplikasi Anda dan unggah.

   > **Note**
   >
   > `.ipa`Paket Anda harus dibangun untuk pengujian.

8. Di bawah Configure test, di bagian Select test framework, pilih XCTest. Kemudian, pilih Pilih file di bawah Unggah aplikasi.
9. Jelajahi `.zip` file yang berisi XCTest paket untuk aplikasi Anda dan unggah.
10. Selesaikan langkah-langkah yang tersisa dalam proses pembuatan proyek. Anda akan memilih perangkat yang ingin Anda uji dan menentukan status perangkat.
11. Pilih Buat jalankan. Device Farm menjalankan pengujian Anda dan menunjukkan hasilnya di konsol.

Mengintegrasikan XCTest UI untuk iOS dengan Device Farm

Device Farm menyediakan dukungan untuk framework pengujian XCTest UI. Secara khusus, Device Farm mendukung pengujian XCTest UI yang ditulis dalam Objective-C dan Swift.

Kerangka kerja XCTest UI memungkinkan pengujian UI dalam pengembangan iOS, dibangun di atasnya XCTest. Untuk informasi selengkapnya, lihat Pengujian Antarmuka Pengguna di Pustaka Pengembang iOS.

Untuk informasi umum tentang pengujian di Device Farm, lihatUji kerangka kerja dan pengujian bawaan di AWS Device Farm.

Gunakan petunjuk berikut untuk mengintegrasikan Device Farm dengan framework pengujian XCTest UI untuk iOS.

Topik

- Siapkan pengujian XCTest UI iOS Anda
- Opsi 1: Membuat XCTest paket.ipa UI
- Opsi 2: Membuat paket XCTest UI.zip
- Unggah pengujian XCTest UI iOS Anda

Siapkan pengujian XCTest UI iOS Anda

Anda dapat mengunggah `.ipa` file atau `.zip` file untuk paket pengujian XCTEST_UI Anda.

`.ipa`File adalah arsip aplikasi yang berisi aplikasi iOS Runner dalam format bundel. File tambahan tidak dapat disertakan di dalam *`.ipa`* file.

Jika Anda mengunggah `.zip` file, file tersebut dapat berisi aplikasi iOS Runner secara langsung atau `.ipa` file. Anda juga dapat menyertakan file lain dalam `.zip` file jika Anda ingin menggunakannya selama pengujian. Misalnya Anda dapat menyertakan file seperti`.xctestrun`, `.xcworkspace` atau `.xcodeproj` di dalam `.zip` file untuk menjalankan Rencana Uji XCUI di peternakan perangkat. Instruksi terperinci tentang cara menjalankan Rencana Uji tersedia di file spesifikasi pengujian default untuk jenis Uji XCUI.

Opsi 1: Membuat XCTest paket.ipa UI

Bundel yourAppNameUITest-Runner.app diproduksi oleh Xcode saat Anda membangun proyek untuk pengujian. Hal ini dapat ditemukan di direktori Produk untuk proyek Anda.

Untuk membuat file.ipa:

1. Buat direktori yang disebut*`Payload`*.

2. Tambahkan direktori aplikasi Anda ke direktori Payload.
3. Arsipkan direktori Payload ke dalam `.zip` file dan kemudian ubah ekstensi file menjadi `.ipa`.

Struktur folder berikut menunjukkan bagaimana contoh aplikasi bernama *`my-project-nameUITest-Runner.app`* akan dikemas sebagai `.ipa` file:

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

Opsi 2: Membuat paket XCTest UI.zip

Device Farm secara otomatis menghasilkan `.xctestrun` file untuk Anda untuk menjalankan rangkaian pengujian XCTest UI lengkap Anda. Jika Anda ingin menggunakan `.xctestrun` file Anda sendiri di Device Farm, Anda dapat mengompres `.xctestrun` file dan direktori aplikasi menjadi `.zip` file. Jika Anda sudah memiliki `.ipa` file untuk paket pengujian Anda, Anda dapat memasukkannya di sini alih-alih*`*-Runner.app`*.

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

Jika Anda ingin menjalankan rencana pengujian Xcode untuk pengujian XCUI di Device Farm, Anda dapat membuat zip yang berisi file my-project-nameUITest-Runner.app atau my-project-name UITest.ipa dan file kode sumber xcode yang diperlukan untuk menjalankan XCTEST_UI dengan rencana pengujian, termasuk file atau file. `.xcworkspace` `.xcodeproj`

Berikut adalah contoh zip menggunakan `.xcodeproj` file:

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

```
   ### (any directory)
   ### SampleXcodeProject.xcodeproj
        ### Testplan_1.xctestplan
        ### Testplan_2.xctestplan
        ### (any other source code files created by xcode with .xcodeproj)
```

Berikut adalah contoh zip menggunakan `.xcworkspace` file:

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

> **Note**
>
> Harap pastikan bahwa Anda tidak memiliki direktori bernama “Payload” di dalam paket XCTest UI.zip Anda.

Unggah pengujian XCTest UI iOS Anda

Gunakan konsol Device Farm untuk mengunggah pengujian Anda.

1. Masuk ke konsol Device Farm di https://console.aws.amazon.com/devicefarm.
2. Pada panel navigasi Device Farm, pilih Pengujian Perangkat Seluler, lalu pilih Proyek.
3. Dalam daftar proyek, pilih proyek yang ingin Anda unggah pengujian.

> **Tip**
>
> Anda dapat menggunakan bilah pencarian untuk memfilter daftar proyek berdasarkan nama.

Untuk membuat proyek, ikuti instruksi di Membuat proyek di AWS Device Farm

4. Pilih Buat jalankan.
5. Di bawah Run settings, di bagian Run type, pilih iOS app.
6. Di bawah Pilih aplikasi, di bagian Opsi pemilihan aplikasi, pilih Unggah aplikasi sendiri. Kemudian, pilih Pilih file di bawah Unggah aplikasi.
7. Jelajahi dan pilih file aplikasi iOS Anda. File harus berupa file.ipa.

> **Note**
>
> Pastikan file.ipa Anda dibuat untuk perangkat iOS dan bukan untuk simulator.

8. Di bawah Configure test, di bagian Select test framework, pilih XCTest UI. Kemudian, pilih Pilih file di bawah Unggah aplikasi.
9. Jelajahi dan pilih file.ipa atau.zip yang berisi runner uji XCTest UI iOS Anda.
10. Selesaikan langkah-langkah yang tersisa dalam proses pembuatan run. Anda akan memilih perangkat yang ingin Anda uji dan secara opsional menentukan konfigurasi tambahan.
11. Pilih Buat jalankan. Device Farm menjalankan pengujian Anda dan menunjukkan hasilnya di konsol.

Pengujian aplikasi web di AWS Device Farm

Device Farm menyediakan pengujian dengan Appium untuk aplikasi web. Untuk informasi selengkapnya tentang menyiapkan pengujian Appium di Device Farm, lihat. the section called “Tes Appium otomatis”

Untuk informasi selengkapnya tentang pengujian di Device Farm, lihatUji kerangka kerja dan pengujian bawaan di AWS Device Farm.

Aturan untuk perangkat terukur dan tidak terukur

Pengukuran mengacu pada penagihan untuk perangkat. Secara default, perangkat Device Farm diukur dan Anda dikenakan biaya per menit setelah menit uji coba gratis habis. Anda juga dapat memilih untuk membeli perangkat yang tidak diukur, yang memungkinkan pengujian tanpa batas dengan biaya bulanan tetap. Untuk informasi selengkapnya tentang harga, lihat Harga AWS Device Farm.

Jika Anda memilih untuk memulai proses dengan kumpulan perangkat yang berisi perangkat iOS dan Android, ada aturan untuk perangkat terukur dan tidak terukur. Misalnya, jika Anda memiliki lima perangkat Android yang tidak diukur dan lima perangkat iOS yang tidak diukur, pengujian web Anda berjalan menggunakan perangkat yang tidak diukur.

Berikut adalah contoh lain: Misalkan Anda memiliki lima perangkat Android yang tidak diukur dan 0 perangkat iOS yang tidak diukur. Jika Anda hanya memilih perangkat Android untuk menjalankan web, perangkat yang tidak diukur akan digunakan. Jika Anda memilih perangkat Android dan iOS untuk menjalankan web, metode penagihan diukur, dan perangkat yang tidak diukur tidak digunakan.

Pengujian bawaan di AWS Device Farm

Device Farm menyediakan dukungan untuk jenis pengujian bawaan untuk perangkat Android dan iOS.

Dengan pengujian bawaan, Anda dapat menguji aplikasi di beberapa perangkat tanpa harus menulis dan memelihara skrip otomatisasi pengujian. Ini dapat menghemat waktu dan tenaga Anda, terutama ketika Anda memulai dengan Device Farm. Device Farm menawarkan jenis pengujian bawaan berikut:

- Bawaan: fuzz (Android dan iOS)— Tes fuzz secara acak mengirimkan peristiwa antarmuka pengguna ke perangkat dan kemudian melaporkan hasilnya.

Untuk informasi selengkapnya tentang kerangka pengujian dan pengujian di Device Farm, lihatUji kerangka kerja dan pengujian bawaan di AWS Device Farm.

Menjalankan uji fuzz bawaan Device Farm (Android dan iOS)

Tes fuzz bawaan Device Farm secara acak mengirimkan peristiwa antarmuka pengguna ke perangkat dan kemudian melaporkan hasilnya.

Untuk informasi selengkapnya tentang pengujian di Device Farm, lihatUji kerangka kerja dan pengujian bawaan di AWS Device Farm.

Untuk menjalankan uji fuzz bawaan

1. Masuk ke konsol Device Farm di https://console.aws.amazon.com/devicefarm.
2. Pada panel navigasi Device Farm, pilih Pengujian Perangkat Seluler, lalu pilih Proyek.

3. Dalam daftar proyek, pilih proyek tempat Anda ingin menjalankan uji fuzz bawaan.

> **Tip**
>
> Anda dapat menggunakan bilah pencarian untuk memfilter daftar proyek berdasarkan nama.
> Untuk membuat proyek, ikuti instruksi di[Membuat proyek di AWS Device Farm](#).

4. Pilih Buat jalankan.
5. Di bawah Run settings, pilih tipe run Anda di bagian Run type. Pilih aplikasi Android jika Anda tidak memiliki aplikasi yang siap untuk diuji, atau jika Anda sedang menguji aplikasi android (.apk). Pilih aplikasi iOS jika Anda menguji aplikasi iOS (.ipa).
6. Di bawah Pilih aplikasi, pilih Pilih contoh aplikasi yang disediakan oleh Device Farm jika Anda tidak memiliki aplikasi yang tersedia untuk pengujian. Jika Anda membawa aplikasi sendiri, pilih Unggah aplikasi sendiri, dan pilih file aplikasi Anda.
7. Di bawah Configure test, di bagian Select test framework, pilih Built-in: Fuzz.
8. Jika salah satu pengaturan berikut muncul, Anda dapat menerima nilai default atau menentukan sendiri:

   - Jumlah peristiwa: Tentukan angka antara 1 dan 10.000, yang mewakili jumlah peristiwa antarmuka pengguna untuk pengujian fuzz yang akan dilakukan.
   - Event throttle: Tentukan angka antara 0 dan 1.000, mewakili jumlah milidetik untuk pengujian fuzz menunggu sebelum melakukan acara antarmuka pengguna berikutnya.
   - Biji pengacak: Tentukan nomor untuk uji fuzz yang akan digunakan untuk mengacak peristiwa antarmuka pengguna. Menentukan nomor yang sama untuk tes fuzz berikutnya memastikan urutan peristiwa yang identik.

9. Lengkapi instruksi yang tersisa untuk memilih perangkat dan mulai menjalankan.

Lingkungan pengujian khusus di AWS Device Farm

AWS Device Farm memungkinkan mengonfigurasi lingkungan khusus untuk pengujian otomatis (mode kustom), yang merupakan pendekatan yang disarankan untuk semua pengguna Device Farm. Untuk mempelajari lebih lanjut tentang lingkungan di Device Farm, lihat Lingkungan pengujian.

Manfaat Mode Kustom sebagai lawan dari Mode Standar meliputi:

- Eksekusi end-to-end pengujian lebih cepat: Paket pengujian tidak diuraikan untuk mendeteksi setiap pengujian di suite, menghindari preprocessing/postprocessing overhead.
- Log langsung dan streaming video: Log pengujian sisi klien dan video Anda disiarkan langsung saat menggunakan Mode Kustom. Fitur ini tidak tersedia dalam mode standar.
- Menangkap semua artefak: Pada host dan perangkat, Mode Kustom memungkinkan Anda untuk menangkap semua artefak pengujian. Ini mungkin tidak dimungkinkan dalam mode standar.
- Lingkungan lokal yang lebih konsisten dan dapat direplikasi: Ketika dalam Mode Standar, artefak akan disediakan untuk setiap pengujian individu secara terpisah, yang dapat bermanfaat dalam keadaan tertentu. Namun, lingkungan pengujian lokal Anda mungkin menyimpang dari konfigurasi asli karena Device Farm menangani setiap pengujian yang dijalankan secara berbeda.

  Sebaliknya, Mode Kustom memungkinkan Anda membuat lingkungan eksekusi pengujian Device Farm secara konsisten sesuai dengan lingkungan pengujian lokal Anda.

Lingkungan khusus dikonfigurasi menggunakan file spesifikasi pengujian (spesifikasi pengujian) yang diformat YAML. Device Farm menyediakan file spesifikasi pengujian default untuk setiap jenis pengujian yang didukung yang dapat digunakan sebagaimana adanya atau disesuaikan; kustomisasi seperti filter pengujian atau file konfigurasi dapat ditambahkan ke spesifikasi pengujian. Spesifikasi pengujian yang diedit dapat disimpan untuk uji coba di masa mendatang.

Untuk informasi selengkapnya, lihat Mengunggah Spesifikasi Uji Kustom Menggunakan dan. AWS CLI Membuat uji coba di Device Farm

Topik

- Uji referensi spesifikasi dan sintaks
- Host untuk lingkungan pengujian khusus
- Akses sumber daya AWS menggunakan Peran Eksekusi IAM
- Variabel lingkungan untuk lingkungan pengujian khusus

- Praktik terbaik untuk eksekusi lingkungan pengujian kustom
- Memigrasi pengujian dari lingkungan pengujian standar ke kustom
- Memperluas lingkungan pengujian khusus di Device Farm

Uji referensi spesifikasi dan sintaks

Spesifikasi pengujian (spesifikasi pengujian) adalah file yang Anda gunakan untuk menentukan lingkungan pengujian kustom di Device Farm.

Alur kerja spesifikasi uji

Spesifikasi pengujian Device Farm mengeksekusi fase dan perintahnya dalam urutan yang telah ditentukan sebelumnya, memungkinkan Anda menyesuaikan cara lingkungan Anda disiapkan dan dijalankan. Ketika setiap fase dijalankan, perintahnya dijalankan dalam urutan yang tercantum dalam file spesifikasi pengujian. Fase dieksekusi dalam urutan berikut

1. `install`- Di sinilah tindakan seperti mengunduh, menginstal, dan menyiapkan perkakas harus ditentukan.
2. `pre_test`- Di sinilah tindakan pra-tes seperti memulai proses latar belakang harus didefinisikan.
3. `test`- Di sinilah perintah yang memanggil pengujian Anda harus didefinisikan.
4. `post_test`- Di sinilah tugas akhir apa pun yang perlu dijalankan setelah pengujian Anda selesai harus ditentukan, seperti pembuatan laporan pengujian dan agregasi file artefak.

Sintaks spesifikasi uji

Berikut ini adalah skema YAMAL untuk file spesifikasi pengujian

```
version: 0.1

android_test_host: "string"
ios_test_host: "string"

phases:
  install:
    commands:
      - "string"
      - "string"
```

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

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

`version`

(Diperlukan, nomor)

Mencerminkan versi spesifikasi pengujian yang didukung Device Farm. Nomor versi saat ini adalah `0.1`.

`android_test_host`

(Opsional, string)

Host pengujian yang akan dipilih untuk uji coba dilakukan di perangkat Android. Bidang ini diperlukan untuk pengujian yang dijalankan di perangkat Android. Untuk informasi selengkapnya, lihat Host uji yang tersedia untuk lingkungan pengujian khusus.

`ios_test_host`

(Opsional, string)

Host uji yang akan dipilih untuk uji coba dilakukan pada perangkat iOS. Bidang ini diperlukan untuk uji coba pada perangkat iOS dengan versi utama lebih besar dari 26. Untuk informasi selengkapnya, lihat Host uji yang tersedia untuk lingkungan pengujian khusus.

`phases`

Bagian ini berisi kelompok perintah yang dijalankan selama uji coba, di mana setiap fase adalah opsional Nama fase uji yang diizinkan adalah:`install`,`pre_test`,`test`, dan`post_test`.

- `install`- Dependensi default untuk kerangka pengujian yang didukung oleh Device Farm sudah diinstal. Fase ini berisi perintah tambahan, jika ada, bahwa Device Farm berjalan selama instalasi.
- `pre_test`- Perintah, jika ada, dijalankan sebelum pengujian otomatis Anda.
- `test`- Perintah dijalankan selama uji coba otomatis Anda dijalankan. Jika ada perintah dalam fase pengujian gagal (artinya mengembalikan kode keluar bukan nol), pengujian ditandai sebagai gagal
- `post_test`- Perintah, jika ada, dijalankan setelah uji coba otomatis Anda dijalankan. Ini akan dieksekusi apakah tes Anda dalam `test` fase berhasil atau gagal.

`commands`

(Opsional, Daftar [string])

Daftar string untuk dieksekusi sebagai perintah shell selama fase.

`artifacts`

(Opsional, Daftar [string])

Device Farm mengumpulkan artefak seperti laporan kustom, file log, dan gambar dari lokasi yang ditentukan di sini. Karakter wildcard tidak didukung sebagai bagian dari lokasi artefak, jadi Anda harus menentukan jalur yang valid untuk setiap lokasi.

Artefak pengujian ini tersedia untuk setiap perangkat dalam uji coba Anda. Untuk informasi tentang mengambil artefak pengujian Anda, lihat. Mengunduh artefak di lingkungan pengujian khusus

> **Important**
>
> Spesifikasi pengujian harus diformat sebagai file YAMM yang valid. Jika indentasi atau spasi dalam spesifikasi pengujian Anda tidak valid, uji coba Anda bisa gagal. Tab tidak diizinkan dalam file YAMM. Anda dapat menggunakan validator YAMM untuk menguji apakah spesifikasi pengujian Anda adalah file YAMM yang valid. Untuk informasi selengkapnya, lihat situs web YAMM.

Contoh spesifikasi uji

Contoh berikut menunjukkan spesifikasi pengujian yang dapat dijalankan di Device Farm.

Simple Demo

Berikut ini adalah contoh file spesifikasi pengujian yang hanya log `Hello world!` sebagai artefak uji coba.

```
version: 0.1

android_test_host: amazon_linux_2
ios_test_host: macos_sequoia

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

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

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

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

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

Appium Android

Berikut ini adalah contoh file spesifikasi pengujian yang mengonfigurasi pengujian Appium Java TestNG yang dijalankan di Android..

```
version: 0.1

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

phases:

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

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

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

      # When running Android tests with Appium version 2, the uiautomator2 driver is
 preinstalled using driver
      # version 2.44.1 for Appium 2.5.1  If you want to install a different version
 of the driver,
      # you can use the Appium extension CLI to uninstall the existing uiautomator2
 driver
```

```
      # and install your desired version:
      # - |-
      #   if [ $DEVICEFARM_DEVICE_PLATFORM_NAME = "Android" ];
      #   then
      #     appium driver uninstall uiautomator2;
      #     appium driver install uiautomator2@2.34.0;
      #   fi;

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

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

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

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

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

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

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

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

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

     # - echo "Unzipping the tests JAR file"
     # - unzip *-tests.jar
     # - |-
```

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

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

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

Appium iOS

Berikut ini adalah contoh file spesifikasi pengujian yang mengkonfigurasi uji Appium Java TestNG yang dijalankan di iOS.

```
version: 0.1

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

phases:

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

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

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

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

      # When running iOS tests with Appium version 2, the XCUITest driver is
 preinstalled using driver
      # version 9.10.5 for Appium 2.5.4. If you want to install a different version
 of the driver,
      # you can use the Appium extension CLI to uninstall the existing XCUITest
 driver
      # and install your desired version:
      # - |-
      #   if [ $DEVICEFARM_DEVICE_PLATFORM_NAME = "iOS" ];
      #   then
      #     appium driver uninstall xcuitest;
      #     appium driver install xcuitest@10.0.0;
      #   fi;

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

```
      - export APPIUM_BASE_PATH=

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

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

     # Device Farm provides multiple pre-built versions of WebDriverAgent (WDA), a
required
     # Appium dependency for iOS, where each version corresponds to the XCUITest
driver version selected.
     # If Device Farm cannot find a corresponding version of WDA for your XCUITest
driver,
     # the latest available version is selected by default.
     - |-
       APPIUM_DRIVER_VERSION=$(appium driver list --installed --json | jq -r
".xcuitest.version" | cut -d "." -f 1);
       CORRESPONDING_APPIUM_WDA=$(env | grep
"DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V${APPIUM_DRIVER_VERSION}")
       if [[ ! -z "$APPIUM_DRIVER_VERSION" ]] && [[ ! -z
"$CORRESPONDING_APPIUM_WDA" ]]; then
         echo "Using Device Farm's prebuilt WDA version ${APPIUM_DRIVER_VERSION}.x,
which corresponds with your driver";
         DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH=$(echo $CORRESPONDING_APPIUM_WDA |
cut -d "=" -f2)
       else
         LATEST_SUPPORTED_WDA_VERSION=$(env | grep
"DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V" | sort -V -r | head -n 1)
         echo "Unknown driver version $APPIUM_DRIVER_VERSION; falling back to the
Device Farm default version of $LATEST_SUPPORTED_WDA_VERSION";
         DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH=$(echo
$LATEST_SUPPORTED_WDA_VERSION | cut -d "=" -f2)
       fi;

     # For iOS versions 16 and below only, the device unique identifier (UDID)
needs to modified for Appium tests
     # on Device Farm to remove the hypens.
```

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

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

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

```
          echo "Waiting for Appium to start on port 4723...";
          sleep 1;
        done;

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

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

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

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

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

# Artifacts are a list of paths on the filesystem where you can store test output
 and reports.
# All files in these paths will be collected by Device Farm, with certain limits
 (see limit details
```

```
# here: https://docs.aws.amazon.com/devicefarm/latest/developerguide/
limits.html#file-limits).
# These files will be available through the ListArtifacts API as your "Customer
 Artifacts".
artifacts:
  # By default, Device Farm will collect your artifacts from the $DEVICEFARM_LOG_DIR
 directory.
  - $DEVICEFARM_LOG_DIR
```

Appium (Both Platforms)

Berikut ini adalah contoh file spesifikasi pengujian yang mengonfigurasi pengujian Appium Java TestNG yang dijalankan di Android dan iOS.

```
version: 0.1

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

phases:

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

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

```
      - appium --version

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

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

      # When running iOS tests with Appium version 2, the XCUITest driver is
preinstalled using driver
      # version 9.10.5 for Appium 2.5.4. If you want to install a different version
of the driver,
      # you can use the Appium extension CLI to uninstall the existing XCUITest
driver
      # and install your desired version:
      # - |-
      #   if [ $DEVICEFARM_DEVICE_PLATFORM_NAME = "iOS" ];
      #   then
      #     appium driver uninstall xcuitest;
      #     appium driver install xcuitest@10.0.0;
      #   fi;

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

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

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

     # Device Farm provides multiple pre-built versions of WebDriverAgent (WDA), a
required
     # Appium dependency for iOS, where each version corresponds to the XCUITest
driver version selected.
     # If Device Farm cannot find a corresponding version of WDA for your XCUITest
driver,
     # the latest available version is selected by default.
     - |-
       if [ $DEVICEFARM_DEVICE_PLATFORM_NAME = "iOS" ]; then
         APPIUM_DRIVER_VERSION=$(appium driver list --installed --json | jq -r
".xcuitest.version" | cut -d "." -f 1);
         CORRESPONDING_APPIUM_WDA=$(env | grep
"DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V${APPIUM_DRIVER_VERSION}")
         if [[ ! -z "$APPIUM_DRIVER_VERSION" ]] && [[ ! -z
"$CORRESPONDING_APPIUM_WDA" ]]; then
           echo "Using Device Farm's prebuilt WDA version
${APPIUM_DRIVER_VERSION}.x, which corresponds with your driver";
           DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH=$(echo $CORRESPONDING_APPIUM_WDA
| cut -d "=" -f2)
         else
           LATEST_SUPPORTED_WDA_VERSION=$(env | grep
"DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V" | sort -V -r | head -n 1)
           echo "Unknown driver version $APPIUM_DRIVER_VERSION; falling back to the
Device Farm default version of $LATEST_SUPPORTED_WDA_VERSION";
           DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH=$(echo
$LATEST_SUPPORTED_WDA_VERSION | cut -d "=" -f2)
         fi;
       fi;

     # For iOS versions 16 and below only, the device unique identifier (UDID)
needs to modified for Appium tests
     # on Device Farm to remove the hypens.
```

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

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

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

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

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

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

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

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

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

Host untuk lingkungan pengujian khusus

Device Farm mendukung satu set sistem operasi dengan perangkat lunak pra-konfigurasi melalui penggunaan lingkungan host uji. Selama eksekusi pengujian, Device Farm menggunakan instance (host) yang dikelola Amazon yang terhubung secara dinamis ke perangkat yang dipilih yang sedang diuji. Instance ini sepenuhnya dibersihkan dan tidak digunakan kembali di antara proses, dan diakhiri dengan artefak yang dihasilkan setelah uji coba selesai.

Topik

- Host uji yang tersedia untuk lingkungan pengujian khusus
- Memilih host uji untuk lingkungan pengujian khusus
- Perangkat lunak yang didukung dalam lingkungan pengujian khusus
- Lingkungan uji untuk perangkat Android
- Lingkungan uji untuk perangkat iOS

Host uji yang tersedia untuk lingkungan pengujian khusus

Host uji sepenuhnya dikelola oleh Device Farm. Tabel berikut mencantumkan host uji Device Farm yang saat ini tersedia dan didukung untuk lingkungan pengujian kustom.

Platform Perangkat	Tuan Rumah Uji	Sistem Operasi	Arsitektur	Perangkat yang Didukung
Android	amazon_linux_2	Amazon Linux 2	x86_64	Android6 dan di atas
iOS	macos_sequoia	macOS Sequoia(versi 15)	arm64	iOS15 hingga 26

> **Note**
>
> Secara berkala, Device Farm menambahkan host uji baru untuk platform perangkat untuk mendukung versi OS perangkat yang lebih baru dan dependensinya. Ketika ini terjadi, host uji yang lebih lama untuk platform perangkat masing-masing tunduk pada akhir dukungan.

Versi Sistem Operasi

Setiap host uji yang tersedia menggunakan versi tertentu dari sistem operasi yang didukung di Device Farm pada saat itu. Meskipun kami mencoba menggunakan versi OS terbaru, ini mungkin bukan versi terbaru yang didistribusikan secara publik yang tersedia. Device Farm akan memperbarui sistem operasi secara berkala dengan pembaruan versi minor dan patch keamanan.

Untuk mengetahui versi tertentu (termasuk versi minor) dari sistem operasi yang digunakan selama uji coba, Anda dapat menambahkan cuplikan kode berikut ke fase file spesifikasi pengujian Anda.

Example

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

Memilih host uji untuk lingkungan pengujian khusus

Anda dapat menentukan host pengujian Android dan iOS dalam variabel yang sesuai `android_test_host` dan `ios_test_host` variabel file spesifikasi pengujian Anda.

Jika Anda tidak menentukan pilihan host pengujian untuk platform perangkat yang diberikan, pengujian akan dijalankan pada host pengujian yang telah ditetapkan Device Farm sebagai default untuk perangkat dan konfigurasi pengujian yang ditentukan.

> **Important**
>
> Saat menguji di iOS 18 dan di bawahnya, host uji lama akan digunakan saat host tidak dipilih. Untuk informasi lebih lanjut, lihat topik diHost uji iOS lama.

Sebagai contoh, tinjau cuplikan kode berikut:

Example

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

phases:
  # ...
```

Perangkat lunak yang didukung dalam lingkungan pengujian khusus

Device Farm menggunakan mesin host yang sudah diinstal sebelumnya dengan banyak pustaka perangkat lunak yang diperlukan untuk menjalankan kerangka kerja pengujian yang didukung pada layanan kami, menyediakan lingkungan pengujian siap saat diluncurkan. Device Farm mendukung berbagai bahasa melalui penggunaan mekanisme pemilihan perangkat lunak kami, dan akan memperbarui versi bahasa yang disertakan dalam lingkungan secara berkala.

Untuk perangkat lunak lain yang diperlukan, Anda dapat memodifikasi file spesifikasi pengujian untuk diinstal dari paket pengujian Anda, mengunduh dari internet, atau mengakses sumber pribadi dalam VPC Anda (lihat VPC ENI untuk informasi lebih lanjut). Untuk informasi selengkapnya, lihat Contoh spesifikasi uji.

Perangkat lunak pra-konfigurasi

Untuk memfasilitasi pengujian perangkat pada setiap platform, perkakas berikut disediakan pada host uji:

Alat	Platform Perangkat
Android SDK Build-Tools	Android
Android SDK Platform-Tools(termasuk`adb`)	Android
Xcode	iOS

Perangkat lunak yang dapat dipilih

Selain perangkat lunak pra-konfigurasi pada host, Device Farm menawarkan cara untuk memilih versi tertentu dari perangkat lunak yang didukung melalui `devicefarm-cli` perkakas.

Tabel berikut berisi perangkat lunak yang dapat dipilih dan host uji yang berisi mereka.

Perangkat Lunak/Alat	Host yang mendukung perangkat lunak ini	Perintah untuk digunakan dalam spesifikasi pengujian Anda
Java 17	amazon_linux_2	`devicefarm-cli use java 17`

Perangkat Lunak/Alat	Host yang mendukung perangkat lunak ini	Perintah untuk digunakan dalam spesifikasi pengujian Anda
	macos_sequoia	
Java 11	amazon_linux_2 macos_sequoia	`devicefarm-cli use java 11`
Java 8	amazon_linux_2 macos_sequoia	`devicefarm-cli use java 8`
Node.js 20	amazon_linux_2 macos_sequoia	`devicefarm-cli use node 20`
Node.js 18	amazon_linux_2 macos_sequoia	`devicefarm-cli use node 18`
Node.js 16	amazon_linux_2	`devicefarm-cli use node 16`
Python 3.11	amazon_linux_2 macos_sequoia	`devicefarm-cli use python 3.11`
Python 3.10	amazon_linux_2 macos_sequoia	`devicefarm-cli use python 3.10`
Python 3.9	amazon_linux_2 macos_sequoia	`devicefarm-cli use python 3.9`
Python 3.8	amazon_linux_2	`devicefarm-cli use python 3.8`

Perangkat Lunak/Alat	Host yang mendukung perangkat lunak ini	Perintah untuk digunakan dalam spesifikasi pengujian Anda
Ruby 3.2	amazon_linux_2 macos_sequoia	`devicefarm-cli use ruby 3.2`
Ruby 2.7	amazon_linux_2	`devicefarm-cli use ruby 2.7`
Appium 3	amazon_linux_2	`devicefarm-cli use appium 3`
Appium 2	amazon_linux_2 macos_sequoia	`devicefarm-cli use appium 2`
Appium 1	amazon_linux_2	`devicefarm-cli use appium 1`
Xcode 26	macos_sequoia	`devicefarm-cli use xcode 26`
Xcode 16	macos_sequoia	`devicefarm-cli use xcode 16`

Host pengujian juga mencakup alat pendukung yang umum digunakan untuk setiap versi perangkat lunak, seperti manajer `npm` paket `pip` dan (disertakan dengan Python dan Node.js masing-masing) dan dependensi (seperti UIAutomator2 Driver Appium) untuk alat seperti Appium. Ini memastikan Anda memiliki alat yang diperlukan untuk bekerja dengan kerangka kerja pengujian yang didukung.

Menggunakan alat devicefarm-cli di lingkungan pengujian khusus

Host uji menggunakan alat manajemen versi standar yang dipanggil `devicefarm-cli` untuk memilih versi perangkat lunak. Alat ini terpisah dari AWS CLI dan hanya tersedia di host uji Device Farm. Dengan`devicefarm-cli`, Anda dapat beralih ke versi perangkat lunak yang sudah diinstal sebelumnya pada host uji. Ini memberikan cara mudah untuk memelihara file spesifikasi pengujian

Device Farm Anda dari waktu ke waktu dan memberi Anda mekanisme yang dapat diprediksi untuk meningkatkan versi perangkat lunak di masa mendatang.

> **Important**
>
> Alat baris perintah ini tidak tersedia di host iOS lama. Untuk informasi lebih lanjut, lihat topik diHost uji iOS lama.

Cuplikan di bawah ini menunjukkan `help` halaman: `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.
```

Mari kita tinjau beberapa contoh menggunakan`devicefarm-cli`. Untuk menggunakan alat untuk mengubah versi Python dari *`3.10`* ke *`3.9`* dalam file spesifikasi pengujian Anda, jalankan perintah berikut:

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

Untuk mengubah versi Appium dari *`1`* menjadi: *`2`*

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

> **Tip**
>
> Perhatikan bahwa ketika Anda memilih versi perangkat lunak, `devicefarm-cli` juga beralih alat pendukung untuk bahasa-bahasa tersebut, seperti `pip` untuk Python dan `npm` NodeJS.

Untuk informasi selengkapnya tentang perangkat lunak prainstal pada host uji, lihatPerangkat lunak yang didukung dalam lingkungan pengujian khusus.

Lingkungan uji untuk perangkat Android

AWS Device Farm menggunakan mesin host Amazon Elastic Compute Cloud (EC2) Amazon Elastic Compute Cloud (EC2) yang menjalankan Amazon Linux 2 untuk menjalankan pengujian Android. Saat Anda menjadwalkan uji coba, Device Farm mengalokasikan host khusus untuk setiap perangkat untuk menjalankan pengujian secara independen. Mesin host berakhir setelah pengujian dijalankan bersama dengan artefak yang dihasilkan.

Host Amazon Linux 2 memberikan beberapa keuntungan:

- Pengujian yang lebih cepat dan lebih andal: Dibandingkan dengan host lama, host uji baru secara signifikan meningkatkan kecepatan pengujian, terutama mengurangi waktu mulai pengujian. Host Amazon Linux 2 juga menunjukkan stabilitas dan keandalan yang lebih besar selama pengujian.
- Akses Jarak Jauh yang Ditingkatkan untuk pengujian manual: Peningkatan ke host pengujian terbaru dan peningkatan menghasilkan latensi yang lebih rendah dan kinerja video yang lebih baik untuk pengujian manual Android.
- Pemilihan versi perangkat lunak standar: Device Farm sekarang menstandarisasi dukungan bahasa pemrograman utama pada host uji serta versi kerangka Appium. Untuk bahasa yang didukung (saat ini Java, Python, Node.js, dan Ruby) dan Appium, host uji baru menyediakan rilis stabil jangka panjang segera setelah peluncuran. Manajemen versi terpusat melalui `devicefarm-cli` alat ini memungkinkan pengembangan file spesifikasi pengujian dengan pengalaman yang konsisten di seluruh kerangka kerja.

Topik

- Rentang IP yang didukung untuk lingkungan pengujian Amazon Linux 2 di Device Farm

Rentang IP yang didukung untuk lingkungan pengujian Amazon Linux 2 di Device Farm

Pelanggan sering perlu mengetahui rentang IP dari mana lalu lintas Device Farm berasal, terutama untuk mengonfigurasi firewall dan pengaturan keamanan mereka. Untuk host uji Amazon EC2, rentang IP mencakup seluruh wilayah. `us-west-2` Untuk host uji Amazon Linux 2, yang merupakan opsi default untuk menjalankan Android baru, rentangnya telah dibatasi. Lalu lintas sekarang berasal dari satu set gateway NAT tertentu, membatasi rentang IP ke alamat berikut:

Rentang IP
44.236.137.143
52.13.151.244
52.35.189.191
54.201.250.26

Untuk informasi selengkapnya tentang lingkungan pengujian Android di Device Farm, lihatLingkungan uji untuk perangkat Android.

Lingkungan uji untuk perangkat iOS

Device Farm menggunakan instance (host) macOS yang dikelola Amazon yang terhubung secara dinamis ke perangkat iOS selama pengujian dijalankan. Setiap host sudah dikonfigurasi sebelumnya dengan perangkat lunak yang memungkinkan pengujian perangkat pada berbagai platform pengujian populer, seperti XCTest UI dan Appium.

Iterasi host uji iOS saat ini telah meningkatkan pengalaman pengujian jika dibandingkan dengan versi sebelumnya, termasuk:

- Pengalaman OS dan perkakas host yang konsisten untuk iOS 15 hingga iOS 26 Sebelumnya, host pengujian ditentukan oleh perangkat yang digunakan, yang mengarah ke lingkungan perangkat lunak yang terfragmentasi saat dijalankan di beberapa versi iOS. Pengalaman saat ini memungkinkan pemilihan host sederhana untuk memungkinkan lingkungan yang konsisten di seluruh perangkat. Ini akan memungkinkan versi dan perkakas macOS yang sama (seperti Xcode) tersedia di setiap perangkat iOS.

- Peningkatan kinerja untuk pengujian iOS 15 dan 16 Menggunakan infrastruktur yang diperbarui, waktu penyiapan telah meningkat secara substansif untuk pengujian iOS 15 dan 16.
- Versi perangkat lunak yang dapat dipilih standar untuk dependensi yang didukung Kami sekarang memiliki sistem pemilihan `devicefarm-cli` perangkat lunak di host uji iOS dan Android, memungkinkan Anda memilih versi pilihan dari dependensi kami yang didukung. Untuk dependensi yang didukung (seperti Java, Python, Node.js, Ruby, dan Appium), versi akan dipilih melalui spesifikasi pengujian. Untuk gambaran tentang cara kerja fitur ini, silakan lihat topik diPerangkat lunak yang didukung dalam lingkungan pengujian khusus.

> **Important**
>
> Jika dijalankan di iOS 18 dan di bawahnya, pengujian Anda akan dijalankan pada host uji lama secara default. Lihat topik di bawah tentang cara bermigrasi dari host lama.

Host uji iOS lama

Untuk pengujian yang ada di iOS 18 dan di bawahnya, host uji lama dipilih secara default untuk lingkungan pengujian khusus. Tabel berikut berisi versi host uji yang dijalankan dengan versi perangkat iOS.

Sistem Operasi	Arsitektur	Default untuk Perangkat
macOS Sonoma(versi 14)	arm64	iOS 18
macOS Ventura(versi 13)	arm64	iOS 17
macOS Monterey(versi 12)	x86_64	iOS 16dan di bawah

Untuk memilih host uji yang lebih baru, lihat topik terkaitMemigrasi lingkungan pengujian kustom Anda ke host uji iOS baru.

Perangkat lunak yang didukung untuk perangkat iOS

Untuk mendukung pengujian perangkat iOS, host uji Device Farm untuk perangkat iOS telah dikonfigurasi sebelumnya dengan Xcode dan perkakas baris perintah terkait. Untuk perangkat lunak

lain yang tersedia, silakan tinjau topik terkaitPerangkat lunak yang didukung dalam lingkungan pengujian khusus.

Memigrasi lingkungan pengujian kustom Anda ke host uji iOS baru

Untuk memigrasikan pengujian yang ada dari host lama ke host pengujian macOS baru, Anda perlu mengembangkan file spesifikasi pengujian baru berdasarkan yang sudah ada sebelumnya.

Pendekatan yang disarankan adalah memulai dengan contoh file spesifikasi pengujian untuk jenis pengujian yang Anda inginkan, lalu memigrasikan perintah yang relevan dari file spesifikasi pengujian lama Anda ke yang baru. Ini memungkinkan Anda memanfaatkan fitur baru dan pengoptimalan contoh spesifikasi pengujian untuk host baru saat menggunakan kembali cuplikan kode Anda yang ada.

Topik

- Tutorial: Memigrasi file spesifikasi pengujian iOS dengan konsol
- Perbedaan antara host uji baru dan lama

Tutorial: Memigrasi file spesifikasi pengujian iOS dengan konsol

Dalam contoh ini, konsol Device Farm akan digunakan untuk onboard spesifikasi pengujian perangkat iOS yang ada untuk menggunakan host pengujian baru.

Langkah 1: Membuat file spesifikasi pengujian baru dengan konsol

1. Masuk ke konsol AWS Device Farm.
2. Arahkan ke proyek Device Farm yang berisi pengujian otomatisasi Anda.
3. Unduh salinan spesifikasi pengujian yang ada yang ingin Anda ikuti.
    a. Klik opsi “Pengaturan Proyek” dan arahkan ke tab Unggahan.
    b. Arahkan ke file spesifikasi pengujian yang ingin Anda gunakan.
    c. Klik tombol Unduh untuk membuat salinan lokal file ini.
4. Arahkan kembali ke halaman Project dan klik Create run.
5. Isi opsi pada wizard seolah-olah Anda akan memulai proses baru, tetapi berhenti di opsi Pilih spesifikasi pengujian.
6. Menggunakan spesifikasi pengujian iOS yang dipilih secara default, klik tombol Buat spesifikasi pengujian.

7. Ubah spesifikasi pengujian yang dipilih secara default di editor teks.

   a. Jika belum ada, ubah file spesifikasi pengujian untuk memilih host baru menggunakan:

   ```
   ios_test_host: macos_sequoia
   ```

   b. Dari salinan spesifikasi pengujian Anda yang diunduh pada langkah sebelumnya, tinjau masing-masing `phase`.

   c. Salin perintah dari fase spesifikasi pengujian lama ke setiap fase masing-masing dalam spesifikasi pengujian baru, mengabaikan perintah yang terkait dengan menginstal atau memilih Java, Python, Node.js, Ruby, Appium, atau Xcode.

8. Masukkan nama file baru di kotak teks Simpan sebagai.
9. Klik tombol Simpan sebagai baru untuk menyimpan perubahan Anda.

Untuk contoh file spesifikasi pengujian yang dapat Anda gunakan sebagai referensi, lihat contoh yang disediakan diContoh spesifikasi uji.

Langkah 2: Memilih perangkat lunak pra-instal perangkat lunak

Di host uji baru, versi perangkat lunak pra-instal dipilih menggunakan alat manajemen versi standar baru yang disebut. `devicefarm-cli` Perkakas ini sekarang merupakan pendekatan yang direkomendasikan untuk menggunakan berbagai perangkat lunak yang kami sediakan pada host uji.

Sebagai contoh, Anda akan menambahkan baris berikut untuk menggunakan JDK 17 yang berbeda lingkungan pengujian Anda:

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

Untuk informasi lebih lanjut tentang perangkat lunak yang didukung tersedia, silakan tinjau:Perangkat lunak yang didukung dalam lingkungan pengujian khusus.

Langkah 3: Menggunakan Appium dan dependensinya melalui alat pemilihan perangkat lunak

Host uji baru hanya mendukung Appium 2.x ke atas. Harap secara eksplisit pilih versi Appium menggunakan`devicefarm-cli`, sambil menghapus alat lama seperti. `avm` Contoh:

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

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

Versi Appium yang dipilih `devicefarm-cli` dilengkapi dengan prainstal dengan versi driver yang kompatibel XCUITest untuk iOS.

Selain itu, Anda perlu memperbarui spesifikasi pengujian Anda untuk digunakan `DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V9` alih-alih. `DEVICEFARM_WDA_DERIVED_DATA_PATH` Variabel lingkungan baru menunjuk ke versi pra-bangun WebDriverAgent 9.x, yang merupakan versi terbaru yang didukung untuk pengujian Appium 2.

Untuk informasi lebih lanjut, tinjau Memilih WebDriverAgent versi untuk pengujian iOS danVariabel lingkungan untuk tes Appium.

Perbedaan antara host uji baru dan lama

Saat mengedit file spesifikasi pengujian untuk menggunakan host uji iOS baru dan mentransisikan pengujian Anda dari host pengujian lama, perhatikan perbedaan lingkungan utama berikut:

- Versi Xcode: Di lingkungan host uji lama, versi Xcode yang tersedia didasarkan pada versi iOS perangkat yang digunakan untuk pengujian. Misalnya, pengujian pada perangkat iOS 18 menggunakan Xcode 16 di host lama, sedangkan pengujian di iOS 17 menggunakan Xcode 15. Di lingkungan host baru, semua perangkat dapat mengakses versi Xcode yang sama, memungkinkan lingkungan yang konsisten untuk pengujian pada perangkat dengan versi berbeda. Untuk daftar versi Xcode yang tersedia saat ini, lihatPerangkat lunak yang didukung.
- Memilih versi perangkat lunak: Dalam banyak kasus, versi perangkat lunak default telah berubah, jadi jika Anda tidak secara eksplisit memilih versi perangkat lunak Anda di host uji lama sebelumnya, Anda mungkin ingin menentukannya sekarang di host uji baru menggunakan. `devicefarm-cli` Dalam sebagian besar kasus penggunaan, kami menyarankan agar pelanggan secara eksplisit memilih versi perangkat lunak yang mereka gunakan. Dengan memilih versi perangkat lunak, `devicefarm-cli` Anda akan memiliki pengalaman yang dapat diprediksi dan konsisten dengannya dan menerima banyak peringatan jika Device Farm berencana untuk menghapus versi tersebut dari host pengujian.

  Selain itu, alat pemilihan perangkat lunak seperti `nvm` `pyenv` `avm`,,, dan `rvm` telah dihapus demi sistem pemilihan `devicefarm-cli` perangkat lunak baru.
- Versi perangkat lunak yang tersedia: Banyak versi perangkat lunak pra-instal sebelumnya telah dihapus, dan banyak versi baru telah ditambahkan. Jadi, pastikan bahwa ketika menggunakan

`devicefarm-cli` untuk memilih versi perangkat lunak Anda, Anda memilih versi yang ada dalam daftar versi yang didukung.

- **`libimobiledevice`**Rangkaian alat telah dihapus demi perkakas yang lebih baru/pihak pertama untuk melacak pengujian perangkat iOS saat ini dan standar industri. Untuk iOS 17 ke atas, Anda dapat memigrasikan sebagian besar perintah untuk menggunakan perkakas Xcode serupa, yang disebut. `devicectl` Untuk informasi tentang`devicectl`, Anda dapat menjalankan `xcrun devicectl help` dari mesin dengan Xcode diinstal.
- Jalur file yang dikodekan keras dalam file spesifikasi pengujian host lama Anda sebagai jalur absolut kemungkinan besar tidak akan berfungsi seperti yang diharapkan di host pengujian baru, dan umumnya tidak direkomendasikan untuk penggunaan file spesifikasi pengujian. Kami menyarankan Anda menggunakan jalur relatif dan variabel lingkungan untuk semua kode file spesifikasi pengujian. Untuk informasi lebih lanjut, tinjau topik diPraktik terbaik untuk eksekusi lingkungan pengujian kustom.
- Versi dan arsitektur sistem operasi: Host uji lama menggunakan berbagai versi macOS dan arsitektur CPU berdasarkan perangkat yang ditetapkan. Akibatnya, pengguna mungkin melihat beberapa perbedaan dalam pustaka sistem yang tersedia yang tersedia di lingkungan. Untuk informasi lebih lanjut tentang versi OS host sebelumnya, tinjauHost uji iOS lama.
- Untuk pengguna Appium, cara memilih WebDriverAgent telah berubah menjadi awalan variabel lingkungan penggunaan `DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V` alih-alih awalan lama. `DEVICEFARM_WDA_DERIVED_DATA_PATH_V` Untuk informasi lebih lanjut tentang variabel yang diperbarui, tinjauVariabel lingkungan untuk tes Appium.
- Untuk pengguna Appium Java, host uji baru tidak berisi file JAR pra-instal di jalur kelasnya, sedangkan host sebelumnya berisi satu untuk kerangka TestNG (melalui variabel lingkungan). `$DEVICEFARM_TESTNG_JAR` Kami menyarankan agar pelanggan mengemas file JAR yang diperlukan untuk kerangka kerja pengujian mereka di dalam paket pengujian mereka dan menghapus instance `$DEVICEFARM_TESTNG_JAR` variabel dari file spesifikasi pengujian mereka.

Kami merekomendasikan untuk menghubungi tim layanan melalui kasus dukungan jika Anda memiliki umpan balik atau pertanyaan tentang perbedaan antara host uji dari perspektif perangkat lunak.

Akses sumber daya AWS menggunakan Peran Eksekusi IAM

Device Farm mendukung penetapan peran IAM yang akan diasumsikan oleh lingkungan runtime pengujian kustom selama eksekusi pengujian. Fitur ini memungkinkan pengujian Anda mengakses

sumber daya AWS dengan aman di akun Anda, seperti bucket Amazon S3, tabel DynamoDB, atau layanan AWS lainnya yang menjadi sandaran aplikasi Anda.

Topik

Gambaran umum

Saat Anda menentukan peran eksekusi IAM, Device Farm akan mengasumsikan peran ini selama eksekusi pengujian, memungkinkan pengujian Anda berinteraksi dengan layanan AWS menggunakan izin yang ditentukan dalam peran tersebut.

Kasus penggunaan umum untuk peran eksekusi IAM meliputi:

- Mengakses data uji yang disimpan di bucket Amazon S3
- Mendorong artefak uji ke ember Amazon S3
- Mengambil konfigurasi aplikasi dari AWS AppConfig
- Menulis log pengujian dan metrik ke Amazon CloudWatch
- Mengirim hasil pengujian atau pesan status ke antrian Amazon SQS
- Memanggil fungsi AWS Lambda sebagai bagian dari alur kerja pengujian

Persyaratan peran IAM

Untuk menggunakan peran eksekusi IAM dengan Device Farm, peran Anda harus memenuhi persyaratan berikut:

- Hubungan kepercayaan: Prinsipal layanan Device Farm harus dipercaya untuk mengambil peran tersebut. Kebijakan kepercayaan harus dimasukkan `devicefarm.amazonaws.com` sebagai entitas tepercaya.
- Izin: Peran harus memiliki izin yang diperlukan untuk mengakses sumber daya AWS yang diperlukan pengujian Anda.

- Durasi sesi: Durasi sesi maksimum peran harus setidaknya selama pengaturan batas waktu kerja proyek Device Farm Anda. Secara default, proyek Device Farm memiliki batas waktu kerja 150 menit, sehingga peran Anda harus mendukung durasi sesi minimal 150 menit.
- Persyaratan akun yang sama: Peran IAM harus berada di akun AWS yang sama dengan yang digunakan untuk memanggil Device Farm. Asumsi peran lintas akun tidak didukung.
- PassRole izin: Penelepon harus diberi wewenang untuk meneruskan peran IAM dengan kebijakan yang mengizinkan `iam:PassRole` tindakan pada peran eksekusi yang ditentukan.

Contoh kebijakan kepercayaan

Contoh berikut menunjukkan kebijakan kepercayaan yang memungkinkan Device Farm mengambil peran eksekusi Anda. Kebijakan kepercayaan ini hanya boleh dilampirkan pada peran IAM tertentu yang ingin Anda gunakan dengan Device Farm, bukan peran lain di akun Anda:

Example

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

Contoh kebijakan izin

Contoh berikut menunjukkan kebijakan izin yang memberikan akses ke layanan AWS umum yang digunakan dalam pengujian:

Example

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

Mengkonfigurasi peran eksekusi IAM

Anda dapat menentukan peran eksekusi IAM di tingkat proyek atau untuk pengujian individual. Ketika dikonfigurasi pada tingkat proyek, semua berjalan dalam proyek itu akan mewarisi peran eksekusi. Peran eksekusi yang dikonfigurasi saat menjalankan akan menggantikan apa pun yang dikonfigurasi pada proyek induknya.

Untuk petunjuk terperinci tentang mengonfigurasi peran eksekusi, lihat:

- Membuat proyek di AWS Device Farm- untuk mengkonfigurasi peran eksekusi di tingkat proyek
- Membuat uji coba di Device Farm- untuk mengonfigurasi peran eksekusi untuk proses individu

Anda juga dapat mengonfigurasi peran eksekusi menggunakan Device Farm API. Untuk informasi selengkapnya, lihat Referensi API Device Farm.

Praktik terbaik

Ikuti praktik terbaik ini saat mengonfigurasi peran eksekusi IAM untuk pengujian Device Farm Anda:

- Prinsip hak istimewa terkecil: Berikan hanya izin minimum yang diperlukan agar pengujian Anda berfungsi. Hindari menggunakan izin yang terlalu luas seperti * tindakan atau sumber daya.
- Gunakan izin khusus sumber daya: Jika memungkinkan, batasi izin untuk sumber daya tertentu (misalnya, bucket S3 tertentu atau tabel DynamoDB) daripada semua sumber daya dari suatu jenis.
- Sumber daya pengujian dan produksi terpisah: Gunakan sumber daya dan peran pengujian khusus untuk menghindari pengaruh sistem produksi secara tidak sengaja selama pengujian.
- Tinjauan peran reguler: Tinjau dan perbarui peran eksekusi Anda secara berkala untuk memastikan peran tersebut tetap memenuhi kebutuhan pengujian Anda dan mengikuti praktik terbaik keamanan.
- Gunakan tombol kondisi: Pertimbangkan untuk menggunakan kunci kondisi IAM untuk lebih membatasi kapan dan bagaimana peran dapat digunakan.

Pemecahan masalah

Jika Anda mengalami masalah dengan peran eksekusi IAM, periksa hal berikut:

- Hubungan kepercayaan: Verifikasi bahwa kebijakan kepercayaan peran termasuk `devicefarm.amazonaws.com` sebagai layanan tepercaya.
- Izin: Periksa apakah peran tersebut memiliki izin yang diperlukan untuk layanan AWS yang coba diakses pengujian Anda.
- Log pengujian: Tinjau log eksekusi pengujian untuk pesan kesalahan tertentu yang terkait dengan panggilan AWS API atau penolakan izin.

Variabel lingkungan untuk lingkungan pengujian khusus

Device Farm secara dinamis mengonfigurasi beberapa variabel lingkungan untuk digunakan sebagai bagian dari lingkungan pengujian kustom yang dijalankan.

Topik

Variabel lingkungan khusus

Device Farm mendukung konfigurasi pasangan nilai kunci yang diterapkan sebagai variabel lingkungan pada host pengujian. Ini dapat dikonfigurasi pada proyek Device Farm atau selama pembuatan run; variabel apa pun yang dikonfigurasi saat menjalankan akan menggantikan variabel apa pun yang mungkin dikonfigurasi pada proyek induknya. Pembatasan berikut berlaku:

- Variabel lingkungan khusus tidak didukung pada host uji iOS lama. Untuk informasi selengkapnya, lihat Host uji iOS lama.
- Nama variabel yang dimulai dengan `$DEVICEFARM_` dicadangkan untuk penggunaan layanan internal.
- Variabel lingkungan khusus tidak dapat digunakan untuk mengonfigurasi pemilihan komputasi host pengujian dalam spesifikasi pengujian Anda.

Variabel lingkungan umum

Bagian ini menjelaskan variabel lingkungan yang umum untuk semua pengujian di Device Farm.

$DEVICEFARM_DEVICE_NAME

Perangkat tempat pengujian Anda dijalankan. Ini mewakili pengenal perangkat unik (UDID) perangkat.

$DEVICEFARM_DEVICE_UDID

Pengidentifikasi unik perangkat.

$DEVICEFARM_DEVICE_PLATFORM_NAME

Nama platform perangkat. Itu salah satu `Android` atau`iOS`.

$DEVICEFARM_DEVICE_OS_VERSION

Versi OS perangkat.

$DEVICEFARM_APP_PATH

(tes aplikasi seluler)

Jalur ke aplikasi seluler di mesin host tempat pengujian dijalankan. Variabel ini tidak tersedia selama pengujian web.

$DEVICEFARM_LOG_DIR

Jalur ke direktori default tempat log pelanggan, artefak, dan file yang diinginkan lainnya akan disimpan untuk pengambilan nanti. Menggunakan contoh spesifikasi pengujian, file dalam direktori ini diarsipkan dalam file ZIP dan tersedia sebagai artefak setelah pengujian Anda dijalankan.

$DEVICEFARM_SCREENSHOT_PATH

Jalur ke tangkapan layar, jika ada, ditangkap selama uji coba.

$DEVICEFARM_PROJECT_ARN

ARN dari proyek induk pekerjaan.

$DEVICEFARM_RUN_ARN

ARN dari orang tua pekerjaan dijalankan.

$DEVICEFARM_DEVICE_ARN

ARN perangkat yang diuji.

$DEVICEFARM_TOTAL_JOBS

Jumlah total pekerjaan yang terkait dengan induk Device Farm run.

$DEVICEFARM_JOB_NUMBER

Jumlah pekerjaan ini di dalamnya$DEVICEFARM_TOTAL_JOBS. Misalnya, sebuah run mungkin berisi 5 pekerjaan, dan masing-masing akan memiliki unik $DEVICEFARM_JOB_NUMBER mulai dari 0 hingga 4.

$AWS_REGION

Wilayah AWS. Layanan akan mengatur ini agar sesuai dengan wilayah di mana perangkat yang diuji berada. Hal ini dapat diganti oleh variabel lingkungan kustom jika diperlukan.

$ANDROID_HOME

(Hanya Android)

Jalur ke direktori instalasi Android SDK.

Variabel lingkungan untuk tes Appium

Bagian ini menjelaskan variabel lingkungan yang digunakan oleh pengujian Appium apa pun di lingkungan pengujian kustom di Device Farm.

$DEVICEFARM_CHROMEDRIVER_EXECUTABLE_DIR

(Hanya Android)

Lokasi direktori yang berisi ChromeDriver executable yang diperlukan untuk digunakan dalam web Appium dan tes hybrid.

$DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V<N>

(Hanya iOS)

Jalur data turunan dari versi yang WebDriverAgent dibuat untuk dijalankan di Device Farm. Penomoran pada variabel akan sesuai dengan versi utama dari. WebDriverAgent Sebagai contoh, DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V9 akan menunjuk ke WebDriverAgent versi 9.x. Untuk informasi selengkapnya, lihat Memilih WebDriverAgent versi untuk pengujian iOS.

> **Note**
>
> Variabel $DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V<N> lingkungan hanya ada di host iOS non-warisan. Untuk informasi selengkapnya, lihat Host uji iOS lama.

$DEVICEFARM_WDA_DERIVED_DATA_PATH_V9

(Hanya iOS, tidak digunakan lagi)

Jalur data turunan dari versi yang WebDriverAgent dibuat untuk dijalankan di Device Farm. Lihat skema `$DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V<N>` penamaan pengganti.

Variabel lingkungan untuk XCUITest pengujian

Bagian ini menjelaskan variabel lingkungan yang digunakan oleh XCUITest pengujian di lingkungan pengujian kustom di Device Farm.

$DEVICEFARM_XCUITESTRUN_FILE

Path ke `.xctestun` file Device Farm. Ini dihasilkan dari aplikasi dan paket pengujian Anda.

$DEVICEFARM_DERIVED_DATA_PATH

Jalur yang diharapkan dari keluaran Device Farm xcodebuild.

$DEVICEFARM_XCTEST_BUILD_DIRECTORY

Jalur ke konten yang tidak di-zip dari file paket pengujian.

Praktik terbaik untuk eksekusi lingkungan pengujian kustom

Topik berikut mencakup praktik terbaik yang direkomendasikan untuk menggunakan eksekusi pengujian kustom dengan Device Farm.

Jalankan konfigurasi

- Andalkan perangkat lunak yang dikelola Device Farm dan fitur API untuk menjalankan konfigurasi sedapat mungkin, sebagai lawan menerapkan konfigurasi serupa melalui perintah shell dalam file spesifikasi pengujian. Ini termasuk konfigurasi host uji dan perangkat, karena ini akan lebih berkelanjutan dan konsisten di seluruh host dan perangkat uji.

  Meskipun Device Farm mendorong Anda untuk menyesuaikan file spesifikasi pengujian sebanyak yang Anda butuhkan untuk menjalankan pengujian, file spesifikasi pengujian dapat menjadi sulit dipertahankan seiring waktu karena lebih banyak perintah khusus ditambahkan ke dalamnya. Menggunakan perangkat lunak yang dikelola Device Farm (melalui alat seperti `devicefarm-cli` dan alat default yang tersedia di`$PATH`), dan menggunakan fitur terkelola (seperti parameter

`deviceProxy`permintaan) untuk menyederhanakan file spesifikasi pengujian dengan mengalihkan tanggung jawab pemeliharaan ke Device Farm itu sendiri.

Spesifikasi uji dan kode paket uji

- Jangan gunakan jalur absolut atau mengandalkan versi minor tertentu dalam file spesifikasi pengujian atau kode paket pengujian Anda. Device Farm menerapkan pembaruan rutin ke host pengujian yang dipilih dan versi perangkat lunak yang disertakan. Menggunakan jalur spesifik atau absolut (seperti `/usr/local/bin/python` alih-alih`python`) atau memerlukan versi minor tertentu (seperti Node.js `20.3.1` bukan hanya `20`) dapat menyebabkan pengujian Anda gagal menemukan file/yang dapat dieksekusi yang diperlukan.

  Sebagai bagian dari eksekusi pengujian kustom, Device Farm menyiapkan berbagai variabel lingkungan dan `$PATH` variabel untuk memastikan pengujian memiliki pengalaman yang konsisten dalam lingkungan dinamis kami. Lihat Variabel lingkungan untuk lingkungan pengujian khusus dan Perangkat lunak yang didukung dalam lingkungan pengujian khusus untuk informasi lebih lanjut.
- Simpan file yang dihasilkan atau disalin dalam direktori temp selama uji coba. Hari ini, kami memastikan bahwa direktori temp (`/tmp`) akan dapat diakses oleh pengguna selama eksekusi pengujian (selain direktori terkelola, seperti). `$DEVICEFARM_LOG_DIR` Direktori lain yang dapat diakses pengguna dapat berubah seiring waktu karena kebutuhan layanan atau sistem operasi yang digunakan.
- Simpan log eksekusi pengujian Anda ke **`$DEVICEFARM_LOG_DIR`**. Ini adalah direktori artefak default yang disediakan untuk eksekusi Anda untuk menambahkan log eksekusi /artefak ke dalam. Contoh spesifikasi pengujian yang kami sediakan masing-masing menggunakan direktori ini untuk artefak secara default.
- Pastikan perintah Anda mengembalikan kode bukan nol pada kegagalan selama `test` fase spesifikasi pengujian Anda. Kami menentukan apakah eksekusi Anda gagal dengan memeriksa kode keluar bukan nol dari setiap perintah shell yang dipanggil selama fase. `test` Anda harus memastikan logika atau kerangka pengujian Anda akan mengembalikan kode keluar bukan nol untuk semua skenario yang diinginkan, yang mungkin memerlukan konfigurasi tambahan.

  Misalnya, kerangka kerja pengujian tertentu (seperti JUnit5) tidak menganggap nol pengujian dijalankan sebagai kegagalan, yang akan menyebabkan pengujian Anda terdeteksi telah berjalan dengan sukses meskipun tidak ada yang dieksekusi. Menggunakan JUnit5 sebagai contoh, Anda perlu menentukan opsi baris perintah `--fail-if-no-tests` untuk memastikan skenario ini keluar dengan kode keluar bukan nol.

- Tinjau kompatibilitas perangkat lunak dengan versi OS perangkat dan versi host uji yang akan Anda gunakan untuk uji coba. Sebagai contoh, ada fitur tertentu dalam menguji kerangka kerja perangkat lunak (yaitu: Appium) yang mungkin tidak berfungsi sebagaimana dimaksud pada semua versi OS perangkat yang sedang diuji.

Keamanan

- Hindari menyimpan atau mencatat variabel sensitif (seperti kunci AWS) di file spesifikasi pengujian Anda. File spesifikasi pengujian, skrip yang dihasilkan spesifikasi pengujian, dan log skrip spesifikasi pengujian semuanya disediakan sebagai artefak yang dapat diunduh di akhir eksekusi pengujian. Hal ini dapat menyebabkan pemaparan rahasia yang tidak diinginkan untuk pengguna lain di akun Anda dengan akses baca ke uji coba Anda.

Memigrasi pengujian dari lingkungan pengujian standar ke kustom

Anda dapat beralih dari mode eksekusi pengujian standar ke mode eksekusi khusus di AWS Device Farm. Migrasi terutama melibatkan dua bentuk eksekusi yang berbeda:

1. Mode standar: Mode eksekusi pengujian ini terutama dibangun untuk menyediakan pelaporan terperinci dan lingkungan yang dikelola sepenuhnya kepada pelanggan.
2. Mode kustom: Mode eksekusi pengujian ini dibuat untuk berbagai kasus penggunaan yang memerlukan uji coba lebih cepat, kemampuan untuk mengangkat dan menggeser dan mencapai paritas dengan lingkungan lokal mereka, dan streaming video langsung.

Untuk informasi selengkapnya tentang mode standar dan kustom di Device Farm, lihat Uji lingkungan di AWS Device Farm danLingkungan pengujian khusus di AWS Device Farm.

Pertimbangan saat bermigrasi

Bagian ini mencantumkan beberapa kasus penggunaan yang menonjol untuk dipertimbangkan saat bermigrasi ke mode kustom:

1. Kecepatan: Dalam mode eksekusi standar, Device Farm mem-parsing metadata pengujian yang telah dikemas dan diunggah menggunakan instruksi pengemasan untuk kerangka kerja khusus Anda. Parsing mendeteksi jumlah tes dalam paket Anda. Setelah itu, Device Farm menjalankan setiap pengujian secara terpisah dan menyajikan log, video, dan artefak hasil lainnya secara

individual untuk setiap pengujian. Namun, ini terus menambah total waktu eksekusi end-to-end pengujian karena ada pra dan pasca pemrosesan pengujian dan artefak hasil pada akhir layanan.

Sebaliknya, mode eksekusi kustom tidak mengurai paket pengujian Anda; ini berarti tidak ada pra-pemrosesan dan pasca-pemrosesan minimal untuk pengujian atau artefak hasil. Ini menghasilkan total waktu end-to-end eksekusi dekat dengan pengaturan lokal Anda. Pengujian dijalankan dalam format yang sama seperti jika dijalankan pada mesin lokal Anda. Hasil tes sama dengan apa yang Anda dapatkan secara lokal dan tersedia untuk diunduh di akhir pelaksanaan pekerjaan.

2. Kustomisasi atau Fleksibilitas: Mode eksekusi standar mem-parsing paket pengujian Anda untuk mendeteksi jumlah pengujian dan kemudian menjalankan setiap pengujian secara terpisah. Perhatikan bahwa tidak ada jaminan bahwa tes akan berjalan sesuai urutan yang Anda tentukan. Akibatnya, tes yang membutuhkan urutan eksekusi tertentu mungkin tidak berfungsi seperti yang diharapkan. Selain itu, tidak ada cara untuk menyesuaikan lingkungan mesin host atau meneruskan file konfigurasi yang mungkin diperlukan untuk menjalankan pengujian Anda dengan cara tertentu.

   Sebaliknya, mode kustom memungkinkan Anda mengonfigurasi lingkungan mesin host termasuk kemampuan untuk menginstal perangkat lunak tambahan, meneruskan filter ke pengujian Anda, meneruskan file konfigurasi, dan mengontrol pengaturan eksekusi pengujian. Ini mencapai ini melalui file yaml (juga disebut file testspec) yang dapat Anda modifikasi dengan menambahkan perintah shell ke dalamnya. File yaml ini akan dikonversi ke skrip shell yang dieksekusi pada mesin host uji. Anda dapat menyimpan beberapa file yaml dan memilih satu secara dinamis sesuai kebutuhan Anda saat Anda menjadwalkan proses.

3. Video langsung dan logging: Mode eksekusi standar dan kustom memberi Anda video dan log untuk pengujian Anda. Namun, dalam mode standar, Anda mendapatkan video dan log pengujian yang telah ditentukan sebelumnya hanya setelah pengujian Anda selesai.

   Sebaliknya, mode kustom memberi Anda streaming langsung video dan log sisi klien pengujian Anda. Selain itu, Anda dapat mengunduh video dan artefak lainnya di akhir tes.

> **Tip**
>
> Jika kasus penggunaan Anda melibatkan setidaknya satu dari faktor di atas, kami sangat menyarankan untuk beralih ke mode eksekusi kustom.

Langkah migrasi

Untuk bermigrasi dari mode standar ke mode khusus, lakukan hal berikut:

1. Masuk ke Konsol Manajemen AWS dan buka konsol Device Farm di https://console.aws.amazon.com/devicefarm/.
2. Pilih proyek Anda dan kemudian mulai menjalankan otomatisasi baru.
3. Unggah aplikasi Anda (atau pilih`web app`), pilih jenis kerangka pengujian Anda, unggah paket pengujian Anda, lalu di bawah `Choose your execution environment` parameter, pilih opsi untuk`Run your test in a custom environment`.
4. Secara default, file spesifikasi pengujian contoh Device Farm akan muncul untuk Anda lihat dan edit. File contoh ini dapat digunakan sebagai tempat awal untuk mencoba pengujian Anda dalam mode lingkungan khusus. Kemudian, setelah Anda memverifikasi bahwa pengujian Anda berfungsi dengan baik dari konsol, Anda kemudian dapat mengubah integrasi API, CLI, dan pipeline apa pun dengan Device Farm untuk menggunakan file spesifikasi pengujian ini sebagai parameter saat penjadwalan pengujian berjalan. Untuk informasi tentang cara menambahkan file spesifikasi pengujian sebagai parameter untuk proses Anda, lihat bagian `testSpecArn` parameter untuk `ScheduleRun` API di panduan API kami.

Kerangka Appium

Dalam lingkungan pengujian khusus, Device Farm tidak menyisipkan atau mengganti kemampuan Appium apa pun dalam pengujian kerangka kerja Appium Anda. Anda harus menentukan kemampuan Appium pengujian Anda baik dalam file YAMM spesifikasi pengujian atau kode pengujian Anda.

Instrumentasi Android

Anda tidak perlu membuat perubahan untuk memindahkan pengujian instrumentasi Android ke lingkungan pengujian khusus.

iOS XCUITest

Anda tidak perlu membuat perubahan untuk memindahkan XCUITest pengujian iOS Anda ke lingkungan pengujian khusus.

Memperluas lingkungan pengujian khusus di Device Farm

AWS Device Farm memungkinkan mengonfigurasi lingkungan khusus untuk pengujian otomatis (mode kustom), yang merupakan pendekatan yang disarankan untuk semua pengguna Device Farm. Mode kustom Device Farm memungkinkan Anda menjalankan lebih dari sekadar rangkaian pengujian. Di bagian ini, Anda mempelajari cara memperluas rangkaian pengujian dan mengoptimalkan pengujian Anda.

Untuk informasi selengkapnya tentang lingkungan pengujian kustom di Device Farm, lihatLingkungan pengujian khusus di AWS Device Farm.

Topik

- Menyetel PIN perangkat saat menjalankan pengujian di Device Farm
- Mempercepat pengujian berbasis Appium di Device Farm melalui kemampuan yang diinginkan
- Menggunakan Webhook dan lainnya APIs setelah pengujian dijalankan di Device Farm
- Menambahkan file tambahan ke paket pengujian Anda di Device Farm

Menyetel PIN perangkat saat menjalankan pengujian di Device Farm

Beberapa aplikasi mengharuskan Anda mengatur PIN pada perangkat. Device Farm tidak mendukung pengaturan PIN pada perangkat secara native. Namun, ini dimungkinkan dengan peringatan berikut:

- Perangkat harus menjalankan Android 8 atau lebih tinggi.
- PIN harus dilepas setelah tes selesai.

Untuk menyetel PIN dalam pengujian Anda, gunakan `post_test` fase `pre_test` dan untuk mengatur dan menghapus PIN, seperti yang ditunjukkan berikut:

```
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"
```

Saat rangkaian pengujian Anda dimulai, PIN 1234 disetel. Setelah rangkaian pengujian Anda keluar, PIN akan dihapus.

> **Warning**
>
> Jika Anda tidak menghapus PIN dari perangkat setelah pengujian selesai, perangkat dan akun Anda akan dikarantina.

Untuk lebih banyak cara untuk memperluas rangkaian pengujian dan mengoptimalkan pengujian, lihatMemperluas lingkungan pengujian khusus di Device Farm.

Mempercepat pengujian berbasis Appium di Device Farm melalui kemampuan yang diinginkan

Saat menggunakan Appium, Anda mungkin menemukan bahwa rangkaian pengujian mode standar sangat lambat. Ini karena Device Farm menerapkan pengaturan default dan tidak membuat asumsi tentang bagaimana Anda ingin menggunakan lingkungan Appium. Meskipun default ini dibangun di sekitar praktik terbaik industri, mereka mungkin tidak berlaku untuk situasi Anda. Untuk menyempurnakan parameter server Appium, Anda dapat menyesuaikan kemampuan Appium default dalam spesifikasi pengujian Anda. Misalnya, berikut ini menyetel `usePrebuildWDA` kemampuan ke `true` dalam rangkaian pengujian iOS untuk mempercepat waktu mulai awal:

```
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 &
```

Kemampuan Appium harus berupa struktur JSON yang dikutip dari cangkang.

Kemampuan Appium berikut adalah sumber umum peningkatan kinerja:

`noReset` dan `fullReset`

Kedua kemampuan ini, yang saling eksklusif, menggambarkan perilaku Appium setelah setiap sesi selesai. Ketika `noReset` disetel `ketrue`, server Appium tidak menghapus data dari aplikasi Anda ketika sesi Appium berakhir, secara efektif tidak melakukan pembersihan apa pun. `fullReset`menghapus instalasi dan menghapus semua data aplikasi dari perangkat setelah sesi ditutup. Untuk informasi selengkapnya, lihat Reset Strategi dalam dokumentasi Appium.

`ignoreUnimportantViews`(Hanya Android)

Menginstruksikan Appium untuk mengompres hierarki UI Android hanya ke tampilan yang relevan untuk pengujian, mempercepat pencarian elemen tertentu. Namun, ini dapat merusak beberapa suite pengujian XPath berbasis karena hierarki tata letak UI telah diubah.

`skipUnlock`(Hanya Android)

Menginformasikan Appium bahwa tidak ada kode PIN yang saat ini disetel, yang mempercepat pengujian setelah peristiwa layar mati atau peristiwa kunci lainnya.

`webDriverAgentUrl`(Hanya iOS)

Menginstruksikan Appium untuk menganggap bahwa ketergantungan iOS penting,`webDriverAgent`, sudah berjalan dan tersedia untuk menerima permintaan HTTP di URL yang ditentukan. Jika `webDriverAgent` belum aktif dan berjalan, Appium membutuhkan waktu beberapa saat di awal rangkaian pengujian untuk memulai. `webDriverAgent` Jika Anda memulai `webDriverAgent` sendiri dan mengatur `webDriverAgentUrl` ke `http://localhost:8100` saat memulai Appium, Anda dapat mem-boot suite pengujian Anda lebih cepat. Perhatikan bahwa kemampuan ini tidak boleh digunakan bersama `useNewWDA` kemampuan.

Anda dapat menggunakan kode berikut untuk memulai `webDriverAgent` dari file spesifikasi pengujian di port lokal perangkat`8100`, lalu meneruskannya ke port lokal host pengujian `8100` (ini memungkinkan Anda untuk menetapkan `webDriverAgentUrl` nilainyahttp://localhost:8100). Kode ini harus dijalankan selama fase penginstalan setelah kode apa pun untuk menyiapkan variabel Appium dan `webDriverAgent` lingkungan telah ditentukan:

```
      # 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 &
```

Kemudian, Anda dapat menambahkan kode berikut ke file spesifikasi pengujian Anda untuk memastikannya berhasil `webDriverAgent` dimulai. Kode ini harus dijalankan pada akhir fase pra-pengujian setelah memastikan bahwa Appium berhasil dimulai:

```
      # 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;
```

Untuk informasi selengkapnya tentang kemampuan yang didukung Appium, lihat Kemampuan yang Diinginkan Appium dalam dokumentasi Appium.

Untuk lebih banyak cara untuk memperluas rangkaian pengujian dan mengoptimalkan pengujian, lihatMemperluas lingkungan pengujian khusus di Device Farm.

Menggunakan Webhook dan lainnya APIs setelah pengujian dijalankan di Device Farm

Anda dapat meminta Device Farm memanggil webhook setelah setiap rangkaian pengujian selesai digunakan. curl Proses untuk melakukan ini bervariasi dengan tujuan dan pemformatan. Untuk webhook spesifik Anda, lihat dokumentasi untuk webhook tersebut. Contoh berikut memposting pesan setiap kali rangkaian pengujian selesai ke webhook Slack:

```
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
```

Untuk informasi selengkapnya tentang penggunaan webhook dengan Slack, lihat Mengirim pesan Slack pertama Anda menggunakan Webhook di referensi Slack API.

Untuk lebih banyak cara untuk memperluas rangkaian pengujian dan mengoptimalkan pengujian, lihatMemperluas lingkungan pengujian khusus di Device Farm.

Anda tidak terbatas pada menggunakan curl untuk memanggil webhooks. Paket pengujian dapat menyertakan skrip dan alat tambahan, asalkan kompatibel dengan lingkungan eksekusi Device Farm. Misalnya, paket pengujian Anda mungkin menyertakan skrip tambahan yang membuat permintaan ke yang lain APIs. Pastikan bahwa setiap paket yang diperlukan diinstal bersamaan dengan persyaratan suite pengujian Anda. Untuk menambahkan skrip yang berjalan setelah rangkaian pengujian Anda selesai, sertakan skrip dalam paket pengujian Anda dan tambahkan yang berikut ini ke spesifikasi pengujian Anda:

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

> **Note**
>
> Mempertahankan kunci API atau token otentikasi lain yang digunakan dalam paket pengujian Anda adalah tanggung jawab Anda. Kami menyarankan agar Anda menyimpan segala bentuk kredensi keamanan di luar kendali sumber, menggunakan kredensil dengan hak istimewa sesedikit mungkin, dan menggunakan token yang dapat diulang dan berumur pendek bila memungkinkan. Untuk memverifikasi persyaratan keamanan, lihat dokumentasi untuk pihak ketiga APIs yang Anda gunakan.

Jika Anda berencana menggunakan AWS layanan sebagai bagian dari rangkaian eksekusi pengujian, Anda harus menggunakan kredenal sementara IAM, yang dihasilkan di luar rangkaian pengujian dan disertakan dalam paket pengujian Anda. Kredensi ini harus memiliki izin paling sedikit yang diberikan dan umur sesingkat mungkin. Untuk informasi selengkapnya tentang cara membuat kredensil sementara, lihat Meminta kredensil keamanan sementara di Panduan Pengguna IAM.

Untuk lebih banyak cara untuk memperluas rangkaian pengujian dan mengoptimalkan pengujian, lihatMemperluas lingkungan pengujian khusus di Device Farm.

Menambahkan file tambahan ke paket pengujian Anda di Device Farm

Anda mungkin ingin menggunakan file tambahan sebagai bagian dari pengujian Anda baik sebagai file konfigurasi tambahan atau data pengujian tambahan. Anda dapat menambahkan file tambahan ini ke paket pengujian sebelum mengunggahnya AWS Device Farm, lalu mengaksesnya dari mode lingkungan khusus. Pada dasarnya, semua format unggahan paket uji (ZIP, IPA, APK, JAR, dll.) Adalah format arsip paket yang mendukung operasi ZIP standar.

Anda dapat menambahkan file ke arsip pengujian sebelum mengunggahnya AWS Device Farm dengan menggunakan perintah berikut:

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

Untuk direktori file tambahan:

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

Perintah ini berfungsi seperti yang diharapkan untuk semua format unggahan paket pengujian kecuali untuk file IPA. Untuk file IPA, terutama saat digunakan XCUITests, kami sarankan Anda meletakkan

file tambahan di lokasi yang sedikit berbeda karena cara AWS Device Farm mengundurkan diri paket uji iOS. Saat membuat pengujian iOS Anda, direktori aplikasi pengujian akan berada di dalam direktori lain bernama*Payload*.

Misalnya, ini adalah bagaimana satu direktori pengujian iOS seperti itu terlihat:

```
$ 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
```

Untuk XCUITest paket-paket ini, tambahkan file tambahan ke direktori yang berakhir di *.app* dalam *Payload* direktori. Misalnya, perintah berikut menunjukkan bagaimana Anda dapat menambahkan file ke paket pengujian ini:

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

Saat menambahkan file ke paket pengujian, Anda dapat mengharapkan perilaku interaksi yang sedikit berbeda AWS Device Farm berdasarkan format unggahannya. Jika unggahan menggunakan ekstensi file ZIP, secara otomatis AWS Device Farm akan membuka zip unggahan sebelum pengujian Anda dan membiarkan file yang tidak di-zip di lokasi dengan variabel lingkungan. *$DEVICEFARM_TEST_PACKAGE_PATH* (Ini berarti bahwa jika Anda menambahkan file yang dipanggil *extra_file* ke root arsip seperti pada contoh pertama, itu akan ditempatkan di *$DEVICEFARM_TEST_PACKAGE_PATH/extra_file* selama pengujian).

Untuk menggunakan contoh yang lebih praktis, jika Anda adalah pengguna Appium TestNG yang ingin menyertakan *testng.xml* file dengan pengujian Anda, Anda dapat memasukkannya ke dalam arsip menggunakan perintah berikut:

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

Kemudian, Anda dapat mengubah perintah pengujian Anda dalam mode lingkungan khusus menjadi berikut:

```
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
```

Jika ekstensi unggahan paket pengujian Anda bukan ZIP (misalnya, file APK, IPA, atau JAR), file paket yang diunggah itu sendiri dapat ditemukan di. *$DEVICEFARM_TEST_PACKAGE_PATH* Karena ini masih file format arsip, Anda dapat unzip file untuk mengakses file tambahan dari dalam. Misalnya, perintah berikut akan membuka zip isi paket pengujian (untuk file APK, IPA, atau JAR) ke direktori: */tmp*

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

Dalam kasus file APK atau JAR, Anda akan menemukan file tambahan Anda dibuka ritsleting ke */tmp* direktori (misalnya,*/tmp/extra_file*). Dalam kasus file IPA, seperti yang dijelaskan

sebelumnya, file tambahan akan berada di lokasi yang sedikit berbeda di dalam folder yang diakhiri `.app`, yang ada di dalam `Payload` direktori. Misalnya, berdasarkan contoh IPA di atas, file akan ditemukan di lokasi `/tmp/Payload/ADFiOSReferenceAppUITests-Runner.app/extra_file` (dapat direferensikan sebagai). `/tmp/Payload/*.app/extra_file`

Untuk lebih banyak cara untuk memperluas rangkaian pengujian dan mengoptimalkan pengujian, lihatMemperluas lingkungan pengujian khusus di Device Farm.

Akses jarak jauh di AWS Device Farm

Akses jarak jauh memungkinkan Anda untuk menggesek, memberi isyarat, dan berinteraksi dengan perangkat melalui browser web Anda secara real time untuk menguji fungsionalitas dan mereproduksi masalah pelanggan. Anda berinteraksi dengan perangkat tertentu dengan membuat sesi akses jarak jauh dengan perangkat tersebut.

Sesi di Device Farm adalah interaksi real-time dengan perangkat fisik aktual yang dihosting di browser web. Sesi menampilkan perangkat tunggal yang Anda pilih saat memulai sesi. Seorang pengguna dapat memulai lebih dari satu sesi pada satu waktu dengan jumlah total perangkat simultan dibatasi oleh jumlah slot perangkat yang Anda miliki. Anda dapat membeli slot perangkat berdasarkan keluarga perangkat (perangkat Android atau iOS). Untuk informasi selengkapnya, lihat Harga Device Farm.

Device Farm saat ini menawarkan subset perangkat untuk pengujian akses jarak jauh. Perangkat baru ditambahkan ke kumpulan perangkat setiap saat.

Device Farm menangkap video dari setiap sesi akses jarak jauh dan menghasilkan log aktivitas selama sesi berlangsung. Hasil ini mencakup informasi apa pun yang Anda berikan selama sesi.

> **Note**
>
> Untuk alasan keamanan, kami menyarankan Anda menghindari memberikan atau memasukkan informasi sensitif, seperti nomor akun, informasi login pribadi, dan detail lainnya selama sesi akses jarak jauh. Jika memungkinkan, gunakan alternatif yang dikembangkan khusus untuk pengujian, seperti akun uji.

Topik

Membuat sesi akses jarak jauh di AWS Device Farm

Untuk informasi tentang sesi akses jarak jauh, lihatSesi.

Prasyarat

- Buat proyek di Device Farm. Ikuti instruksi diMembuat proyek di AWS Device Farm, dan kemudian kembali ke halaman ini.

Buat sesi jarak jauh

Console

1. Masuk ke konsol Device Farm di https://console.aws.amazon.com/devicefarm.
2. Pada panel navigasi Device Farm, pilih Pengujian Perangkat Seluler, lalu pilih Proyek.
3. Jika Anda sudah memiliki proyek, pilih dari daftar. Jika tidak, buat proyek dengan mengikuti instruksi diMembuat proyek di AWS Device Farm.
4. Pada tab Akses jarak jauh, pilih Buat sesi akses jarak jauh.
5. Pilih perangkat untuk sesi Anda. Anda dapat memilih dari daftar perangkat yang tersedia atau mencari perangkat menggunakan bilah pencarian di bagian atas daftar.
6. (Opsional) Sertakan aplikasi dan aplikasi tambahan sebagai bagian dari sesi. Ini dapat berupa aplikasi yang baru diunggah, atau aplikasi yang sebelumnya diunggah dalam proyek ini dari 30 hari terakhir (setelah 30 hari, unggahan aplikasi akan kedaluwarsa).
7. Dalam nama Sesi, masukkan nama untuk sesi tersebut.
8. Pilih Konfirmasi dan mulai sesi.

AWS CLI

Catatan: petunjuk ini hanya berfokus pada pembuatan sesi akses jarak jauh. Untuk petunjuk tentang cara mengunggah aplikasi untuk digunakan selama sesi Anda, silakan lihat mengotomatiskan unggahan aplikasi.

Pertama, verifikasi bahwa versi AWS CLI Anda adalah up-to-date dengan mengunduh dan menginstal versi terbaru.

> **Important**
>
> Perintah tertentu yang disebutkan dalam dokumen ini tidak tersedia di AWS CLI versi lama.

Kemudian, Anda dapat menentukan perangkat mana yang ingin Anda uji:

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

Ini akan menampilkan output seperti berikut:

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

Kemudian, Anda dapat membuat sesi akses jarak jauh dengan perangkat ARN pilihan Anda:

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

```
          "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
      ]
  }'
```

Ini akan menampilkan output seperti berikut:

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

Sekarang, secara opsional, kita dapat melakukan polling dan menunggu sesi siap:

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

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

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

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

done
```

Python

Catatan: petunjuk ini hanya berfokus pada pembuatan sesi akses jarak jauh. Untuk petunjuk tentang cara mengunggah aplikasi untuk digunakan selama sesi Anda, silakan lihat mengotomatiskan unggahan aplikasi.

Contoh ini pertama-tama menemukan perangkat Google Pixel yang tersedia di Device Farm, kemudian membuat sesi akses jarak jauh dengannya dan menunggu hingga sesi berjalan.

```
import random
import time
import boto3

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

# 1) Gather all matching devices via paginated ListDevices with filters
filters = [
    {"attribute": "MODEL",        "operator": "CONTAINS", "values": ["Pixel"]},
    {"attribute": "AVAILABILITY", "operator": "EQUALS",   "values": ["AVAILABLE"]},
]

matching_arns = []
next_token = None
while True:
    args = {"filters": filters}
    if next_token:
        args["nextToken"] = next_token
    page = client.list_devices(**args)
    for d in page.get("devices", []):
        matching_arns.append(d["arn"])
    next_token = page.get("nextToken")
    if not next_token:
        break

if not matching_arns:
    raise RuntimeError("No available Google Pixel device found.")

# Randomly select one device from the full matching set
device_arn = random.choice(matching_arns)
print("Selected device ARN:", device_arn)

# 2) Create remote access session and wait until RUNNING
resp = client.create_remote_access_session(
```

```
    projectArn="arn:aws:devicefarm:us-
west-2:111122223333:project:12345678-1111-2222-333-456789abcdef",
    deviceArn=device_arn,
    appArn="arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
  # optional
    configuration={
        "auxiliaryApps": [  # optional
            "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
            "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
        ]
    },
)

session_arn = resp["remoteAccessSession"]["arn"]
print(f"Created Remote Access Session: {session_arn}")

poll_interval = 3
timeout = 600
deadline = time.time() + timeout
terminal_states = ["STOPPING", "COMPLETED"]

while True:
    out = client.get_remote_access_session(arn=session_arn)
    status = out["remoteAccessSession"]["status"]
    print(f"Current status: {status}")

    if status == "RUNNING":
        print(f"Session is ready with status: {status}")
        break
    if status in terminal_states:
        raise RuntimeError(f"Session ended early with status: {status}")
    if time.time() >= deadline:
        raise RuntimeError("Timed out waiting for session to be ready.")
    time.sleep(poll_interval)
```

Java

Catatan: petunjuk ini hanya berfokus pada pembuatan sesi akses jarak jauh. Untuk petunjuk tentang cara mengunggah aplikasi untuk digunakan selama sesi Anda, silakan lihat mengotomatiskan unggahan aplikasi.

Catatan: contoh ini menggunakan AWS SDK for Java v2, dan kompatibel dengan JDK versi 11 dan yang lebih tinggi.

Contoh ini pertama-tama menemukan perangkat Google Pixel yang tersedia di Device Farm, kemudian membuat sesi akses jarak jauh dengannya dan menunggu hingga sesi berjalan.

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.concurrent.ThreadLocalRandom;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.devicefarm.DeviceFarmClient;
import
 software.amazon.awssdk.services.devicefarm.model.CreateRemoteAccessSessionConfiguration;
import
 software.amazon.awssdk.services.devicefarm.model.CreateRemoteAccessSessionRequest;
import
 software.amazon.awssdk.services.devicefarm.model.CreateRemoteAccessSessionResponse;
import software.amazon.awssdk.services.devicefarm.model.Device;
import software.amazon.awssdk.services.devicefarm.model.DeviceFilter;
import software.amazon.awssdk.services.devicefarm.model.DeviceFilterAttribute;
import
 software.amazon.awssdk.services.devicefarm.model.GetRemoteAccessSessionRequest;
import
 software.amazon.awssdk.services.devicefarm.model.GetRemoteAccessSessionResponse;
import software.amazon.awssdk.services.devicefarm.model.ListDevicesRequest;
import software.amazon.awssdk.services.devicefarm.model.ListDevicesResponse;
import software.amazon.awssdk.services.devicefarm.model.RuleOperator;

public class CreateRemoteAccessSession {
  public static void main(String[] args) throws Exception {
    DeviceFarmClient client = DeviceFarmClient.builder()
        .region(Region.US_WEST_2)
        .build();

    String projectArn = "arn:aws:devicefarm:us-
west-2:111122223333:project:12345678-1111-2222-333-456789abcdef";
    String appArn     = "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
    String aux1       = "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
    String aux2       = "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
```

```
    // 1) Gather all matching devices via paginated ListDevices with filters
    List<DeviceFilter> filters = Arrays.asList(
        DeviceFilter.builder()
            .attribute(DeviceFilterAttribute.MODEL)
            .operator(RuleOperator.CONTAINS)
            .values("Pixel")
            .build(),
        DeviceFilter.builder()
            .attribute(DeviceFilterAttribute.AVAILABILITY)
            .operator(RuleOperator.EQUALS)
            .values("AVAILABLE")
            .build()
    );

    List<String> matchingDeviceArns = new ArrayList<>();
    String next = null;
    do {
      ListDevicesResponse page = client.listDevices(
          ListDevicesRequest.builder().filters(filters).nextToken(next).build());
      for (Device d : page.devices()) {
        matchingDeviceArns.add(d.arn());
      }
      next = page.nextToken();
    } while (next != null);

    if (matchingDeviceArns.isEmpty()) {
      throw new RuntimeException("No available Google Pixel device found.");
    }

    // Randomly select one device from the full matching set
    String deviceArn = matchingDeviceArns.get(
        ThreadLocalRandom.current().nextInt(matchingDeviceArns.size()));
    System.out.println("Selected device ARN: " + deviceArn);

    // 2) Create Remote Access session and wait until it is RUNNING
    CreateRemoteAccessSessionConfiguration cfg =
CreateRemoteAccessSessionConfiguration.builder()
        .auxiliaryApps(Arrays.asList(aux1, aux2))
        .build();

    CreateRemoteAccessSessionResponse res = client.createRemoteAccessSession(
        CreateRemoteAccessSessionRequest.builder()
            .projectArn(projectArn)
```

```
            .deviceArn(deviceArn)
            .appArn(appArn)       // optional
            .configuration(cfg)   // optional
            .build());

    String sessionArn = res.remoteAccessSession().arn();
    System.out.println("Created Remote Access Session: " + sessionArn);

    int pollIntervalMs = 3000;
    long timeoutMs = 600_000L;
    long deadline = System.currentTimeMillis() + timeoutMs;

    while (true) {
      GetRemoteAccessSessionResponse get = client.getRemoteAccessSession(
          GetRemoteAccessSessionRequest.builder().arn(sessionArn).build());
      String status = get.remoteAccessSession().statusAsString();
      System.out.println("Current status: " + status);

      if ("RUNNING".equals(status)) {
        System.out.println("Session is ready with status: " + status);
        break;
      }
      if ("STOPPING".equals(status) || "COMPLETED".equals(status)) {
        throw new RuntimeException("Session ended early with status: " + status);
      }
      if (System.currentTimeMillis() >= deadline) {
        throw new RuntimeException("Timed out waiting for session to be ready.");
      }
      Thread.sleep(pollIntervalMs);
    }
  }
}
```

JavaScript

Catatan: petunjuk ini hanya berfokus pada pembuatan sesi akses jarak jauh. Untuk petunjuk tentang cara mengunggah aplikasi untuk digunakan selama sesi Anda, silakan lihat mengotomatiskan unggahan aplikasi.

Catatan: contoh ini menggunakan AWS SDK untuk JavaScript v3.

Contoh ini pertama-tama menemukan perangkat Google Pixel yang tersedia di Device Farm, kemudian membuat sesi akses jarak jauh dengannya dan menunggu hingga sesi berjalan.

```
import {
  DeviceFarmClient,
  ListDevicesCommand,
  CreateRemoteAccessSessionCommand,
  GetRemoteAccessSessionCommand,
} from "@aws-sdk/client-device-farm";

const client = new DeviceFarmClient({ region: "us-west-2" });

// 1) Gather all matching devices via paginated ListDevices with filters
const filters = [
  { attribute: "MODEL",        operator: "CONTAINS", values: ["Pixel"] },
  { attribute: "AVAILABILITY", operator: "EQUALS",   values: ["AVAILABLE"] },
];

let nextToken;
const matching = [];

while (true) {
  const page = await client.send(new ListDevicesCommand({ filters, nextToken }));
  for (const d of page.devices ?? []) {
    matching.push(d.arn);
  }
  nextToken = page.nextToken;
  if (!nextToken) break;
}

if (matching.length === 0) {
  throw new Error("No available Google Pixel device found.");
}

// Randomly select one device from the full matching set
const deviceArn = matching[Math.floor(Math.random() * matching.length)];
console.log("Selected device ARN:", deviceArn);

// 2) Create remote access session and wait until RUNNING
const out = await client.send(new CreateRemoteAccessSessionCommand({
  projectArn: "arn:aws:devicefarm:us-
west-2:111122223333:project:12345678-1111-2222-333-456789abcdef",
  deviceArn,
  appArn: "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
 optional
```

```
  configuration: {
    auxiliaryApps: [ // optional
      "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
      "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
    ],
  },
}));

const sessionArn = out.remoteAccessSession?.arn;
console.log("Created Remote Access Session:", sessionArn);

const pollIntervalMs = 3000;
const timeoutMs = 600000;
const deadline = Date.now() + timeoutMs;

while (true) {
  const get = await client.send(new GetRemoteAccessSessionCommand({ arn:
 sessionArn }));
  const status = get.remoteAccessSession?.status;
  console.log("Current status:", status);

  if (status === "RUNNING") {
    console.log("Session is ready with status:", status);
    break;
  }
  if (status === "STOPPING" || status === "COMPLETED") {
    throw new Error(`Session ended early with status: ${status}`);
  }
  if (Date.now() >= deadline) {
    throw new Error("Timed out waiting for session to be ready.");
  }
  await new Promise((r) => setTimeout(r, pollIntervalMs));
}
```

C#

Catatan: petunjuk ini hanya berfokus pada pembuatan sesi akses jarak jauh. Untuk petunjuk tentang cara mengunggah aplikasi untuk digunakan selama sesi Anda, silakan lihat mengotomatiskan unggahan aplikasi.

Contoh ini pertama-tama menemukan perangkat Google Pixel yang tersedia di Device Farm, kemudian membuat sesi akses jarak jauh dengannya dan menunggu hingga sesi berjalan.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon;
using Amazon.DeviceFarm;
using Amazon.DeviceFarm.Model;

class Program
{
    static async Task Main()
    {
        var client = new AmazonDeviceFarmClient(RegionEndpoint.USWest2);

        // 1) Gather all matching devices via paginated ListDevices with filters
        var filters = new List<DeviceFilter>
        {
            new DeviceFilter { Attribute = DeviceAttribute.MODEL,        Operator =
 RuleOperator.CONTAINS, Values = new List<string>{ "Pixel" } },
            new DeviceFilter { Attribute = DeviceAttribute.AVAILABILITY, Operator =
 RuleOperator.EQUALS,   Values = new List<string>{ "AVAILABLE" } },
        };

        var matchingArns = new List<string>();
        string nextToken = null;

        do
        {
            var list = await client.ListDevicesAsync(new ListDevicesRequest
            {
                Filters = filters,
                NextToken = nextToken
            });

            foreach (var d in list.Devices)
                matchingArns.Add(d.Arn);

            nextToken = list.NextToken;
        }
        while (nextToken != null);
```

```
        if (matchingArns.Count == 0)
            throw new Exception("No available Google Pixel device found.");

        // Randomly select one device from the full matching set
        var rnd = new Random();
        var deviceArn = matchingArns[rnd.Next(matchingArns.Count)];
        Console.WriteLine($"Selected device ARN: {deviceArn}");

        // 2) Create remote access session and wait until RUNNING
        var request = new CreateRemoteAccessSessionRequest
        {
            ProjectArn = "arn:aws:devicefarm:us-
west-2:111122223333:project:12345678-1111-2222-333-456789abcdef",
            DeviceArn  = deviceArn,
            AppArn     = "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
 optional
            Configuration = new CreateRemoteAccessSessionConfiguration
            {
                AuxiliaryApps = new List<string>
                {
                    "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
                    "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
                }
            }
        };

        request.Configuration.AuxiliaryApps.RemoveAll(string.IsNullOrWhiteSpace);

        var response = await client.CreateRemoteAccessSessionAsync(request);
        var sessionArn = response.RemoteAccessSession.Arn;
        Console.WriteLine($"Created Remote Access Session: {sessionArn}");

        var pollIntervalMs = 3000;
        var timeoutMs = 600000;
        var deadline = DateTime.UtcNow.AddMilliseconds(timeoutMs);

        while (true)
        {
            var get = await client.GetRemoteAccessSessionAsync(new
 GetRemoteAccessSessionRequest { Arn = sessionArn });
            var status = get.RemoteAccessSession.Status.Value;
```

```
            Console.WriteLine($"Current status: {status}");

            if (status == "RUNNING")
            {
                Console.WriteLine($"Session is ready with status: {status}");
                break;
            }
            if (status == "STOPPING" || status == "COMPLETED")
            {
                throw new Exception($"Session ended early with status: {status}");
            }
            if (DateTime.UtcNow >= deadline)
            {
                throw new TimeoutException("Timed out waiting for session to be
 ready.");
            }

            await Task.Delay(pollIntervalMs);
        }
    }
}
```

Ruby

Catatan: petunjuk ini hanya berfokus pada pembuatan sesi akses jarak jauh. Untuk petunjuk tentang cara mengunggah aplikasi untuk digunakan selama sesi Anda, silakan lihat mengotomatiskan unggahan aplikasi.

Contoh ini pertama-tama menemukan perangkat Google Pixel yang tersedia di Device Farm, kemudian membuat sesi akses jarak jauh dengannya dan menunggu hingga sesi berjalan.

```
require 'aws-sdk-devicefarm'

client = Aws::DeviceFarm::Client.new(region: 'us-west-2')

# 1) Gather all matching devices via paginated ListDevices with filters
filters = [
  { attribute: 'MODEL',        operator: 'CONTAINS', values: ['Pixel'] },
  { attribute: 'AVAILABILITY', operator: 'EQUALS',   values: ['AVAILABLE'] },
]

matching_arns = []
next_token = nil
```

```
loop do
  resp = client.list_devices(filters: filters, next_token: next_token)
  resp.devices&.each { |d| matching_arns << d.arn }
  next_token = resp.next_token
  break unless next_token
end

abort "No available Google Pixel device found." if matching_arns.empty?

# Randomly select one device from the full matching set
device_arn = matching_arns.sample
puts "Selected device ARN: #{device_arn}"

# 2) Create remote access session and wait until RUNNING
resp = client.create_remote_access_session(
  project_arn: "arn:aws:devicefarm:us-
west-2:111122223333:project:12345678-1111-2222-333-456789abcdef",
  device_arn:  device_arn,
  app_arn:     "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
 # optional
  configuration: {
    auxiliary_apps: [ # optional
      "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
      "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
    ].compact
  }
)

session_arn = resp.remote_access_session.arn
puts "Created Remote Access Session: #{session_arn}"

poll_interval = 3
timeout = 600
deadline = Time.now + timeout
terminal = %w[STOPPING COMPLETED]

loop do
  get = client.get_remote_access_session(arn: session_arn)
  status = get.remote_access_session.status
  puts "Current status: #{status}"
```

```
  if status == 'RUNNING'
    puts "Session is ready with status: #{status}"
    break
  end

  abort "Session ended early with status: #{status}" if terminal.include?(status)
  abort "Timed out waiting for session to be ready." if Time.now >= deadline
  sleep poll_interval
end
```

Langkah selanjutnya

Device Farm memulai sesi segera setelah perangkat dan infrastruktur yang diminta tersedia, biasanya dalam beberapa menit. Kotak dialog Device Required muncul sampai sesi dimulai. Untuk membatalkan permintaan sesi, pilih Batalkan permintaan.

Jika perangkat yang dipilih tidak tersedia atau sibuk, status sesi ditampilkan sebagai Perangkat Tertunda, yang menunjukkan bahwa Anda mungkin perlu menunggu beberapa saat sebelum perangkat tersedia untuk pengujian.

Jika akun Anda telah mencapai batas konkurensi untuk perangkat terukur atau tidak terukur publik, status sesi ditampilkan sebagai Konkurensi Tertunda. Untuk slot perangkat yang tidak diukur, Anda dapat meningkatkan konkurensi dengan membeli lebih banyak slot perangkat. Untuk pay-as-you-go perangkat terukur, silakan hubungi AWS melalui tiket dukungan untuk meminta peningkatan kuota layanan.

Saat penyiapan sesi dimulai, pertama kali menampilkan status In-Progress, lalu status Connecting saat browser web lokal Anda mencoba membuka koneksi jarak jauh ke perangkat.

Setelah sesi dimulai, jika Anda harus menutup browser atau tab browser tanpa menghentikan sesi atau jika koneksi antara browser dan internet terputus, sesi tetap aktif selama lima menit. Setelah itu, Device Farm mengakhiri sesi. Akun Anda dikenakan biaya untuk waktu idle.

Setelah sesi dimulai, Anda dapat berinteraksi dengan perangkat di browser web, atau menguji perangkat menggunakan Appium.

Menggunakan sesi akses jarak jauh di AWS Device Farm

Untuk informasi tentang melakukan pengujian interaktif aplikasi Android dan iOS melalui sesi akses jarak jauh, lihatSesi.

Prasyarat

- Buat sesi. Ikuti instruksi diMembuat sesi, dan kemudian kembali ke halaman ini.

Menggunakan sesi di konsol Device Farm

Segera setelah perangkat yang Anda minta untuk sesi akses jarak jauh tersedia, konsol akan menampilkan layar perangkat. Sesi ini memiliki panjang maksimum 150 menit. Waktu yang tersisa dalam sesi muncul di bidang Waktu Kiri di dekat nama perangkat.

Menginstal aplikasi

Untuk menginstal aplikasi pada perangkat sesi, di Instal aplikasi, pilih Pilih File, lalu pilih file.apk (Android) atau file.ipa (iOS) yang ingin Anda instal. Aplikasi yang Anda jalankan dalam sesi akses jarak jauh tidak memerlukan instrumentasi pengujian atau penyediaan apa pun.

> **Note**
>
> Saat Anda mengunggah aplikasi, terkadang ada penundaan sebelum aplikasi tersedia. Pesan konfirmasi akan muncul untuk memberi tahu Anda apakah aplikasi berhasil diinstal atau tidak.

Mengontrol perangkat

Anda dapat berinteraksi dengan perangkat yang ditampilkan di konsol seperti halnya perangkat fisik yang sebenarnya, dengan menggunakan mouse Anda atau perangkat yang sebanding untuk sentuhan dan keyboard di layar perangkat. Untuk perangkat Android, ada tombol di kontrol Tampilan yang berfungsi seperti tombol Beranda dan Kembali pada perangkat Android. Untuk perangkat iOS, ada tombol Beranda yang berfungsi seperti tombol beranda di perangkat iOS. Anda juga dapat beralih antar aplikasi yang berjalan di perangkat dengan memilih Aplikasi Terbaru.

Beralih antara mode potret dan lanskap

Anda juga dapat beralih antara mode potret (vertikal) dan lanskap (horizontal) untuk perangkat yang Anda gunakan.

Langkah selanjutnya

Device Farm melanjutkan sesi hingga Anda menghentikannya secara manual atau batas waktu 150 menit tercapai. Untuk mengakhiri sesi, pilih Stop Session. Setelah sesi berhenti, Anda dapat mengakses video yang diambil dan log yang dihasilkan. Untuk informasi selengkapnya, lihat Mengambil hasil sesi.

Kiat dan trik

Anda mungkin mengalami masalah kinerja dengan sesi akses jarak jauh di beberapa AWS Wilayah. Hal ini disebabkan, sebagian, latensi di beberapa Wilayah. Jika Anda mengalami masalah kinerja, berikan sesi akses jarak jauh kesempatan untuk mengejar ketinggalan sebelum Anda berinteraksi dengan aplikasi lagi.

Mengambil hasil sesi akses jarak jauh di AWS Device Farm

Untuk informasi tentang sesi, lihatSesi.

- Prasyarat
- Melihat detail sesi
- Mengunduh video sesi atau log

Prasyarat

- Selesaikan sesi. Ikuti instruksi diMenggunakan sesi akses jarak jauh di AWS Device Farm, dan kemudian kembali ke halaman ini.

Melihat detail sesi

Saat sesi akses jarak jauh berakhir, konsol Device Farm menampilkan tabel yang berisi detail tentang aktivitas selama sesi berlangsung. Untuk informasi selengkapnya, lihat Menganalisis Informasi Log.

Untuk kembali ke detail sesi di lain waktu:

1. Pada panel navigasi Device Farm, pilih Pengujian Perangkat Seluler, lalu pilih Proyek.
2. Pilih proyek yang berisi sesi.
3. Pilih Akses jarak jauh, lalu pilih sesi yang ingin Anda tinjau dari daftar.

Mengunduh video sesi atau log

Saat sesi akses jarak jauh berakhir, konsol Device Farm menyediakan akses ke rekaman video sesi dan log aktivitas. Dalam hasil sesi, pilih tab File untuk daftar tautan ke video sesi dan log. Anda dapat melihat file-file ini di browser atau menyimpannya secara lokal.

Pengujian appium di AWS Device Farm

Selama sesi akses jarak jauh, Anda dapat menjalankan pengujian Appium dari lingkungan lokal Anda, menargetkan perangkat sesi menggunakan titik akhir Appium yang dikelola. Dengan titik akhir Appium, Anda dapat mengembangkan, menguji, dan mengeksekusi kode Appium dengan umpan balik cepat dan iterasi cepat. Pendekatan sisi klien untuk pengujian ini menawarkan fleksibilitas untuk terhubung ke perangkat Device Farm dari lingkungan klien Appium pilihan Anda.

Untuk melengkapi pengujian sisi klien, Device Farm juga mendukung menjalankan pengujian pada infrastruktur yang dikelola oleh layanan, yang disebut eksekusi sisi server. Dalam pendekatan ini, Anda dapat mengunggah aplikasi dan pengujian ke layanan, lalu menjalankan pengujian secara paralel di beberapa perangkat menggunakan host pengujian yang dikelola layanan. Pendekatan ini berskala baik untuk pengujian pada banyak perangkat secara independen, serta pengujian dari konteks CI/CD pipa.

Untuk mempelajari lebih lanjut tentang eksekusi sisi server, silakan lihat. Kerangka kerja uji dan pengujian bawaan

Topik

Apa itu titik akhir Appium?

Appium adalah kerangka pengujian perangkat lunak open-source yang populer untuk menguji aplikasi web asli, hibrida, dan seluler pada perangkat yang berbeda, termasuk ponsel dan tablet, untuk iOS dan Android. Ini memungkinkan pengembang dan insinyur QA (Quality Assurance) untuk menulis skrip yang dapat mengontrol perangkat dari jarak jauh, mensimulasikan interaksi pengguna, dan memverifikasi bahwa aplikasi yang diuji berperilaku seperti yang diharapkan. Appium berinteraksi dengan aplikasi dari perspektif pengguna akhir, memungkinkan penguji untuk mengembangkan pengujian yang mensimulasikan bagaimana pengguna nyata akan menggunakan aplikasi untuk pengujian mereka.

Appium dibangun pada model client-server, di mana klien lokal meminta server Appium (lokal atau jarak jauh) untuk memerintahkan perangkat atas nama mereka. Server Appium mengelola driver untuk berkomunikasi dengan perangkat, seperti UIAutomator2 driver untuk Android atau driver untuk iOSXCUITest . Semua perintah mengikuti WebDriver standar W3C untuk cara mengontrol perangkat.

Titik akhir Appium Device Farm memperlihatkan URL server Appium untuk perangkat dalam sesi akses jarak jauh Anda. URL titik akhir Appium akan spesifik untuk perangkat tersebut dalam sesi tersebut, dan tetap valid selama durasi sesi, memungkinkan Anda untuk melakukan iterasi pada perangkat yang sama tanpa waktu penyiapan tambahan. Untuk informasi lebih lanjut tentang Remote Access, silakan lihatAkses jarak jauh.

Memulai dengan pengujian Appium

Untuk sebagian besar pengguna Appium, menggunakan Device Farm untuk pengujian Appium hanya memerlukan sedikit perubahan pada konfigurasi pengujian yang ada.

Pada tingkat tinggi, ada tiga langkah untuk menggunakan Device Farm untuk pengujian Appium sisi klien:

1. Pertama, Anda perlu membuat sesi akses jarak jauh untuk menguji perangkat Device Farm. Anda dapat menyertakan aplikasi sebagai bagian dari permintaan akses jarak jauh, atau menginstal aplikasi setelah sesi dimulai.
2. Setelah sesi berjalan, Anda dapat menyalin URL titik akhir Appium, dan menggunakannya baik melalui alat yang berdiri sendiri (seperti Appium Inspector) atau dari kode pengujian Appium di IDE Anda. URL akan berlaku selama sesi akses jarak jauh.
3. Dan akhirnya, setelah pengujian Appium Anda dimulai, Anda dapat meninjau log server Appium Anda secara langsung selama eksekusi pengujian di samping aliran video perangkat Anda.

Berinteraksi dengan perangkat menggunakan Appium

Setelah Anda membuat sesi akses jarak jauh, perangkat akan tersedia untuk pengujian Appium. Untuk seluruh durasi sesi akses jarak jauh, Anda dapat menjalankan sesi Appium sebanyak yang Anda inginkan di perangkat, tanpa batasan pada klien yang Anda gunakan. Misalnya, Anda dapat memulai dengan menjalankan pengujian menggunakan kode Appium lokal dari IDE, lalu beralih menggunakan Appium Inspector untuk memecahkan masalah apa pun yang Anda temui. Sesi dapat berlangsung hingga 150 menit, namun, jika tidak ada aktivitas selama lebih dari 5 menit (baik melalui konsol interaktif atau melalui titik akhir Appium), sesi akan habis waktu.

Menggunakan Aplikasi untuk pengujian dengan sesi Appium Anda

Device Farm memungkinkan Anda menggunakan aplikasi sebagai bagian dari permintaan pembuatan sesi akses jarak jauh, atau untuk menginstal aplikasi selama sesi akses jarak jauh itu sendiri. Aplikasi ini secara otomatis diinstal ke perangkat yang sedang diuji dan disuntikkan sebagai kemampuan default untuk permintaan sesi Appium apa pun. Saat Anda membuat sesi akses jarak jauh, Anda memiliki opsi untuk meneruskan ARN aplikasi, yang akan digunakan secara default sebagai `appium:app` kemampuan untuk semua sesi Appium berikutnya, serta aplikasi tambahan ARNs, yang akan digunakan sebagai kemampuan. `appium:otherApps`

Misalnya, jika Anda membuat sesi akses jarak jauh menggunakan aplikasi `com.aws.devicefarm.sample` sebagai aplikasi Anda, dan `com.aws.devicefarm.other.sample` sebagai salah satu aplikasi tambahan, maka ketika Anda pergi untuk membuat sesi Appium, itu akan memiliki kemampuan yang mirip dengan yang berikut:

```
{
    "value":
    {
        "sessionId": "abcdef123456-1234-5678-abcd-abcdef123456",
        "capabilities":
        {
            "app": "/tmp/com.aws.devicefarm.sample.apk",
            "otherApps": "[\"/tmp/com.aws.devicefarm.other.sample.apk\"]",
            ...
        }
    }
}
```

Selama sesi, Anda dapat menginstal aplikasi tambahan (baik di dalam konsol, atau menggunakan `InstallToRemoteAccessSession`API). Ini akan menggantikan aplikasi yang ada yang sebelumnya digunakan sebagai `appium:app` kemampuan. Jika aplikasi yang sebelumnya digunakan memiliki nama paket yang berbeda, mereka akan tetap berada di perangkat dan digunakan sebagai bagian dari `appium:otherApps` kemampuan.

Misalnya, jika Anda awalnya menggunakan aplikasi `com.aws.devicefarm.sample` saat membuat sesi akses jarak jauh, tetapi kemudian menginstal aplikasi baru yang diberi nama `com.aws.devicefarm.other.sample` selama sesi, maka sesi Appium Anda akan memiliki kemampuan yang mirip dengan yang berikut ini:

```
{
```

```
    "value":
    {
        "sessionId": "abcdef123456-1234-5678-abcd-abcdef123456",
        "capabilities":
        {
            "app": "/tmp/com.aws.devicefarm.other.sample.apk",
            "otherApps": "[\"/tmp/com.aws.devicefarm.sample.apk\"]",
            ...
        }
    }
}
```

Jika mau, Anda dapat secara eksplisit menentukan kemampuan untuk aplikasi menggunakan nama aplikasi (masing-masing menggunakan `appium:bundleId` kemampuan `appium:appPackage` atau untuk Android dan iOS).

Jika Anda menguji aplikasi web, tentukan `browserName` kemampuan untuk permintaan pembuatan sesi Appium Anda. `Chrome`Browser tersedia di semua perangkat Android, dan `Safari` browser tersedia di semua perangkat iOS.

Device Farm tidak mendukung penerusan URL jarak jauh atau jalur sistem file lokal `appium:app` selama sesi akses jarak jauh. Unggah aplikasi ke Device Farm dan sertakan dalam sesi sebagai gantinya.

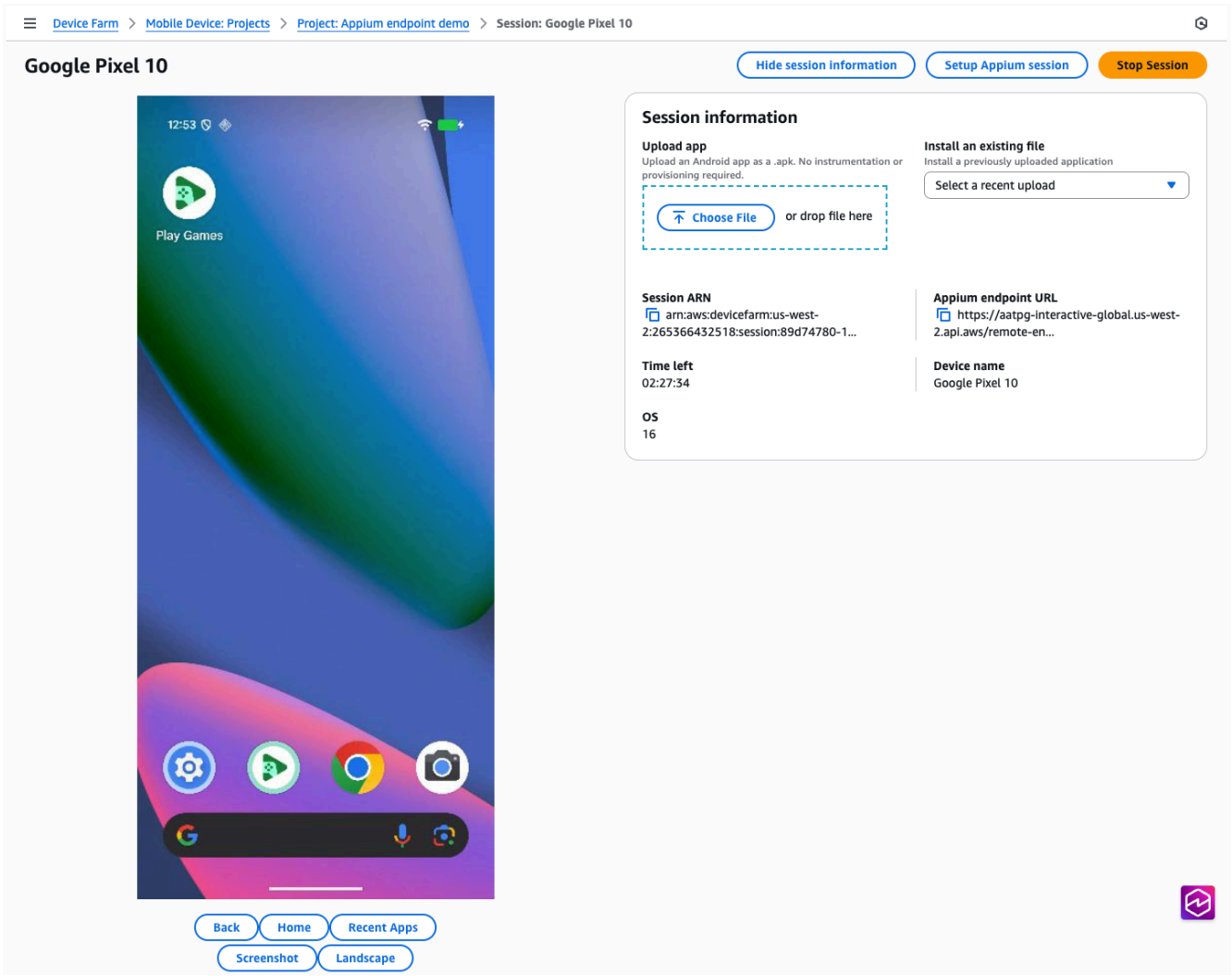
> **Note**
>
> Untuk informasi selengkapnya tentang mengunggah aplikasi secara otomatis sebagai bagian dari sesi akses jarak jauh Anda, lihat mengotomatiskan unggahan aplikasi.

Cara menggunakan titik akhir Appium

Berikut adalah langkah-langkah untuk mengakses titik akhir Appium sesi dari konsol, konsol AWS CLI, dan. AWS SDKs Langkah-langkah ini mencakup cara memulai menjalankan pengujian menggunakan berbagai kerangka kerja pengujian klien Appium:

Console

1. Buka halaman sesi akses jarak jauh Anda di browser web Anda:



2. Untuk menjalankan sesi menggunakan Appium Inspector, lakukan hal berikut:

   a. Klik tombolnya Pengaturan sesi Appium

   b. Ikuti petunjuk di halaman untuk cara memulai sesi menggunakan Appium Inspector.

3. Untuk menjalankan pengujian Appium dari IDE lokal Anda, lakukan hal berikut:

   a. Klik ikon “salin” di sebelah teks URL titik akhir Appium

   b. Tempel URL ini ke kode Appium lokal Anda di mana pun Anda saat ini menentukan alamat jarak jauh atau pelaksana perintah Anda. Untuk contoh khusus bahasa, silakan klik salah satu tab di jendela contoh ini untuk bahasa pilihan Anda.

AWS CLI

Pertama, verifikasi bahwa versi AWS CLI Anda adalah up-to-date dengan mengunduh dan menginstal versi terbaru.

> **Important**
>
> Bidang titik akhir Appium tidak tersedia di AWS CLI versi lama.

Setelah sesi Anda aktif dan berjalan, URL titik akhir Appium akan tersedia melalui `remoteDriverEndpoint` bidang bernama sebagai respons terhadap panggilan ke API: `GetRemoteAccessSession`

```
$ aws devicefarm get-remote-access-session \
    --arn "arn:aws:devicefarm:us-west-2:123456789876:session:abcdef123456-1234-5678-
abcd-abcdef123456/abcdef123456-1234-5678-abcd-abcdef123456/00000"
```

Ini akan menampilkan output seperti berikut:

```
{
    "remoteAccessSession": {
        "arn": "arn:aws:devicefarm:us-
west-2:111122223333:session:abcdef123456-1234-5678-abcd-abcdef123456/
abcdef123456-1234-5678-abcd-abcdef123456/00000",
        "name": "Google Pixel 8",
        "status": "RUNNING",
        "endpoints": {
            "remoteDriverEndpoint": "https://devicefarm-interactive-global.us-
west-2.api.aws/remote-endpoint/ABCD1234...",
        ...
}
```

Anda dapat menggunakan URL ini dalam kode Appium lokal Anda di mana pun Anda saat ini menentukan alamat jarak jauh atau pelaksana perintah Anda. Untuk contoh khusus bahasa, silakan klik salah satu tab di jendela contoh ini untuk bahasa pilihan Anda.

Untuk contoh cara berinteraksi dengan titik akhir langsung dari baris perintah, Anda dapat menggunakan alat baris perintah curl untuk memanggil titik akhir secara langsung: WebDriver

```
$ curl "https://devicefarm-interactive-global.us-west-2.api.aws/remote-endpoint/
ABCD1234.../status"
```

Ini akan menampilkan output seperti berikut:

```
{
    "value":
    {
        "ready": true,
        "message": "The server is ready to accept new connections",
        "build":
        {
            "version": "2.5.1"
        }
    }
}
```

Python

Setelah sesi Anda aktif dan berjalan, URL titik akhir Appium akan tersedia melalui `remoteDriverEndpoint` bidang bernama sebagai respons terhadap panggilan ke API: `GetRemoteAccessSession`

```
# To get the URL
import sys
import boto3
from botocore.exceptions import ClientError

def get_appium_endpoint() -> str:
    session_arn = "arn:aws:devicefarm:us-
west-2:111122223333:session:abcdef123456-1234-5678-abcd-abcdef123456/
abcdef123456-1234-5678-abcd-abcdef123456/00000"
    device_farm_client = boto3.client("devicefarm", region_name="us-west-2")

    try:
        resp = device_farm_client.get_remote_access_session(arn=session_arn)
    except ClientError as exc:
        sys.exit(f"Failed to call Device Farm: {exc}")

    remote_access_session = resp.get("remoteAccessSession", {})
    endpoints = remote_access_session.get("endpoints", {})
    endpoint = endpoints.get("remoteDriverEndpoint")
```

```
    if not endpoint:
        sys.exit("Device Farm response did not include
 endpoints.remoteDriverEndpoint")

    return endpoint

# To use the URL
from appium import webdriver
from appium.options.android import UiAutomator2Options

opts = UiAutomator2Options()
driver = webdriver.Remote(get_appium_endpoint(), options=opts)
# ...
driver.quit()
```

Java

Catatan: contoh ini menggunakan AWS SDK for Java v2, dan kompatibel dengan JDK versi 11 dan yang lebih tinggi.

Setelah sesi Anda aktif dan berjalan, URL titik akhir Appium akan tersedia melalui `remoteDriverEndpoint` bidang bernama sebagai respons terhadap panggilan ke API: GetRemoteAccessSession

```
// To get the URL
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.devicefarm.DeviceFarmClient;
import
 software.amazon.awssdk.services.devicefarm.model.GetRemoteAccessSessionRequest;
import
 software.amazon.awssdk.services.devicefarm.model.GetRemoteAccessSessionResponse;

public class AppiumEndpointBuilder {
    public static String getAppiumEndpoint() throws Exception {
        String session_arn = "arn:aws:devicefarm:us-
west-2:111122223333:session:abcdef123456-1234-5678-abcd-abcdef123456/
abcdef123456-1234-5678-abcd-abcdef123456/00000";

        try (DeviceFarmClient client = DeviceFarmClient.builder()
                .region(Region.US_WEST_2)
                .credentialsProvider(DefaultCredentialsProvider.create())
```

```
                .build()) {

            GetRemoteAccessSessionResponse resp = client.getRemoteAccessSession(
                    GetRemoteAccessSessionRequest.builder().arn(session_arn).build()
            );

            String endpoint =
 resp.remoteAccessSession().endpoints().remoteDriverEndpoint();
            if (endpoint == null || endpoint.isEmpty()) {
                throw new IllegalStateException("remoteDriverEndpoint missing from
 response");
            }
            return endpoint;
        }
    }
}

// To use the URL
import io.appium.java_client.android.AndroidDriver;
import io.appium.java_client.android.options.UiAutomator2Options;

import java.net.URL;

public class ExampleTest {
    public static void main(String[] args) throws Exception {
        String endpoint = AppiumEndpointBuilder.getAppiumEndpoint();
        UiAutomator2Options options = new UiAutomator2Options();
        AndroidDriver driver = new AndroidDriver(new URL(endpoint), options);

        try {
            // ... your test ...
        } finally {
            driver.quit();
        }
    }
}
```

JavaScript

Catatan: contoh ini menggunakan AWS SDK untuk JavaScript v3 dan WebDriverIO v8+ menggunakan Node 18+.

Setelah sesi Anda aktif dan berjalan, URL titik akhir Appium akan tersedia melalui `remoteDriverEndpoint` bidang bernama sebagai respons terhadap panggilan ke API: [GetRemoteAccessSession]

```
// To get the URL
import { DeviceFarmClient, GetRemoteAccessSessionCommand } from "@aws-sdk/client-
device-farm";

export async function getAppiumEndpoint() {
  const sessionArn = "arn:aws:devicefarm:us-
west-2:111122223333:session:abcdef123456-1234-5678-abcd-abcdef123456/
abcdef123456-1234-5678-abcd-abcdef123456/00000";

  const client = new DeviceFarmClient({ region: "us-west-2" });
  const resp = await client.send(new GetRemoteAccessSessionCommand({ arn:
 sessionArn }));

  const endpoint = resp?.remoteAccessSession?.endpoints?.remoteDriverEndpoint;
  if (!endpoint) throw new Error("remoteDriverEndpoint missing from response");
  return endpoint;
}

// To use the URL with WebdriverIO
import { remote } from "webdriverio";

(async () => {
  const endpoint = await getAppiumEndpoint();
  const u = new URL(endpoint);

  const driver = await remote({
    protocol: u.protocol.replace(":", ""),
    hostname: u.hostname,
    port: u.port ? Number(u.port) : (u.protocol === "https:" ? 443 : 80),
    path: u.pathname + u.search,
    capabilities: {
      platformName: "Android",
      "appium:automationName": "UiAutomator2",
      // ...other caps...
    },
  });

  try {
    // ... your test ...
```

```
  } finally {
    await driver.deleteSession();
  }
})();
```

C#

Setelah sesi Anda aktif dan berjalan, URL titik akhir Appium akan tersedia melalui `remoteDriverEndpoint` bidang bernama sebagai respons terhadap panggilan ke API: [GetRemoteAccessSession]

```
// To get the URL
using System;
using System.Threading.Tasks;
using Amazon;
using Amazon.DeviceFarm;
using Amazon.DeviceFarm.Model;

public static class AppiumEndpointBuilder
{
    public static async Task<string> GetAppiumEndpointAsync()
    {
        var sessionArn = "arn:aws:devicefarm:us-
west-2:111122223333:session:abcdef123456-1234-5678-abcd-abcdef123456/
abcdef123456-1234-5678-abcd-abcdef123456/00000";

        var config = new AmazonDeviceFarmConfig
        {
            RegionEndpoint = RegionEndpoint.USWest2
        };
        using var client = new AmazonDeviceFarmClient(config);

        var resp = await client.GetRemoteAccessSessionAsync(new
 GetRemoteAccessSessionRequest { Arn = sessionArn });
        var endpoint = resp?.RemoteAccessSession?.Endpoints?.RemoteDriverEndpoint;

        if (string.IsNullOrWhiteSpace(endpoint))
            throw new InvalidOperationException("RemoteDriverEndpoint missing from
 response");

        return endpoint;
    }
}
```

```
// To use the URL
using OpenQA.Selenium.Appium;
using OpenQA.Selenium.Appium.Android;

class Example
{
    static async Task Main()
    {
        var endpoint = await AppiumEndpointBuilder.GetAppiumEndpointAsync();

        var options = new AppiumOptions();
        options.PlatformName = "Android";
        options.AutomationName = "UiAutomator2";

        using var driver = new AndroidDriver(new Uri(endpoint), options);
        try
        {
            // ... your test ...
        }
        finally
        {
            driver.Quit();
        }
    }
}
```

Ruby

Setelah sesi Anda aktif dan berjalan, URL titik akhir Appium akan tersedia melalui `remoteDriverEndpoint` bidang bernama sebagai respons terhadap panggilan ke API: GetRemoteAccessSession

```
# To get the URL
require 'aws-sdk-devicefarm'

def get_appium_endpoint
  session_arn = "arn:aws:devicefarm:us-
west-2:111122223333:session:abcdef123456-1234-5678-abcd-abcdef123456/
abcdef123456-1234-5678-abcd-abcdef123456/00000"

  client = Aws::DeviceFarm::Client.new(region: 'us-west-2')
  resp = client.get_remote_access_session(arn: session_arn)
```

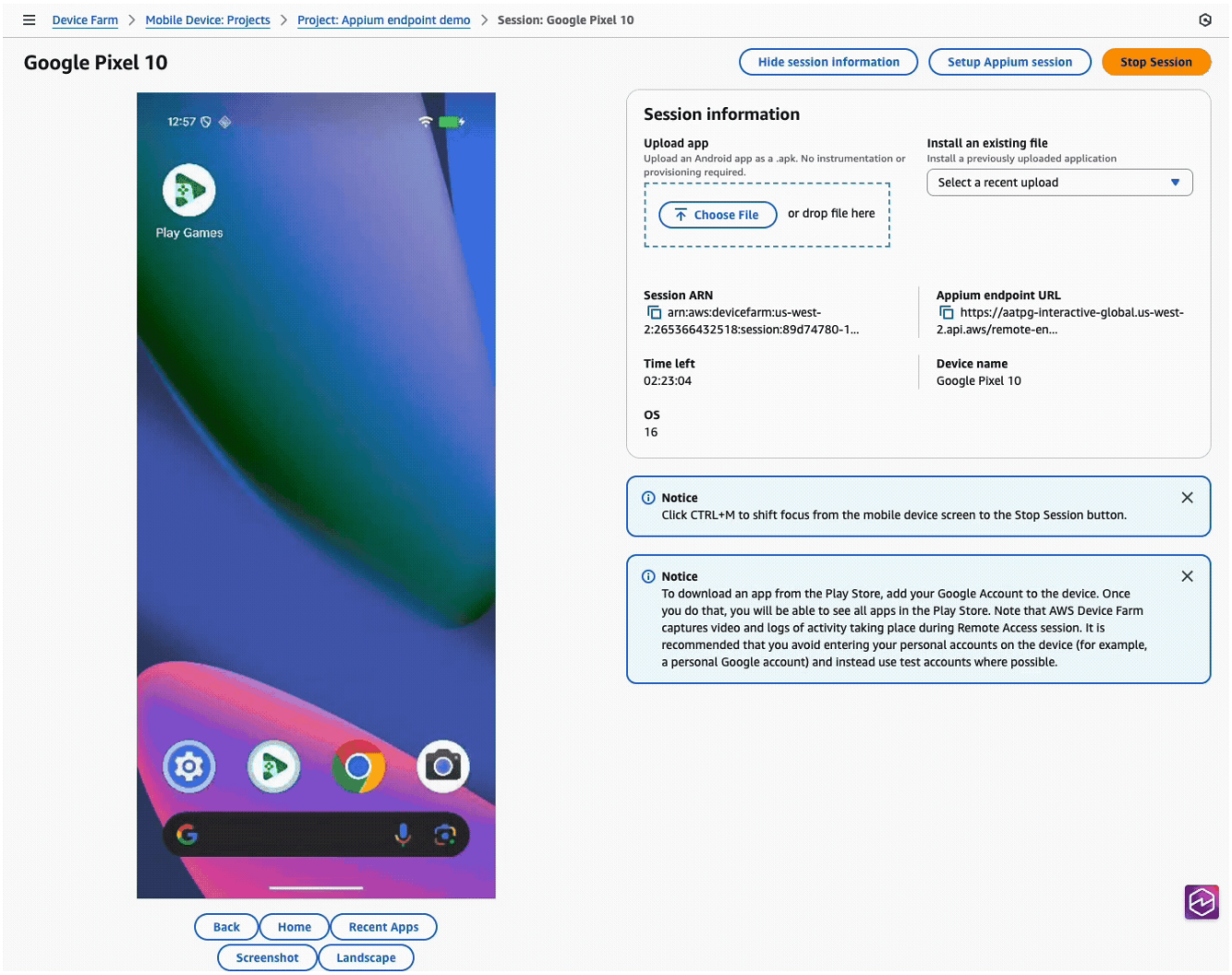
```
  endpoint = resp.remote_access_session.endpoints.remote_driver_endpoint
  raise "remote_driver_endpoint missing from response" if endpoint.nil? ||
 endpoint.empty?
  endpoint
end

# To use the URL
require 'appium_lib_core'

endpoint = get_appium_endpoint
opts = {
  server_url: endpoint,
  capabilities: {
    'platformName' => 'Android',
    'appium:automationName' => 'UiAutomator2'
  }
}

driver = Appium::Core.for(opts).start_driver
begin
  # ... your test ...
ensure
  driver.quit
end
```

Meninjau log server Appium Anda

Setelah memulai sesi Appium, Anda dapat melihat log server Appium secara langsung di konsol Device Farm, atau mengunduhnya setelah sesi akses jarak jauh berakhir. Berikut adalah instruksi untuk melakukannya:

Console

1. Di konsol Device Farm, buka sesi akses jarak jauh untuk perangkat Anda.
2. Memulai sesi titik akhir Appium dengan perangkat dari IDE lokal atau Appium Inspector
3. Kemudian, log server Appium akan muncul di samping perangkat di halaman sesi akses jarak jauh, dengan “informasi sesi” tersedia di bagian bawah halaman di bawah perangkat:



AWS CLI

Catatan: contoh ini menggunakan alat baris perintah *curl* untuk menarik log dari Device Farm.

Selama atau setelah sesi, Anda dapat menggunakan `ListArtifacts`API Device Farm untuk mengunduh log server Appium.

```
$ aws devicefarm list-artifacts \
  --type FILE \
  --arn arn:aws:devicefarm:us-
west-2:111122223333:session:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-45678
```

Ini akan menampilkan output seperti berikut selama sesi:

```
{
```

```
    "artifacts": [
        {
            "arn": "arn:aws:devicefarm:us-
west-2:111122223333:artifact:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-4567
            "name": "AppiumServerLogOutput",
            "type": "APPIUM_SERVER_LOG_OUTPUT",
            "extension": "",
            "url": "https://prod-us-west-2-results.s3.dualstack.us-
west-2.amazonaws.com/111122223333/12345678..."
        }
    ]
}
```

Dan berikut ini setelah sesi selesai:

```
{
    "artifacts": [
        {
            "arn": "arn:aws:devicefarm:us-
west-2:111122223333:artifact:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-4567
            "name": "Appium Server Output",
            "type": "APPIUM_SERVER_OUTPUT",
            "extension": "log",
            "url": "https://prod-us-west-2-results.s3.dualstack.us-
west-2.amazonaws.com/111122223333/12345678..."
        }
    ]
}
```

```
$ curl "https://prod-us-west-2-results.s3.dualstack.us-
west-2.amazonaws.com/111122223333/12345678..."
```

Ini akan menampilkan output seperti berikut:

```
info Appium Welcome to Appium v2.5.4
info Appium Non-default server args:
info Appium { address: '127.0.0.1',
info Appium   allowInsecure:
info Appium    [ 'execute_driver_script',
info Appium      'session_discovery',
info Appium      'perf_record',
info Appium      'adb_shell',
```

```
info Appium      'chromedriver_autodownload',
info Appium      'get_server_logs' ],
info Appium   keepAliveTimeout: 0,
info Appium   logNoColors: true,
info Appium   logTimestamp: true,
info Appium   longStacktrace: true,
info Appium   sessionOverride: true,
info Appium   strictCaps: true,
info Appium   useDrivers: [ 'uiautomator' ] }
```

Python

Catatan: contoh ini menggunakan *`requests`* paket pihak ketiga untuk mengunduh log, serta AWS SDK untuk Python*`boto3`*.

Selama atau setelah sesi, Anda dapat menggunakan `ListArtifacts`API Device Farm untuk mengambil URL log server Appium, lalu mengunduhnya.

```
import pathlib
import requests
import boto3

def download_appium_log():
    session_arn = "arn:aws:devicefarm:us-
west-2:111122223333:session:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-45678
    client = boto3.client("devicefarm", region_name="us-west-2")

    # 1) List artifacts for the session (FILE artifacts), handling pagination
    artifacts = []
    token = None
    while True:
        kwargs = {"arn": session_arn, "type": "FILE"}
        if token:
            kwargs["nextToken"] = token
        resp = client.list_artifacts(**kwargs)
        artifacts.extend(resp.get("artifacts", []))
        token = resp.get("nextToken")
        if not token:
            break

    if not artifacts:
        raise RuntimeError("No artifacts found in this session")
```

```
    # Filter strictly to Appium server logs
    allowed = {"APPIUM_SERVER_OUTPUT", "APPIUM_SERVER_LOG_OUTPUT"}
    filtered = [a for a in artifacts if a.get("type") in allowed]
    if not filtered:
        raise RuntimeError("No Appium server log artifacts found (expected
 APPIUM_SERVER_OUTPUT or APPIUM_SERVER_LOG_OUTPUT)")

    # Prefer the final 'OUTPUT' log, else the live 'LOG_OUTPUT'
    chosen = (next((a for a in filtered if a.get("type") == "APPIUM_SERVER_OUTPUT"),
 None)
              or next((a for a in filtered if a.get("type") ==
 "APPIUM_SERVER_LOG_OUTPUT"), None))

    url = chosen["url"]
    ext = chosen.get("extension") or "log"
    out = pathlib.Path(f"./appium_server_log.{ext}")

    # 2) Download the artifact
    with requests.get(url, stream=True) as r:
        r.raise_for_status()
        with open(out, "wb") as fh:
            for chunk in r.iter_content(chunk_size=1024 * 1024):
                if chunk:
                    fh.write(chunk)

    print(f"Saved Appium server log to: {out.resolve()}")

download_appium_log()
```

Ini akan menampilkan output seperti berikut:

```
info Appium Welcome to Appium v2.5.4
info Appium Non-default server args:
info Appium { address: '127.0.0.1', allowInsecure: [ 'execute_driver_script', ... ],
 useDrivers: [ 'uiautomator' ] }
```

Java

Catatan: contoh ini menggunakan AWS SDK for Java v2 *`HttpClient`* dan untuk mengunduh log, dan kompatibel dengan JDK versi 11 dan yang lebih tinggi.

Selama atau setelah sesi, Anda dapat menggunakan `ListArtifacts`API Device Farm untuk mengambil URL log server Appium, lalu mengunduhnya.

```
import java.io.IOException;
import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.nio.file.Path;
import java.time.Duration;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.devicefarm.DeviceFarmClient;
import software.amazon.awssdk.services.devicefarm.model.Artifact;
import software.amazon.awssdk.services.devicefarm.model.ArtifactCategory;
import software.amazon.awssdk.services.devicefarm.model.ListArtifactsRequest;
import software.amazon.awssdk.services.devicefarm.model.ListArtifactsResponse;

public class AppiumLogDownloader {

    public static void main(String[] args) throws Exception {
        String sessionArn = "arn:aws:devicefarm:us-
west-2:111122223333:session:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-45678

        try (DeviceFarmClient client = DeviceFarmClient.builder()
                .region(Region.US_WEST_2)
                .build()) {

            // 1) List artifacts for the session (FILE artifacts) with pagination
            List<Artifact> all = new ArrayList<>();
            String token = null;
            do {
                ListArtifactsRequest.Builder b = ListArtifactsRequest.builder()
                        .arn(sessionArn)
                        .type(ArtifactCategory.FILE);
                if (token != null) b.nextToken(token);
                ListArtifactsResponse page = client.listArtifacts(b.build());
                all.addAll(page.artifacts());
                token = page.nextToken();
            } while (token != null && !token.isBlank());

            // Filter strictly to Appium logs
            List<Artifact> filtered = all.stream()
```

```
                    .filter(a -> {
                        String t = a.typeAsString();
                        return "APPIUM_SERVER_OUTPUT".equals(t) ||
"APPIUM_SERVER_LOG_OUTPUT".equals(t);
                    })
                    .toList();

            if (filtered.isEmpty()) {
                throw new RuntimeException("No Appium server log artifacts found
(expected APPIUM_SERVER_OUTPUT or APPIUM_SERVER_LOG_OUTPUT).");
            }

            // Prefer OUTPUT; else LOG_OUTPUT
            Artifact chosen = filtered.stream()
                    .filter(a -> "APPIUM_SERVER_OUTPUT".equals(a.typeAsString()))
                    .findFirst()
                    .orElseGet(() -> filtered.stream()
                            .filter(a ->
"APPIUM_SERVER_LOG_OUTPUT".equals(a.typeAsString()))
                            .findFirst()
                            .get());

            String url = chosen.url();
            String ext = (chosen.extension() == null ||
chosen.extension().isBlank()) ? "log" : chosen.extension();
            Path out = Path.of("appium_server_log." + ext);

            // 2) Download the artifact with HttpClient
            HttpClient http = HttpClient.newBuilder()
                    .connectTimeout(Duration.ofSeconds(10))
                    .build();

            HttpRequest get = HttpRequest.newBuilder(URI.create(url))
                    .timeout(Duration.ofMinutes(5))
                    .GET()
                    .build();

            HttpResponse<Path> resp = http.send(get,
HttpResponse.BodyHandlers.ofFile(out));
            if (resp.statusCode() / 100 != 2) {
                throw new IOException("Failed to download log, HTTP " +
resp.statusCode());
            }
```

```
            System.out.println("Saved Appium server log to: " +
 out.toAbsolutePath());
        }
    }
}
```

Ini akan menampilkan output seperti berikut:

```
info Appium Welcome to Appium v2.5.4
info Appium Non-default server args:
info Appium { address: '127.0.0.1', ..., useDrivers: [ 'uiautomator' ] }
```

JavaScript

Catatan: contoh ini menggunakan AWS SDK for JavaScript (v3) dan Node 18+ *fetch* untuk mengunduh log.

Selama atau setelah sesi, Anda dapat menggunakan [ListArtifacts](#)API Device Farm untuk mengambil URL log server Appium, lalu mengunduhnya.

```
import { DeviceFarmClient, ListArtifactsCommand } from "@aws-sdk/client-device-
farm";
import { createWriteStream } from "fs";
import { pipeline } from "stream";
import { promisify } from "util";

const pipe = promisify(pipeline);
const client = new DeviceFarmClient({ region: "us-west-2" });

const sessionArn = "arn:aws:devicefarm:us-
west-2:111122223333:session:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-45678

// 1) List artifacts for the session (FILE artifacts), handling pagination
const artifacts = [];
let nextToken;
do {
  const page = await client.send(new ListArtifactsCommand({
    arn: sessionArn,
    type: "FILE",
    nextToken
  }));
  artifacts.push(...(page.artifacts ?? []));
```

```
  nextToken = page.nextToken;
} while (nextToken);

if (!artifacts.length) throw new Error("No artifacts found");

// Strict filter to Appium logs
const filtered = (artifacts ?? []).filter(a =>
  a.type === "APPIUM_SERVER_OUTPUT" || a.type === "APPIUM_SERVER_LOG_OUTPUT"
);
if (!filtered.length) {
  throw new Error("No Appium server log artifacts found (expected
 APPIUM_SERVER_OUTPUT or APPIUM_SERVER_LOG_OUTPUT).");
}

// Prefer OUTPUT; else LOG_OUTPUT
const chosen =
  filtered.find(a => a.type === "APPIUM_SERVER_OUTPUT") ??
  filtered.find(a => a.type === "APPIUM_SERVER_LOG_OUTPUT");

const url = chosen.url;
const ext = chosen.extension || "log";
const outPath = `./appium_server_log.${ext}`;

// 2) Download the artifact
const resp = await fetch(url);
if (!resp.ok) {
  throw new Error(`Failed to download log: ${resp.status} ${await
 resp.text().catch(()=>"")}`);
}
await pipe(resp.body, createWriteStream(outPath));
console.log("Saved Appium server log to:", outPath);
```

Ini akan menampilkan output seperti berikut:

```
info Appium Welcome to Appium v2.5.4
info Appium Non-default server args:
info Appium { address: '127.0.0.1', allowInsecure: [ 'execute_driver_script', ... ],
 useDrivers: [ 'uiautomator' ] }
```

C#

Catatan: contoh ini menggunakan AWS SDK for .NET *`HttpClient`* dan untuk men-download log.

Selama atau setelah sesi, Anda dapat menggunakan ListArtifactsAPI Device Farm untuk mengambil URL log server Appium, lalu mengunduhnya.

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Net.Http;
using System.Threading.Tasks;
using System.Linq;
using Amazon;
using Amazon.DeviceFarm;
using Amazon.DeviceFarm.Model;

class AppiumLogDownloader
{
    static async Task Main()
    {
        var sessionArn = "arn:aws:devicefarm:us-
west-2:111122223333:session:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-45678

        using var client = new AmazonDeviceFarmClient(RegionEndpoint.USWest2);

        // 1) List artifacts for the session (FILE artifacts), handling pagination
        var all = new List<Artifact>();
        string? token = null;
        do
        {
            var page = await client.ListArtifactsAsync(new ListArtifactsRequest
            {
                Arn = sessionArn,
                Type = ArtifactCategory.FILE,
                NextToken = token
            });
            if (page.Artifacts != null) all.AddRange(page.Artifacts);
            token = page.NextToken;
        } while (!string.IsNullOrEmpty(token));

        if (all.Count == 0)
            throw new Exception("No artifacts found");

        // Strict filter to Appium logs
        var filtered = all.Where(a =>
```

```
            a.Type == "APPIUM_SERVER_OUTPUT" || a.Type ==
 "APPIUM_SERVER_LOG_OUTPUT").ToList();

        if (filtered.Count == 0)
            throw new Exception("No Appium server log artifacts found (expected
 APPIUM_SERVER_OUTPUT or APPIUM_SERVER_LOG_OUTPUT).");

        // Prefer OUTPUT; else LOG_OUTPUT
        var chosen = filtered.FirstOrDefault(a => a.Type == "APPIUM_SERVER_OUTPUT")
                    ?? filtered.First(a => a.Type == "APPIUM_SERVER_LOG_OUTPUT");

        var url = chosen.Url;
        var ext = string.IsNullOrWhiteSpace(chosen.Extension) ? "log" :
 chosen.Extension;
        var outPath = $"./appium_server_log.{ext}";

        // 2) Download the artifact
        using var http = new HttpClient();
        using var resp = await http.GetAsync(url,
 HttpCompletionOption.ResponseHeadersRead);
        resp.EnsureSuccessStatusCode();
        await using (var fs = File.Create(outPath))
        {
            await resp.Content.CopyToAsync(fs);
        }
        Console.WriteLine($"Saved Appium server log to:
 {Path.GetFullPath(outPath)}");
    }
}
```

Ini akan menampilkan output seperti berikut:

```
info Appium Welcome to Appium v2.5.4
info Appium Non-default server args:
info Appium { address: '127.0.0.1', ..., useDrivers: [ 'uiautomator' ] }
```

Ruby

Catatan: contoh ini menggunakan AWS SDK for *Net::HTTP* Ruby dan mengunduh log.

Selama atau setelah sesi, Anda dapat menggunakan [ListArtifacts](#)API Device Farm untuk mengambil URL log server Appium, lalu mengunduhnya.

```
require "aws-sdk-devicefarm"
require "net/http"
require "uri"

client = Aws::DeviceFarm::Client.new(region: "us-west-2")
session_arn = "arn:aws:devicefarm:us-
west-2:111122223333:session:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-45678

# 1) List artifacts for the session (FILE artifacts), handling pagination
artifacts = []
token = nil
loop do
  page = client.list_artifacts(arn: session_arn, type: "FILE", next_token: token)
  artifacts.concat(page.artifacts || [])
  token = page.next_token
  break if token.nil? || token.empty?
end

raise "No artifacts found" if artifacts.empty?

# Strict filter to Appium logs
filtered = (artifacts || []).select { |a| ["APPIUM_SERVER_OUTPUT",
 "APPIUM_SERVER_LOG_OUTPUT"].include?(a.type) }
raise "No Appium server log artifacts found (expected APPIUM_SERVER_OUTPUT or
 APPIUM_SERVER_LOG_OUTPUT)." if filtered.empty?

# Prefer OUTPUT; else LOG_OUTPUT
chosen = filtered.find { |a| a.type == "APPIUM_SERVER_OUTPUT" } ||
         filtered.find { |a| a.type == "APPIUM_SERVER_LOG_OUTPUT" }

url = chosen.url
ext = (chosen.extension && !chosen.extension.empty?) ? chosen.extension : "log"
out_path = "./appium_server_log.#{ext}"

# 2) Download the artifact
uri = URI.parse(url)
Net::HTTP.start(uri.host, uri.port, use_ssl: (uri.scheme == "https")) do |http|
  req = Net::HTTP::Get.new(uri)
  http.request(req) do |resp|
    raise "Failed GET: #{resp.code} #{resp.body}" unless resp.code.to_i / 100 == 2
    File.open(out_path, "wb") { |f| resp.read_body { |chunk| f.write(chunk) } }
  end
end
```

```
puts "Saved Appium server log to: #{File.expand_path(out_path)}"
```

Ini akan menampilkan output seperti berikut:

```
info Appium Welcome to Appium v2.5.4
info Appium Non-default server args:
info Appium { address: '127.0.0.1', allowInsecure: [ 'execute_driver_script', ... ],
 useDrivers: [ 'uiautomator' ] }
```

Kemampuan dan perintah Appium yang didukung

Titik akhir Appium Device Farm mendukung sebagian besar perintah yang sama dan kemampuan yang diinginkan yang Anda gunakan di perangkat lokal, dengan beberapa pengecualian. Daftar berikut menunjukkan kemampuan dan perintah mana yang saat ini tidak didukung. Jika pengujian Anda tidak dapat berjalan seperti yang diharapkan karena kemampuan terbatas, silakan buka kasus dukungan untuk panduan tambahan.

Kemampuan yang didukung

Saat membuat sesi Appium di Device Farm, sebaiknya Anda memiliki serangkaian kemampuan berbeda yang mengecualikan kemampuan apa pun yang spesifik untuk perangkat lokal Anda. Di Device Farm, pembuatan sesi mungkin gagal jika kemampuan tertentu yang tidak didukung disetel. Ini termasuk kemampuan khusus perangkat seperti dan. `udid` `platformVersion` Selain itu, kemampuan tertentu yang terkait dengan ChromeDriver WebDriverAgent Android dan iOS tidak didukung, serta kemampuan yang hanya didukung pada emulator dan simulator.

Perintah yang Didukung

Sebagian besar perintah Appium yang berjalan dengan benar di perangkat Android dan iOS nyata akan berjalan seperti yang diharapkan di Device Farm, dengan pengecualian berikut:

Perintah perangkat Appium () **`/appium/device`**

- `install_app`
- `finger_print`
- `send_sms`
- `gsm_call`

- gsm_signal
- gsm_voice
- power_ac
- power_capacity
- network_speed
- shake

Appium mengeksekusi metode dan skrip () **/execute**

- installApp
- execEmuConsoleCommand
- fingerprint
- gsmCall
- gsmSignal
- sendSms
- gsmVoice
- powerAC
- powerCapacity
- networkSpeed
- sensorSet
- injectEmulatorCameraImage
- isGpsEnabled
- shake
- clearApp
- clearKeychains
- configureLocalization
- enrollBiometric
- getPasteboard
- installXCTestBundle
- listXCTestBundles
- listXCTestsInTestBundle

- runXCTest
- sendBiometricMatch
- setPasteboard
- setPermission
- startAudioRecording
- startLogsBroadcast
- startRecordingScreen
- startScreenStreaming
- startXCTestScreenRecording
- stopAudioRecording
- stopLogsBroadcast
- stopRecordingScreen
- stopScreenStreaming
- stopXCTestScreenRecording
- updateSafariPreferences

Perangkat pribadi di AWS Device Farm

Perangkat pribadi adalah perangkat seluler fisik yang digunakan AWS Device Farm atas nama Anda di pusat data Amazon. Perangkat ini eksklusif untuk AWS akun Anda.

> **Note**
>
> Saat ini, perangkat pribadi hanya tersedia di Wilayah AWS AS Barat (Oregon) (`us-west-2`).

Jika Anda memiliki armada perangkat pribadi, Anda dapat membuat sesi akses jarak jauh dan menjadwalkan uji coba dengan perangkat pribadi Anda. Untuk informasi selengkapnya, lihat Membuat uji coba atau memulai sesi akses jarak jauh di AWS Device Farm. Anda juga dapat membuat profil instance untuk mengontrol perilaku perangkat pribadi Anda selama sesi akses jarak jauh atau uji coba. Untuk informasi selengkapnya, lihat Membuat profil instans di AWS Device Farm. Secara opsional, Anda dapat meminta agar perangkat pribadi Android tertentu digunakan sebagai perangkat yang di-root.

Anda juga dapat membuat layanan endpoint Amazon Virtual Private Cloud untuk menguji aplikasi pribadi yang dapat diakses perusahaan Anda, tetapi tidak dapat dijangkau melalui internet. Misalnya, Anda mungkin memiliki aplikasi web yang berjalan di VPC yang ingin Anda uji di perangkat seluler. Untuk informasi selengkapnya, lihat Menggunakan layanan endpoint Amazon VPC dengan Device Farm - Legacy (tidak disarankan).

Jika Anda tertarik untuk menggunakan armada perangkat pribadi, hubungi kami. Tim Device Farm harus bekerja sama dengan Anda untuk menyiapkan dan menyebarkan armada perangkat pribadi untuk AWS akun Anda.

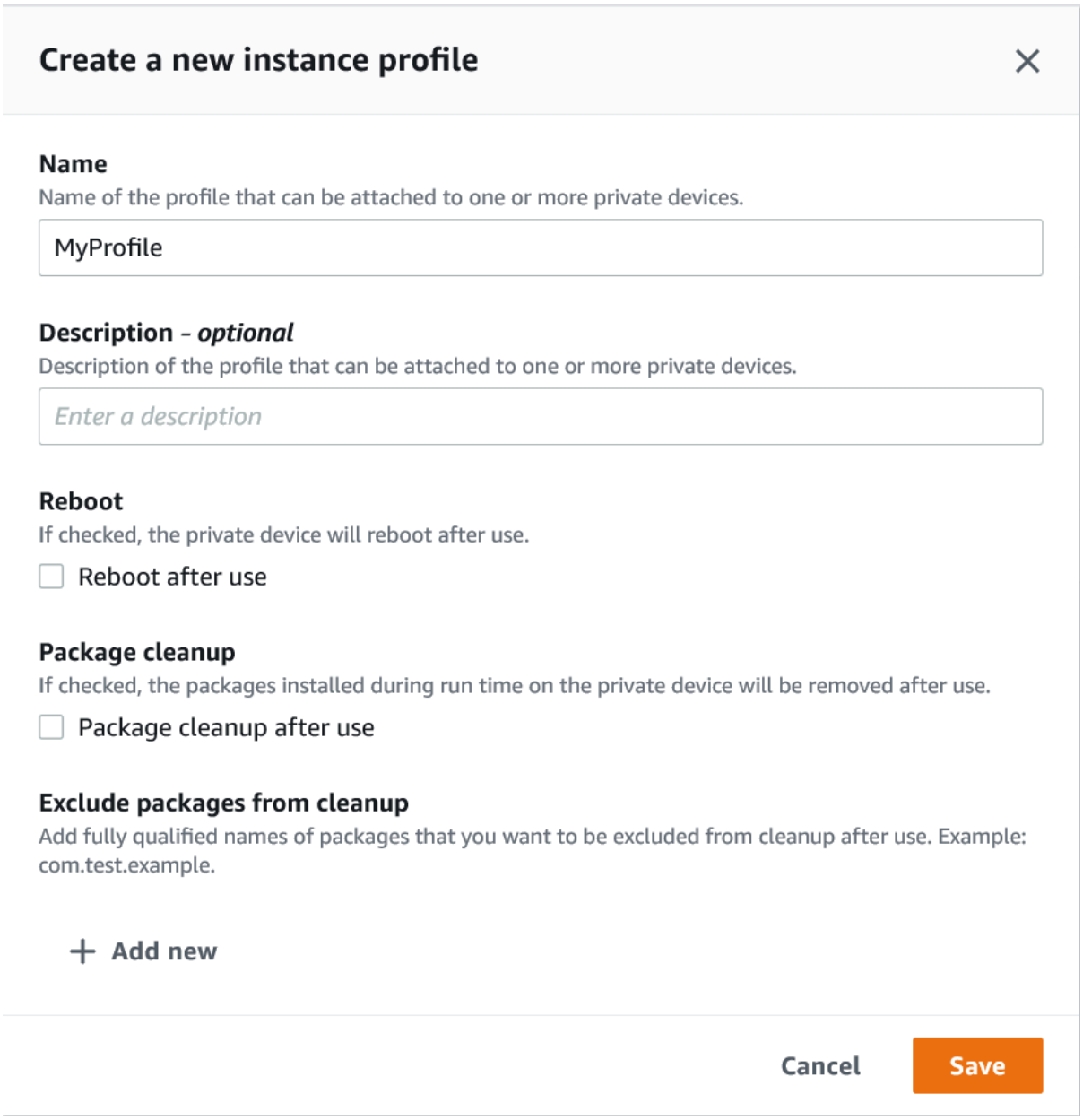
Topik

Membuat profil instans di AWS Device Farm

Anda dapat mengatur armada yang berisi satu atau lebih perangkat pribadi. Perangkat ini didedikasikan untuk AWS akun Anda. Setelah menyiapkan perangkat, Anda dapat membuat satu atau beberapa profil instans secara opsional untuk perangkat tersebut. Profil instans dapat membantu Anda mengotomatiskan proses pengujian dan secara konsisten menerapkan pengaturan yang sama ke instance perangkat. Profil instans juga dapat membantu Anda mengontrol perilaku sesi akses jarak jauh. Untuk informasi selengkapnya tentang perangkat pribadi di Device Farm, lihatPerangkat pribadi di AWS Device Farm.

Untuk membuat instans

1. Buka konsol Device Farm di https://console.aws.amazon.com/devicefarm/.
2. Pada panel navigasi Device Farm, pilih Pengujian Perangkat Seluler, lalu pilih Perangkat pribadi.
3. Pilih profil Instance.
4. Pilih Buat profil contoh.
5. Masukkan nama untuk profil instance.



6. (Opsional) Masukkan deskripsi untuk profil instance.
7. (Opsional) Ubah salah satu setelan berikut untuk menentukan tindakan yang Anda inginkan Device Farm lakukan pada perangkat setelah setiap uji coba atau sesi berakhir:

   - Reboot setelah digunakan — Untuk me-reboot perangkat, pilih kotak centang ini. Secara default, kotak centang ini dihapus (`false`).
   - Package cleanup — Untuk menghapus semua paket aplikasi yang diinstal pada perangkat, pilih kotak centang ini. Secara default, kotak centang ini dihapus (`false`). Untuk menyimpan semua paket aplikasi yang Anda instal di perangkat, biarkan kotak centang ini dihapus.

- Kecualikan paket dari pembersihan — Untuk menyimpan hanya paket aplikasi yang dipilih di perangkat, pilih kotak centang Package Cleanup, lalu pilih Tambah baru. Untuk nama paket, masukkan nama paket aplikasi yang sepenuhnya memenuhi syarat yang ingin Anda simpan di perangkat (misalnya,`com.test.example`). Untuk menyimpan lebih banyak paket aplikasi di perangkat, pilih Tambah baru, lalu masukkan nama yang memenuhi syarat untuk setiap paket.

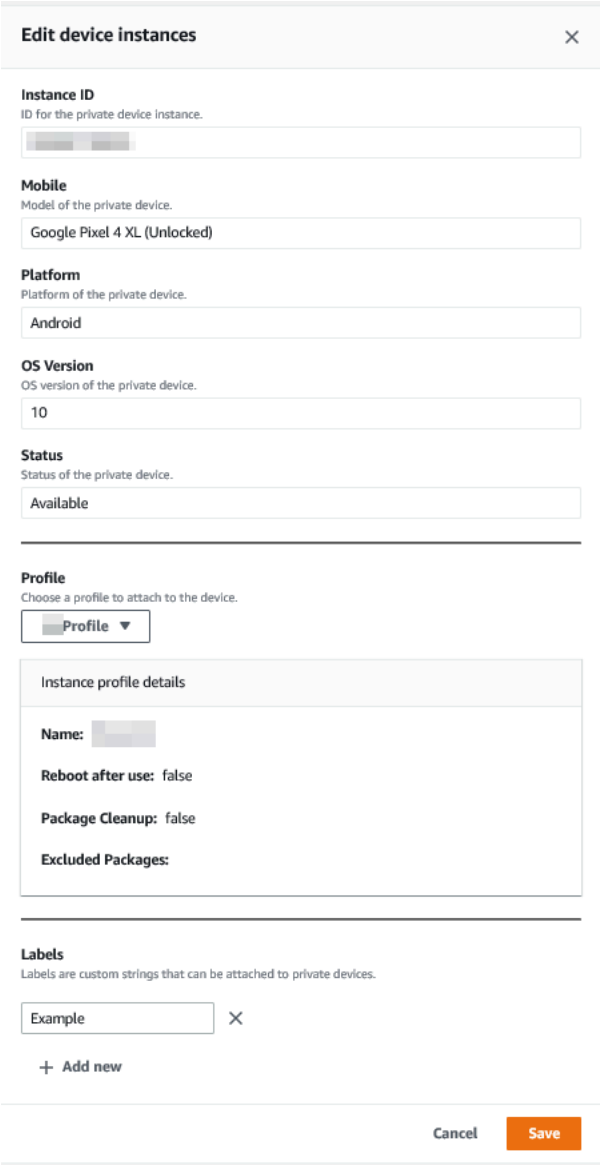
8. Pilih Simpan.

Minta perangkat pribadi tambahan di AWS Device Farm

Di AWS Device Farm, Anda dapat meminta instans perangkat pribadi tambahan untuk ditambahkan ke armada Anda. Anda juga dapat melihat dan mengubah pengaturan instans perangkat pribadi yang ada di armada Anda. Untuk informasi selengkapnya tentang perangkat pribadi, lihatPerangkat pribadi di AWS Device Farm.

Untuk meminta perangkat pribadi tambahan atau mengubah pengaturannya

1. Buka konsol Device Farm di https://console.aws.amazon.com/devicefarm/.
2. Pada panel navigasi Device Farm, pilih Pengujian Perangkat Seluler, lalu pilih Perangkat pribadi.
3. Pilih Instans perangkat. Tab Instans Perangkat menampilkan tabel perangkat pribadi yang ada di armada Anda. Untuk mencari atau memfilter tabel dengan cepat, masukkan istilah pencarian di bilah pencarian di atas kolom.
4. Untuk meminta instans perangkat pribadi baru, pilih Minta instance perangkat atau hubungi kami. Perangkat pribadi memerlukan pengaturan tambahan dengan bantuan dari tim Device Farm.
5. Dalam tabel instance perangkat, pilih opsi sakelar di sebelah instance yang ingin Anda lihat atau kelola informasi, lalu pilih Edit.



6. Untuk melampirkan profil instance ke instance perangkat, pilih dari daftar drop-down Profil. Melampirkan profil instance dapat membantu jika Anda ingin selalu mengecualikan paket aplikasi tertentu dari tugas pembersihan, misalnya. Untuk informasi selengkapnya tentang penggunaan profil instans dengan perangkat, lihatMembuat profil instans di AWS Device Farm.
7. (Opsional) Di bawah Label, pilih Tambahkan baru untuk menambahkan label ke instance perangkat. Label dapat membantu Anda mengkategorikan perangkat Anda dan menemukan perangkat tertentu dengan lebih mudah.
8. Pilih Simpan.

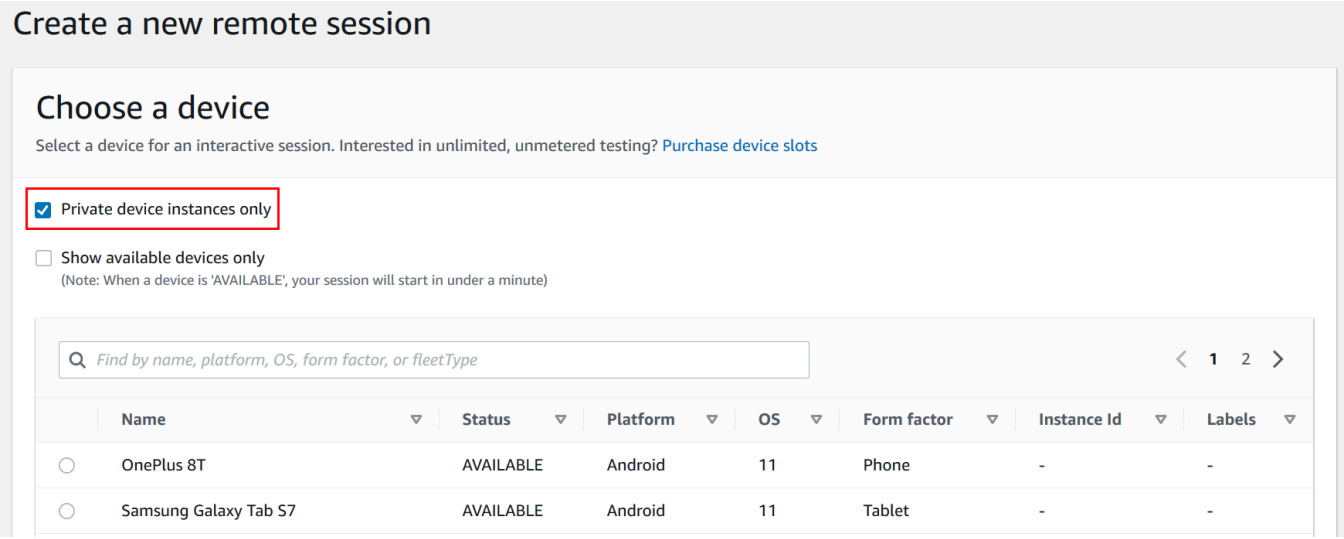
Membuat uji coba atau memulai sesi akses jarak jauh di AWS Device Farm

Di AWS Device Farm, setelah menyiapkan armada perangkat pribadi, Anda dapat membuat uji coba atau memulai sesi akses jarak jauh dengan satu atau beberapa perangkat pribadi di armada Anda. Untuk informasi selengkapnya tentang perangkat pribadi, lihatPerangkat pribadi di AWS Device Farm.

Untuk membuat uji coba atau memulai sesi akses jarak jauh

1. Buka konsol Device Farm di https://console.aws.amazon.com/devicefarm/.
2. Pada panel navigasi Device Farm, pilih Pengujian Perangkat Seluler, lalu pilih Proyek.
3. Pilih proyek yang sudah ada dari daftar atau buat yang baru. Untuk membuat proyek baru, pilih Proyek baru, masukkan nama untuk proyek, lalu pilih Kirim.
4. Lakukan salah satu tindakan berikut:

   - Untuk membuat uji coba, pilih Pengujian otomatis, lalu pilih Buat proses baru. Wizard memandu Anda melalui langkah-langkah untuk membuat run. Untuk langkah Pilih perangkat, Anda dapat mengedit kumpulan perangkat yang ada atau membuat kumpulan perangkat baru yang hanya menyertakan perangkat pribadi yang disiapkan oleh tim Device Farm dan dikaitkan dengan AWS akun Anda. Untuk informasi selengkapnya, lihat the section called “Buat kolam perangkat pribadi”.
   - Untuk memulai sesi akses jarak jauh, pilih Akses jarak jauh, lalu pilih Mulai sesi baru. Pada halaman Pilih perangkat, pilih Instans perangkat pribadi hanya untuk membatasi daftar hanya perangkat pribadi yang disiapkan oleh tim Device Farm dan terkait dengan AWS akun Anda. Kemudian, pilih perangkat yang ingin Anda akses, masukkan nama untuk sesi akses jarak jauh, dan pilih Konfirmasi dan mulai sesi.



Memilih perangkat pribadi di kumpulan perangkat di AWS Device Farm

Untuk menggunakan perangkat pribadi dalam uji coba, Anda dapat membuat kumpulan perangkat yang memilih perangkat pribadi Anda. Kumpulan perangkat memungkinkan Anda memilih perangkat pribadi terutama melalui tiga jenis aturan kumpulan perangkat:

1. Aturan berdasarkan perangkat ARN
2. Aturan berdasarkan label instance perangkat
3. Aturan berdasarkan ARN instance perangkat

Pada bagian berikut, setiap jenis aturan dan kasus penggunaannya dijelaskan secara mendalam. Anda dapat menggunakan konsol Device Farm, AWS Command Line Interface (AWS CLI), atau Device Farm API untuk membuat atau memodifikasi kumpulan perangkat dengan perangkat pribadi menggunakan aturan ini.

Topik

- Membuat kumpulan perangkat pribadi dengan perangkat pribadi (API)

Perangkat ARN

ARN perangkat adalah pengidentifikasi yang mewakili jenis perangkat daripada instance perangkat fisik tertentu. Jenis perangkat ditentukan oleh atribut berikut:

- ID armada perangkat
- OEM perangkat
- Nomor model perangkat
- Versi sistem operasi perangkat
- Status perangkat yang menunjukkan apakah itu di-root atau tidak

Banyak instance perangkat fisik dapat diwakili oleh satu jenis perangkat di mana setiap instance dari tipe tersebut memiliki nilai yang sama untuk atribut ini. Misalnya, jika Anda memiliki tiga *Apple iPhone 13* perangkat pada versi iOS *16.1.0* di armada pribadi Anda, setiap perangkat akan berbagi ARN perangkat yang sama. Jika ada perangkat yang ditambahkan atau dihapus dari armada Anda dengan atribut yang sama, perangkat ARN akan terus mewakili perangkat apa pun yang tersedia yang Anda miliki di armada Anda untuk jenis perangkat tersebut.

Perangkat ARN adalah cara paling kuat untuk memilih perangkat pribadi untuk kumpulan perangkat karena memungkinkan kumpulan perangkat untuk terus memilih perangkat terlepas dari instance perangkat tertentu yang telah Anda gunakan pada waktu tertentu. Instance perangkat pribadi individu dapat mengalami kegagalan perangkat keras, sehingga Device Farm menggantinya secara otomatis dengan instans kerja baru dari jenis perangkat yang sama. Dalam skenario ini, aturan ARN perangkat memastikan bahwa kumpulan perangkat Anda dapat terus memilih perangkat jika terjadi kegagalan perangkat keras.

Saat Anda menggunakan aturan ARN perangkat untuk perangkat pribadi di kumpulan perangkat Anda dan menjadwalkan uji coba dengan kumpulan tersebut, Device Farm akan secara otomatis memeriksa instance perangkat pribadi mana yang diwakili oleh ARN perangkat tersebut. Dari contoh yang saat ini tersedia, salah satunya akan ditugaskan untuk menjalankan pengujian Anda. Jika saat ini tidak ada instance yang tersedia, Device Farm akan menunggu instans ARN perangkat pertama yang tersedia, dan menetapkannya untuk menjalankan pengujian Anda.

Label instance perangkat

Label instance perangkat adalah pengenal tekstual yang dapat Anda lampirkan sebagai metadata untuk instance perangkat. Anda dapat melampirkan beberapa label ke setiap instance perangkat dan label yang sama ke beberapa instance perangkat. Untuk informasi selengkapnya tentang menambahkan, memodifikasi, atau menghapus label perangkat dari instance perangkat, lihat Mengelola perangkat pribadi.

Label instans perangkat dapat menjadi cara yang kuat untuk memilih perangkat pribadi untuk kumpulan perangkat karena, jika Anda memiliki beberapa instance perangkat dengan label yang sama, maka ini memungkinkan kumpulan perangkat untuk memilih salah satu dari mereka untuk pengujian Anda. Jika ARN perangkat bukan aturan yang baik untuk kasus penggunaan Anda (misalnya, jika Anda ingin memilih dari perangkat dari beberapa jenis perangkat, atau jika Anda ingin memilih dari subset semua perangkat dari jenis perangkat), maka label instance perangkat dapat memungkinkan Anda memilih dari beberapa perangkat untuk kumpulan perangkat Anda dengan perincian yang lebih besar. Instance perangkat pribadi individu dapat mengalami kegagalan perangkat keras, sehingga Device Farm menggantinya secara otomatis dengan instans kerja baru dari jenis perangkat yang sama. Dalam skenario ini, instance perangkat pengganti tidak akan menyimpan metadata label instance apa pun dari perangkat yang diganti. Jadi, jika Anda menerapkan label instance perangkat yang sama ke beberapa instance perangkat, maka aturan label instance perangkat memastikan bahwa kumpulan perangkat Anda dapat terus memilih instance perangkat jika terjadi kegagalan perangkat keras.

Saat Anda menggunakan aturan label instance perangkat untuk perangkat pribadi di kumpulan perangkat Anda dan menjadwalkan pengujian yang dijalankan dengan kumpulan tersebut, Device Farm akan secara otomatis memeriksa instance perangkat pribadi mana yang diwakili oleh label instance perangkat tersebut, dan instance tersebut, pilih secara acak yang tersedia untuk menjalankan pengujian Anda. Jika tidak ada yang tersedia, Device Farm akan secara acak memilih instance perangkat apa pun dengan label instance perangkat untuk menjalankan pengujian dan mengantre pengujian untuk dijalankan di perangkat setelah tersedia.

Contoh ARN

Sebuah contoh perangkat ARN adalah pengidentifikasi yang mewakili instance perangkat logam kosong fisik yang digunakan dalam armada pribadi. Misalnya, jika Anda memiliki tiga *iPhone 13* perangkat di OS *15.0.0* di armada pribadi Anda, sementara setiap perangkat akan berbagi ARN perangkat yang sama, setiap perangkat juga akan memiliki ARN instans sendiri yang mewakili instance itu sendiri.

ARN instance perangkat adalah cara yang paling tidak kuat untuk memilih perangkat pribadi untuk kumpulan perangkat dan hanya disarankan jika label instance perangkat ARNs dan perangkat tidak sesuai dengan kasus penggunaan Anda. Instans perangkat ARNs sering digunakan sebagai aturan untuk kumpulan perangkat ketika instance perangkat tertentu dikonfigurasi dengan cara yang unik dan spesifik sebagai prasyarat untuk pengujian Anda dan jika konfigurasi tersebut perlu diketahui dan diverifikasi sebelum pengujian dijalankan di atasnya. Instance perangkat pribadi individu dapat mengalami kegagalan perangkat keras, sehingga Device Farm menggantinya secara otomatis dengan instans kerja baru dari jenis perangkat yang sama. Dalam skenario ini, instance perangkat pengganti akan memiliki ARN instance perangkat yang berbeda dari perangkat yang diganti. Jadi, jika Anda mengandalkan instance perangkat ARNs untuk kumpulan perangkat Anda, maka Anda harus mengubah definisi aturan kumpulan perangkat secara manual dari menggunakan ARN lama menjadi menggunakan ARN baru. Jika Anda perlu mengkonfigurasi perangkat secara manual untuk pengujiannya, maka ini bisa menjadi alur kerja yang efektif (dibandingkan dengan perangkat ARNs). Untuk pengujian pada skala besar, disarankan untuk mencoba mengadaptasi kasus penggunaan ini agar berfungsi dengan label instans perangkat dan jika memungkinkan, memiliki beberapa instance perangkat yang telah dikonfigurasi sebelumnya untuk pengujian.

Saat Anda menggunakan aturan ARN instance perangkat untuk perangkat pribadi di kumpulan perangkat Anda dan menjadwalkan pengujian yang dijalankan dengan kumpulan tersebut, Device Farm akan secara otomatis menetapkan pengujian tersebut ke instance perangkat tersebut. Jika instance perangkat tersebut tidak tersedia, Device Farm akan mengantri pengujian pada perangkat setelah tersedia.

Membuat kolam perangkat pribadi dengan perangkat pribadi (konsol)

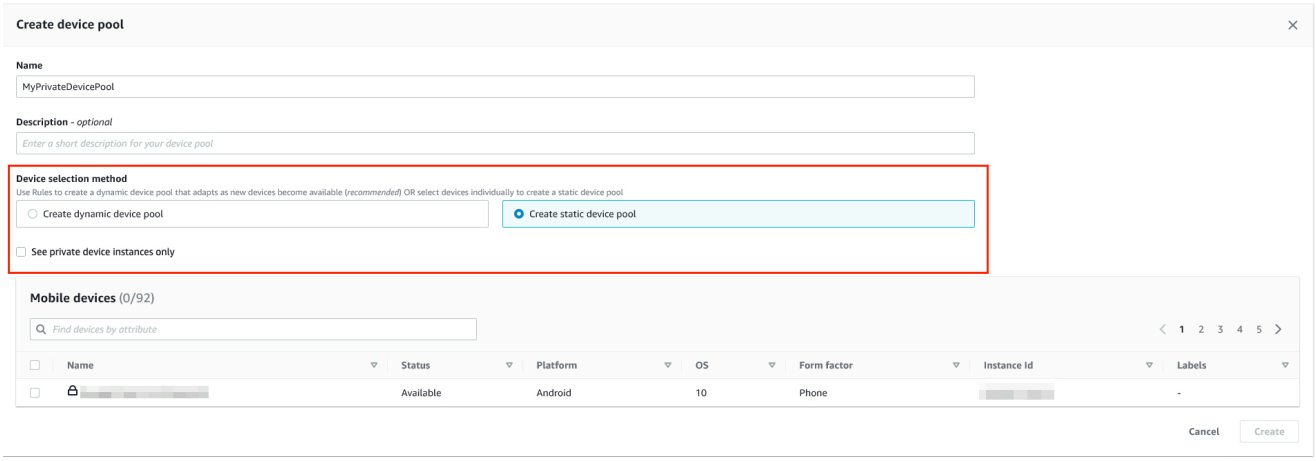
Saat membuat uji coba, Anda dapat membuat kumpulan perangkat untuk uji coba dan memastikan bahwa kumpulan tersebut hanya menyertakan perangkat pribadi Anda.

> **Note**
>
> Saat membuat kumpulan perangkat dengan perangkat pribadi di konsol, Anda hanya dapat menggunakan salah satu dari tiga aturan yang tersedia untuk memilih perangkat pribadi. Jika Anda ingin membuat kumpulan perangkat yang berisi beberapa jenis aturan untuk perangkat pribadi (misalnya, kumpulan perangkat yang berisi aturan untuk instance perangkat ARNs dan perangkat ARNs), maka Anda perlu membuat kumpulan melalui CLI atau API.

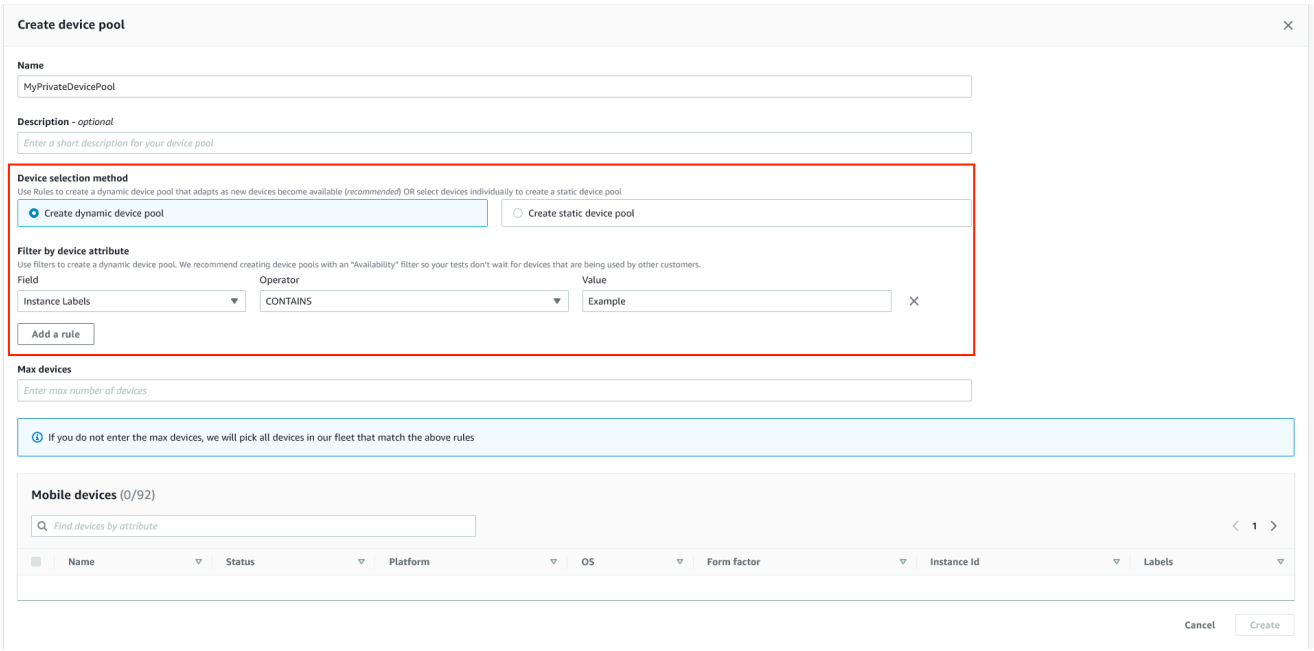
1. Buka konsol Device Farm di https://console.aws.amazon.com/devicefarm/.

2. Pada panel navigasi Device Farm, pilih Pengujian Perangkat Seluler, lalu pilih Proyek.
3. Pilih proyek yang sudah ada dari daftar atau buat yang baru. Untuk membuat proyek baru, pilih Proyek baru, masukkan nama untuk proyek, lalu pilih Kirim.
4. Pilih Pengaturan proyek, lalu arahkan ke tab Kumpulan perangkat.
5. Pilih Buat kumpulan perangkat, lalu masukkan nama dan deskripsi opsional untuk kumpulan perangkat Anda.

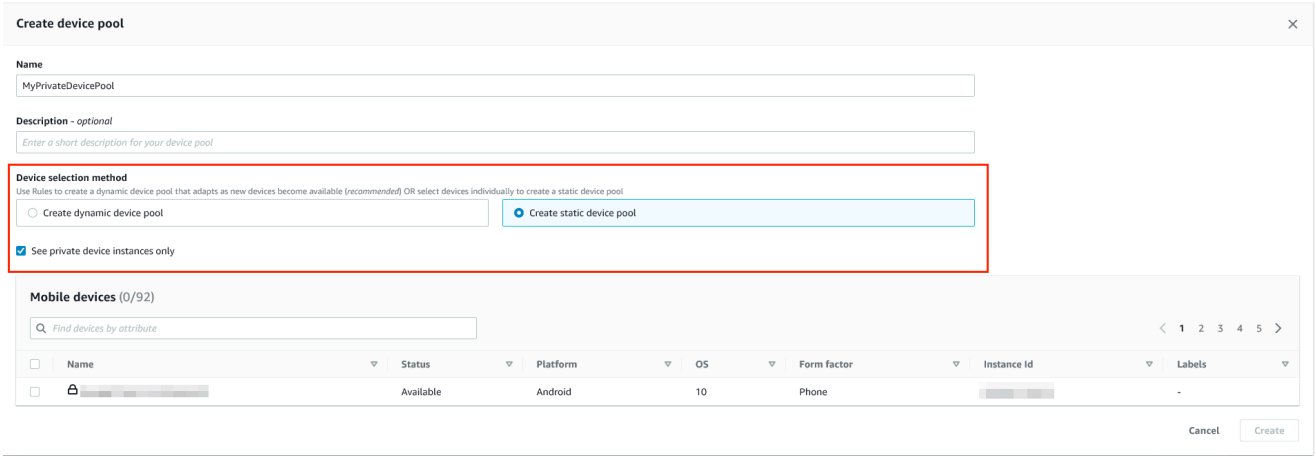
    a. Untuk menggunakan aturan ARN perangkat untuk kumpulan perangkat Anda, pilih Buat kumpulan perangkat statis, lalu pilih jenis perangkat tertentu dari daftar yang ingin Anda gunakan di kumpulan perangkat. Jangan pilih Instans perangkat pribadi hanya karena opsi ini menyebabkan kumpulan perangkat dibuat dengan aturan ARN instance perangkat (bukan aturan ARN perangkat).



    b. Untuk menggunakan aturan label instance perangkat untuk kumpulan perangkat Anda, pilih Buat kumpulan perangkat dinamis. Kemudian, untuk setiap label yang ingin Anda gunakan di kumpulan perangkat, pilih Tambahkan aturan. Untuk setiap aturan, pilih Label Instance sebagai`Field`, pilih Berisi sebagai`Operator`, dan tentukan label instance perangkat yang Anda inginkan sebagai`Value`.



c. Untuk menggunakan aturan ARN instance perangkat untuk kumpulan perangkat Anda, pilih Buat kumpulan perangkat statis, lalu pilih Instans perangkat pribadi hanya untuk membatasi daftar perangkat hanya untuk instance perangkat pribadi yang telah dikaitkan Device Farm dengan akun Anda. AWS



6. Pilih Buat.

Membuat kolam perangkat pribadi dengan perangkat pribadi (AWS CLI)

- Jalankan perintah create-device-pool.

Untuk informasi tentang menggunakan Device Farm dengan AWS CLI, lihatAWS CLI referensi.

Membuat kumpulan perangkat pribadi dengan perangkat pribadi (API)

- Panggil `CreateDevicePool`API.

Untuk informasi tentang menggunakan Device Farm API, lihatMengotomatisasi Device Farm.

Melewati penandatanganan ulang aplikasi pada perangkat pribadi di AWS Device Farm

Penandatanganan aplikasi adalah proses yang melibatkan penandatanganan paket aplikasi secara digital (misalnya, APK, IPA) dengan kunci pribadi sebelum dapat diinstal pada perangkat atau dipublikasikan ke toko aplikasi seperti Google Play Store atau Apple App Store. Untuk merampingkan pengujian dengan mengurangi jumlah tanda tangan dan profil yang diperlukan serta meningkatkan keamanan data pada perangkat jarak jauh, AWS Device Farm akan menandatangani ulang aplikasi Anda setelah diunggah ke layanan.

Setelah mengunggah aplikasi ke AWS Device Farm, layanan akan menghasilkan tanda tangan baru untuk aplikasi menggunakan sertifikat penandatanganan dan profil penyediaannya sendiri. Proses ini menggantikan tanda tangan aplikasi asli dengan tanda tangan AWS Device Farm. Aplikasi yang ditandatangani ulang kemudian diinstal pada perangkat uji yang disediakan oleh AWS Device Farm. Tanda tangan baru memungkinkan aplikasi diinstal dan dijalankan di perangkat ini tanpa memerlukan sertifikat pengembang asli.

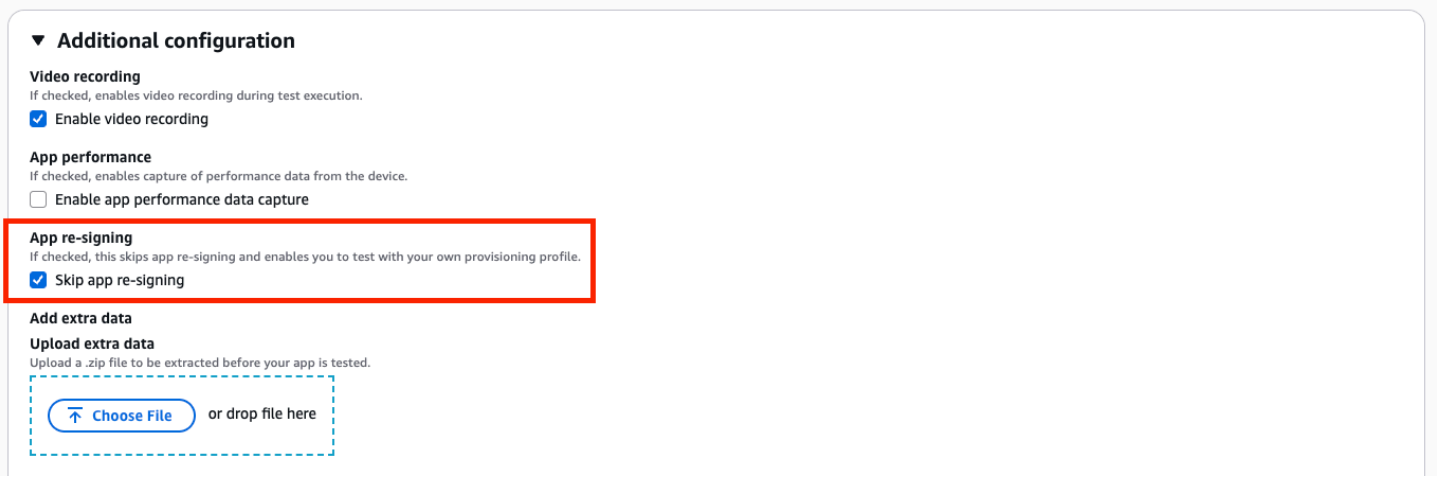
Di iOS, kami mengganti profil penyediaan yang disematkan dengan profil wildcard dan menandatangani ulang aplikasi. Jika Anda menyediakannya, kami akan menambahkan data tambahan ke paket aplikasi sebelum instalasi sehingga data akan ada di kotak pasir aplikasi Anda. Penandatanganan ulang aplikasi iOS menghasilkan penghapusan semua hak.

Di Android, kami menandatangani ulang aplikasi. Ini dapat merusak fungsionalitas yang bergantung pada tanda tangan aplikasi, seperti Google Maps Android API. Ini juga dapat memicu deteksi anti-pembajakan dan anti-tamper yang tersedia dari produk seperti. DexGuard Untuk pengujian bawaan, kami dapat memodifikasi manifes untuk menyertakan izin yang diperlukan untuk menangkap dan menyimpan tangkapan layar.

Saat menggunakan perangkat pribadi, Anda dapat melewati langkah di mana AWS Device Farm menandatangani ulang aplikasi Anda. Ini berbeda dengan perangkat publik, di mana Device Farm selalu menandatangani ulang aplikasi Anda di platform Android dan iOS.

Anda dapat melewati penandatanganan ulang aplikasi saat membuat sesi akses jarak jauh atau uji coba. Ini dapat membantu jika aplikasi Anda memiliki fungsionalitas yang rusak saat Device Farm menandatangani ulang aplikasi Anda. Misalnya, pemberitahuan push mungkin tidak berfungsi setelah penandatanganan ulang. Untuk informasi selengkapnya tentang perubahan yang dilakukan Device Farm saat menguji aplikasi Anda, lihat AWS Device Farm FAQs atau halaman Aplikasi.

Untuk melewati penandatanganan ulang aplikasi untuk uji coba, pilih Lewati penandatanganan ulang aplikasi di bawah Konfigurasi tambahan. Opsi ini hanya tersedia untuk perangkat pribadi.



> **Note**
>
> Jika Anda menggunakan XCTest kerangka kerja, opsi Lewati penandatanganan ulang aplikasi tidak tersedia. Untuk informasi selengkapnya, lihat Mengintegrasikan Device Farm dengan XCTest iOS.

Langkah-langkah tambahan untuk mengonfigurasi setelan penandatanganan aplikasi bervariasi, tergantung apakah Anda menggunakan perangkat Android atau iOS pribadi.

Melewatkan penandatanganan ulang aplikasi di perangkat Android

Jika Anda menguji aplikasi di perangkat Android pribadi, pilih Lewati penandatanganan ulang aplikasi saat Anda membuat uji coba atau sesi akses jarak jauh. Tidak ada konfigurasi lain yang diperlukan.

Melewatkan penandatanganan ulang aplikasi di perangkat iOS

Apple mengharuskan Anda menandatangani aplikasi untuk pengujian sebelum Anda memuatnya ke perangkat. Untuk perangkat iOS, Anda memiliki dua opsi untuk menandatangani aplikasi.

- Jika Anda menggunakan profil pengembang internal (Enterprise), Anda dapat melompat ke bagian berikutnya. the section called “Membuat sesi akses jarak jauh untuk mempercayai aplikasi Anda”

- Jika Anda menggunakan profil pengembangan aplikasi iOS ad hoc, Anda harus terlebih dahulu mendaftarkan perangkat dengan akun pengembang Apple Anda, lalu memperbarui profil penyediaan Anda untuk menyertakan perangkat pribadi. Anda kemudian harus menandatangani ulang aplikasi Anda dengan profil penyediaan yang Anda perbarui. Anda kemudian dapat menjalankan aplikasi yang ditandatangani ulang di Device Farm.

Untuk mendaftarkan perangkat dengan profil penyediaan pengembangan aplikasi iOS ad hoc

1. Masuk ke akun pengembang Apple Anda.
2. Arahkan ke bagian Sertifikat IDs,, dan Profil di konsol.
3. Pergi ke Perangkat.
4. Daftarkan perangkat di akun pengembang Apple Anda. Untuk mendapatkan nama dan UDID perangkat, gunakan `ListDeviceInstances` pengoperasian Device Farm API.
5. Buka profil penyediaan Anda dan pilih Edit.
6. Pilih perangkat dari daftar.
7. Di Xcode, ambil profil penyediaan Anda yang diperbarui, lalu tandatangani ulang aplikasi.

Tidak ada konfigurasi lain yang diperlukan. Anda sekarang dapat membuat sesi akses jarak jauh atau uji coba dan pilih Lewati penandatanganan ulang aplikasi.

Membuat sesi akses jarak jauh untuk mempercayai aplikasi iOS Anda

Jika Anda menggunakan profil penyediaan pengembang internal (Enterprise), Anda harus melakukan prosedur satu kali untuk mempercayai sertifikat pengembang aplikasi internal di setiap perangkat pribadi Anda.

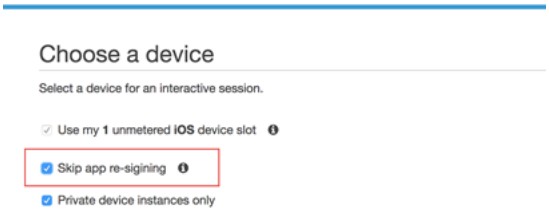
Untuk melakukannya, Anda harus menginstal aplikasi placeholder yang ditandatangani dengan sertifikat yang sama dengan aplikasi yang ingin Anda uji. Setelah perangkat mempercayai profil konfigurasi atau pengembang aplikasi perusahaan, semua aplikasi dari pengembang tersebut dipercaya di perangkat pribadi hingga Anda menghapusnya. Oleh karena itu, ketika Anda menginstal versi baru aplikasi yang ingin Anda uji, Anda tidak perlu mempercayai pengembang aplikasi lagi setiap kali. Ini sangat berguna jika Anda menjalankan otomatisasi pengujian dan Anda tidak ingin membuat sesi akses jarak jauh setiap kali Anda menguji aplikasi Anda.

Prosedur umum yang digunakan banyak pelanggan adalah menandatangani ulang aplikasi sampel Device Farm untuk iOS, lalu menginstalnya ke perangkat mereka sebagai aplikasi placeholder.

Sebelum memulai sesi akses jarak jauh, ikuti langkah-langkah Membuat profil instans di AWS Device Farm untuk membuat atau memodifikasi profil instance di Device Farm. Di profil instance, tambahkan ID bundel aplikasi placeholder ke setelan Kecualikan paket dari pembersihan. Kemudian, lampirkan profil instance ke instance perangkat pribadi untuk memastikan bahwa Device Farm tidak menghapus aplikasi ini dari perangkat sebelum memulai uji coba baru. Ini memastikan bahwa sertifikat pengembang Anda tetap tepercaya.

Anda dapat mengunggah aplikasi placeholder ke perangkat dengan menggunakan sesi akses jarak jauh, yang memungkinkan Anda meluncurkan aplikasi dan mempercayai pengembang.

1. Ikuti petunjuk Membuat sesi untuk membuat sesi akses jarak jauh yang menggunakan profil instans perangkat pribadi yang Anda buat. Saat membuat sesi, pastikan untuk memilih Lewati penandatanganan ulang aplikasi.



> **Important**
>
> Untuk memfilter daftar perangkat yang hanya menyertakan perangkat pribadi, pilih Instans perangkat pribadi hanya untuk memastikan bahwa Anda menggunakan perangkat pribadi dengan profil instans yang benar.

   Pastikan juga menambahkan aplikasi placeholder atau aplikasi yang ingin Anda uji ke pengaturan Kecualikan paket dari pembersihan untuk profil instance yang dilampirkan ke instance ini.
2. Saat sesi jarak jauh Anda dimulai, pilih Pilih File untuk menginstal aplikasi yang menggunakan profil penyediaan internal Anda.
3. Luncurkan aplikasi yang baru saja Anda unggah.
4. Konfirmasikan bahwa kotak dialog iOS muncul yang menunjukkan bahwa pengembang aplikasi perusahaan tidak dipercaya.

5. Kemudian, jika perangkat iOS menggunakan iOS versi 18 atau lebih tinggi, buka tiket dukungan dengan tim AWS Device Farm agar tim kami mempercayai aplikasi untuk Anda, karena perangkat ini mengharuskan aplikasi dipercaya secara manual. Jika tidak, jika versi iOS 17 atau lebih rendah, Anda dapat masuk ke aplikasi Pengaturan, dan, di bawah Pengaturan umum, percayai aplikasi sendiri dari menu VPN dan Profil.

Semua aplikasi dari profil konfigurasi atau pengembang aplikasi perusahaan ini sekarang dipercaya di perangkat pribadi ini hingga Anda menghapusnya.

Amazon VPC di seluruh AWS Wilayah di AWS Device Farm

Layanan Device Farm hanya berlokasi di Wilayah AS Barat (Oregon) (`us-west-2`). Anda dapat menggunakan Amazon Virtual Private Cloud (Amazon VPC) untuk menjangkau layanan di Amazon Virtual Private Cloud di AWS Wilayah lain menggunakan Device Farm. Jika Device Farm dan layanan Anda berada di Wilayah yang sama, lihatMenggunakan layanan endpoint Amazon VPC dengan Device Farm - Legacy (tidak disarankan).

Ada dua cara untuk mengakses layanan pribadi Anda yang berlokasi di Wilayah yang berbeda. Jika Anda memiliki layanan yang terletak di satu Wilayah lain yang tidak`us-west-2`, Anda dapat menggunakan VPC Peering untuk mengintip VPC Wilayah tersebut ke VPC lain yang berinteraksi dengan Device Farm in. `us-west-2` Namun, jika Anda memiliki layanan di beberapa Wilayah, Transit Gateway akan memungkinkan Anda mengakses layanan tersebut dengan konfigurasi jaringan yang lebih sederhana.

Untuk informasi selengkapnya, lihat skenario peering VPC di Panduan Peering VPC Amazon.

Ikhtisar peering VPC untuk Wilayah yang berbeda VPCs di AWS Device Farm

Anda dapat mengintip dua VPCs di Wilayah yang berbeda selama mereka memiliki blok CIDR yang berbeda dan tidak tumpang tindih. Ini memastikan bahwa semua alamat IP pribadi unik, dan memungkinkan semua sumber daya di dalam VPCs untuk saling menangani tanpa perlu bentuk terjemahan alamat jaringan (NAT) apa pun. Untuk informasi selengkapnya tentang notasi CIDR, lihat RFC 4632.

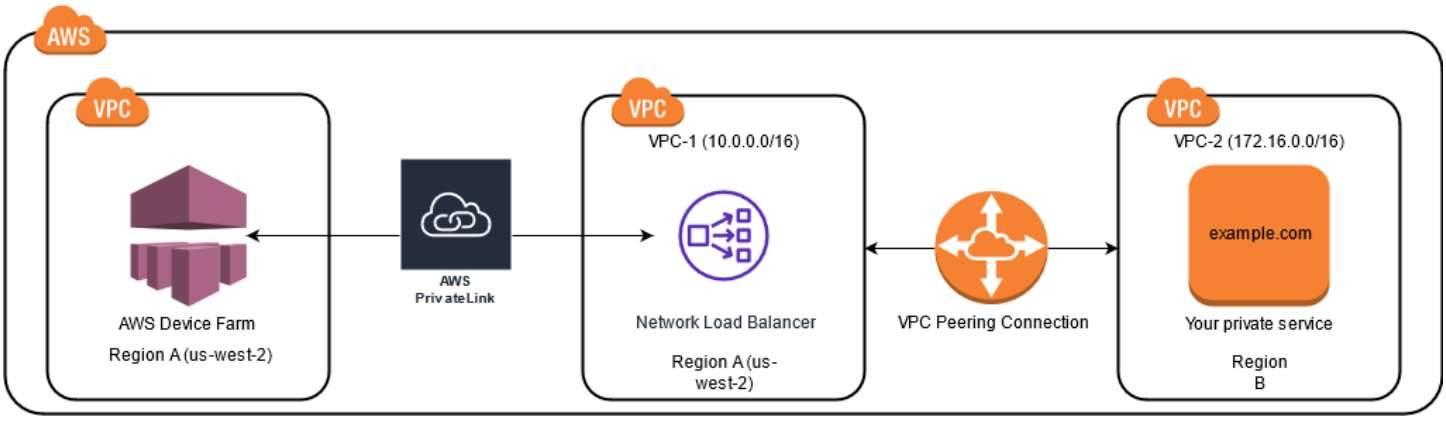
Topik ini mencakup contoh skenario Lintas wilayah di mana Device Farm (disebut sebagai VPC-1) berada di Wilayah AS Barat (Oregon) (). `us-west-2` VPC kedua dalam contoh ini (disebut sebagai VPC-2) ada di Wilayah lain.

Contoh Device Farm VPC Lintas wilayah

Komponen VPC	VPC-1	VPC-2
CIDR	10.0.0.0/16	172.16.0.0/16

> **Important**
>
> Membangun koneksi peering antara dua VPCs dapat mengubah postur keamanan. VPCs Selain itu, menambahkan entri baru ke tabel rute mereka dapat mengubah postur keamanan sumber daya di dalam VPCs. Adalah tanggung jawab Anda untuk menerapkan konfigurasi ini dengan cara yang memenuhi persyaratan keamanan organisasi Anda. Untuk informasi lebih lanjut, silakan lihat Model tanggung jawab bersama.

Diagram berikut menunjukkan komponen dalam contoh dan interaksi antara komponen-komponen ini.



Topik

- Prasyarat untuk menggunakan Amazon VPC di AWS Device Farm
- Langkah 1: Menyiapkan koneksi peering antara VPC-1 dan VPC-2
- Langkah 2: Memperbarui tabel rute di VPC-1 dan VPC-2
- Langkah 3: Membuat grup target
- Langkah 4: Membuat Network Load Balancer
- Langkah 5: Membuat layanan titik akhir VPC untuk menghubungkan VPC Anda ke Device Farm
- Langkah 6: Buat konfigurasi titik akhir VPC antara VPC dan Device Farm

- Langkah 7: Membuat uji coba untuk menggunakan konfigurasi titik akhir VPC
- Membuat jaringan yang dapat diskalakan dengan Transit Gateway

Prasyarat untuk menggunakan Amazon VPC di AWS Device Farm

Contoh ini membutuhkan yang berikut:

- Dua VPCs yang dikonfigurasi dengan subnet yang berisi blok CIDR yang tidak tumpang tindih.
- VPC-1 harus berada di `us-west-2` Wilayah dan berisi subnet untuk Availability Zones`us-west-2a`,, `us-west-2b` dan. `us-west-2c`

Untuk informasi selengkapnya tentang membuat VPCs dan mengonfigurasi subnet, lihat Bekerja dengan VPCs dan subnet di Panduan Peering VPC Amazon.

Langkah 1: Menyiapkan koneksi peering antara VPC-1 dan VPC-2

Buat koneksi peering antara keduanya yang VPCs mengandung blok CIDR yang tidak tumpang tindih. Untuk melakukannya, lihat Membuat dan menerima koneksi peering VPC di Panduan Peering VPC Amazon. Menggunakan skenario Lintas wilayah topik ini dan Panduan Peering VPC Amazon, contoh konfigurasi koneksi peering berikut dibuat:

Nama

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

ID VPC (Pemohon)

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

Akun

```
My account
```

Wilayah

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

ID VPC (Penerima)

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

> **Note**
>
> Pastikan Anda berkonsultasi dengan kuota koneksi peering VPC Anda saat membuat koneksi peering baru. Untuk informasi lebih lanjut, silakan lihat kuota Amazon VPC di Panduan Peering VPC Amazon.

Langkah 2: Memperbarui tabel rute di VPC-1 dan VPC-2

Setelah mengatur koneksi peering, Anda harus membuat rute tujuan antara keduanya VPCs untuk mentransfer data di antara keduanya. Untuk menetapkan rute ini, Anda dapat memperbarui tabel rute VPC-1 secara manual untuk menunjuk ke subnet VPC-2 dan sebaliknya. Untuk melakukannya, lihat Memperbarui tabel rute Anda untuk koneksi peering VPC di Panduan Peering VPC Amazon. Menggunakan skenario Lintas wilayah topik ini dan Panduan Peering VPC Amazon, contoh konfigurasi tabel rute berikut dibuat:

Contoh tabel rute VPC Device Farm

Komponen VPC	VPC-1	VPC-2
ID tabel rute	rtb-1234567890abcdefg	rtb-0987654321gfedcba
Rentang alamat lokal	10.0.0.0/16	172.16.0.0/16
Rentang alamat tujuan	172.16.0.0/16	10.0.0.0/16

Langkah 3: Membuat grup target

Setelah mengatur rute tujuan, Anda dapat mengonfigurasi Network Load Balancer di VPC-1 untuk merutekan permintaan ke VPC-2.

Network Load Balancer harus terlebih dahulu berisi grup target yang berisi alamat IP tempat permintaan dikirim.

Untuk membuat grup target

1. Identifikasi alamat IP layanan yang ingin Anda targetkan di VPC-2.
   - Alamat IP ini harus menjadi anggota subnet yang digunakan dalam koneksi peering.

- Alamat IP yang ditargetkan harus statis dan tidak dapat diubah. Jika layanan Anda memiliki alamat IP dinamis, pertimbangkan untuk menargetkan sumber daya statis (seperti Network Load Balancer) dan meminta rute sumber daya statis tersebut ke target Anda yang sebenarnya.

  > **Note**
  >
  > - Jika Anda menargetkan satu atau beberapa instans Amazon Elastic Compute Cloud (Amazon EC2) yang berdiri sendiri, buka konsol Amazon EC2 di, lalu pilih Instans. https://console.aws.amazon.com/ec2/
  > - Jika Anda menargetkan grup Amazon EC2 Auto Scaling dari instans Amazon EC2, Anda harus mengaitkan grup Amazon EC2 Auto Scaling ke Network Load Balancer. Untuk Informasi Selengkapnya, Lihat Memasang load balancer to your Auto Scaling group pada Amazon EC2 Auto Scaling User Guide.
  >
  >   Kemudian, Anda dapat membuka konsol Amazon EC2 di https://console.aws.amazon.com/ec2/, dan kemudian memilih Antarmuka Jaringan. Dari sana Anda dapat melihat alamat IP untuk masing-masing antarmuka jaringan Network Load Balancer di setiap Availability Zone.

2. Buat grup target di VPC-1. Untuk melakukannya, lihat Membuat grup target untuk Network Load Balancer Anda di Panduan Pengguna untuk Network Load Balancers.

   Grup sasaran untuk layanan di VPC yang berbeda memerlukan konfigurasi berikut:

   - Untuk Pilih jenis target, pilih alamat IP.
   - Untuk VPC, pilih VPC yang akan menjadi tuan rumah penyeimbang beban. Untuk contoh topik, ini akan menjadi VPC-1.
   - Pada halaman Daftar target, daftarkan target untuk setiap alamat IP di VPC-2.

     Untuk Jaringan, pilih Alamat IP pribadi lainnya.

     Untuk Availability Zone, pilih zona yang Anda inginkan di VPC-1.

     Untuk IPv4 alamat, pilih alamat IP VPC-2.

     Untuk Port, pilih port Anda.
   - Pilih Sertakan sebagai tertunda di bawah ini. Setelah selesai menentukan alamat, pilih Daftarkan target yang tertunda.

Menggunakan skenario lintas wilayah topik ini dan Panduan Pengguna untuk Network Load Balancers, nilai berikut digunakan dalam konfigurasi grup target:

Jenis target

```
IP addresses
```

Nama grup sasaran

```
my-target-group
```

Protokol/Pelabuhan

```
TCP : 80
```

VPC

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

Jaringan

```
Other private IP address
```

Zona Ketersediaan

```
all
```

IPv4 alamat

```
172.16.100.60
```

Pelabuhan

```
80
```

Langkah 4: Membuat Network Load Balancer

Buat Network Load Balancer menggunakan grup target yang dijelaskan pada langkah 3. Untuk melakukannya, lihat Membuat Network Load Balancer.

Menggunakan skenario Lintas wilayah topik ini, nilai berikut digunakan dalam contoh konfigurasi Network Load Balancer:

Nama penyeimbang beban

```
my-nlb
```

Skema

```
Internal
```

VPC

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

Pemetaan

```
us-west-2a - subnet-4i23iuufkdiufsloi
```

```
us-west-2b - subnet-7x989pkjj78nmn23j
```

```
us-west-2c - subnet-0231ndmas12bnnsds
```

Protokol/Pelabuhan

```
TCP : 80
```

Kelompok Sasaran

```
my-target-group
```

Langkah 5: Membuat layanan titik akhir VPC untuk menghubungkan VPC Anda ke Device Farm

Anda dapat menggunakan Network Load Balancer untuk membuat layanan endpoint VPC. Melalui layanan titik akhir VPC ini, Device Farm dapat terhubung ke layanan Anda di VPC-2 tanpa infrastruktur tambahan, seperti gateway internet, instans NAT, atau koneksi VPN.

Untuk melakukannya, lihat Membuat layanan endpoint Amazon VPC.

Langkah 6: Buat konfigurasi titik akhir VPC antara VPC dan Device Farm

Sekarang Anda dapat membuat koneksi pribadi antara VPC dan Device Farm Anda. Anda dapat menggunakan Device Farm untuk menguji layanan pribadi tanpa mengeksposnya melalui internet publik. Untuk melakukannya, lihat Membuat konfigurasi titik akhir VPC di Device Farm.

Menggunakan skenario lintas wilayah topik ini, nilai berikut digunakan dalam contoh konfigurasi titik akhir VPC:

Nama

```
My VPCE Configuration
```

Nama layanan VPCE

```
com.amazonaws.vpce.us-west-2.vpce-svc-1234567890abcdefg
```

Nama DNS layanan

```
devicefarm.com
```

Langkah 7: Membuat uji coba untuk menggunakan konfigurasi titik akhir VPC

Anda dapat membuat uji coba yang menggunakan konfigurasi titik akhir VPC yang dijelaskan pada langkah 6. Untuk informasi selengkapnya, lihat Membuat uji coba di Device Farm atau Membuat sesi.

Membuat jaringan yang dapat diskalakan dengan Transit Gateway

Untuk membuat jaringan yang dapat diskalakan menggunakan lebih dari dua VPCs, Anda dapat menggunakan Transit Gateway untuk bertindak sebagai hub transit jaringan untuk menghubungkan jaringan Anda VPCs dan lokal. Untuk mengonfigurasi VPC di wilayah yang sama dengan Device Farm agar menggunakan Gateway Transit, Anda dapat mengikuti layanan titik akhir Amazon VPC dengan panduan Device Farm untuk menargetkan sumber daya di wilayah lain berdasarkan alamat IP pribadinya.

Untuk informasi selengkapnya tentang Transit Gateway, lihat Apa itu gateway transit? di Panduan Gerbang Transit VPC Amazon.

Mengakhiri perangkat pribadi di Device Farm

<Untuk mengakhiri perangkat pribadi setelah jangka waktu awal yang disepakati, A
Untuk informasi selengkapnya tentang perangkat pribadi, lihatPerangkat pribadi di AWS Device Farm.

> **Important**
>
> Petunjuk ini hanya berlaku untuk mengakhiri perjanjian perangkat pribadi. Untuk semua masalah AWS layanan dan penagihan lainnya, lihat dokumentasi masing-masing untuk produk tersebut atau hubungi AWS dukungan.

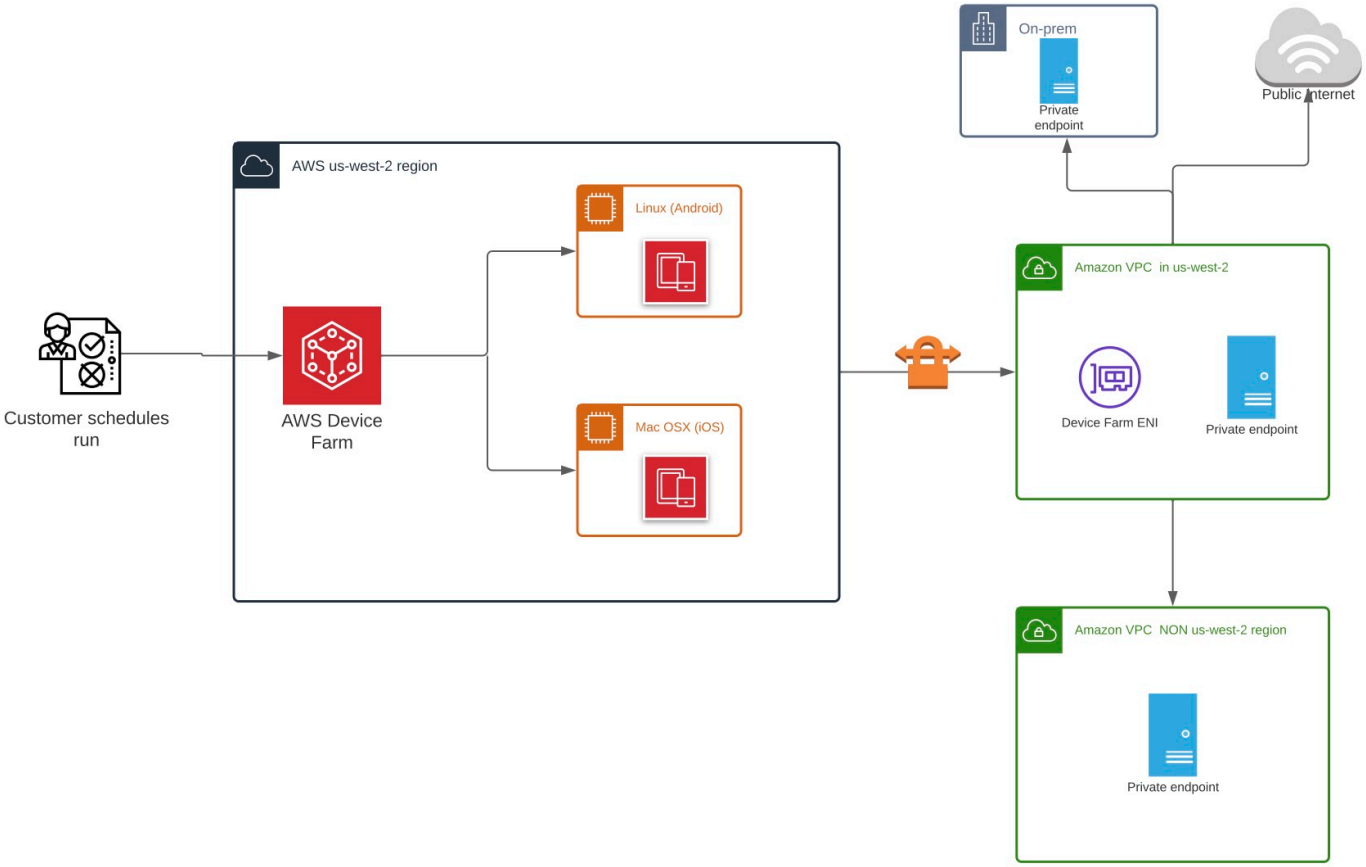
VPC-ENI di AWS Device Farm

> **Warning**
>
> Fitur ini hanya tersedia di perangkat pribadi. Untuk meminta penggunaan perangkat pribadi di AWS akun Anda, silakan hubungi kami. Jika Anda sudah memiliki perangkat pribadi yang ditambahkan ke AWS akun Anda, kami sangat menyarankan untuk menggunakan metode konektivitas VPC ini.

Fitur konektivitas VPC-ENI AWS Device Farm membantu pelanggan terhubung dengan aman ke titik akhir pribadi mereka yang dihosting di AWS, perangkat lunak lokal, atau penyedia cloud lainnya.

Anda dapat menghubungkan perangkat seluler Device Farm dan mesin hostnya ke lingkungan Amazon Virtual Private Cloud (Amazon VPC) di `us-west-2` Wilayah ini, yang memungkinkan akses ke non-internet-facing layanan, dan aplikasi yang terisolasi melalui antarmuka jaringan elastis. Untuk informasi selengkapnya VPCs, lihat Panduan Pengguna Amazon VPC.

Jika titik akhir pribadi atau VPC Anda tidak ada di `us-west-2` Wilayah, Anda dapat menautkannya dengan VPC di `us-west-2` Wilayah menggunakan solusi seperti Transit Gateway atau VPC Peering. Dalam situasi seperti itu, Device Farm akan membuat ENI di subnet yang Anda sediakan untuk VPC `us-west-2` Wilayah Anda, dan Anda akan bertanggung jawab untuk memastikan bahwa koneksi dapat dibuat antara VPC `us-west-2` Wilayah dan VPC di Wilayah lain.



Untuk informasi tentang penggunaan AWS CloudFormation untuk membuat dan mengintip secara otomatis VPCs, lihat VPCPeering templat di repositori AWS CloudFormation templat. GitHub

> **Note**
>
> Device Farm tidak mengenakan biaya apa pun untuk membuat ENIs di VPC pelanggan. `us-west-2` Biaya untuk konektivitas antar VPC lintas wilayah atau eksternal tidak termasuk dalam fitur ini.

Setelah Anda mengonfigurasi akses VPC, perangkat dan mesin host yang Anda gunakan untuk pengujian Anda tidak akan dapat terhubung ke sumber daya di luar VPC (misalnya, publik CDNs)

kecuali ada gateway NAT yang Anda tentukan dalam VPC. Untuk informasi lebih lanjut, lihat Gateway NAT dalam Panduan Pengguna Amazon VPC.

Topik

AWS kontrol akses dan IAM

AWS Device Farm memungkinkan Anda menggunakan AWS Identity and Access Management(IAM) untuk membuat kebijakan yang memberikan atau membatasi akses ke fitur Device Farm. Untuk menggunakan fitur Konektivitas VPC dengan AWS Device Farm, Kebijakan IAM berikut diperlukan untuk akun pengguna atau peran yang Anda gunakan untuk mengakses AWS Device Farm:

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [{
      "Effect": "Allow",
      "Action": [
        "devicefarm:*",
        "ec2:DescribeVpcs",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups",
        "ec2:CreateNetworkInterface"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "iam:CreateServiceLinkedRole",
```

```
      "Resource": "arn:aws:iam::*:role/aws-service-role/devicefarm.amazonaws.com/
AWSServiceRoleForDeviceFarm",
      "Condition": {
        "StringLike": {
          "iam:AWSServiceName": "devicefarm.amazonaws.com"
        }
      }
    }
  ]
}
```

Untuk membuat atau memperbarui proyek Device Farm dengan konfigurasi VPC, kebijakan IAM Anda harus mengizinkan Anda memanggil tindakan berikut terhadap sumber daya yang tercantum dalam konfigurasi VPC:

```
"ec2:DescribeVpcs"
"ec2:DescribeSubnets"
"ec2:DescribeSecurityGroups"
"ec2:CreateNetworkInterface"
```

Selain itu, kebijakan IAM Anda juga harus mengizinkan pembuatan peran terkait layanan:

```
"iam:CreateServiceLinkedRole"
```

> **Note**
>
> Tak satu pun dari izin ini diperlukan untuk pengguna yang tidak menggunakan konfigurasi VPC dalam proyek mereka.

Peran terkait layanan

AWS Device Farm menggunakan AWS Identity and Access Management peran terkait layanan (IAM). Peran terkait layanan adalah jenis peran IAM unik yang ditautkan langsung ke Device Farm. Peran terkait layanan telah ditentukan sebelumnya oleh Device Farm dan menyertakan semua izin yang diperlukan layanan untuk memanggil AWS layanan lain atas nama Anda.

Peran terkait layanan membuat pengaturan Device Farm lebih mudah karena Anda tidak perlu menambahkan izin yang diperlukan secara manual. Device Farm mendefinisikan izin peran terkait

layanan, dan kecuali ditentukan lain, hanya Device Farm yang dapat menjalankan perannya. Izin yang ditentukan mencakup kebijakan kepercayaan dan kebijakan izin, dan kebijakan izin tersebut tidak dapat dilampirkan ke entitas IAM lainnya.

Anda dapat menghapus peran tertaut layanan hanya setelah menghapus sumber daya terkait terlebih dahulu. Ini melindungi sumber daya Device Farm karena Anda tidak dapat secara tidak sengaja menghapus izin untuk mengakses sumber daya.

Untuk informasi tentang layanan lain yang mendukung peran yang terhubung dengan layanan, lihat Layanan AWS yang Berfungsi dengan IAM dan cari layanan yang memiliki Ya di kolom Peran yang Terhubung dengan Layanan. Pilih Ya dengan tautan untuk melihat dokumentasi peran tertaut layanan untuk layanan tersebut.

Izin peran terkait layanan untuk Device Farm

Device Farm menggunakan peran terkait layanan bernama AWSServiceRoleForDeviceFarm—Memungkinkan Device Farm mengakses sumber daya AWS atas nama Anda.

Peran AWSService RoleForDeviceFarm terkait layanan mempercayai layanan berikut untuk mengambil peran:

- `devicefarm.amazonaws.com`

Kebijakan izin peran memungkinkan Device Farm menyelesaikan tindakan berikut:

- Untuk akun Anda
  - Buat antarmuka jaringan
  - Jelaskan antarmuka jaringan
  - Jelaskan VPCs
  - Jelaskan subnet
  - Jelaskan kelompok keamanan
  - Hapus antarmuka
  - Memodifikasi antarmuka jaringan
- Untuk antarmuka jaringan
  - Buat tag
- Untuk antarmuka jaringan EC2 yang dikelola oleh Device Farm

- Buat izin antarmuka jaringan

Kebijakan IAM lengkap berbunyi:

JSON

```
{
"Version":"2012-10-17",
"Statement": [
 {
  "Effect": "Allow",
  "Action": [
   "ec2:DescribeNetworkInterfaces",
   "ec2:DescribeVpcs",
   "ec2:DescribeSubnets",
   "ec2:DescribeSecurityGroups"
  ],
  "Resource": "*"
 },
 {
  "Effect": "Allow",
  "Action": [
   "ec2:CreateNetworkInterface"
  ],
  "Resource": [
   "arn:aws:ec2:*:*:subnet/*",
   "arn:aws:ec2:*:*:security-group/*"
  ]
 },
 {
  "Effect": "Allow",
  "Action": [
   "ec2:CreateNetworkInterface"
  ],
  "Resource": [
   "arn:aws:ec2:*:*:network-interface/*"
  ],
  "Condition": {
   "StringEquals": {
    "aws:RequestTag/AWSDeviceFarmManaged": "true"
   }
  }
```

```
},
{
 "Effect": "Allow",
 "Action": [
  "ec2:CreateTags"
 ],
 "Resource": "arn:aws:ec2:*:*:network-interface/*",
 "Condition": {
  "StringEquals": {
   "ec2:CreateAction": "CreateNetworkInterface"
  }
 }
},
{
 "Effect": "Allow",
 "Action": [
  "ec2:CreateNetworkInterfacePermission",
  "ec2:DeleteNetworkInterface"
 ],
 "Resource": "arn:aws:ec2:*:*:network-interface/*",
 "Condition": {
  "StringEquals": {
   "aws:ResourceTag/AWSDeviceFarmManaged": "true"
  }
 }
},
{
 "Effect": "Allow",
 "Action": [
  "ec2:ModifyNetworkInterfaceAttribute"
 ],
 "Resource": [
  "arn:aws:ec2:*:*:security-group/*",
  "arn:aws:ec2:*:*:instance/*"
 ]
},
{
 "Effect": "Allow",
 "Action": [
  "ec2:ModifyNetworkInterfaceAttribute"
 ],
 "Resource": "arn:aws:ec2:*:*:network-interface/*",
 "Condition": {
  "StringEquals": {
```

```
      "aws:ResourceTag/AWSDeviceFarmManaged": "true"
    }
   }
  }
 ]
}
```

Anda harus mengonfigurasi izin untuk mengizinkan entitas IAM (seperti pengguna, grup, atau peran) untuk membuat, mengedit, atau menghapus peran terkait layanan. Untuk informasi selengkapnya, silakan lihat Izin Peran Tertaut Layanan di Panduan Pengguna IAM.

Membuat peran terkait layanan untuk Device Farm

Saat Anda menyediakan konfigurasi VPC untuk proyek pengujian seluler, Anda tidak perlu membuat peran terkait layanan secara manual. Saat Anda membuat resource Device Farm pertama Anda di Konsol Manajemen AWS, API AWS CLI, atau AWS API, Device Farm membuat peran terkait layanan untuk Anda.

Jika Anda menghapus peran terkait layanan ini, dan ingin membuatnya lagi, Anda dapat mengulangi proses yang sama untuk membuat kembali peran tersebut di akun Anda. Saat Anda membuat sumber daya Device Farm pertama Anda, Device Farm membuat peran terkait layanan untuk Anda lagi.

Anda juga dapat menggunakan konsol IAM untuk membuat peran terkait layanan dengan kasus penggunaan Device Farm. Di AWS CLI atau AWS API, buat peran terkait layanan dengan nama `devicefarm.amazonaws.com` layanan. Untuk informasi lebih lanjut, lihat Membuat Peran yang Terhubung dengan Layanan di Panduan Pengguna IAM. Jika Anda menghapus peran tertaut layanan ini, Anda dapat mengulang proses yang sama untuk membuat peran tersebut lagi.

Mengedit peran terkait layanan untuk Device Farm

Device Farm tidak mengizinkan Anda mengedit peran AWSService RoleForDeviceFarm terkait layanan. Setelah membuat peran terkait layanan, Anda tidak dapat mengubah nama peran karena berbagai entitas mungkin merujuk peran tersebut. Namun, Anda dapat mengedit penjelasan peran menggunakan IAM. Untuk informasi selengkapnya, lihat Mengedit Peran Tertaut Layanan dalam Panduan Pengguna IAM.

Menghapus peran terkait layanan untuk Device Farm

Jika Anda tidak perlu lagi menggunakan fitur atau layanan yang memerlukan peran terkait layanan, sebaiknya hapus peran tersebut. Dengan begitu, Anda tidak memiliki entitas yang tidak digunakan yang tidak dipantau atau dipelihara secara aktif. Tetapi, Anda harus membersihkan sumber daya peran yang terhubung dengan layanan sebelum menghapusnya secara manual.

> **Note**
>
> Jika layanan Device Farm menggunakan peran saat Anda mencoba menghapus sumber daya, penghapusan mungkin gagal. Jika hal itu terjadi, tunggu beberapa menit dan coba mengoperasikannya lagi.

Untuk menghapus peran tertaut layanan secara manual menggunakan IAM

Gunakan konsol IAM, the AWS CLI, atau AWS API untuk menghapus peran AWSService RoleForDeviceFarm terkait layanan. Untuk informasi selengkapnya, silakan lihat Menghapus Peran Terkait Layanan di Panduan Pengguna IAM.

Wilayah yang Didukung untuk peran terkait layanan Device Farm

Device Farm mendukung penggunaan peran terkait layanan di semua wilayah tempat layanan tersedia. Untuk informasi selengkapnya, lihat AWS Wilayah dan Titik Akhir.

Device Farm tidak mendukung penggunaan peran terkait layanan di setiap wilayah tempat layanan tersedia. Anda dapat menggunakan AWSService RoleForDeviceFarm peran di wilayah berikut.

Nama wilayah	Identitas wilayah	Support di Device Farm
US East (Northern Virginia)	us-east-1	Tidak
AS Timur (Ohio)	us-east-2	Tidak
AS Barat (California Utara)	us-west-1	Tidak
AS Barat (Oregon)	us-west-2	Ya
Asia Pacific (Mumbai)	ap-south-1	Tidak

Nama wilayah	Identitas wilayah	Support di Device Farm
Asia Pacific (Osaka)	ap-northeast-3	Tidak
Asia Pasifik (Seoul)	ap-northeast-2	Tidak
Asia Pasifik (Singapura)	ap-southeast-1	Tidak
Asia Pasifik (Sydney)	ap-southeast-2	Tidak
Asia Pasifik (Tokyo)	ap-northeast-1	Tidak
Kanada (Pusat)	ca-central-1	Tidak
Eropa (Frankfurt)	eu-central-1	Tidak
Eropa (Irlandia)	eu-west-1	Tidak
Eropa (London)	eu-west-2	Tidak
Eropa (Paris)	eu-west-3	Tidak
Amerika Selatan (Sao Paulo)	sa-east-1	Tidak
AWS GovCloud (US)	us-gov-west-1	Tidak

Prasyarat

Daftar berikut menjelaskan beberapa persyaratan dan saran untuk ditinjau saat membuat konfigurasi VPC-ENI:

- Perangkat pribadi harus ditetapkan ke AWS Akun Anda.
- Anda harus memiliki pengguna AWS akun atau peran dengan izin untuk membuat peran terkait Layanan. Saat menggunakan titik akhir Amazon VPC dengan fitur pengujian seluler Device Farm, Device Farm membuat peran terkait layanan AWS Identity and Access Management (IAM).
- Device Farm VPCs hanya dapat terhubung di `us-west-2` Wilayah. Jika Anda tidak memiliki VPC di `us-west-2` Wilayah, Anda harus membuatnya. Kemudian, untuk mengakses sumber daya di VPC di Wilayah lain, Anda harus membuat koneksi peering antara VPC di Wilayah `us-west-2`

dan VPC di Wilayah lain. Untuk informasi tentang mengintip VPCs, lihat Panduan Peering VPC Amazon.

Anda harus memverifikasi bahwa Anda memiliki akses ke VPC yang Anda tentukan saat Anda mengonfigurasi koneksi. Anda harus mengonfigurasi izin Amazon Elastic Compute Cloud (Amazon EC2) tertentu untuk Device Farm.

- Resolusi DNS diperlukan dalam VPC yang Anda gunakan.
- Setelah VPC Anda dibuat, Anda memerlukan informasi berikut tentang VPC di Wilayah: `us-west-2`
  - VPC ID
  - Subnet IDs (hanya subnet pribadi)
  - Kelompok keamanan IDs
- Anda harus mengonfigurasi koneksi Amazon VPC berdasarkan per proyek. Pada saat ini, Anda hanya dapat mengonfigurasi satu konfigurasi VPC per proyek. Saat Anda mengonfigurasi VPC, Amazon VPC membuat antarmuka di dalam VPC Anda dan menetapkannya ke subnet dan grup keamanan yang ditentukan. Semua sesi future yang terkait dengan proyek akan menggunakan koneksi VPC yang dikonfigurasi.
- Anda tidak dapat menggunakan konfigurasi VPC-ENI bersama dengan fitur VPCE lama.
- Kami sangat menyarankan untuk tidak memperbarui proyek yang ada dengan konfigurasi VPC-ENI karena proyek yang ada mungkin memiliki pengaturan VPCE yang bertahan pada tingkat proses. Sebaliknya, jika Anda sudah menggunakan fitur VPCE yang ada, gunakan VPC-ENI untuk semua proyek baru.

Menghubungkan ke Amazon VPC

Anda dapat mengonfigurasi dan memperbarui proyek Anda untuk menggunakan titik akhir Amazon VPC. Konfigurasi VPC-ENI dikonfigurasi berdasarkan per proyek. Sebuah proyek hanya dapat memiliki satu titik akhir VPC-ENI pada waktu tertentu. Untuk mengonfigurasi akses VPC untuk suatu proyek, Anda harus mengetahui detail berikut:

- ID VPC masuk `us-west-2` jika aplikasi Anda di-host di sana atau ID VPC yang terhubung ke beberapa `us-west-2` VPC lain di Wilayah yang berbeda.
- Grup keamanan yang berlaku untuk diterapkan pada koneksi.

- Subnet yang akan dikaitkan dengan koneksi. Ketika sesi dimulai, subnet terbesar yang tersedia digunakan. Kami merekomendasikan memiliki beberapa subnet yang terkait dengan zona ketersediaan yang berbeda untuk meningkatkan postur ketersediaan konektivitas VPC Anda.
- Saat menggunakan VPC-ENI, resolver DNS yang digunakan oleh host dan perangkat uji Device Farm akan menjadi server yang disediakan oleh layanan DHCP di subnet pelanggan. Dalam konfigurasi default, ini akan menjadi resolver default VPC. Pelanggan yang ingin menentukan resolver DNS kustom dapat mengonfigurasi Set Opsi DHCP di VPC mereka.

Setelah Anda membuat konfigurasi VPC-ENI, Anda dapat memperbarui detailnya menggunakan konsol atau CLI menggunakan langkah-langkah di bawah ini.

Console

1. Masuk ke konsol Device Farm di https://console.aws.amazon.com/devicefarm.
2. Pada panel navigasi Device Farm, pilih Pengujian Perangkat Seluler, lalu pilih Proyek.
3. Di bawah proyek Pengujian Seluler, pilih nama proyek Anda dari daftar.
4. Pilih Pengaturan proyek.
5. Di bagian Pengaturan Virtual Private Cloud (VPC), Anda dapat mengubah, `Subnets` (subnet pribadi VPC saja), dan. `Security Groups`
6. Pilih Simpan.

CLI

Gunakan perintah AWS CLI berikut untuk memperbarui 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
```

Anda juga dapat mengonfigurasi VPC Amazon saat membuat proyek Anda:

```
$  aws devicefarm create-project \
```

```
--name VPCDemo \
--vpc-config \
securityGroupIds=sg-02c1537701a7e3763,sg-005dadf9311efda25,\
subnetIds=subnet-09b1a45f9cac53717,subnet-09b1a45f9cac12345,\
vpcId=vpc-0238fb322af81a368
```

Batas

Batasan berikut berlaku untuk fitur VPC-ENI:

- Anda dapat menyediakan hingga lima grup keamanan dalam konfigurasi VPC proyek Device Farm.
- Anda dapat menyediakan hingga delapan subnet dalam konfigurasi VPC proyek Device Farm.
- Saat mengonfigurasi proyek Device Farm agar berfungsi dengan VPC Anda, subnet terkecil yang dapat Anda berikan harus memiliki minimal lima alamat yang tersedia. IPv4
- Alamat IP publik tidak didukung saat ini. Sebagai gantinya, kami menyarankan Anda menggunakan subnet pribadi dalam proyek Device Farm Anda. Jika Anda membutuhkan akses internet publik selama pengujian, gunakan gateway terjemahan alamat jaringan (NAT). Mengonfigurasi proyek Device Farm dengan subnet publik tidak memberikan akses internet pengujian Anda atau alamat IP publik.
- Integrasi VPC-ENI hanya mendukung subnet pribadi di VPC Anda.
- Hanya lalu lintas keluar dari ENI yang dikelola layanan yang didukung. Ini berarti bahwa ENI tidak dapat menerima permintaan masuk yang tidak diminta dari VPC.

Menggunakan layanan endpoint Amazon VPC dengan Device Farm - Legacy (tidak disarankan)

> **Warning**
>
> Kami sangat menyarankan untuk menggunakan konektivitas VPC-ENI yang dijelaskan di halaman ini untuk konektivitas titik akhir pribadi karena VPCE sekarang dianggap sebagai fitur lama. VPC-ENI memberikan lebih banyak fleksibilitas, konfigurasi yang lebih sederhana, lebih hemat biaya, dan membutuhkan overhead pemeliharaan yang jauh lebih sedikit jika dibandingkan dengan metode konektivitas VPCE.

> **Note**
>
> Menggunakan Amazon VPC Endpoint Services dengan Device Farm hanya didukung untuk pelanggan dengan perangkat pribadi yang dikonfigurasi. Untuk mengaktifkan akun AWS Anda menggunakan fitur ini dengan perangkat pribadi, silakan hubungi kami.

Amazon Virtual Private Cloud (Amazon VPC) adalah AWS layanan yang dapat Anda gunakan untuk meluncurkan AWS sumber daya di jaringan virtual yang Anda tentukan. Dengan VPC, Anda memiliki kontrol atas pengaturan jaringan Anda, seperti rentang alamat IP, subnet, tabel perutean, dan gateway jaringan.

Jika Anda menggunakan Amazon VPC untuk meng-host aplikasi pribadi di AWS Wilayah AS Barat (Oregon) (`us-west-2`), Anda dapat membuat koneksi pribadi antara VPC dan Device Farm. Dengan koneksi ini, Anda dapat menggunakan Device Farm untuk menguji aplikasi pribadi tanpa mengeksposnya melalui internet publik. Untuk mengaktifkan AWS akun Anda menggunakan fitur ini dengan perangkat pribadi, hubungi kami.

Untuk menghubungkan sumber daya di VPC ke Device Farm, Anda dapat menggunakan konsol Amazon VPC untuk membuat layanan titik akhir VPC. Layanan endpoint ini memungkinkan Anda menyediakan sumber daya di VPC ke Device Farm melalui titik akhir VPC Device Farm. Layanan endpoint menyediakan konektivitas yang andal dan dapat diskalakan ke Device Farm tanpa memerlukan gateway internet, instance terjemahan alamat jaringan (NAT), atau koneksi VPN. Untuk informasi selengkapnya, lihat layanan titik akhir VPC (AWS PrivateLink) di Panduan.AWS PrivateLink

> **Important**
>
> Fitur titik akhir VPC Device Farm membantu Anda menghubungkan layanan internal pribadi secara aman di VPC Anda ke VPC publik Device Farm dengan menggunakan koneksi. AWS PrivateLink Meskipun koneksi aman dan pribadi, keamanan itu tergantung pada perlindungan Anda atas AWS kredensi Anda. Jika AWS kredensi Anda dikompromikan, penyerang dapat mengakses atau mengekspos data layanan Anda ke dunia luar.

Setelah membuat layanan titik akhir VPC di Amazon VPC, Anda dapat menggunakan konsol Device Farm untuk membuat konfigurasi titik akhir VPC di Device Farm. Topik ini menunjukkan cara membuat koneksi VPC Amazon dan konfigurasi titik akhir VPC di Device Farm.

Sebelum Anda mulai

Informasi berikut adalah untuk pengguna VPC Amazon di Wilayah AS Barat (Oregon) (`us-west-2`), dengan subnet di masing-masing Availability Zone berikut: us-west-2a, us-west-2b, dan us-west-2c.

Device Farm memiliki persyaratan tambahan untuk layanan endpoint VPC yang dapat Anda gunakan. Saat membuat dan mengonfigurasi layanan titik akhir VPC agar berfungsi dengan Device Farm, pastikan Anda memilih opsi yang memenuhi persyaratan berikut:

- Availability Zone untuk layanan harus mencakup us-west-2a, us-west-2b, dan us-west-2c. Network Load Balancer yang terkait dengan layanan endpoint VPC menentukan Availability Zones untuk layanan endpoint VPC tersebut. Jika layanan titik akhir VPC Anda tidak menampilkan ketiga Availability Zone ini, Anda harus membuat ulang Network Load Balancer untuk mengaktifkan ketiga zona ini, lalu mengasosiasikan kembali Network Load Balancer dengan layanan endpoint Anda.
- Prinsipal yang diizinkan untuk layanan titik akhir harus menyertakan Nama Sumber Daya Amazon (ARN) dari titik akhir VPC Device Farm (ARN layanan). Setelah Anda membuat layanan endpoint, tambahkan ARN layanan endpoint VPC Device Farm ke daftar izin untuk memberikan izin kepada Device Farm untuk mengakses layanan endpoint VPC Anda. Untuk mendapatkan layanan endpoint Device Farm VPC ARN, hubungi kami.

Selain itu, jika Anda tetap mengaktifkan pengaturan Acceptance required saat membuat layanan endpoint VPC, Anda harus secara manual menerima setiap permintaan koneksi yang dikirim Device Farm ke layanan endpoint. Untuk mengubah setelan ini untuk layanan titik akhir yang ada, pilih layanan titik akhir di konsol VPC Amazon, pilih Tindakan, lalu pilih Ubah setelan penerimaan titik akhir. Untuk informasi selengkapnya, lihat Mengubah penyeimbang beban dan setelan penerimaan di Panduan.AWS PrivateLink

Bagian selanjutnya menjelaskan cara membuat layanan endpoint VPC Amazon yang memenuhi persyaratan ini.

Langkah 1: Membuat Network Load Balancer

Langkah pertama dalam membangun koneksi pribadi antara VPC dan Device Farm Anda adalah membuat Network Load Balancer untuk merutekan permintaan ke grup target.

New console

Untuk membuat Network Load Balancer menggunakan konsol baru

1. Buka konsol Amazon Elastic Compute Cloud (Amazon EC2) di. https://console.aws.amazon.com/ec2/
2. Di panel navigasi, di bawah Load balancing, pilih Load balancer.
3. Pilih Buat Penyeimbang Beban.
4. Di bawah Network load balancer, pilih Create.
5. Pada halaman Buat penyeimbang beban jaringan, di bawah konfigurasi Dasar, lakukan hal berikut:
   a. Masukkan Nama penyeimbang beban.
   b. Untuk Skema, pilih Internal.
6. Di bawah Pemetaan jaringan, lakukan hal berikut:
   a. Pilih VPC untuk grup target Anda.
   b. Pilih Pemetaan berikut:
      - `us-west-2a`
      - `us-west-2b`
      - `us-west-2c`
7. Di bawah Pendengar dan perutean, gunakan opsi Protokol dan Port untuk memilih grup target Anda.

   > **Note**
   >
   > Secara default, penyeimbangan beban zona ketersediaan silang dinonaktifkan. Karena penyeimbang beban menggunakan Availability Zones`us-west-2a`,`us-west-2b`, dan`us-west-2c`, itu memerlukan target untuk didaftarkan di masing-masing Availability Zone tersebut, atau, jika Anda mendaftarkan target di kurang dari ketiga zona, itu mengharuskan Anda mengaktifkan penyeimbangan beban lintas zona. Jika tidak, penyeimbang beban mungkin tidak berfungsi seperti yang diharapkan.

8. Pilih Buat Penyeimbang Beban.

Old console

Untuk membuat Network Load Balancer menggunakan konsol lama

1. Buka konsol Amazon Elastic Compute Cloud (Amazon EC2) di. https://console.aws.amazon.com/ec2/
2. Di panel navigasi, di bawah Load balancing, pilih load balancer.
3. Pilih Buat Penyeimbang Beban.
4. Di bawah Network load balancer, pilih Create.
5. Pada halaman Configure load balancer, di bawah konfigurasi Basic, lakukan hal berikut:
   a. Masukkan Nama penyeimbang beban.
   b. Untuk Skema, pilih Internal.
6. Di bawah Pendengar, pilih Protokol dan Port yang digunakan grup target Anda.
7. Di bawah Availability zone, lakukan hal berikut:
   a. Pilih VPC untuk grup target Anda.
   b. Pilih zona Ketersediaan berikut:
      - `us-west-2a`
      - `us-west-2b`
      - `us-west-2c`
   c. Pilih Berikutnya: konfigurasikan pengaturan keamanan.
8. (Opsional) Konfigurasikan pengaturan keamanan Anda, lalu pilih Berikutnya: konfigurasikan perutean.
9. Pada halaman Configure Routing, lakukan hal berikut:
   a. Untuk Grup target, pilih Grup target yang ada.
   b. Untuk Nama, pilih grup target Anda.
   c. Pilih Berikutnya: daftarkan target.
10. Pada halaman Daftar target, tinjau target Anda, lalu pilih Berikutnya: tinjau.

> **Note**
>
> Secara default, penyeimbangan beban zona ketersediaan silang dinonaktifkan. Karena penyeimbang beban menggunakan Availability Zones`us-west-2a`,`us-west-2b`, dan`us-west-2c`, itu memerlukan target untuk didaftarkan di masing-masing Availability Zone tersebut, atau, jika Anda mendaftarkan target di kurang dari ketiga zona, itu mengharuskan Anda mengaktifkan penyeimbangan beban

lintas zona. Jika tidak, penyeimbang beban mungkin tidak berfungsi seperti yang diharapkan.

11. Tinjau konfigurasi penyeimbang beban Anda, lalu pilih Buat.

Langkah 2: Membuat layanan endpoint Amazon VPC

Setelah membuat Network Load Balancer, gunakan konsol Amazon VPC untuk membuat layanan endpoint di VPC Anda.

1. Buka konsol Amazon VPC di. https://console.aws.amazon.com/vpc/
2. Di bawah Sumber daya menurut wilayah, pilih Layanan titik akhir.
3. Pilih Buat layanan endpoint.
4. Lakukan salah satu tindakan berikut:
   - Jika Anda sudah memiliki Network Load Balancer yang ingin digunakan layanan endpoint, pilih di bawah Penyeimbang beban yang tersedia, lalu lanjutkan ke langkah 5.
   - Jika Anda belum membuat Network Load Balancer, pilih Buat penyeimbang beban baru. Konsol Amazon EC2 terbuka. Ikuti langkah-langkah dalam Membuat Network Load Balancer dimulai dengan langkah 3, lalu lanjutkan dengan langkah-langkah ini di konsol Amazon VPC.
5. Untuk zona ketersediaan yang disertakan, verifikasi itu `us-west-2aus-west-2b`,, dan `us-west-2c` muncul dalam daftar.
6. Jika Anda tidak ingin secara manual menerima atau menolak setiap permintaan koneksi yang dikirim ke layanan endpoint, di bawah Pengaturan tambahan, hapus Penerimaan diperlukan. Jika Anda menghapus kotak centang ini, layanan endpoint secara otomatis menerima setiap permintaan koneksi yang diterimanya.
7. Pilih Buat.
8. Di layanan endpoint baru, pilih Izinkan prinsipal.
9. Hubungi kami untuk mendapatkan ARN dari titik akhir VPC Device Farm (layanan ARN) untuk ditambahkan ke daftar izinkan untuk layanan endpoint, dan kemudian tambahkan layanan ARN itu ke daftar izin untuk layanan tersebut.
10. Pada tab Detail untuk layanan titik akhir, buat catatan nama layanan (nama layanan). Anda memerlukan nama ini saat membuat konfigurasi titik akhir VPC di langkah berikutnya.

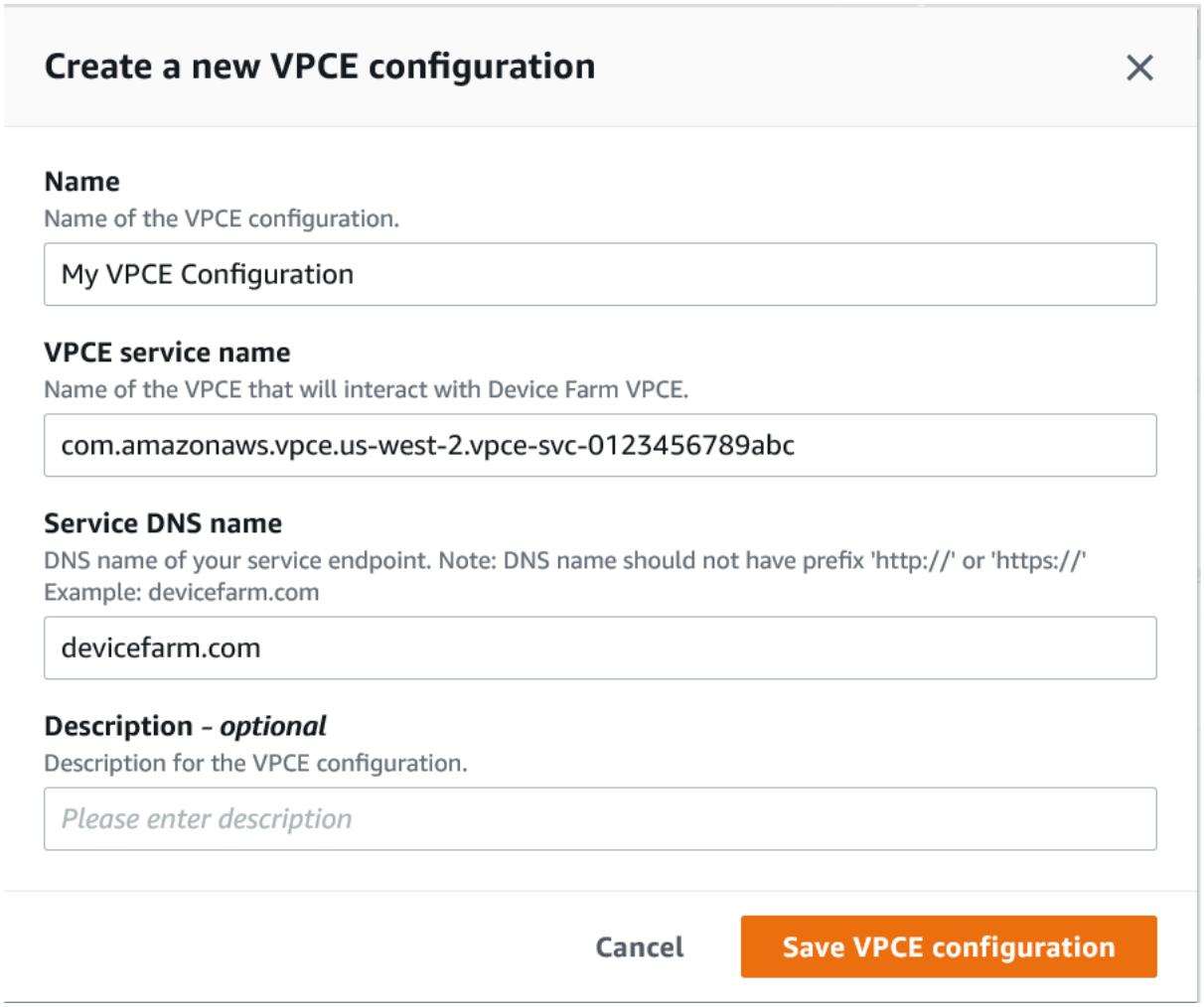
Layanan endpoint VPC Anda sekarang siap digunakan dengan Device Farm.

Langkah 3: Membuat konfigurasi titik akhir VPC di Device Farm

Setelah membuat layanan endpoint di Amazon VPC, Anda dapat membuat konfigurasi endpoint Amazon VPC di Device Farm.

1. Masuk ke konsol Device Farm di https://console.aws.amazon.com/devicefarm.
2. Di panel navigasi, pilih Pengujian perangkat seluler, lalu Perangkat pribadi.
3. Pilih konfigurasi VPCE.
4. Pilih Buat konfigurasi VPCE.
5. Di bawah Buat konfigurasi VPCE baru, masukkan Nama untuk konfigurasi titik akhir VPC.
6. Untuk nama layanan VPCE, masukkan nama layanan titik akhir VPC Amazon (nama layanan) yang Anda catat di konsol VPC Amazon. Namanya terlihat seperti`com.amazonaws.vpce.us-west-2.vpce-svc-id`.
7. Untuk nama DNS Layanan, masukkan nama DNS layanan untuk aplikasi yang ingin Anda uji (misalnya,`devicefarm.com`). Jangan tentukan `http` atau `https` sebelum nama DNS layanan.

   Nama domain tidak dapat diakses melalui internet publik. Selain itu, nama domain baru ini, yang dipetakan ke layanan titik akhir VPC Anda, dibuat oleh Amazon Route 53 dan tersedia secara eksklusif untuk Anda di sesi Device Farm Anda.
8. Pilih Simpan.



Langkah 4: Membuat uji coba

Setelah menyimpan konfigurasi titik akhir VPC, Anda dapat menggunakan konfigurasi untuk membuat pengujian berjalan atau mengakses sesi dari jarak jauh. Untuk informasi selengkapnya, lihat Membuat uji coba di Device Farm atau Membuat sesi.

Mencatat panggilan AWS Device Farm API dengan AWS CloudTrail

AWS Device Farm terintegrasi dengan AWS CloudTrail, layanan yang menyediakan catatan tindakan yang diambil oleh pengguna, peran, atau AWS layanan di AWS Device Farm. CloudTrail menangkap semua panggilan API untuk AWS Device Farm sebagai peristiwa. Panggilan yang diambil mencakup panggilan dari konsol AWS Device Farm dan panggilan kode ke operasi AWS Device Farm API. Jika Anda membuat jejak, Anda dapat mengaktifkan pengiriman CloudTrail peristiwa secara berkelanjutan ke bucket Amazon S3, termasuk peristiwa untuk AWS Device Farm. Jika Anda tidak mengonfigurasi jejak, Anda masih dapat melihat peristiwa terbaru di CloudTrail konsol dalam Riwayat acara. Dengan menggunakan informasi yang dikumpulkan CloudTrail, Anda dapat menentukan permintaan yang dibuat ke AWS Device Farm, alamat IP tempat permintaan dibuat, siapa yang mengajukan permintaan, kapan permintaan dibuat, dan detail tambahan.

Untuk mempelajari selengkapnya CloudTrail, lihat Panduan AWS CloudTrail Pengguna.

Informasi AWS Device Farm di CloudTrail

CloudTrail diaktifkan di AWS akun Anda saat Anda membuat akun. Saat aktivitas terjadi di AWS Device Farm, aktivitas tersebut direkam dalam suatu CloudTrail peristiwa bersama dengan peristiwa AWS layanan lainnya dalam riwayat Acara. Anda dapat melihat, mencari, dan mengunduh acara terbaru di AWS akun Anda. Untuk informasi selengkapnya, lihat Melihat Acara dengan Riwayat CloudTrail Acara.

Untuk catatan peristiwa yang sedang berlangsung di AWS akun Anda, termasuk peristiwa untuk AWS Device Farm, buat jejak. Jejak memungkinkan CloudTrail untuk mengirimkan file log ke bucket Amazon S3. Secara default, ketika Anda membuat jejak di konsol tersebut, jejak tersebut diterapkan ke semua Wilayah AWS. Jejak mencatat peristiwa dari semua Wilayah di AWS partisi dan mengirimkan file log ke bucket Amazon S3 yang Anda tentukan. Selain itu, Anda dapat mengonfigurasi AWS layanan lain untuk menganalisis lebih lanjut dan menindaklanjuti data peristiwa yang dikumpulkan dalam CloudTrail log. Untuk informasi selengkapnya, lihat berikut:

- Gambaran Umum untuk Membuat Jejak
- CloudTrail Layanan dan Integrasi yang Didukung
- Mengkonfigurasi Notifikasi Amazon SNS untuk CloudTrail

- Menerima File CloudTrail Log dari Beberapa Wilayah dan Menerima File CloudTrail Log dari Beberapa Akun

Saat CloudTrail logging diaktifkan di AWS akun Anda, panggilan API yang dilakukan ke tindakan Device Farm dilacak dalam file log. Catatan Device Farm ditulis bersama dengan catatan AWS layanan lain dalam file log. CloudTrail menentukan kapan harus membuat dan menulis ke file baru berdasarkan periode waktu dan ukuran file.

Semua tindakan Device Farm dicatat dan didokumentasikan dalam AWS CLI referensi dan fileMengotomatisasi Device Farm. Misalnya, panggilan untuk membuat proyek baru atau berjalan di Device Farm menghasilkan entri dalam file CloudTrail log.

Setiap entri peristiwa atau log berisi informasi tentang siapa yang membuat permintaan tersebut. Informasi identitas membantu Anda menentukan berikut ini:

- Apakah permintaan itu dibuat dengan kredenal pengguna root atau AWS Identity and Access Management (IAM).
- Apakah permintaan tersebut dibuat dengan kredensial keamanan sementara untuk satu peran atau pengguna gabungan.
- Apakah permintaan itu dibuat oleh AWS layanan lain.

Untuk informasi lain, lihat Elemen userIdentity CloudTrail .

Memahami entri file log AWS Device Farm

Trail adalah konfigurasi yang memungkinkan pengiriman peristiwa sebagai file log ke bucket Amazon S3 yang Anda tentukan. CloudTrail file log berisi satu atau lebih entri log. Peristiwa mewakili permintaan tunggal dari sumber mana pun dan mencakup informasi tentang tindakan yang diminta, tanggal dan waktu tindakan, parameter permintaan, dan sebagainya. CloudTrail file log bukanlah jejak tumpukan yang diurutkan dari panggilan API publik, jadi file tersebut tidak muncul dalam urutan tertentu.

Contoh berikut menunjukkan entri CloudTrail log yang menunjukkan `ListRuns` tindakan Device Farm:

```
{
  "Records": [
    {
```

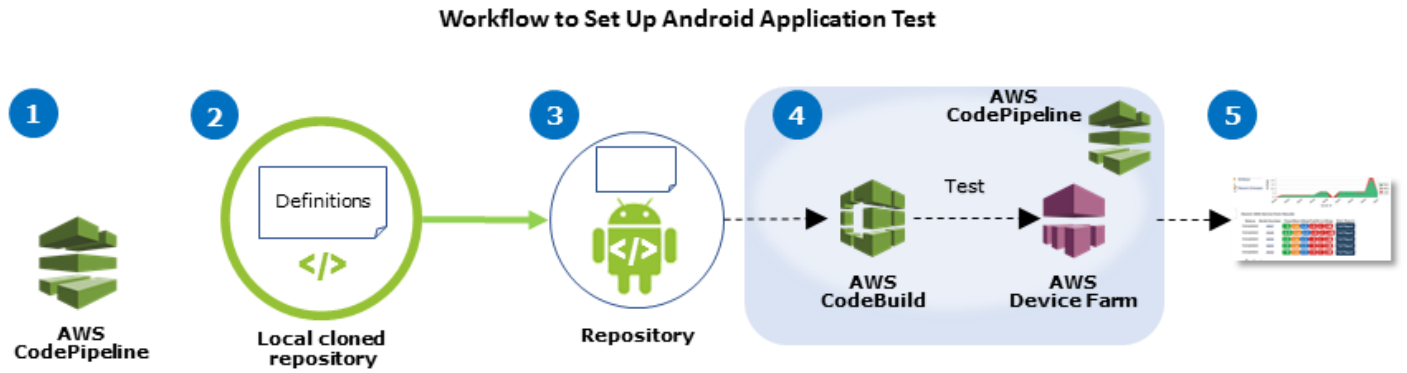
```
      "eventVersion": "1.03",
      "userIdentity": {
        "type": "Root",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "arn:aws:iam::123456789012:root",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
          "attributes":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

Mengintegrasikan AWS Device Farm dalam tahap CodePipeline pengujian

Anda dapat menggunakannya AWS CodePipelineuntuk menggabungkan pengujian aplikasi seluler yang dikonfigurasi di Device Farm ke dalam pipeline rilis otomatis yang dikelola AWS. Anda dapat mengonfigurasi pipeline untuk menjalankan pengujian sesuai permintaan, sesuai jadwal, atau sebagai bagian dari alur integrasi berkelanjutan.

Diagram berikut menunjukkan alur integrasi berkelanjutan di mana aplikasi Android dibangun dan diuji setiap kali push dilakukan ke repositorinya. Untuk membuat konfigurasi pipeline ini, lihat Tutorial: Membangun dan Menguji Aplikasi Android Saat Didorong ke GitHub.



1. Konfigurasikan	2. Tambahkan definisi	3. Dorong	4. Bangun dan uji	5. Laporkan
Konfigurasikan sumber daya pipa	Tambahkan definisi build dan test ke paket Anda	Dorong paket ke repositori Anda	Pembuatan aplikasi dan pengujian artefak keluaran build dimulai secara otomatis	Lihat hasil tes

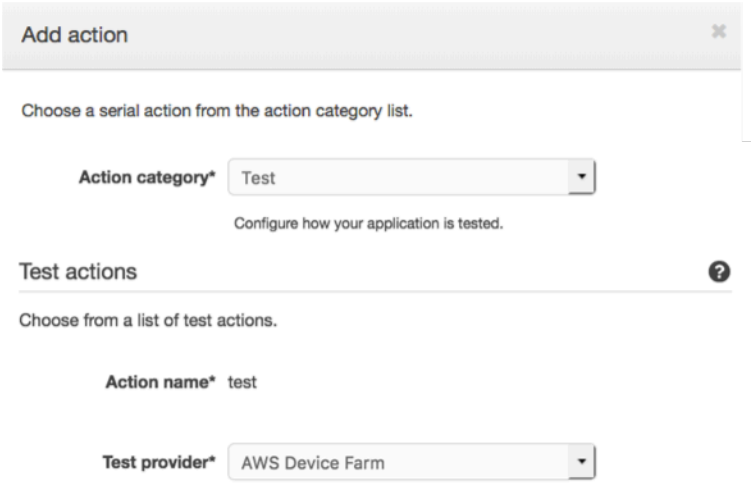
Untuk mempelajari cara mengonfigurasi pipeline yang terus-menerus menguji aplikasi yang dikompilasi (seperti `.apk` file iOS `.ipa` atau Android) sebagai sumbernya, lihat Tutorial: Menguji Aplikasi iOS Setiap Kali Anda Mengunggah File.ipa ke Bucket Amazon S3.

Konfigurasikan CodePipeline untuk menggunakan pengujian Device Farm

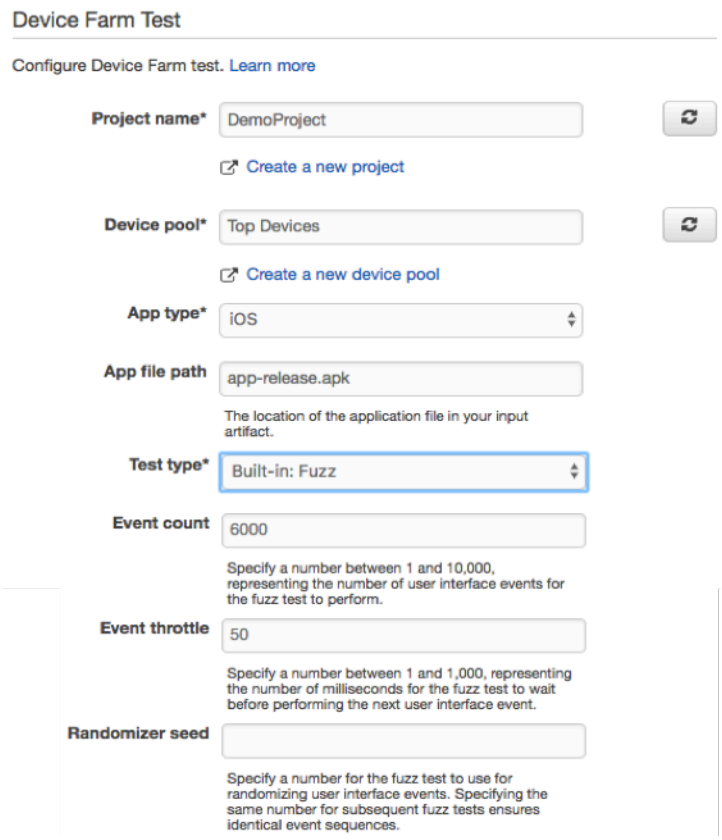
Dalam langkah-langkah ini, kami berasumsi bahwa Anda telah mengonfigurasi proyek Device Farm dan membuat pipeline. Pipeline harus dikonfigurasi dengan tahap pengujian yang menerima artefak input yang berisi definisi pengujian dan file paket aplikasi yang dikompilasi. Artefak input tahap pengujian dapat berupa artefak keluaran dari sumber atau tahap build yang dikonfigurasi dalam pipeline Anda.

Untuk mengonfigurasi uji coba Device Farm sebagai tindakan CodePipeline pengujian

1. Masuk ke Konsol Manajemen AWS dan buka CodePipeline konsol di https://console.aws.amazon.com/codepipeline/.
2. Pilih pipeline untuk rilis aplikasi Anda.
3. Pada panel tahap uji, pilih ikon pensil, lalu pilih Tindakan.
4. Pada panel Add action, untuk kategori Action, pilih Test.
5. Di Nama tindakan, masukkan nama.
6. Di penyedia Uji, pilih AWS Device Farm.

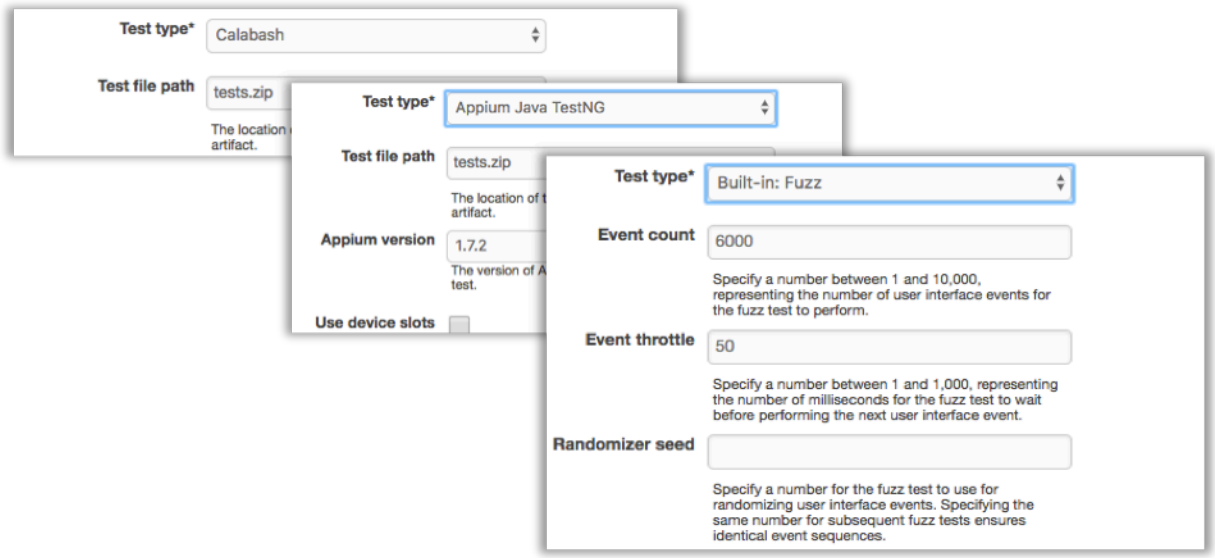


7. Dalam nama Proyek, pilih proyek Device Farm yang ada atau pilih Buat proyek baru.
8. Di kumpulan Perangkat, pilih kumpulan perangkat yang ada atau pilih Buat kumpulan perangkat baru. Jika Anda membuat kumpulan perangkat, Anda harus memilih satu set perangkat uji.
9. Di tipe Aplikasi, pilih platform untuk aplikasi Anda.

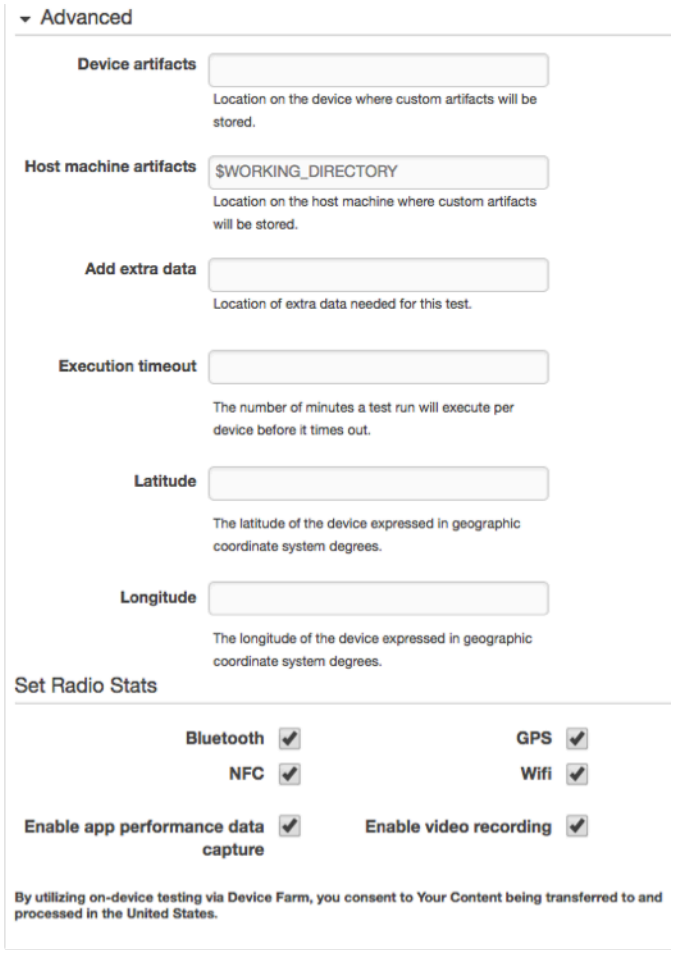


10. Di jalur file App, masukkan jalur paket aplikasi yang dikompilasi. Jalur relatif terhadap akar artefak input untuk pengujian Anda.
11. Pada tipe Test, lakukan salah satu hal berikut:

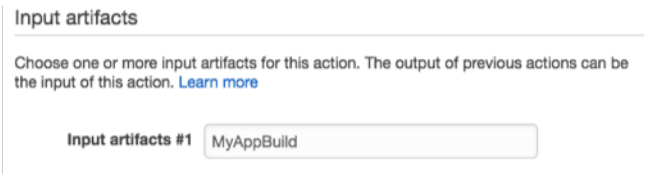
    - Jika Anda menggunakan salah satu pengujian Device Farm bawaan, pilih jenis pengujian yang dikonfigurasi dalam proyek Device Farm Anda.
    - Jika Anda tidak menggunakan salah satu pengujian bawaan Device Farm, di jalur file Uji, masukkan jalur file definisi pengujian. Jalur relatif terhadap akar artefak input untuk pengujian Anda.



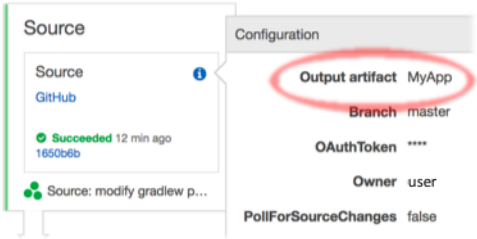
12. Di bidang yang tersisa, berikan konfigurasi yang sesuai untuk pengujian dan jenis aplikasi Anda.
13. (Opsional) Di Advanced, berikan konfigurasi terperinci untuk uji coba Anda.



14. Dalam artefak Input, pilih artefak input yang cocok dengan artefak keluaran dari tahap yang datang sebelum tahap pengujian dalam pipa.



Di CodePipeline konsol, Anda dapat menemukan nama artefak keluaran untuk setiap tahap dengan mengarahkan kursor ke ikon informasi di diagram pipa. Jika pipeline menguji aplikasi langsung dari tahap Sumber, pilih MyApp. Jika pipeline Anda menyertakan tahap Build, pilih MyAppBuild.



15. Di bagian bawah panel, pilih Tambah Tindakan.
16. Di CodePipeline panel, pilih Simpan perubahan pipeline, lalu pilih Simpan perubahan.
17. Untuk mengirimkan perubahan dan memulai pembuatan pipeline, pilih Rilis perubahan, lalu pilih Rilis.

AWS CLI referensi untuk AWS Device Farm

Untuk menggunakan AWS Command Line Interface (AWS CLI) untuk menjalankan perintah Device Farm, lihat AWS CLI Referensi untuk AWS Device Farm.

Untuk informasi umum tentang AWS CLI, lihat Panduan AWS Command Line Interface Pengguna dan Referensi AWS CLI Perintah.

PowerShell Referensi Windows untuk AWS Device Farm

Untuk menggunakan Windows PowerShell untuk menjalankan perintah Device Farm, lihat Referensi Cmdlet Device Farm di Referensi AWS Tools for Windows PowerShell Cmdlet. Untuk informasi selengkapnya, lihat Menyiapkan AWS Tools untuk Windows PowerShell di Panduan AWS Tools for PowerShell Pengguna.

Mengotomatisasi AWS Device Farm

Akses terprogram ke Device Farm adalah cara ampuh untuk mengotomatiskan tugas-tugas umum yang perlu Anda selesaikan, seperti menjadwalkan proses atau mengunduh artefak untuk dijalankan, suite, atau pengujian. AWS SDK dan AWS CLI menyediakan sarana untuk melakukannya.

AWS SDK menyediakan akses ke setiap AWS layanan, termasuk Device Farm, Amazon S3, dan banyak lagi. Untuk informasi selengkapnya, silakan lihat

- AWS alat-alat dan SDKs
- Referensi API AWS Device Farm

Contoh: Menggunakan AWS CLI atau SDK untuk mengunggah aplikasi atau menguji ke Device Farm

Contoh berikut menunjukkan cara membuat unggahan di Device Farm menggunakan AWS CLI atau menggunakan AWS SDK dalam berbagai bahasa. Unggahan adalah blok bangunan inti untuk penjadwalan pengujian yang dijalankan di Device Farm, dan termasuk yang berikut ini:

- Aplikasi Anda
- Tes Anda
- File spesifikasi pengujian Anda

Unggahan dibuat menggunakan `CreateUpload`API. API ini mengembalikan URL presigned S3 yang dapat Anda dorong upload menggunakan permintaan HTTP PUT. URL kedaluwarsa setelah 24 jam.

AWS CLI

> Catatan: contoh ini menggunakan alat baris perintah *`curl`* untuk mendorong aplikasi ke Device Farm.
>
> Pertama, buat proyek jika Anda belum melakukannya.
>
> ```
> $ aws devicefarm create-project --name MyProjectName
> ```
>
> Ini akan menampilkan output seperti berikut:

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

Kemudian, lakukan hal berikut untuk membuat unggahan Anda dan mendorongnya ke Device Farm. Dalam contoh ini, kita akan membuat unggahan aplikasi Android menggunakan file APK lokal. Untuk informasi jenis unggahan selengkapnya, termasuk detail tentang jenis unggahan aplikasi iOS, lihat dokumentasi API kami untuk membuat file `Upload`.

```
$ export APP_PATH="/local/path/to/my_sample_app.apk"
$ export APP_TYPE="ANDROID_APP"
```

Pertama, kami membuat unggahan di Device Farm:

```
$ aws devicefarm create-upload \
  --project-arn "arn:aws:devicefarm:us-west-2:123456789101:project:5e01a8c7-
c861-4c0a-b1d5-12345EXAMPLE" \
  --name "$(basename "$APP_PATH")" \
  --type "$APP_TYPE"
```

Ini akan menampilkan output seperti berikut:

```
{
    "upload": {
        "arn": "arn:aws:devicefarm:us-
west-2:385076942068:upload:490a6350-0ba3-43e5-83f5-d2896b069a34/a120e848-c57b-4e8d-
a720-d750a0c4d936",
        "name": "my_sample_app.apk",
        "created": 1760747318.266,
        "type": "ANDROID_APP",
        "status": "INITIALIZED",
        "url": "https://prod-us-west-2-uploads.s3.dualstack.us-west-2.amazonaws.com/
arn%3Aaws%3Adevicefarm%3Aus-west-2...",
        "category": "PRIVATE"
    }
}
```

Kemudian, lakukan panggilan PUT menggunakan curl untuk mendorong aplikasi ke bucket S3 Device Farm:

```
$ curl -T "$APP_PATH" "https://prod-us-west-2-uploads.s3.dualstack.us-
west-2.amazonaws.com/arn%3Aaws%3Adevicefarm%3Aus-west-2..."
```

Terakhir, tunggu aplikasi dalam status “berhasil”:

```
$ aws devicefarm get-upload --arn "arn:aws:devicefarm:us-
west-2:385076942068:upload:490a6350-0ba3-43e5-83f5-d2896b069a34/a120e848-c57b-4e8d-
a720-d750a0c4d936"
```

Ini akan menampilkan output seperti berikut:

```
{
    "upload": {
        "arn": "arn:aws:devicefarm:us-
west-2:385076942068:upload:490a6350-0ba3-43e5-83f5-d2896b069a34/a120e848-c57b-4e8d-
a720-d750a0c4d936",
        "name": "my_sample_app.apk",
        "created": 1760747318.266,
        "type": "ANDROID_APP",
        "status": "SUCCEEDED",
        "url": "https://prod-us-west-2-uploads.s3.dualstack.us-west-2.amazonaws.com/
arn%3Aaws%3Adevicefarm%3Aus-west-2...",
        "metadata": "{\"activity_name\":
\"com.amazonaws.devicefarm.android.referenceapp.Activities.MainActivity\",
\"package_name\":\"com.amazonaws.devicefarm.android.referenceapp\",...}",
        "category": "PRIVATE"
    }
}
```

Python

Catatan: contoh ini menggunakan *`requests`* paket pihak ketiga untuk mendorong aplikasi ke Device Farm, serta AWS SDK untuk Python*`boto3`*.

Pertama, buat proyek jika Anda belum melakukannya.

```
import boto3

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

```
resp = client.create_project(name="MyProjectName")

print(resp)
# Response will be something like:
# {
#     "project": {
#         "name": "MyProjectName",
#         "arn": "arn:aws:devicefarm:us-west-2:123456789101:project:5e01a8c7-
c861-4c0a-b1d5-12345EXAMPLE",
#         "created": 1535675814.414
#     }
# }
```

Kemudian, lakukan hal berikut untuk membuat unggahan Anda dan mendorongnya ke Device Farm. Dalam contoh ini, kita akan membuat unggahan aplikasi Android menggunakan file APK lokal. Untuk informasi jenis unggahan selengkapnya, termasuk detail tentang jenis unggahan aplikasi iOS, lihat dokumentasi API kami untuk membuat file `Upload`.

```
import os
import time
import datetime
import requests
from pathlib import Path
import boto3


def upload_device_farm_file():
    project_arn = "arn:aws:devicefarm:us-west-2:123456789101:project:5e01a8c7-
c861-4c0a-b1d5-12345EXAMPLE"
    app_path = Path("/local/path/to/my_sample_app.apk")
    file_type = "ANDROID_APP"

    if not app_path.is_file():
        raise RuntimeError(f"{app_path} is not a valid app file path")

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

    # 1) Create the upload in Device Farm
    create = client.create_upload(
        projectArn=project_arn,
        name=app_path.name,
        type=file_type,
        contentType="application/octet-stream",
```

```
    )
    upload = create["upload"]
    upload_arn = upload["arn"]
    upload_url = upload["url"]
    # This will show output such as the following:
    # { "upload": { "arn": "...", "name": "my_sample_app.apk", "type":
 "ANDROID_APP", "status": "INITIALIZED", "url": "https://..." } }

    # 2) Do an HTTP PUT command to push the file to the pre-signed S3 URL
    with app_path.open("rb") as fh:
        print(f"Uploading {app_path.name} to Device Farm...")
        put_resp = requests.put(upload_url, data=fh, headers={"Content-Type":
 "application/octet-stream"})
        put_resp.raise_for_status()

    # 3) Wait for the app to be in "SUCCEEDED" status (or fail/timeout)
    timeout_seconds = 30
    start = time.time()
    while True:
        get_resp = client.get_upload(arn=upload_arn)
        status = get_resp["upload"]["status"]
        msg = get_resp["upload"].get("message") or
 get_resp["upload"].get("metadata") or ""
        elapsed = datetime.timedelta(seconds=int(time.time() - start))
        print(f"[{elapsed}] status={status}{' - ' + msg if msg else ''}")

        if status == "SUCCEEDED":
            print(f"Upload complete: {upload_arn}")
            return upload_arn
        if status == "FAILED":
            raise RuntimeError(f"Device Farm failed to process upload: {msg}")

        if (time.time() - start) > timeout_seconds:
            raise RuntimeError(f"Timed out after {timeout_seconds}s waiting for
 upload to process (last status={status}).")

        time.sleep(1)

upload_device_farm_file()
```

Java

Catatan: contoh ini menggunakan AWS SDK for Java v2 *`HttpClient`* dan untuk mendorong aplikasi ke Device Farm, dan kompatibel dengan JDK versi 11 dan yang lebih tinggi.

Pertama, buat proyek jika Anda belum melakukannya.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.devicefarm.DeviceFarmClient;
import software.amazon.awssdk.services.devicefarm.model.CreateProjectRequest;
import software.amazon.awssdk.services.devicefarm.model.CreateProjectResponse;

try (DeviceFarmClient client = DeviceFarmClient.builder()
        .region(Region.US_WEST_2)
        .build()) {
    CreateProjectResponse resp = client.createProject(
        CreateProjectRequest.builder().name("MyProjectName").build());
    System.out.println(resp.project());
    // Response will be something like:
    // Project{name=MyProjectName, arn=arn:aws:devicefarm:us-
west-2:123456789101:project:5e01a8c7-..., created=...}
}
```

Kemudian, lakukan hal berikut untuk membuat unggahan Anda dan mendorongnya ke Device Farm. Dalam contoh ini, kita akan membuat unggahan aplikasi Android menggunakan file APK lokal. Untuk informasi jenis unggahan selengkapnya, termasuk detail tentang jenis unggahan aplikasi iOS, lihat dokumentasi API kami untuk membuat file `Upload`.

```
import java.io.IOException;
import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.time.Duration;
import java.time.Instant;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.devicefarm.DeviceFarmClient;
import software.amazon.awssdk.services.devicefarm.model.CreateUploadRequest;
import software.amazon.awssdk.services.devicefarm.model.CreateUploadResponse;
import software.amazon.awssdk.services.devicefarm.model.GetUploadRequest;
import software.amazon.awssdk.services.devicefarm.model.GetUploadResponse;
import software.amazon.awssdk.services.devicefarm.model.Upload;
import software.amazon.awssdk.services.devicefarm.model.UploadType;
```

```
public class DeviceFarmUploader {

    public static String upload(String projectArn, Path appPath) throws Exception {
        if (projectArn == null || projectArn.isEmpty()) {
            throw new IllegalArgumentException("Missing projectArn");
        }
        if (!Files.isRegularFile(appPath)) {
            throw new IllegalArgumentException("Invalid app path: " + appPath);
        }

        String fileName = appPath.getFileName().toString().trim();
        UploadType type = UploadType.ANDROID_APP;

        // Build a reusable HttpClient
        HttpClient http = HttpClient.newBuilder()
                .version(HttpClient.Version.HTTP_1_1)
                .connectTimeout(Duration.ofSeconds(10))
                .build();

        try (DeviceFarmClient client = DeviceFarmClient.builder()
                .region(Region.US_WEST_2)
                .build()) {

            // 1) Create the upload in Device Farm
            CreateUploadResponse create =
 client.createUpload(CreateUploadRequest.builder()
                    .projectArn(projectArn)
                    .name(fileName)
                    .type(type)
                    .contentType("application/octet-stream")
                    .build());

            Upload upload = create.upload();
            String uploadArn = upload.arn();
            String url = upload.url();
            // This will show output such as the following:
            // { "upload": { "arn": "...", "name": "my_sample_app.apk", "type":
 "ANDROID_APP", "status": "INITIALIZED", "url": "https://..." } }

            // 2) PUT file to pre-signed URL using HttpClient
            HttpRequest put = HttpRequest.newBuilder(URI.create(url))
                    .timeout(Duration.ofMinutes(15))
                    .header("Content-Type", "application/octet-stream")
```

```
                    .PUT(HttpRequest.BodyPublishers.ofFile(appPath))
                    .build();

            HttpResponse<Void> resp = http.send(put,
 HttpResponse.BodyHandlers.discarding());
            int code = resp.statusCode();
            if (code / 100 != 2) {
                throw new IOException("Failed PUT to S3 pre-signed URL, HTTP " +
 code);
            }

            // 3) Wait for the app to be in "SUCCEEDED" status (or fail/timeout)
            Instant deadline = Instant.now().plusSeconds(30); // 30-second timeout
            while (true) {
                GetUploadResponse got = client.getUpload(GetUploadRequest.builder()
                        .arn(uploadArn)
                        .build());

                String status = got.upload().statusAsString();
                String msg = got.upload().metadata();
                System.out.println("status=" + status + (msg != null ? " - " + msg :
 ""));

                if ("SUCCEEDED".equals(status)) return uploadArn;
                if ("FAILED".equals(status)) throw new RuntimeException("Upload
 failed: " + msg);
                if (Instant.now().isAfter(deadline)) {
                    throw new RuntimeException("Timeout waiting for processing, last
 status=" + status);
                }
                Thread.sleep(2000);
            }
        }
    }

    public static void main(String[] args) throws Exception {
        String projectArn = "arn:aws:devicefarm:us-
west-2:123456789101:project:5e01a8c7-c861-4c0a-b1d5-12345EXAMPLE";
        Path appPath = Paths.get("/local/path/to/my_sample_app.apk");
        String result = upload(projectArn, appPath);
        System.out.println("Upload ARN: " + result);
    }
}
```

JavaScript

Catatan: contoh ini menggunakan AWS SDK for JavaScript (v3) dan Node 18+ *fetch* untuk mendorong aplikasi ke Device Farm.

Pertama, buat proyek jika Anda belum melakukannya.

```
import { DeviceFarmClient, CreateProjectCommand } from "@aws-sdk/client-device-
farm";

const df = new DeviceFarmClient({ region: "us-west-2" });
const resp = await df.send(new CreateProjectCommand({ name: "MyProjectName" }));
console.log(resp);
// Response will be something like:
// { project: { name: 'MyProjectName', arn: 'arn:aws:devicefarm:us-
west-2:123456789101:project:5e01a8c7-...', created: 1535675814.414 } }
```

Kemudian, lakukan hal berikut untuk membuat unggahan Anda dan mendorongnya ke Device Farm. Dalam contoh ini, kita akan membuat unggahan aplikasi Android menggunakan file APK lokal. Untuk informasi jenis unggahan selengkapnya, termasuk detail tentang jenis unggahan aplikasi iOS, lihat dokumentasi API kami untuk membuat file `Upload`.

```
import { DeviceFarmClient, CreateUploadCommand, GetUploadCommand } from "@aws-sdk/
client-device-farm";
import { createReadStream } from "fs";
import { basename } from "path";

const projectArn = "arn:aws:devicefarm:us-west-2:123456789101:project:5e01a8c7-
c861-4c0a-b1d5-12345EXAMPLE";
const appPath = "/local/path/to/my_sample_app.apk";
const name = basename(appPath).trim();
const type = "ANDROID_APP";

const client = new DeviceFarmClient({ region: "us-west-2" });

// 1) Create the upload in Device Farm
const create = await client.send(new CreateUploadCommand({
  projectArn,
  name,
  type,
  contentType: "application/octet-stream",
}));
```

```
const uploadArn = create.upload.arn;
const url = create.upload.url;
// This will show output such as the following:
// { upload: { arn: '...', name: 'my_sample_app.apk', type: 'ANDROID_APP', status:
 'INITIALIZED', url: 'https://...' } }

// 2) PUT to pre-signed URL
const putResp = await fetch(url, {
  method: "PUT",
  headers: { "Content-Type": "application/octet-stream" },
  body: createReadStream(appPath),
});
if (!putResp.ok) {
  throw new Error(`Failed PUT to pre-signed URL: ${putResp.status} ${await
 putResp.text().catch(()=>"")}`);
}

// 3) Wait for the app to be in "SUCCEEDED" status (or fail/timeout)
const deadline = Date.now() + (30 * 1000); // 30-second timeout
while (true) {
  const response = await client.send(new GetUploadCommand({ arn: uploadArn }));
  const { status, message, metadata } = response.upload;
  console.log(`status=${status}${message ? " - " + message : metadata ? " - " +
 metadata : ""}`);
  if (status === "SUCCEEDED") {
    console.log("Upload complete:", uploadArn);
    break;
  }
  if (status === "FAILED") {
    throw new Error(`Upload failed: ${message || metadata || "unknown"}`);
  }
  if (Date.now() > deadline) throw new Error(`Timeout waiting for processing (last
 status=${status})`);
  await new Promise(r => setTimeout(r, 2000));
}
```

C#

Catatan: contoh ini menggunakan AWS SDK for .NET *`HttpClient`* dan untuk mendorong aplikasi ke Device Farm.

Pertama, buat proyek jika Anda belum melakukannya.

```
using System;
```

```
using Amazon;
using Amazon.DeviceFarm;
using Amazon.DeviceFarm.Model;

using var client = new AmazonDeviceFarmClient(RegionEndpoint.USWest2);
var resp = await client.CreateProjectAsync(new CreateProjectRequest { Name =
 "MyProjectName" });
Console.WriteLine(resp.Project);
// Response will be something like:
// { Name = MyProjectName, Arn = arn:aws:devicefarm:us-
west-2:123456789101:project:5e01a8c7-..., Created = ... }
```

Kemudian, lakukan hal berikut untuk membuat unggahan Anda dan mendorongnya ke Device Farm. Dalam contoh ini, kita akan membuat unggahan aplikasi Android menggunakan file APK lokal. Untuk informasi jenis unggahan selengkapnya, termasuk detail tentang jenis unggahan aplikasi iOS, lihat dokumentasi API kami untuk membuat file `Upload`.

```
using System;
using System.IO;
using System.Net.Http;
using System.Threading.Tasks;
using System.Net.Http.Headers;
using Amazon;
using Amazon.DeviceFarm;
using Amazon.DeviceFarm.Model;

class DeviceFarmUploader
{
    public static async Task<string> UploadAsync(string projectArn, string appPath)
    {
        if (string.IsNullOrWhiteSpace(projectArn)) throw new
 ArgumentException("Missing projectArn");
        if (!File.Exists(appPath)) throw new ArgumentException($"Invalid app path:
 {appPath}");
        var type = UploadType.ANDROID_APP;

        using var client = new AmazonDeviceFarmClient(RegionEndpoint.USWest2);
        // 1) Create the upload in Device Farm
        var create = await client.CreateUploadAsync(new CreateUploadRequest
        {
            ProjectArn = projectArn,
            Name = Path.GetFileName(appPath),
            Type = type,
```

```
            ContentType = "application/octet-stream"
        });

        var uploadArn = create.Upload.Arn;
        var url = create.Upload.Url;
        // This will show output such as the following:
        // { Upload: { Arn = ..., Name = my_sample_app.apk, Type = ANDROID_APP,
Status = INITIALIZED, Url = https://... } }

        // 2) PUT file to pre-signed URL
        using (var http = new HttpClient())
        using (var fs = File.OpenRead(appPath))
        using (var content = new StreamContent(fs))
        {
            content.Headers.Add("Content-Type", "application/octet-stream");
            var resp = await http.PutAsync(url, content);
            if (!resp.IsSuccessStatusCode)
                throw new Exception($"Failed PUT to pre-signed URL:
{(int)resp.StatusCode} {await resp.Content.ReadAsStringAsync()}");
        }

        // 3) Wait for the app to be in "SUCCEEDED" status (or fail/timeout)
        var deadline = DateTime.UtcNow.AddSeconds(30); // 30-second timeout
        while (true)
        {
            var got = await client.GetUploadAsync(new GetUploadRequest { Arn =
uploadArn });
            var status = got.Upload.Status.Value;
            var msg = got.Upload.Message ?? got.Upload.Metadata;
            Console.WriteLine($"status={status}{(string.IsNullOrEmpty(msg) ? "" : "
- " + msg)}");

            if (status == UploadStatus.SUCCEEDED.Value) return uploadArn;
            if (status == UploadStatus.FAILED.Value) throw new Exception($"Upload
failed: {msg}");
            if (DateTime.UtcNow > deadline) throw new TimeoutException($"Timeout
waiting for processing (last status={status})");
            await Task.Delay(2000);
        }
    }

    static async Task Main()
    {
```

```
        var projectArn = "arn:aws:devicefarm:us-
west-2:123456789101:project:5e01a8c7-c861-4c0a-b1d5-12345EXAMPLE";
        var appPath = "/local/path/to/my_sample_app.apk";
        var result = await UploadAsync(projectArn!, appPath!);
        Console.WriteLine("Upload ARN: " + result);
    }
}
```

Ruby

Catatan: contoh ini menggunakan AWS SDK for *`Net::HTTP`* Ruby dan mendorong aplikasi ke Device Farm.

Pertama, buat proyek jika Anda belum melakukannya.

```
require "aws-sdk-devicefarm"

client = Aws::DeviceFarm::Client.new(region: "us-west-2")
resp = client.create_project(name: "MyProjectName")
puts resp.project.inspect
# Response will be something like:
# #<struct Aws::DeviceFarm::Types::Project name="MyProjectName",
 arn="arn:aws:devicefarm:us-west-2:123456789101:project:5e01a8c7-...",
 created=1535675814.414>
```

Kemudian, lakukan hal berikut untuk membuat unggahan Anda dan mendorongnya ke Device Farm. Dalam contoh ini, kita akan membuat unggahan aplikasi Android menggunakan file APK lokal. Untuk informasi jenis unggahan selengkapnya, termasuk detail tentang jenis unggahan aplikasi iOS, lihat dokumentasi API kami untuk membuat file `Upload`.

```
require "aws-sdk-devicefarm"
require "net/http"
require "uri"

project_arn = "arn:aws:devicefarm:us-west-2:123456789101:project:5e01a8c7-c861-4c0a-
b1d5-12345EXAMPLE"
app_path    = "/local/path/to/my_sample_app.apk"
raise "Invalid APP_PATH: #{app_path}" unless File.file?(app_path)
type = "ANDROID_APP"

client = Aws::DeviceFarm::Client.new(region: "us-west-2")

# 1) Create the upload in Device Farm
```

```
create = client.create_upload(
  project_arn: project_arn,
  name: File.basename(app_path),
  type: type,
  content_type: "application/octet-stream"
)

upload_arn = create.upload.arn
url = create.upload.url
# This will show output such as the following:
# #<Upload arn="...", name="my_sample_app.apk", type="ANDROID_APP",
 status="INITIALIZED", url="https://...">

# 2) PUT the file to the pre-signed URL
uri = URI.parse(url)
Net::HTTP.start(uri.host, uri.port, use_ssl: (uri.scheme == "https")) do |http|
  req = Net::HTTP::Put.new(uri)
  req["Content-Type"] = "application/octet-stream"
  req.body_stream = File.open(app_path, "rb")
  req.content_length = File.size(app_path)
  resp = http.request(req)
  raise "Failed PUT: #{resp.code} #{resp.body}" unless resp.code.to_i / 100 == 2
end

# 3) Wait for the app to be in "SUCCEEDED" status (or fail/timeout)
deadline = Time.now + 30 # 30-second timeout
loop do
  got = client.get_upload(arn: upload_arn)
  status = got.upload.status
  msg = got.upload.message || got.upload.metadata
  puts "status=#{status}#{msg ? " - #{msg}" : ""}"

  case status
  when "SUCCEEDED" then puts "Upload complete: #{upload_arn}"; break
  when "FAILED"    then raise "Upload failed: #{msg}"
  end
  raise "Timeout waiting for processing (last status=#{status})" if Time.now >
 deadline
  sleep 2
end
```

Contoh: Menggunakan AWS SDK untuk memulai proses Device Farm dan mengumpulkan artefak

Contoh berikut memberikan beginning-to-end demonstrasi bagaimana Anda dapat menggunakan AWS SDK untuk bekerja dengan Device Farm. Contoh ini melakukan hal berikut:

- Mengunggah paket pengujian dan aplikasi ke Device Farm
- Memulai uji coba dan menunggu penyelesaiannya (atau kegagalan)
- Mengunduh semua artefak yang diproduksi oleh suite uji

Contoh ini tergantung pada `requests` paket pihak ketiga untuk berinteraksi dengan HTTP.

```
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")
```

Memecahkan masalah kesalahan Device Farm

Di bagian ini, Anda akan menemukan pesan kesalahan dan prosedur untuk membantu Anda memperbaiki masalah umum dengan Device Farm.

> **Note**
>
> Untuk memecahkan masalah pengujian Appium yang gagal secara tak terduga di Device Farm, silakan lihat panduan kami untuk pengujian Appium sisi klien

Topik

Memecahkan masalah pengujian aplikasi Android di AWS Device Farm

Topik berikut mencantumkan pesan galat yang terjadi selama pengunggahan pengujian aplikasi Android dan merekomendasikan solusi untuk mengatasi setiap kesalahan.

> **Note**
>
> Petunjuk di bawah ini didasarkan pada Linux x86_64 dan Mac.

ANDROID_APP_UNZIP_FAILED

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat membuka aplikasi Anda. Harap verifikasi bahwa file tersebut valid dan coba lagi.

Pastikan Anda dapat unzip paket aplikasi tanpa kesalahan. Dalam contoh berikut, nama paket adalah app-debug.apk.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip app-debug.apk
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   Paket aplikasi Android yang valid harus menghasilkan output seperti berikut:

   ```
   .
   |-- AndroidManifest.xml
   |-- classes.dex
   |-- resources.arsc
   |-- assets (directory)
   |-- res (directory)
   `-- META-INF (directory)
   ```

   Untuk informasi selengkapnya, lihat Pengujian Android di AWS Device Farm.

ANDROID_APP_AAPT_DEBUG_BADGING_FAILED

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat mengekstrak informasi tentang aplikasi Anda. Harap verifikasi bahwa aplikasi tersebut valid dengan menjalankan perintah`aapt debug badging <path to your test package>`, dan coba lagi setelah perintah tidak mencetak kesalahan apa pun.

Selama proses validasi unggahan, AWS Device Farm mem-parsing informasi dari output perintah`aapt debug badging <path to your package>`.

Pastikan Anda dapat menjalankan perintah ini di aplikasi Android Anda dengan sukses. Dalam contoh berikut, nama paket adalah app-debug.apk.

- Salin paket aplikasi Anda ke direktori kerja Anda, lalu jalankan perintah:

  ```
  $ aapt debug badging app-debug.apk
  ```

  Paket aplikasi Android yang valid harus menghasilkan output seperti berikut:

  ```
  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'
  ```

  Untuk informasi selengkapnya, lihat Pengujian Android di AWS Device Farm.

ANDROID_APP_PACKAGE_NAME_VALUE_MISSING

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat menemukan nilai nama paket dalam aplikasi Anda. Harap verifikasi bahwa aplikasi tersebut valid dengan menjalankan perintah`aapt debug badging <path to your test package>`, dan coba lagi setelah menemukan nilai nama paket di belakang kata kunci “package: name.”

Selama proses validasi upload, AWS Device Farm mem-parsing nilai nama paket dari output perintah`aapt debug badging <path to your package>`.

Pastikan Anda dapat menjalankan perintah ini di aplikasi Android Anda dan menemukan nilai nama paket dengan sukses. Dalam contoh berikut, nama paket adalah app-debug.apk.

- Salin paket aplikasi Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

  ```
  $ aapt debug badging app-debug.apk | grep "package: name="
  ```

  Paket aplikasi Android yang valid harus menghasilkan output seperti berikut:

  ```
  package: name='com.amazon.aws.adf.android.referenceapp' versionCode='1'
   versionName='1.0' platformBuildVersionName='5.1.1-1819727'
  ```

  Untuk informasi selengkapnya, lihat Pengujian Android di AWS Device Farm.

ANDROID_APP_SDK_VERSION_VALUE_MISSING

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat menemukan nilai versi SDK di aplikasi Anda. Harap verifikasi bahwa aplikasi valid dengan menjalankan perintah`aapt debug badging <path to your`

test package>, dan coba lagi setelah menemukan nilai versi SDK di belakang kata kuncisdkVersion.

Selama proses validasi upload, AWS Device Farm mem-parsing nilai versi SDK dari output perintah. `aapt debug badging` *<path to your package>*

Pastikan Anda dapat menjalankan perintah ini di aplikasi Android Anda dan menemukan nilai nama paket dengan sukses. Dalam contoh berikut, nama paket adalah app-debug.apk.

- Salin paket aplikasi Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

  ```
  $ aapt debug badging app-debug.apk | grep "sdkVersion"
  ```

  Paket aplikasi Android yang valid harus menghasilkan output seperti berikut:

  ```
  sdkVersion:'9'
  ```

  Untuk informasi selengkapnya, lihat Pengujian Android di AWS Device Farm.

ANDROID_APP_AAPT_DUMP_XMLTREE_FAILED

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat menemukan AndroidManifest .xml.xl yang valid dalam aplikasi Anda. Harap verifikasi bahwa paket pengujian valid dengan menjalankan perintah`aapt dump xmltree` *<path to your test package>* `AndroidManifest.xml`, dan coba lagi setelah perintah tidak mencetak kesalahan apa pun.

Selama proses validasi upload, AWS Device Farm mem-parsing informasi dari pohon parse XMLuntuk file XMLyang terdapat dalam paket menggunakan perintah. `aapt dump xmltree` *<path to your package>* `AndroidManifest.xml`

Pastikan Anda dapat menjalankan perintah ini di aplikasi Android Anda dengan sukses. Dalam contoh berikut, nama paket adalah app-debug.apk.

- Salin paket aplikasi Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

```
$ aapt dump xmltree app-debug.apk. AndroidManifest.xml
```

Paket aplikasi Android yang valid harus menghasilkan output seperti berikut:

```
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")
```

Untuk informasi selengkapnya, lihat Pengujian Android di AWS Device Farm.

ANDROID_APP_DEVICE_ADMIN_PERMISSIONS

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami menemukan bahwa aplikasi Anda memerlukan izin admin perangkat. Harap verifikasi bahwa izin tidak diperlukan dengan menjalankan perintah`aapt dump xmltree` *`<path to your test package>`* `AndroidManifest.xml`, dan coba lagi setelah memastikan bahwa output tidak mengandung kata kunci`android.permission.BIND_DEVICE_ADMIN`.

Selama proses validasi unggahan, AWS Device Farm mem-parsing informasi izin dari pohon parse xmluntuk file xmlyang terdapat dalam paket menggunakan perintah. `aapt dump xmltree <path to your package> AndroidManifest.xml`

Pastikan aplikasi Anda tidak memerlukan izin admin perangkat. Dalam contoh berikut, nama paket adalah app-debug.apk.

- Salin paket aplikasi Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

```
$ aapt dump xmltree app-debug.apk AndroidManifest.xml
```

Anda harus menemukan output seperti berikut:

```
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")
        ……
```

Jika aplikasi Android valid, output tidak boleh berisi yang berikut:`A: android:name(0x01010003)="android.permission.BIND_DEVICE_ADMIN" (Raw: "android.permission.BIND_DEVICE_ADMIN")`.

Untuk informasi selengkapnya, lihat Pengujian Android di AWS Device Farm.

Jendela tertentu di aplikasi Android saya menampilkan layar kosong atau hitam

Jika Anda menguji aplikasi Android dan melihat bahwa jendela tertentu dalam aplikasi muncul dengan layar hitam dalam perekaman video Device Farm dari pengujian Anda, aplikasi Anda mungkin menggunakan `FLAG_SECURE` fitur Android. Bendera ini (seperti yang dijelaskan dalam dokumentasi resmi Android) digunakan untuk mencegah jendela aplikasi tertentu direkam oleh alat perekam layar. Akibatnya, fitur perekaman layar Device Farm (untuk otomatisasi dan pengujian akses jarak jauh) dapat menampilkan layar hitam sebagai pengganti jendela aplikasi Anda jika jendela menggunakan bendera ini.

Bendera ini sering digunakan oleh pengembang untuk halaman dalam aplikasi mereka yang berisi informasi sensitif seperti halaman login. Jika Anda melihat layar hitam di tempat layar aplikasi Anda untuk halaman tertentu seperti halaman loginnya, bekerja dengan pengembang Anda untuk mendapatkan build aplikasi yang tidak menggunakan bendera ini untuk pengujian.

Selain itu, perhatikan bahwa Device Farm masih dapat berinteraksi dengan jendela aplikasi yang memiliki tanda ini. Jadi, jika halaman login aplikasi Anda muncul sebagai layar hitam, Anda mungkin masih dapat memasukkan kredensi Anda untuk masuk ke aplikasi (dan dengan demikian melihat halaman yang tidak diblokir oleh `FLAG_SECURE` bendera).

Memecahkan masalah pengujian Appium Java JUnit di AWS Device Farm

Topik berikut mencantumkan pesan kesalahan yang terjadi selama pengunggahan JUnit pengujian Appium Java dan merekomendasikan solusi untuk menyelesaikan setiap kesalahan.

> **Note**
>
> Petunjuk di bawah ini didasarkan pada Linux x86_64 dan Mac.

APPIUM_JAVA_JUNIT_TEST_PACKAGE_UNZIP_FAILED

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat membuka file ZIP pengujian Anda. Harap verifikasi bahwa file tersebut valid dan coba lagi.

Pastikan Anda dapat membuka zip paket pengujian tanpa kesalahan. Dalam contoh berikut, nama paket adalah zip-with-dependencies.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   JUnit Paket Appium Java yang valid harus menghasilkan output seperti berikut:

   ```
   .
   |— 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
   ```

   Untuk informasi selengkapnya, lihat Jalankan pengujian Appium secara otomatis di Device Farm.

APPIUM_JAVA_JUNIT_TEST_PACKAGE_DEPENDENCY_DIR_MISSING

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat menemukan direktori dependency-jars di dalam paket pengujian Anda. Harap unzip paket pengujian Anda, verifikasi bahwa direktori dependency-jars ada di dalam paket, dan coba lagi.

Dalam contoh berikut, nama paket adalah zip-with-dependencies.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   Jika JUnit paket Appium Java valid, Anda akan menemukan *`dependency-jars`* direktori di dalam direktori kerja:

   ```
   .
   |— 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
   ```

   Untuk informasi selengkapnya, lihat Jalankan pengujian Appium secara otomatis di Device Farm.

APPIUM_JAVA_JUNIT_TEST_PACKAGE_JAR_MISSING_IN_DEPENDENCY_DI

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat menemukan file JAR di pohon direktori dependency-jars. Harap unzip paket pengujian Anda dan kemudian buka direktori dependency-jars, verifikasi bahwa setidaknya satu file JAR ada di direktori, dan coba lagi.

Dalam contoh berikut, nama paket adalah zip-with-dependencies.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   Jika JUnit paket Appium Java valid, Anda akan menemukan setidaknya satu *jar* file di dalam direktori: *dependency-jars*

   ```
   .
   |— 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
   ```

   Untuk informasi selengkapnya, lihat Jalankan pengujian Appium secara otomatis di Device Farm.

APPIUM_JAVA_JUNIT_TEST_PACKAGE_TESTS_JAR_FILE_MISSING

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat menemukan file*-tests.jar dalam paket pengujian Anda. Harap unzip paket pengujian Anda, verifikasi bahwa setidaknya satu file*-tests.jar ada dalam paket, dan coba lagi.

Dalam contoh berikut, nama paket adalah zip-with-dependencies.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   Jika JUnit paket Appium Java valid, Anda akan menemukan setidaknya satu *jar* file seperti *acme-android-appium-1.0-SNAPSHOT-tests.jar* dalam contoh kami. Nama file mungkin berbeda, tetapi harus diakhiri dengan*-tests.jar*.

   ```
   .
   |— 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
   ```

   Untuk informasi selengkapnya, lihat Jalankan pengujian Appium secara otomatis di Device Farm.

APPIUM_JAVA_JUNIT_TEST_PACKAGE_CLASS_FILE_MISSING_IN_TESTS_JA

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat menemukan file kelas dalam file JAR tes. Harap unzip paket pengujian Anda dan kemudian unjar file JAR tes, verifikasi bahwa setidaknya satu file kelas ada di dalam file JAR, dan coba lagi.

Dalam contoh berikut, nama paket adalah zip-with-dependencies.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   Anda harus menemukan setidaknya satu file jar seperti *acme-android-appium-1.0-SNAPSHOT-tests.jar* dalam contoh kami. Nama file mungkin berbeda, tetapi harus diakhiri dengan*-tests.jar*.

   ```
   .
   |— 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. Setelah Anda berhasil mengekstrak file, Anda harus menemukan setidaknya satu kelas di pohon direktori kerja dengan menjalankan perintah:

```
$ tree .
```

Anda akan melihat output seperti ini:

```
.
|— 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
```

Untuk informasi selengkapnya, lihat Jalankan pengujian Appium secara otomatis di Device Farm.

APPIUM_JAVA_JUNIT_TEST_PACKAGE_JUNIT_VERSION_VALUE_UNKNOWN

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat menemukan nilai JUnit versi. Harap unzip paket pengujian Anda dan buka direktori dependency-jars, verifikasi bahwa file JUnit JAR ada di dalam direktori, dan coba lagi.

Dalam contoh berikut, nama paket adalah zip-with-dependencies.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

```
$ unzip zip-with-dependencies.zip
```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

```
tree .
```

Outputnya akan terlihat seperti ini:

```
.
|— 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
```

Jika JUnit paket Appium Java valid, Anda akan menemukan file JUnit dependensi yang mirip dengan file jar *junit-4.10.jar* dalam contoh kita. Nama harus terdiri dari kata kunci *junit* dan nomor versinya, yang dalam contoh ini adalah 4.10.

Untuk informasi selengkapnya, lihat Jalankan pengujian Appium secara otomatis di Device Farm.

APPIUM_JAVA_JUNIT_TEST_PACKAGE_INVALID_JUNIT_VERSION

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami menemukan JUnit versinya lebih rendah dari versi minimum 4.10 yang kami dukung. Silakan ubah JUnit versi dan coba lagi.

Dalam contoh berikut, nama paket adalah zip-with-dependencies.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

```
$ unzip zip-with-dependencies.zip
```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

```
$ tree .
```

Anda harus menemukan file JUnit ketergantungan seperti *junit-4.10.jar* dalam contoh kami dan nomor versinya, yang dalam contoh kami adalah 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**
>
> Pengujian Anda mungkin tidak dijalankan dengan benar jika JUnit versi yang ditentukan dalam paket pengujian Anda lebih rendah dari versi minimum 4.10 yang kami dukung.

Untuk informasi selengkapnya, lihat Jalankan pengujian Appium secara otomatis di Device Farm.

Memecahkan masalah pengujian JUnit aplikasi web Appium Java di AWS Device Farm

Topik berikut mencantumkan pesan kesalahan yang terjadi selama pengunggahan pengujian aplikasi Appium Java JUnit Web dan merekomendasikan solusi untuk menyelesaikan setiap kesalahan. Untuk informasi selengkapnya tentang penggunaan Appium dengan Device Farm, lihat. the section called “Tes Appium otomatis”

APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_UNZIP_FAILED

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat membuka file ZIP pengujian Anda. Harap verifikasi bahwa file tersebut valid dan coba lagi.

Pastikan Anda dapat membuka zip paket pengujian tanpa kesalahan. Dalam contoh berikut, nama paket adalah zip-with-dependencies.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   JUnit Paket Appium Java yang valid harus menghasilkan output seperti berikut:

   ```
   .
   |— 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

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat menemukan direktori dependency-jars di dalam paket pengujian Anda. Harap unzip paket pengujian Anda, verifikasi bahwa direktori dependency-jars ada di dalam paket, dan coba lagi.

Dalam contoh berikut, nama paket adalah zip-with-dependencies.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   Jika JUnit paket Appium Java valid, Anda akan menemukan *`dependency-jars`* direktori di dalam direktori kerja:

   ```
   .
   |— 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

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat menemukan file JAR di pohon direktori dependency-jars. Harap unzip paket pengujian Anda dan kemudian buka direktori dependency-jars, verifikasi bahwa setidaknya satu file JAR ada di direktori, dan coba lagi.

Dalam contoh berikut, nama paket adalah zip-with-dependencies.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   Jika JUnit paket Appium Java valid, Anda akan menemukan setidaknya satu *jar* file di dalam direktori: *dependency-jars*

   ```
   .
   |– 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

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat menemukan file*-tests.jar dalam paket pengujian Anda. Harap unzip paket pengujian Anda, verifikasi bahwa setidaknya satu file*-tests.jar ada dalam paket, dan coba lagi.

Dalam contoh berikut, nama paket adalah zip-with-dependencies.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   Jika JUnit paket Appium Java valid, Anda akan menemukan setidaknya satu *jar* file seperti *acme-android-appium-1.0-SNAPSHOT-tests.jar* dalam contoh kami. Nama file mungkin berbeda, tetapi harus diakhiri dengan*-tests.jar*.

   ```
   .
   |– 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

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat menemukan file kelas dalam file JAR tes. Harap unzip paket pengujian Anda dan kemudian unjar file JAR tes, verifikasi bahwa setidaknya satu file kelas ada di dalam file JAR, dan coba lagi.

Dalam contoh berikut, nama paket adalah zip-with-dependencies.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   Anda harus menemukan setidaknya satu file jar seperti *acme-android-appium-1.0-SNAPSHOT-tests.jar* dalam contoh kami. Nama file mungkin berbeda, tetapi harus diakhiri dengan*-tests.jar*.

   ```
   .
   |– 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. Setelah Anda berhasil mengekstrak file, Anda harus menemukan setidaknya satu kelas di pohon direktori kerja dengan menjalankan perintah:

```
$ tree .
```

Anda akan melihat output seperti ini:

```
.
|– 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

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat menemukan nilai JUnit versi. Harap unzip paket pengujian Anda dan buka direktori dependency-jars, verifikasi bahwa file JUnit JAR ada di dalam direktori, dan coba lagi.

Dalam contoh berikut, nama paket adalah zip-with-dependencies.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

```
$ unzip zip-with-dependencies.zip
```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

```
tree .
```

Outputnya akan terlihat seperti ini:

```
.
|— 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
```

Jika JUnit paket Appium Java valid, Anda akan menemukan file JUnit dependensi yang mirip dengan file jar *junit-4.10.jar* dalam contoh kita. Nama harus terdiri dari kata kunci *junit* dan nomor versinya, yang dalam contoh ini adalah 4.10.

APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_INVALID_JUNIT_VERSION

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami menemukan JUnit versinya lebih rendah dari versi minimum 4.10 yang kami dukung. Silakan ubah JUnit versi dan coba lagi.

Dalam contoh berikut, nama paket adalah zip-with-dependencies.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

```
$ unzip zip-with-dependencies.zip
```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

```
$ tree .
```

Anda harus menemukan file JUnit ketergantungan seperti *junit-4.10.jar* dalam contoh kami dan nomor versinya, yang dalam contoh kami adalah 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**
>
> Pengujian Anda mungkin tidak dijalankan dengan benar jika JUnit versi yang ditentukan dalam paket pengujian Anda lebih rendah dari versi minimum 4.10 yang kami dukung.

Lihat informasi yang lebih lengkap di Jalankan pengujian Appium secara otomatis di Device Farm.

Memecahkan masalah tes Appium Java TestNG di AWS Device Farm

Topik berikut mencantumkan pesan kesalahan yang terjadi selama pengunggahan tes Appium Java TestNG dan merekomendasikan solusi untuk menyelesaikan setiap kesalahan.

> **Note**
>
> Petunjuk di bawah ini didasarkan pada Linux x86_64 dan Mac.

APPIUM_JAVA_TESTNG_TEST_PACKAGE_UNZIP_FAILED

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat membuka file ZIP pengujian Anda. Harap verifikasi bahwa file tersebut valid dan coba lagi.

Pastikan Anda dapat membuka zip paket pengujian tanpa kesalahan. Dalam contoh berikut, nama paket adalah zip-with-dependencies.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   JUnit Paket Appium Java yang valid harus menghasilkan output seperti berikut:

   ```
   .
   |— 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
```

Untuk informasi selengkapnya, lihat Jalankan pengujian Appium secara otomatis di Device Farm.

APPIUM_JAVA_TESTNG_TEST_PACKAGE_DEPENDENCY_DIR_MISSING

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat menemukan `dependency-jars` direktori di dalam paket pengujian Anda. Harap unzip paket pengujian Anda, verifikasi bahwa `dependency-jars` direktori ada di dalam paket, dan coba lagi.

Dalam contoh berikut, nama paket adalah zip-with-dependencies.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   Jika JUnit paket Appium Java valid, Anda akan menemukan *`dependency-jars`* direktori di dalam direktori kerja.

   ```
   .
   |– 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
```

Untuk informasi selengkapnya, lihat Jalankan pengujian Appium secara otomatis di Device Farm.

APPIUM_JAVA_TESTNG_TEST_PACKAGE_JAR_MISSING_IN_DEPENDENCY_

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat menemukan file JAR di pohon direktori dependency-jars. Harap unzip paket pengujian Anda dan kemudian buka direktori dependency-jars, verifikasi bahwa setidaknya satu file JAR ada di direktori, dan coba lagi.

Dalam contoh berikut, nama paket adalah zip-with-dependencies.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   Jika JUnit paket Appium Java valid, Anda akan menemukan setidaknya satu *jar* file di dalam direktori. *dependency-jars*

   ```
   .
   |– 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
```

Untuk informasi selengkapnya, lihat Jalankan pengujian Appium secara otomatis di Device Farm.

APPIUM_JAVA_TESTNG_TEST_PACKAGE_TESTS_JAR_FILE_MISSING

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat menemukan file*-tests.jar dalam paket pengujian Anda. Harap unzip paket pengujian Anda, verifikasi bahwa setidaknya satu file*-tests.jar ada dalam paket, dan coba lagi.

Dalam contoh berikut, nama paket adalah zip-with-dependencies.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   Jika JUnit paket Appium Java valid, Anda akan menemukan setidaknya satu *jar* file seperti *acme-android-appium-1.0-SNAPSHOT-tests.jar* dalam contoh kami. Nama file mungkin berbeda, tetapi harus diakhiri dengan*-tests.jar*.

   ```
   .
   ```

```
|– 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
```

Untuk informasi selengkapnya, lihat Jalankan pengujian Appium secara otomatis di Device Farm.

APPIUM_JAVA_TESTNG_TEST_PACKAGE_CLASS_FILE_MISSING_IN_TESTS

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat menemukan file kelas dalam file JAR tes. Harap unzip paket pengujian Anda dan kemudian unjar file JAR tes, verifikasi bahwa setidaknya satu file kelas ada di dalam file JAR, dan coba lagi.

Dalam contoh berikut, nama paket adalah zip-with-dependencies.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   Anda harus menemukan setidaknya satu file jar seperti *acme-android-appium-1.0-SNAPSHOT-tests.jar* dalam contoh kami. Nama file mungkin berbeda, tetapi harus diakhiri dengan*-tests.jar*.

```
.
|– 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. Untuk mengekstrak file dari file jar, Anda dapat menjalankan perintah berikut:

```
$ jar xf acme-android-appium-1.0-SNAPSHOT-tests.jar
```

4. Setelah Anda berhasil mengekstrak file, jalankan perintah berikut:

```
$ tree .
```

Anda harus menemukan setidaknya satu kelas di pohon direktori kerja:

```
.
|– 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
```

Untuk informasi selengkapnya, lihat Jalankan pengujian Appium secara otomatis di Device Farm.

Pemecahan Masalah Aplikasi web Appium Java TestNG di AWS Device Farm

Topik berikut mencantumkan pesan kesalahan yang terjadi selama pengunggahan pengujian aplikasi Appium Java TestNG Web dan merekomendasikan solusi untuk menyelesaikan setiap kesalahan.

APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_UNZIP_FAILED

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat membuka file ZIP pengujian Anda. Harap verifikasi bahwa file tersebut valid dan coba lagi.

Pastikan Anda dapat membuka zip paket pengujian tanpa kesalahan. Dalam contoh berikut, nama paket adalah zip-with-dependencies.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   JUnit Paket Appium Java yang valid harus menghasilkan output seperti berikut:

   ```
   .
   |— 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
```

Untuk informasi selengkapnya, lihat Jalankan pengujian Appium secara otomatis di Device Farm.

APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_DEPENDENCY_DIR_MISSIN

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat menemukan direktori dependency-jars di dalam paket pengujian Anda. Harap unzip paket pengujian Anda, verifikasi bahwa direktori dependency-jars ada di dalam paket, dan coba lagi.

Dalam contoh berikut, nama paket adalah zip-with-dependencies.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   Jika JUnit paket Appium Java valid, Anda akan menemukan *dependency-jars* direktori di dalam direktori kerja.

   ```
   .
   |– 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
```

Untuk informasi selengkapnya, lihat Jalankan pengujian Appium secara otomatis di Device Farm.

APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_JAR_MISSING_IN_DEPENDE

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat menemukan file JAR di pohon direktori dependency-jars. Harap unzip paket pengujian Anda dan kemudian buka direktori dependency-jars, verifikasi bahwa setidaknya satu file JAR ada di direktori, dan coba lagi.

Dalam contoh berikut, nama paket adalah zip-with-dependencies.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   Jika JUnit paket Appium Java valid, Anda akan menemukan setidaknya satu *jar* file di dalam direktori. *dependency-jars*

   ```
   .
   |— 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
```

Untuk informasi selengkapnya, lihat Jalankan pengujian Appium secara otomatis di Device Farm.

APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_TESTS_JAR_FILE_MISSING

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat menemukan file*-tests.jar dalam paket pengujian Anda. Harap unzip paket pengujian Anda, verifikasi bahwa setidaknya satu file*-tests.jar ada dalam paket, dan coba lagi.

Dalam contoh berikut, nama paket adalah zip-with-dependencies.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   Jika JUnit paket Appium Java valid, Anda akan menemukan setidaknya satu *jar* file seperti *acme-android-appium-1.0-SNAPSHOT-tests.jar* dalam contoh kami. Nama file mungkin berbeda, tetapi harus diakhiri dengan*-tests.jar*.

   ```
   .
   |– 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
```

Untuk informasi selengkapnya, lihat Jalankan pengujian Appium secara otomatis di Device Farm.

APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_CLASS_FILE_MISSING_IN_T

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat menemukan file kelas dalam file JAR tes. Harap unzip paket pengujian Anda dan kemudian unjar file JAR tes, verifikasi bahwa setidaknya satu file kelas ada di dalam file JAR, dan coba lagi.

Dalam contoh berikut, nama paket adalah zip-with-dependencies.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   Anda harus menemukan setidaknya satu file jar seperti *acme-android-appium-1.0-SNAPSHOT-tests.jar* dalam contoh kami. Nama file mungkin berbeda, tetapi harus diakhiri dengan*-tests.jar*.

   ```
   .
   |– 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. Untuk mengekstrak file dari file jar, Anda dapat menjalankan perintah berikut:

```
$ jar xf acme-android-appium-1.0-SNAPSHOT-tests.jar
```

4. Setelah Anda berhasil mengekstrak file, jalankan perintah berikut:

```
$ tree .
```

Anda harus menemukan setidaknya satu kelas di pohon direktori kerja:

```
.
|— 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
```

Lihat informasi yang lebih lengkap di Jalankan pengujian Appium secara otomatis di Device Farm.

Memecahkan masalah pengujian Appium Python di AWS Device Farm

Topik berikut mencantumkan pesan kesalahan yang terjadi selama pengunggahan pengujian Appium Python dan merekomendasikan solusi untuk menyelesaikan setiap kesalahan.

APPIUM_PYTHON_TEST_PACKAGE_UNZIP_FAILED

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat membuka file ZIP uji Appium Anda. Harap verifikasi bahwa file tersebut valid dan coba lagi.

Pastikan Anda dapat membuka zip paket pengujian tanpa kesalahan. Dalam contoh berikut, nama paket adalah test_bundle.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip test_bundle.zip
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   Paket Appium Python yang valid harus menghasilkan output seperti berikut:

   ```
   .
   |-- 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
```

Untuk informasi selengkapnya, lihat Jalankan pengujian Appium secara otomatis di Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_DEPENDENCY_WHEEL_MISSING

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat menemukan file roda ketergantungan di pohon direktori ruang kemudi. Harap unzip paket pengujian Anda dan kemudian buka direktori ruang kemudi, verifikasi bahwa setidaknya satu file roda ada di direktori, dan coba lagi.

Pastikan Anda dapat membuka zip paket pengujian tanpa kesalahan. Dalam contoh berikut, nama paket adalah test_bundle.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip test_bundle.zip
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   Jika paket Appium Python valid, Anda akan menemukan setidaknya satu file *.whl* dependen seperti file yang disorot di dalam direktori. *wheelhouse*

   ```
   .
   |-- 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
```

Untuk informasi selengkapnya, lihat Jalankan pengujian Appium secara otomatis di Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_INVALID_PLATFORM

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami menemukan setidaknya satu file roda menentukan platform yang tidak kami dukung. Harap unzip paket pengujian Anda dan kemudian buka direktori ruang kemudi, verifikasi bahwa nama file roda diakhiri dengan -any.whl atau -linux_x86_64.whl, dan coba lagi.

Pastikan Anda dapat membuka zip paket pengujian tanpa kesalahan. Dalam contoh berikut, nama paket adalah test_bundle.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip test_bundle.zip
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   Jika paket Appium Python valid, Anda akan menemukan setidaknya satu file *.whl* dependen seperti file yang disorot di dalam direktori. *wheelhouse* Nama file mungkin berbeda, tetapi harus diakhiri dengan *-any.whl* atau*-linux_x86_64.whl*, yang menentukan platform. Platform lain seperti windows tidak didukung.

   ```
   .
   |-- 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
```

Untuk informasi selengkapnya, lihat Jalankan pengujian Appium secara otomatis di Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_TEST_DIR_MISSING

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat menemukan direktori tes di dalam paket pengujian Anda. Harap unzip paket pengujian Anda, verifikasi bahwa direktori tes ada di dalam paket, dan coba lagi.

Pastikan Anda dapat membuka zip paket pengujian tanpa kesalahan. Dalam contoh berikut, nama paket adalah test_bundle.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip test_bundle.zip
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   Jika paket Appium Python valid, Anda akan menemukan *tests* direktori di dalam direktori kerja.

   ```
   .
   |-- 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
```

Untuk informasi selengkapnya, lihat Jalankan pengujian Appium secara otomatis di Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_INVALID_TEST_FILE_NAME

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat menemukan file pengujian yang valid di pohon direktori tes. Harap unzip paket pengujian Anda dan kemudian buka direktori tes, verifikasi bahwa setidaknya satu nama file dimulai atau diakhiri dengan kata kunci “test”, dan coba lagi.

Pastikan Anda dapat membuka zip paket pengujian tanpa kesalahan. Dalam contoh berikut, nama paket adalah test_bundle.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip test_bundle.zip
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   Jika paket Appium Python valid, Anda akan menemukan *tests* direktori di dalam direktori kerja. Nama file mungkin berbeda, tetapi harus dimulai dengan *test_* atau diakhiri dengan*_test.py*.

   ```
   .
   |-- 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
```

Untuk informasi selengkapnya, lihat Jalankan pengujian Appium secara otomatis di Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_REQUIREMENTS_TXT_FILE_MISSING

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat menemukan file requirements.txt di dalam paket pengujian Anda. Harap unzip paket pengujian Anda, verifikasi bahwa file requirements.txt ada di dalam paket, dan coba lagi.

Pastikan Anda dapat membuka zip paket pengujian tanpa kesalahan. Dalam contoh berikut, nama paket adalah test_bundle.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip test_bundle.zip
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   Jika paket Appium Python valid, Anda akan menemukan *`requirements.txt`* file di dalam direktori kerja.

   ```
   .
   |-- 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
```

Untuk informasi selengkapnya, lihat Jalankan pengujian Appium secara otomatis di Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_INVALID_PYTEST_VERSION

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami menemukan versi pytest lebih rendah dari versi minimum 2.8.0 yang kami dukung. Harap ubah versi pytest di dalam file requirements.txt, dan coba lagi.

Pastikan Anda dapat membuka zip paket pengujian tanpa kesalahan. Dalam contoh berikut, nama paket adalah test_bundle.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip test_bundle.zip
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   Anda harus menemukan *requirements.txt* file di dalam direktori kerja.

   ```
   .
   |-- 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. Untuk mendapatkan versi pytest, Anda dapat menjalankan perintah berikut:

```
$ grep "pytest" requirements.txt
```

Anda harus menemukan output seperti berikut:

```
pytest==2.9.0
```

Ini menunjukkan versi pytest, yang dalam contoh ini adalah 2.9.0. Jika paket Appium Python valid, versi pytest harus lebih besar dari atau sama dengan 2.8.0.

Untuk informasi selengkapnya, lihat Jalankan pengujian Appium secara otomatis di Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_INSTALL_DEPENDENCY_WHEELS_FAIL

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami gagal memasang roda ketergantungan. Harap unzip paket pengujian Anda dan kemudian buka file requirements.txt dan direktori ruang kemudi, verifikasi bahwa roda ketergantungan yang ditentukan dalam file requirements.txt sama persis dengan roda ketergantungan di dalam direktori ruang kemudi, dan coba lagi.

Kami sangat menyarankan Anda mengatur virtualenv Python untuk pengujian pengemasan. Berikut adalah contoh aliran menciptakan lingkungan virtual menggunakan Python virtualenv dan kemudian mengaktifkannya:

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

Pastikan Anda dapat membuka zip paket pengujian tanpa kesalahan. Dalam contoh berikut, nama paket adalah test_bundle.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

```
$ unzip test_bundle.zip
```

2. Untuk menguji menginstal file roda, Anda dapat menjalankan perintah berikut:

```
$ pip install --use-wheel --no-index --find-links=./wheelhouse --requirement=./
requirements.txt
```

   Paket Appium Python yang valid harus menghasilkan output seperti berikut:

```
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. Untuk menonaktifkan lingkungan virtual, Anda dapat menjalankan perintah berikut:

```
$ deactivate
```

   Untuk informasi selengkapnya, lihat Jalankan pengujian Appium secara otomatis di Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_PYTEST_COLLECT_FAILED

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami gagal mengumpulkan tes di direktori tes. Harap unzip paket pengujian Anda, sehingga paket pengujian valid dengan menjalankan perintah`py.test --collect-only <path to`

`your tests directory>`, dan coba lagi setelah perintah tidak mencetak kesalahan apa pun.

Kami sangat menyarankan Anda mengatur virtualenv Python untuk pengujian pengemasan. Berikut adalah contoh aliran menciptakan lingkungan virtual menggunakan Python virtualenv dan kemudian mengaktifkannya:

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

Pastikan Anda dapat membuka zip paket pengujian tanpa kesalahan. Dalam contoh berikut, nama paket adalah test_bundle.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip test_bundle.zip
   ```

2. Untuk menginstal file roda, Anda dapat menjalankan perintah berikut:

   ```
   $ pip install --use-wheel --no-index --find-links=./wheelhouse --requirement=./
   requirements.txt
   ```

3. Untuk mengumpulkan tes, Anda dapat menjalankan perintah berikut:

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

   Paket Appium Python yang valid harus menghasilkan output seperti berikut:

   ```
   ==================== 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. Untuk menonaktifkan lingkungan virtual, Anda dapat menjalankan perintah berikut:

```
$ deactivate
```

Untuk informasi selengkapnya, lihat Jalankan pengujian Appium secara otomatis di Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_DEPENDENCY_WHEELS_INSUFFICIENT

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat menemukan dependensi roda yang cukup di direktori ruang kemudi. Silakan unzip paket pengujian Anda, lalu buka direktori ruang kemudi. Verifikasi bahwa Anda memiliki semua dependensi roda yang ditentukan dalam file requirements.txt.

Pastikan Anda dapat membuka zip paket pengujian tanpa kesalahan. Dalam contoh berikut, nama paket adalah test_bundle.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

```
$ unzip test_bundle.zip
```

2. Periksa panjang *requirements.txt* file serta jumlah file *.whl* dependen di direktori ruang kemudi:

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

Jika jumlah file *.whl* dependen kurang dari jumlah baris yang tidak kosong di *requirements.txt* file Anda, maka Anda perlu memastikan yang berikut:

- Ada file *.whl* dependen yang sesuai dengan setiap baris dalam *requirements.txt* file.
- Tidak ada baris lain dalam *requirements.txt* file yang berisi informasi selain nama paket ketergantungan.

- Tidak ada nama ketergantungan yang diduplikasi dalam beberapa baris dalam *`requirements.txt`* file sehingga dua baris dalam file mungkin sesuai dengan satu file *`.whl`* dependen.

AWS Device Farm tidak mendukung baris dalam *`requirements.txt`* file yang tidak secara langsung sesuai dengan paket dependensi, seperti baris yang menentukan opsi global untuk `pip install` perintah tersebut. Lihat Format file Persyaratan untuk daftar opsi global.

Untuk informasi selengkapnya, lihat Jalankan pengujian Appium secara otomatis di Device Farm.

Memecahkan masalah pengujian aplikasi web Appium Python di AWS Device Farm

Topik berikut mencantumkan pesan kesalahan yang terjadi selama pengunggahan pengujian aplikasi Web Appium Python dan merekomendasikan solusi untuk menyelesaikan setiap kesalahan.

APPIUM_WEB_PYTHON_TEST_PACKAGE_UNZIP_FAILED

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat membuka file ZIP uji Appium Anda. Harap verifikasi bahwa file tersebut valid dan coba lagi.

Pastikan Anda dapat membuka zip paket pengujian tanpa kesalahan. Dalam contoh berikut, nama paket adalah test_bundle.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

```
$ unzip test_bundle.zip
```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

```
$ tree .
```

Paket Appium Python yang valid harus menghasilkan output seperti berikut:

```
.
|-- 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
```

Untuk informasi selengkapnya, lihat Jalankan pengujian Appium secara otomatis di Device Farm.

APPIUM_WEB_PYTHON_TEST_PACKAGE_DEPENDENCY_WHEEL_MISSING

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat menemukan file roda ketergantungan di pohon direktori ruang kemudi. Harap unzip paket pengujian Anda dan kemudian buka direktori ruang kemudi, verifikasi bahwa setidaknya satu file roda ada di direktori, dan coba lagi.

Pastikan Anda dapat membuka zip paket pengujian tanpa kesalahan. Dalam contoh berikut, nama paket adalah test_bundle.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip test_bundle.zip
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

Jika paket Appium Python valid, Anda akan menemukan setidaknya satu file *.whl* dependen seperti file yang disorot di dalam direktori. *wheelhouse*

```
.
|-- 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
```

Untuk informasi selengkapnya, lihat Jalankan pengujian Appium secara otomatis di Device Farm.

APPIUM_WEB_PYTHON_TEST_PACKAGE_INVALID_PLATFORM

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami menemukan setidaknya satu file roda menentukan platform yang tidak kami dukung. Harap unzip paket pengujian Anda dan kemudian buka direktori ruang kemudi, verifikasi bahwa nama file roda diakhiri dengan -any.whl atau -linux_x86_64.whl, dan coba lagi.

Pastikan Anda dapat membuka zip paket pengujian tanpa kesalahan. Dalam contoh berikut, nama paket adalah test_bundle.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip test_bundle.zip
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

```
$ tree .
```

Jika paket Appium Python valid, Anda akan menemukan setidaknya satu file *.whl* dependen seperti file yang disorot di dalam direktori. *wheelhouse* Nama file mungkin berbeda, tetapi harus diakhiri dengan *-any.whl* atau*-linux_x86_64.whl*, yang menentukan platform. Platform lain seperti windows tidak didukung.

```
.
|-- 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
```

Untuk informasi selengkapnya, lihat Jalankan pengujian Appium secara otomatis di Device Farm.

APPIUM_WEB_PYTHON_TEST_PACKAGE_TEST_DIR_MISSING

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat menemukan direktori tes di dalam paket pengujian Anda. Harap unzip paket pengujian Anda, verifikasi bahwa direktori tes ada di dalam paket, dan coba lagi.

Pastikan Anda dapat membuka zip paket pengujian tanpa kesalahan. Dalam contoh berikut, nama paket adalah test_bundle.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

```
$ unzip test_bundle.zip
```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

```
$ tree .
```

Jika paket Appium Python valid, Anda akan menemukan *tests* direktori di dalam direktori kerja.

```
.
|-- 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
```

Untuk informasi selengkapnya, lihat Jalankan pengujian Appium secara otomatis di Device Farm.

APPIUM_WEB_PYTHON_TEST_PACKAGE_INVALID_TEST_FILE_NAME

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat menemukan file pengujian yang valid di pohon direktori tes. Harap unzip paket pengujian Anda dan kemudian buka direktori tes, verifikasi bahwa setidaknya satu nama file dimulai atau diakhiri dengan kata kunci “test”, dan coba lagi.

Pastikan Anda dapat membuka zip paket pengujian tanpa kesalahan. Dalam contoh berikut, nama paket adalah test_bundle.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

```
$ unzip test_bundle.zip
```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

```
$ tree .
```

Jika paket Appium Python valid, Anda akan menemukan *tests* direktori di dalam direktori kerja. Nama file mungkin berbeda, tetapi harus dimulai dengan *test_* atau diakhiri dengan*_test.py*.

```
.
|-- 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
```

Untuk informasi selengkapnya, lihat Jalankan pengujian Appium secara otomatis di Device Farm.

APPIUM_WEB_PYTHON_TEST_PACKAGE_REQUIREMENTS_TXT_FILE_MISS

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat menemukan file requirements.txt di dalam paket pengujian Anda. Harap unzip paket pengujian Anda, verifikasi bahwa file requirements.txt ada di dalam paket, dan coba lagi.

Pastikan Anda dapat membuka zip paket pengujian tanpa kesalahan. Dalam contoh berikut, nama paket adalah test_bundle.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

```
$ unzip test_bundle.zip
```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

```
$ tree .
```

Jika paket Appium Python valid, Anda akan menemukan *requirements.txt* file di dalam direktori kerja.

```
.
|-- 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
```

Untuk informasi selengkapnya, lihat Jalankan pengujian Appium secara otomatis di Device Farm.

APPIUM_WEB_PYTHON_TEST_PACKAGE_INVALID_PYTEST_VERSION

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami menemukan versi pytest lebih rendah dari versi minimum 2.8.0 yang kami dukung. Harap ubah versi pytest di dalam file requirements.txt, dan coba lagi.

Pastikan Anda dapat membuka zip paket pengujian tanpa kesalahan. Dalam contoh berikut, nama paket adalah test_bundle.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

```
$ unzip test_bundle.zip
```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

```
$ tree .
```

Anda harus menemukan *requirement.txt* file di dalam direktori kerja.

```
.
|-- 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. Untuk mendapatkan versi pytest, Anda dapat menjalankan perintah berikut:

```
$ grep "pytest" requirements.txt
```

Anda harus menemukan output seperti berikut:

```
pytest==2.9.0
```

Ini menunjukkan versi pytest, yang dalam contoh ini adalah 2.9.0. Jika paket Appium Python valid, versi pytest harus lebih besar dari atau sama dengan 2.8.0.

Untuk informasi selengkapnya, lihat Jalankan pengujian Appium secara otomatis di Device Farm.

APPIUM_WEB_PYTHON_TEST_PACKAGE_INSTALL_DEPENDENCY_WHEELS

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami gagal memasang roda ketergantungan. Harap unzip paket pengujian Anda dan kemudian buka file requirements.txt dan direktori ruang kemudi, verifikasi bahwa roda ketergantungan yang ditentukan dalam file requirements.txt sama persis dengan roda ketergantungan di dalam direktori ruang kemudi, dan coba lagi.

Kami sangat menyarankan Anda mengatur virtualenv Python untuk pengujian pengemasan. Berikut adalah contoh aliran menciptakan lingkungan virtual menggunakan Python virtualenv dan kemudian mengaktifkannya:

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

Pastikan Anda dapat membuka zip paket pengujian tanpa kesalahan. Dalam contoh berikut, nama paket adalah test_bundle.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip test_bundle.zip
   ```

2. Untuk menguji menginstal file roda, Anda dapat menjalankan perintah berikut:

   ```
   $ pip install --use-wheel --no-index --find-links=./wheelhouse --requirement=./
   requirements.txt
   ```

   Paket Appium Python yang valid harus menghasilkan output seperti berikut:

   ```
   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. Untuk menonaktifkan lingkungan virtual, Anda dapat menjalankan perintah berikut:

```
$ deactivate
```

Untuk informasi selengkapnya, lihat Jalankan pengujian Appium secara otomatis di Device Farm.

APPIUM_WEB_PYTHON_TEST_PACKAGE_PYTEST_COLLECT_FAILED

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami gagal mengumpulkan tes di direktori tes. Harap unzip paket pengujian Anda, sehingga paket pengujian valid dengan menjalankan perintah “py.test --collect-only <path to your tests directory>“, dan coba lagi setelah perintah tidak mencetak kesalahan apa pun.

Kami sangat menyarankan Anda mengatur virtualenv Python untuk pengujian pengemasan. Berikut adalah contoh aliran menciptakan lingkungan virtual menggunakan Python virtualenv dan kemudian mengaktifkannya:

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

Pastikan Anda dapat membuka zip paket pengujian tanpa kesalahan. Dalam contoh berikut, nama paket adalah test_bundle.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

```
$ unzip test_bundle.zip
```

2. Untuk menginstal file roda, Anda dapat menjalankan perintah berikut:

```
$ pip install --use-wheel --no-index --find-links=./wheelhouse --requirement=./
requirements.txt
```

3. Untuk mengumpulkan tes, Anda dapat menjalankan perintah berikut:

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

Paket Appium Python yang valid harus menghasilkan output seperti berikut:

```
==================== 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. Untuk menonaktifkan lingkungan virtual, Anda dapat menjalankan perintah berikut:

```
$ deactivate
```

Lihat informasi yang lebih lengkap di Jalankan pengujian Appium secara otomatis di Device Farm.

Memecahkan masalah pengujian instrumentasi di AWS Device Farm

Topik berikut mencantumkan pesan galat yang terjadi selama pengunggahan pengujian Instrumentasi dan merekomendasikan solusi untuk mengatasi setiap kesalahan.

> **Note**
>
> Untuk pertimbangan penting saat menggunakan pengujian Instrumentasi di AWS Device Farm, lihat. Instrumentasi untuk Android dan AWS Device Farm

INSTRUMENTATION_TEST_PACKAGE_UNZIP_FAILED

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

```
Warning: We could not open your test APK file. Please verify that the file is valid and
    try again.
```

Pastikan Anda dapat membuka zip paket pengujian tanpa kesalahan. Dalam contoh berikut, nama paket adalah app-debug-androidTest-unaligned.apk.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip app-debug-androidTest-unaligned.apk
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   Paket uji Instrumentasi yang valid akan menghasilkan output seperti berikut:

   ```
   .
   |-- AndroidManifest.xml
   |--  classes.dex
   |-- resources.arsc
   |-- LICENSE-junit.txt
   |-- junit (directory)
   `-- META-INF (directory)
   ```

   Untuk informasi selengkapnya, lihat Instrumentasi untuk Android dan AWS Device Farm.

INSTRUMENTATION_TEST_PACKAGE_AAPT_DEBUG_BADGING_FAILED

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

```
We could not extract information about your test package. Please verify that the
      test package is valid by running the command "aapt debug badging <path to your
 test
      package>", and try again after the command does not print any error.
```

Selama proses validasi upload, Device Farm mem-parsing informasi dari output perintah. `aapt debug badging <path to your package>`

Pastikan Anda dapat menjalankan perintah ini pada paket pengujian Instrumentasi Anda dengan sukses.

Dalam contoh berikut, nama paket adalah app-debug-androidTest-unaligned.apk.

- Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

```
$ aapt debug badging app-debug-androidTest-unaligned.apk
```

Paket uji Instrumentasi yang valid akan menghasilkan output seperti berikut:

```
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'
```

Untuk informasi selengkapnya, lihat Instrumentasi untuk Android dan AWS Device Farm.

INSTRUMENTATION_TEST_PACKAGE_INSTRUMENTATION_RUNNER_VALU

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

```
We could not find the instrumentation runner value in the AndroidManifest.xml.
      Please verify the test package is valid by running the command "aapt dump xmltree
 <path to
      your test package> AndroidManifest.xml", and try again after finding the
 instrumentation
      runner value behind the keyword "instrumentation."
```

Selama proses validasi upload, Device Farm mem-parsing nilai instrumentasi runner dari pohon parse XMLuntuk file XMLyang terdapat di dalam paket. Anda dapat menggunakan perintah berikut:`aapt dump xmltree <path to your package> AndroidManifest.xml`.

Pastikan Anda dapat menjalankan perintah ini pada paket pengujian Instrumentasi Anda dan menemukan nilai instrumentasi dengan sukses.

Dalam contoh berikut, nama paket adalah app-debug-androidTest-unaligned.apk.

- Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

```
$ aapt dump xmltree app-debug-androidTest-unaligned.apk AndroidManifest.xml | grep
 -A5 "instrumentation"
```

  Paket uji Instrumentasi yang valid akan menghasilkan output seperti berikut:

```
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
```

  Untuk informasi selengkapnya, lihat Instrumentasi untuk Android dan AWS Device Farm.

INSTRUMENTATION_TEST_PACKAGE_AAPT_DUMP_XMLTREE_FAILED

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

```
We could not find the valid AndroidManifest.xml in your test package. Please
      verify that the test package is valid by running the command "aapt dump xmltree
 <path to
      your test package> AndroidManifest.xml", and try again after the command does not
 print any
```

```
      error.
```

Selama proses validasi upload, Device Farm mem-parsing informasi dari pohon parse XMLuntuk file XMLyang terdapat di dalam paket menggunakan perintah berikut: `aapt dump xmltree <path to your package> AndroidManifest.xml`

Pastikan bahwa Anda dapat menjalankan perintah ini pada paket pengujian instrumentasi Anda berhasil.

Dalam contoh berikut, nama paket adalah app-debug-androidTest-unaligned.apk.

- Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

  ```
  $ aapt dump xmltree app-debug-androidTest-unaligned.apk AndroidManifest.xml
  ```

  Paket uji Instrumentasi yang valid akan menghasilkan output seperti berikut:

  ```
  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")
```

Untuk informasi selengkapnya, lihat Instrumentasi untuk Android dan AWS Device Farm.

INSTRUMENTASI_TEST_PACKAGE_TEST_PACKAGE_NAME_VALUE_MISSIN

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

```
We could not find the package name in your test package. Please verify that the
      test package is valid by running the command "aapt debug badging <path to your
 test
      package>", and try again after finding the package name value behind the keyword
 "package:
      name."
```

Selama proses validasi upload, Device Farm mem-parsing nilai nama paket dari output perintah berikut:. `aapt debug badging <path to your package>`

Pastikan bahwa Anda dapat menjalankan perintah ini pada paket pengujian Instrumentasi Anda dan menemukan nilai nama paket berhasil.

Dalam contoh berikut, nama paket adalah app-debug-androidTest-unaligned.apk.

- Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

  ```
  $ aapt debug badging app-debug-androidTest-unaligned.apk | grep "package: name="
  ```

  Paket uji Instrumentasi yang valid akan menghasilkan output seperti berikut:

  ```
  package: name='com.amazon.aws.adf.android.referenceapp.test' versionCode=''
   versionName='' platformBuildVersionName='5.1.1-1819727'
  ```

  Untuk informasi selengkapnya, lihat Instrumentasi untuk Android dan AWS Device Farm.

Memecahkan masalah pengujian aplikasi iOS di AWS Device Farm

Topik berikut mencantumkan pesan kesalahan yang terjadi selama pengunggahan pengujian aplikasi iOS dan merekomendasikan solusi untuk menyelesaikan setiap kesalahan.

> **Note**
>
> Petunjuk di bawah ini didasarkan pada Linux x86_64 dan Mac.

IOS_APP_UNZIP_FAILED

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat membuka aplikasi Anda. Harap verifikasi bahwa file tersebut valid dan coba lagi.

Pastikan Anda dapat unzip paket aplikasi tanpa kesalahan. Dalam contoh berikut, nama paket adalah AWSDeviceFarmi OSReference App.ipa.

1. Salin paket aplikasi Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip AWSDeviceFarmiOSReferenceApp.ipa
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   Paket aplikasi iOS yang valid harus menghasilkan output seperti berikut:

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

   Untuk informasi selengkapnya, lihat Pengujian iOS di AWS Device Farm.

IOS_APP_PAYLOAD_DIR_MISSING

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat menemukan direktori Payload di dalam aplikasi Anda. Silakan unzip aplikasi Anda, verifikasi bahwa direktori Payload ada di dalam paket, dan coba lagi.

Dalam contoh berikut, nama paket adalah AWSDeviceFarmi OSReference App.ipa.

1. Salin paket aplikasi Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip AWSDeviceFarmiOSReferenceApp.ipa
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   Jika paket aplikasi iOS valid, Anda akan menemukan *Payload* direktori di dalam direktori kerja.

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

   Untuk informasi selengkapnya, lihat Pengujian iOS di AWS Device Farm.

IOS_APP_APP_DIR_MISSING

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat menemukan direktori.app di dalam direktori Payload. Silakan unzip aplikasi Anda dan kemudian buka direktori Payload, verifikasi bahwa direktori.app ada di dalam direktori, dan coba lagi.

Dalam contoh berikut, nama paket adalah AWSDeviceFarmi OSReference App.ipa.

1. Salin paket aplikasi Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip AWSDeviceFarmiOSReferenceApp.ipa
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   Jika paket aplikasi iOS valid, Anda akan menemukan *`.app`* direktori seperti *`AWSDeviceFarmiOSReferenceApp.app`* dalam contoh kita di dalam *`Payload`* direktori.

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

   Untuk informasi selengkapnya, lihat Pengujian iOS di AWS Device Farm.

IOS_APP_PLIST_FILE_MISSING

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat menemukan file Info.plist di dalam direktori.app. Silakan unzip aplikasi Anda dan kemudian buka direktori.app, verifikasi bahwa file Info.plist ada di dalam direktori, dan coba lagi.

Dalam contoh berikut, nama paket adalah AWSDeviceFarmi OSReference App.ipa.

1. Salin paket aplikasi Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

```
$ tree .
```

Jika paket aplikasi iOS valid, Anda akan menemukan *Info.plist* file di dalam *.app* direktori seperti *AWSDeviceFarmiOSReferenceApp.app* dalam contoh kita.

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

Untuk informasi selengkapnya, lihat Pengujian iOS di AWS Device Farm.

IOS_APP_CPU_ARCHITECTURE_VALUE_MISSING

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat menemukan nilai arsitektur CPU di file Info.plist. Silakan unzip aplikasi Anda dan kemudian buka file Info.plist di dalam direktori.app, verifikasi bahwa kunci "UIRequiredDeviceCapabilities" ditentukan, dan coba lagi.

Dalam contoh berikut, nama paket adalah AWSDeviceFarmi OSReference App.ipa.

1. Salin paket aplikasi Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

```
$ tree .
```

Anda harus menemukan *Info.plist* file di dalam *.app* direktori seperti *AWSDeviceFarmiOSReferenceApp.app* dalam contoh kita:

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

3. Untuk menemukan nilai arsitektur CPU, Anda dapat membuka Info.plist menggunakan Xcode atau Python.

Untuk Python, Anda dapat menginstal modul biplist dengan menjalankan perintah berikut:

```
$ pip install biplist
```

4. Selanjutnya, buka Python dan jalankan perintah berikut:

```
import biplist
info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
Info.plist')
print info_plist['UIRequiredDeviceCapabilities']
```

Paket aplikasi iOS yang valid harus menghasilkan output seperti berikut:

```
['armv7']
```

Untuk informasi selengkapnya, lihat Pengujian iOS di AWS Device Farm.

IOS_APP_PLATFORM_VALUE_MISSING

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat menemukan nilai platform di file Info.plist. Silakan unzip aplikasi Anda dan kemudian buka file Info.plist di dalam direktori.app, verifikasi bahwa kunci "CFBundleSupportedPlatforms" ditentukan, dan coba lagi.

Dalam contoh berikut, nama paket adalah AWSDeviceFarmi OSReference App.ipa.

1. Salin paket aplikasi Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip AWSDeviceFarmiOSReferenceApp.ipa
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   Anda harus menemukan *Info.plist* file di dalam *.app* direktori seperti *AWSDeviceFarmiOSReferenceApp.app* dalam contoh kita:

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

3. Untuk menemukan nilai platform, Anda dapat membuka Info.plist menggunakan Xcode atau Python.

   Untuk Python, Anda dapat menginstal modul biplist dengan menjalankan perintah berikut:

   ```
   $ pip install biplist
   ```

4. Selanjutnya, buka Python dan jalankan perintah berikut:

   ```
   import biplist
   info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
   Info.plist')
   print info_plist['CFBundleSupportedPlatforms']
   ```

Paket aplikasi iOS yang valid harus menghasilkan output seperti berikut:

```
['iPhoneOS']
```

Untuk informasi selengkapnya, lihat Pengujian iOS di AWS Device Farm.

IOS_APP_WRONG_PLATFORM_DEVICE_VALUE

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami menemukan nilai perangkat platform salah dalam file Info.plist. Silakan unzip aplikasi Anda dan kemudian buka file Info.plist di dalam direktori.app, verifikasi bahwa nilai kunci "CFBundleSupportedPlatforms" tidak mengandung kata kunci “simulator”, dan coba lagi.

Dalam contoh berikut, nama paket adalah AWSDeviceFarmi OSReference App.ipa.

1. Salin paket aplikasi Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip AWSDeviceFarmiOSReferenceApp.ipa
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   Anda harus menemukan *Info.plist* file di dalam *.app* direktori seperti *AWSDeviceFarmiOSReferenceApp.app* dalam contoh kita:

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

3. Untuk menemukan nilai platform, Anda dapat membuka Info.plist menggunakan Xcode atau Python.

   Untuk Python, Anda dapat menginstal modul biplist dengan menjalankan perintah berikut:

   ```
   $ pip install biplist
   ```

4. Selanjutnya, buka Python dan jalankan perintah berikut:

   ```
   import biplist
   info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
   Info.plist')
   print info_plist['CFBundleSupportedPlatforms']
   ```

   Paket aplikasi iOS yang valid harus menghasilkan output seperti berikut:

   ```
   ['iPhoneOS']
   ```

   Jika aplikasi iOS valid, nilainya tidak boleh mengandung kata kunci`simulator`.

   Untuk informasi selengkapnya, lihat Pengujian iOS di AWS Device Farm.

IOS_APP_FORM_FACTOR_VALUE_MISSING

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat menemukan nilai faktor bentuk dalam file Info.plist. Silakan unzip aplikasi Anda dan kemudian buka file Info.plist di dalam direktori.app, verifikasi bahwa kunci "UIDeviceKeluarga” ditentukan, dan coba lagi.

Dalam contoh berikut, nama paket adalah AWSDeviceFarmi OSReference App.ipa.

1. Salin paket aplikasi Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip AWSDeviceFarmiOSReferenceApp.ipa
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

```
$ tree .
```

Anda harus menemukan *Info.plist* file di dalam *.app* direktori seperti *AWSDeviceFarmiOSReferenceApp.app* dalam contoh kita:

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

3. Untuk menemukan nilai faktor bentuk, Anda dapat membuka Info.plist menggunakan Xcode atau Python.

Untuk Python, Anda dapat menginstal modul biplist dengan menjalankan perintah berikut:

```
$ pip install biplist
```

4. Selanjutnya, buka Python dan jalankan perintah berikut:

```
import biplist
info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
Info.plist')
print info_plist['UIDeviceFamily']
```

Paket aplikasi iOS yang valid harus menghasilkan output seperti berikut:

```
[1, 2]
```

Untuk informasi selengkapnya, lihat Pengujian iOS di AWS Device Farm.

IOS_APP_PACKAGE_NAME_VALUE_MISSING

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat menemukan nilai nama paket di file Info.plist. Silakan unzip aplikasi Anda dan kemudian buka file Info.plist di dalam direktori.app, verifikasi bahwa kunci "CFBundleIdentifier” ditentukan, dan coba lagi.

Dalam contoh berikut, nama paket adalah AWSDeviceFarmi OSReference App.ipa.

1. Salin paket aplikasi Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip AWSDeviceFarmiOSReferenceApp.ipa
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   Anda harus menemukan *Info.plist* file di dalam *.app* direktori seperti *AWSDeviceFarmiOSReferenceApp.app* dalam contoh kita:

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

3. Untuk menemukan nilai nama paket, Anda dapat membuka Info.plist menggunakan Xcode atau Python.

   Untuk Python, Anda dapat menginstal modul biplist dengan menjalankan perintah berikut:

   ```
   $ pip install biplist
   ```

4. Selanjutnya, buka Python dan jalankan perintah berikut:

   ```
   import biplist
   info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
   Info.plist')
   print info_plist['CFBundleIdentifier']
   ```

Paket aplikasi iOS yang valid harus menghasilkan output seperti berikut:

```
Amazon.AWSDeviceFarmiOSReferenceApp
```

Untuk informasi selengkapnya, lihat Pengujian iOS di AWS Device Farm.

IOS_APP_EXECUTABLE_VALUE_MISSING

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat menemukan nilai yang dapat dieksekusi di file Info.plist. Silakan unzip aplikasi Anda dan kemudian buka file Info.plist di dalam direktori.app, verifikasi bahwa kunci "CFBundleExecutable" ditentukan, dan coba lagi.

Dalam contoh berikut, nama paket adalah AWSDeviceFarmi OSReference App.ipa.

1. Salin paket aplikasi Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip AWSDeviceFarmiOSReferenceApp.ipa
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   Anda harus menemukan *Info.plist* file di dalam *.app* direktori seperti *AWSDeviceFarmiOSReferenceApp.app* dalam contoh kita:

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

3. Untuk menemukan nilai yang dapat dieksekusi, Anda dapat membuka Info.plist menggunakan Xcode atau Python.

   Untuk Python, Anda dapat menginstal modul biplist dengan menjalankan perintah berikut:

   ```
   $ pip install biplist
   ```

4. Selanjutnya, buka Python dan jalankan perintah berikut:

   ```
   import biplist
   info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
   Info.plist')
   print info_plist['CFBundleExecutable']
   ```

   Paket aplikasi iOS yang valid harus menghasilkan output seperti berikut:

   ```
   AWSDeviceFarmiOSReferenceApp
   ```

   Untuk informasi selengkapnya, lihat Pengujian iOS di AWS Device Farm.

XCTest Pengujian pemecahan masalah di AWS Device Farm

Topik berikut mencantumkan pesan kesalahan yang terjadi selama pengunggahan XCTest pengujian dan merekomendasikan solusi untuk menyelesaikan setiap kesalahan.

> **Note**
>
> Petunjuk di bawah ini menganggap Anda menggunakan macOS.

XCTEST_TEST_PACKAGE_UNZIP_FAILED

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat membuka file ZIP pengujian Anda. Harap verifikasi bahwa file tersebut valid dan coba lagi.

Pastikan Anda dapat unzip paket aplikasi tanpa kesalahan. Dalam contoh berikut, nama paket adalah swiftExampleTests.xctest-1.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

```
$ unzip swiftExampleTests.xctest-1.zip
```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

```
$ tree .
```

XCTest Paket yang valid harus menghasilkan output seperti berikut:

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

Untuk informasi selengkapnya, lihat Mengintegrasikan Device Farm dengan XCTest iOS.

XCTEST_TEST_PACKAGE_XCTEST_DIR_MISSING

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat menemukan direktori.xctest di dalam paket pengujian Anda. Harap unzip paket pengujian Anda, verifikasi bahwa direktori.xctest ada di dalam paket, dan coba lagi.

Dalam contoh berikut, nama paket adalah swiftExampleTests.xctest-1.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

```
$ unzip swiftExampleTests.xctest-1.zip
```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

```
$ tree .
```

Jika XCTest paket valid, Anda akan menemukan direktori dengan nama yang mirip dengan *swiftExampleTests.xctest* di dalam direktori kerja. Nama harus diakhiri dengan*.xctest*.

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

Untuk informasi selengkapnya, lihat Mengintegrasikan Device Farm dengan XCTest iOS.

XCTEST_TEST_PACKAGE_PLIST_FILE_MISSING

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat menemukan file Info.plist di dalam direktori.xctest. Silakan unzip paket pengujian Anda dan kemudian buka direktori.xctest, verifikasi bahwa file Info.plist ada di dalam direktori, dan coba lagi.

Dalam contoh berikut, nama paket adalah swiftExampleTests.xctest-1.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip swiftExampleTests.xctest-1.zip
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   Jika XCTest paket valid, Anda akan menemukan *Info.plist* file di dalam *.xctest* direktori. Dalam contoh kita di bawah ini, direktori disebut*swiftExampleTests.xctest*.

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

Untuk informasi selengkapnya, lihat Mengintegrasikan Device Farm dengan XCTest iOS.

XCTEST_TEST_PACKAGE_PACKAGE_NAME_VALUE_MISSING

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat menemukan nilai nama paket di file Info.plist. Harap unzip paket pengujian Anda dan kemudian buka file Info.plist, verifikasi bahwa kunci "CFBundleIdentifier" ditentukan, dan coba lagi.

Dalam contoh berikut, nama paket adalah swiftExampleTests.xctest-1.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip swiftExampleTests.xctest-1.zip
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   Anda harus menemukan *Info.plist* file di dalam *.xctest* direktori seperti *swiftExampleTests.xctest* dalam contoh kita:

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

3. Untuk menemukan nilai nama paket, Anda dapat membuka Info.plist menggunakan Xcode atau Python.

   Untuk Python, Anda dapat menginstal modul biplist dengan menjalankan perintah berikut:

   ```
   $ pip install biplist
   ```

4. Selanjutnya, buka Python dan jalankan perintah berikut:

   ```
   import biplist
   info_plist = biplist.readPlist('swiftExampleTests.xctest/Info.plist')
   print info_plist['CFBundleIdentifier']
   ```

   Paket XCtest aplikasi yang valid harus menghasilkan output seperti berikut:

   ```
   com.amazon.kanapka.swiftExampleTests
   ```

   Untuk informasi selengkapnya, lihat Mengintegrasikan Device Farm dengan XCTest iOS.

XCTEST_TEST_PACKAGE_EXECUTABLE_VALUE_MISSING

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

> **Warning**
>
> Kami tidak dapat menemukan nilai yang dapat dieksekusi di file Info.plist. Silakan unzip paket pengujian Anda dan kemudian buka file Info.plist, verifikasi bahwa kunci "CFBundleDapat dieksekusi” ditentukan, dan coba lagi.

Dalam contoh berikut, nama paket adalah swiftExampleTests.xctest-1.zip.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip swiftExampleTests.xctest-1.zip
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

```
$ tree .
```

Anda harus menemukan *Info.plist* file di dalam *.xctest* direktori seperti *swiftExampleTests.xctest* dalam contoh kita:

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

3. Untuk menemukan nilai nama paket, Anda dapat membuka Info.plist menggunakan Xcode atau Python.

   Untuk Python, Anda dapat menginstal modul biplist dengan menjalankan perintah berikut:

   ```
   $ pip install biplist
   ```

4. Selanjutnya, buka Python dan jalankan perintah berikut:

   ```
   import biplist
   info_plist = biplist.readPlist('swiftExampleTests.xctest/Info.plist')
   print info_plist['CFBundleExecutable']
   ```

   Paket XCtest aplikasi yang valid harus menghasilkan output seperti berikut:

   ```
   swiftExampleTests
   ```

   Untuk informasi selengkapnya, lihat Mengintegrasikan Device Farm dengan XCTest iOS.

Memecahkan masalah pengujian XCTest UI di AWS Device Farm

Topik berikut mencantumkan pesan galat yang terjadi selama pengunggahan pengujian XCTest UI dan merekomendasikan solusi untuk mengatasi setiap kesalahan.

> **Note**
>
> Petunjuk di bawah ini didasarkan pada Linux x86_64 dan Mac.

XCTEST_UI_TEST_PACKAGE_UNZIP_FAILED

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

```
We could not open your test IPA file. Please verify that the file is valid and try again.
```

Pastikan Anda dapat unzip paket aplikasi tanpa kesalahan. Dalam contoh berikut, nama paket adalah Swift-sample-ui.ipa.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   Paket aplikasi iOS yang valid harus menghasilkan output seperti berikut:

   ```
   .
   `-- Payload (directory)
           `-- swift-sampleUITests-Runner.app (directory)
                       |-- Info.plist
                       |-- Plugins (directory)
                       |       `swift-sampleUITests.xctest (directory)
                       |                         |-- Info.plist
                       |                         `-- (any other files)
                       `-- (any other files)
   ```

   Untuk informasi selengkapnya, lihat Mengintegrasikan XCTest UI untuk iOS dengan Device Farm.

XCTEST_UI_TEST_PACKAGE_PAYLOAD_DIR_MISSING

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

```
We could not find the Payload directory inside your test package. Please
unzip your test package, verify that the Payload directory is inside the
package, and try again.
```

Dalam contoh berikut, nama paket adalah Swift-sample-ui.ipa.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   Jika paket XCTest UI valid, Anda akan menemukan *`Payload`* direktori di dalam direktori kerja.

   ```
   .
   `-- Payload (directory)
           `-- swift-sampleUITests-Runner.app (directory)
                       |-- Info.plist
                       |-- Plugins (directory)
                       |       `swift-sampleUITests.xctest (directory)
                       |                       |-- Info.plist
                       |                       `-- (any other files)
                       `-- (any other files)
   ```

   Untuk informasi selengkapnya, lihat Mengintegrasikan XCTest UI untuk iOS dengan Device Farm.

XCTEST_UI_TEST_PACKAGE_APP_DIR_MISSING

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

```
We could not find the .app directory inside the Payload directory. Please
unzip your test package and then open the Payload directory, verify that
the .app directory is inside the directory, and try again.
```

Dalam contoh berikut, nama paket adalah Swift-sample-ui.ipa.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

```
$ unzip swift-sample-UI.ipa
```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

```
$ tree .
```

Jika paket XCTest UI valid, Anda akan menemukan *.app* direktori seperti *swift-sampleUITests-Runner.app* dalam contoh kita di dalam *Payload* direktori.

```
.
`-- Payload (directory)
        `-- swift-sampleUITests-Runner.app (directory)
                    |-- Info.plist
                    |-- Plugins (directory)
                    |       `swift-sampleUITests.xctest (directory)
                    |                       |-- Info.plist
                    |                       `-- (any other files)
                    `-- (any other files)
```

Untuk informasi selengkapnya, lihat Mengintegrasikan XCTest UI untuk iOS dengan Device Farm.

XCTEST_UI_TEST_PACKAGE_PLUGINS_DIR_MISSING

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

```
We could not find the Plugins directory inside the .app directory. Please unzip your test package and then open the .app directory, verify that the Plugins directory is inside the directory, and try again.
```

Dalam contoh berikut, nama paket adalah Swift-sample-ui.ipa.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

```
$ unzip swift-sample-UI.ipa
```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

```
$ tree .
```

Jika paket XCTest UI valid, Anda akan menemukan *Plugins* direktori di dalam *.app* direktori. Dalam contoh kita, direktori disebut*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)
```

Untuk informasi selengkapnya, lihat Mengintegrasikan XCTest UI untuk iOS dengan Device Farm.

XCTEST_UI_TEST_PACKAGE_XCTEST_DIR_MISSING_IN_PLUGINS_DIR

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

```
We could not find the .xctest directory inside the plugins directory.
Please unzip your test package and then open the plugins directory, verify
that the .xctest directory is inside the directory, and try again.
```

Dalam contoh berikut, nama paket adalah Swift-sample-ui.ipa.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

```
$ unzip swift-sample-UI.ipa
```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

```
$ tree .
```

Jika paket XCTest UI valid, Anda akan menemukan *.xctest* direktori di dalam *Plugins* direktori. Dalam contoh kita, direktori disebut*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)
```

Untuk informasi selengkapnya, lihat Mengintegrasikan XCTest UI untuk iOS dengan Device Farm.

XCTEST_UI_TEST_PACKAGE_PLIST_FILE_MISSING

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

`We could not find the Info.plist file inside the .app directory. Please unzip your test package and then open the .app directory, verify that the Info.plist file is inside the directory, and try again.`

Dalam contoh berikut, nama paket adalah Swift-sample-ui.ipa.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   Jika paket XCTest UI valid, Anda akan menemukan *Info.plist* file di dalam *.app* direktori. Dalam contoh kita di bawah ini, direktori disebut*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)
```

Untuk informasi selengkapnya, lihat Mengintegrasikan XCTest UI untuk iOS dengan Device Farm.

XCTEST_UI_TEST_PACKAGE_PLIST_FILE_MISSING_IN_XCTEST_DIR

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

```
We could not find the Info.plist file inside the .xctest directory. Please unzip your test package and then open the .xctest directory, verify that the Info.plist file is inside the directory, and try again.
```

Dalam contoh berikut, nama paket adalah Swift-sample-ui.ipa.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   Jika paket XCTest UI valid, Anda akan menemukan *Info.plist* file di dalam *.xctest* direktori. Dalam contoh kita di bawah ini, direktori disebut*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)
```

Untuk informasi selengkapnya, lihat Mengintegrasikan XCTest UI untuk iOS dengan Device Farm.

XCTEST_UI_TEST_PACKAGE_CPU_ARCHITECTURE_VALUE_MISSING

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

```
We could not the CPU architecture value in the Info.plist file. Please unzip your test package and then open the Info.plist file inside the .app directory, verify that the key "UIRequiredDeviceCapabilities" is specified, and try again.
```

Dalam contoh berikut, nama paket adalah Swift-sample-ui.ipa.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   Anda harus menemukan *Info.plist* file di dalam *.app* direktori seperti *swift-sampleUITests-Runner.app* dalam contoh kita:

   ```
   .
   `-- Payload (directory)
           `-- swift-sampleUITests-Runner.app (directory)
                       |-- Info.plist
                       |-- Plugins (directory)
                       |       `swift-sampleUITests.xctest (directory)
                       |                       |-- Info.plist
                       |                       `-- (any other files)
                       `-- (any other files)
   ```

3. Untuk menemukan nilai arsitektur CPU, Anda dapat membuka Info.plist menggunakan Xcode atau Python.

   Untuk Python, Anda dapat menginstal modul biplist dengan menjalankan perintah berikut:

   ```
   $ pip install biplist
   ```

4. Selanjutnya, buka Python dan jalankan perintah berikut:

   ```
   import biplist
   info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/
   Info.plist')
   print info_plist['UIRequiredDeviceCapabilities']
   ```

   Paket XCtest UI yang valid harus menghasilkan output seperti berikut:

   ```
   ['armv7']
   ```

   Untuk informasi selengkapnya, lihat Mengintegrasikan XCTest UI untuk iOS dengan Device Farm.

XCTEST_UI_TEST_PACKAGE_PLATFORM_VALUE_MISSING

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

```
We could not find the platform value in the Info.plist. Please unzip your
test package and then open the Info.plist file inside the .app directory,
verify that the key "CFBundleSupportedPlatforms" is specified, and try
again.
```

Dalam contoh berikut, nama paket adalah Swift-sample-ui.ipa.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

```
$ tree .
```

Anda harus menemukan *Info.plist* file di dalam *.app* direktori seperti *swift-sampleUITests-Runner.app* dalam contoh kita:

```
.
`-- Payload (directory)
        `-- swift-sampleUITests-Runner.app (directory)
                        |-- Info.plist
                        |-- Plugins (directory)
                        |       `swift-sampleUITests.xctest (directory)
                        |                         |-- Info.plist
                        |                         `-- (any other files)
                        `-- (any other files)
```

3. Untuk menemukan nilai platform, Anda dapat membuka Info.plist menggunakan Xcode atau Python.

   Untuk Python, Anda dapat menginstal modul biplist dengan menjalankan perintah berikut:

   ```
   $ pip install biplist
   ```

4. Selanjutnya, buka Python dan jalankan perintah berikut:

   ```
   import biplist
   info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
   print info_plist['CFBundleSupportedPlatforms']
   ```

   Paket XCtest UI yang valid harus menghasilkan output seperti berikut:

   ```
   ['iPhoneOS']
   ```

   Untuk informasi selengkapnya, lihat Mengintegrasikan XCTest UI untuk iOS dengan Device Farm.

XCTEST_UI_TEST_PACKAGE_WRONG_PLATFORM_DEVICE_VALUE

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

```
We found the platform device value was wrong in the Info.plist file. Please
unzip your test package and then open the Info.plist file inside the .app
directory, verify that the value of the key "CFBundleSupportedPlatforms"
does not contain the keyword "simulator", and try again.
```

Dalam contoh berikut, nama paket adalah Swift-sample-ui.ipa.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   Anda harus menemukan *Info.plist* file di dalam *.app* direktori seperti *swift-sampleUITests-Runner.app* dalam contoh kita:

   ```
   .
   `-- Payload (directory)
           `-- swift-sampleUITests-Runner.app (directory)
                       |-- Info.plist
                       |-- Plugins (directory)
                       |       `swift-sampleUITests.xctest (directory)
                       |                       |-- Info.plist
                       |                       `-- (any other files)
                       `-- (any other files)
   ```

3. Untuk menemukan nilai platform, Anda dapat membuka Info.plist menggunakan Xcode atau Python.

   Untuk Python, Anda dapat menginstal modul biplist dengan menjalankan perintah berikut:

   ```
   $ pip install biplist
   ```

4. Selanjutnya, buka Python dan jalankan perintah berikut:

   ```
   import biplist
   info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
   ```

```
print info_plist['CFBundleSupportedPlatforms']
```

Paket XCtest UI yang valid harus menghasilkan output seperti berikut:

```
['iPhoneOS']
```

Jika paket XCTest UI valid, nilainya tidak boleh berisi kata kunci`simulator`.

Untuk informasi selengkapnya, lihat Mengintegrasikan XCTest UI untuk iOS dengan Device Farm.

XCTEST_UI_TEST_PACKAGE_FORM_FACTOR_VALUE_MISSING

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

```
We could not the form factor value in the Info.plist. Please unzip your test package and then open the Info.plist file inside the .app directory, verify that the key "UIDeviceFamily" is specified, and try again.
```

Dalam contoh berikut, nama paket adalah Swift-sample-ui.ipa.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   Anda harus menemukan *`Info.plist`* file di dalam *`.app`* direktori seperti *`swift-sampleUITests-Runner.app`* dalam contoh kita:

   ```
   .
   `-- Payload (directory)
           `-- swift-sampleUITests-Runner.app (directory)
                       |-- Info.plist
                       |-- Plugins (directory)
                       |       `swift-sampleUITests.xctest (directory)
                       |                       |-- Info.plist
   ```

```
                        |                         `-- (any other files)
                        `-- (any other files)
```

3. Untuk menemukan nilai faktor bentuk, Anda dapat membuka Info.plist menggunakan Xcode atau Python.

   Untuk Python, Anda dapat menginstal modul biplist dengan menjalankan perintah berikut:

   ```
   $ pip install biplist
   ```

4. Selanjutnya, buka Python dan jalankan perintah berikut:

   ```
   import biplist
   info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
   print info_plist['UIDeviceFamily']
   ```

   Paket XCtest UI yang valid harus menghasilkan output seperti berikut:

   ```
   [1, 2]
   ```

   Untuk informasi selengkapnya, lihat Mengintegrasikan XCTest UI untuk iOS dengan Device Farm.

XCTEST_UI_TEST_PACKAGE_PACKAGE_NAME_VALUE_MISSING

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

```
We could not find the package name value in the Info.plist file. Please unzip your test package and then open the Info.plist file inside the .app directory, verify that the key "CFBundleIdentifier" is specified, and try again.
```

Dalam contoh berikut, nama paket adalah Swift-sample-ui.ipa.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

```
$ tree .
```

Anda harus menemukan *Info.plist* file di dalam *.app* direktori seperti *swift-sampleUITests-Runner.app* dalam contoh kita:

```
.
`-- Payload (directory)
        `-- swift-sampleUITests-Runner.app (directory)
                        |-- Info.plist
                        |-- Plugins (directory)
                        |       `swift-sampleUITests.xctest (directory)
                        |                         |-- Info.plist
                        |                         `-- (any other files)
                        `-- (any other files)
```

3. Untuk menemukan nilai nama paket, Anda dapat membuka Info.plist menggunakan Xcode atau Python.

   Untuk Python, Anda dapat menginstal modul biplist dengan menjalankan perintah berikut:

   ```
   $ pip install biplist
   ```

4. Selanjutnya, buka Python dan jalankan perintah berikut:

   ```
   import biplist
   info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
   print info_plist['CFBundleIdentifier']
   ```

   Paket XCtest UI yang valid harus menghasilkan output seperti berikut:

   ```
   com.apple.test.swift-sampleUITests-Runner
   ```

   Untuk informasi selengkapnya, lihat Mengintegrasikan XCTest UI untuk iOS dengan Device Farm.

XCTEST_UI_TEST_PACKAGE_EXECUTABLE_VALUE_MISSING

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

```
We could not find the executable value in the Info.plist file. Please
unzip your test package and then open the Info.plist file inside the .app
directory, verify that the key "CFBundleExecutable" is specified, and try
again.
```

Dalam contoh berikut, nama paket adalah Swift-sample-ui.ipa.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   Anda harus menemukan *Info.plist* file di dalam *.app* direktori seperti *swift-sampleUITests-Runner.app* dalam contoh kita:

   ```
   .
   `-- Payload (directory)
           `-- swift-sampleUITests-Runner.app (directory)
                       |-- Info.plist
                       |-- Plugins (directory)
                       |       `swift-sampleUITests.xctest (directory)
                       |                       |-- Info.plist
                       |                       `-- (any other files)
                       `-- (any other files)
   ```

3. Untuk menemukan nilai yang dapat dieksekusi, Anda dapat membuka Info.plist menggunakan Xcode atau Python.

   Untuk Python, Anda dapat menginstal modul biplist dengan menjalankan perintah berikut:

   ```
   $ pip install biplist
   ```

4. Selanjutnya, buka Python dan jalankan perintah berikut:

   ```
   import biplist
   info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
   ```

```
print info_plist['CFBundleExecutable']
```

Paket XCtest UI yang valid harus menghasilkan output seperti berikut:

```
XCTRunner
```

Untuk informasi selengkapnya, lihat Mengintegrasikan XCTest UI untuk iOS dengan Device Farm.

XCTEST_UI_TEST_PACKAGE_TEST_PACKAGE_NAME_VALUE_MISSING

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

```
We could not find the package name value in the Info.plist file inside
the .xctest directory. Please unzip your test package and then open
the Info.plist file inside the .xctest directory, verify that the key
"CFBundleIdentifier" is specified, and try again.
```

Dalam contoh berikut, nama paket adalah Swift-sample-ui.ipa.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   Anda harus menemukan *Info.plist* file di dalam *.app* direktori seperti *swift-sampleUITests-Runner.app* dalam contoh kita:

   ```
   .
   `-- Payload (directory)
           `-- swift-sampleUITests-Runner.app (directory)
                       |-- Info.plist
                       |-- Plugins (directory)
                       |       `swift-sampleUITests.xctest (directory)
                       |                       |-- Info.plist
   ```

```
                        |                         `-- (any other files)
                        `-- (any other files)
```

3. Untuk menemukan nilai nama paket, Anda dapat membuka Info.plist menggunakan Xcode atau Python.

   Untuk Python, Anda dapat menginstal modul biplist dengan menjalankan perintah berikut:

   ```
   $ pip install biplist
   ```

4. Selanjutnya, buka Python dan jalankan perintah berikut:

   ```
   import biplist
   info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Plugins/
   swift-sampleUITests.xctest/Info.plist')
   print info_plist['CFBundleIdentifier']
   ```

   Paket XCtest UI yang valid harus menghasilkan output seperti berikut:

   ```
   com.amazon.swift-sampleUITests
   ```

   Untuk informasi selengkapnya, lihat Mengintegrasikan XCTest UI untuk iOS dengan Device Farm.

XCTEST_UI_TEST_PACKAGE_TEST_EXECUTABLE_VALUE_MISSING

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

```
We could not find the executable value in the Info.plist file inside
the .xctest directory. Please unzip your test package and then open
the Info.plist file inside the .xctest directory, verify that the key
"CFBundleExecutable" is specified, and try again.
```

Dalam contoh berikut, nama paket adalah Swift-sample-ui.ipa.

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

```
$ tree .
```

Anda harus menemukan *Info.plist* file di dalam *.app* direktori seperti *swift-sampleUITests-Runner.app* dalam contoh kita:

```
.
`-- Payload (directory)
        `-- swift-sampleUITests-Runner.app (directory)
                        |-- Info.plist
                        |-- Plugins (directory)
                        |       `swift-sampleUITests.xctest (directory)
                        |                         |-- Info.plist
                        |                         `-- (any other files)
                        `-- (any other files)
```

3. Untuk menemukan nilai yang dapat dieksekusi, Anda dapat membuka Info.plist menggunakan Xcode atau Python.

Untuk Python, Anda dapat menginstal modul biplist dengan menjalankan perintah berikut:

```
$ pip install biplist
```

4. Selanjutnya, buka Python dan jalankan perintah berikut:

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Plugins/
swift-sampleUITests.xctest/Info.plist')
print info_plist['CFBundleExecutable']
```

Paket XCtest UI yang valid harus menghasilkan output seperti berikut:

```
swift-sampleUITests
```

Untuk informasi selengkapnya, lihat Mengintegrasikan XCTest UI untuk iOS dengan Device Farm.

XCTEST_UI_TEST_PACKAGE_MULTIPLE_APP_DIRS

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

`We found multiple .app directories inside your test package. Please unzip your test package, verify that only a single .app directory is present inside the package, then try again.`

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

   ```
   $ unzip swift-sample-UI.zip
   ```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

   ```
   $ tree .
   ```

   Jika paket XCTest UI valid, Anda harus menemukan hanya satu `.app` direktori seperti `swift-sampleUITests-Runner.app` dalam contoh kita di dalam paket tes.zip.

   ```
   .
   `--swift-sample-UI.zip--(directory)
       `-- swift-sampleUITests-Runner.app (directory)
               |-- Info.plist
               |-- Plugins (directory)
               |       `swift-sampleUITests.xctest (directory)
              |            |-- Info.plist
              |            `-- (any other files)
               `-- (any other files)
       `-- (any other files)
   ```

   Untuk informasi selengkapnya, lihat Mengintegrasikan XCTest UI untuk iOS dengan Device Farm.

XCTEST_UI_TEST_PACKAGE_MULTIPLE_IPA_DIRS

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

`We found multiple .ipa directories inside your test package. Please unzip your test package, verify that only a single .ipa directory is present inside the package, then try again.`

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

```
$ unzip swift-sample-UI.zip
```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

```
$ tree .
```

   Jika paket XCTest UI valid, Anda harus menemukan hanya satu `.ipa` direktori seperti `sampleUITests.ipa` dalam contoh kita di dalam paket tes.zip.

```
.
`--swift-sample-UI.zip--(directory)
    `-- sampleUITests.ipa (directory)
            `-- Payload (directory)
                `-- swift-sampleUITests-Runner.app (directory)
    `-- (any other files)
```

   Untuk informasi selengkapnya, lihat Mengintegrasikan XCTest UI untuk iOS dengan Device Farm.

XCTEST_UI_TEST_PACKAGE_BOTH_APP_AND_IPA_DIR_PRESENT

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

`We found both .app and .ipa files inside your test package. Please unzip your test package, verify that only a single .app or .ipa file is present inside the package, then try again.`

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

```
$ unzip swift-sample-UI.zip
```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

```
$ tree .
```

Jika paket XCTest UI valid, Anda harus menemukan `.ipa` direktori seperti `sampleUITests.ipa` atau `.app` direktori seperti `swift-sampleUITests-Runner.app` dalam contoh kita di dalam paket tes.zip. Anda dapat merujuk ke contoh paket Uji XCTEST_UI yang valid dalam dokumentasi kami di. Mengintegrasikan XCTest UI untuk iOS dengan Device Farm

```
.
`--swift-sample-UI.zip--(directory)
    `-- sampleUITests.ipa (directory)
            `-- Payload (directory)
                `-- swift-sampleUITests-Runner.app (directory)
   `-- (any other files)
```

atau

```
.
`--swift-sample-UI.zip--(directory)
    `-- swift-sampleUITests-Runner.app (directory)
            |-- Info.plist
            |-- Plugins (directory)
            `-- (any other files)
   `-- (any other files)
```

Untuk informasi selengkapnya, lihat Mengintegrasikan XCTest UI untuk iOS dengan Device Farm.

XCTEST_UI_TEST_PACKAGE_PAYLOAD_DIR_PRESENT_IN_ZIP

Jika Anda melihat pesan berikut, ikuti langkah-langkah berikut untuk memperbaiki masalah.

```
We found a Payload directory inside your .zip test package. Please unzip your test package, ensure that a Payload directory is not present in the package, then try again.
```

1. Salin paket pengujian Anda ke direktori kerja Anda, lalu jalankan perintah berikut:

```
$ unzip swift-sample-UI.zip
```

2. Setelah Anda berhasil unzip paket, Anda dapat menemukan struktur pohon direktori kerja dengan menjalankan perintah berikut:

```
$ tree .
```

Jika paket XCTest UI valid, Anda seharusnya tidak menemukan Direktori Payload di dalam paket pengujian Anda.

```
.
`--swift-sample-UI.zip--(directory)
    `-- swift-sampleUITests-Runner.app (directory)
            |-- Info.plist
            |-- Plugins (directory)
            `-- (any other files)
   `-- Payload (directory) [This directory should not be present]
            |-- (any other files)
   `-- (any other files)
```

Untuk informasi selengkapnya, lihat Mengintegrasikan XCTest UI untuk iOS dengan Device Farm.

Keamanan di AWS Device Farm

Keamanan cloud di AWS adalah prioritas tertinggi. Sebagai AWS pelanggan, Anda mendapat manfaat dari pusat data dan arsitektur jaringan yang dibangun untuk memenuhi persyaratan organisasi yang paling sensitif terhadap keamanan.

Keamanan adalah tanggung jawab bersama antara Anda AWS dan Anda. Model tanggung jawab bersama menjelaskan hal ini sebagai keamanan dari cloud dan keamanan dalam cloud:

- Keamanan cloud — AWS bertanggung jawab untuk melindungi infrastruktur yang menjalankan AWS layanan di AWS Cloud. AWS juga memberi Anda layanan yang dapat Anda gunakan dengan aman. Auditor pihak ketiga secara teratur menguji dan memverifikasi efektivitas keamanan kami sebagai bagian dari Program AWS Kepatuhan Program AWS Kepatuhan . Untuk mempelajari tentang program kepatuhan yang berlaku AWS Device Farm, lihat AWS Services in Scope by Compliance Program .
- Keamanan di cloud – Tanggung jawab Anda ditentukan menurut layanan AWS yang Anda gunakan. Anda juga bertanggung jawab atas faktor lain, yang mencakup sensitivitas data Anda, persyaratan perusahaan Anda, serta undang-undang dan peraturan yang berlaku.

Dokumentasi ini membantu Anda memahami cara menerapkan model tanggung jawab bersama saat menggunakan Device Farm. Topik berikut menunjukkan cara mengonfigurasi Device Farm untuk memenuhi tujuan keamanan dan kepatuhan Anda. Anda juga mempelajari cara menggunakan layanan AWS lain yang membantu Anda memantau dan mengamankan sumber daya Device Farm Anda.

Topik

- Praktik terbaik keamanan untuk Device Farm

Manajemen identitas dan akses di AWS Device Farm

Audiens

Cara Anda menggunakan AWS Identity and Access Management (IAM) berbeda berdasarkan peran Anda:

- Pengguna layanan - minta izin dari administrator Anda jika Anda tidak dapat mengakses fitur (lihat Memecahkan masalah identitas dan akses AWS Device Farm)
- Administrator layanan - tentukan akses pengguna dan mengirimkan permintaan izin (lihat Cara AWS Device Farm bekerja dengan IAM)
- Administrator IAM - tulis kebijakan untuk mengelola akses (lihat Contoh kebijakan berbasis identitas AWS Device Farm)

Mengautentikasi dengan identitas

Otentikasi adalah cara Anda masuk AWS menggunakan kredensi identitas Anda. Anda harus diautentikasi sebagai Pengguna root akun AWS, pengguna IAM, atau dengan mengambil peran IAM.

Anda dapat masuk sebagai identitas federasi menggunakan kredensional dari sumber identitas seperti AWS IAM Identity Center (Pusat Identitas IAM), autentikasi masuk tunggal, atau kredensional. Google/Facebook Untuk informasi selengkapnya tentang cara masuk, lihat Cara masuk ke Akun AWS Anda dalam Panduan Pengguna AWS Sign-In .

Untuk akses terprogram, AWS sediakan SDK dan CLI untuk menandatangani permintaan secara kriptografis. Untuk informasi selengkapnya, lihat AWS Signature Version 4 untuk permintaan API dalam Panduan Pengguna IAM.

Akun AWS pengguna root

Saat Anda membuat Akun AWS, Anda mulai dengan satu identitas masuk yang disebut pengguna Akun AWS root yang memiliki akses lengkap ke semua Layanan AWS dan sumber daya. Kami sangat menyarankan agar Anda tidak menggunakan pengguna root untuk tugas sehari-hari. Untuk tugas yang memerlukan kredensial pengguna root, lihat Tugas yang memerlukan kredensial pengguna root dalam Panduan Pengguna IAM.

Pengguna dan grup IAM

Pengguna IAM adalah identitas dengan izin khusus untuk satu orang atau aplikasi. Sebaiknya gunakan kredensial sementara alih-alih pengguna IAM dengan kredensial jangka panjang. Untuk informasi selengkapnya, lihat Mewajibkan pengguna manusia untuk menggunakan federasi dengan penyedia identitas untuk mengakses AWS menggunakan kredensi sementara di Panduan Pengguna IAM.

Grup IAM menentukan kumpulan pengguna IAM dan mempermudah pengelolaan izin untuk pengguna dalam jumlah besar. Untuk mempelajari selengkapnya, lihat Kasus penggunaan untuk pengguna IAM dalam Panduan Pengguna IAM.

Peran IAM

Peran IAM adalah identitas dengan izin khusus yang menyediakan kredensial sementara. Anda dapat mengambil peran dengan beralih dari pengguna ke peran IAM (konsol) atau dengan memanggil operasi AWS CLI atau AWS API. Untuk informasi selengkapnya, lihat Metode untuk mengambil peran dalam Panduan Pengguna IAM.

Peran IAM berguna untuk akses pengguna terfederasi, izin pengguna IAM sementara, akses lintas akun, akses lintas layanan, dan aplikasi yang berjalan di Amazon EC2. Untuk informasi selengkapnya, lihat Akses sumber daya lintas akun di IAM dalam Panduan Pengguna IAM.

Cara AWS Device Farm bekerja dengan IAM

Sebelum Anda menggunakan IAM untuk mengelola akses ke Device Farm, Anda harus memahami fitur IAM mana yang tersedia untuk digunakan dengan Device Farm. Untuk mendapatkan tampilan tingkat tinggi tentang cara kerja Device Farm dan AWS layanan lainnya dengan IAM, lihat AWS Layanan yang Bekerja dengan IAM di Panduan Pengguna IAM.

Topik

- Kebijakan berbasis identitas Device Farm
- Kebijakan berbasis sumber daya Device Farm
- Daftar kontrol akses
- Otorisasi berdasarkan tag Device Farm
- Peran IAM Device Farm

Kebijakan berbasis identitas Device Farm

Dengan kebijakan berbasis identitas IAM, Anda dapat menentukan tindakan dan sumber daya yang diizinkan atau ditolak dan ketentuan di mana tindakan tersebut diperbolehkan atau ditolak. Device Farm mendukung tindakan, sumber daya, dan kunci kondisi tertentu. Untuk mempelajari semua elemen yang Anda gunakan dalam kebijakan JSON, lihat Referensi Elemen Kebijakan JSON IAM dalam Panduan Pengguna IAM.

Tindakan

Administrator dapat menggunakan kebijakan AWS JSON untuk menentukan siapa yang memiliki akses ke apa. Yaitu, di mana utama dapat melakukan tindakan pada sumber daya, dan dalam kondisi apa.

Elemen `Action` dari kebijakan JSON menjelaskan tindakan yang dapat Anda gunakan untuk mengizinkan atau menolak akses dalam sebuah kebijakan. Sertakan tindakan dalam kebijakan untuk memberikan izin untuk melakukan operasi terkait.

Tindakan kebijakan di Device Farm menggunakan awalan berikut sebelum tindakan:`devicefarm:`. Misalnya, untuk memberikan izin kepada seseorang untuk memulai sesi Selenium dengan operasi `CreateTestGridUrl` API pengujian browser desktop Device Farm, Anda menyertakan `devicefarm:CreateTestGridUrl` tindakan tersebut dalam kebijakan. Pernyataan kebijakan harus memuat elemen `Action` atau `NotAction`. Device Farm mendefinisikan serangkaian tindakannya sendiri yang menjelaskan tugas yang dapat Anda lakukan dengan layanan ini.

Untuk menetapkan beberapa tindakan dalam satu pernyataan, pisahkan dengan koma seperti berikut:

```
"Action": [
      "devicefarm:action1",
      "devicefarm:action2"
```

Anda dapat menentukan beberapa tindakan menggunakan wildcard (*). Sebagai contoh, untuk menentukan semua tindakan yang dimulai dengan kata `List`, sertakan tindakan berikut:

```
"Action": "devicefarm:List*"
```

Untuk melihat daftar tindakan Device Farm, lihat Tindakan yang ditentukan oleh AWS Device Farm dalam Referensi Otorisasi Layanan IAM.

Sumber daya

Administrator dapat menggunakan kebijakan AWS JSON untuk menentukan siapa yang memiliki akses ke apa. Yaitu, di mana utama dapat melakukan tindakan pada sumber daya, dan dalam kondisi apa.

Elemen kebijakan JSON `Resource` menentukan objek yang menjadi target penerapan tindakan. Praktik terbaiknya, tentukan sumber daya menggunakan Amazon Resource Name (ARN). Untuk tindakan yang tidak mendukung izin di tingkat sumber daya, gunakan wildcard (*) untuk menunjukkan bahwa pernyataan tersebut berlaku untuk semua sumber daya.

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

Sumber daya instans Amazon EC2 memiliki ARN berikut:

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

Untuk informasi selengkapnya tentang format ARNs, lihat Amazon Resource Names (ARNs) dan Ruang Nama AWS Layanan.

Misalnya, untuk menentukan instans `i-1234567890abcdef0` dalam pernyataan Anda, gunakan ARN berikut:

```
"Resource": "arn:aws:ec2:us-east-1:123456789012:instance/i-1234567890abcdef0"
```

Untuk menentukan semua instance milik akun, gunakan wildcard (*):

```
"Resource": "arn:aws:ec2:us-east-1:123456789012:instance/*"
```

Beberapa tindakan Device Farm, seperti untuk membuat sumber daya, tidak dapat dilakukan pada sumber daya. Dalam kasus tersebut, Anda harus menggunakan wildcard (*).

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

Banyak tindakan API Amazon EC2 yang melibatkan beberapa sumber daya. Sebagai contoh, `AttachVolume` melampirkan volume Amazon EBS pada instans, sehingga pengguna IAM harus memiliki izin untuk menggunakan volume dan instans tersebut. Untuk menentukan beberapa sumber daya dalam satu pernyataan, pisahkan ARNs dengan koma.

```
"Resource": [
      "resource1",
      "resource2"
```

Untuk melihat daftar jenis sumber daya Device Farm dan jenisnya ARNs, lihat Jenis sumber daya yang ditentukan oleh AWS Device Farm dalam Referensi Otorisasi Layanan IAM. Untuk mempelajari tindakan mana yang dapat Anda tentukan ARN dari setiap sumber daya, lihat Tindakan yang ditentukan oleh AWS Device Farm dalam Referensi Otorisasi Layanan IAM.

Kunci syarat

Administrator dapat menggunakan kebijakan AWS JSON untuk menentukan siapa yang memiliki akses ke apa. Yaitu, principal dapat melakukan tindakan pada suatu sumber daya, dan dalam suatu syarat.

Elemen `Condition` menentukan ketika pernyataan dieksekusi berdasarkan kriteria yang ditetapkan. Anda dapat membuat ekspresi bersyarat yang menggunakan operator kondisi, misalnya sama dengan atau kurang dari, untuk mencocokkan kondisi dalam kebijakan dengan nilai-nilai yang diminta. Untuk melihat semua kunci kondisi AWS global, lihat kunci konteks kondisi AWS global di Panduan Pengguna IAM.

Device Farm mendefinisikan rangkaian kunci kondisinya sendiri dan juga mendukung penggunaan beberapa kunci kondisi global. Untuk melihat semua kunci kondisi AWS global, lihat Kunci Konteks Kondisi AWS Global di Panduan Pengguna IAM.

Untuk melihat daftar kunci kondisi Device Farm, lihat Kunci kondisi untuk AWS Device Farm Referensi Otorisasi Layanan IAM. Untuk mempelajari tindakan dan sumber daya yang dapat Anda gunakan kunci kondisi, lihat Tindakan yang ditentukan oleh AWS Device Farm dalam Referensi Otorisasi Layanan IAM.

Contoh

Untuk melihat contoh kebijakan berbasis identitas Device Farm, lihat. Contoh kebijakan berbasis identitas AWS Device Farm

Kebijakan berbasis sumber daya Device Farm

Device Farm tidak mendukung kebijakan berbasis sumber daya.

Daftar kontrol akses

Device Farm tidak mendukung daftar kontrol akses (ACLs).

Otorisasi berdasarkan tag Device Farm

Anda dapat melampirkan tag ke sumber daya Device Farm atau meneruskan tag dalam permintaan ke Device Farm. Untuk mengendalikan akses berdasarkan tanda, berikan informasi tentang tanda di elemen kondisi dari kebijakan menggunakan kunci kondisi `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, atau `aws:TagKeys`. Untuk informasi selengkapnya tentang menandai sumber daya Device Farm, lihatMenandai di Device Farm.

Untuk melihat contoh kebijakan berbasis identitas untuk membatasi akses ke sumber daya berdasarkan tag pada sumber daya tersebut, lihat Melihat proyek pengujian browser desktop Device Farm berdasarkan tag.

Peran IAM Device Farm

Peran IAM adalah entitas di AWS akun Anda yang memiliki izin tertentu.

Menggunakan kredensil sementara dengan Device Farm

Device Farm mendukung penggunaan kredensil sementara.

Anda dapat menggunakan kredensi sementara untuk masuk dengan federasi untuk mengambil peran IAM atau peran lintas akun. Anda memperoleh kredensi keamanan sementara dengan memanggil operasi AWS STS API seperti AssumeRoleatau. GetFederationToken

Peran terkait layanan

Peran terkait AWS layanan memungkinkan layanan mengakses sumber daya di layanan lain untuk menyelesaikan tindakan atas nama Anda. Peran tertaut layanan muncul di akun IAM Anda dan dimiliki oleh layanan tersebut. Administrator IAM dapat melihat, tetapi tidak dapat mengedit, izin untuk peran terkait layanan.

Device Farm menggunakan peran terkait layanan dalam fitur pengujian browser desktop Device Farm. Untuk informasi tentang peran ini, lihat Menggunakan Peran Tertaut Layanan dalam pengujian browser desktop Device Farm dalam panduan pengembang.

Peran layanan

Device Farm tidak mendukung peran layanan.

Fitur ini memungkinkan layanan untuk menerima peran layanan atas nama Anda. Peran ini mengizinkan layanan untuk mengakses sumber daya di layanan lain untuk menyelesaikan tindakan atas nama Anda. Peran layanan muncul di akun IAM Anda dan dimiliki oleh akun tersebut. Ini berarti administrator IAM dapat mengubah izin untuk peran ini. Namun, melakukan hal itu dapat merusak fungsionalitas layanan.

Mengelola akses menggunakan kebijakan

Anda mengontrol akses AWS dengan membuat kebijakan dan melampirkannya ke AWS identitas atau sumber daya. Kebijakan menentukan izin saat dikaitkan dengan identitas atau sumber daya. AWS mengevaluasi kebijakan ini ketika kepala sekolah membuat permintaan. Sebagian besar kebijakan disimpan AWS sebagai dokumen JSON. Untuk informasi selengkapnya tentang dokumen kebijakan JSON, lihat Gambaran umum kebijakan JSON dalam Panduan Pengguna IAM.

Menggunakan kebijakan, administrator menentukan siapa yang memiliki akses ke apa dengan mendefinisikan principal mana yang dapat melakukan tindakan pada sumber daya apa, dan dalam kondisi apa.

Secara default, pengguna dan peran tidak memiliki izin. Administrator IAM membuat kebijakan IAM dan menambahkannya ke peran, yang kemudian dapat diambil oleh pengguna. Kebijakan IAM mendefinisikan izin terlepas dari metode yang Anda gunakan untuk melakukan operasinya.

Kebijakan berbasis identitas

Kebijakan berbasis identitas adalah dokumen kebijakan izin JSON yang Anda lampirkan ke identitas (pengguna, grup, atau peran). Kebijakan ini mengontrol tindakan apa yang bisa dilakukan oleh identitas tersebut, terhadap sumber daya yang mana, dan dalam kondisi apa. Untuk mempelajari cara membuat kebijakan berbasis identitas, lihat Tentukan izin IAM kustom dengan kebijakan yang dikelola pelanggan dalam Panduan Pengguna IAM.

Kebijakan berbasis identitas dapat berupa kebijakan inline (disematkan langsung ke dalam satu identitas) atau kebijakan terkelola (kebijakan mandiri yang dilampirkan pada banyak identitas). Untuk mempelajari cara memilih antara kebijakan terkelola dan kebijakan inline, lihat Pilih antara kebijakan terkelola dan kebijakan inline dalam Panduan Pengguna IAM.

Tabel berikut menguraikan kebijakan terkelola Device Farm AWS.

Ubah	Deskripsi	Date
AWSDeviceFarmFullAccess	Menyediakan akses penuh ke semua operasi AWS Device Farm.	15 Juli 2015
AWSServiceRoleForDeviceFarmTestGrid	Memungkinkan Device Farm mengakses sumber daya AWS atas nama Anda.	20 Mei 2021

Jenis-jenis kebijakan lain

AWS mendukung jenis kebijakan tambahan yang dapat menetapkan izin maksimum yang diberikan oleh jenis kebijakan yang lebih umum:

- Batasan izin – Menetapkan izin maksimum yang dapat diberikan oleh kebijakan berbasis identitas kepada entitas IAM. Untuk informasi selengkapnya, lihat Batasan izin untuk entitas IAM dalam Panduan Pengguna IAM.
- Kebijakan kontrol layanan (SCPs) — Tentukan izin maksimum untuk organisasi atau unit organisasi di AWS Organizations. Untuk informasi selengkapnya, lihat Kebijakan kontrol layanan dalam Panduan Pengguna AWS Organizations .
- Kebijakan kontrol sumber daya (RCPs) — Tetapkan izin maksimum yang tersedia untuk sumber daya di akun Anda. Untuk informasi selengkapnya, lihat Kebijakan kontrol sumber daya (RCPs) di Panduan AWS Organizations Pengguna.
- Kebijakan sesi – Kebijakan lanjutan yang diteruskan sebagai parameter saat membuat sesi sementara untuk peran atau pengguna terfederasi. Untuk informasi selengkapnya, lihat Kebijakan sesi dalam Panduan Pengguna IAM.

Berbagai jenis kebijakan

Ketika beberapa jenis kebijakan berlaku pada suatu permintaan, izin yang dihasilkan lebih rumit untuk dipahami. Untuk mempelajari cara AWS menentukan apakah akan mengizinkan permintaan saat beberapa jenis kebijakan terlibat, lihat Logika evaluasi kebijakan di Panduan Pengguna IAM.

Contoh kebijakan berbasis identitas AWS Device Farm

Secara default, pengguna dan peran IAM tidak memiliki izin untuk membuat atau memodifikasi sumber daya Device Farm. Mereka juga tidak dapat melakukan tugas menggunakan Konsol Manajemen AWS, AWS CLI, atau AWS API. Administrator IAM harus membuat kebijakan IAM yang memberikan izin kepada pengguna dan peran untuk melakukan operasi API tertentu pada sumber daya yang diperlukan. Administrator kemudian harus melampirkan kebijakan tersebut ke pengguna IAM atau grup yang memerlukan izin tersebut.

Untuk mempelajari cara membuat kebijakan berbasis identitas IAM menggunakan contoh dokumen kebijakan JSON ini, lihat Membuat Kebijakan pada Tab JSON dalam Panduan Pengguna IAM.

Topik

Praktik terbaik kebijakan

Kebijakan berbasis identitas menentukan apakah seseorang dapat membuat, mengakses, atau menghapus sumber daya Device Farm di akun Anda. Tindakan ini membuat Akun AWS Anda dikenai biaya. Ketika Anda membuat atau mengedit kebijakan berbasis identitas, ikuti panduan dan rekomendasi ini:

- Mulailah dengan kebijakan AWS terkelola dan beralih ke izin hak istimewa paling sedikit — Untuk mulai memberikan izin kepada pengguna dan beban kerja Anda, gunakan kebijakan AWS terkelola yang memberikan izin untuk banyak kasus penggunaan umum. Mereka tersedia di Anda Akun AWS. Kami menyarankan Anda mengurangi izin lebih lanjut dengan menentukan kebijakan yang dikelola AWS pelanggan yang khusus untuk kasus penggunaan Anda. Untuk informasi selengkapnya, lihat Kebijakan yang dikelola AWS atau Kebijakan yang dikelola AWS untuk fungsi tugas dalam Panduan Pengguna IAM.
- Menerapkan izin dengan hak akses paling rendah – Ketika Anda menetapkan izin dengan kebijakan IAM, hanya berikan izin yang diperlukan untuk melakukan tugas. Anda melakukannya dengan mendefinisikan tindakan yang dapat diambil pada sumber daya tertentu dalam kondisi tertentu, yang juga dikenal sebagai izin dengan hak akses paling rendah. Untuk informasi

selengkapnya tentang cara menggunakan IAM untuk mengajukan izin, lihat Kebijakan dan izin dalam IAM dalam Panduan Pengguna IAM.

- Gunakan kondisi dalam kebijakan IAM untuk membatasi akses lebih lanjut – Anda dapat menambahkan suatu kondisi ke kebijakan Anda untuk membatasi akses ke tindakan dan sumber daya. Sebagai contoh, Anda dapat menulis kondisi kebijakan untuk menentukan bahwa semua permintaan harus dikirim menggunakan SSL. Anda juga dapat menggunakan ketentuan untuk memberikan akses ke tindakan layanan jika digunakan melalui yang spesifik Layanan AWS, seperti CloudFormation. Untuk informasi selengkapnya, lihat Elemen kebijakan JSON IAM: Kondisi dalam Panduan Pengguna IAM.
- Gunakan IAM Access Analyzer untuk memvalidasi kebijakan IAM Anda untuk memastikan izin yang aman dan fungsional – IAM Access Analyzer memvalidasi kebijakan baru dan yang sudah ada sehingga kebijakan tersebut mematuhi bahasa kebijakan IAM (JSON) dan praktik terbaik IAM. IAM Access Analyzer menyediakan lebih dari 100 pemeriksaan kebijakan dan rekomendasi yang dapat ditindaklanjuti untuk membantu Anda membuat kebijakan yang aman dan fungsional. Untuk informasi selengkapnya, lihat Validasi kebijakan dengan IAM Access Analyzer dalam Panduan Pengguna IAM.
- Memerlukan otentikasi multi-faktor (MFA) - Jika Anda memiliki skenario yang mengharuskan pengguna IAM atau pengguna root di Anda, Akun AWS aktifkan MFA untuk keamanan tambahan. Untuk meminta MFA ketika operasi API dipanggil, tambahkan kondisi MFA pada kebijakan Anda. Untuk informasi selengkapnya, lihat Amankan akses API dengan MFA dalam Panduan Pengguna IAM.

Untuk informasi selengkapnya tentang praktik terbaik dalam IAM, lihat Praktik terbaik keamanan di IAM dalam Panduan Pengguna IAM.

Mengizinkan pengguna melihat izin mereka sendiri

Contoh ini menunjukkan cara membuat kebijakan yang mengizinkan pengguna IAM melihat kebijakan inline dan terkelola yang dilampirkan ke identitas pengguna mereka. Kebijakan ini mencakup izin untuk menyelesaikan tindakan ini di konsol atau menggunakan API atau secara terprogram. AWS CLI AWS

```
{
    "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": "*"
        }
    ]
}
```

Mengakses satu proyek pengujian browser desktop Device Farm

Dalam contoh ini, Anda ingin memberikan pengguna IAM di AWS akun Anda akses ke salah satu proyek pengujian browser Device Farm Anda,. `arn:aws:devicefarm:us-west-2:111122223333:testgrid-project:123e4567-e89b-12d3-a456-426655441111` Anda ingin akun dapat melihat item yang terkait dengan proyek.

Selain `devicefarm:GetTestGridProject` titik akhir, akun harus memiliki,`devicefarm:ListTestGridSessions`, `devicefarm:GetTestGridSessiondevicefarm:ListTestGridSessionActions`, dan titik `devicefarm:ListTestGridSessionArtifacts` akhir.

Jika Anda menggunakan sistem CI, Anda harus memberikan setiap kredensi akses unik pelari CI. Misalnya, sistem CI tidak mungkin membutuhkan lebih banyak izin daripada

`devicefarm:ScheduleRun` atau`devicefarm:CreateUpload`. Kebijakan IAM berikut menguraikan kebijakan minimal untuk memungkinkan pelari CI memulai pengujian pengujian aplikasi bawaan Device Farm baru dengan membuat unggahan dan menggunakannya untuk menjadwalkan uji coba:

Melihat proyek pengujian browser desktop Device Farm berdasarkan tag

Anda dapat menggunakan kondisi dalam kebijakan berbasis identitas untuk mengontrol akses ke sumber daya Device Farm berdasarkan tag. Contoh ini menunjukkan cara membuat kebijakan yang memungkinkan penayangan proyek dan sesi. Izin diberikan jika `0wner` tag sumber daya yang diminta cocok dengan nama pengguna akun yang meminta.

Anda dapat melampirkan kebijakan ini ke pengguna IAM di akun Anda. Jika pengguna bernama `richard-roe` mencoba melihat proyek atau sesi Device Farm, proyek harus diberi tag `0wner=richard-roe` atau`owner=richard-roe`. Jika tidak, pengguna ditolak aksesnya. Kunci tag kondisi `0wner` cocok dengan keduanya `0wner` dan `owner` karena nama kunci kondisi tidak peka huruf besar/kecil. Untuk informasi selengkapnya, lihat Elemen kebijakan IAM JSON: Syarat dalam Panduan Pengguna IAM.

Memecahkan masalah identitas dan akses AWS Device Farm

Gunakan informasi berikut untuk membantu Anda mendiagnosis dan memperbaiki masalah umum yang mungkin Anda temui saat bekerja dengan Device Farm dan IAM.

Saya tidak berwenang untuk melakukan tindakan di Device Farm

Jika Anda menerima kesalahan dalam Konsol Manajemen AWS yang mengatakan Anda tidak berwenang untuk melakukan tindakan, Anda harus menghubungi administrator Anda untuk bantuan. Administrator Anda adalah orang yang memberikan nama pengguna dan kata sandi Anda.

Contoh kesalahan berikut terjadi ketika pengguna IAM,`mateojackson`, mencoba menggunakan konsol untuk melihat detail tentang proses, tetapi tidak memiliki `devicefarm:GetRun` izin.

```
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
```

Dalam hal ini, Mateo meminta administratornya untuk memperbarui kebijakannya untuk memungkinkannya mengakses `arn:aws:devicefarm:us-`

west-2:123456789101:run:123e4567-e89b-12d3-a456-426655440000/123e4567-e89b-12d3-a456-426655441111 sumber daya devicefarm:GetRun pada menggunakan devicefarm:GetRun tindakan.

Saya tidak berwenang untuk melakukan iam: PassRole

Jika Anda menerima kesalahan yang tidak diizinkan untuk melakukan iam:PassRole tindakan, kebijakan Anda harus diperbarui agar Anda dapat meneruskan peran ke Device Farm.

Beberapa Layanan AWS memungkinkan Anda untuk meneruskan peran yang ada ke layanan tersebut alih-alih membuat peran layanan baru atau peran terkait layanan. Untuk melakukannya, Anda harus memiliki izin untuk meneruskan peran ke layanan.

Contoh kesalahan berikut terjadi ketika pengguna IAM bernama marymajor mencoba menggunakan konsol untuk melakukan tindakan di Device Farm. Namun, tindakan tersebut memerlukan layanan untuk mendapatkan izin yang diberikan oleh peran layanan. Mary tidak memiliki izin untuk meneruskan peran tersebut pada layanan.

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

Dalam kasus ini, kebijakan Mary harus diperbarui agar dia mendapatkan izin untuk melakukan tindakan iam:PassRole tersebut.

Jika Anda memerlukan bantuan, hubungi AWS administrator Anda. Administrator Anda adalah orang yang memberi Anda kredensial masuk.

Saya ingin melihat access key saya

Setelah membuat access key pengguna IAM, Anda dapat melihat access key ID Anda setiap saat. Namun, Anda tidak dapat melihat secret access key Anda lagi. Jika Anda kehilangan secret key, Anda harus membuat pasangan access key baru.

Access key terdiri dari dua bagian: access key ID (misalnya, AKIAIOSFODNN7EXAMPLE) dan secret access key (misalnya, wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY). Seperti nama pengguna dan kata sandi, Anda harus menggunakan access key ID dan secret access key sekaligus untuk mengautentikasi permintaan Anda. Kelola access key Anda seaman nama pengguna dan kata sandi Anda.

> **Important**
>
> Jangan memberikan access key Anda kepada pihak ke tiga, bahkan untuk membantu menemukan ID pengguna kanonis Anda. Dengan melakukan ini, Anda mungkin memberi seseorang akses permanen ke Anda Akun AWS.

Saat Anda membuat pasangan access key, Anda diminta menyimpan access key ID dan secret access key di lokasi yang aman. secret access key hanya tersedia saat Anda membuatnya. Jika Anda kehilangan secret access key Anda, Anda harus menambahkan access key baru ke pengguna IAM Anda. Anda dapat memiliki maksimum dua access key. Jika Anda sudah memiliki dua, Anda harus menghapus satu pasangan kunci sebelum membuat pasangan baru. Untuk melihat instruksi, lihat Mengelola access keys di Panduan Pengguna IAM.

Saya seorang administrator dan ingin mengizinkan orang lain mengakses Device Farm

Untuk mengizinkan orang lain mengakses Device Farm, Anda harus memberikan izin kepada orang atau aplikasi yang memerlukan akses. Jika Anda menggunakan AWS IAM Identity Center untuk mengelola orang dan aplikasi, Anda menetapkan set izin kepada pengguna atau grup untuk menentukan tingkat akses mereka. Set izin secara otomatis membuat dan menetapkan kebijakan IAM ke peran IAM yang terkait dengan orang atau aplikasi. Untuk informasi selengkapnya, lihat Set izin di Panduan AWS IAM Identity Center Pengguna.

Jika Anda tidak menggunakan IAM Identity Center, Anda harus membuat entitas IAM (pengguna atau peran) untuk orang atau aplikasi yang membutuhkan akses. Anda kemudian harus melampirkan kebijakan ke entitas yang memberi mereka izin yang benar di Device Farm. Setelah izin diberikan, berikan kredensialnya kepada pengguna atau pengembang aplikasi. Mereka akan menggunakan kredensil tersebut untuk mengakses. AWSUntuk mempelajari selengkapnya tentang membuat pengguna, grup, kebijakan, dan izin IAM, lihat Identitas dan Kebijakan IAM dan izin di IAM di Panduan Pengguna IAM.

Saya ingin mengizinkan orang di luar AWS akun saya untuk mengakses sumber daya Device Farm saya

Anda dapat membuat peran yang dapat digunakan pengguna di akun lain atau orang-orang di luar organisasi Anda untuk mengakses sumber daya Anda. Anda dapat menentukan siapa saja yang dipercaya untuk mengambil peran tersebut. Untuk layanan yang mendukung kebijakan berbasis

sumber daya atau daftar kontrol akses (ACLs), Anda dapat menggunakan kebijakan tersebut untuk memberi orang akses ke sumber daya Anda.

Untuk mempelajari selengkapnya, periksa referensi berikut:

- Untuk mengetahui apakah Device Farm mendukung fitur ini, lihatCara AWS Device Farm bekerja dengan IAM.
- Untuk mempelajari cara menyediakan akses ke sumber daya Anda di seluruh sumber daya Akun AWS yang Anda miliki, lihat Menyediakan akses ke pengguna IAM di pengguna lain Akun AWS yang Anda miliki di Panduan Pengguna IAM.
- Untuk mempelajari cara menyediakan akses ke sumber daya Anda kepada pihak ketiga Akun AWS, lihat Menyediakan akses yang Akun AWS dimiliki oleh pihak ketiga dalam Panduan Pengguna IAM.
- Untuk mempelajari cara memberikan akses melalui federasi identitas, lihat Menyediakan akses ke pengguna terautentikasi eksternal (federasi identitas) dalam Panduan Pengguna IAM.
- Untuk mempelajari perbedaan antara menggunakan peran dan kebijakan berbasis sumber daya untuk akses lintas akun, lihat Akses sumber daya lintas akun di IAM di Panduan Pengguna IAM.

Validasi kepatuhan untuk AWS Device Farm

Auditor pihak ketiga menilai keamanan dan kepatuhan AWS Device Farm sebagai bagian dari beberapa program AWS kepatuhan. Ini termasuk SOC, PCI, FedRAMP, HIPAA, dan lainnya. AWS Device Farm tidak dalam lingkup program AWS kepatuhan apa pun.

Untuk daftar AWS layanan dalam lingkup program kepatuhan tertentu, lihat AWS Services in Scope by Compliance Program . Untuk informasi umum, lihat Program AWS Kepatuhan Program AWS .

Anda dapat mengunduh laporan audit pihak ketiga menggunakan AWS Artifact. Untuk informasi selengkapnya, lihat Pengunduhan Laporan dalam AWS Artifact.

Tanggung jawab kepatuhan Anda saat menggunakan Device Farm ditentukan oleh sensitivitas data Anda, tujuan kepatuhan perusahaan Anda, serta undang-undang dan peraturan yang berlaku. AWS menyediakan sumber daya berikut untuk membantu kepatuhan:

- Panduan Quick Start Keamanan dan Kepatuhan – Panduan deployment ini membahas pertimbangan arsitektur dan menyediakan langkah–langkah untuk melakukan deployment terhadap lingkungan dasar di AWS yang menjadi fokus keamanan dan kepatuhan.

- AWS Sumber Daya AWS — Kumpulan buku kerja dan panduan ini mungkin berlaku untuk industri dan lokasi Anda.
- Mengevaluasi Sumber Daya dengan Aturan dalam Panduan AWS Config Pengembang — AWS Config menilai seberapa baik konfigurasi sumber daya Anda mematuhi praktik internal, pedoman industri, dan peraturan.
- AWS Security Hub CSPM— AWS Layanan ini memberikan pandangan komprehensif tentang keadaan keamanan Anda di dalamnya AWS yang membantu Anda memeriksa kepatuhan Anda terhadap standar industri keamanan dan praktik terbaik.

Perlindungan data di AWS Device Farm

Model tanggung jawab AWS bersama model berlaku untuk perlindungan data di AWS Device Farm (Device Farm). Seperti yang dijelaskan dalam model AWS ini, bertanggung jawab untuk melindungi infrastruktur global yang menjalankan semua AWS Cloud. Anda bertanggung jawab untuk mempertahankan kendali atas konten yang di-host pada infrastruktur ini. Anda juga bertanggung jawab atas tugas-tugas konfigurasi dan manajemen keamanan untuk Layanan AWS yang Anda gunakan. Lihat informasi yang lebih lengkap tentang privasi data dalam Pertanyaan Umum Privasi Data. Lihat informasi tentang perlindungan data di Eropa di pos blog Model Tanggung Jawab Bersama dan GDPR AWS di Blog Keamanan AWS .

Untuk tujuan perlindungan data, kami menyarankan Anda melindungi Akun AWS kredensil dan mengatur pengguna individu dengan AWS IAM Identity Center atau AWS Identity and Access Management (IAM). Dengan cara itu, setiap pengguna hanya diberi izin yang diperlukan untuk memenuhi tanggung jawab tugasnya. Kami juga menyarankan supaya Anda mengamankan data dengan cara-cara berikut:

- Gunakan autentikasi multi-faktor (MFA) pada setiap akun.
- Gunakan SSL/TLS untuk berkomunikasi dengan AWS sumber daya. Kami mensyaratkan TLS 1.2 dan menganjurkan TLS 1.3.
- Siapkan API dan pencatatan aktivitas pengguna dengan AWS CloudTrail. Untuk informasi tentang penggunaan CloudTrail jejak untuk menangkap AWS aktivitas, lihat Bekerja dengan CloudTrail jejak di AWS CloudTrail Panduan Pengguna.
- Gunakan solusi AWS enkripsi, bersama dengan semua kontrol keamanan default di dalamnya Layanan AWS.
- Gunakan layanan keamanan terkelola tingkat lanjut seperti Amazon Macie, yang membantu menemukan dan mengamankan data sensitif yang disimpan di Amazon S3.

- Jika Anda memerlukan modul kriptografi tervalidasi FIPS 140-3 saat mengakses AWS melalui antarmuka baris perintah atau API, gunakan titik akhir FIPS. Lihat informasi selengkapnya tentang titik akhir FIPS yang tersedia di Standar Pemrosesan Informasi Federal (FIPS) 140-3.

Kami sangat merekomendasikan agar Anda tidak pernah memasukkan informasi identifikasi yang sensitif, seperti nomor rekening pelanggan Anda, ke dalam tanda atau bidang isian bebas seperti bidang Nama. Ini termasuk saat Anda bekerja dengan Device Farm atau lainnya Layanan AWS menggunakan konsol, API AWS CLI, atau AWS SDKs. Data apa pun yang Anda masukkan ke dalam tanda atau bidang isian bebas yang digunakan untuk nama dapat digunakan untuk log penagihan atau log diagnostik. Saat Anda memberikan URL ke server eksternal, kami sangat menganjurkan supaya Anda tidak menyertakan informasi kredensial di dalam URL untuk memvalidasi permintaan Anda ke server itu.

Enkripsi saat bergerak

Titik akhir Device Farm hanya mendukung HTTPS yang ditandatangani (SSL/TLS) requests except where otherwise noted. All content retrieved from or placed in Amazon S3 through upload URLs is encrypted using SSL/TLS. Untuk informasi selengkapnya tentang cara permintaan HTTPS masuk AWS, lihat Menandatangani permintaan AWS API di Referensi AWS Umum.

Merupakan tanggung jawab Anda untuk mengenkripsi dan mengamankan komunikasi apa pun yang dibuat oleh aplikasi Anda yang diuji dan aplikasi apa pun yang diinstal dalam proses menjalankan pengujian di perangkat.

Enkripsi saat diam

Fitur pengujian browser desktop Device Farm mendukung enkripsi saat istirahat untuk artefak yang dihasilkan selama pengujian.

Data pengujian perangkat seluler fisik Device Farm tidak dienkripsi saat istirahat.

Retensi data

Data di Device Farm disimpan untuk waktu yang terbatas. Setelah periode retensi berakhir, data akan dihapus dari penyimpanan dukungan Device Farm.

Jenis konten	Periode retensi (hari)	Periode Retensi Metadata (hari)
Aplikasi yang diunggah	30	30
Paket uji yang diunggah	30	30
Beberapa catatan	400	400
Rekaman video dan artefak lainnya	400	400

Anda bertanggung jawab untuk mengarsipkan konten apa pun yang ingin Anda simpan untuk waktu yang lebih lama.

Manajemen data

Data di Device Farm dikelola secara berbeda tergantung pada fitur mana yang digunakan. Bagian ini menjelaskan bagaimana data dikelola saat dan setelah Anda menggunakan Device Farm.

Pengujian browser desktop

Contoh yang digunakan selama sesi Selenium tidak disimpan. Semua data yang dihasilkan sebagai hasil dari interaksi browser dibuang ketika sesi berakhir.

Fitur ini saat ini mendukung enkripsi saat istirahat untuk artefak yang dihasilkan selama pengujian.

Pengujian perangkat fisik

Bagian berikut memberikan informasi tentang langkah-langkah yang AWS diperlukan untuk membersihkan atau menghancurkan perangkat setelah Anda menggunakan Device Farm.

Data pengujian perangkat seluler fisik Device Farm tidak dienkripsi saat istirahat.

Armada perangkat publik

Setelah eksekusi pengujian selesai, Device Farm melakukan serangkaian tugas pembersihan di setiap perangkat dalam armada perangkat publik, termasuk penghapusan instalasi aplikasi Anda.

Jika kami tidak dapat memverifikasi penghapusan instalasi aplikasi Anda atau salah satu langkah pembersihan lainnya, perangkat akan menerima reset pabrik sebelum digunakan kembali.

> **Note**
>
> Data dapat bertahan di antara sesi dalam beberapa kasus, terutama jika Anda menggunakan sistem perangkat di luar konteks aplikasi Anda. Untuk alasan ini, dan karena Device Farm menangkap video dan log aktivitas yang terjadi selama Anda menggunakan setiap perangkat, kami menyarankan Anda untuk tidak memasukkan informasi sensitif (misalnya, akun Google atau ID Apple), informasi pribadi, dan detail sensitif keamanan lainnya selama sesi pengujian otomatis dan akses jarak jauh Anda.

Perangkat pribadi

Setelah kedaluwarsa atau penghentian kontrak perangkat pribadi Anda, perangkat dihapus dari penggunaan dan dihancurkan dengan aman sesuai dengan kebijakan penghancuran AWS. Untuk informasi selengkapnya, lihat Perangkat pribadi di AWS Device Farm.

Manajemen kunci

Saat ini, Device Farm tidak menawarkan manajemen kunci eksternal untuk enkripsi data, saat istirahat atau dalam perjalanan.

Privasi lalu lintas antarjaringan

Device Farm dapat dikonfigurasi, hanya untuk perangkat pribadi, untuk menggunakan titik akhir Amazon VPC untuk terhubung ke sumber daya Anda. AWS Akses ke AWS infrastruktur non-publik apa pun yang terkait dengan akun Anda (misalnya, EC2 instans Amazon tanpa alamat IP publik) harus menggunakan titik akhir VPC Amazon. Terlepas dari konfigurasi titik akhir VPC, Device Farm mengisolasi lalu lintas Anda dari pengguna lain di seluruh jaringan Device Farm.

Koneksi Anda di luar AWS jaringan tidak dijamin aman atau aman, dan Anda bertanggung jawab untuk mengamankan koneksi internet apa pun yang dibuat aplikasi Anda.

Ketahanan di AWS Device Farm

Infrastruktur AWS global dibangun di sekitar AWS Wilayah dan Zona Ketersediaan. AWS Wilayah menyediakan beberapa Availability Zone yang terpisah secara fisik dan terisolasi, yang terhubung

dengan latensi rendah, throughput tinggi, dan jaringan yang sangat redundan. Dengan Zona Ketersediaan, Anda dapat merancang serta mengoperasikan aplikasi dan basis data yang secara otomatis melakukan fail over di antara zona tanpa gangguan. Zona Ketersediaan memiliki ketersediaan dan toleransi kesalahan yang lebih baik, dan dapat diskalakan dibandingkan infrastruktur pusat data tunggal atau multi tradisional.

Untuk informasi selengkapnya tentang AWS Wilayah dan Availability Zone, lihat Infrastruktur AWS Global.

Karena Device Farm hanya tersedia di `us-west-2` Wilayah, kami sangat menyarankan Anda menerapkan proses pencadangan dan pemulihan. Device Farm tidak boleh menjadi satu-satunya sumber konten yang diunggah.

Device Farm tidak menjamin ketersediaan perangkat publik. Perangkat ini dibawa masuk dan keluar dari kumpulan perangkat publik tergantung pada berbagai faktor, seperti tingkat kegagalan dan status karantina. Kami tidak menyarankan Anda bergantung pada ketersediaan satu perangkat di kolam perangkat publik.

Keamanan infrastruktur di AWS Device Farm

Sebagai layanan terkelola, AWS Device Farm dilindungi oleh keamanan jaringan AWS global. Untuk informasi tentang layanan AWS keamanan dan cara AWS melindungi infrastruktur, lihat Keamanan AWS Cloud. Untuk mendesain AWS lingkungan Anda menggunakan praktik terbaik untuk keamanan infrastruktur, lihat Perlindungan Infrastruktur dalam Kerangka Kerja yang AWS Diarsiteksikan dengan Baik Pilar Keamanan.

Anda menggunakan panggilan API yang AWS dipublikasikan untuk mengakses Device Farm melalui jaringan. Klien harus mendukung hal-hal berikut:

- Keamanan Lapisan Pengangkutan (TLS). Kami mensyaratkan TLS 1.2 dan menganjurkan TLS 1.3.
- Sandi cocok dengan sistem kerahasiaan maju sempurna (perfect forward secrecy, PFS) seperti DHE (Ephemeral Diffie-Hellman) atau ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Sebagian besar sistem modern seperti Java 7 dan versi lebih baru mendukung mode-mode ini.

Selain itu, permintaan harus ditandatangani menggunakan ID kunci akses dan kunci akses rahasia yang terkait dengan principal IAM. Atau Anda dapat menggunakan AWS Security Token Service (AWS STS) untuk menghasilkan kredensial keamanan sementara untuk menandatangani permintaan.

Keamanan infrastruktur untuk pengujian perangkat fisik

Perangkat dipisahkan secara fisik selama pengujian perangkat fisik. Isolasi jaringan mencegah komunikasi lintas perangkat melalui jaringan nirkabel.

Perangkat publik dibagikan, dan Device Farm melakukan upaya terbaik untuk menjaga keamanan perangkat dari waktu ke waktu. Tindakan tertentu, seperti upaya untuk memperoleh hak administrator lengkap pada perangkat (praktik yang disebut sebagai rooting atau jailbreaking), menyebabkan perangkat publik dikarantina. Mereka dihapus dari kolam umum secara otomatis dan ditempatkan ke dalam tinjauan manual.

Perangkat pribadi hanya dapat diakses oleh AWS akun yang secara eksplisit diizinkan untuk melakukannya. Device Farm secara fisik mengisolasi perangkat ini dari perangkat lain dan menyimpannya di jaringan terpisah.

Pada perangkat yang dikelola secara pribadi, pengujian dapat dikonfigurasi untuk menggunakan titik akhir VPC Amazon untuk mengamankan koneksi masuk dan keluar dari akun Anda. AWS

Keamanan infrastruktur untuk pengujian browser desktop

Saat Anda menggunakan fitur pengujian browser desktop, semua sesi pengujian dipisahkan satu sama lain. Contoh selenium tidak dapat berkomunikasi silang tanpa pihak ketiga perantara, di luar. AWS

Semua lalu lintas ke WebDriver pengontrol Selenium harus dilakukan melalui titik akhir HTTPS yang dihasilkan dengan. `createTestGridUrl`

Anda bertanggung jawab untuk memastikan bahwa setiap instance pengujian Device Farm memiliki akses aman ke sumber daya yang diuji. Secara default, instance pengujian browser desktop Device Farm memiliki akses ke internet publik. Saat Anda melampirkan instance Anda ke VPC, instans tersebut berperilaku seperti instans EC2 lainnya, dengan akses ke sumber daya yang ditentukan oleh konfigurasi VPC dan komponen jaringan yang terkait. AWS menyediakan grup keamanan dan Daftar Kontrol Akses jaringan (ACLs) untuk meningkatkan keamanan di VPC Anda. Grup keamanan mengontrol lalu lintas masuk dan keluar untuk sumber daya Anda, dan jaringan ACLs mengontrol lalu lintas masuk dan keluar untuk subnet Anda. Grup keamanan menyediakan kontrol akses yang cukup untuk sebagian besar subnet. Anda dapat menggunakan jaringan ACLs jika Anda menginginkan lapisan keamanan tambahan untuk VPC Anda. Untuk panduan umum tentang praktik terbaik keamanan saat menggunakan Amazon VPCs, lihat praktik terbaik keamanan untuk VPC Anda di Panduan Pengguna Amazon Virtual Private Cloud.

Analisis dan manajemen kerentanan konfigurasi di Device Farm

Device Farm memungkinkan Anda menjalankan perangkat lunak yang tidak dipelihara atau ditambal secara aktif oleh vendor, seperti vendor OS, vendor perangkat keras, atau operator telepon. Device Farm melakukan upaya terbaik untuk mempertahankan perangkat lunak terbaru, tetapi tidak menjamin bahwa versi tertentu dari perangkat lunak pada perangkat fisik adalah yang terbaru, dengan desain yang memungkinkan perangkat lunak yang berpotensi rentan untuk digunakan.

Misalnya, jika pengujian dilakukan pada perangkat yang menjalankan Android 4.4.2, Device Farm tidak menjamin bahwa perangkat ditambal terhadap kerentanan di Android yang dikenal sebagai. StageFright Terserah vendor (dan terkadang operator) perangkat untuk memberikan pembaruan keamanan ke perangkat. Aplikasi berbahaya yang menggunakan kerentanan ini tidak dijamin akan ditangkap oleh karantina otomatis kami.

Perangkat pribadi dikelola sesuai perjanjian Anda dengan AWS.

Device Farm melakukan upaya terbaik untuk mencegah aplikasi pelanggan dari tindakan seperti rooting atau jailbreaking. Device Farm menghapus perangkat yang dikarantina dari kolam umum hingga ditinjau secara manual.

Anda bertanggung jawab untuk menjaga setiap pustaka atau versi perangkat lunak yang Anda gunakan dalam pengujian Anda, seperti roda Python dan permata Ruby, up to date. Device Farm menyarankan agar Anda memperbarui pustaka pengujian.

Sumber daya ini dapat membantu menjaga dependensi pengujian Anda tetap mutakhir:

- Untuk informasi tentang cara mengamankan permata Ruby, lihat Praktik Keamanan di RubyGems situs web.
- Untuk informasi tentang paket keamanan yang digunakan oleh Pipenv dan didukung oleh Otoritas Pengemasan Python untuk memindai grafik ketergantungan Anda untuk mengetahui kerentanan yang diketahui, lihat Deteksi Kerentanan Keamanan pada. GitHub
- Untuk informasi tentang Open Web Application Security Project (OWASP) Maven pemeriksa ketergantungan, lihat OWASP di situs web OWASP. DependencyCheck

Penting untuk diingat bahwa meskipun sistem otomatis tidak percaya ada masalah keamanan yang diketahui, itu tidak berarti bahwa tidak ada masalah keamanan. Selalu gunakan uji tuntas saat menggunakan perpustakaan atau alat dari pihak ketiga dan verifikasi tanda tangan kriptografi bila memungkinkan atau masuk akal.

Respon insiden di Device Farm

Device Farm terus memantau perangkat untuk perilaku yang mungkin mengindikasikan masalah keamanan. Jika AWS dibuat mengetahui kasus di mana data pelanggan, seperti hasil pengujian atau file yang ditulis ke perangkat publik, dapat diakses oleh pelanggan lain, AWS kontak pelanggan yang terpengaruh, sesuai dengan kebijakan peringatan dan pelaporan insiden standar yang digunakan di seluruh AWS layanan.

Pencatatan dan pemantauan di Device Farm

Layanan ini mendukung AWS CloudTrail, yaitu layanan yang merekam AWS panggilan untuk Anda Akun AWS dan mengirimkan file log ke bucket Amazon S3. Dengan menggunakan informasi yang dikumpulkan oleh CloudTrail, Anda dapat menentukan permintaan apa yang berhasil dibuat Layanan AWS, siapa yang membuat permintaan, kapan dibuat, dan sebagainya. Untuk mempelajari selengkapnya CloudTrail, termasuk cara mengaktifkannya dan menemukan file log Anda, lihat Panduan AWS CloudTrail Pengguna.

Untuk informasi tentang penggunaan CloudTrail dengan Device Farm, lihatMencatat panggilan AWS Device Farm API dengan AWS CloudTrail.

Praktik terbaik keamanan untuk Device Farm

Device Farm menyediakan sejumlah fitur keamanan untuk dipertimbangkan saat Anda mengembangkan dan menerapkan kebijakan keamanan Anda sendiri. Praktik terbaik berikut adalah pedoman umum dan tidak mewakili solusi keamanan yang lengkap. Karena praktik terbaik ini mungkin tidak sesuai atau tidak memadai untuk lingkungan Anda, perlakukan itu sebagai pertimbangan yang bermanfaat, bukan sebagai resep.

- Berikan sistem integrasi berkelanjutan (CI) apa pun yang Anda gunakan sesedikit mungkin hak istimewa di bawah IAM. Pertimbangkan untuk menggunakan kredensil sementara untuk setiap pengujian sistem CI sehingga meskipun sistem CI dikompromikan, itu tidak dapat membuat permintaan palsu. Untuk informasi selengkapnya tentang kredensil sementara, lihat Panduan Pengguna IAM.
- Gunakan `adb` perintah di lingkungan pengujian khusus untuk membersihkan konten apa pun yang dibuat oleh aplikasi Anda. Untuk informasi selengkapnya tentang lingkungan pengujian kustom, lihatLingkungan uji kustom.

Batas di AWS Device Farm

Topik

Batas layanan

- Tidak ada batasan jumlah perangkat yang dapat Anda sertakan dalam uji coba. Namun, jumlah maksimum perangkat yang akan diuji oleh Device Farm secara bersamaan selama uji coba adalah lima. Jumlah ini dapat ditingkatkan berdasarkan permintaan dan dievaluasi berdasarkan per kasus oleh tim layanan.
- Tidak ada batasan jumlah lari yang dapat Anda jadwalkan. Perhatikan bahwa mereka hanya dapat tetap mengantri hingga 24 jam.
- Ada batas keras 150 menit untuk durasi sesi akses jarak jauh.
- Ada batas keras 150 menit untuk durasi uji coba otomatis
- Jumlah maksimum pekerjaan dalam penerbangan, termasuk pekerjaan antrian yang tertunda di seluruh akun Anda, adalah 250. Ini adalah batas lunak.
- Tidak ada batasan jumlah perangkat yang dapat Anda sertakan dalam uji coba. Jumlah perangkat (pekerjaan) yang dapat menjalankan pengujian Anda secara paralel pada waktu tertentu sama dengan konkurensi tingkat akun Anda. Konkurensi tingkat akun default untuk penggunaan terukur di Device Farm adalah lima.
- Batas konkurensi terukur dapat ditingkatkan berdasarkan permintaan hingga ambang batas tertentu tergantung pada kasus penggunaan. Konkurensi tingkat akun default untuk penggunaan yang tidak diukur sama dengan jumlah slot yang Anda berlangganan untuk platform itu.

  Untuk informasi selengkapnya mengenai batas konkurensi terukur default atau kuota secara umum, lihat halaman Kuota.
- Proses otomatisasi yang tidak menggunakan lingkungan pengujian khusus hanya dapat memiliki hingga 250 kasus pengujian individual di dalamnya. Jika tidak, lari dapat dilewati.

Batas file

- Ukuran file maksimum aplikasi yang dapat Anda unggah adalah 4 GB. Perhatikan bahwa saat ini kami tidak menerima file format.aab untuk Android.
- Ukuran maksimum video yang dihasilkan secara otomatis Device Farm selama pengujian Anda adalah 1GB. Setiap video yang melebihi ukuran ini akan memiliki semua konten video yang tersisa terpotong. Pelanggan masih dapat menggunakan solusi perekaman video mereka sendiri, jika ada, dan menyimpannya di luar penyimpanan terkelola Device Farm.
- Ukuran maksimum log perangkat yang dibuat secara otomatis Device Farm (logcat di Android atau syslog di iOS) selama pengujian Anda dijalankan adalah 1GB. Setiap log yang melebihi ukuran ini akan memiliki semua log yang tersisa terpotong. Untuk log yang lebih besar dari 1 GB, Pelanggan dapat menyimpan log ini di luar penyimpanan terkelola Device Farm.
- Ukuran maksimum kumulatif artefak pelanggan mode lingkungan kustom Device Farm adalah 1GB. Jika ukuran ini dilampaui oleh artefak Anda, maka tidak ada artefak yang akan tersedia.
- Jika ukuran kumulatif semua artefak yang dihasilkan selama uji coba melebihi 4GB, maka beberapa artefak dapat dijatuhkan (termasuk video, log perangkat, dan artefak pelanggan).

Batas API

- Device Farm mengikuti algoritma token-bucket untuk membatasi tingkat panggilan API. Misalnya, bayangkan membuat ember yang menyimpan token. Setiap token mewakili satu transaksi, dan satu panggilan API menggunakan token. Token ditambahkan ke bucket dengan tarif tetap (misalnya, 10 token per detik), dan bucket memiliki kapasitas maksimum (misalnya, 100 token). Ketika permintaan atau paket tiba, ia harus mengklaim token dari ember untuk diproses. Jika ada cukup token, permintaan diizinkan dan token dihapus. Jika tidak ada cukup token, permintaan akan tertunda atau dibatalkan, tergantung pada implementasinya.

  Di Device Farm, ini adalah bagaimana algoritma diimplementasikan:

  - Permintaan Burst API adalah jumlah maksimum permintaan yang dapat ditanggapi oleh layanan untuk API tertentu dalam ID akun pelanggan tertentu. Dengan kata lain, ini adalah kapasitas ember. Anda dapat memanggil API sebanyak ada token yang tersisa di bucket, dan setiap permintaan menggunakan satu token.
  - Tingkat Transactions-per-second (TPS) adalah tingkat minimum di mana permintaan API Anda dapat dieksekusi. Dengan kata lain, ini adalah tingkat di mana ember diisi ulang dengan token per detik. Misalnya, jika API memiliki angka burst sepuluh tetapi TPS satu, Anda dapat

memanggilnya sepuluh kali secara instan. Namun, bucket hanya akan mendapatkan kembali token dengan kecepatan satu token per detik, sehingga dibatasi menjadi satu panggilan per detik kecuali Anda berhenti memanggil API untuk membiarkan ember diisi ulang.

Berikut adalah tarif untuk Device Farm APIs:

- Untuk List dan Get APIs, kapasitas permintaan Burst API adalah`50`, dan tingkat Transactions-per-second (TPS) adalah`10`.
- Untuk semua yang lain APIs, kapasitas permintaan Burst API adalah`10`, dan tingkat Transactions-per-second (TPS) adalah`1`.

Batas titik akhir Appium

Batasan berikut berlaku untuk semua sesi titik akhir Appium. Untuk pertanyaan dan panduan tentang cara terbaik menangani batasan, silakan buka kasus dukungan.

- Setiap perintah Appium memiliki batas durasi eksekusi 4 menit, setelah itu perintah habis.
- Titik akhir menerima ukuran muatan input hingga 20MB, dan memungkinkan ukuran muatan keluaran hingga 20MB. Setiap permintaan dengan ukuran input atau output yang lebih besar dari ini akan menerima WebDriver kesalahan`'unsupported operation'`.
- Permintaan dijalankan secara berurutan pada perangkat dalam urutan yang diterima. Akibatnya, kami sangat menyarankan untuk mengirim perintah secara berurutan, dan menunggu respons setiap perintah sebelum mengirim yang baru. Konon, perintah server Appium tertentu dapat dikirim secara paralel, khususnya:
  - GetStatus
  - GetSessions
- Endpoint tidak mendukung WebDriver BiDi protokol saat ini.
- Titik akhir tidak mendukung Plugin Appium atau driver selain driver dan. XCUITest UIAutomator2
- Maksimal 3 aplikasi dapat digunakan sebagai aplikasi tambahan dengan permintaan pembuatan sesi akses jarak jauh. Konon, tidak ada batasan berapa banyak aplikasi yang dapat diinstal selama sesi menggunakan `InstallToRemoteAccessSession`API.

Batas variabel lingkungan kustom

Batasan berikut berlaku untuk semua variabel lingkungan kustom. Untuk pertanyaan dan panduan tentang cara terbaik menangani batasan, silakan buka kasus dukungan.

- Maxiumum 32 variabel dapat dikonfigurasi pada proyek Device Farm tertentu atau dijalankan.
- Nama variabel tidak boleh melebihi 256 karakter panjangnya.
- Nama variabel tunduk pada batasan yang diberlakukan oleh`bash`. Yaitu, mereka harus berisi hanya karakter alfanumerik dan garis bawah, dan tidak dapat memulai dengan angka.
- Nama variabel yang dimulai dengan `$DEVICEFARM_` dicadangkan untuk penggunaan layanan internal.
- Nilai variabel tidak boleh melebihi 256 karakter panjangnya.
- Variabel lingkungan tidak dapat digunakan untuk mengonfigurasi pemilihan komputasi host uji dalam file spesifikasi pengujian.

Alat dan plugin untuk AWS Device Farm

Bagian ini berisi tautan dan informasi tentang bekerja dengan alat dan plugin AWS Device Farm. Anda dapat menemukan plugin Device Farm di AWS Labs. GitHub

Jika Anda adalah pengembang Android, kami juga memiliki aplikasi contoh AWS Device Farm untuk Android GitHub. Anda dapat menggunakan aplikasi dan contoh pengujian sebagai referensi untuk skrip pengujian Device Farm Anda sendiri.

Topik

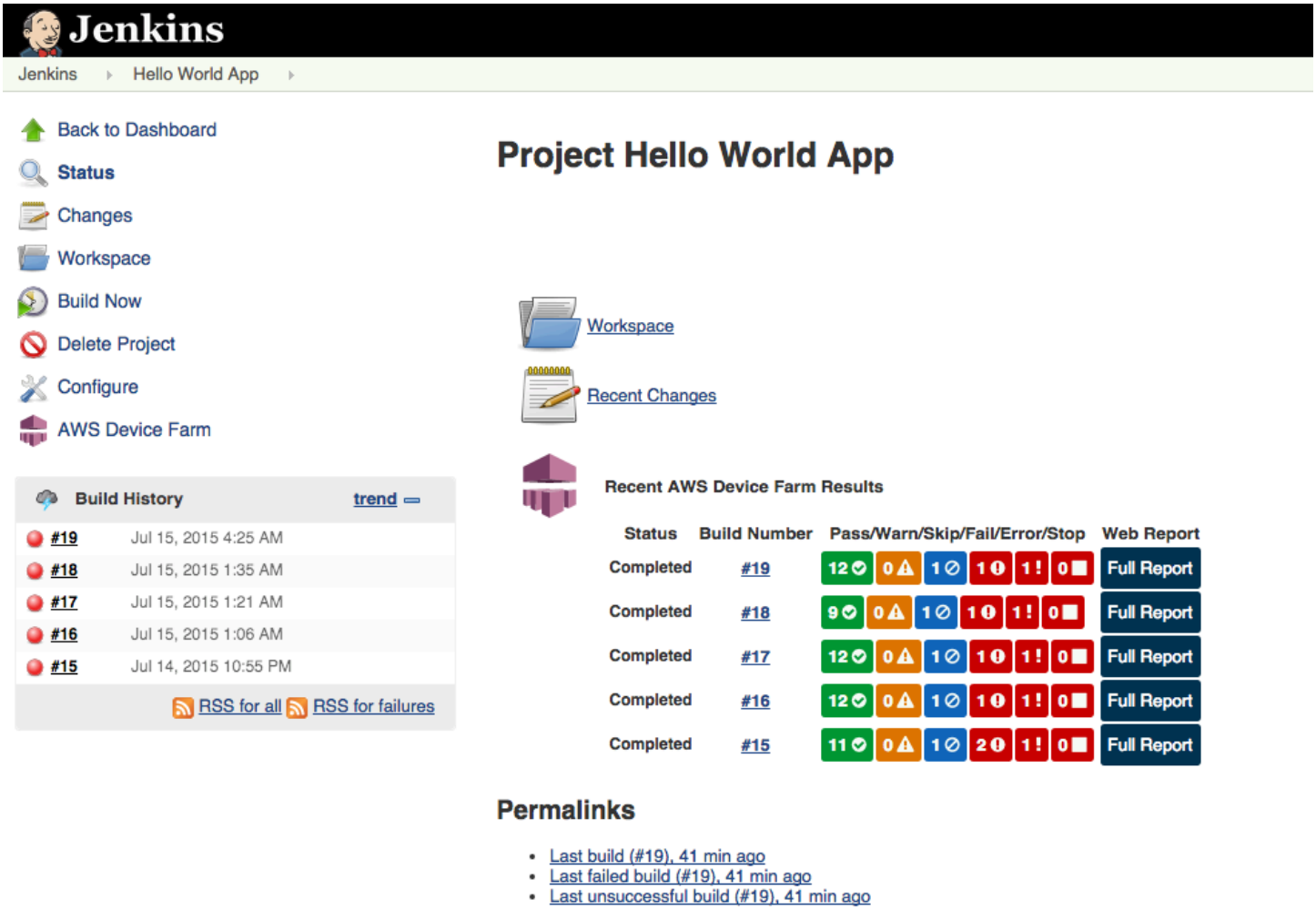
Mengintegrasikan Device Farm dengan server Jenkins CI

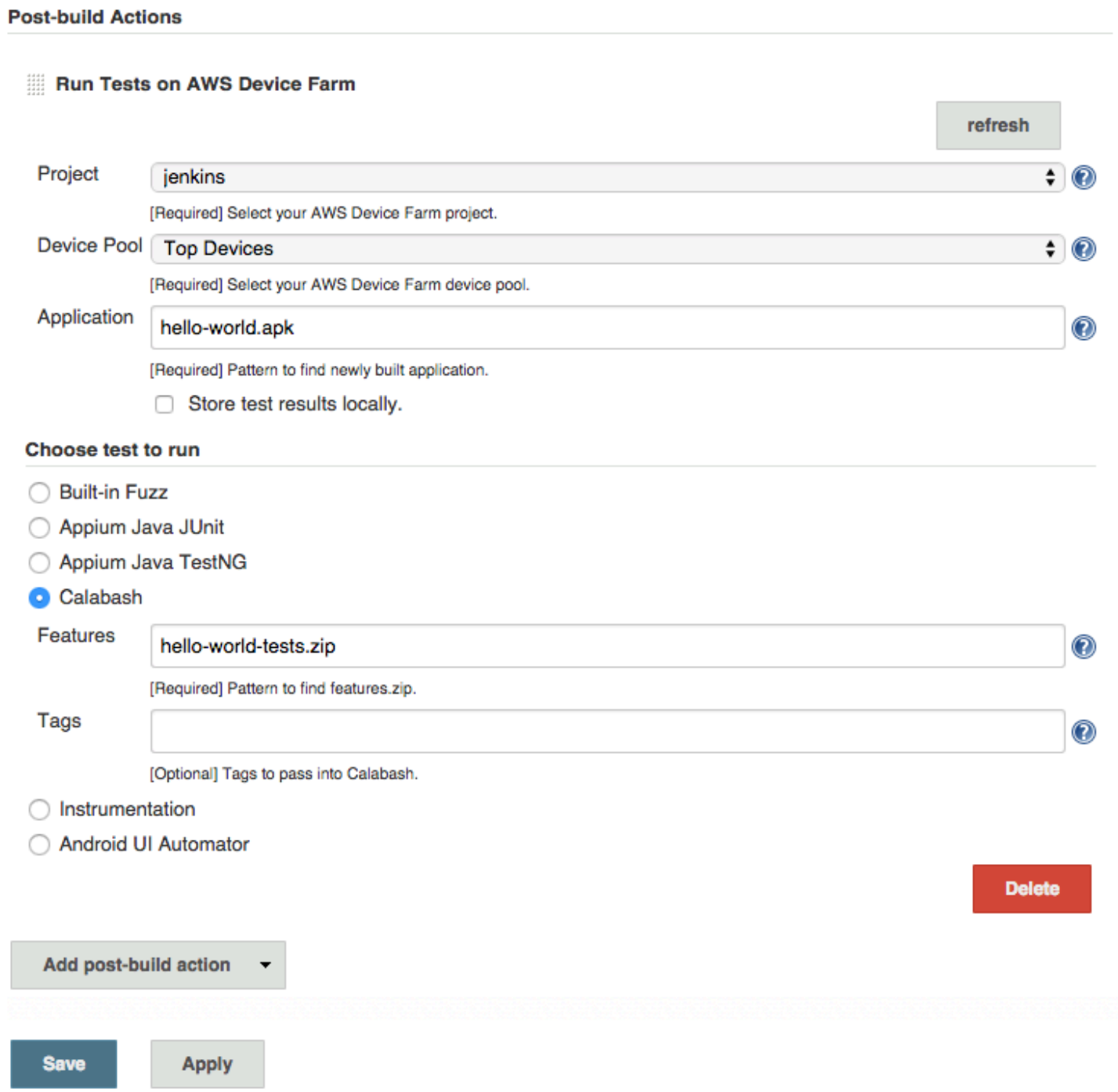
Plugin Jenkins CI menyediakan fungsionalitas AWS Device Farm dari server integrasi berkelanjutan (CI) Jenkins Anda sendiri. Untuk informasi lebih lanjut, lihat Jenkins (perangkat lunak).

> **Note**
>
> Untuk mengunduh plugin Jenkins, buka GitHubdan ikuti instruksi diLangkah 1: Menginstal plugin Jenkins CI untuk AWS Device Farm.

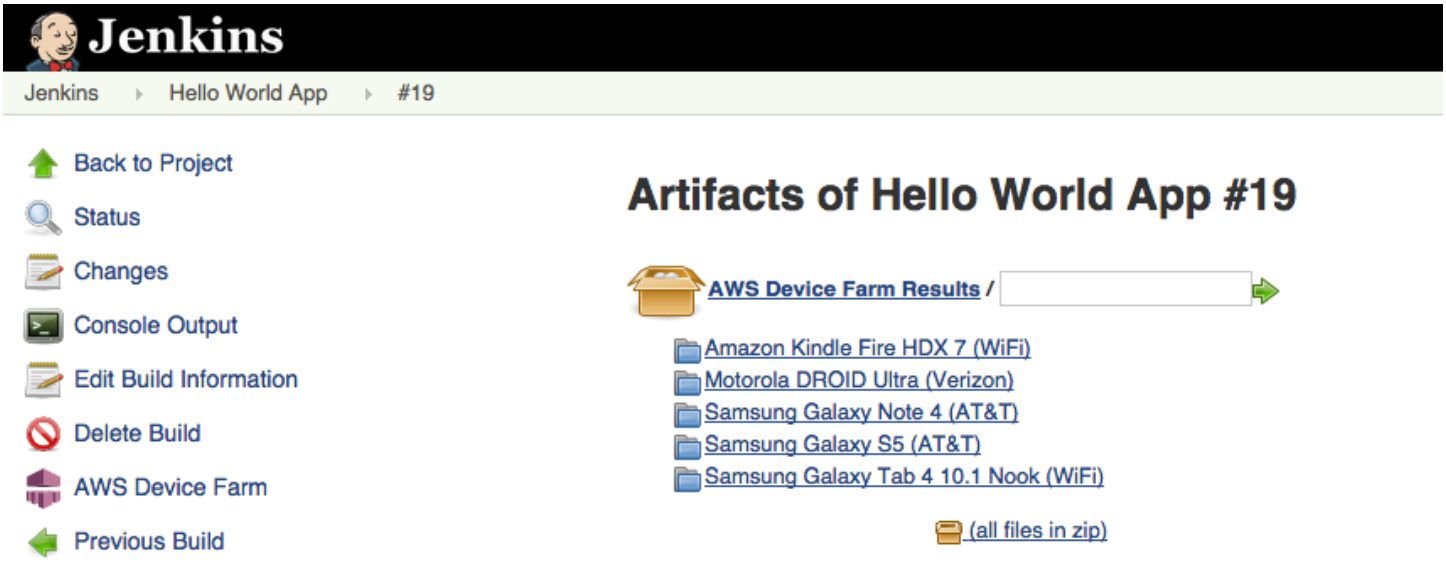
Bagian ini berisi serangkaian prosedur untuk menyiapkan dan menggunakan plugin Jenkins CI dengan AWS Device Farm.

Gambar-gambar berikut menunjukkan fitur plugin Jenkins CI.


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



Plugin ini juga dapat menarik semua artefak pengujian (log, tangkapan layar, dll.) Secara lokal:



Topik

- Dependensi
- Langkah 1: Menginstal plugin Jenkins CI untuk AWS Device Farm
- Langkah 2: Membuat AWS Identity and Access Management pengguna untuk Plugin Jenkins CI Anda untuk AWS Device Farm
- Langkah 3: Mengonfigurasi plugin Jenkins CI untuk pertama kalinya di AWS Device Farm
- Langkah 4: Menggunakan plugin dalam pekerjaan Jenkins

Dependensi

Plugin Jenkins CI memerlukan AWS Mobile SDK 1.10.5 atau yang lebih baru. Untuk informasi selengkapnya dan untuk menginstal SDK, lihat AWS Mobile SDK.

Langkah 1: Menginstal plugin Jenkins CI untuk AWS Device Farm

Ada dua opsi untuk menginstal plugin Jenkins continuous integration (CI) untuk AWS Device Farm. Anda dapat mencari plugin dari dalam dialog Plugin yang Tersedia di UI Web Jenkins, atau Anda dapat mengunduh `hpi` file dan menginstalnya dari dalam Jenkins.

Instal dari dalam UI Jenkins

1. Temukan plugin dalam UI Jenkins dengan memilih Kelola Jenkins, Kelola Plugin, dan kemudian pilih Tersedia.

2. Cari aws-device-farm.
3. Instal plugin AWS Device Farm.
4. Pastikan plugin tersebut dimiliki oleh `Jenkins` pengguna.
5. Mulai ulang Jenkins.

Unduh pluginnya

1. Unduh `hpi` file langsung dari http://updates.jenkins-ci. org/latest/aws-perangkat-farm.hpi.
2. Pastikan plugin tersebut dimiliki oleh `Jenkins` pengguna.
3. Instal plugin menggunakan salah satu opsi berikut:
   - Upload plugin dengan memilih Manage Jenkins, Manage Plugins, Advanced, dan kemudian pilih Upload plugin.
   - Tempatkan `hpi` file di direktori plugin Jenkins (biasanya`/var/lib/jenkins/plugins`).
4. Mulai ulang Jenkins.

Langkah 2: Membuat AWS Identity and Access Management pengguna untuk Plugin Jenkins CI Anda untuk AWS Device Farm

Kami menyarankan Anda untuk tidak menggunakan akun AWS root Anda untuk mengakses Device Farm. Sebagai gantinya, buat pengguna baru AWS Identity and Access Management (IAM) (atau gunakan pengguna IAM yang sudah ada) di AWS akun Anda, lalu akses Device Farm dengan pengguna IAM tersebut.

Untuk membuat pengguna IAM baru, lihat Membuat Pengguna IAM ()Konsol Manajemen AWS. Pastikan untuk membuat kunci akses untuk setiap pengguna dan mengunduh atau menyimpan kredensi keamanan pengguna. Anda akan membutuhkan kredensialnya nanti.

Berikan izin kepada pengguna IAM untuk mengakses Device Farm

Untuk memberikan izin kepada pengguna IAM untuk mengakses Device Farm, buat kebijakan akses baru di IAM, lalu tetapkan kebijakan akses ke pengguna IAM sebagai berikut.

> **Note**
>
> Akun AWS root atau pengguna IAM yang Anda gunakan untuk menyelesaikan langkah-langkah berikut harus memiliki izin untuk membuat kebijakan IAM berikut dan melampirkannya ke pengguna IAM. Untuk informasi selengkapnya, lihat Bekerja dengan Kebijakan

Untuk membuat kebijakan akses di IAM

1. Buka konsol IAM di https://console.aws.amazon.com/iam/.
2. Pilih Policies (Kebijakan).
3. Pilih Buat Kebijakan. (Jika tombol Memulai muncul, pilihlah, lalu pilih Buat Kebijakan.)
4. Di sebelah Buat Kebijakan Anda Sendiri, pilih Pilih.
5. Untuk Nama Kebijakan, ketikkan nama untuk kebijakan (misalnya,**`AWSDeviceFarmAccessPolicy`**).
6. Untuk Deskripsi, ketikkan deskripsi yang membantu Anda mengaitkan pengguna IAM ini dengan proyek Jenkins Anda.
7. Untuk Dokumen Kebijakan, ketik pernyataan berikut:

   JSON

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

8. Pilih Buat Kebijakan.

Untuk menetapkan kebijakan akses ke pengguna IAM

1. Buka konsol IAM di https://console.aws.amazon.com/iam/.
2. Pilih Pengguna.
3. Pilih pengguna IAM kepada siapa Anda akan menetapkan kebijakan akses.
4. Di area Izin, untuk Kebijakan Terkelola, pilih Lampirkan Kebijakan.
5. Pilih kebijakan yang baru saja Anda buat (misalnya, AWSDeviceFarmAccessPolicy).
6. Pilih Lampirkan Kebijakan.

Langkah 3: Mengonfigurasi plugin Jenkins CI untuk pertama kalinya di AWS Device Farm

Pertama kali Anda menjalankan server Jenkins Anda, Anda perlu mengkonfigurasi sistem sebagai berikut.

> **Note**
>
> Jika Anda menggunakan slot perangkat, fitur slot perangkat dinonaktifkan secara default.

1. Masuk ke antarmuka pengguna Web Jenkins Anda.
2. Di sisi kiri layar, pilih Kelola Jenkins.
3. Pilih Konfigurasi Sistem.
4. Gulir ke bawah ke header AWS Device Farm.
5. Salin kredensi keamanan Anda dari Membuat pengguna IAM untuk Plugin Jenkins CI Anda dan tempel ID Kunci Akses dan Kunci Akses Rahasia Anda ke dalam kotak masing-masing.
6. Pilih Simpan.

Langkah 4: Menggunakan plugin dalam pekerjaan Jenkins

Setelah Anda menginstal plugin Jenkins, ikuti petunjuk ini untuk menggunakan plugin dalam pekerjaan Jenkins.

1. Masuk ke UI web Jenkins Anda.

2. Klik pekerjaan yang ingin Anda edit.
3. Di sisi kiri layar, pilih Konfigurasi.
4. Gulir ke bawah ke header Post-build Actions.
5. Klik Tambahkan tindakan pasca-pembuatan dan pilih Jalankan Pengujian di AWS Device Farm.
6. Pilih proyek yang ingin Anda gunakan.
7. Pilih kumpulan perangkat yang ingin Anda gunakan.
8. Pilih apakah Anda ingin artefak pengujian (seperti log dan tangkapan layar) diarsipkan secara lokal.
9. Di Aplikasi, isi jalur ke aplikasi yang dikompilasi Anda.
10. Pilih tes yang ingin Anda jalankan dan isi semua bidang yang diperlukan.
11. Pilih Simpan.

Mengintegrasikan Device Farm dengan sistem build Gradle

Plugin Device Farm Gradle menyediakan integrasi AWS Device Farm dengan sistem build Gradle di Android Studio. Untuk informasi selengkapnya, lihat Gradle.

> **Note**
>
> Untuk mengunduh plugin Gradle, buka GitHubdan ikuti instruksi diMembangun plugin Device Farm Gradle.

Plugin Device Farm Gradle menyediakan fungsionalitas Device Farm dari lingkungan Android Studio Anda. Anda dapat memulai tes di ponsel dan tablet Android nyata yang dihosting oleh Device Farm.

Bagian ini berisi serangkaian prosedur untuk menyiapkan dan menggunakan Plugin Device Farm Gradle.

Topik

- Dependensi
- Langkah 1: Membangun plugin AWS Device Farm Gradle
- Langkah 2: Menyiapkan plugin AWS Device Farm Gradle

- Langkah 3: Menghasilkan pengguna IAM di plugin Device Farm Gradle
- Langkah 4: Mengkonfigurasi jenis tes

Dependensi

Runtime

- Plugin Device Farm Gradle memerlukan AWS Mobile SDK 1.10.15 atau yang lebih baru. Untuk informasi selengkapnya dan untuk menginstal SDK, lihat AWS Mobile SDK.
- API uji pembuat alat Android 0.5.2
- Apache Commons Lang3 3.3.4

Untuk Tes Unit

- Testng 6.8.8
- Jmockit 1,19
- Alat gradle Android 1.3.0

Langkah 1: Membangun plugin AWS Device Farm Gradle

Plugin ini menyediakan integrasi AWS Device Farm dengan sistem build Gradle di Android Studio. Untuk informasi selengkapnya, lihat Gradle.

> **Note**
>
> Membangun plugin adalah opsional. Plugin ini diterbitkan melalui Maven Central. Jika Anda ingin mengizinkan Gradle mengunduh plugin secara langsung, lewati langkah ini dan langsung keLangkah 2: Menyiapkan plugin AWS Device Farm Gradle.

Untuk membangun plugin

1. Pergi ke GitHubdan kloning repositori.
2. Membangun plugin menggunakan`gradle install`.

   Plugin diinstal ke repositori maven lokal Anda.

Langkah selanjutnya: Langkah 2: Menyiapkan plugin AWS Device Farm Gradle

Langkah 2: Menyiapkan plugin AWS Device Farm Gradle

Jika Anda belum melakukannya, kloning repositori dan instal plugin menggunakan prosedur di sini:. Membangun plugin Device Farm Gradle

Untuk mengonfigurasi Plugin AWS Device Farm Gradle

1. Tambahkan artefak plugin ke daftar ketergantungan Anda di. `build.gradle`

```
    buildscript {

        repositories {
            mavenLocal()
            mavenCentral()
        }

        dependencies {
            classpath 'com.android.tools.build:gradle:1.3.0'
            classpath 'com.amazonaws:aws-devicefarm-gradle-plugin:1.0'
        }
    }
```

2. Konfigurasikan plugin di `build.gradle` file Anda. Konfigurasi khusus pengujian berikut harus berfungsi sebagai panduan Anda:

```
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. Jalankan pengujian Device Farm Anda menggunakan tugas berikut:`gradle devicefarmUpload`.

   Output build akan mencetak link ke konsol Device Farm tempat Anda dapat memantau eksekusi pengujian.

Langkah selanjutnya: Menghasilkan pengguna IAM di plugin Device Farm Gradle

Langkah 3: Menghasilkan pengguna IAM di plugin Device Farm Gradle

AWS Identity and Access Management (IAM) membantu Anda mengelola izin dan kebijakan untuk bekerja dengan AWS sumber daya. Topik ini memandu Anda untuk menghasilkan pengguna IAM dengan izin untuk mengakses sumber daya AWS Device Farm.

Jika Anda belum melakukannya, selesaikan langkah 1 dan 2 sebelum membuat pengguna IAM.

Kami menyarankan Anda untuk tidak menggunakan akun AWS root Anda untuk mengakses Device Farm. Sebagai gantinya, buat pengguna IAM baru (atau gunakan pengguna IAM yang sudah ada) di AWS akun Anda, lalu akses Device Farm dengan pengguna IAM tersebut.

> **Note**
>
> Akun AWS root atau pengguna IAM yang Anda gunakan untuk menyelesaikan langkah-langkah berikut harus memiliki izin untuk membuat kebijakan IAM berikut dan melampirkannya ke pengguna IAM. Untuk informasi selengkapnya, lihat Bekerja dengan Kebijakan.

Untuk membuat pengguna baru dengan kebijakan akses yang tepat di IAM

1. Buka konsol IAM di https://console.aws.amazon.com/iam/.
2. Pilih Pengguna.
3. Pilih Buat Antrean Baru.
4. Masukkan nama pengguna pilihan Anda.

   Misalnya, **`GradleUser`**.
5. Pilih Buat.
6. Pilih Unduh Kredensial dan simpan di lokasi di mana Anda dapat dengan mudah mengambilnya nanti.
7. Pilih Tutup.
8. Pilih nama pengguna dalam daftar.
9. Di bawah Izin, perluas header Kebijakan Sebaris dengan mengklik panah bawah di sebelah kanan.
10. Pilih Klik di sini di mana dikatakan, Tidak ada kebijakan sebaris untuk ditampilkan. Untuk membuatnya, klik di sini.
11. Pada layar Atur izin, pilih Kebijakan Kustom.
12. Pilih Pilih.
13. Beri nama kebijakan Anda, seperti **`AWSDeviceFarmGradlePolicy`**.
14. Tempel kebijakan berikut ke dalam Dokumen Kebijakan.

    JSON

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

15. Pilih Terapkan Kebijakan.

Langkah selanjutnya:Mengkonfigurasi jenis pengujian.

Untuk informasi selengkapnya, lihat Membuat Pengguna IAM (Konsol Manajemen AWS) atauMenyiapkan.

Langkah 4: Mengkonfigurasi jenis tes

Secara default, plugin AWS Device Farm Gradle menjalankan Instrumentasi untuk Android dan AWS Device Farm pengujian. Jika Anda ingin menjalankan pengujian Anda sendiri atau menentukan parameter tambahan, Anda dapat memilih untuk mengonfigurasi jenis pengujian. Topik ini memberikan informasi tentang setiap jenis pengujian yang tersedia dan apa yang perlu Anda lakukan di Android Studio untuk mengonfigurasinya agar dapat digunakan. Untuk informasi selengkapnya tentang jenis pengujian yang tersedia di Device Farm, lihatUji kerangka kerja dan pengujian bawaan di AWS Device Farm.

Jika Anda belum melakukannya, selesaikan langkah 1 — 3 sebelum mengonfigurasi jenis pengujian.

> **Note**
>
> Jika Anda menggunakan slot perangkat, fitur slot perangkat dinonaktifkan secara default.

Appium

Device Farm menyediakan dukungan untuk Appium Java dan JUnit TestNG untuk Android.

- Appium (di bawah Jawa ()) JUnit
- Appium (di bawah Jawa (TestNG))

Anda dapat memilih `useTestNG()` atau`useJUnit()`. `JUnit`adalah default dan tidak perlu ditentukan secara eksplisit.

```
    appium {
        tests file("path to zip file") // required
        useTestNG() // or useJUnit()
    }
```

Bawaan: bulu halus

Device Farm menyediakan tipe uji fuzz bawaan, yang secara acak mengirimkan peristiwa antarmuka pengguna ke perangkat dan kemudian melaporkan hasilnya.

```
    fuzz {

       eventThrottle 50 // optional default
       eventCount 6000  // optional default
       randomizerSeed 1234 // optional default blank

     }
```

Untuk informasi selengkapnya, lihat Menjalankan uji fuzz bawaan Device Farm (Android dan iOS).

Instrumentasi

Device Farm menyediakan dukungan untuk instrumentasi (JUnit, Espresso, Robotium, atau pengujian berbasis instrumen apa pun) untuk Android. Untuk informasi selengkapnya, lihat Instrumentasi untuk Android dan AWS Device Farm.

Saat menjalankan pengujian instrumentasi di Gradle, Device Farm menggunakan `.apk` file yang dihasilkan dari direktori AndroidTest sebagai sumber pengujian Anda.

```
    instrumentation {

        filter "test filter per developer docs" // optional

    }
```

Riwayat dokumen AWS Device Farm

Tabel berikut menjelaskan perubahan penting pada dokumentasi sejak rilis terakhir panduan ini.

Ubah	Deskripsi	Tanggal Diubah
Dukungan titik akhir Appium	Device Farm sekarang menawarkan titik akhir Appium yang dikelola sepenuhnya untuk pengujian perangkat jarak jauh, memungkinkan pengembangan pengujian cepat dan debugging. Ini melengkapi metode eksekusi sisi server yang ada, tempat pengujian diunggah dan dijalankan langsung di Device Farm. Meskipun eksekusi sisi server sangat ideal untuk CI/CD pipeline dan pengujian skala besar, titik akhir Appium lokal yang baru memungkinkan iterasi dan pengembangan pengujian yang lebih cepat pada perangkat nyata.	November 17, 2025
Peningkatan host uji iOS	Device Farm kini mendukung pengalaman terbaru untuk lingkungan pengujian iOS, memungkinkan konsistensi dalam pengaturan antara pengujian Android dan iOS. Lihat Host untuk lingkungan pengujian khusus untuk mempelajari selengkapnya. Selain itu, informasi yang terkait dengan pensiunan host uji Android telah dihapus. Pengguna Android didorong untuk menggunakan host uji Amazon Linux 2.	Oktober 31, 2025
AL2 dukungan	Device Farm sekarang mendukung lingkungan AL2 pengujian untuk Android. Pelajari lebih lanjut tentang AL2.	6 November 2023
Migrasi dari Standar ke lingkungan pengujian Kustom	Panduan migrasi yang diperbarui untuk mendokume ntasikan penghentian pengujian mode standar pada Desember 2023.	September 3, 2023
Dukungan VPC ENI	Device Farm sekarang memungkinkan perangkat pribadi untuk menggunakan fitur konektivitas VPC-ENI untuk membantu pelanggan terhubung dengan aman ke titik akhir	15 Mei 2023

Ubah	Deskripsi	Tanggal Diubah
	pribadi mereka yang dihosting di AWS, perangkat lunak lokal, atau penyedia cloud lainnya. Pelajari lebih lanjut tentang VPC-ENI.	
Pembaruan UI Polaris	Konsol Device Farm sekarang mendukung kerangka kerja Polaris.	28 Juli 2021
Dukungan Python 3	Device Farm sekarang mendukung Python 3 dalam pengujian mode kustom. Pelajari lebih lanjut tentang menggunakan Python 3 dalam paket pengujian Anda: • Appium (Python) • Appium (Python)	20 April 2020
Informasi keamanan baru dan informasi tentang AWS sumber daya penandaan.	Untuk membuat AWS layanan pengamanan lebih mudah dan lebih komprehensif, bagian baru tentang keamanan telah dibangun. Untuk membaca lebih lanjut, lihat Keamanan di AWS Device Farm Bagian baru tentang penandaan di Device Farm telah ditambahkan. Untuk mempelajari lebih lanjut tentang penandaan, lihatMenandai di Device Farm.	27 Maret 2020
Penghapusan Akses Perangkat Langsung.	Direct Device Access (debugging jarak jauh pada perangkat pribadi) tidak lagi tersedia untuk penggunaa n umum. Untuk pertanyaan tentang ketersediaan Direct Device Access di masa mendatang, silakan hubungi kami.	9 September 2019
Perbarui konfigurasi plugin Gradle	Konfigurasi plugin Gradle yang direvisi sekarang menyertak an versi konfigurasi gradle yang dapat disesuaikan, dengan parameter opsional yang dikomentari. Pelajari lebih lanjut tentang Menyiapkan plugin Device Farm Gradle.	16 Agustus 2019

Ubah	Deskripsi	Tanggal Diubah
Persyaratan baru untuk uji coba dengan XCTest	Untuk pengujian yang menggunakan XCTest framework , Device Farm sekarang memerlukan paket aplikasi yang dibuat untuk pengujian. Pelajari lebih lanjut tentang the section called “XCTest”.	4 Februari 2019
Support untuk jenis pengujian Appium Node.js dan Appium Ruby di lingkungan khusus	Anda sekarang dapat menjalankan pengujian di lingkunga n pengujian kustom Appium Node.js dan Appium Ruby. Pelajari lebih lanjut tentang Uji kerangka kerja dan pengujian bawaan di AWS Device Farm.	Januari 10, 2019
Support untuk server Appium versi 1.7.2 di lingkungan standar dan kustom. Support untuk versi 1.8.1 menggunakan file YAMM spesifika si pengujian kustom di lingkungan pengujian khusus.	Sekarang Anda dapat menjalankan pengujian di lingkunga n pengujian standar dan kustom dengan server Appium versi 1.72, 1.71, dan 1.6.5. Anda juga dapat menjalank an pengujian dengan versi 1.8.1 dan 1.8.0 menggunak an file YAMM spesifikasi pengujian kustom di lingkungan pengujian kustom. Pelajari lebih lanjut tentang Uji kerangka kerja dan pengujian bawaan di AWS Device Farm.	2 Oktober 2018
Lingkungan uji kustom	Dengan lingkungan pengujian khusus, Anda dapat memastikan pengujian berjalan seperti yang dilakukan di lingkungan lokal Anda. Device Farm sekarang menyediak an dukungan untuk live log dan streaming video, sehingga Anda bisa mendapatkan umpan balik instan tentang pengujian yang dijalankan di lingkungan pengujian khusus. Pelajari lebih lanjut tentang Lingkungan pengujian khusus di AWS Device Farm.	Agustus, 16 2018

Ubah	Deskripsi	Tanggal Diubah
Support untuk menggunakan Device Farm sebagai penyedia AWS CodePipeline pengujian	Anda sekarang dapat mengonfigurasi pipeline AWS CodePipeline untuk menggunakan AWS Device Farm berjalan sebagai tindakan pengujian dalam proses rilis Anda. CodePipeline memungkinkan Anda untuk dengan cepat menghubungkan repositori Anda untuk membangun dan menguji tahapan untuk mencapai sistem integrasi berkelanjutan yang disesuaikan dengan kebutuhan Anda. Pelajari lebih lanjut tentang Mengintegrasikan AWS Device Farm dalam tahap CodePipeline pengujian.	Juli, 19 2018
Support untuk Perangkat Pribadi	Anda sekarang dapat menggunakan perangkat pribadi untuk menjadwalkan uji coba dan memulai sesi akses jarak jauh. Anda dapat mengelola profil dan pengaturan untuk perangkat ini, membuat titik akhir VPC Amazon untuk menguji aplikasi pribadi, dan membuat sesi debugging jarak jauh. Pelajari lebih lanjut tentang Perangkat pribadi di AWS Device Farm.	2 Mei 2018
Support untuk Appium 1.6.3	Anda sekarang dapat mengatur versi Appium untuk pengujian kustom Appium Anda.	21 Maret 2017
Tetapkan batas waktu eksekusi untuk uji coba	Anda dapat mengatur batas waktu eksekusi untuk uji coba atau untuk semua pengujian dalam proyek. Pelajari lebih lanjut tentang Menyetel batas waktu eksekusi untuk pengujian berjalan di AWS Device Farm.	9 Februari 2017
Pembentukan Jaringan	Anda sekarang dapat mensimulasikan koneksi jaringan dan kondisi untuk uji coba. Pelajari lebih lanjut tentang Mensimulasikan koneksi dan kondisi jaringan untuk AWS Device Farm Anda berjalan.	8 Desember 2016

Ubah	Deskripsi	Tanggal Diubah
Bagian Pemecahan Masalah Baru	Sekarang Anda dapat memecahkan masalah unggahan paket pengujian menggunakan serangkaian prosedur yang dirancang untuk mengatasi pesan kesalahan yang mungkin Anda temui di konsol Device Farm. Pelajari lebih lanjut tentang Memecahkan masalah kesalahan Device Farm.	Agustus 10, 2016
Sesi Akses Jarak Jauh	Anda sekarang dapat mengakses dan berinteraksi dari jarak jauh dengan satu perangkat di konsol. Pelajari lebih lanjut tentang Akses jarak jauh.	19 April 2016
Slots Perangkat Layanan Mandiri	Anda sekarang dapat membeli slot perangkat menggunak an Konsol Manajemen AWS, AWS Command Line Interface, atau API. Pelajari lebih lanjut tentang cara Membeli slot perangkat di Device Farm.	22 Maret 2016
Cara menghentikan uji coba	Anda sekarang dapat menghentikan pengujian berjalan menggunakan Konsol Manajemen AWS, the AWS Command Line Interface, atau API. Pelajari lebih lanjut tentang cara Menghentikan proses di AWS Device Farm.	22 Maret 2016
Jenis uji XCTest UI baru	Anda sekarang dapat menjalankan pengujian kustom XCTest UI pada aplikasi iOS. Pelajari lebih lanjut tentang jenis Mengintegrasikan XCTest UI untuk iOS dengan Device Farm tes.	Maret 8, 2016
Jenis pengujian Appium Python baru	Anda sekarang dapat menjalankan pengujian kustom Appium Python di aplikasi Android, iOS, dan web. Pelajari lebih lanjut tentang Uji kerangka kerja dan pengujian bawaan di AWS Device Farm.	19 Januari 2016
Jenis pengujian Aplikasi Web	Anda sekarang dapat menjalankan tes kustom Appium Java dan JUnit TestNG pada aplikasi web. Pelajari lebih lanjut tentang Pengujian aplikasi web di AWS Device Farm.	19 November 2015

Ubah	Deskripsi	Tanggal Diubah
Plugin Gradle AWS Device Farm	Pelajari lebih lanjut tentang cara menginstal dan menggunakanPlugin Device Farm Gradle.	28 September 2015
Uji Bawaan Android Baru: Explorer	Uji explorer meng-crawl aplikasi Anda dengan menganali sis setiap layar seolah-olah itu adalah pengguna akhir dan mengambil tangkapan layar saat menjelajah.	16 September 2015
Dukungan iOS ditambahkan	Pelajari lebih lanjut tentang menguji perangkat iOS dan menjalankan pengujian iOS (termasuk XCTest) diUji kerangka kerja dan pengujian bawaan di AWS Device Farm.	4 Agustus 2015
Rilis publik awal	Ini adalah rilis publik awal dari AWS Device Farm Developer Guide.	Juli 13, 2015

AWS Glosarium

Untuk AWS terminologi terbaru, lihat AWS glosarium di Referensi.Glosarium AWS

Terjemahan disediakan oleh mesin penerjemah. Jika konten terjemahan yang diberikan bertentangan dengan versi bahasa Inggris aslinya, utamakan versi bahasa Inggris.