Amazon Pinpoint

Developer Guide

Amazon Pinpoint: Developer Guide

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

What Is Amazon Pinpoint?

Amazon Pinpoint is an AWS service that you can use to engage with your customers across multiple messaging channels. You can use Amazon Pinpoint to send push notifications, emails, SMS text messages, or voice messages.

The information in this developer guide is intended for application developers. This guide contains information about using the features of Amazon Pinpoint programmatically. It also contains information of particular interest to mobile app developers, such as procedures for integrating analytics and messaging features with your application (p. 3).

There are several other documents that are companions to this document. The following documents provide reference information related to the Amazon Pinpoint APIs:

- Amazon Pinpoint API Reference
- Amazon Pinpoint Email API
- Amazon Pinpoint SMS and Voice API

If you're new to Amazon Pinpoint, you might find it helpful to review the Amazon Pinpoint User Guide before proceeding with this document.

Amazon Pinpoint Features

This section describes the major features of Amazon Pinpoint.

Define Audience Segments

Reach the right audience for your messages by defining audience segments (p. 80). A segment designates which users receive the messages that are sent from a campaign. You can define dynamic segments based on data that's reported by your application, such as operating system or mobile device type. You can also import static segments that you define outside of Amazon Pinpoint.

Engage Your Audience with Messaging Campaigns

Engage your audience by creating a messaging campaign (p. 90). A campaign sends tailored messages on a schedule that you define. You can create campaigns that send mobile push, email, or SMS messages.

To experiment with alternative campaign strategies, set up your campaign as an A/B test, and analyze the results with Amazon Pinpoint analytics.

Send Transactional Messages

Keep your customers informed by sending transactional mobile push and SMS messages—such as new account activation messages, order confirmations, and password reset notifications—to specific users. You can send transactional messages by using the Amazon Pinpoint REST API.

Analyze User Behavior

Gain insights about your audience and the effectiveness of your campaigns by using the analytics that Amazon Pinpoint provides. You can view trends about your users' level of engagement, purchase activity, and demographics. You can monitor your message traffic with metrics for messages sent and opened. Through the Amazon Pinpoint API, your application can report custom data, which Amazon Pinpoint makes available for analysis.

To analyze or store the analytics data outside of Amazon Pinpoint, you can configure Amazon Pinpoint to stream the data (p. 93) to Amazon Kinesis.

Regional Availability

Amazon Pinpoint is available in the US East (N. Virginia) and EU (Ireland) AWS Regions. For a list of Amazon Pinpoint endpoint URLs, see the Amazon Pinpoint API Reference.

Integrating Amazon Pinpoint with Your Application

Integrate Amazon Pinpoint with your client code to understand and engage your users.

After you integrate, as users launch your application, it connects to the Amazon Pinpoint service to add or update *endpoints*. Endpoints represent the destinations that you can message—such as user devices, email addresses, or phone numbers.

Also, your application provides usage data, or *events*. View event data in the Amazon Pinpoint console to learn how many users you have, how often they use your application, when they use it, and more.

After your application supplies endpoints and events, you can use this information to tailor messaging campaigns for specific audiences, or *segments*. (You can also directly message simple lists of recipients without creating campaigns.)

Use the topics in this section to integrate Amazon Pinpoint with a mobile or web client. These topics include code examples and procedures to integrate with an Android, iOS, or JavaScript application.

Outside of your client, you can use supported AWS SDKs (p. 3) or the Amazon Pinpoint API to import endpoints, export event data, define customer segments, create and run campaigns, and more.

To start integrating, see the section called “Integrating the Mobile SDKs or JS Library” (p. 4).

Topics

AWS SDK Support for Amazon Pinpoint

One of the easiest ways to interact with the Amazon Pinpoint API is to use an AWS SDK. The following AWS SDKs include support for Amazon Pinpoint API operations:

- AWS Mobile SDK for Android version 2.3.5 or later
- AWS Mobile SDK for iOS version 2.4.14 or later
- AWS Amplify JavaScript Library for Web
- AWS Amplify JavaScript Library for React Native
- AWS SDK for JavaScript version 2.7.10 or later
- AWS SDK for Java version 1.11.63 or later
- AWS SDK for .NET version 3.3.27.0 or later
- AWS SDK for PHP version 3.20.1
- AWS SDK for Python (Boto) version 1.4.2 or later
- AWS SDK for Ruby version 1.0.0.rc2 or later
- AWS SDK for Go version 1.5.13 or later
- AWS SDK for C++ version 1.0.20151208.143 or later

You can also interact with the Amazon Pinpoint API by using version 1.11.24 or later of the AWS Command Line Interface (AWS CLI). The AWS CLI requires Python 2.6.5 or later, or Python 3.3 or later. For more information about installing and configuring the AWS CLI, see Installing the AWS Command Line Interface in the *AWS Command Line Interface User Guide*.

> **Note**
> The version numbers shown in this section are the first versions of each SDK or CLI that included support for the Amazon Pinpoint API. New resources or operations are occasionally added to the Amazon Pinpoint API. In order to use all of the features of Amazon Pinpoint through the API, ensure that you're using the latest version of the SDK or CLI.

Regional Availability

Amazon Pinpoint is available in the US East (N. Virginia) and EU (Ireland) AWS Regions. For a list of Amazon Pinpoint endpoint URLs, see the Amazon Pinpoint API Reference.

Integrating the AWS Mobile SDKs or JavaScript Library with Your Application

To connect to the Amazon Pinpoint service from your web or mobile application, integrate the AWS Mobile SDKs or JavaScript library with your code. With these resources, you can use the native programming language for your platform to issue requests to the Amazon Pinpoint API. For example, you can add endpoints, apply custom endpoint attributes, report usage data, and more.

For Android or iOS apps, use the AWS Mobile SDKs. For web or mobile JavaScript applications, use the AWS Amplify JavaScript library for web and React Native.

Integrating the AWS Mobile SDKs for Android or iOS

To integrate the AWS Mobile SDK for Android or iOS with your app, see Get Started (Android and iOS) in the *AWS Mobile Developer Guide*. This topic helps you:

- Create a project with AWS Mobile Hub. The **Messaging & Analytics** feature is enabled by default.
- Connect your app to the backend AWS resources that Mobile Hub provisions.
- Integrate the AWS Mobile SDK for iOS or Android with your app.

After you integrate the SDK, return to this topic in the *Amazon Pinpoint Developer Guide* for the next step.

Integrating the AWS Amplify JavaScript Library

To integrate AWS Amplify with your JavaScript application, see Get Started (Web) or Get Started (React Native) in the *AWS Mobile Developer Guide*. These topics help you:

- Use the AWS Mobile CLI to create a project. Analytics and web hosting are enabled by default.
- Create backend AWS resources for your application.
- Connect your app to the backend resources.
- Integrate the AWS Amplify library with your application.

After you integrate AWS Amplify, return to this topic in the *Amazon Pinpoint Developer Guide* for the next step.

Migrating from Amazon Mobile Analytics in the AWS Mobile SDKs

Amazon Mobile Analytics preceded Amazon Pinpoint as an AWS service that you could use to report and monitor analytics from your mobile app. Mobile Analytics features are now provided exclusively by Amazon Pinpoint. If you're currently using Mobile Analytics with the AWS Mobile SDKs, your apps will continue to report events, and this event data is shown in the Amazon Pinpoint console. However, we recommend that you switch to the Amazon Pinpoint analytics features in the latest AWS Mobile SDK or JavaScript library, which offer matching functionality. By migrating, you help ensure that you're able to use the latest features and that you receive support for any issues.

To migrate, see Migrating to Amazon Pinpoint in the AWS Mobile SDKs or JavaScript Library in the *Amazon Mobile Analytics User Guide*.

Next Step

You've integrated an AWS Mobile SDK or AWS Amplify with your application. Now you can easily add Amazon Pinpoint API requests to your code. Next, update your code to register your users' devices as endpoints. See Registering Endpoints in Your Application (p. 5).

Registering Endpoints in Your Application

When a user starts a session (for example, by launching your mobile app), your mobile or web application can automatically register (or update) an *endpoint* with Amazon Pinpoint. The endpoint represents the device that the user starts the session with. It includes attributes that describe the device, and it can also include custom attributes that you define. Endpoints can also represent other methods of communicating with customers, such as email addresses or mobile phone numbers.

After your application registers endpoints, you can segment your audience based on endpoint attributes. You can then engage these segments with tailored messaging campaigns. You can also use the **Analytics** page in the Amazon Pinpoint console to view charts about endpoint registration and activity, such as **New endpoints** and **Daily active endpoints**.

You can assign a single user ID to multiple endpoints. A user ID represents a single user, while each endpoint that is assigned the user ID represents one of the user's devices. After you assign user IDs to your endpoints, you can view charts about user activity in the console, such as **Daily active users** and **Monthly active users**.

Before You Begin

If you haven't already, integrate the AWS Mobile SDK for Android or iOS, or integrate the AWS Amplify JavaScript library with your application. See Integrating the AWS Mobile SDKs or JavaScript Library with Your Application (p. 4).

Registering Endpoints with the AWS Mobile SDKs for Android or iOS

Use the AWS Mobile SDKs for Android or iOS to register and customize endpoints.

Initializing the Amazon Pinpoint Client

To update your application to register endpoints, initialize the Amazon Pinpoint client that's provided by the AWS Mobile SDKs.

Android - Java

1. Add the Amazon Pinpoint library to the `app/build.gradle` file:

```
dependencies{
   compile 'com.amazonaws:aws-android-sdk-pinpoint:2.6.+'
}
```

2. Add the following import statements to classes that use the Amazon Pinpoint client:

```
import com.amazonaws.mobileconnectors.pinpoint.PinpointConfiguration;
import com.amazonaws.mobileconnectors.pinpoint.PinpointManager;
```

3. Create an instance of the `PinpointManager` class. The following example defines a global variable with a type of `PinpointManager`. The variable is initialized in the `onCreate()` method of the `Application` class:

```
public class Application extends MultiDexApplication {
    public static PinpointManager pinpointManager;

    @Override
    public void onCreate() {
        super.onCreate();
        awsConfiguration = new AWSConfiguration(this);

        if (IdentityManager.getDefaultIdentityManager() == null) {
            final IdentityManager identityManager = new
 IdentityManager(getApplicationContext(), awsConfiguration);
            IdentityManager.setDefaultIdentityManager(identityManager);
        }

        try {
            final PinpointConfiguration config =
                    new PinpointConfiguration(this,

 IdentityManager.getDefaultIdentityManager().getCredentialsProvider(),
                            awsConfiguration);
            Application.pinpointManager = new PinpointManager(config);
        } catch (final AmazonClientException ex) {
            Log.e(LOG_TAG, "Unable to initialize PinpointManager. " +
 ex.getMessage(), ex);
        }

}
```

iOS - Swift

1. Include the `AWSPinpoint` pod in the `Podfile` that you configure to install the AWS Mobile SDK:

```
platform :ios, '9.0'
target :'YourAppName' do
 use_frameworks!

 pod 'AWSPinpoint', '~> 2.6.6'

 # other pods

end
```

2. Add the following import statements to classes that use the Amazon Pinpoint client:

```
import AWSCore
import AWSPinpoint
```

3. Create an instance of the `AWSPinpoint` class. The following example defines a global variable with a type of `AWSPinpoint?`. The variable is initialized in the `application(_:didFinishLaunchingWithOptions:)` method of the `AppDelegate` class:

```
class AppDelegate: UIResponder, UIApplicationDelegate {

    var pinpoint: AWSPinpoint?

    func application(_ application: UIApplication, didFinishLaunchingWithOptions
 launchOptions:
    [UIApplicationLaunchOptionsKey: Any]?) -> Bool {

    // . . .

    // Initialize the Amazon Pinpoint client
    pinpoint = AWSPinpoint(configuration:
            AWSPinpointConfiguration.defaultPinpointConfiguration(launchOptions:
 launchOptions))

    // . . .
    }
}
```

iOS - Objective-C

1. Include the `AWSPinpoint` pod in the `Podfile` that you configure to install the AWS Mobile SDK:

```
platform :ios, '9.0'
target :'YourAppName' do
 use_frameworks!

 pod 'AWSPinpoint', '~> 2.6.6'

 # other pods

end
```

2. Add the following import statements to classes that use the Amazon Pinpoint client:

```
#import <AWSCore/AWSCore.h>
#import <AWSPinpoint/AWSPinpoint.h>
```

3. Create an instance of the `AWSPinpoint` class. The following example defines a global variable with a type of `AWSPinpoint`. The variable is initialized in the `application:didFinishLaunchingWithOptions:` method of the `AppDelegate` class:

```

@interface AppDelegate : UIResponder <UIApplicationDelegate>

@property(atomic) AWSPinpoint *pinpoint;

@end

@implementation AppDelegate
```

```
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {

    // . . .

    // Initialize the Amazon Pinpoint client
    AWSPinpointConfiguration *config =
     [AWSPinpointConfiguration
 defaultPinpointConfigurationWithLaunchOptions:launchOptions];
    _pinpoint = [AWSPinpoint pinpointWithConfiguration:config];

    // . . .

    return YES;
}

@end
```

Adding Custom Endpoint Attributes

After you initialize the Amazon Pinpoint client in your application, you can add custom attributes to endpoints.

Android - Java

To add custom endpoint attributes in an Android application, use the `TargetingClient` class to store your custom attributes. Then, call the `updateEndpointProfile()` method. This method generates an endpoint that has the attributes that are stored in the client, and it updates the endpoint with Amazon Pinpoint:

```
final PinpointManager pinpoint = Application.pinpointManager;
final List<String> interestsList = Arrays.asList("science", "politics", "travel");
pinpoint.getTargetingClient().addAttribute("Interests", interestsList);
pinpoint.getTargetingClient().updateEndpointProfile();
```

Alternatively, you can create and modify an instance of the `EndpointProfile` class, and you can pass the endpoint object to the `updateEndpointProfile()` call:

```
final PinpointManager pinpoint = Application.pinpointManager;
final List<String> interestsLists = Arrays.asList("science", "politics", "travel");
EndpointProfile endpointProfile = pinpoint.getTargetingClient().currentEndpoint();
endpointProfile.addAttribute("Interests", interestsLists);
pinpoint.getTargetingClient().updateEndpointProfile(endpointProfile);
```

iOS - Swift

To add custom endpoint attributes in Swift, use the `AWSPinpointTargetingClient` class. The following example updates the endpoint in the `applicationDidEnterBackground(_:)` method. The example adds a custom attribute called `interests`, and it assigns the values `science`, `politics`, and `travel`:

```
func applicationDidEnterBackground(_ application: UIApplication) {

    // . . .

    // Add a custom attribute to the endpoint
    if let targetingClient = pinpoint?.targetingClient {
        targetingClient.addAttribute(["science", "politics", "travel"], forKey:
 "interests")
        targetingClient.updateEndpointProfile()
        let endpointId = targetingClient.currentEndpointProfile().endpointId
        print("Updated custom attributes for endpoint: \(endpointId)")
    }

    // . . .

}
```

iOS - Objective-C

To add custom endpoint attributes in Objective-C, use the `AWSPinpointTargetingClient` class. The following example updates the targeting client in the `application:didFinishLaunchingWithOptions:` method. The example adds a custom attribute called `interests`, and it assigns the values `science`, `politics`, and `travel`. The example also adds a custom metric called `scienceInterestLevel` with a value of `100`:

```
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {

    // . . .

    // Add a custom attribute to the endpoint
    AWSPinpointEndpointProfile *profile = [_pinpoint.targetingClient
 currentEndpointProfile];
    [profile addAttribute:@[@"science", @"politics", @"travel"]
                   forKey:@"interests"];
    [profile addMetric:@100 forKey:@"scienceInterestLevel"];
    [[_pinpoint targetingClient] updateEndpointProfile: profile];

    // . . .

    return YES;
}
```

Assigning User IDs to Endpoints

Assign user IDs to endpoints by doing either of the following:

- Manage user sign-up and sign-in with Amazon Cognito user pools.
- Use the Amazon Pinpoint client to assign user IDs without using Amazon Cognito user pools.

Amazon Cognito user pools are user directories that make it easier to add sign-up and sign-in to your app. When the AWS Mobile SDKs for iOS and Android register an endpoint with Amazon Pinpoint, Amazon Cognito automatically assigns a user ID from the user pool. For more information, see Using Amazon Pinpoint Analytics with Amazon Cognito User Pools in the *Amazon Cognito Developer Guide*.

If you don't want to use Amazon Cognito user pools, you can use the Amazon Pinpoint client in your application to assign user IDs to endpoints.

Android - Java

To assign user IDs to endpoints in an Android application, create an instance of the `EndpointProfileUser` class, and call its `setUserId` method. Then, assign this user object to the endpoint. Finally, pass the updated endpoint to the `TargetingClient` of the `PinpointManager` class:

```
final PinpointManager pinpoint = Application.pinpointManager;
EndpointProfile endpointProfile = pinpoint.getTargetingClient().currentEndpoint();
EndpointProfileUser user = new EndpointProfileUser();
```

```
user.setUserId("UserIdValue");
endpointProfile.setUser(user);
pinpoint.getTargetingClient().updateEndpointProfile(endpointProfile);
```

iOS - Swift

To assign user IDs to endpoints with Swift, use the `AWSPinpointTargetingClient` class. First, create an `AWSPinpointEndpointProfileUser` object, and set its `userId` property. Then, assign this user object to the endpoint. Finally, pass the updated endpoint to the targeting client with the `update(_endpointProfile:)` method. The following example assigns a user ID in the `application(_:didFinishLaunchingWithOptions:)` method:

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions
    launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {

    // . . .

    if let targetingClient = pinpoint?.targetingClient {
        let endpoint = targetingClient.currentEndpointProfile()
        // Create a user and set its userId property
        let user = AWSPinpointEndpointProfileUser()
        user.userId = "UserIdValue"
        // Assign the user to the endpoint
        endpoint.user = user
        // Update the endpoint with the targeting client
        targetingClient.update(endpoint)
        print("Assigned user ID \(user.userId ?? "nil") to endpoint
 \(endpoint.endpointId)")
    }

    // . . .

    return didFinishLaunching
}
```

iOS - Objective-C

To assign user IDs with Objective-C, use the `AWSPinpointTargetingClient` class. First, create an `AWSPinpointEndpointProfileUser` object, and set its `userId` property. Then, assign this user object to the endpoint. Finally, pass the updated endpoint to the targeting client with the `updateEndpointProfile:` method. The following example assigns a user ID in the `application:didFinishLaunchingWithOptions:` method:

```
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {

    // . . .

    AWSPinpointEndpointProfile *profile = [_pinpoint.targetingClient
 currentEndpointProfile];
    // Create a user and set its userId property
    AWSPinpointEndpointProfileUser *user = [AWSPinpointEndpointProfileUser new];
    [user setUserId:@"UserIdValue"];
    // Assign the user to the endpoint
    [profile setUser:user];
    // Update the endpoint with the targeting client
    [[_pinpoint targetingClient] updateEndpointProfile: profile];

    // . . .

    return YES;
}
```

Registering Endpoints with the AWS Amplify JavaScript Library

You can enable a web or mobile JavaScript application to register endpoints by using the AWS Amplify JavaScript library for web and React Native.

Adding Custom Endpoint Attributes

After you integrate AWS Amplify with your application, you can use the `Analytics` module to add custom attributes to endpoints.

```
import { Analytics } from 'aws-amplify';

Analytics.updateEndpoint({
    Attributes: {
        interests: ['science', 'politics', 'travel']
        // ...
    },
})
```

Assigning User IDs to Endpoints

You can use the `Analytics` module to assign user IDs:

```
import { Analytics } from 'aws-amplify';

Analytics.updateEndpoint({
    // Customized userId
    UserId: 'UserIdValue,
})
```

Next Steps

You've updated your app to register endpoints. Now, as users launch your app, device information and custom attributes are provided to Amazon Pinpoint. You can use this information to define audience segments. In the console, you can see metrics about endpoints and, if applicable, users who are assigned user IDs.

Next, do one of the following:

- If you're working with an Android app, update your code to track the application lifecycle and report session events to Amazon Pinpoint. See Managing Sessions in Your Application (Android Only) (p. 11).
- For other types of applications, update your app to report usage data. See Reporting Events in Your Application (p. 16).

Managing Sessions in Your Application (Android Only)

As users engage with your app, it reports information about app sessions to Amazon Pinpoint, such as session start times, session end times, and events that occur during sessions. To report this information from an Android application, your app must include methods that handle events as your app enters the foreground and the background on the user's Android device. (For iOS applications, the AWS Mobile SDK for iOS automatically reports session information.)

Before You Begin

If you haven't already, do the following:

- Integrate the AWS Mobile SDK for Android with your app. See Integrating the AWS Mobile SDKs for Android or iOS (p. 4).
- Update your application to register endpoints. See Registering Endpoints in Your Application (p. 5).

Example Lifecycle Manager

The following example class, `AbstractApplicationLifeCycleHelper`, implements the `Application.ActivityLifecycleCallbacks` interface to track when the application enters the foreground or background, among other states. Add this class to your app, or use it as an example for how to update your code:

```
package com.amazonaws.mobile.util;

import android.app.Activity;
import android.app.Application;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;
import android.util.Log;

import java.util.WeakHashMap;

/**
 * Aids in determining when your application has entered or left the foreground.
 * The constructor registers to receive Activity lifecycle events and also registers a
 * broadcast receiver to handle the screen being turned off.  Abstract methods are
 * provided to handle when the application enters the background or foreground.
 * Any activity lifecycle callbacks can easily be overriden if additional handling
 * is needed. Just be sure to call through to the super method so that this class
 * will still behave as intended.
 **/
public abstract class AbstractApplicationLifeCycleHelper implements
 Application.ActivityLifecycleCallbacks {
    private static final String LOG_TAG =
 AbstractApplicationLifeCycleHelper.class.getSimpleName();
    private static final String ACTION_SCREEN_OFF = "android.intent.action.SCREEN_OFF";
    private boolean inForeground = false;
    /** Tracks the lifecycle of activities that have not stopped (including those
 restarted). */
    private WeakHashMap<Activity, String> activityLifecycleStateMap = new WeakHashMap<>();

    /**
     * Constructor. Registers to receive activity lifecycle events.
     * @param application The Android Application class.
     */
    public AbstractApplicationLifeCycleHelper(final Application application) {
        application.registerActivityLifecycleCallbacks(this);
        final ScreenOffReceiver screenOffReceiver = new ScreenOffReceiver();
        application.registerReceiver(screenOffReceiver, new
 IntentFilter(ACTION_SCREEN_OFF));
    }
    @Override
    public void onActivityCreated(final Activity activity, final Bundle bundle) {
        Log.d(LOG_TAG, "onActivityCreated " + activity.getLocalClassName());
        handleOnCreateOrOnStartToHandleApplicationEnteredForeground();
        activityLifecycleStateMap.put(activity, "created");
    }

    @Override
    public void onActivityStarted(final Activity activity) {
        Log.d(LOG_TAG, "onActivityStarted " + activity.getLocalClassName());
        handleOnCreateOrOnStartToHandleApplicationEnteredForeground();
        activityLifecycleStateMap.put(activity, "started");
    }

    @Override
    public void onActivityResumed(final Activity activity) {
        Log.d(LOG_TAG, "onActivityResumed " + activity.getLocalClassName());
        activityLifecycleStateMap.put(activity, "resumed");
    }

    @Override
    public void onActivityPaused(final Activity activity) {
        Log.d(LOG_TAG, "onActivityPaused " + activity.getLocalClassName());
        activityLifecycleStateMap.put(activity, "paused");
    }

    @Override
    public void onActivityStopped(final Activity activity) {
        Log.d(LOG_TAG, "onActivityStopped " + activity.getLocalClassName());
        // When the activity is stopped, we remove it from the lifecycle state map since we
        // no longer consider it keeping a session alive.
        activityLifecycleStateMap.remove(activity);
    }

    @Override
    public void onActivitySaveInstanceState(final Activity activity, final Bundle outState)
 {
        Log.d(LOG_TAG, "onActivitySaveInstanceState " + activity.getLocalClassName());
    }

    @Override
    public void onActivityDestroyed(final Activity activity) {
        Log.d(LOG_TAG, "onActivityDestroyed " + activity.getLocalClassName());
        // Activity should not be in the activityLifecycleStateMap any longer.
        if (activityLifecycleStateMap.containsKey(activity)) {
            Log.wtf(LOG_TAG, "Destroyed activity present in activityLifecycleMap!?");
            activityLifecycleStateMap.remove(activity);
        }
    }

    /**
     * Call this method when your Application trims memory.
     * @param level the level passed through from Application.onTrimMemory().
     */
    public void handleOnTrimMemory(final int level) {
        Log.d(LOG_TAG, "onTrimMemory " + level);
        // If no activities are running and the app has gone into the background.
        if (level >= Application.TRIM_MEMORY_UI_HIDDEN) {
            checkForApplicationEnteredBackground();
        }
    }

    class ScreenOffReceiver extends BroadcastReceiver {
        @Override
        public void onReceive(Context context, Intent intent) {
            checkForApplicationEnteredBackground();
        }
    }

    /**
     * Called back when your application enters the Foreground.
     */
    protected abstract void applicationEnteredForeground();

    /**
     * Called back when your application enters the Background.
     */
    protected abstract void applicationEnteredBackground();

    /**
     * Called from onActivityCreated and onActivityStarted to handle when the application
 enters
     * the foreground.
     */
    private void handleOnCreateOrOnStartToHandleApplicationEnteredForeground() {
        // if nothing is in the activity lifecycle map indicating that we are likely in the
 background, and the flag
        // indicates we are indeed in the background.
        if (activityLifecycleStateMap.size() == 0 && !inForeground) {
            inForeground = true;
            // Since this is called when an activity has started, we now know the app has
 entered the foreground.
            applicationEnteredForeground();
        }
    }

    private void checkForApplicationEnteredBackground() {
        ThreadUtils.runOnUiThread(new Runnable() {
            @Override
            public void run() {
                // If the App is in the foreground and there are no longer any activities
 that have not been stopped.
                if ((activityLifecycleStateMap.size() == 0) && inForeground) {
                    inForeground = false;
                    applicationEnteredBackground();
                }
            }
        });
    }
}
```

Reporting Session Events

After you include the `AbstractApplicationLifeCycleHelper` class, implement the two abstract methods, `applicationEnteredForeground` and `applicationEnteredBackground`, in the `Application` class. These methods enable your app to report the following information to Amazon Pinpoint:

- Session start times (when the app enters the foreground).
- Session end times (when the app enters the background).
- The events that occur during the app session, such as monetization events. This information is reported when the app enters the background.

The following example shows how to implement `applicationEnteredForeground` and `applicationEnteredBackground`. It also shows how to call `handleOnTrimMemory` from inside the `onTrimMemory` function of the `Application` class:

```
import com.amazonaws.mobileconnectors.pinpoint.PinpointConfiguration;
import com.amazonaws.mobileconnectors.pinpoint.PinpointManager;

public class Application extends MultiDexApplication {
    public static PinpointManager pinpointManager;
    private AbstractApplicationLifeCycleHelper applicationLifeCycleHelper;

    @Override
    public void onCreate() {
        super.onCreate();

        // . . .

        // The Helper registers itself to receive application lifecycle events when it is
 constructed.
        // A reference is kept here in order to pass through the onTrimMemory() call from
        // the Application class to properly track when the application enters the
 background.
        applicationLifeCycleHelper = new AbstractApplicationLifeCycleHelper(this) {
            @Override
            protected void applicationEnteredForeground() {
                Application.pinpointManager.getSessionClient().startSession();
                // handle any events that should occur when your app has come to the
 foreground...
            }

            @Override
            protected void applicationEnteredBackground() {
                Log.d(LOG_TAG, "Detected application has entered the background.");
                Application.pinpointManager.getSessionClient().stopSession();
                Application.pinpointManager.getAnalyticsClient().submitEvents();
                // handle any events that should occur when your app has gone into the
 background...
            }
        };
    }

    private void updateGCMToken() {
        try {
            final String gcmToken = InstanceID.getInstance(this).getToken(
                    "YOUR_SENDER_ID",
                    GoogleCloudMessaging.INSTANCE_ID_SCOPE
            );

 Application.pinpointManager.getNotificationClient().registerGCMDeviceToken(gcmToken);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void onTrimMemory(final int level) {
        Log.d(LOG_TAG, "onTrimMemory " + level);
        applicationLifeCycleHelper.handleOnTrimMemory(level);
        super.onTrimMemory(level);
    }

}
```

Next Step

You've updated your Android app to report session information. Now, when users open and close your app, you can see session metrics in the Amazon Pinpoint console, including those shown by the **Sessions** and **Session heat map** charts.

Next, update your app to report usage data. See Reporting Events in Your Application (p. 16).

Reporting Events in Your Application

In your mobile or web application, you can use AWS Mobile SDKs or the Amazon Pinpoint Events API to report usage data, or *events*, to Amazon Pinpoint. You can report events to capture information such as session times, users' purchasing behavior, sign-in attempts, or any custom event type that you need.

After your application reports events, you can view analytics in the Amazon Pinpoint console. The charts on the **Analytics** page provide metrics for many aspects of user behavior. For more information, see Chart Reference for Amazon Pinpoint Analytics in the *Amazon Pinpoint User Guide*.

To analyze and store your event data outside of Amazon Pinpoint, you can configure Amazon Pinpoint to stream the data to Amazon Kinesis. For more information, see Streaming Amazon Pinpoint Events to Kinesis (p. 93).

By using the AWS Mobile SDKs and the AWS Amplify JavaScript libraries, you can call the Amazon Pinpoint API to report the following types of events:

Session events

Indicate when and how often users open and close your app.

After your application reports session events, use the **Analytics** page in the Amazon Pinpoint console to view charts for **Sessions**, **Daily active endpoints**, **7-day retention rate**, and more.

Custom events

Are nonstandard events that you define by assigning a custom event type. You can add custom attributes and metrics to a custom event.

On the **Analytics** page in the console, the **Events** tab displays metrics for all custom events that are reported by your app.

Monetization events

Report the revenue that's generated by your application and the number of items that are purchased by users.

On the **Analytics** page, the **Revenue** tab displays charts for **Revenue**, **Paying users**, **Units sold**, and more.

Authentication events

Indicate how frequently users authenticate with your application.

On the **Analytics** page, the **Users** tab displays charts for **Sign-ins**, **Sign-ups**, and **Authentication failures**.

Before You Begin

If you haven't already, do the following:

- Integrate an AWS Mobile SDK or library with your app. See Integrating the AWS Mobile SDKs or JavaScript Library with Your Application (p. 4).
- Update your application to register endpoints. See Registering Endpoints in Your Application (p. 5).
- If you're integrating Amazon Pinpoint with an Android app, update your code to track the application lifecycle and report session events to Amazon Pinpoint. See Managing Sessions in Your Application (Android Only) (p. 11).

Reporting Events with the AWS Mobile SDKs for Android or iOS

You can enable a mobile app to report events to Amazon Pinpoint by using the AWS Mobile SDKs for iOS and Android.

To update your app code to record and submit events, see Add Analytics to your Mobile App with Amazon Pinpoint in the *AWS Mobile Developer Guide*.

Reporting Authentication Events From a Mobile App

To learn how frequently users authenticate with your app, update your application code so that Amazon Pinpoint receives the following standard event types for authentication:

- `_userauth.sign_in`
- `_userauth.sign_up`
- `_userauth.auth_fail`

You can report authentication events by doing either of the following:

- Managing user sign-up and sign-in with Amazon Cognito user pools.
- Reporting authentication events by using the Amazon Pinpoint client that's provided by the AWS Mobile SDK for iOS or Android.

Amazon Cognito user pools are user directories that make it easier to add sign-up and sign-in to your app. As users authenticate with your app, Amazon Cognito reports authentication events to Amazon Pinpoint. For more information, see Using Amazon Pinpoint Analytics with Amazon Cognito User Pools in the *Amazon Cognito Developer Guide*.

If you don't want to use Amazon Cognito user pools, you can use the Amazon Pinpoint client to record and submit authentication events, as shown in the following examples. In these examples, the event type is set to `_userauth.sign_in`, but you can substitute any authentication event type.

Android - Java

```
final AnalyticsEvent event =  AWSMobileClient.defaultMobileClient()
        .getPinpointManager().getAnalyticsClient().createEvent("_userauth.sign_in")
AWSMobileClient.defaultMobileClient()
        .getPinpointManager().getAnalyticsClient().recordEvent(event);
AWSMobileClient.defaultMobileClient()
        .getPinpointManager().getAnalyticsClient().submitEvents();
```

iOS - Swift

```
let pinpointAnalyticsClient = pinpoint!.analyticsClient
let event =
    pinpointAnalyticsClient.createEventWithEventType("_userauth.sign_in")

pinpointAnalyticsClient.recordEvent(event)
pinpointAnalyticsClient.submitEvents()
```

iOS - Objective-C

```
AWSPinpointEvent *event = _pinpoint.analyticsClient createEventWithEventType
    :@"_userauth.sign_in"];

_pinpoint.analyticsClient recordEvent:event];
_pinpoint.analyticsClient submitEvents];
```

Reporting Events with the AWS Amplify JavaScript Library

To record analytics in a JavaScript application, see either of the following topics in the *AWS Mobile Developer Guide*:

- Add Analytics (Web)
- Add Analytics (React Native)

Reporting Events by Using the Amazon Pinpoint API

You can use the Amazon Pinpoint API or an AWS SDK to submit events to Amazon Pinpoint in bulk. For more information, see Events in the *Amazon Pinpoint API Reference*.

Next Step

You've updated your app to report events. Now when users interact with your app, it sends usage data to Amazon Pinpoint. You can view this data in the console, and you can stream it to Amazon Kinesis.

Next, update your app to handle push notifications that you send with Amazon Pinpoint. See Handling Push Notifications (p. 18).

Handling Push Notifications

The following topics describe how to modify your iOS or Android app so that it receives push notifications that you send by using Amazon Pinpoint.

Topics

Setting Up Push Notifications for Amazon Pinpoint

Before using Amazon Pinpoint, you must add your app as a project in AWS Mobile Hub. When you add your project, you provide the credentials that authorize Amazon Pinpoint to send messages to your app through the push notification services for iOS and Android:

- For iOS apps, you provide an SSL certificate, which you obtain from the Apple Developer portal. The certificate authorizes Amazon Pinpoint to send messages to your app through Apple Push Notification service (APNs).
- For Android apps, you provide an API Key and a sender ID, which you obtain from the Firebase console or the Google API console. These credentials authorize Amazon Pinpoint to send messages to your app through Firebase Cloud Messaging or its predecessor, Google Cloud Messaging.

If you already have the credentials, you can skip this section.

Topics

Setting Up iOS Push Notifications

Push notifications for iOS apps are sent using Apple Push Notification service (APNs). Before you can send push notifications to iOS devices, you must create an app ID on the Apple Developer portal, and you must create the required certificates.

This section describes how to use the Apple Developer portal to obtain iOS and APNs credentials. These credentials enable you to create an iOS project in AWS Mobile Hub and launch a sample app that can receive push notifications.

You do not need an existing iOS app to complete the steps in this section. After you create an iOS project in Mobile Hub, you can download and launch a working sample app. Mobile Hub automatically provisions the AWS resources that your app requires.

After completing the steps in this section, you will have the following items in your Apple Developer account:

- An app ID.
- An SSL certificate, which authorizes you to send push notifications to your app through APNs.
- A registration for your test device, such as an iPhone, with your Apple Developer account.
- An iOS distribution certificate, which enables you to install your app on your test device.
- A provisioning profile, which allows your app to run on your test device.

Before you begin, you must have an account with the Apple Developer Program as an individual or as part of an organization, and you must have agent or admin privileges in that account.

The steps in this tutorial are based on the following versions of Mac OS software:

- OS X El Capitan version 10.11.6
- Xcode version 8.1

Topics

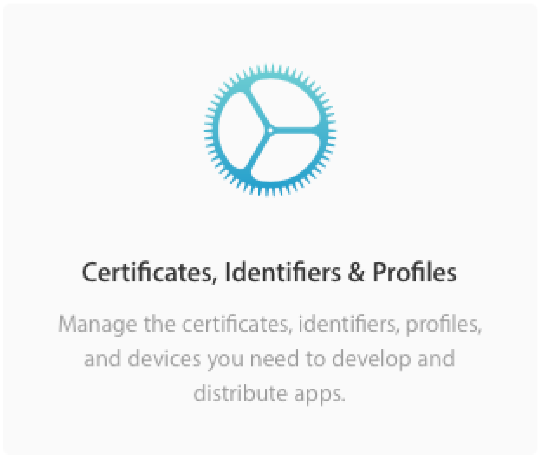
Step 1: Create an App ID

Create an app ID to identify your app in the Apple Developer portal. You need this ID when you create an SSL certificate for sending push notifications, when you create an iOS distribution certificate, and when you create a provisioning profile.

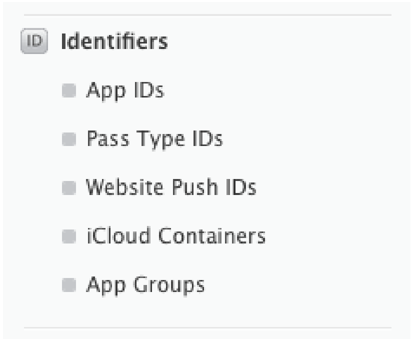
If you already have an ID assigned to your app, you can skip this step. You can use an existing app ID provided it doesn't contain a wildcard character ("*").

To assign an App ID to your app

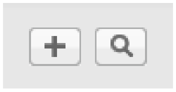
1. Sign in to your Apple Developer account at https://developer.apple.com/membercenter/index.action.
2. Choose **Certificates, Identifiers & Profiles**.


Certificates, Identifiers & Profiles
Manage the certificates, identifiers, profiles, and devices you need to develop and distribute apps.

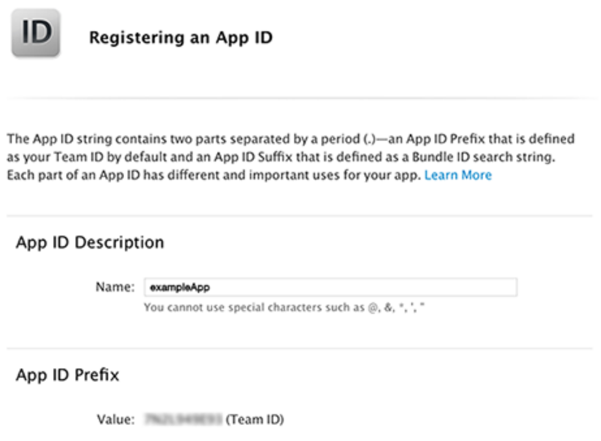
3. In the **Identifiers** section, choose **App IDs**.



4. In the **iOS App IDs** pane, choose the **Add** button (+).



5. In the **Registering an App ID** pane, for **Name**, type a custom name for your app ID that makes it easy to recognize later.



Registering an App ID

The App ID string contains two parts separated by a period (.)—an App ID Prefix that is defined as your Team ID by default and an App ID Suffix that is defined as a Bundle ID search string. Each part of an App ID has different and important uses for your app. Learn More

App ID Description

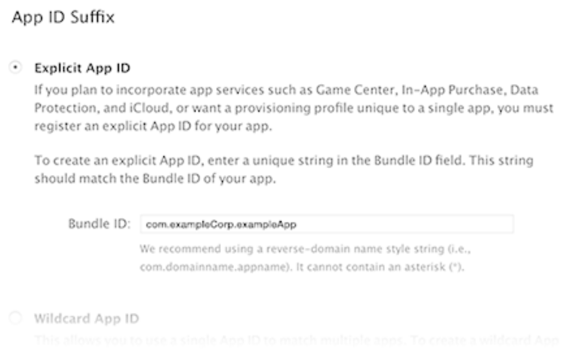
Name: exampleApp

You cannot use special characters such as @, &, *, ', "

App ID Prefix

Value: [redacted] (Team ID)

6. Choose the default selection for an App ID Prefix.
7. For **App ID Suffix**, select **Explicit App ID**, and type a bundle ID for your app. If you already have an app, use the bundle ID assigned to it. You can find this ID in the app project in Xcode on your Mac. Otherwise, take note of the bundle ID because you will assign it to your app in Xcode later.



App ID Suffix

● Explicit App ID

If you plan to incorporate app services such as Game Center, In-App Purchase, Data Protection, and iCloud, or want a provisioning profile unique to a single app, you must register an explicit App ID for your app.

To create an explicit App ID, enter a unique string in the Bundle ID field. This string should match the Bundle ID of your app.

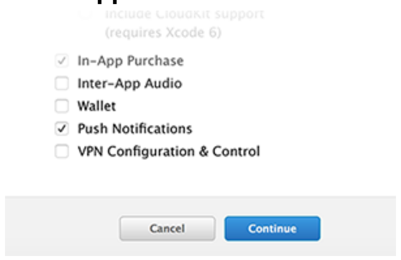
Bundle ID: com.exampleCorp.exampleApp

We recommend using a reverse-domain name style string (i.e., com.domainname.appname). It cannot contain an asterisk (*).

○ Wildcard App ID

This allows you to use a single App ID to match multiple apps. To create a wildcard App

8. Under **App Services** select **Push Notifications**.



Include CloudKit support (requires Xcode 6)

- ✓ In-App Purchase
- ☐ Inter-App Audio
- ☐ Wallet
- ✓ Push Notifications
- ☐ VPN Configuration & Control

9. Choose **Continue**. In the **Confirm your App ID** pane, check that all values were entered correctly. The identifier should match your app ID and bundle ID.
10. Choose **Register** to register the new app ID.

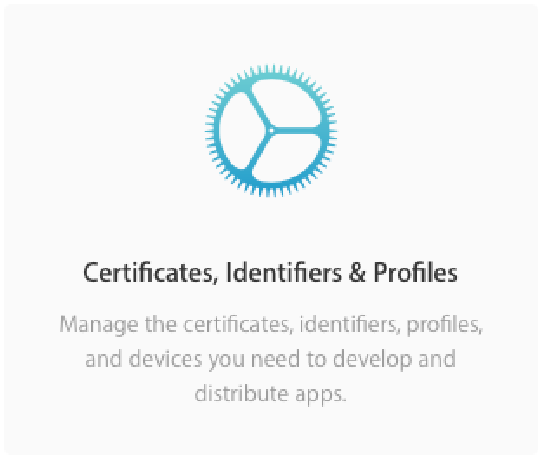
Step 2: Create an APNs SSL Certificate

Create an APNs SSL certificate so that you can deliver push notifications to your app through APNs. You will create and download the certificate from your Apple Developer account. Then, you will install the certificate in Keychain Access and export it as a .p12 file.

If you already have an SSL certificate for your app, you can skip this step.

To create an SSL certificate for push notifications

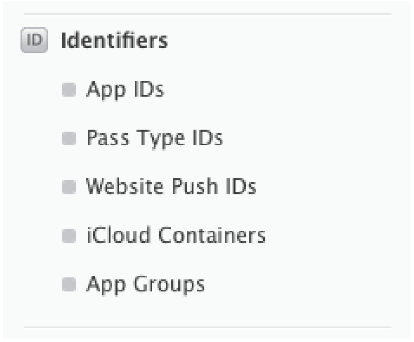
1. Sign in to your Apple Developer account at https://developer.apple.com/membercenter/index.action.
2. Choose **Certificates, Identifiers & Profiles**.



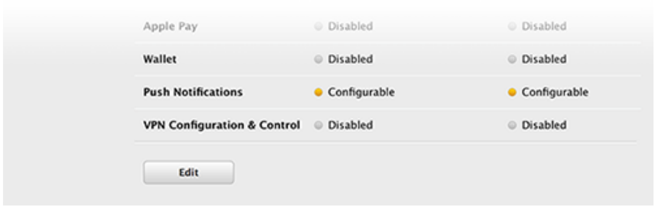
Certificates, Identifiers & Profiles

Manage the certificates, identifiers, profiles, and devices you need to develop and distribute apps.

3. From the **Identifiers** section, choose **App IDs**.

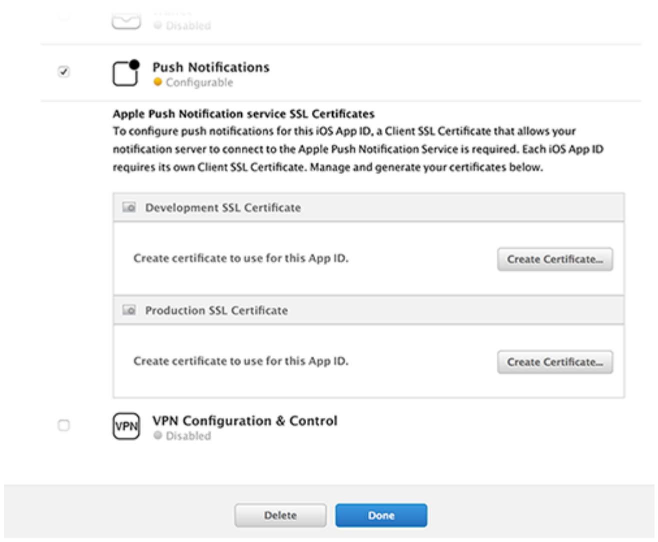


4. From the list of iOS app IDs, select the app ID that you created in Step 1: Create an App ID (p. 20).
5. Choose **Edit**.



Apple Pay	Disabled	Disabled
Wallet	Disabled	Disabled
Push Notifications	Configurable	Configurable
VPN Configuration & Control	Disabled	Disabled

6. Under **Push Notifications**, in the **Production SSL Certificate** section, choose **Create Certificate...**.


Push Notifications
Configurable

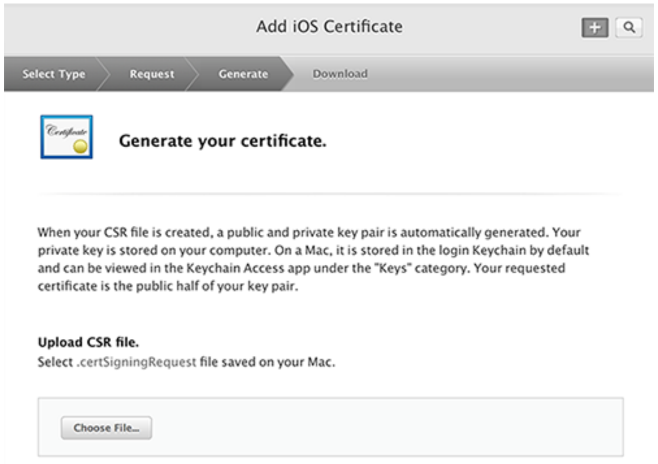
Apple Push Notification service SSL Certificates
To configure push notifications for this iOS App ID, a Client SSL Certificate that allows your notification server to connect to the Apple Push Notification Service is required. Each iOS App ID requires its own Client SSL Certificate. Manage and generate your certificates below.

Development SSL Certificate
Create certificate to use for this App ID.

Production SSL Certificate
Create certificate to use for this App ID.

VPN Configuration & Control
Disabled

7. In the **About Creating a Certificate Signing Request (CSR)** pane, follow the instructions for creating a Certificate Signing Request (CSR) file. You use the Keychain Access application on your Mac to create the request and save it on your local disk. When you are done, choose **Continue**.
8. In the **Generate your certificate** pane, choose **Choose File...**, and then select the CSR file you created.

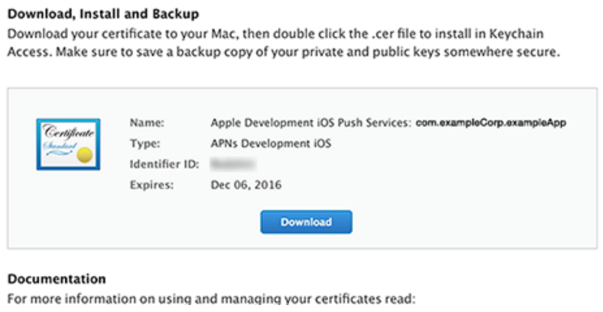

Add iOS Certificate

Generate your certificate.

When your CSR file is created, a public and private key pair is automatically generated. Your private key is stored on your computer. On a Mac, it is stored in the login Keychain by default and can be viewed in the Keychain Access app under the "Keys" category. Your requested certificate is the public half of your key pair.

Upload CSR file.
Select .certSigningRequest file saved on your Mac.

9. Choose **Continue**.
10. When the certificate is ready, choose **Download** to save the certificate to your computer.

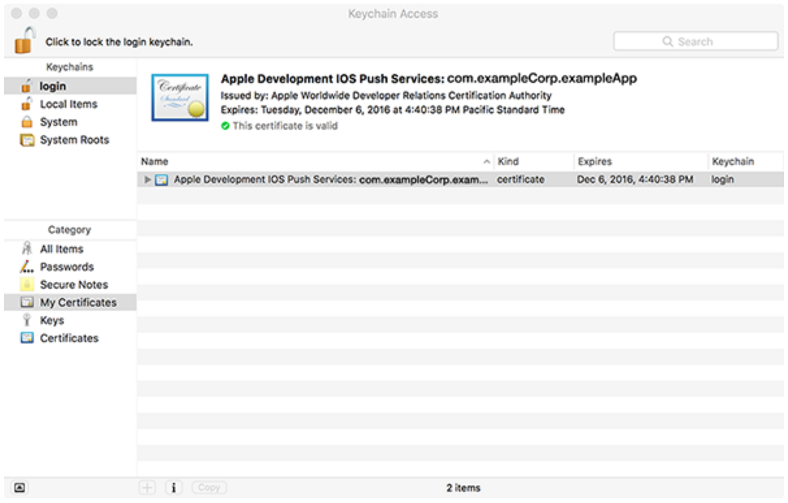

Download, Install and Backup
Download your certificate to your Mac, then double click the .cer file to install in Keychain Access. Make sure to save a backup copy of your private and public keys somewhere secure.

Name:	Apple Development iOS Push Services: com.exampleCorp.exampleApp
Type:	APNs Development iOS
Identifier ID:	[redacted]
Expires:	Dec 06, 2016

Documentation
For more information on using and managing your certificates read:

11. Double-click the downloaded certificate to install it to the Keychain on your Mac.
12. On your Mac, start the Keychain Access application.

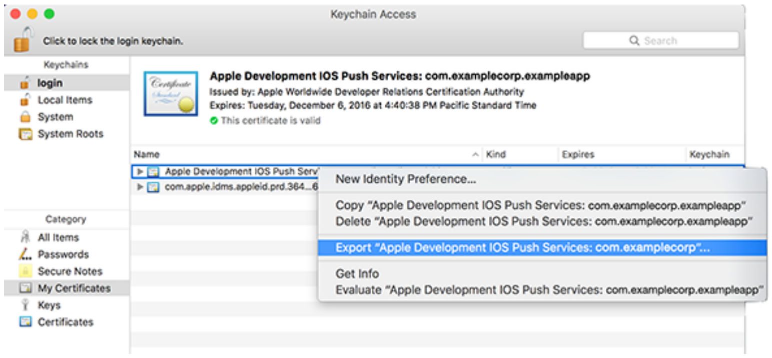
13. In **My Certificates**, find the certificate you just added. The certificate is named "Apple Push Services:`com.my.app.id`", where com.my.app.id is the app ID for which the certificate was created.



Apple Development IOS Push Services: com.exampleCorp.exampleApp
Issued by: Apple Worldwide Developer Relations Certification Authority
Expires: Tuesday, December 6, 2016 at 4:40:38 PM Pacific Standard Time
This certificate is valid

Name	Kind	Expires	Keychain
Apple Development IOS Push Services: com.exampleCorp.exam...	certificate	Dec 6, 2016, 4:40:38 PM	login

14. Context-select the push certificate and then select **Export...** from the context menu to export a file containing the certificate.


Keychain Access
Click to lock the login keychain.
Search
Keychains
login
Local Items
System
System Roots
Category
All Items
Passwords
Secure Notes
My Certificates
Keys
Certificates
Name
Kind
Expires
Keychain
Apple Development IOS Push Serv
com.apple.idms.appleid.prd.364...6
New Identity Preference...
Copy "Apple Development IOS Push Services: com.examplecorp.exampleapp"
Delete "Apple Development IOS Push Services: com.examplecorp.exampleapp"
Export "Apple Development IOS Push Services: com.examplecorp"...
Get Info
Evaluate "Apple Development IOS Push Services: com.examplecorp.exampleapp"

Apple Development IOS Push Services: com.examplecorp.exampleapp
Issued by: Apple Worldwide Developer Relations Certification Authority
Expires: Tuesday, December 6, 2016 at 4:40:38 PM Pacific Standard Time
This certificate is valid

15. Type a name for the certificate that is easy to recognize and save it to your computer. Do not provide an export password when prompted. You need to upload this certificate when creating your app in AWS Mobile Hub.

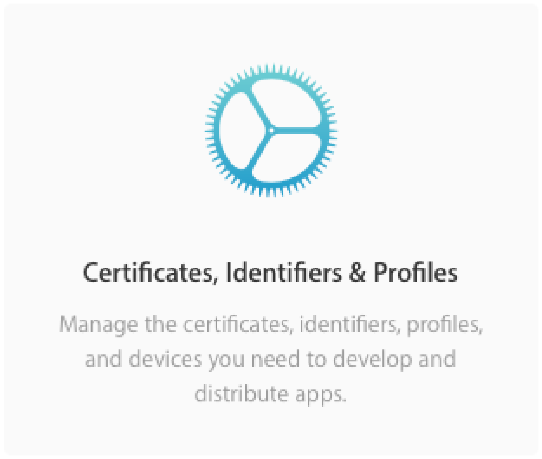
Step 3: Register a Test Device

Register a test device with your Apple Developer account so that you can test your app on that device. Later, you associate this test device with your provisioning profile, which allows your app to launch on your device.

If you already have a registered device, you can skip this step.

To add a device

1. Sign in to your Apple Developer account at https://developer.apple.com/membercenter/index.action.
2. Choose **Certificates, Identifiers & Profiles**.


Certificates, Identifiers & Profiles
Manage the certificates, identifiers, profiles, and devices you need to develop and distribute apps.

3. In the **Devices** section, choose the type of device that you want to add, such **iPhone**.
4. Choose the **Add** button (+).
5. In the **Register Device** section, for **Name**, type a name that is easy to recognize later.
6. For **UDID**, type the unique device ID. For an iPhone, you can find the UDID by completing the following steps:

    a. Connect your iPhone to your Mac with a USB cable.
    b. Open the iTunes app.
    c. In the top left corner of the iTunes window, a button with an iPhone icon is shown. Choose this button. iTunes displays the summary page for your iPhone.
    d. In the top box, the summary page provides your iPhone's **Capacity**, **Phone Number**, and **Serial Number**. Click **Serial Number**, and the value changes to **UDID**.
    e. Context-select your UDID, and choose **Copy**.
    f. Paste your UDID into the **UDID** field in the Apple Developer website.
7. Choose **Continue**.
8. On the **Review and register** pane, verify the details for your device, and choose **Register**. Your device name and identifier are added to the list of devices.

Step 4: Create an iOS Distribution Certificate

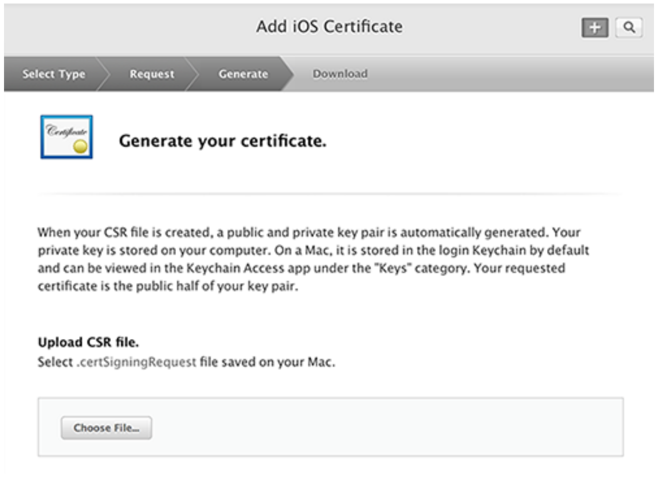
An iOS distribution certificate enables you to install your app on a test device and deliver push notifications to that device. You specify your iOS distribution certificate later when you create a provisioning profile for your app.

If you already have an iOS distribution certificate, you can skip this step.

To create an iOS distribution certificate

1. On the **Certificates, Identifiers & Profiles** page of your Apple Developer account, in the **Certificates section**, choose **Production**.
2. In the **iOS Certificates (Production)** pane, choose the Add button (+). The **Add iOS Certificate** pane opens.
3. In the **Production** section, select **App Store and Ad Hoc**, and then choose **Continue**.
4. On the **About Creating a Certificate Signing Request (CSR)** page, choose **Continue**.
5. In the **About Creating a Certificate Signing Request (CSR)** pane, follow the instructions for creating a Certificate Signing Request (CSR) file. You use the Keychain Access application on your Mac to create the request and save it on your local disk. When you are done, choose **Continue**.

6. In the **Generate your certificate** pane, choose **Choose File...**, and then select the CSR file you created.



Generate your certificate.

When your CSR file is created, a public and private key pair is automatically generated. Your private key is stored on your computer. On a Mac, it is stored in the login Keychain by default and can be viewed in the Keychain Access app under the "Keys" category. Your requested certificate is the public half of your key pair.

Upload CSR file.
Select .certSigningRequest file saved on your Mac.

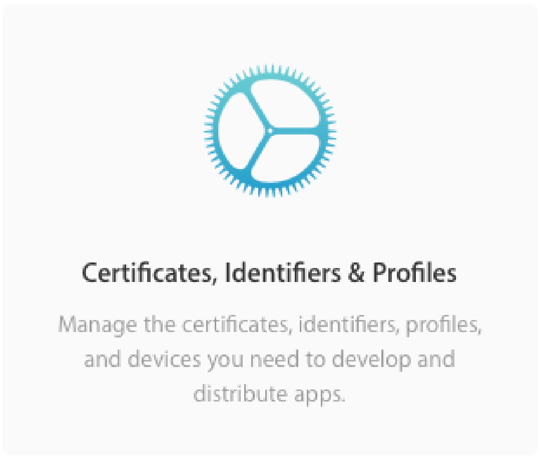
7. Choose **Continue**.
8. When the certificate is ready, choose **Download** to save the certificate to your computer.
9. Double-click the downloaded certificate to install it in Keychain on your Mac.

Step 5: Create a Provisioning Profile

A provisioning profile allows your app to run on your test device. You create and download a provisioning profile from your Apple Developer account and then install the provisioning profile in Xcode.

To create a provisioning profile

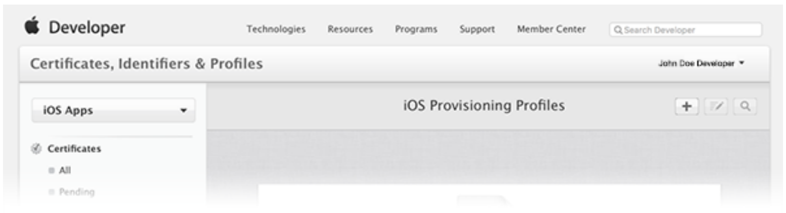
1. Sign in to your Apple Developer account at https://developer.apple.com/membercenter/index.action.
2. Select **Certificates, Identifiers & Profiles**.



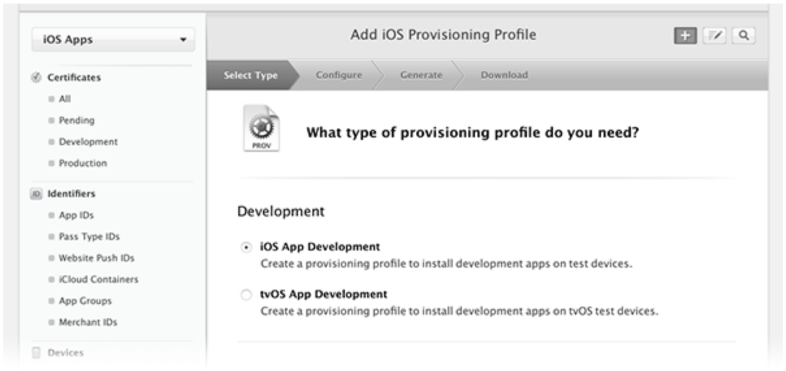
Certificates, Identifiers & Profiles

Manage the certificates, identifiers, profiles, and devices you need to develop and distribute apps.

3. In the **Provisioning Profiles** section, choose **Distribution**.
4. In the **iOS Provisioning Profiles (Distribution)** pane, choose the add button (+). The **Add iOS Provisioning Profiles (Distribution)** pane is shown.



5. In the **Distribution** section, select **Ad Hoc**, and then choose **Continue**.



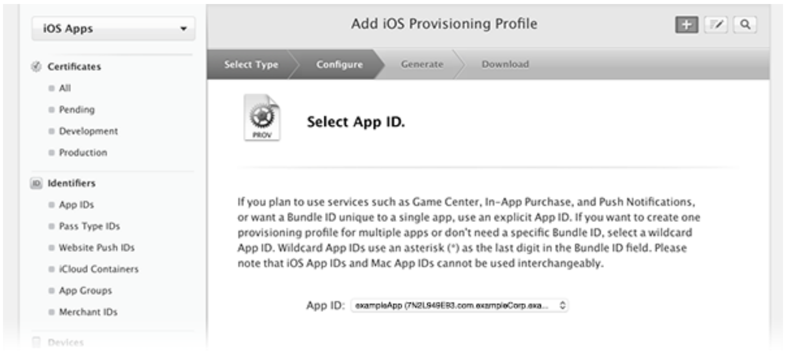
What type of provisioning profile do you need?

Development

● iOS App Development
Create a provisioning profile to install development apps on test devices.

○ tvOS App Development
Create a provisioning profile to install development apps on tvOS test devices.

6. For **App ID**, select the app ID you created for your app, and then choose **Continue**.



Select App ID.

If you plan to use services such as Game Center, In-App Purchase, and Push Notifications, or want a Bundle ID unique to a single app, use an explicit App ID. If you want to create one provisioning profile for multiple apps or don't need a specific Bundle ID, select a wildcard App ID. Wildcard App IDs use an asterisk (*) as the last digit in the Bundle ID field. Please note that iOS App IDs and Mac App IDs cannot be used interchangeably.

App ID: exampleApp (7N2L949EB3.com.exampleCorp.exa...

7. Select your iOS Development certificate and then choose **Continue**.
8. For **Select devices**, select the device that you registered for testing, and choose **Continue**.
9. Type a name for this provisioning profile, such as `ApnsDistributionProfile`, and choose **Continue**.
10. Select **Download** to download the generated provisioning profile.
11. Install the provisioning profile by double-clicking the downloaded file. The Xcode app opens in response.
12. To verify that the provisioning profile is installed, check the list of installed provisioning profiles in Xcode by doing the following:

    a. In the Xcode menu bar, choose **Xcode** and then choose **Preferences**.
    b. In the preferences window, choose **Accounts**.
    c. In the **Accounts** tab, select your Apple ID, and then choose **View Details**.
    d. Check that your provisioning profile is listed in the **Provisioning Profiles** section.

Setting Up Android Push Notifications

This section describes how to obtain the credentials required to send push notifications to Android apps. The platform notification services you can use for push notifications on Android are Firebase Cloud

Messaging (FCM) and its predecessor, Google Cloud Messaging (GCM). Your FCM or GCM credentials enable you to create an Android project in AWS Mobile Hub and launch a sample app that can receive push notifications.

You do not need an existing Android app to complete the steps in this section. After you create an Android project in Mobile Hub, you can download and launch a working sample app. Mobile Hub automatically provisions the AWS resources that your app requires.

Topics

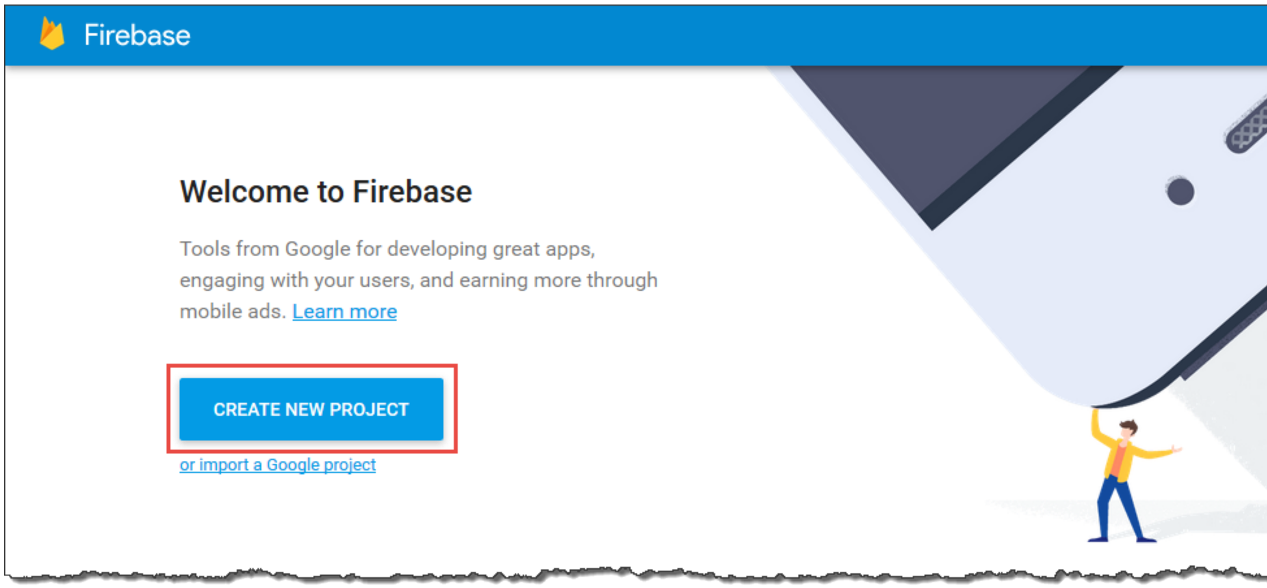
Step 1: Create a Firebase Project

To send push notifications to Android apps, you must have a project that is enabled with an Android push notification service. The push notification services for Android are Firebase Cloud Messaging (FCM) and its predecessor, Google Cloud Messaging (GCM).

If you are new to push messaging on Android, you must create a Firebase project, as this topic describes. However, if you have an existing Google Cloud Messaging project that has push messaging enabled, you can skip this step and use that project instead.

To create a Firebase project

1. Go to the Firebase console at https://console.firebase.google.com/. If you are not signed in to Google, the link takes you to a sign-in page. After you sign in, you see the Firebase console.
2. Choose **Create New Project**.


Firebase
Welcome to Firebase
Tools from Google for developing great apps, engaging with your users, and earning more through mobile ads. Learn more
CREATE NEW PROJECT
or import a Google project

3. Type a project name, and then choose **Create Project**.

   Firebase projects support push messaging by default.

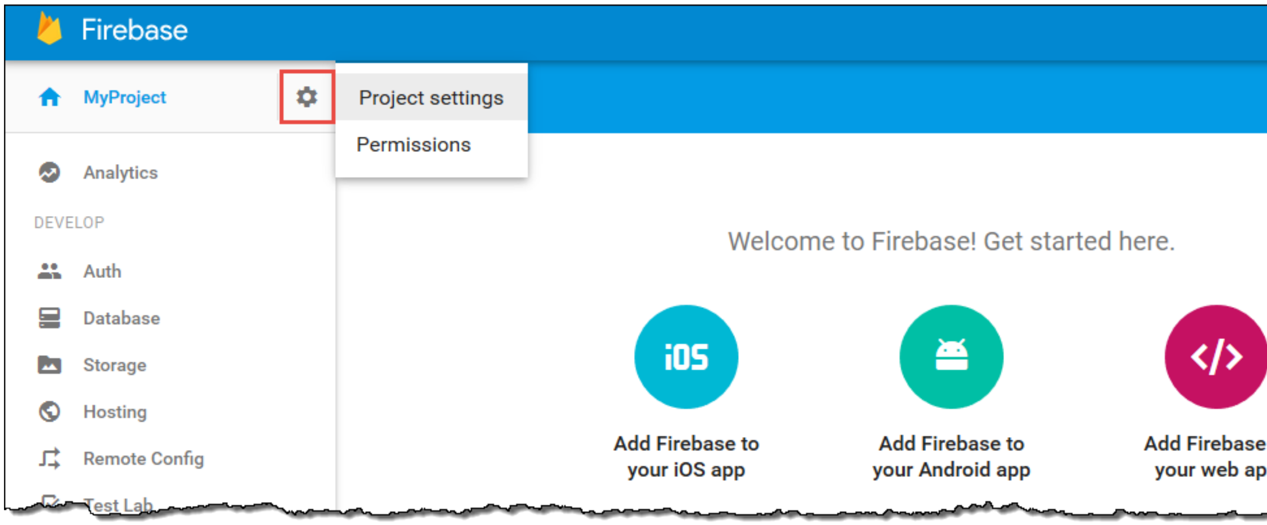
Step 2: Get Push Messaging Credentials for Android

To send push notifications to Android apps, you must have credentials from either Firebase Cloud Messaging (FCM) or its predecessor, Google Cloud Messaging (GCM). The credentials are an API key and a sender ID (also called project number). You get these credentials from a project that has push messaging enabled. This project could either be in the Firebase console or the Google Cloud Platform console, depending on where you created it.

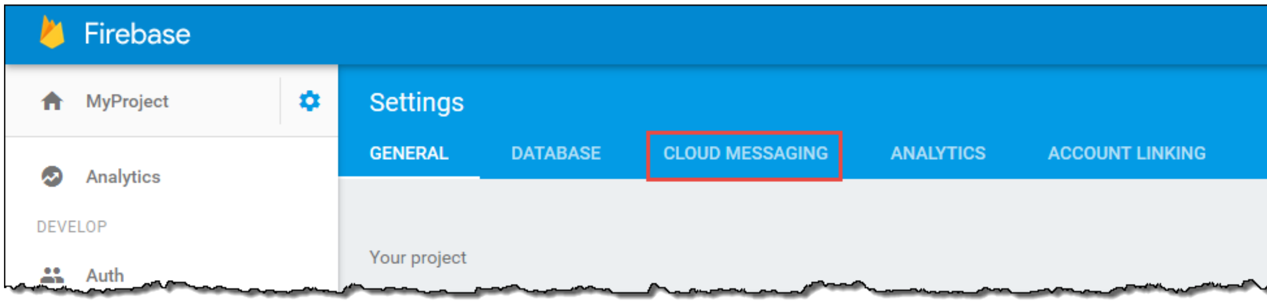
This topic describes how to retrieve your credentials from FCM or GCM. Use FCM for new Android apps. Use GCM only if you have a preexisting GCM project that you have not yet updated for FCM support.

To obtain your credentials from FCM

1. Go to the Firebase console at https://console.firebase.google.com/ and open your project.
2. In the left pane, to the right of your project name, choose the gear icon, and then choose **Project Settings**.



3. In the top menu, choose **Cloud Messaging**.



4. Under **Project credentials**, you find the API key and sender ID. Save these values somewhere you can access later.

To obtain your credentials from GCM

1. Go to the Google API Console at https://console.developers.google.com.
2. In the left pane, choose **Credentials**.
3. If you already have credentials for your app, your server key is shown in the **API keys** section. Save this key somewhere you can access later.
4. If you don't have credentials for your app, the console displays the **Credentials** dialog box. Create a server key by completing the following steps:

    a. Choose **Create credentials**.

   b. Save the API key somewhere you can access later.
5. To retrieve your sender ID (also called project number), go to the Google Cloud Platform console at https://console.cloud.google.com/. Select your project from the **Project** menu. Then, choose the arrow next to the project name.
6. Save the displayed project number somewhere you can access later.

   Note
   Project number is another name for sender ID.

Handling Push Notifications

The following topics describe how to modify your iOS app so that it receives push notifications sent by Amazon Pinpoint.

Topics

Handling Push Notifications from Apple Push Notification Service

You can enable your iOS app to receive push notifications that you send by using Amazon Pinpoint. With Amazon Pinpoint, you can send push notifications to iOS apps through Apple Push Notification service (APNs).

> **Prerequisite**
> Before you update your app to receive push notifications, integrate the AWS Mobile SDK for iOS. For more information, see Integrating the AWS Mobile SDKs for Android or iOS (p. 4).

Enabling Push Notifications

To enable push notifications in your app, update your app to include push listening code. For more information, see Add Push Notifications to Your Mobile App with Amazon Pinpoint in the *AWS Mobile Developer Guide*.

Setting Up Deep Linking

Amazon Pinpoint campaigns can take one of three actions when a user taps a notification. One of those possible actions is a deep link, which opens the app to a specified activity.

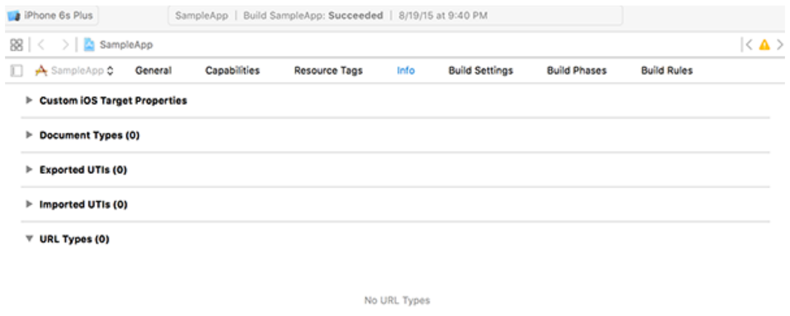
Registering a Custom URL Scheme

To specify a destination activity for deep links, the app must have set up deep linking. This setup requires registering a custom URL scheme the deep links will use. To register a custom URL identifier, go to your Xcode project's target **Info** tab and expand the **URL Types** section.

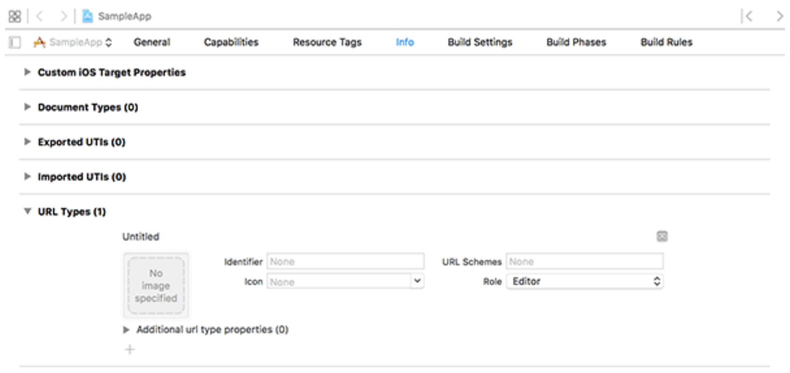
To open the app via a `pinpoint://` URL scheme you need to assign a unique identifier to the scheme. Apple recommends reverse DNS notation to avoid name collisions on the platform. The following example uses `com.exampleCorp.exampleApp`:

To register a custom URL scheme in Xcode:

1. In Xcode, select the **Info** tab.



2. In the **URL Types** section, select + to add a URL type.



3. Enter the reverse DNS notation identifier for this URL type in **Identifier**.
4. Enter the URL you want to use for your app in **URL Schemes**.
5. Save the project.

After this custom URL scheme is registered, test it in the iOS simulator. Open Safari and navigate to your custom URL; in this example, `pinpoint://`. Your app launches and opens to its home screen.

Listening for Custom URLs

To direct the app to a specific view, implement a callback in your `AppDelegate` that is called when your app launches from a deep link. The scheme launches the app, using the host and path to go to a separate screen within the app.

Handling Push Notifications from Firebase Cloud Messaging or Google Cloud Messaging

You can enable your Android or iOS app to receive push notifications that you send through by using Amazon Pinpoint. With Amazon Pinpoint, you can send push notifications through Firebase Cloud Messaging (FCM) or its predecessor, Google Cloud Messaging (GCM).

Prerequisite
Before you update your app to receive push notifications, integrate the AWS Mobile SDK for Android. For more information, see Integrating the AWS Mobile SDKs for Android or iOS (p. 4).

Enabling Push Notifications

To enable push notifications in your app, update your app to include push listening code. For more information, see Add Push Notifications to Your Mobile App with Amazon Pinpoint in the *AWS Mobile Developer Guide*.

Setting up Deep Linking

Amazon Pinpoint campaigns can take one of three actions when a user taps a notification. One of those possible actions is a deep link, which opens the app to a specified activity.

To specify a destination activity for deep links, the app must have set up deep linking. This setup requires an intent filter that registers a URL scheme the deep links will use. After the app creates an intent filter, the data provided by the intent determines the activity to render.

Creating an Intent Filter

Begin to set up deep linking by creating an intent filter in your `AndroidManifest.xml` file. For example:

```
<!-- This activity allows your application to receive a deep link that navigates directly
 to the
"Deeplink Page"-->
<activity
    android:name=".DeepLinkActivity"
    android:label="A deeplink!" >
    <intent-filter android:label="inAppReceiver">
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <category android:name="android.intent.category.BROWSABLE" />
        <!-- Accepts URIs of type "pinpoint://deeplink" -->
        <data android:scheme="pinpoint"
            android:host="deeplink" />
    </intent-filter>
</activity>
```

The data element in the previous example registers a URL scheme, `pinpoint://`, as well as the host, `deeplink`. As a result, when given a URL in the form of `pinpoint://deeplink`, the manifest is prepared to execute the action.

Handling the Intent

Next, set up an intent handler to present the screen associated with the registered URL scheme and host. Intent data is retrieved in the `onCreate()` method, which then can use `Uri data` to create an activity. The following example shows an alert and tracks an event.

```
public class DeeplinkActivity extends Activity {

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        if (getIntent().getAction() == Intent.ACTION_VIEW) {
            Uri data = getIntent().getData();

            if (data != null) {

                // show an alert with the "custom" param
                new AlertDialog.Builder(this)
                        .setTitle("An example of a Deeplink")
                        .setMessage("Found custom param: "
 +data.getQueryParameter("custom"))
                        .setPositiveButton(android.R.string.yes, new
 DialogInterface.OnClickListener() {
                            public void onClick(DialogInterface dialog, int which) {
                                dialog.dismiss();
                            }
                        })
                        .setIcon(android.R.drawable.ic_dialog_alert)
```

```
                            .show();
                }
            }
        }
}
```

Handling Push Notifications from Amazon Device Messaging

Amazon Device Messaging (ADM) is a service used to send push notifications to apps running on Amazon devices, such as Kindle Fire tablets. By integrating ADM with your app, you can use Amazon Pinpoint to send notifications to your app through the ADM mobile push channel.

Prerequisites

To send push notifications to your app using Amazon Pinpoint and ADM, you need the following:

- Amazon Developer account.
- Client ID and client secret from Amazon Device Messaging.
- ADM registration ID (provided by the end device that contains the ADM platform).

You must also integrate the AWS Mobile SDK for Android before you begin. For more information, see Integrating the AWS Mobile SDKs or JavaScript Library with Your Application (p. 4).

Integrating ADM with Your App

If you are already familiar with ADM and have ADM credentials, you can follow the steps for Integrating Your App with Amazon Device Messaging in the Amazon Developer documentation. Otherwise, for an introduction to ADM, see Understanding Amazon Device Messaging.

To integrate with Amazon Pinpoint, your subclass implementation of `com.amazon.device.messaging.ADMMessageHandlerBase` should include the following methods and perform the corresponding calls:

`onRegistered`

Called when the device is registered with the ADM service. Provides the ADM registration ID that is needed to register the device with Amazon Pinpoint. Include the following call as part of this method:

```
pinpointManager.getNotificationClient().registerDeviceToken(registrationId)
```

`onUnregistered`

Called when the device is no longer registered with the ADM service.

`onMessage`

Called when the device receives a message notification from ADM. Include the following as part of this method:

```
NotificaitonDetails details = NotificationDetailsBuilder.builder()
                                .intent(intent);
                                .intentAction(NotificationClient.ADM_INTENT_ACTION)
                                .build();

pinpointManager.getNotificationClient().handleCampaignPush(details)
```

Testing ADM Push Notifications

To test, you need an Amazon Pinpoint project, an ADM client ID, and an ADM client secret.

Before you begin, augment your app to display the device token after registration. The device token can be retrieved by calling:

```
pinpointManager.getNotificationClient().getDeviceToken()
```

Complete the following steps using the AWS CLI.

To test ADM push notifications

1. Register ADM as a channel with your Amazon Pinpoint project. Provide the ADM client ID and the ADM client secret.

```
aws pinpoint update-adm-channel --application-id [YourPinpointAppId] --adm-channel-
request "{
    \"ClientId\": \"ADM_CLIENT_ID",
    \"Enabled\": true,
    \"ClientSecret\": \"ADM_CLIENT_SECRET"
}"
```

2. Install your app on a device that has ADM enabled, and capture the generated device token.
3. Send a direct message to the device specifying the device token as the address.

```
aws pinpoint send-messages --application-id YourPinpointAppId --message-request "{
  \"Addresses\": {
      \"DeviceToken\": {
          \"ChannelType\": \"ADM\"
      }
  },
  \"MessageConfiguration\": {
      \"ADMMessage\": {
          \"RawContent\":
\"{'pinpoint.campaign.campaign_id':'_DIRECT','pinpoint.notification.silentPush':0,'pinpoint.openApp
 World.'}\"
      }
  }
}"
```

Handling Push Notifications from Baidu Cloud Push

Baidu Cloud Push is the push notification service provided by Baidu, a Chinese cloud service. By integrating Baidu Cloud Push with your mobile app, you can use Amazon Pinpoint to send notifications to your app through the Baidu mobile push channel.

Prerequisites

To send push notifications to mobile devices using Amazon Pinpoint and Baidu, you need the following:

- Baidu account.
- Registration as a Baidu developer.
- Baidu Cloud Push project.
- API key and secret key from a Baidu Cloud Push project.
- Baidu user ID and channel ID.

You must also integrate the AWS Mobile SDK for Android before you begin. For more information, see Integrating the AWS Mobile SDKs or JavaScript Library with Your Application (p. 4).

Integrating Baidu Cloud Push with Your App

The following procedure is based on version 5.7.1.65 of the Baidu push service jar.

To integrate Baidu with your app

1. Download the latest Baidu Cloud Push SDK Android client from http://push.baidu.com/.
2. Extract the zip file and import the `pushservice-x.x.xx.jar` file from the Baidu-Push-SDK-Android `libs` folder into your Android app's `lib` folder.
3. The Baidu-Push-SDK-Android `libs` folder should also include the following folders:

   - `arm64-v8a/`
   - `armeabi/`
   - `armeabi-v7a/`
   - `mips/`
   - `mips64/`
   - `x86/`
   - `x86_64/`

   Add each complete folder to your Android app's `src/main/jniLibs` folder.
4. In the Android app's `AndroidManifest.xml` file, declare the following permissions:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />

<!-- Baidu permissions -->
<uses-permission android:name="android.permission.WAKE_LOCK"/>
<uses-permission android:name="com.google.android.c2dm.permission.RECEIVE" />

<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
<uses-permission android:name="android.permission.WRITE_SETTINGS" />
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.ACCESS_DOWNLOAD_MANAGER" />
<uses-permission android:name="android.permission.DOWNLOAD_WITHOUT_NOTIFICATION" />
<uses-permission android:name="android.permission.DISABLE_KEYGUARD" />
```

5. Under `<application>`, specify the following receivers and intent filters:

```
<!-- Baidu settings -->
<receiver android:name="com.baidu.android.pushservice.PushServiceReceiver"
          android:process=":bdservice_v1">
  <intent-filter>
    <action android:name="android.intent.action.BOOT_COMPLETED" />
    <action android:name="android.net.conn.CONNECTIVITY_CHANGE" />
    <action android:name="com.baidu.android.pushservice.action.notification.SHOW" />
    <action android:name="com.baidu.android.pushservice.action.media.CLICK" />
    <!-- ########action########service########## -->
    <action android:name="android.intent.action.MEDIA_MOUNTED" />
    <action android:name="android.intent.action.USER_PRESENT" />
    <action android:name="android.intent.action.ACTION_POWER_CONNECTED" />
    <action android:name="android.intent.action.ACTION_POWER_DISCONNECTED" />
  </intent-filter>
</receiver>
<receiver
          android:name="com.baidu.android.pushservice.RegistrationReceiver"
          android:process=":bdservice_v1">
  <intent-filter>
    <action android:name="com.baidu.android.pushservice.action.METHOD" />
    <action android:name="com.baidu.android.pushservice.action.BIND_SYNC" />
  </intent-filter>
  <intent-filter>
    <action android:name="android.intent.action.PACKAGE_REMOVED" />

    <data android:scheme="package" />
  </intent-filter>
</receiver>

<service
         android:name="com.baidu.android.pushservice.PushService"
         android:exported="true"
         android:process=":bdservice_v1">
  <intent-filter>
    <action android:name="com.baidu.android.pushservice.action.PUSH_SERVICE" />
  </intent-filter>
</service>
<service
         android:name="com.baidu.android.pushservice.CommandService"
         android:exported="true" />
<!-- Amazon Pinpoint Notification Receiver -->
<receiver
 android:name="com.amazonaws.mobileconnectors.pinpoint.targeting.notification.PinpointNotificationR
  <intent-filter>
  <action android:name="com.amazonaws.intent.baidu.NOTIFICATION_OPEN" />
  </intent-filter>
</receiver>
```

6. Update the `AndroidManifest.xml` file with the following permissions, which are specific to your application. Remember to replace *`YourPackageName`* with the name of your package.

```
<!-- ##Android N#####ContentProvider##############-->
<uses-permission
 android:name="baidu.push.permission.WRITE_PUSHINFOPROVIDER.YourPackageName" />
<permission
            android:name="baidu.push.permission.WRITE_PUSHINFOPROVIDER.YourPackageName"
            android:protectionLevel="normal"></permission>
 <!-- ##Android N#####ContentProvider###########-->
<provider
          android:name="com.baidu.android.pushservice.PushInfoProvider"
          android:authorities="YourPackageName.bdpush"
          android:exported="true"
          android:protectionLevel="signature"

 android:writePermission="baidu.push.permission.WRITE_PUSHINFOPROVIDER.YourPackageName" /
>
```

7. Inside your Android application, create a `MessageReceiver` class that subclasses `com.baidu.android.pushservice.PushMessageReceiver`. The subclass should implement the following methods and perform the corresponding calls:

   `onBind`

      Called when the device is registered with Baidu Cloud Push. Provides the Baidu user ID and channel ID that are needed to register the device with Amazon Pinpoint. Include the following call as part of this method:

```
pinpointManager.getNotificationClient().registerDeviceToken(registrationId)
```

`onUnbind`

Called when the device is no longer registered with Baidu Cloud Push.

`onMessage`

Called when the device receives a raw message from Baidu Cloud Push. Amazon Pinpoint transmits campaign push notifications with the Baidu Cloud Push raw message format. Include the following call as part of this method:

```
NotificaitonDetails details = NotificationDetailsBuilder.builder()
                              .message(message);
                              .intentAction(NotificationClient.BAIDU_INTENT_ACTION)
                              .build();

pinpointManager.getNotificationClient().handleCampaignPush(details)
```

Only the message parameter contains data. The `customContentString` is not used with raw messages.

8. After creating the subclass, modify the `AndoriodManifest.xml` file to register it as a receiver. In the following example, the `PushMessageReceiver` subclass is named `com.baidu.push.example.MyPushMessageReceiver`.

```
<!-- push######receiver## -->
<receiver android:name="com.baidu.push.example.MyPushMessageReceiver">
  <intent-filter>
    <!-- ##push## -->
    <action android:name="com.baidu.android.pushservice.action.MESSAGE" />
    <!-- ##bind,unbind,fetch,delete##### -->
    <action android:name="com.baidu.android.pushservice.action.RECEIVE" />
    <action android:name="com.baidu.android.pushservice.action.notification.CLICK" />
  </intent-filter>
</receiver>
```

9. To start the Baidu listener service, in your Android app's main activity, add the following code to the `onCreate` method:

```
// Push: #apikey###########Activity#onCreate##
// ## ATTENTION#You need to modify the value of api_key to your own !!
// ####push
PushManager.startWork(getApplicationContext(), PushConstants.LOGIN_TYPE_API_KEY,
 api_key);

// Push: ############API#########################
// ##############->#####->##############1#
// ###### PushManager.setNotificationBuilder(this, 1, cBuilder)#########
CustomPushNotificationBuilder cBuilder = new CustomPushNotificationBuilder(
  getResources().getIdentifier("notification_custom_builder", "layout",
 getPackageName()),
  getResources().getIdentifier("notification_icon", "id", getPackageName()),
  getResources().getIdentifier("notification_title", "id", getPackageName()),
  getResources().getIdentifier("notification_text", "id", getPackageName()));
cBuilder.setNotificationFlags(Notification.FLAG_AUTO_CANCEL);
cBuilder.setNotificationDefaults(Notification.DEFAULT_VIBRATE);

cBuilder.setStatusbarIcon(this.getApplicationInfo().icon);
cBuilder.setLayoutDrawable(getResources().getIdentifier(
  "simple_notification_icon", "drawable", getPackageName()));
cBuilder.setNotificationSound(Uri.withAppendedPath(
  Audio.Media.INTERNAL_CONTENT_URI, "6").toString());
// ###################ID
PushManager.setNotificationBuilder(this, 1, cBuilder);
```

10. Remember to properly initialize your `PinpointManager` reference. Use a `PinpointConfiguration` with a `ChannelType` value of `ChannelType.BAIDU`. You can do this programmatically, as in the following example:

    ```
    final PinpointConfiguration config =
      new PinpointConfiguration(this,
                                IdentityManager.getDefaultIdentityManager()
                                .getCredentialsProvider(),
                                awsConfiguration)
      .withChannelType(ChannelType.BAIDU);
    Application.pinpointManager = new PinpointManager(config);
    ```

    Or, you can define a configuration file to be consumed by `AWSConfiguration`:

    ```
    "PinpointAnalytics": {
        "Default": {
          "AppId": "[YourPinpointAppId]",
          "Region": "us-east-1",
          "ChannelType": "BAIDU"
        }
      }
    ```

Testing Baidu Push Notifications

To test, you need an Amazon Pinpoint project, a Baidu API key, and a Baidu Secret key.

Before you begin, augment your app to display the device token after registration. The device token can be retrieved by calling:

```
PinpointManager::getNotificationClient().getDeviceToken()
```

Complete the following steps using the AWS CLI.

To test Baidu push notifications

1. Register Baidu as a channel with your Amazon Pinpoint project. Provide the Baidu API key and Secret key.

   ```
   aws pinpoint update-baidu-channel --application-id YourPinpointAppId --baidu-channel-request "{

       \"ApiKey\": \"BAIDU_API_KEY\",

       \"Enabled\": true,

       \"SecretKey\": \"BAIDU_SECRET_KEY\"

   }"
   ```

2. Install your app on to a Baidu-enabled device and capture the generated device token.
3. Send a direct message to the device specifying the device token as the address.

```
aws pinpoint send-messages --application-id YourPinpointAppId --message-request "{
  \"Addresses\": {
      \"DeviceToken\": {
          \"ChannelType\": \"BAIDU\"
      }
  },
  \"MessageConfiguration\": {
      \"BaiduMessage\": {
          \"RawContent\":
\"{'pinpoint.campaign.campaign_id':'_DIRECT','pinpoint.notification.silentPush':0,'pinpoint.openApp
 World.'}\"
      }
  }
}"
```

Defining Your Audience to Amazon Pinpoint

In Amazon Pinpoint, each member of your audience is represented by one or more endpoints. When you use Amazon Pinpoint to send a message, you direct the message to the endpoints that represent the members of your target audience. Each endpoint definition includes a message destination—such as a device token, email address, or phone number. It also includes data about your users and their devices. Before you analyze, segment, or engage your audience, the first step is to add endpoints to your Amazon Pinpoint project.

To add endpoints, you can:

- Integrate Amazon Pinpoint with your Android, iOS, or JavaScript client so that endpoints are added automatically when users visit your application.
- Use the Amazon Pinpoint API to add endpoints individually or in batches.
- Import endpoint definitions that are stored outside of Amazon Pinpoint.

After you add endpoints, you can:

- View analytics about your audience in the Amazon Pinpoint console.
- Learn about your audience by looking up or exporting endpoint data.
- Define audience segments based on endpoint attributes, such as demographic data or user interests.
- Engage your target audiences with tailored messaging campaigns.
- Send messages directly to lists of endpoints.

Use the topics in this section to add, update, and delete endpoints by using the Amazon Pinpoint API. If you want to add endpoints automatically from your Android, iOS, or JavaScript client, see Registering Endpoints in Your Application (p. 5) instead.

Topics

Adding Endpoints to Amazon Pinpoint

An *endpoint* represents a destination that you can message—such as a mobile device, phone number, or email address. Before you can message a member of your audience, you must define one or more endpoints for that individual.

When you define an endpoint, you specify the *channel* and *address*. The channel is the type of platform that you use to message the endpoint. Examples of channels include a push notification service, SMS, or email. The address specifies where to message the endpoint, such as a device token, phone number, or email address.

To add more details about your audience, you can enrich your endpoints with custom and standard attributes. These attributes might include data about your users, their preferences, their devices, the versions of the client that they use, or their locations. When you add this type of data to your endpoints, you're able to:

- View charts about your audience in the Amazon Pinpoint console.
- Segment your audience based on endpoint attributes so that you can send your messages to the right target audience.
- Personalize your messages by incorporating message variables that are substituted with endpoint attribute values.

A mobile or JavaScript client application registers endpoints automatically if you integrate Amazon Pinpoint by using the AWS Mobile SDKs or the AWS Amplify JavaScript library. The client registers an endpoint for each new user, and it updates endpoints for returning users. To register endpoints from a mobile or JavaScript client, see Registering Endpoints in Your Application (p. 5).

Examples

The following examples show you how to add an endpoint to an Amazon Pinpoint project. The endpoint represents an audience member who lives in Seattle and uses an iPhone. This person can be messaged through the Apple Push Notification service (APNs). The endpoint's address is the device token that's provided by APNs.

AWS CLI

You can use Amazon Pinpoint by running commands with the AWS CLI.

Example Update Endpoint Command

To add or update an endpoint, use the update-endpoint command:

```
$ aws pinpoint update-endpoint \
> --application-id application-id \
> --endpoint-id endpoint-id \
> --endpoint-request file://endpoint-request-file.json
```

Where:

- *application-id* is the ID of the Amazon Pinpoint project in which you're adding or updating an endpoint.
- *example-endpoint* is the ID that you're assigning to a new endpoint, or it's the ID of an existing endpoint that you're updating.
- *endpoint-request-file.json* is the file path to a local JSON file that contains the input for the `--endpoint-request` parameter.

Example Endpoint Request File

The example `update-endpoint` command uses a JSON file as the argument for the `--endpoint-request` parameter. This file contains an endpoint definition like the following:

```
{
  "ChannelType": "APNS",
  "Address": "1a2b3c4d5e6f7g8h9i0j1k2l3m4n5o6p7q8r9s0t1u2v3w4x5y6z7a8b9c0d1e2f",
  "Attributes": {
    "interests": [
      "technology",
      "music",
      "travel"
    ]
  },
  "Metrics": {
    "technology_interest_level": 9.0,
    "music_interest_level": 6.0,
    "travel_interest_level": 4.0
  },
  "Demographic": {
    "AppVersion": "1.0",
    "Make": "apple",
    "Model": "iPhone",
    "ModelVersion": "8",
    "Platform": "ios",
    "PlatformVersion": "11.3.1",
    "Timezone": "America/Los_Angeles"
  },
  "Location": {
    "Country": "US",
    "City": "Seattle",
    "PostalCode": "98121",
    "Latitude": 47.61,
    "Longitude": -122.33
  }
}
```

For the attributes that you can use to define an endpoint, see the EndpointRequest schema in the *Amazon Pinpoint API Reference*.

AWS SDK for Java

You can use the Amazon Pinpoint API in your Java applications by using the client that's provided by the AWS SDK for Java.

Example Code

To add an endpoint, initialize an `EndpointRequest` object, and pass it to the `updateEndpoint` method of the `AmazonPinpoint` client:

```
import com.amazonaws.regions.Regions;
import com.amazonaws.services.pinpoint.AmazonPinpoint;
import com.amazonaws.services.pinpoint.AmazonPinpointClientBuilder;
import com.amazonaws.services.pinpoint.model.*;
import java.util.Arrays;

public class AddExampleEndpoint {

    public static void main(String[] args) {

        final String USAGE = "\n" +
                "AddExampleEndpoint - Adds an example endpoint to an Amazon Pinpoint
 application." +
                "Usage: AddExampleEndpoint <applicationId>" +
                "Where:\n" +
                "  applicationId - The ID of the Amazon Pinpoint application to add the
 example " +
                "endpoint to.";

        if (args.length < 1) {
            System.out.println(USAGE);
            System.exit(1);
        }

        String applicationId = args[0];
        // The device token assigned to the user's device by Apple Push Notification
 service (APNs).
        String deviceToken =
 "1a2b3c4d5e6f7g8h9i0j1k2l3m4n5o6p7q8r9s0t1u2v3w4x5y6z7a8b9c0d1e2f";

        // Initializes an endpoint definition with channel type and address.
        EndpointRequest wangXiulansIphoneEndpoint = new EndpointRequest()
                .withChannelType(ChannelType.APNS)
                .withAddress(deviceToken);

        // Adds custom attributes to the endpoint.
        wangXiulansIphoneEndpoint.addAttributesEntry("interests", Arrays.asList(
                "technology",
                "music",
                "travel"));

        // Adds custom metrics to the endpoint.
        wangXiulansIphoneEndpoint.addMetricsEntry("technology_interest_level", 9.0);
        wangXiulansIphoneEndpoint.addMetricsEntry("music_interest_level", 6.0);
        wangXiulansIphoneEndpoint.addMetricsEntry("travel_interest_level", 4.0);

        // Adds standard demographic attributes.
        wangXiulansIphoneEndpoint.setDemographic(new EndpointDemographic()
                .withAppVersion("1.0")
                .withMake("apple")
                .withModel("iPhone")
                .withModelVersion("8")
                .withPlatform("ios")
                .withPlatformVersion("11.3.1")
                .withTimezone("America/Los_Angeles"));

        // Adds standard location attributes.
        wangXiulansIphoneEndpoint.setLocation(new EndpointLocation()
                .withCountry("US")
                .withCity("Seattle")
                .withPostalCode("98121")
                .withLatitude(47.61)
                .withLongitude(-122.33));

        // Initializes the Amazon Pinpoint client.
        AmazonPinpoint pinpointClient = AmazonPinpointClientBuilder.standard()
                .withRegion(Regions.US_EAST_1).build();

        // Updates or creates the endpoint with Amazon Pinpoint.
        UpdateEndpointResult result = pinpointClient.updateEndpoint(new
 UpdateEndpointRequest()
                .withApplicationId(applicationId)
                .withEndpointId("example_endpoint")
                .withEndpointRequest(wangXiulansIphoneEndpoint));

        System.out.format("Update endpoint result: %s\n",
 result.getMessageBody().getMessage());

    }

}
```

HTTP

You can use Amazon Pinpoint by making HTTP requests directly to the REST API.

Example PUT Endpoint Request

To add an endpoint, issue a `PUT` request to the Endpoint resource at the following URI:

```
/v1/apps/application-id/endpoints/endpoint-id
```

Where:

- *application-id* is the ID of the Amazon Pinpoint project in which you're adding or updating an endpoint.
- *endpoint-id* is the ID that you're assigning to a new endpoint, or it's the ID of an existing endpoint that you're updating.

In your request, include the required headers, and provide the EndpointRequest JSON as the body:

```
PUT /v1/apps/application_id/endpoints/example_endpoint HTTP/1.1
Host: pinpoint.us-east-1.amazonaws.com
X-Amz-Date: 20180415T182538Z
Content-Type: application/json
Accept: application/json
X-Amz-Date: 20180428T004705Z
Authorization: AWS4-HMAC-SHA256 Credential=AKIAIOSFODNN7EXAMPLE/20180428/us-east-1/
mobiletargeting/aws4_request, SignedHeaders=accept;content-length;content-type;host;x-
amz-date, Signature=c25cbd6bf61bd3b3667c571ae764b9bf2d8af61b875cacced95d1e68d91b4170
Cache-Control: no-cache

{
  "ChannelType": "APNS",
  "Address": "1a2b3c4d5e6f7g8h9i0j1k2l3m4n5o6p7q8r9s0t1u2v3w4x5y6z7a8b9c0d1e2f",
  "Attributes": {
    "interests": [
      "technology",
      "music",
      "travel"
    ]
  },
  "Metrics": {
    "technology_interest_level": 9.0,
    "music_interest_level": 6.0,
    "travel_interest_level": 4.0
  },
  "Demographic": {
    "AppVersion": "1.0",
    "Make": "apple",
    "Model": "iPhone",
    "ModelVersion": "8",
    "Platform": "ios",
    "PlatformVersion": "11.3.1",
    "Timezone": "America/Los_Angeles"
  },
  "Location": {
    "Country": "US",
    "City": "Seattle",
    "PostalCode": "98121",
    "Latitude": 47.61,
    "Longitude": -122.33
  }
}
```

If your request succeeds, you receive a response like the following:

```
{
    "RequestID": "67e572ed-41d5-11e8-9dc5-db288f3cbb72",
    "Message": "Accepted"
}
```

Related Information

For more information about the Endpoint resource in the Amazon Pinpoint API, including the supported HTTP methods and request parameters, see Endpoint in the *Amazon Pinpoint API Reference.*

For more information about personalizing messages with variables, see Message Variables in the *Amazon Pinpoint User Guide*.

For the limits that apply to endpoints, such as the number of attributes you can assign, see the section called "Endpoint Limits" (p. 135).

Associating Users with Amazon Pinpoint Endpoints

An endpoint can include attributes that define a *user*, which represents a person in your audience. For example, a user might represent someone who installed your mobile app, or someone who has an account on your website.

You define a user by specifying a unique user ID and, optionally, custom user attributes. If someone uses your app on multiple devices, or if that person can be messaged at multiple addresses, you can assign the same user ID to multiple endpoints. In this case, Amazon Pinpoint synchronizes user attributes across the endpoints. So, if you add a user attribute to one endpoint, Amazon Pinpoint adds that attribute to each endpoint that includes the same user ID.

You can add user attributes to track data that applies to an individual and doesn't vary based on which device the person is using. For example, you can add attributes for a person's name, age, or account status.

Tip
If your application uses Amazon Cognito user pools to handle user authentication, Amazon Cognito can add user IDs and attributes to your endpoints automatically. For the endpoint user ID value, Amazon Cognito assigns the `sub` value that's assigned to the user in the user pool. To add users with Amazon Cognito, see Using Amazon Pinpoint Analytics with Amazon Cognito User Pools.

After you add user definitions to your endpoints, you have more options for how you segment your audience. You can define a segment based on user attributes, or you can define a segment by importing a list of user IDs. When you send a message to a segment that's based on users, the potential destinations include each endpoint that's associated with each user in the segment.

You also have more options for how you message your audience. You can use a campaign to message a segment of users, or you can send a message directly to a list of user IDs. To personalize your message, you can include message variables that are substituted with user attribute values.

Examples

The following examples show you how to add a user definition to an endpoint.

AWS CLI

You can use Amazon Pinpoint by running commands with the AWS CLI.

Example Update Endpoint Command

To add a user to an endpoint, use the update-endpoint command. For the `--endpoint-request` parameter, you can define a new endpoint, which can include a user. Or, to update an existing endpoint, you can provide just the attributes that you want to change. The following example adds a user to an existing endpoint by providing only the user attributes:

```
$ aws pinpoint update-endpoint \
> --application-id application-id \
> --endpoint-id endpoint-id \
> --endpoint-request file://endpoint-request-file.json
```

Where:

- *application-id* is the ID of the Amazon Pinpoint project in which you're adding or updating an endpoint.
- *endpoint-id* is the ID that you're assigning to a new endpoint, or it's the ID of an existing endpoint that you're updating.
- *endpoint-request-file.json* is the file path to a local JSON file that contains the input for the `--endpoint-request` parameter.

Example Endpoint Request File

The example `update-endpoint` command uses a JSON file as the argument for the `--endpoint-request` parameter. This file contains a user definition like the following:

```
{
  "User": {
    "UserId": "example_user",
    "UserAttributes": {
      "name": [
        "Wang",
        "Xiulan"
      ],
      "gender": [
        "female"
      ],
      "age": [
        "39"
      ]
    }
  }
}
```

For the attributes that you can use to define a user, see the `User` object in the EndpointRequest schema in the *Amazon Pinpoint API Reference*.

AWS SDK for Java

You can use the Amazon Pinpoint API in your Java applications by using the client that's provided by the AWS SDK for Java.

Example Code

To add a user to an endpoint, initialize an EndpointRequest object, and pass it to the `updateEndpoint` method of the `AmazonPinpoint` client. You can use this object to define a new endpoint, which can include a user. Or, to update an existing endpoint, you can update just the properties that you want to change. The following example adds a user to an existing endpoint by adding an EndpointUser object to the EndpointRequest object:

```
import com.amazonaws.regions.Regions;
import com.amazonaws.services.pinpoint.AmazonPinpoint;
import com.amazonaws.services.pinpoint.AmazonPinpointClientBuilder;
import com.amazonaws.services.pinpoint.model.EndpointRequest;
import com.amazonaws.services.pinpoint.model.EndpointUser;
import com.amazonaws.services.pinpoint.model.UpdateEndpointRequest;
import com.amazonaws.services.pinpoint.model.UpdateEndpointResult;

import java.util.Arrays;
import java.util.Collections;

public class AddExampleUser {

    public static void main(String[] args) {

        final String USAGE = "\n" +
                "AddExampleUser - Adds a user definition to the specified Amazon
 Pinpoint endpoint." +
                "Usage: AddExampleUser <endpointId> <applicationId>" +
                "Where:\n" +
                "  endpointId - The ID of the endpoint to add the user to." +
                "  applicationId - The ID of the Amazon Pinpoint application that
 contains the endpoint.";

        if (args.length < 1) {
            System.out.println(USAGE);
            System.exit(1);
        }

        String endpointId = args[0];
        String applicationId = args[1];

        // Creates a new user definition.
        EndpointUser wangXiulan = new EndpointUser().withUserId("example_user");

        // Assigns custom user attributes.
        wangXiulan.addUserAttributesEntry("name", Arrays.asList("Wang", "Xiulan"));
        wangXiulan.addUserAttributesEntry("gender",
 Collections.singletonList("female"));
        wangXiulan.addUserAttributesEntry("age", Collections.singletonList("39"));

        // Adds the user definition to the EndpointRequest that is passed to the Amazon
 Pinpoint client.
        EndpointRequest wangXiulansIphone = new EndpointRequest()
                .withUser(wangXiulan);

        // Initializes the Amazon Pinpoint client.
        AmazonPinpoint pinpointClient = AmazonPinpointClientBuilder.standard()
                .withRegion(Regions.US_EAST_1).build();

        // Updates the specified endpoint with Amazon Pinpoint.
        UpdateEndpointResult result = pinpointClient.updateEndpoint(new
 UpdateEndpointRequest()
                .withEndpointRequest(wangXiulansIphone)
                .withApplicationId(applicationId)
                .withEndpointId(endpointId));

        System.out.format("Update endpoint result: %s\n",
 result.getMessageBody().getMessage());

    }
}
```

HTTP

You can use Amazon Pinpoint by making HTTP requests directly to the REST API.

Example Put Endpoint Request with User Definition

To add a user to an endpoint, issue a `PUT` request to the Endpoint resource at the following URI:

```
/v1/apps/application-id/endpoints/endpoint-id
```

Where:

- *application-id* is the ID of the Amazon Pinpoint project in which you're adding or updating an endpoint.
- *endpoint-id* is the ID that you're assigning to a new endpoint, or it's the ID of an existing endpoint that you're updating.

In your request, include the required headers, and provide the EndpointRequest JSON as the body. The request body can define a new endpoint, which can include a user. Or, to update an existing endpoint, you can provide just the attributes that you want to change. The following example adds a user to an existing endpoint by providing only the user attributes:

```
PUT /v1/apps/application_id/endpoints/example_endpoint HTTP/1.1
Host: pinpoint.us-east-1.amazonaws.com
X-Amz-Date: 20180415T182538Z
Content-Type: application/json
Accept: application/json
Authorization: AWS4-HMAC-SHA256 Credential=AKIAIOSFODNN7EXAMPLE/20180501/us-east-1/
mobiletargeting/aws4_request, SignedHeaders=accept;content-length;content-type;host;x-
amz-date, Signature=c25cbd6bf61bd3b3667c571ae764b9bf2d8af61b875cacced95d1e68d91b4170
Cache-Control: no-cache

{
  "User": {
    "UserId": "example_user",
    "UserAttributes": {
      "name": [
        "Wang",
        "Xiulan"
      ],
      "gender": [
        "female"
      ],
      "age": [
        "39"
      ]
    }
  }
}
```

If the request succeeds, you receive a response like the following:

```
{
    "RequestID": "67e572ed-41d5-11e8-9dc5-db288f3cbb72",
    "Message": "Accepted"
}
```

Related Information

For more information about the Endpoint resource in the Amazon Pinpoint API, including the supported HTTP methods and request parameters, see Endpoint in the *Amazon Pinpoint API Reference.*

For more information about personalizing messages with variables, see Message Variables in the *Amazon Pinpoint User Guide*.

To define a segment by importing a list of user IDs, see Importing Segments in the *Amazon Pinpoint User Guide*.

To send a direct message to up to 100 user IDs, see Users Messages in the *Amazon Pinpoint API Reference*.

For the limits that apply to endpoints, including the number of user attributes you can assign, see the section called "Endpoint Limits" (p. 135).

Adding a Batch of Endpoints to Amazon Pinpoint

You can add or update multiple endpoints in a single operation by providing the endpoints in batches. Each batch request can include up to 100 endpoint definitions.

If you want to add or update more than 100 endpoints in a single operation, see Importing Endpoints into Amazon Pinpoint (p. 54) instead.

Examples

The following examples show you how to add two endpoints at once by including the endpoints in a batch request.

AWS CLI

You can use Amazon Pinpoint by running commands with the AWS CLI.

Example Update Endpoints Batch Command

To submit an endpoint batch request, use the `update-endpoints-batch` command:

```
$ aws pinpoint update-endpoints-batch \
> --application-id application-id \
> --endpoint-batch-request file://endpoint_batch_request_file.json
```

Where:

- *application-id* is the ID of the Amazon Pinpoint project in which you're adding or updating the endpoints.
- *endpoint_batch_request_file.json* is the file path to a local JSON file that contains the input for the `--endpoint-batch-request` parameter.

Example Endpoint Batch Request File

The example `update-endpoints-batch` command uses a JSON file as the argument for the `--endpoint-request` parameter. This file contains a batch of endpoint definitions like the following:

```
{
  "Item": [
    {
      "ChannelType": "EMAIL",
      "Address": "richard_roe@example.com",
      "Attributes": {
        "interests": [
          "music",
          "books"
        ]
      },
      "Metrics": {
        "music_interest_level": 3.0,
        "books_interest_level": 7.0
      },
      "Id": "example_endpoint_1",
      "User": {
        "UserId": "example_user_1",
        "UserAttributes": {
          "name": [
            "Richard",
            "Roe"
          ]
        }
      }
    },
    {
      "ChannelType": "SMS",
      "Address": "+16145550100",
      "Attributes": {
        "interests": [
          "cooking",
          "politics",
          "finance"
        ]
      },
      "Metrics": {
        "cooking_interest_level": 5.0,
        "politics_interest_level": 8.0,
        "finance_interest_level": 4.0
      },
      "Id": "example_endpoint_2",
      "User": {
        "UserId": "example_user_2",
        "UserAttributes": {
          "name": [
            "Mary",
            "Major"
          ]
        }
      }
    }
  ]
}
```

For the attributes that you can use to define a batch of endpoints, see the EndpointBatchRequest schema in the *Amazon Pinpoint API Reference*.

AWS SDK for Java

You can use the Amazon Pinpoint API in your Java applications by using the client that's provided by the AWS SDK for Java.

Example Code

To submit an endpoint batch request, initialize an `EndpointBatchRequest` object, and pass it to the `updateEndpointsBatch` method of the `AmazonPinpoint` client. The following example populates an `EndpointBatchRequest` object with two `EndpointBatchItem` objects:

```
import com.amazonaws.regions.Regions;
import com.amazonaws.services.pinpoint.AmazonPinpoint;
import com.amazonaws.services.pinpoint.AmazonPinpointClientBuilder;
import com.amazonaws.services.pinpoint.model.ChannelType;
import com.amazonaws.services.pinpoint.model.EndpointBatchItem;
import com.amazonaws.services.pinpoint.model.EndpointBatchRequest;
import com.amazonaws.services.pinpoint.model.EndpointUser;
import com.amazonaws.services.pinpoint.model.UpdateEndpointsBatchRequest;
import com.amazonaws.services.pinpoint.model.UpdateEndpointsBatchResult;

import java.util.Arrays;

public class AddExampleEndpoints {

    public static void main(String[] args) {

        final String USAGE = "\n" +
                "AddExampleEndpoints - Adds example endpoints to an Amazon Pinpoint
 application." +
                "Usage: AddExampleEndpoints <applicationId>" +
                "Where:\n" +
                "  applicationId - The ID of the Amazon Pinpoint application to add the
 example endpoints to.";

        if (args.length < 1) {
            System.out.println(USAGE);
            System.exit(1);
        }


        String applicationId = args[0];

        // Initializes an endpoint definition with channel type, address, and ID.
        EndpointBatchItem richardRoesEmailEndpoint = new EndpointBatchItem()
                .withChannelType(ChannelType.EMAIL)
                .withAddress("richard_roe@example.com")
                .withId("example_endpoint_1");

        // Adds custom attributes to the endpoint.
        richardRoesEmailEndpoint.addAttributesEntry("interests", Arrays.asList(
                "music",
                "books"));

        // Adds custom metrics to the endpoint.
        richardRoesEmailEndpoint.addMetricsEntry("music_interest_level", 3.0);
        richardRoesEmailEndpoint.addMetricsEntry("books_interest_level", 7.0);

        // Initializes a user definition with a user ID.
        EndpointUser richardRoe = new EndpointUser().withUserId("example_user_1");

        // Adds custom user attributes.
        richardRoe.addUserAttributesEntry("name", Arrays.asList("Richard", "Roe"));

        // Adds the user definition to the endpoint.
        richardRoesEmailEndpoint.setUser(richardRoe);

        // Initializes an endpoint definition with channel type, address, and ID.
        EndpointBatchItem maryMajorsSmsEndpoint = new EndpointBatchItem()
                .withChannelType(ChannelType.SMS)
                .withAddress("+16145550100")
                .withId("example_endpoint_2");

        // Adds custom attributes to the endpoint.
        maryMajorsSmsEndpoint.addAttributesEntry("interests", Arrays.asList(
                "cooking",
                "politics",
                "finance"));

        // Adds custom metrics to the endpoint.
        maryMajorsSmsEndpoint.addMetricsEntry("cooking_interest_level", 5.0);
        maryMajorsSmsEndpoint.addMetricsEntry("politics_interest_level", 8.0);
        maryMajorsSmsEndpoint.addMetricsEntry("finance_interest_level", 4.0);
        // Initializes a user definition with a user ID.
        EndpointUser maryMajor = new EndpointUser().withUserId("example_user_2");

        // Adds custom user attributes.
        maryMajor.addUserAttributesEntry("name", Arrays.asList("Mary", "Major"));

        // Adds the user definition to the endpoint.
        maryMajorsSmsEndpoint.setUser(maryMajor);

        // Adds multiple endpoint definitions to a single request object.
        EndpointBatchRequest endpointList = new EndpointBatchRequest()
                .withItem(richardRoesEmailEndpoint)
                .withItem(maryMajorsSmsEndpoint);

        // Initializes the Amazon Pinpoint client.
        AmazonPinpoint pinpointClient = AmazonPinpointClientBuilder.standard()
                .withRegion(Regions.US_EAST_1).build();

        // Updates or creates the endpoints with Amazon Pinpoint.
        UpdateEndpointsBatchResult result = pinpointClient.updateEndpointsBatch(
                new UpdateEndpointsBatchRequest()
                .withApplicationId(applicationId)
                .withEndpointBatchRequest(endpointList));

        System.out.format("Update endpoints batch result: %s\n",
                result.getMessageBody().getMessage());

    }
}
```

HTTP

You can use Amazon Pinpoint by making HTTP requests directly to the REST API.

Example Put Endpoints Request

To submit an endpoint batch request, issue a `PUT` request to the Endpoints resource at the following URI:

`/v1/apps/application-id/endpoints`

Where *application-id* is the ID of the Amazon Pinpoint project in which you're adding or updating the endpoints.

In your request, include the required headers, and provide the EndpointBatchRequest JSON as the body:

```
PUT /v1/apps/application_id/endpoints HTTP/1.1
Host: pinpoint.us-east-1.amazonaws.com
Content-Type: application/json
Accept: application/json
X-Amz-Date: 20180501T184948Z
Authorization: AWS4-HMAC-SHA256 Credential=AKIAIOSFODNN7EXAMPLE/20180501/us-east-1/
mobiletargeting/aws4_request, SignedHeaders=accept;content-length;content-type;host;x-
amz-date, Signature=c25cbd6bf61bd3b3667c571ae764b9bf2d8af61b875cacced95d1e68d91b4170
Cache-Control: no-cache

{
  "Item": [
    {
      "ChannelType": "EMAIL",
      "Address": "richard_roe@example.com",
      "Attributes": {
        "interests": [
          "music",
          "books"
        ]
      },
      "Metrics": {
        "music_interest_level": 3.0,
        "books_interest_level": 7.0
      },
      "Id": "example_endpoint_1",
      "User": {
        "UserId": "example_user_1",
        "UserAttributes": {
          "name": [
            "Richard",
            "Roe"
          ]
        }
      }
    },
    {
      "ChannelType": "SMS",
      "Address": "+16145550100",
      "Attributes": {
        "interests": [
          "cooking",
          "politics",
          "finance"
        ]
      },
      "Metrics": {
        "cooking_interest_level": 5.0,
        "politics_interest_level": 8.0,
        "finance_interest_level": 4.0
      },
      "Id": "example_endpoint_2",
      "User": {
        "UserId": "example_user_2",
        "UserAttributes": {
          "name": [
            "Mary",
            "Major"
          ]
        }
      }
    }
  ]
}
```

If your request succeeds, you receive a response like the following:

```
{
    "RequestID": "67e572ed-41d5-11e8-9dc5-db288f3cbb72",
    "Message": "Accepted"
}
```

Related Information

For more information about the Endpoint resource in the Amazon Pinpoint API, including the supported HTTP methods and request parameters, see Endpoint in the *Amazon Pinpoint API Reference.*

Importing Endpoints into Amazon Pinpoint

You can add or update endpoints in large numbers by importing them from an Amazon S3 bucket. Importing endpoints is useful if you have records about your audience outside of Amazon Pinpoint, and you want to add this information to an Amazon Pinpoint project. In this case, you would:

1. Create endpoint definitions that are based on your own audience data.
2. Save these endpoint definitions in one or more files, and upload the files to an Amazon S3 bucket.
3. Add the endpoints to your Amazon Pinpoint project by importing them from the bucket.

Each import job can transfer up to 1 GB of data. In a typical job, where each endpoint is 4 KB or less, you could import around 250,000 endpoints. You can run up to two concurrent import jobs per AWS account. If you need more bandwidth for your import jobs, you can submit a service limit increase request with AWS Support. For more information, see Requesting a Limit Increase (p. 141).

Before You Begin

Before you can import endpoints, you need the following resources in your AWS account:

- An Amazon S3 bucket. To create a bucket, see Create a Bucket in the *Amazon Simple Storage Service Getting Started Guide*.
- An AWS Identity and Access Management (IAM) role that grants Amazon Pinpoint read permissions for your Amazon S3 bucket. To create the role, see IAM Role for Importing Endpoints or Segments (p. 127).

Examples

The following examples demonstrate how to add endpoint definitions to your Amazon S3 bucket, and then import those endpoints into an Amazon Pinpoint project.

Files with Endpoint Definitions

The files that you add to your Amazon S3 bucket can contain endpoint definitions in CSV or newline-delimited JSON format. For the attributes that you can use to define your endpoints, see the EndpointRequest JSON schema in the *Amazon Pinpoint API Reference*.

CSV

You can import endpoints that are defined in a CSV file, as in the following example:

```
ChannelType,Address,Location.Country,Demographic.Platform,Demographic.Make,User.UserId
SMS,2065550182,CAN,Android,LG,example-user-id-1
APNS,1a2b3c4d5e6f7g8h9i0j1a2b3c4d5e6f,USA,iOS,Apple,example-user-id-2
EMAIL,john.stiles@example.com,USA,iOS,Apple,example-user-id-2
```

The first line is the header, which contains the endpoint attributes. Specify nested attributes by using dot notation, as in `Location.Country`.

The subsequent lines define the endpoints by providing values for each of the attributes in the header.

To include a comma, line break, or double quote in a value, enclose the value in double quotes, as in `"aaa,bbb"`. For more information about the CSV format, see RFC 4180 Common Format and MIME Type for Comma-Separated Values (CSV) Files.

JSON

You can import endpoints that are defined in a newline-delimited JSON file, as in the following example:

```
{"ChannelType":"SMS","Address":"2065550182","Location":{"Country":"CAN"},"Demographic":
{"Platform":"Android","Make":"LG"},"User":{"UserId":"example-user-id-1"}}
{"ChannelType":"APNS","Address":"1a2b3c4d5e6f7g8h9i0j1a2b3c4d5e6f","Location":
{"Country":"USA"},"Demographic":{"Platform":"iOS","Make":"Apple"},"User":
{"UserId":"example-user-id-2"}}
{"ChannelType":"EMAIL","Address":"john.stiles@example.com","Location":
{"Country":"USA"},"Demographic":{"Platform":"iOS","Make":"Apple"},"User":
{"UserId":"example-user-id-2"}}
```

In this format, each line is a complete JSON object that contains an individual endpoint definition.

Import Job Requests

The following examples show you how to add endpoint definitions to Amazon S3 by uploading a local file to a bucket. Then, the examples import the endpoint definitions into an Amazon Pinpoint project.

AWS CLI

You can use Amazon Pinpoint by running commands with the AWS CLI.

Example S3 CP Command

To upload a local file to an Amazon S3 bucket, use the Amazon S3 `cp` command:

```
$ aws s3 cp ./endpoints-file s3://bucket-name/prefix/
```

Where:

- *./endpoints-file* is the file path to a local file that contains the endpoint definitions.
- *bucket-name/prefix/* is the name of your Amazon S3 bucket and, optionally, a prefix that helps you organize the objects in your bucket hierarchically. For example, a useful prefix might be `pinpoint/imports/endpoints/`.

Example Create Import Job Command

To import endpoint definitions from an Amazon S3 bucket, use the `create-import-job` command:

```
$ aws pinpoint create-import-job \
> --application-id application-id \
> --import-job-request \
> S3Url=s3://bucket-name/prefix/key,\
> RoleArn=iam-import-role-arn,\
> Format=format,\
> RegisterEndpoints=true
```

Where:

- *application-id* is the ID of the Amazon Pinpoint project that you're importing endpoints for.
- *bucket-name/prefix/key* is the location in Amazon S3 that contains one or more objects to import. The location can end with the key for an individual object, or it can end with a prefix that qualifies multiple objects.

- *iam-import-role-arn* is the Amazon Resource Name (ARN) of an IAM role that grants Amazon Pinpoint read access to the bucket.
- *format* can be either `JSON` or `CSV`, depending on which format you used to define your endpoints. If the Amazon S3 location includes multiple objects of mixed formats, Amazon Pinpoint imports only the objects that match the specified format.

The response includes details about the import job:

```
{
    "ImportJobResponse": {
        "CreationDate": "2018-05-24T21:26:33.995Z",
        "Definition": {
            "DefineSegment": false,
            "ExternalId": "463709046829",
            "Format": "JSON",
            "RegisterEndpoints": true,
            "RoleArn": "iam-import-role-arn",
            "S3Url": "s3://bucket-name/prefix/key"
        },
        "Id": "d5ecad8e417d498389e1d5b9454d4e0c",
        "JobStatus": "CREATED",
        "Type": "IMPORT"
    }
}
```

The response provides the job ID with the `Id` attribute. You can use this ID to check the current status of the import job.

Example Get Import Job Command

To check the current status of an import job, use the `get-import-job` command:

```
$ aws pinpoint get-import-job \
> --application-id application-id \
> --job-id job-id
```

Where:

- *application-id* is the ID of the Amazon Pinpoint project that the import job was initiated for.
- *job-id* is the ID of the import job that you're checking.

The response to this command provides the current state of the import job:

```
{
    "ImportJobResponse": {
        "ApplicationId": "application-id",
        "CompletedPieces": 1,
        "CompletionDate": "2018-05-24T21:26:45.308Z",
        "CreationDate": "2018-05-24T21:26:33.995Z",
        "Definition": {
            "DefineSegment": false,
            "ExternalId": "463709046829",
            "Format": "JSON",
            "RegisterEndpoints": true,
            "RoleArn": "iam-import-role-arn",
            "S3Url": "s3://s3-bucket-name/prefix/endpoint-definitions.json"
        },
        "FailedPieces": 0,
        "Id": "job-id",
        "JobStatus": "COMPLETED",
        "TotalFailures": 0,
        "TotalPieces": 1,
        "TotalProcessed": 3,
        "Type": "IMPORT"
    }
}
```

The response provides the job status with the `JobStatus` attribute.

AWS SDK for Java

You can use the Amazon Pinpoint API in your Java applications by using the client that's provided by the AWS SDK for Java.

Example Code

To upload a file with endpoint definitions to Amazon S3, use the `putObject` method of the `AmazonS3` client.

To import the endpoints into an Amazon Pinpoint project, initialize a `CreateImportJobRequest` object. Then, pass this object to the `createImportJob` method of the `AmazonPinpoint` client.

```
package com.amazonaws.examples.pinpoint;

import com.amazonaws.AmazonServiceException;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.pinpoint.AmazonPinpoint;
import com.amazonaws.services.pinpoint.AmazonPinpointClientBuilder;
import com.amazonaws.services.pinpoint.model.CreateImportJobRequest;
import com.amazonaws.services.pinpoint.model.CreateImportJobResult;
import com.amazonaws.services.pinpoint.model.Format;
import com.amazonaws.services.pinpoint.model.GetImportJobRequest;
import com.amazonaws.services.pinpoint.model.GetImportJobResult;
import com.amazonaws.services.pinpoint.model.ImportJobRequest;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.AmazonS3Exception;
import java.io.File;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.List;
import java.util.concurrent.TimeUnit;

public class ImportEndpoints {

    public static void main(String[] args) {

        final String USAGE = "\n" +
        "ImportEndpoints - Adds endpoints to an Amazon Pinpoint application by: \n" +
        "1.) Uploading the endpoint definitions to an Amazon S3 bucket. \n" +
        "2.) Importing the endpoint definitions from the bucket to an Amazon Pinpoint "
 +
                "application.\n\n" +
        "Usage: ImportEndpoints <endpointsFileLocation> <s3BucketName>
 <iamImportRoleArn> " +
                "<applicationId>\n\n" +
        "Where:\n" +
        "  endpointsFileLocation - The relative location of the JSON file that contains
 the " +
                "endpoint definitions.\n" +
        "  s3BucketName - The name of the Amazon S3 bucket to upload the JSON file to.
 If the " +
                "bucket doesn't exist, a new bucket is created.\n" +
        "  iamImportRoleArn - The ARN of an IAM role that grants Amazon Pinpoint read " +
                "permissions to the S3 bucket.\n" +
        "  applicationId - The ID of the Amazon Pinpoint application to add the endpoints to.";

        if (args.length < 1) {
            System.out.println(USAGE);
            System.exit(1);
        }

        String endpointsFileLocation = args[0];
        String s3BucketName = args[1];
        String iamImportRoleArn = args[2];
        String applicationId = args[3];

        Path endpointsFilePath = Paths.get(endpointsFileLocation);
        File endpointsFile = new File(endpointsFilePath.toAbsolutePath().toString());
        uploadToS3(endpointsFile, s3BucketName);

        importToPinpoint(endpointsFile.getName(), s3BucketName, iamImportRoleArn, applicationId);

    }

    private static void uploadToS3(File endpointsFile, String s3BucketName) {

        // Initializes Amazon S3 client.
        final AmazonS3 s3 = AmazonS3ClientBuilder.defaultClient();

        // Checks whether the specified bucket exists. If not, attempts to create one.
        if (!s3.doesBucketExistV2(s3BucketName)) {
            try {
                s3.createBucket(s3BucketName);
                System.out.format("Created S3 bucket %s.\n", s3BucketName);
            } catch (AmazonS3Exception e) {
                System.err.println(e.getErrorMessage());
                System.exit(1);
            }
        }

        // Uploads the endpoints file to the bucket.
        String endpointsFileName = endpointsFile.getName();
        System.out.format("Uploading %s to S3 bucket %s . . .\n", endpointsFileName, s3BucketName);
        try {
            s3.putObject(s3BucketName, "imports/" + endpointsFileName, endpointsFile);
            System.out.println("Finished uploading to S3.");
        } catch (AmazonServiceException e) {
            System.err.println(e.getErrorMessage());
            System.exit(1);
        }
    }

    private static void importToPinpoint(String endpointsFileName, String s3BucketName,
                                         String iamImportRoleArn, String applicationId) {

        // The S3 URL that Amazon Pinpoint requires to find the endpoints file.
        String s3Url = "s3://" + s3BucketName + "/imports/" + endpointsFileName;

        // Defines the import job that Amazon Pinpoint runs.
        ImportJobRequest importJobRequest = new ImportJobRequest()
                .withS3Url(s3Url)
                .withRegisterEndpoints(true)
                .withRoleArn(iamImportRoleArn)
```

```
                .withFormat(Format.JSON);
        CreateImportJobRequest createImportJobRequest = new CreateImportJobRequest()
                .withApplicationId(applicationId)
                .withImportJobRequest(importJobRequest);

        // Initializes the Amazon Pinpoint client.
        AmazonPinpoint pinpointClient = AmazonPinpointClientBuilder.standard()
                .withRegion(Regions.US_EAST_1).build();

        System.out.format("Importing endpoints in %s to Amazon Pinpoint application
 %s . . .\n",
                endpointsFileName, applicationId);

        try {

            // Runs the import job with Amazon Pinpoint.
            CreateImportJobResult importResult =
                    pinpointClient.createImportJob(createImportJobRequest);

            String jobId = importResult.getImportJobResponse().getId();
            GetImportJobResult getImportJobResult = null;
            String jobStatus = null;

            // Checks the job status until the job completes or fails.
            do {
                getImportJobResult = pinpointClient.getImportJob(new
 GetImportJobRequest()
                        .withJobId(jobId)
                        .withApplicationId(applicationId));
                jobStatus = getImportJobResult.getImportJobResponse().getJobStatus();
                System.out.format("Import job %s . . .\n", jobStatus.toLowerCase());
                TimeUnit.SECONDS.sleep(3);
            } while (!jobStatus.equals("COMPLETED") && !jobStatus.equals("FAILED"));

            if (jobStatus.equals("COMPLETED")) {
                System.out.println("Finished importing endpoints.");
            } else {
                System.err.println("Failed to import endpoints.");
                System.exit(1);
            }

            // Checks for entries that failed to import.
            // getFailures provides up to 100 of the first failed entries for the job,
 if any exist.
            List<String> failedEndpoints =
 getImportJobResult.getImportJobResponse().getFailures();
            if (failedEndpoints != null) {
                System.out.println("Failed to import the following entries:");
                for (String failedEndpoint : failedEndpoints) {
                    System.out.println(failedEndpoint);
                }
            }

        } catch (AmazonServiceException | InterruptedException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }

    }

}
```

HTTP

You can use Amazon Pinpoint by making HTTP requests directly to the REST API.

Example S3 PUT Object Request

To add your endpoint definitions to a bucket, use the Amazon S3 PUT Object operation, and provide the endpoint definitions as the body:

```
PUT /prefix/key HTTP/1.1
Content-Type: text/plain
Accept: application/json
Host: bucket-name.s3.amazonaws.com
X-Amz-Content-Sha256: c430dc094b0cec2905bc88d96314914d058534b14e2bc6107faa9daa12fdff2d
X-Amz-Date: 20180605T184132Z
Authorization: AWS4-HMAC-SHA256 Credential=AKIAIOSFODNN7EXAMPLE/20180605/
us-east-1/s3/aws4_request, SignedHeaders=accept;cache-control;content-
length;content-type;host;postman-token;x-amz-content-sha256;x-amz-date,
 Signature=c25cbd6bf61bd3b3667c571ae764b9bf2d8af61b875cacced95d1e68d91b4170
Cache-Control: no-cache

{"ChannelType":"SMS","Address":"2065550182","Location":{"Country":"CAN"},"Demographic":
{"Platform":"Android","Make":"LG"},"User":{"UserId":"example-user-id-1"}}
{"ChannelType":"APNS","Address":"1a2b3c4d5e6f7g8h9i0j1a2b3c4d5e6f","Location":
{"Country":"USA"},"Demographic":{"Platform":"iOS","Make":"Apple"},"User":
{"UserId":"example-user-id-2"}}
{"ChannelType":"EMAIL","Address":"john.stiles@example.com","Location":
{"Country":"USA"},"Demographic":{"Platform":"iOS","Make":"Apple"},"User":
{"UserId":"example-user-id-2"}}
```

Where:

- */prefix/key* is the prefix and key name for the object that will contain the endpoint definitions after the upload. You can use the prefix to organize your objects hierarchically. For example, a useful prefix might be `pinpoint/imports/endpoints/`.
- *bucket-name* is the name of the Amazon S3 bucket that you're adding the endpoint definitions to.

Example POST Import Job Request

To import endpoint definitions from an Amazon S3 bucket, issue a POST request to the Import Jobs resource. In your request, include the required headers and provide the ImportJobRequest JSON as the body:

```
POST /v1/apps/application_id/jobs/import HTTP/1.1
Content-Type: application/json
Accept: application/json
Host: pinpoint.us-east-1.amazonaws.com
X-Amz-Date: 20180605T214912Z
Authorization: AWS4-HMAC-SHA256 Credential=AKIAIOSFODNN7EXAMPLE/20180605/
us-east-1/mobiletargeting/aws4_request, SignedHeaders=accept;cache-
control;content-length;content-type;host;postman-token;x-amz-date,
 Signature=c25cbd6bf61bd3b3667c571ae764b9bf2d8af61b875cacced95d1e68d91b4170
Cache-Control: no-cache

{
  "S3Url": "s3://bucket-name/prefix/key",
  "RoleArn": "iam-import-role-arn",
  "Format": "format",
  "RegisterEndpoints": true
}
```

Where:

- *application-id* is the ID of the Amazon Pinpoint project that you're importing endpoints for.

- *bucket-name/prefix/key* is the location in Amazon S3 that contains one or more objects to import. The location can end with the key for an individual object, or it can end with a prefix that qualifies multiple objects.
- *iam-import-role-arn* is the Amazon Resource Name (ARN) of an IAM role that grants Amazon Pinpoint read access to the bucket.
- *format* can be either `JSON` or `CSV`, depending on which format you used to define your endpoints. If the Amazon S3 location includes multiple files of mixed formats, Amazon Pinpoint imports only the files that match the specified format.

If your request succeeds, you receive a response like the following:

```
{
    "Id": "a995ce5d70fa44adb563b7d0e3f6c6f5",
    "JobStatus": "CREATED",
    "CreationDate": "2018-06-05T21:49:15.288Z",
    "Type": "IMPORT",
    "Definition": {
        "S3Url": "s3://bucket-name/prefix/key",
        "RoleArn": "iam-import-role-arn",
        "ExternalId": "external-id",
        "Format": "JSON",
        "RegisterEndpoints": true,
        "DefineSegment": false
    }
}
```

The response provides the job ID with the `Id` attribute. You can use this ID to check the current status of the import job.

Example GET Import Job Request

To check the current status of an import job, issue a `GET` request to the Import Job resource:

```
GET /v1/apps/application_id/jobs/import/job_id HTTP/1.1
Content-Type: application/json
Accept: application/json
Host: pinpoint.us-east-1.amazonaws.com
X-Amz-Date: 20180605T220744Z
Authorization: AWS4-HMAC-SHA256 Credential=AKIAIOSFODNN7EXAMPLE/20180605/
us-east-1/mobiletargeting/aws4_request, SignedHeaders=accept;cache-
control;content-type;host;postman-token;x-amz-date,
 Signature=c25cbd6bf61bd3b3667c571ae764b9bf2d8af61b875cacced95d1e68d91b4170
Cache-Control: no-cache
```

Where:

- *application_id* is the ID of the Amazon Pinpoint project for which the import job was initiated.
- *job_id* is the ID of the import job that you're checking.

If your request succeeds, you receive a response like the following:

```
{
    "ApplicationId": "application_id",
    "Id": "70a51b2cf442447492d2c8e50336a9e8",
    "JobStatus": "COMPLETED",
    "CompletedPieces": 1,
    "FailedPieces": 0,
    "TotalPieces": 1,
    "CreationDate": "2018-06-05T22:04:49.213Z",
    "CompletionDate": "2018-06-05T22:04:58.034Z",
    "Type": "IMPORT",
    "TotalFailures": 0,
    "TotalProcessed": 3,
    "Definition": {
        "S3Url": "s3://bucket-name/prefix/key.json",
        "RoleArn": "iam-import-role-arn",
        "ExternalId": "external-id",
        "Format": "JSON",
        "RegisterEndpoints": true,
        "DefineSegment": false
    }
}
```

The response provides the job status with the `JobStatus` attribute.

Related Information

For more information about the Import Jobs resource in the Amazon Pinpoint API, including the supported HTTP methods and request parameters, see Import Jobs in the *Amazon Pinpoint API Reference*.

Deleting Endpoints from Amazon Pinpoint

You can delete endpoints when you no longer want to message a certain destination—such as when the destination becomes unreachable, or when a customer closes an account.

Examples

The following examples show you how to delete an endpoint.

AWS CLI

You can use Amazon Pinpoint by running commands with the AWS CLI.

Example Delete Endpoint Command

To delete an endpoint, use the `delete-endpoint` command:

```
$ aws pinpoint delete-endpoint \
> --application-id application-id \
> --endpoint-id endpoint-id
```

Where:

- *application-id* is the ID of the Amazon Pinpoint project that contains the endpoint.
- *endpoint-id* is the ID of the endpoint that you're deleting.

The response to this command is the JSON definition of the endpoint that you deleted.

AWS SDK for Java

You can use the Amazon Pinpoint API in your Java applications by using the client that's provided by the AWS SDK for Java.

Example Code

To delete an endpoint, use the `deleteEndpoint` method of the `AmazonPinpoint` client. Provide a `DeleteEndpointRequest` object as the method argument:

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.pinpoint.AmazonPinpoint;
import com.amazonaws.services.pinpoint.AmazonPinpointClientBuilder;
import com.amazonaws.services.pinpoint.model.DeleteEndpointRequest;
import com.amazonaws.services.pinpoint.model.DeleteEndpointResult;

import java.util.Arrays;

public class DeleteEndpoints {

    public static void main(String[] args) {

        final String USAGE = "\n" +
                "DeleteEndpoints - Removes one or more endpoints from an " +
                "Amazon Pinpoint application.\n\n" +

                "Usage: DeleteEndpoints <applicationId> <endpointId1>
 [endpointId2 ...]\n";

        if (args.length < 2) {
            System.out.println(USAGE);
            System.exit(1);
        }

        String applicationId = args[0];
        String[] endpointIds = Arrays.copyOfRange(args, 1, args.length);

        // Initializes the Amazon Pinpoint client.
        AmazonPinpoint pinpointClient = AmazonPinpointClientBuilder.standard()
                .withRegion(Regions.US_EAST_1).build();

        try {
            // Deletes each of the specified endpoints with the Amazon Pinpoint client.
            for (String endpointId: endpointIds) {
                DeleteEndpointResult result =
                        pinpointClient.deleteEndpoint(new DeleteEndpointRequest()
                        .withEndpointId(endpointId)
                        .withApplicationId(applicationId));
                System.out.format("Deleted endpoint %s.\n",
 result.getEndpointResponse().getId());
            }
        } catch (AmazonServiceException e) {
            System.err.println(e.getErrorMessage());
            System.exit(1);
        }
    }
}
```

HTTP

You can use Amazon Pinpoint by making HTTP requests directly to the REST API.

Example DELETE Endpoint Request

To delete an endpoint, issue a `DELETE` request to the Endpoint resource:

```
DELETE /v1/apps/application-id/endpoints/endpoint-id HTTP/1.1
```

```
Host: pinpoint.us-east-1.amazonaws.com
Content-Type: application/json
Accept: application/json
Cache-Control: no-cache
```

Where:

- *application-id* is the ID of the Amazon Pinpoint project that contains the endpoint.
- *endpoint-id* is the ID of the endpoint that you're deleting.

The response to this request is the JSON definition of the endpoint that you deleted.

Accessing Audience Data in Amazon Pinpoint

As you add endpoints to Amazon Pinpoint, it grows as a repository of audience data. This data consists of:

- The endpoints that you add or update by using the Amazon Pinpoint API.
- The endpoints that your client code adds or updates as users come to your application.

As your audience grows and changes, so does your endpoint data. To view the latest information that Amazon Pinpoint has about your audience, you can look up endpoints individually, or you can export all of the endpoints for an Amazon Pinpoint project. By viewing your endpoint data, you can learn about the audience characteristics that you record in your endpoints, such as:

- Your users' devices and platforms.
- Your users' time zones.
- The versions of your app that are installed on users' devices.
- Your users' locations, such as their cities or countries.
- Any custom attributes or metrics that you record.

The Amazon Pinpoint console also provides analytics for the demographics and custom attributes that are captured in your endpoints.

Before you can look up endpoints, you must add them to your Amazon Pinpoint project. To add endpoints, see Defining Your Audience to Amazon Pinpoint (p. 40).

Use the topics in this section to look up or export endpoints by using the Amazon Pinpoint API.

Topics

Looking Up Endpoints with Amazon Pinpoint

You can look up the details for any individual endpoint that was added to an Amazon Pinpoint project. These details can include the destination address for your messages, the messaging channel, data about the user's device, data about the user's location, and any custom attributes that you record in your endpoints.

To look up an endpoint, you need the endpoint ID. If you don't know the ID, you can get the endpoint data by exporting instead. To export endpoints, see the section called “Exporting Endpoints” (p. 69).

Examples

The following examples show you how to look up an individual endpoint by specifying its ID.

AWS CLI

You can use Amazon Pinpoint by running commands with the AWS CLI.

Example Get Endpoint Command

To look up an endpoint, use the `get-endpoint` command:

```
$ aws pinpoint get-endpoint \
> --application-id application-id \
> --endpoint-id endpoint-id
```

Where:

- *application-id* is the ID of the Amazon Pinpoint project that contains the endpoint.
- *endpoint-id* is the ID of the endpoint that you're looking up.

The response to this command is the JSON definition of the endpoint, as in the following example:

```
{
    "EndpointResponse": {
        "Address": "1a2b3c4d5e6f7g8h9i0j1k2l3m4n5o6p7q8r9s0t1u2v3w4x5y6z7a8b9c0d1e2f",
        "ApplicationId": "application-id",
        "Attributes": {
            "interests": [
                "technology",
                "music",
                "travel"
            ]
        },
        "ChannelType": "APNS",
        "CohortId": "63",
        "CreationDate": "2018-05-01T17:31:01.046Z",
        "Demographic": {
            "AppVersion": "1.0",
            "Make": "apple",
            "Model": "iPhone",
            "ModelVersion": "8",
            "Platform": "ios",
            "PlatformVersion": "11.3.1",
            "Timezone": "America/Los_Angeles"
        },
        "EffectiveDate": "2018-05-07T19:03:29.963Z",
        "EndpointStatus": "ACTIVE",
        "Id": "example_endpoint",
        "Location": {
            "City": "Seattle",
            "Country": "US",
            "Latitude": 47.6,
            "Longitude": -122.3,
            "PostalCode": "98121"
        },
        "Metrics": {
            "music_interest_level": 6.0,
            "travel_interest_level": 4.0,
            "technology_interest_level": 9.0
        },
        "OptOut": "ALL",
        "RequestId": "7f546cac-6858-11e8-adcd-2b5a07aab338",
        "User": {
            "UserAttributes": {
                "gender": [
                    "female"
                ],
                "name": [
                    "Wang",
                    "Xiulan"
                ],
                "age": [
                    "39"
                ]
            },
            "UserId": "example_user"
        }
    }
}
```

AWS SDK for Java

You can use the Amazon Pinpoint API in your Java applications by using the client that's provided by the AWS SDK for Java.

Example Code

To look up an endpoint, initialize a `GetEndpointRequest` object. Then, pass this object to the `getEndpoint` method of the `AmazonPinpoint` client:

```
import com.amazonaws.regions.Regions;
import com.amazonaws.services.pinpoint.AmazonPinpoint;
import com.amazonaws.services.pinpoint.AmazonPinpointClientBuilder;
import com.amazonaws.services.pinpoint.model.EndpointResponse;
import com.amazonaws.services.pinpoint.model.GetEndpointRequest;
import com.amazonaws.services.pinpoint.model.GetEndpointResult;
import com.google.gson.FieldNamingPolicy;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;

public class LookUpEndpoint {

    public static void main(String[] args) {

        final String USAGE = "\n" +
                "LookUpEndpoint - Prints the definition of the endpoint that has the
 specified ID." +
                "Usage: LookUpEndpoint <applicationId> <endpointId>\n\n" +

                "Where:\n" +
                "  applicationId - The ID of the Amazon Pinpoint application that has
 the " +
                "endpoint." +
                "  endpointId - The ID of the endpoint ";

        if (args.length < 1) {
            System.out.println(USAGE);
            System.exit(1);
        }

        String applicationId = args[0];
        String endpointId = args[1];

        // Specifies the endpoint that the Amazon Pinpoint client looks up.
        GetEndpointRequest request = new GetEndpointRequest()
                .withEndpointId(endpointId)
                .withApplicationId(applicationId);

        // Initializes the Amazon Pinpoint client.
        AmazonPinpoint pinpointClient = AmazonPinpointClientBuilder.standard()
                .withRegion(Regions.US_EAST_1).build();

        // Uses the Amazon Pinpoint client to get the endpoint definition.
        GetEndpointResult result = pinpointClient.getEndpoint(request);
        EndpointResponse endpoint = result.getEndpointResponse();

        // Uses the Google Gson library to pretty print the endpoint JSON.
        Gson gson = new GsonBuilder()
                .setFieldNamingPolicy(FieldNamingPolicy.UPPER_CAMEL_CASE)
                .setPrettyPrinting()
                .create();
        String endpointJson = gson.toJson(endpoint);

        System.out.println(endpointJson);
    }
}
```

To print the endpoint data in a readable format, this example uses the Google GSON library to convert the EndpointResponse object into a JSON string.

HTTP

You can use Amazon Pinpoint by making HTTP requests directly to the REST API.

Example GET Endpoint Request

To look up an endpoint, issue a `GET` request to the Endpoint resource:

```
GET /v1/apps/application_id/endpoints/endpoint_id HTTP/1.1
Host: pinpoint.us-east-1.amazonaws.com
Content-Type: application/json
Accept: application/json
Cache-Control: no-cache
```

Where:

- *application-id* is the ID of the Amazon Pinpoint project that contains the endpoint.
- *endpoint-id* is the ID of the endpoint that you're looking up.

The response to this request is the JSON definition of the endpoint, as in the following example:

```
{
    "ChannelType": "APNS",
    "Address": "1a2b3c4d5e6f7g8h9i0j1k2l3m4n5o6p7q8r9s0t1u2v3w4x5y6z7a8b9c0d1e2f",
    "EndpointStatus": "ACTIVE",
    "OptOut": "NONE",
    "RequestId": "b720cfa8-6924-11e8-aeda-0b22e0b0fa59",
    "Location": {
        "Latitude": 47.6,
        "Longitude": -122.3,
        "PostalCode": "98121",
        "City": "Seattle",
        "Country": "US"
    },
    "Demographic": {
        "Make": "apple",
        "Model": "iPhone",
        "ModelVersion": "8",
        "Timezone": "America/Los_Angeles",
        "AppVersion": "1.0",
        "Platform": "ios",
        "PlatformVersion": "11.3.1"
    },
    "EffectiveDate": "2018-06-06T00:58:19.865Z",
    "Attributes": {
        "interests": [
            "technology",
            "music",
            "travel"
        ]
    },
    "Metrics": {
        "music_interest_level": 6,
        "travel_interest_level": 4,
        "technology_interest_level": 9
    },
    "User": {},
    "ApplicationId": "application_id",
    "Id": "example_endpoint",
    "CohortId": "39",
    "CreationDate": "2018-06-06T00:58:19.865Z"
}
```

Related Information

For more information about the Endpoint resource in the Amazon Pinpoint API, see Endpoint in the *Amazon Pinpoint API Reference.*

Exporting Endpoints from Amazon Pinpoint

To get all of the information that Amazon Pinpoint has about your audience, you can export the endpoint definitions that belong to a project. When you export, Amazon Pinpoint places the endpoint definitions in an Amazon S3 bucket that you specify. Exporting endpoints is useful when you want to:

- View the latest data about new and existing endpoints that your client application registered with Amazon Pinpoint.
- Synchronize the endpoint data in Amazon Pinpoint with your own Customer Relationship Management (CRM) system.
- Create reports about or analyze your customer data.

Before You Begin

Before you can export endpoints, you need the following resources in your AWS account:

- An Amazon S3 bucket. To create a bucket, see Create a Bucket in the *Amazon Simple Storage Service Getting Started Guide*.
- An AWS Identity and Access Management (IAM) role that grants Amazon Pinpoint write permissions for your Amazon S3 bucket. To create the role, see IAM Role for Exporting Endpoints or Segments (p. 128).

Examples

The following examples demonstrate how to export endpoints from an Amazon Pinpoint project, and then download those endpoints from your Amazon S3 bucket.

AWS CLI

You can use Amazon Pinpoint by running commands with the AWS CLI.

Example Create Export Job Command

To export the endpoints in your Amazon Pinpoint project, use the `create-export-job` command:

```
$ aws pinpoint create-export-job \
> --application-id application-id \
> --export-job-request \
> S3UrlPrefix=s3://bucket-name/prefix/,\
> RoleArn=iam-export-role-arn
```

Where:

- *application-id* is the ID of the Amazon Pinpoint project that contains the endpoints.
- *bucket-name/prefix/* is the name of your Amazon S3 bucket and, optionally, a prefix that helps you organize the objects in your bucket hierarchically. For example, a useful prefix might be `pinpoint/exports/endpoints/`.
- *iam-export-role-arn* is the Amazon Resource Name (ARN) of an IAM role that grants Amazon Pinpoint write access to the bucket.

The response to this command provides details about the export job:

```
{
    "ExportJobResponse": {
        "CreationDate": "2018-06-04T22:04:20.585Z",
        "Definition": {
            "RoleArn": "iam-export-role-arn",
            "S3UrlPrefix": "s3://s3-bucket-name/prefix/"
        },
        "Id": "7390e0de8e0b462380603c5a4df90bc4",
        "JobStatus": "CREATED",
        "Type": "EXPORT"
    }
}
```

The response provides the job ID with the `Id` attribute. You can use this ID to check the current status of the export job.

Example Get Export Job Command

To check the current status of an export job, use the `get-export-job` command:

```
$ aws pinpoint get-export-job \
> --application-id application-id \
> --job-id job-id
```

Where:

- *application-id* is the ID the Amazon Pinpoint project that you exported the endpoints from.
- *job-id* is the ID of the job that you're checking.

The response to this command provides the current state of the export job:

```
{
    "ExportJobResponse": {
        "ApplicationId": "application-id",
        "CompletedPieces": 1,
        "CompletionDate": "2018-05-08T22:16:48.228Z",
        "CreationDate": "2018-05-08T22:16:44.812Z",
        "Definition": {},
        "FailedPieces": 0,
        "Id": "6c99c463f14f49caa87fa27a5798bef9",
        "JobStatus": "COMPLETED",
        "TotalFailures": 0,
        "TotalPieces": 1,
        "TotalProcessed": 215,
        "Type": "EXPORT"
    }
}
```

The response provides the job status with the `JobStatus` attribute. When the job status value is `COMPLETED`, you can get your exported endpoints from your Amazon S3 bucket.

Example S3 CP Command

To download your exported endpoints, use the Amazon S3 `cp` command:

```
$ aws s3 cp s3://bucket-name/prefix/key.gz /local/directory/
```

Where:

- *bucket-name/prefix/key* is the location of the .gz file that Amazon Pinpoint added to your bucket when you exported your endpoints. This file contains the exported endpoint definitions.
- */local/directory/* is the file path to the local directory that you want to download the endpoints to.

AWS SDK for Java

You can use the Amazon Pinpoint API in your Java applications by using the client that's provided by the AWS SDK for Java.

Example Code

To export endpoints from an Amazon Pinpoint project, initialize a `CreateExportJobRequest` object. Then, pass this object to the `createExportJob` method of the `AmazonPinpoint` client.

To download the exported endpoints from Amazon Pinpoint, use the `getObject` method of the `AmazonS3` client.

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.pinpoint.AmazonPinpoint;
import com.amazonaws.services.pinpoint.AmazonPinpointClientBuilder;
import com.amazonaws.services.pinpoint.model.*;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.S3Object;
import com.amazonaws.services.s3.model.S3ObjectInputStream;
import com.amazonaws.services.s3.model.S3ObjectSummary;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.Date;
import java.util.List;
import java.util.concurrent.TimeUnit;
import java.util.stream.Collectors;

public class ExportEndpoints {

    public static void main(String[] args) {

        final String USAGE = "\n" +
                "ExportEndpoints - Downloads endpoints from an Amazon Pinpoint
 application by: \n" +
                "1.) Exporting the endpoint definitions to an Amazon S3 bucket. \n" +
                "2.) Downloading the endpoint definitions to the specified file path.\n
\n" +

                "Usage: ExportEndpoints <s3BucketName> <iamExportRoleArn>
 <downloadDirectory> " +
                "<applicationId>\n\n" +

                "Where:\n" +
                "  s3BucketName - The name of the Amazon S3 bucket to export the
 endpoints files " +
                "to. If the bucket doesn't exist, a new bucket is created.\n" +
                "  iamExportRoleArn - The ARN of an IAM role that grants Amazon
 Pinpoint write " +
                "permissions to the S3 bucket.\n" +
                "  downloadDirectory - The directory to download the endpoints files
 to.\n" +
                "  applicationId - The ID of the Amazon Pinpoint application that has
 the " +
                "endpoints.";

        if (args.length < 1) {
            System.out.println(USAGE);
            System.exit(1);
        }

        String s3BucketName = args[0];
        String iamExportRoleArn = args[1];
        String downloadDirectory = args[2];
        String applicationId = args[3];

        // Exports the endpoints to Amazon S3 and stores the keys of the new objects.
        List<String> objectKeys =
                exportEndpointsToS3(s3BucketName, iamExportRoleArn, applicationId);

        // Filters the keys to only those objects that have the endpoint definitions.
        // These objects have the .gz extension.
        List<String> endpointFileKeys = objectKeys
                .stream()
                .filter(o -> o.endsWith(".gz"))
                .collect(Collectors.toList());

        // Downloads the exported endpoints files to the specified directory.
        downloadFromS3(s3BucketName, endpointFileKeys, downloadDirectory);
    }

    public static List<String> exportEndpointsToS3(String s3BucketName, String
 iamExportRoleArn,
                                                   String applicationId) {

        // The S3 path that Amazon Pinpoint exports the endpoints to.
        SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd-
HH_mm:ss.SSS_z");
```

```
       String endpointsKeyPrefix = "exports/" + applicationId + "_" +
dateFormat.format(new Date
               ());
       String s3UrlPrefix = "s3://" + s3BucketName + "/" + endpointsKeyPrefix + "/";

       // Defines the export job that Amazon Pinpoint runs.
       ExportJobRequest exportJobRequest = new ExportJobRequest()
               .withS3UrlPrefix(s3UrlPrefix)
               .withRoleArn(iamExportRoleArn);
       CreateExportJobRequest createExportJobRequest = new CreateExportJobRequest()
               .withApplicationId(applicationId)
               .withExportJobRequest(exportJobRequest);

       // Initializes the Amazon Pinpoint client.
       AmazonPinpoint pinpointClient = AmazonPinpointClientBuilder.standard()
               .withRegion(Regions.US_EAST_1).build();

       System.out.format("Exporting endpoints from Amazon Pinpoint application %s to
Amazon S3 " +
               "bucket %s . . .\n", applicationId, s3BucketName);

       List<String> objectKeys = null;

       try {
           // Runs the export job with Amazon Pinpoint.
           CreateExportJobResult exportResult =
                   pinpointClient.createExportJob(createExportJobRequest);

           // Prints the export job status to the console while the job runs.
           String jobId = exportResult.getExportJobResponse().getId();
           printExportJobStatus(pinpointClient, applicationId, jobId);

           // Initializes the Amazon S3 client.
           AmazonS3 s3Client = AmazonS3ClientBuilder.defaultClient();

           // Lists the objects created by Amazon Pinpoint.
           objectKeys = s3Client
                   .listObjectsV2(s3BucketName, endpointsKeyPrefix)
                   .getObjectSummaries()
                   .stream()
                   .map(S3ObjectSummary::getKey)
                   .collect(Collectors.toList());

       } catch (AmazonServiceException e) {
           System.err.println(e.getMessage());
           System.exit(1);
       }

       return objectKeys;
   }

   private static void printExportJobStatus(AmazonPinpoint pinpointClient,
                                            String applicationId, String jobId) {

       GetExportJobResult getExportJobResult;
       String jobStatus;

       try {
           // Checks the job status until the job completes or fails.
           do {
               getExportJobResult = pinpointClient.getExportJob(new
GetExportJobRequest()
                       .withJobId(jobId)
                       .withApplicationId(applicationId));
               jobStatus = getExportJobResult.getExportJobResponse().getJobStatus();
               System.out.format("Export job %s . . .\n", jobStatus.toLowerCase());
```

```
                TimeUnit.SECONDS.sleep(3);
            } while (!jobStatus.equals("COMPLETED") && !jobStatus.equals("FAILED"));

            if (jobStatus.equals("COMPLETED")) {
                System.out.println("Finished exporting endpoints.");
            } else {
                System.err.println("Failed to export endpoints.");
                System.exit(1);
            }

            // Checks for entries that failed to import.
            // getFailures provides up to 100 of the first failed entries for the job,
if any exist.
            List<String> failedEndpoints =
getExportJobResult.getExportJobResponse().getFailures();
            if (failedEndpoints != null) {
                System.out.println("Failed to import the following entries:");
                for (String failedEndpoint : failedEndpoints) {
                    System.out.println(failedEndpoint);
                }
            }
        } catch (AmazonServiceException | InterruptedException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void downloadFromS3(String s3BucketName, List<String> objectKeys,
                                      String downloadDirectory) {

        // Initializes the Amazon S3 client.
        AmazonS3 s3Client = AmazonS3ClientBuilder.defaultClient();

        try {
            // Downloads each object to the specified file path.
            for (String key : objectKeys) {
                S3Object object = s3Client.getObject(s3BucketName, key);
                String endpointsFileName = key.substring(key.lastIndexOf("/"));
                Path filePath = Paths.get(downloadDirectory + endpointsFileName);

                System.out.format("Downloading %s to %s . . .\n",
                        filePath.getFileName(), filePath.getParent());

                writeObjectToFile(filePath, object);
            }
            System.out.println("Download finished.");
        } catch (AmazonServiceException | NullPointerException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }

    }

    private static void writeObjectToFile(Path filePath, S3Object object) {

        // Writes the contents of the S3 object to a file.
        File endpointsFile = new File(filePath.toAbsolutePath().toString());
        try (FileOutputStream fos = new FileOutputStream(endpointsFile);
             S3ObjectInputStream s3is = object.getObjectContent()) {
            byte[] read_buf = new byte[1024];
            int read_len = 0;
            while ((read_len = s3is.read(read_buf)) > 0) {
                fos.write(read_buf, 0, read_len);
            }
        } catch (IOException e) {
            System.err.println(e.getMessage());
```

```
                System.exit(1);
            }
        }
}
```

HTTP

You can use Amazon Pinpoint by making HTTP requests directly to the REST API.

Example POST Export Job Request

To export the endpoints in your Amazon Pinpoint project, issue a `POST` request to the Export Jobs resource:

```
POST /v1/apps/application_id/jobs/export HTTP/1.1
Content-Type: application/json
Accept: application/json
Host: pinpoint.us-east-1.amazonaws.com
X-Amz-Date: 20180606T001238Z
Authorization: AWS4-HMAC-SHA256 Credential=AKIAIOSFODNN7EXAMPLE/20180606/
us-east-1/mobiletargeting/aws4_request, SignedHeaders=accept;cache-
control;content-length;content-type;host;postman-token;x-amz-date,
 Signature=c25cbd6bf61bd3b3667c571ae764b9bf2d8af61b875cacced95d1e68d91b4170
Cache-Control: no-cache

{
  "S3UrlPrefix": "s3://bucket-name/prefix",
  "RoleArn": "iam-export-role-arn"
}
```

Where:

- *`application-id`* is the ID of the Amazon Pinpoint project that contains the endpoints.
- *`bucket-name/prefix/`* is the name of your Amazon S3 bucket and, optionally, a prefix that helps you organize the objects in your bucket hierarchically. For example, a useful prefix might be `pinpoint/exports/endpoints/`.
- *`iam-export-role-arn`* is the Amazon Resource Name (ARN) of an IAM role that grants Amazon Pinpoint write access to the bucket.

The response to this request provides details about the export job:

```
{
    "Id": "611bdc54c75244bfa51fe7001ddb2e36",
    "JobStatus": "CREATED",
    "CreationDate": "2018-06-06T00:12:43.271Z",
    "Type": "EXPORT",
    "Definition": {
        "S3UrlPrefix": "s3://bucket-name/prefix",
        "RoleArn": "iam-export-role-arn"
    }
}
```

The response provides the job ID with the `Id` attribute. You can use this ID to check the current status of the export job.

Example GET Export Job Request

To check the current status of an export job, issue a `GET` request to the Export Job resource:

```
GET /v1/apps/application_id/jobs/export/job_id HTTP/1.1
Content-Type: application/json
Accept: application/json
Host: pinpoint.us-east-1.amazonaws.com
X-Amz-Date: 20180606T002443Z
Authorization: AWS4-HMAC-SHA256 Credential=AKIAIOSFODNN7EXAMPLE/20180606/
us-east-1/mobiletargeting/aws4_request, SignedHeaders=accept;cache-
control;content-type;host;postman-token;x-amz-date,
 Signature=c25cbd6bf61bd3b3667c571ae764b9bf2d8af61b875cacced95d1e68d91b4170
Cache-Control: no-cache
```

Where:

- `application-id` is the ID the Amazon Pinpoint project that you exported the endpoints from.
- `job-id` is the ID of the job that you're checking.

The response to this request provides the current state of the export job:

```
{
    "ApplicationId": "application_id",
    "Id": "job_id",
    "JobStatus": "COMPLETED",
    "CompletedPieces": 1,
    "FailedPieces": 0,
    "TotalPieces": 1,
    "CreationDate": "2018-06-06T00:12:43.271Z",
    "CompletionDate": "2018-06-06T00:13:01.141Z",
    "Type": "EXPORT",
    "TotalFailures": 0,
    "TotalProcessed": 217,
    "Definition": {}
}
```

The response provides the job status with the `JobStatus` attribute. When the job status value is `COMPLETED`, you can get your exported endpoints from your Amazon S3 bucket.

Related Information

For more information about the Export Jobs resource in the Amazon Pinpoint API, including the supported HTTP methods and request parameters, see Export Jobs in the *Amazon Pinpoint API Reference*.

Listing Endpoint IDs with Amazon Pinpoint

To update or delete an endpoint, you need the endpoint ID. So, if you want to perform these operations on all of the endpoints in an Amazon Pinpoint project, the first step is to list all of the endpoint IDs that belong to that project. Then, you can iterate over these IDs to, for example, add an attribute globally or delete all of the endpoints in your project.

The following example uses the AWS SDK for Java and does the following:

1. Calls the example `exportEndpointsToS3` method from the example code in Exporting Endpoints from Amazon Pinpoint (p. 69). This method exports the endpoint definitions from an Amazon Pinpoint project. The endpoint definitions are added as gzip files to an Amazon S3 bucket.
2. Downloads the exported gzip files.

3. Reads the gzip files and obtains the endpoint ID from each endpoint's JSON definition.
4. Prints the endpoint IDs to the console.
5. Cleans up by deleting the files that Amazon Pinpoint added to Amazon S3.

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.DeleteObjectsRequest;
import com.amazonaws.services.s3.model.S3Object;
import com.google.gson.FieldNamingPolicy;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.google.gson.JsonObject;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;
import java.util.zip.GZIPInputStream;

public class ListEndpointIds {

    public static void main(String[] args) {

        final String USAGE = "\n" +
                "ListEndpointIds - Prints all of the endpoint IDs that belong to an Amazon
 " +
                "Pinpoint application. This program performs the following steps:\n\n" +

                "1) Exports the endpoints to an Amazon S3 bucket.\n" +
                "2) Downloads the exported endpoints files from Amazon S3.\n" +
                "3) Parses the endpoints files to obtain the endpoint IDs and prints them.
\n" +
                "4) Cleans up by deleting the objects that Amazon Pinpoint created in the
 S3 " +
                "bucket.\n\n" +

                "Usage: ListEndpointIds <applicationId> <s3BucketName> <iamExportRoleArn>\n
\n" +

                "Where:\n" +
                "  applicationId - The ID of the Amazon Pinpoint application that has the "
 +
                "endpoint.\n" +
                "  s3BucketName - The name of the Amazon S3 bucket to export the JSON file
 to. If" +
                " the bucket doesn't exist, a new bucket is created.\n" +
                "  iamExportRoleArn - The ARN of an IAM role that grants Amazon Pinpoint
 write " +
                "permissions to the S3 bucket.";

        if (args.length < 1) {
            System.out.println(USAGE);
            System.exit(1);
        }

        String applicationId = args[0];
        String s3BucketName = args[1];
        String iamExportRoleArn = args[2];

        // Exports the endpoints to Amazon S3 and stores the keys of the new objects.
```

```
       List<String> objectKeys =
               ExportEndpoints.exportEndpointsToS3(s3BucketName, iamExportRoleArn,
applicationId);

       // Filters the keys to only those objects that have the endpoint definitions.
       // These objects have the .gz extension.
       List<String> endpointFileKeys = objectKeys
               .stream()
               .filter(o -> o.endsWith(".gz"))
               .collect(Collectors.toList());

       // Gets the endpoint IDs from the exported endpoints files.
       List<String> endpointIds = getEndpointIds(s3BucketName, endpointFileKeys);

       System.out.println("Endpoint IDs:");

       for (String endpointId : endpointIds) {
           System.out.println("\t- " + endpointId);
       }

       // Deletes the objects that Amazon Pinpoint created in the S3 bucket.
       deleteS3Objects(s3BucketName, objectKeys);
   }

   private static List<String> getEndpointIds(String s3bucketName, List<String>
endpointFileKeys) {

       List<String> endpointIds = new ArrayList<>();

       // Initializes the Amazon S3 client.
       AmazonS3 s3Client = AmazonS3ClientBuilder.defaultClient();

       // Gets the endpoint IDs from the exported endpoints files.
       try {
           for (String key : endpointFileKeys) {
               S3Object endpointFile = s3Client.getObject(s3bucketName, key);
               endpointIds.addAll(getEndpointIdsFromFile(endpointFile));
           }
       } catch (AmazonServiceException e) {
           System.err.println(e.getMessage());
           System.exit(1);
       }

       return endpointIds;
   }

   private static List<String> getEndpointIdsFromFile(S3Object endpointsFile) {

       List<String> endpointIdsFromFile = new ArrayList<>();

       // The Google Gson library is used to parse the exported endpoint JSON.
       Gson gson = new GsonBuilder()
               .setFieldNamingPolicy(FieldNamingPolicy.UPPER_CAMEL_CASE)
               .create();

       // Reads each endpoint entry in the file and adds the ID to the list.
       try (GZIPInputStream gzipInputStream =
                    new GZIPInputStream(endpointsFile.getObjectContent());
            BufferedReader reader =
                    new BufferedReader(new InputStreamReader(
                            gzipInputStream, "UTF-8"))) {
           String endpointString;
           while ((endpointString = reader.readLine()) != null) {
               JsonObject endpointJson = gson.fromJson(endpointString, JsonObject.class);
               endpointIdsFromFile.add(endpointJson
                       .getAsJsonPrimitive("Id")
```

```
                        .getAsString());
            }
        } catch (IOException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }

        return endpointIdsFromFile;
    }

    private static void deleteS3Objects(String s3BucketName, List<String> keys) {

        AmazonS3 s3Client = AmazonS3ClientBuilder.defaultClient();

        String[] keysArray = keys.toArray(new String[keys.size()]);
        DeleteObjectsRequest request = new
 DeleteObjectsRequest(s3BucketName).withKeys(keysArray);

        System.out.println("Deleting the following Amazon S3 objects created by Amazon
 Pinpoint:");

        for (String key : keys) {
            System.out.println("\t- " + key);
        }

        try {
            s3Client.deleteObjects(request);
        } catch (AmazonServiceException e) {
            System.err.println(e.getErrorMessage());
            System.exit(1);
        }

        System.out.println("Finished deleting objects.");
    }
}
```

Creating Segments

A user *segment* represents a subset of your users based on shared characteristics, such as how recently the users have used your app or which device platform they use. A segment designates which users receive the messages delivered by a campaign. Define segments so that you can reach the right audience when you want to invite users back to your app, make special offers, or otherwise increase user engagement and purchasing.

After you create a segment, you can use it in one or more campaigns. A campaign delivers tailored messages to the users in the segment.

For more information, see Segments.

Topics

Building Segments

To reach the intended audience for a campaign, build a segment based on the data reported by your app. For example, to reach users who haven't used your app recently, you can define a segment for users who haven't used your app in the last 7 days.

Building Segments With the AWS SDK for Java

The following example demonstrates how to build a segment with the AWS SDK for Java.

```
import com.amazonaws.services.pinpoint.AmazonPinpointClient;
import com.amazonaws.services.pinpoint.model.AttributeDimension;
import com.amazonaws.services.pinpoint.model.AttributeType;
import com.amazonaws.services.pinpoint.model.CreateSegmentRequest;
import com.amazonaws.services.pinpoint.model.CreateSegmentResult;
import com.amazonaws.services.pinpoint.model.RecencyDimension;
import com.amazonaws.services.pinpoint.model.SegmentBehaviors;
import com.amazonaws.services.pinpoint.model.SegmentDemographics;
import com.amazonaws.services.pinpoint.model.SegmentDimensions;
import com.amazonaws.services.pinpoint.model.SegmentLocation;
import com.amazonaws.services.pinpoint.model.SegmentResponse;
import com.amazonaws.services.pinpoint.model.WriteSegmentRequest;

import java.util.HashMap;
import java.util.Map;

public class PinpointSegmentSample {

    public SegmentResponse createSegment(AmazonPinpointClient client, String appId) {
        Map<String, AttributeDimension> segmentAttributes = new HashMap<>();
        segmentAttributes.put("Team", new
 AttributeDimension().withAttributeType(AttributeType.INCLUSIVE).withValues("Lakers"));

        SegmentBehaviors segmentBehaviors = new SegmentBehaviors();
        SegmentDemographics segmentDemographics = new SegmentDemographics();
        SegmentLocation segmentLocation = new SegmentLocation();
        RecencyDimension recencyDimension = new RecencyDimension();
        recencyDimension.withDuration("DAY_30").withRecencyType("ACTIVE");
        segmentBehaviors.setRecency(recencyDimension);

        SegmentDimensions dimensions = new SegmentDimensions()
                .withAttributes(segmentAttributes)
                .withBehavior(segmentBehaviors)
                .withDemographic(segmentDemographics)
                .withLocation(segmentLocation);


        WriteSegmentRequest writeSegmentRequest = new WriteSegmentRequest()
                .withName("MySegment").withDimensions(dimensions);

        CreateSegmentRequest createSegmentRequest = new CreateSegmentRequest()
                .withApplicationId(appId).withWriteSegmentRequest(writeSegmentRequest);

        CreateSegmentResult createSegmentResult =
 client.createSegment(createSegmentRequest);

        System.out.println("Segment ID: " +
 createSegmentResult.getSegmentResponse().getId());

        return createSegmentResult.getSegmentResponse();
    }

}
```

When you run this example, the following is printed to the console window of your IDE:

```
Segment ID: 09cb2967a82b4a2fbab38fead8d1f4c4
```

Importing Segments

With Amazon Pinpoint, you can define a user segment by importing information about the endpoints that belong to the segment. An *endpoint* is a single messaging destination, such as a mobile push device token, a mobile phone number, or an email address.

Importing segments is useful if you've already created segments of your users outside of Amazon Pinpoint but you want to engage your users with Amazon Pinpoint campaigns.

When you import a segment, Amazon Pinpoint gets the segment's endpoints from Amazon Simple Storage Service (Amazon S3). Before you import, you add the endpoints to Amazon S3, and you create an IAM role that grants Amazon Pinpoint access to Amazon S3. Then, you give Amazon Pinpoint the Amazon S3 location where the endpoints are stored, and Amazon Pinpoint adds each endpoint to the segment.

To create the IAM role, see IAM Role for Importing Endpoints or Segments (p. 127). For information about importing a segment by using the Amazon Pinpoint console, see Importing Segments in the *Amazon Pinpoint User Guide*.

Importing a Segment

The following example demonstrates how to import a segment by using the AWS SDK for Java.

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.pinpoint.AmazonPinpoint;
import com.amazonaws.services.pinpoint.AmazonPinpointClientBuilder;
import com.amazonaws.services.pinpoint.model.*;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.AmazonS3Exception;
import java.io.File;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.List;
import java.util.concurrent.TimeUnit;

public class ImportSegment {

    public static void main(String[] args) {

        final String USAGE = "\n" +
        "ImportSegment - Creates a segment by: \n" +
        "1.) Uploading the endpoint definitions that belong to the segment to an Amazon S3
 bucket. \n" +
        "2.) Importing the endpoint definitions from the bucket to an Amazon Pinpoint
 application." +
        "    Amazon Pinpoint creates a segment that has the specified name.\n\n" +
        "Usage: ImportSegment <endpointsFileLocation> <s3BucketName> <iamImportRoleArn>
 <segmentName> <applicationId>\n\n" +
        "Where:\n" +
        "  endpointsFileLocation - The relative location of the JSON file that contains the
 endpoint definitions.\n" +
        "  s3BucketName - The name of the Amazon S3 bucket to upload the JSON file to. If
 the bucket doesn't " +
        "exist, a new bucket is created.\n" +
        "  iamImportRoleArn - The ARN of an IAM role that grants Amazon Pinpoint read
 permissions so the S3 bucket.\n" +
        "  segmentName - The name for the segment that you are creating or updating." +
        "  applicationId - The ID of the Amazon Pinpoint application to add the endpoints
 to.";

        if (args.length < 1) {
            System.out.println(USAGE);
            System.exit(1);
        }

        String endpointsFileLocation = args[0];
        String s3BucketName = args[1];
        String iamImportRoleArn = args[2];
        String segmentName = args[3];
        String applicationId = args[4];

        Path endpointsFilePath = Paths.get(endpointsFileLocation);
        File endpointsFile = new File(endpointsFilePath.toAbsolutePath().toString());
        uploadToS3(endpointsFile, s3BucketName);

        importSegment(endpointsFile.getName(), s3BucketName, iamImportRoleArn, segmentName,
 applicationId);

    }

    private static void uploadToS3(File endpointsFile, String s3BucketName) {

        // Initializes Amazon S3 client.
        final AmazonS3 s3 = AmazonS3ClientBuilder.defaultClient();

        // Checks whether the specified bucket exists. If not, attempts to create one.
        if (!s3.doesBucketExistV2(s3BucketName)) {
            try {
                s3.createBucket(s3BucketName);
                System.out.format("Created S3 bucket %s.\n", s3BucketName);
            } catch (AmazonS3Exception e) {
                System.err.println(e.getErrorMessage());
                System.exit(1);
            }
        }

        // Uploads the endpoints file to the bucket.
        String endpointsFileName = endpointsFile.getName();
        System.out.format("Uploading %s to S3 bucket %s . . .\n", endpointsFileName,
s3BucketName);
        try {
            s3.putObject(s3BucketName, "imports/" + endpointsFileName, endpointsFile);
            System.out.println("Finished uploading to S3.");
        } catch (AmazonServiceException e) {
            System.err.println(e.getErrorMessage());
            System.exit(1);
        }
    }

    private static void importSegment(String endpointsFileName, String s3BucketName, String
iamImportRoleArn,
                                      String segmentName, String applicationId) {

        // The S3 URL that Amazon Pinpoint requires to find the endpoints file.
        String s3Url = "s3://" + s3BucketName + "/imports/" + endpointsFileName;

        // Defines the import job that Amazon Pinpoint runs.
        ImportJobRequest importJobRequest = new ImportJobRequest()
                .withS3Url(s3Url)
                .withFormat(Format.JSON)
                .withRoleArn(iamImportRoleArn)
                .withRegisterEndpoints(true)
                .withDefineSegment(true)
                .withSegmentName(segmentName);
        CreateImportJobRequest createImportJobRequest = new CreateImportJobRequest()
                .withApplicationId(applicationId)
                .withImportJobRequest(importJobRequest);

        // Initializes the Amazon Pinpoint client.
        AmazonPinpoint pinpointClient = AmazonPinpointClientBuilder.standard()
                .withRegion(Regions.US_EAST_1).build();

        System.out.format("Creating segment %s with the endpoints in %s . . .\n",
segmentName, endpointsFileName);

        try {

            // Runs the import job with Amazon Pinpoint.
            CreateImportJobResult importResult =
pinpointClient.createImportJob(createImportJobRequest);
            String jobId = importResult.getImportJobResponse().getId();

            // Checks the job status until the job completes or fails.
            GetImportJobResult getImportJobResult = null;
            String jobStatus = null;
            do {
                getImportJobResult = pinpointClient.getImportJob(new GetImportJobRequest()
                        .withJobId(jobId)
                        .withApplicationId(applicationId));
                jobStatus = getImportJobResult.getImportJobResponse().getJobStatus();
                System.out.format("Import job %s . . .\n", jobStatus.toLowerCase());
                if (jobStatus.equals("FAILED")) {
                    System.err.println("Failed to import segment.");
                    System.exit(1);
                }
                try {
                    TimeUnit.SECONDS.sleep(3);
                } catch (InterruptedException e) {
                    System.err.println(e.getMessage());
                    System.exit(1);
                }
            } while (!jobStatus.equals("COMPLETED"));

            System.out.println("Finished importing segment.");

            // Checks for entries that failed to import.
            List<String> failedEndpoints =
 getImportJobResult.getImportJobResponse().getFailures();
            if (failedEndpoints != null) {
                System.out.println("Failed to import the following entries:");
                for (String failedEndpoint : failedEndpoints) {
                    System.out.println(failedEndpoint);
                }
            }

        } catch (AmazonServiceException e) {
            System.err.println(e.getErrorMessage());
            System.exit(1);
        }

    }

}
```

Customizing Segments with AWS Lambda

This is prerelease documentation for a feature in public beta release. It is subject to change.

You can use AWS Lambda to tailor how an Amazon Pinpoint campaign engages your target audience. With AWS Lambda, you can modify the campaign's segment at the moment that Amazon Pinpoint delivers the campaign's message.

AWS Lambda is a compute service that you can use to run code without provisioning or managing servers. You package your code and upload it to Lambda as *Lambda functions*. Lambda runs a function when the function is invoked, which might be done manually by you or automatically in response to events.

For more information, see Lambda Functions in the *AWS Lambda Developer Guide*.

To assign a Lambda function to a campaign, you define the campaign's `CampaignHook` settings. These settings include the Lambda function name. They also include the `CampaignHook` mode, which sets whether Amazon Pinpoint receives a return value from the function.

A Lambda function that you assign to a campaign is referred to as an Amazon Pinpoint *extension*.

With the `CampaignHook` settings defined, Amazon Pinpoint automatically invokes the Lambda function when it runs the campaign, before it delivers the campaign's message. When Amazon Pinpoint invokes the function, it provides *event data* about the message delivery. This data includes the campaign's segment, which is the list of endpoints that Amazon Pinpoint sends the message to.

If the `CampaignHook` mode is set to `FILTER`, Amazon Pinpoint allows the function to modify and return the segment before sending the message. For example, the function might update the endpoint definitions with attributes that contain data from a source that is external to Amazon Pinpoint. Or, the function might filter the segment by removing certain endpoints, based on conditions in your function code. After Amazon Pinpoint receives the modified segment from your function, it sends the message to each of the segment's endpoints using the campaign's delivery channel.

By processing your segments with AWS Lambda, you have more control over who you send messages to and what those messages contain. You can tailor your campaigns in real time, at the moment campaign messages are delivered. Filtering segments enables you to engage more narrowly defined subsets of your segments. Adding or updating endpoint attributes enables you to make new data available for message variables.

For more information about message variables, see Message Variables in the *Amazon Pinpoint User Guide*.

> **Note**
> You can also use the `CampaignHook` settings to assign a Lambda function that handles the message delivery. This type of function is useful for delivering messages through custom channels that Amazon Pinpoint doesn't support, such as social media platforms. For more information, see Creating Custom Channels with AWS Lambda (p. 143).

To modify campaign segments with AWS Lambda, first create a function that processes the event data sent by Amazon Pinpoint and returns a modified segment. Then, authorize Amazon Pinpoint to invoke the function by assigning a Lambda function policy. Finally, assign the function to one or more campaigns by defining `CampaignHook` settings.

Event Data

When Amazon Pinpoint invokes your Lambda function, it provides the following payload as the event data:

```
{
  "MessageConfiguration": {Message configuration}
  "ApplicationId": ApplicationId,
  "CampaignId": CampaignId,
  "TreatmentId": TreatmentId,
  "ActivityId": ActivityId,
  "ScheduledTime": Scheduled Time,
  "Endpoints": {
    EndpointId: {Endpoint definition}
    . . .
  }
}
```

The event data is passed to your function code by AWS Lambda.

The event data provides the following attributes:

- `MessageConfiguration` – Has the same structure as the `DirectMessageConfiguration` in the `Messages` resource in the Amazon Pinpoint API.
- `ApplicationId` – The ID of the Amazon Pinpoint project that the campaign belongs to.
- `CampaignId` – The ID of the Amazon Pinpoint project that the function is invoked for.
- `TreatmentId` – The ID of a campaign variation that's used for A/B testing.
- `ActivityId` – The ID of the activity that's being performed by the campaign.
- `ScheduledTime` – The date and time when the campaign's messages will be delivered in ISO 8601 format.
- `Endpoints` – A map that associates endpoint IDs with endpoint definitions. Each event data payload contains up to 50 endpoints. If the campaign segment contains more than 50 endpoints, Amazon Pinpoint invokes the function repeatedly, with up to 50 endpoints at a time, until all endpoints have been processed.

Creating a Lambda Function

To create a Lambda function, see Building Lambda Functions in the *AWS Lambda Developer Guide*.

When you create your function, remember that the message delivery fails in the following conditions:

- The Lambda function takes longer than 15 seconds to return the modified segment.
- Amazon Pinpoint cannot decode the function's return value.
- The function requires more than 3 attempts from Amazon Pinpoint to successfully invoke.

Amazon Pinpoint only accepts endpoint definitions in the function's return value. The function cannot modify other elements in the event data.

Example Lambda Function

Your Lambda function processes the event data sent by Amazon Pinpoint, and it returns the modified endpoints, as shown by the following example handler, written in Node.js:

```
'use strict';

exports.handler = (event, context, callback) => {
    for (var key in event.Endpoints) {
        if (event.Endpoints.hasOwnProperty(key)) {
            var endpoint = event.Endpoints[key];
            var attr = endpoint.Attributes;
            if (!attr) {
                attr = {};
                endpoint.Attributes = attr;
            }
            attr["CreditScore"] = [ Math.floor(Math.random() * 200) + 650];
        }
    }
    console.log("Received event:", JSON.stringify(event, null, 2));
    callback(null, event.Endpoints);
};
```

Lambda passes the event data to the handler as the `event` parameter.

In this example, the handler iterates through each endpoint in the `event.Endpoints` object, and it adds a new attribute, `CreditScore`, to the endpoint. The value of the `CreditScore` attribute is simply a random number.

The `console.log()` statement logs the event in CloudWatch Logs.

The `callback()` statement returns the modified endpoints to Amazon Pinpoint. Normally, the `callback` parameter is optional in Node.js Lambda functions, but it is required in this context because the function must return the updated endpoints to Amazon Pinpoint.

Your function must return endpoints in the same format provided by the event data, which is a map that associates endpoint IDs with endpoint definitions, as in the following example:

```
{
    "eqmj8wpxszeqy/b3vch04sn41yw": {
        "ChannelType": "GCM",
        "Address": "4d5e6f1a2b3c4d5e6f7g8h9i0j1a2b3c",
        "EndpointStatus": "ACTIVE",
        "OptOut": "NONE",
        "Demographic": {
            "Make": "android"
        },
        "EffectiveDate": "2017-11-02T21:26:48.598Z",
        "User": {}
    },
    "idrexqqtn8sbwfex0ouscod0yto": {
        "ChannelType": "APNS",
        "Address": "1a2b3c4d5e6f7g8h9i0j1a2b3c4d5e6f",
        "EndpointStatus": "ACTIVE",
        "OptOut": "NONE",
        "Demographic": {
            "Make": "apple"
        },
        "EffectiveDate": "2017-11-02T21:26:48.598Z",
        "User": {}
    }
}
```

The example function modifies and returns the `event.Endpoints` object that it received in the event data.

In the endpoint definitions that you return, Amazon Pinpoint accepts `TitleOverride` and `BodyOverride` attributes.

Assigning a Lambda Function Policy

Before you can use your Lambda function to process your endpoints, you must authorize Amazon Pinpoint to invoke your Lambda function. To grant invocation permission, assign a *Lambda function policy* to the function. A Lambda function policy is a resource-based permissions policy that designates which entities can use your function and what actions those entities can take.

For more information, see Using Resource-Based Policies for AWS Lambda (Lambda Function Policies) in the *AWS Lambda Developer Guide*.

Example Function Policy

The following policy grants permission to the Amazon Pinpoint service principal to use the `lambda:InvokeFunction` action:

```
{
  "Sid": "sid",
  "Effect": "Allow",
  "Principal": {
    "Service": "pinpoint.us-east-1.amazonaws.com"
  },
  "Action": "lambda:InvokeFunction",
  "Resource": "{arn:aws:lambda:us-east-1:account-id:function:function-name}",
  "Condition": {
    "ArnLike": {
      "AWS:SourceArn": "arn:aws:mobiletargeting:us-east-1:account-id:apps/application-id/
campaigns/campaign-id"
    }
  }
}
```

Your function policy requires a `Condition` block that includes an `AWS:SourceArn` key. This code states which Amazon Pinpoint campaign is allowed to invoke the function. In this example, the policy grants permission to only a single campaign ID. To write a more generic policy, use multi-character match wildcards (*). For example, you can use the following `Condition` block to allow any Amazon Pinpoint campaign in your AWS account to invoke the function:

```
"Condition": {
  "ArnLike": {
    "AWS:SourceArn": "arn:aws:mobiletargeting:us-east-1:account-id:apps/*"
  }
}
```

Granting Amazon Pinpoint Invocation Permission

You can use the AWS Command Line Interface (AWS CLI) to add permissions to the Lambda function policy assigned to your Lambda function. To allow Amazon Pinpoint to invoke a function, use the Lambda `add-permission` command, as shown by the following example:

```
$ aws lambda add-permission \
> --function-name function-name \
> --statement-id sid \
> --action lambda:InvokeFunction \
> --principal pinpoint.us-east-1.amazonaws.com \
> --source-arn arn:aws:mobiletargeting:us-east-1:account-id:apps/application-id/
campaigns/campaign-id
```

If you want to provide a campaign ID for the `--source-arn` parameter, you can look up your campaign IDs by using the Amazon Pinpoint `get-campaigns` command with the AWS CLI. This command requires an `--application-id` parameter. To look up your application IDs, sign in to the Amazon Pinpoint console at https://console.aws.amazon.com/pinpoint/, and go to the **Projects** page. The console shows an **ID** for each project, which is the project's application ID.

When you run the Lambda `add-permission` command, AWS Lambda returns the following output:

```
{
  "Statement": "{\"Sid\":\"sid\",
    \"Effect\":\"Allow\",
    \"Principal\":{\"Service\":\"pinpoint.us-east-1.amazonaws.com\"},
    \"Action\":\"lambda:InvokeFunction\",
    \"Resource\":\"arn:aws:lambda:us-east-1:111122223333:function:function-name\",
    \"Condition\":
      {\"ArnLike\":
        {\"AWS:SourceArn\":
         \"arn:aws:mobiletargeting:us-east-1:111122223333:apps/application-id/campaigns/
campaign-id\"}}}"
}
```

The `Statement` value is a JSON string version of the statement added to the Lambda function policy.

Assigning a Lambda Function to a Campaign

You can assign a Lambda function to an individual Amazon Pinpoint campaign. Or, you can set the Lambda function as the default used by all campaigns for a project, except for those campaigns to which you assign a function individually.

To assign a Lambda function to an individual campaign, use the Amazon Pinpoint API to create or update a `Campaign` object, and define its `CampaignHook` attribute. To set a Lambda function as the default for all campaigns in a project, create or update the `Settings` resource for that project, and define its `CampaignHook` object.

In both cases, set the following `CampaignHook` attributes:

- `LambdaFunctionName` – The name or ARN of the Lambda function that Amazon Pinpoint invokes before sending messages for the campaign.

- `Mode` – Set to `FILTER`. With this mode, Amazon Pinpoint invokes the function and waits for it to return the modified endpoints. After receiving them, Amazon Pinpoint sends the message. Amazon Pinpoint waits for up to 15 seconds before failing the message delivery.

With `CampaignHook` settings defined for a campaign, Amazon Pinpoint invokes the specified Lambda function before sending the campaign's messages. Amazon Pinpoint waits to receive the modified endpoints from the function. If Amazon Pinpoint receives the updated endpoints, it proceeds with the message delivery, using the updated endpoint data.

Creating Campaigns

To help increase engagement between your app and its users, use Amazon Pinpoint to create and manage push notification campaigns that reach out to particular segments of users.

For example, your campaign might invite users back to your app who haven't run it recently or offer special promotions to users who haven't purchased recently.

A campaign sends a tailored message to a user segment that you specify. The campaign can send the message to all users in the segment, or you can allocate a holdout, which is a percentage of users who receive no messages.

You can set the campaign schedule to send the message once or at a recurring frequency, such as once a week. To prevent users from receiving the message at inconvenient times, the schedule can include a quiet time during which no messages are sent.

To experiment with alternative campaign strategies, set up your campaign as an A/B test. An A/B test includes two or more treatments of the message or schedule. Treatments are variations of your message or schedule. As your users respond to the campaign, you can view campaign analytics to compare the effectiveness of each treatment.

For more information, see Campaigns.

Creating Standard Campaigns

A standard campaign sends a custom push notification to a specified segment according to a schedule that you define.

Creating Campaigns With the AWS SDK for Java

The following example demonstrates how to create a campaign with the AWS SDK for Java.

```
import com.amazonaws.services.pinpoint.AmazonPinpointClient;
import com.amazonaws.services.pinpoint.model.Action;
import com.amazonaws.services.pinpoint.model.CampaignResponse;
import com.amazonaws.services.pinpoint.model.CreateCampaignRequest;
import com.amazonaws.services.pinpoint.model.CreateCampaignResult;
import com.amazonaws.services.pinpoint.model.Message;
import com.amazonaws.services.pinpoint.model.MessageConfiguration;
import com.amazonaws.services.pinpoint.model.Schedule;
import com.amazonaws.services.pinpoint.model.WriteCampaignRequest;

import java.util.ArrayList;
import java.util.List;

public class PinpointCampaignSample {

    public CampaignResponse createCampaign(AmazonPinpointClient client, String appId,
 String segmentId) {
        Schedule schedule = new Schedule()
                .withStartTime("IMMEDIATE");

        Message defaultMessage = new Message()
                .withAction(Action.OPEN_APP)
                .withBody("My message body.")
                .withTitle("My message title.");

        MessageConfiguration messageConfiguration = new MessageConfiguration()
                .withDefaultMessage(defaultMessage);

        WriteCampaignRequest request = new WriteCampaignRequest()
                .withDescription("My description.")
                .withSchedule(schedule)
                .withSegmentId(segmentId)
                .withName("MyCampaign")
                .withMessageConfiguration(messageConfiguration);

        CreateCampaignRequest createCampaignRequest = new CreateCampaignRequest()
                .withApplicationId(appId).withWriteCampaignRequest(request);

        CreateCampaignResult result = client.createCampaign(createCampaignRequest);

        System.out.println("Campaign ID: " + result.getCampaignResponse().getId());

        return result.getCampaignResponse();
    }

}
```

When you run this example, the following is printed to the console window of your IDE:

```
Campaign ID: b1c3de717aea4408a75bb3287a906b46
```

Creating A/B Test Campaigns

An A/B test behaves like a standard campaign, but enables you to define different treatments for the campaign message or schedule.

Creating A/B Test Campaigns With the AWS SDK for Java

The following example demonstrates how to create an A/B test campaign with the AWS SDK for Java.

```
import com.amazonaws.services.pinpoint.AmazonPinpointClient;
import com.amazonaws.services.pinpoint.model.Action;
import com.amazonaws.services.pinpoint.model.CampaignResponse;
import com.amazonaws.services.pinpoint.model.CreateCampaignRequest;
import com.amazonaws.services.pinpoint.model.CreateCampaignResult;
import com.amazonaws.services.pinpoint.model.Message;
import com.amazonaws.services.pinpoint.model.MessageConfiguration;
import com.amazonaws.services.pinpoint.model.Schedule;
import com.amazonaws.services.pinpoint.model.WriteCampaignRequest;
import com.amazonaws.services.pinpoint.model.WriteTreatmentResource;

import java.util.ArrayList;
import java.util.List;

public class PinpointCampaignSample {

    public CampaignResponse createAbCampaign(AmazonPinpointClient client, String appId,
 String segmentId) {
        Schedule schedule = new Schedule()
```

```
                .withStartTime("IMMEDIATE");

        // Default treatment.
        Message defaultMessage = new Message()
                .withAction(Action.OPEN_APP)
                .withBody("My message body.")
                .withTitle("My message title.");

        MessageConfiguration messageConfiguration = new MessageConfiguration()
                .withDefaultMessage(defaultMessage);

        // Additional treatments
        WriteTreatmentResource treatmentResource = new WriteTreatmentResource()
                .withMessageConfiguration(messageConfiguration)
                .withSchedule(schedule)
                .withSizePercent(40)
                .withTreatmentDescription("My treatment description.")
                .withTreatmentName("MyTreatment");

        List<WriteTreatmentResource> additionalTreatments = new
 ArrayList<WriteTreatmentResource>();
        additionalTreatments.add(treatmentResource);

        WriteCampaignRequest request = new WriteCampaignRequest()
                .withDescription("My description.")
                .withSchedule(schedule)
                .withSegmentId(segmentId)
                .withName("MyCampaign")
                .withMessageConfiguration(messageConfiguration)
                .withAdditionalTreatments(additionalTreatments)
                .withHoldoutPercent(10); // Hold out of A/B test

        CreateCampaignRequest createCampaignRequest = new CreateCampaignRequest()
                .withApplicationId(appId).withWriteCampaignRequest(request);

        CreateCampaignResult result = client.createCampaign(createCampaignRequest);

        System.out.println("Campaign ID: " + result.getCampaignResponse().getId());

        return result.getCampaignResponse();
    }

}
```

When you run this example, the following is printed to the console window of your IDE:

```
Campaign ID: b1c3de717aea4408a75bb3287a906b46
```

Streaming Amazon Pinpoint Events to Kinesis

The Kinesis platform offers services that you can use to load and analyze streaming data on AWS. You can configure Amazon Pinpoint to send events to Amazon Kinesis Data Firehose or Amazon Kinesis Data Streams. By streaming your events, you enable more flexible options for analysis and storage. For more information, and for instructions on how to set up event streaming in the Amazon Pinpoint console, see Streaming App and Campaign Events with Amazon Pinpoint in the *Amazon Pinpoint User Guide*.

Setting up Event Streaming

The following examples demonstrate how to configure Amazon Pinpoint to automatically send the event data from an app to an Kinesis stream or Kinesis Data Firehose delivery stream.

Prerequisites
These examples require the following input:

- The app ID of an app that is integrated with Amazon Pinpoint and reporting events. For information about how to integrate, see Integrating Amazon Pinpoint with Your Application (p. 3).
- The ARN of an Kinesis stream or Kinesis Data Firehose delivery stream in your AWS account. For information about creating these resources, see Amazon Kinesis Data Streams in the *Amazon Kinesis Data Streams Developer Guide* or Creating an Amazon Kinesis Data Firehose Delivery Stream in the *Amazon Kinesis Data Firehose Developer Guide*.
- The ARN of an AWS Identity and Access Management (IAM) role that authorizes Amazon Pinpoint to send data to the stream. For information about creating a role, see IAM Role for Streaming Events to Kinesis (p. 132).

AWS CLI

The following AWS CLI example uses the `put-event-stream` command. This command configures Amazon Pinpoint to send app and campaign events to an Kinesis stream:

```
aws pinpoint put-event-stream --application-id application-id --write-event-stream
 DestinationStreamArn=stream-arn,RoleArn=role-arn
```

AWS SDK for Java

The following Java example configures Amazon Pinpoint to send app and campaign events to an Kinesis stream:

```
public PutEventStreamResult createEventStream(AmazonPinpoint pinClient, String appId,
                                                          String streamArn, String roleArn)
 {
    WriteEventStream stream = new WriteEventStream()
            .withDestinationStreamArn(streamArn)
            .withRoleArn(roleArn);

    PutEventStreamRequest request = new PutEventStreamRequest()
            .withApplicationId(appId)
            .withWriteEventStream(stream);

    return pinClient.putEventStream(request);
}
```

This example constructs a `WriteEventStream` object that stores the ARNs of the Kinesis stream and the IAM role. The `WriteEventStream` object is passed to a `PutEventStreamRequest` object to configure Amazon Pinpoint to stream events for a specific app. The `PutEventStreamRequest` object is passed to the `putEventStream` method of the Amazon Pinpoint client.

You can assign an Kinesis stream to multiple apps. Amazon Pinpoint will send event data from each app to the stream, enabling you to analyze the data as a collection. The following example method accepts a list of app IDs, and it uses the previous example method, `createEventStream`, to assign a stream to each app:

```
public List<PutEventStreamResult> createEventStreamFromAppList(
        AmazonPinpoint pinClient, List<String> appIDs, String streamArn, String roleArn) {
    return appIDs.stream()
            .map(appId -> createEventStream(pinClient, appId, streamArn, roleArn))
            .collect(Collectors.toList());
}
```

With Amazon Pinpoint, you can assign one stream to multiple apps, but you cannot assign multiple streams to one app.

Disabling Event Streaming

If you assigned an Kinesis stream to an app, you can disable event streaming for that app. Amazon Pinpoint stops streaming the events, but you can view analytics based on the events in the Amazon Pinpoint console.

AWS CLI

Use the `delete-event-stream` command:

```
aws pinpoint delete-event-stream --application-id application-id
```

AWS SDK for Java

Use the `deleteEventStream` method of the Amazon Pinpoint client:

```
pinClient.deleteEventStream(new DeleteEventStreamRequest().withApplicationId(appId));
```

Event Data

After you set up event streaming, Amazon Pinpoint sends each event reported by your application, and each campaign event created by Amazon Pinpoint, as a JSON data object to your Kinesis stream.

The event type is indicated by the `event_type` attribute in the event JSON object.

App Events

After you integrate your app with Amazon Pinpoint and you run one or more campaigns, Amazon Pinpoint streams events about user activity and campaign message deliveries.

Example

The JSON object for an app event contains the data shown in the following example.

```
{
  "event_type": "_session.stop",
  "event_timestamp": 1487973802507,
  "arrival_timestamp": 1487973803515,
  "event_version": "3.0",
  "application": {
    "app_id": "a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6",
    "cognito_identity_pool_id": "us-east-1:a1b2c3d4-e5f6-g7h8-i9j0-k1l2m3n4o5p6",
    "package_name": "main.page",
    "sdk": {
      "name": "aws-sdk-mobile-analytics-js",
      "version": "0.9.1:2.4.8"
    },
    "title": "title",
    "version_name": "1.0",
    "version_code": "1"
  },
  "client": {
    "client_id": "m3n4o5p6-a1b2-c3d4-e5f6-g7h8i9j0k1l2",
    "cognito_id": "us-east-1:i9j0k1l2-m3n4-o5p6-a1b2-c3d4e5f6g7h8"
  },
  "device": {
    "locale": {
      "code": "en_US",
      "country": "US",
      "language": "en"
    },
    "make": "generic web browser",
    "model": "Unknown",
    "platform": {
      "name": "android",
      "version": "10.10"
    }
  },
  "session": {
    "session_id": "f549dea9-1090-945d-c3d1-e496780baac5",
    "start_timestamp": 1487973202531,
    "stop_timestamp": 1487973802507
  },
  "attributes": {},
  "metrics": {}
}
```

Standard App Event Types

Amazon Pinpoint streams the following standard types for app events:

- _campaign.send
- _monetization.purchase

- _session.start
- _session.stop
- _session.pause
- _session.resume
- _userauth.sign_in
- _userauth.sign_up
- _userauth.auth_fail

Email Events

If the email channel is enabled, Amazon Pinpoint streams events about email deliveries, complaints, opens, and more.

Example

The JSON object for an email event contains the data shown in the following example.

```
{
  "event_type": "_email.delivered",
  "event_timestamp": 1487973802507,
  "arrival_timestamp": 1487973803515,
  "event_version": "3.0",
  "application": {
    "app_id": "a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6",
    "cognito_identity_pool_id": "us-east-1:a1b2c3d4-e5f6-g7h8-i9j0-k1l2m3n4o5p6",
    "package_name": "main.page",
    "sdk": {
      "name": "aws-sdk-mobile-analytics-js",
      "version": "0.9.1:2.4.8"
    },
    "title": "title",
    "version_name": "1.0",
    "version_code": "1"
  },
  "client": {
    "client_id": "m3n4o5p6-a1b2-c3d4-e5f6-g7h8i9j0k1l2",
    "cognito_id": "us-east-1:i9j0k1l2-m3n4-o5p6-a1b2-c3d4e5f6g7h8"
  },
  "device": {
    "locale": {
      "code": "en_US",
      "country": "US",
      "language": "en"
    },
    "make": "generic web browser",
    "model": "Unknown",
    "platform": {
      "name": "android",
      "version": "10.10"
    }
  },
  "session": {
    "session_id": "f549dea9-1090-945d-c3d1-e496780baac5",
    "start_timestamp": 1487973202531,
    "stop_timestamp": 1487973802507
  },
  "attributes": {},
  "metrics": {}
}
```

Standard Email Event Types

Amazon Pinpoint streams the following standard types for the email channel:

- _email.send
- _email.delivered
- _email.rejected
- _email.hardbounce
- _email.softbounce
- _email.complaint
- _email.open
- _email.click
- _email.unsubscribe

SMS Events

If the SMS channel is enabled, Amazon Pinpoint streams events about SMS deliveries.

Example

The JSON object for an SMS event contains the data shown in the following example.

```
{
  "account_id": "123412341234",
  "event_type": "_SMS.SUCCESS",
  "arrival_timestamp": 2345678,
  "timestamp": 13425345,
  "timestamp_created": "1495756908285",
  "application_key": "AppKey-688037015201",
  "unique_id": "uniqueId-688037015201140879759 69",
  "attributes": {
    "message_id": "12234sdv",
    "sender_request_id": "abdfg",
    "number_of_message_parts": 1,
    "record_status": "SUCCESS",
    "message_type": "Transactional",
    "keyword": "test",
    "mcc_mnc": "456123",
    "iso_country_code": "US"
  },
  "metrics": {
    "price_in_millicents_usd": 0.0
  },
  "facets": {},
  "additional_properties": {}
}
```

Standard SMS Event Types

Amazon Pinpoint streams the following standard types for the SMS channel:

- _sms.send
- _sms.success
- _sms.fail
- _sms.optout

Event Attributes

The JSON object for an event contains the following attributes.

Event

Attribute	Description
`event_type`	The type of event reported by your app.
`event_timestamp`	The time at which the event is reported as Unix time in milliseconds. If your app reports events using the AWS Mobile SDK for Android or the AWS Mobile SDK for iOS, the time stamp is automatically generated.
`arrival_timestamp`	The time at which the event is received by Amazon Pinpoint as Unix time in milliseconds. Does not apply to campaign events.
`event_version`	The version of the event JSON schema. Check this version in your event-processing application so that you know when to update the application in response to a schema update.
`application`	Your Amazon Pinpoint app. See the *Application* table.
`client`	The app client installed on the device that reports the event. See the *Client* table.
`device`	The device that reports the event. See the *Device* table.
`session`	The app session on the device. Typically a session begins when a user opens your app.
`attributes`	Custom attributes that your app reports to Amazon Pinpoint as part of the event.
`metrics`	Custom metrics that your app reports to Amazon Pinpoint as part of the event.

Application

Attribute	Description
`app_id`	The unique ID of the Amazon Pinpoint app that reports the event.
`cognito_identity_pool_id`	The unique ID of the Amazon Cognito identity pool used by your app.
`package_name`	The name of your app package. For example, `com.example.my_app`.
`sdk`	The SDK used to report the event.
`title`	The title of your app.
`version_name`	The customer-facing app version, such as `V2.0`.

Attribute	Description
`version_code`	The internal code that represents your app version.

Client

Attribute	Description
`client_id`	The unique ID for the app client installed on the device. This ID is automatically generated by the AWS Mobile SDK for iOS and the AWS Mobile SDK for Android.
`cognito_id`	The unique ID assigned to the app client in the Amazon Cognito identity pool used by your app.

Device

Attribute	Description
`locale`	The device locale.
`make`	The device make, such as `Apple` or `Samsung`.
`model`	The device model, such as `iPhone`.
`platform`	The device platform, such as `ios` or `android`.

Session

Attribute	Description
`session_id`	The unique ID for the app session.
`start_timestamp`	The time at which the session starts as Unix time in milliseconds.
`stop_timestamp`	The time at which the session stops as Unix time in milliseconds.

Logging Amazon Pinpoint API Calls with AWS CloudTrail

Amazon Pinpoint is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Amazon Pinpoint. CloudTrail captures API calls for Amazon Pinpoint as events. The calls captured include calls from the Amazon Pinpoint console and code calls to the Amazon Pinpoint API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for Amazon Pinpoint. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to Amazon Pinpoint, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, including how to configure and enable it, see the AWS CloudTrail User Guide.

Amazon Pinpoint Information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When supported event activity occurs in Amazon Pinpoint, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing Events with CloudTrail Event History.

For an ongoing record of events in your AWS account, including events for Amazon Pinpoint, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- Overview for Creating a Trail
- CloudTrail Supported Services and Integrations
- Configuring Amazon SNS Notifications for CloudTrail
- Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity Element.

You can create a trail and store your log files in your Amazon S3 bucket for as long as you want, and define Amazon S3 lifecycle rules to archive or delete log files automatically. By default, your log files are encrypted with Amazon S3 server-side encryption (SSE).

To be notified of log file delivery, configure CloudTrail to publish Amazon SNS notifications when new log files are delivered. For more information, see Configuring Amazon SNS Notifications for CloudTrail.

You can also aggregate Amazon Pinpoint log files from multiple AWS Regions and multiple AWS accounts into a single Amazon S3 bucket.

For more information, see Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts.

You can use CloudTrail to log actions for the following Amazon Pinpoint APIs:

- Amazon Pinpoint API (p. 101)
- Amazon Pinpoint Email API (p. 102)
- Amazon Pinpoint SMS and Voice API (p. 103)

Amazon Pinpoint API Actions That Can be Logged by CloudTrail

Amazon Pinpoint supports logging the following actions as events in CloudTrail log files:

- CreateApp
- CreateCampaign
- CreateImportJob
- CreateSegment
- DeleteAdmChannel
- DeleteApnsChannel
- DeleteApnsSandboxChannel
- DeleteApnsVoipChannel
- DeleteApnsVoipSandboxChannel
- DeleteApp
- DeleteBaiduChannel
- DeleteCampaign
- DeleteEmailChannel
- DeleteEventStream
- DeleteGcmChannel
- DeleteSegment
- DeleteSmsChannel
- GetAdmChannel
- GetApnsChannel
- GetApnsSandboxChannel
- GetApnsVoipChannel
- GetApnsVoipSandboxChannel
- GetApp
- GetApplicationSettings
- GetApps
- GetBaiduChannel
- GetCampaign

- GetCampaignActivities
- GetCampaignVersion
- GetCampaignVersions
- GetCampaigns
- GetEmailChannel
- GetEventStream
- GetGcmChannel
- GetImportJob
- GetImportJobs
- GetSegment
- GetSegmentImportJobs
- GetSegmentVersion
- GetSegmentVersions
- GetSegments
- GetSmsChannel
- PutEventStream
- UpdateAdmChannel
- UpdateApnsChannel
- UpdateApnsSandboxChannel
- UpdateApnsVoipChannel
- UpdateApnsVoipSandboxChannel
- UpdateApplicationSettings
- UpdateBaiduChannel
- UpdateCampaign
- UpdateEmailChannel
- UpdateGcmChannel
- UpdateSegment
- UpdateSmsChannel

Note
The following Amazon Pinpoint actions **aren't** logged in CloudTrail:

- GetEndpoint
- SendMessages
- SendUsersMessages
- UpdateEndpoint
- UpdateEndpointsBatch

Amazon Pinpoint SMS Email Actions That Can be Logged by CloudTrail

The Amazon Pinpoint Email API supports logging the following actions as events in CloudTrail log files:

- CreateConfigurationSet
- CreateConfigurationSetEventDestination

- CreateDedicatedIpPool
- CreateEmailIdentity
- DeleteConfigurationSet
- DeleteConfigurationSetEventDestination
- DeleteDedicatedIpPool
- DeleteEmailIdentity
- GetAccount
- GetConfigurationSet
- GetConfigurationSetEventDestinations
- GetDedicatedIp
- GetDedicatedIps
- GetEmailIdentity
- ListConfigurationSets
- ListDedicatedIpPools
- ListEmailIdentities
- PutAccountDedicatedIpWarmupAttributes
- PutAccountSendingAttributes
- PutConfigurationSetDeliveryOptions
- PutConfigurationSetReputationOptions
- PutConfigurationSetSendingOptions
- PutConfigurationSetTrackingOptions
- PutDedicatedIpInPool
- PutDedicatedIpWarmupAttributes
- PutEmailIdentityDkimAttributes
- PutEmailIdentityFeedbackAttributes
- PutEmailIdentityMailFromAttributes
- UpdateConfigurationSetEventDestination

Note
The following Amazon Pinpoint Email API action **isn't** logged in CloudTrail:

- SendEmail

Amazon Pinpoint SMS and Voice API Actions That Can be Logged by CloudTrail

The Amazon Pinpoint SMS and Voice API supports logging the following actions as events in CloudTrail log files:

- CreateConfigurationSet
- DeleteConfigurationSet
- GetConfigurationSetEventDestinations
- CreateConfigurationSetEventDestination
- UpdateConfigurationSetEventDestination
- DeleteConfigurationSetEventDestination

Note
The following Amazon Pinpoint SMS and Voice API action **isn't** logged in CloudTrail:

- SendVoiceMessage

Example: Amazon Pinpoint Log File Entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the `GetCampaigns` and `CreateCampaign` actions of the Amazon Pinpoint API.

```
{
  "Records": [
    {
      "awsRegion": "us-east-1",
      "eventID": "example0-09a3-47d6-a810-c5f9fd2534fe",
      "eventName": "GetCampaigns",
      "eventSource": "pinpoint.amazonaws.com",
      "eventTime": "2018-02-03T00:56:48Z",
      "eventType": "AwsApiCall",
      "eventVersion": "1.05",
      "readOnly": true,
      "recipientAccountId": "123456789012",
      "requestID": "example1-b9bb-50fa-abdb-80f274981d60",
      "requestParameters": {
        "application-id": "example71dfa4c1aab66332a5839798f",
        "page-size": "1000"
      },
      "responseElements": null,
      "sourceIPAddress": "192.0.2.0",
      "userAgent": "Jersey/${project.version} (HttpUrlConnection 1.8.0_144)",
      "userIdentity": {
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "accountId": "123456789012",
        "arn": "arn:aws:iam::123456789012:root",
        "principalId": "123456789012",
        "sessionContext": {
          "attributes": {
            "creationDate": "2018-02-02T16:55:29Z",
            "mfaAuthenticated": "false"
          }
        },
        "type": "Root"
      }
    },
    {
      "awsRegion": "us-east-1",
      "eventID": "example0-09a3-47d6-a810-c5f9fd2534fe",
      "eventName": "CreateCampaign",
      "eventSource": "pinpoint.amazonaws.com",
      "eventTime": "2018-02-03T01:05:16Z",
      "eventType": "AwsApiCall",
      "eventVersion": "1.05",
      "readOnly": false,
      "recipientAccountId": "123456789012",
      "requestID": "example1-b9bb-50fa-abdb-80f274981d60",
      "requestParameters": {
        "Description": "***",
        "HoldoutPercent": 0,
        "IsPaused": false,
        "MessageConfiguration": "***",
        "Name": "***",
        "Schedule": {
          "Frequency": "ONCE",
          "IsLocalTime": true,
          "StartTime": "2018-02-03T00:00:00-08:00",
          "Timezone": "utc-08"
        },
        "SegmentId": "exampleda204adf991a80281aa0e591",
        "SegmentVersion": 1,
        "application-id": "example71dfa4c1aab66332a5839798f"
      },
      "responseElements": {
        "ApplicationId": "example71dfa4c1aab66332a5839798f",
        "CreationDate": "2018-02-03T01:05:16.425Z",
        "Description": "***",
        "HoldoutPercent": 0,
        "Id": "example54a654f80948680cbba240ede",
        "IsPaused": false,
        "LastModifiedDate": "2018-02-03T01:05:16.425Z",
        "MessageConfiguration": "***",
        "Name": "***",
        "Schedule": {
          "Frequency": "ONCE",
          "IsLocalTime": true,
          "StartTime": "2018-02-03T00:00:00-08:00",
          "Timezone": "utc-08"
        },
        "SegmentId": "example4da204adf991a80281example",
        "SegmentVersion": 1,
        "State": {
          "CampaignStatus": "SCHEDULED"
        },
        "Version": 1
      },
      "sourceIPAddress": "192.0.2.0",
      "userAgent": "aws-cli/1.14.9 Python/3.4.3 Linux/3.4.0+ botocore/1.8.34",
      "userIdentity": {
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "accountId": "123456789012",
        "arn": "arn:aws:iam::123456789012:user/userName",
        "principalId": "AIDAIHTHRCDA62EXAMPLE",
        "type": "IAMUser",
        "userName": "userName"
      }
    }
  ]
}
```

The following example shows a CloudTrail log entry that demonstrates the `CreateConfigurationSet` and `CreateConfigurationSetEventDestination` actions in the Amazon Pinpoint SMS and Voice API.

```
{
  "Records": [
    {
      "eventVersion":"1.05",
      "userIdentity":{
        "type":"IAMUser",
        "principalId":"AIDAIHTHRCDA62EXAMPLE",
        "arn":"arn:aws:iam::111122223333:user/SampleUser",
        "accountId":"111122223333",
        "accessKeyId":"AKIAIOSFODNN7EXAMPLE",
        "userName":"SampleUser"
      },
      "eventTime":"2018-11-06T21:45:55Z",
      "eventSource":"sms-voice.amazonaws.com",
      "eventName":"CreateConfigurationSet",
      "awsRegion":"us-east-1",
      "sourceIPAddress":"192.0.0.1",
      "userAgent":"PostmanRuntime/7.3.0",
      "requestParameters":{
        "ConfigurationSetName":"MyConfigurationSet"
      },
      "responseElements":null,
      "requestID":"56dcc091-e20d-11e8-87d2-9994aexample",
      "eventID":"725843fc-8846-41f4-871a-7c52dexample",
      "readOnly":false,
      "eventType":"AwsApiCall",
      "recipientAccountId":"123456789012"
    },
    {
      "eventVersion":"1.05",
      "userIdentity":{
        "type":"IAMUser",
        "principalId":"AIDAIHTHRCDA62EXAMPLE",
        "arn":"arn:aws:iam::111122223333:user/SampleUser",
        "accountId":"111122223333",
        "accessKeyId":"AKIAIOSFODNN7EXAMPLE",
        "userName":"SampleUser"
      },
      "eventTime":"2018-11-06T21:47:08Z",
      "eventSource":"sms-voice.amazonaws.com",
      "eventName":"CreateConfigurationSetEventDestination",
      "awsRegion":"us-east-1",
      "sourceIPAddress":"192.0.0.1",
      "userAgent":"PostmanRuntime/7.3.0",
      "requestParameters":{
        "EventDestinationName":"CloudWatchEventDestination",
        "ConfigurationSetName":"MyConfigurationSet",
        "EventDestination":{
          "Enabled":true,
          "MatchingEventTypes":[
            "INITIATED_CALL",
            "INITIATED_CALL"
          ],
          "CloudWatchLogsDestination":{
            "IamRoleArn":"arn:aws:iam::111122223333:role/iamrole-01",
            "LogGroupArn":"arn:aws:logs:us-east-1:111122223333:log-group:clientloggroup-01"
          }
        }
      },
      "responseElements":null,
      "requestID":"81de1e73-e20d-11e8-b158-d5536example",
      "eventID":"fcafc21f-7c93-4a3f-9e72-fca2dexample",
      "readOnly":false,
      "eventType":"AwsApiCall",
      "recipientAccountId":"111122223333"
    }
  ]
}
```

Deleting Data from Amazon Pinpoint

Depending on how you use it, Amazon Pinpoint might store certain data that could be considered personal. For example, each endpoint in Amazon Pinpoint contains contact information for an end user, such as that person's email address or mobile phone number.

You can use the Amazon Pinpoint API to permanently delete personal data. This section includes procedures for deleting various types of data that could be considered personal.

Deleting Endpoints

An endpoint represents a single method of contacting one of your customers. Each endpoint can refer to a customer's email address, mobile device identifier, or email address. In many jurisdictions, this type of information might be considered personal.

The procedure in this section uses the AWS CLI to interact with the Amazon Pinpoint API. This procedure assumes that you've already installed and configured the AWS CLI. For more information, see Installing the AWS Command Line Interface in the *AWS Command Line Interface User Guide*.

To delete an endpoint by using the AWS CLI

- At the command line, type the following command:

```
aws pinpoint delete-endpoint --application-id example1884c7d659a2feaa0c5 --endpoint-
id ad015a3bf4f1b2b0b82example
```

  In the preceding command, replace *example1884c7d659a2feaa0c5* with the ID of the application or project that the endpoint is located in. Also, replace *ad015a3bf4f1b2b0b82example* with the unique ID of the endpoint itself.

Deleting Segment and Endpoint Data Stored in Amazon S3

You can import segments from a file that's stored in an Amazon S3 bucket by using the Amazon Pinpoint console or the API. You can also export application, segment, or endpoint data from Amazon Pinpoint to an Amazon S3 bucket. Both the imported and exported files can contain personal data, including email addresses, mobile phone numbers, and information about the physical location of the endpoint.

Content delivered to Amazon S3 buckets might contain customer content. For more information about removing sensitive data, see How Do I Empty an S3 Bucket? or How Do I Delete an S3 Bucket?.

Delete All AWS Data by Closing Your AWS Account

It's also possible to delete all data that you've stored in Amazon Pinpoint by closing your AWS account. However, this action also deletes all other data—personal or non-personal—that you've stored in every other AWS service.

When you close your AWS account, we retain the data in your AWS account for 90 days. At the end of this retention period, we delete this data permanently and irreversibly.

Warning
The following procedure completely removes all data that's stored in your AWS account across all AWS services and regions.

You can close your AWS account by using the AWS Management Console.

To close your AWS account

1. Open the AWS Management Console at https://console.aws.amazon.com.
2. Go to the **Account Settings** page at https://console.aws.amazon.com/billing/home?#/account.

   Warning
   The following two steps permanently delete all of the data you've stored in all AWS services across all AWS Regions.
3. Under **Close Account**, read the disclaimer that describes the consequences of closing your AWS account. If you agree to the terms, select the check box, and then choose **Close Account**.
4. On the confirmation dialog box, choose **Close Account**.

Permissions

To use Amazon Pinpoint, users in your AWS account require permissions that allow them to view analytics data, define user segments, create and deploy campaigns, and more. Users of your app also require access to Amazon Pinpoint so that your app can register endpoints and report usage data. To grant access to Amazon Pinpoint features, create AWS Identity and Access Management (IAM) policies that allow Amazon Pinpoint actions (p. 109).

IAM is a service that helps you securely control access to AWS resources. IAM policies include statements that allow or deny specific actions that users can perform on specific resources. Amazon Pinpoint provides a set of actions for IAM policies (p. 111) that you can use to specify granular permissions for Amazon Pinpoint users. You can grant the appropriate level of access to Amazon Pinpoint without creating overly permissive policies that might expose important data or compromise your campaigns. For example, you can grant unrestricted access to an Amazon Pinpoint administrator, and grant read-only access to individuals in your organization who only need access to analytics.

For more information about IAM policies, see Overview of IAM Policies in the *IAM User Guide*.

When you add your app to Amazon Pinpoint by creating a project in AWS Mobile Hub, Mobile Hub automatically provisions AWS resources for app user authentication (p. 124). Mobile Hub creates an Amazon Cognito identity pool so that app users can authenticate with AWS. Mobile Hub also creates an IAM role that allows app users to register with Amazon Pinpoint and report usage data. You can customize these resources as needed for your app.

To import endpoint definitions, you must grant Amazon Pinpoint read-only access to an Amazon S3 bucket (p. 127).

Topics

IAM Policies for Amazon Pinpoint Users

You can add Amazon Pinpoint API actions to AWS Identity and Access Management (IAM) policies to allow or deny specific actions for Amazon Pinpoint users in your account. The Amazon Pinpoint API actions in your policies control what users can do in the Amazon Pinpoint console. These actions also control which programmatic requests users can make with the AWS SDKs, the AWS CLI, or the Amazon Pinpoint REST API.

In a policy, you specify each action with the `mobiletargeting` namespace followed by a colon and the name of the action, such as `GetSegments`. Most actions correspond to a request to the Amazon Pinpoint REST API using a specific URI and HTTP method. For example, if you allow the `mobiletargeting:GetSegments` action in a user's policy, the user is allowed to make an HTTP GET request against the `/apps/`*`project-id`*`/segments` URI. This policy also allows the user to view the segments for a project in the console, and to retrieve the segments by using an AWS SDK or the AWS CLI.

Each action is performed on a specific Amazon Pinpoint resource, which you identify in a policy statement by its Amazon Resource Name (ARN). For example, the `mobiletargeting:GetSegments` action is performed on a specific app, which you identify with the ARN, `arn:aws:mobiletargeting:`*`region`*`:`*`account-id`*`:apps/`*`project-id`*.

You can refer generically to all Amazon Pinpoint actions or resources by using wildcards ("*"). For example, to allow all actions for all resources, include the following in a policy statement:

```
"Effect": "Allow",
"Action": "mobiletargeting:*",
"Resource": "*"
```

Example Policies

The following examples demonstrate how you can manage Amazon Pinpoint access with IAM policies.

Amazon Pinpoint API Actions

Amazon Pinpoint Administrator

The following administrator policy allows full access to Amazon Pinpoint actions and resources:

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

Read-Only Access

The following policy allows read-only access for all apps in an account:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "mobiletargeting:Get*"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:mobiletargeting:*:account-id:apps/*"
            ]
        }
    ]
}
```

In the preceding policy example, replace *accountId* with your AWS Account ID.

You can also create a policy that allows read-only access to a specific Amazon Pinpoint project. To do this, specify an AWS Region and a project ID, as shown in the following example:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "mobiletargeting:Get*"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:mobiletargeting:region:account-id:apps/project-id"
            ]
        }
    ]
}
```

In the preceding policy example, replace *region* with the name of the AWS Region you're using, *account-id* with your AWS account ID, and *project-id* with the unique ID of your Amazon Pinpoint project.

Amazon Pinpoint SMS and Voice API Actions

Admin Access

The following policy grants full access to the Amazon Pinpoint SMS and Voice API:

```
{
    "Version": "2018-09-05",
    "Statement": [
        {
            "Action": [
                "sms-voice:*"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

Read-Only Access

The following policy allows read-only access to the Amazon Pinpoint SMS and Voice API:

```
{
    "Version": "2018-09-05",
    "Statement": [
        {
            "Action": [
                "sms-voice:Get*"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

Amazon Pinpoint API Actions

This section contains API actions that you can add to the IAM policies in your AWS account. By adding these policies to an IAM user account, you can specify which Amazon Pinpoint features that user is allowed to perform.

Categories:

- Campaigns (p. 112)

- Channels (p. 113)
- Endpoints (p. 117)
- Event Streams (p. 117)
- Export Jobs (p. 118)
- Import Jobs (p. 119)
- Messages (p. 119)
- Phone Number Validate (p. 119)
- Projects (p. 120)
- Reports (p. 121)
- Segments (p. 121)
- Users (p. 122)

Campaigns

The following permissions are related to managing campaigns in your Amazon Pinpoint account.

`mobiletargeting:CreateCampaign`

Create a campaign for a project.

- URI – `/apps/`*`project-id`*`/campaigns`
- Method – POST
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`account-id`*`:apps/`*`project-id`*`/campaigns`

`mobiletargeting:DeleteCampaign`

Delete a specific campaign.

- URI – `/apps/`*`project-id`*`/campaigns/`*`campaign-id`*
- Method – DELETE
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`account-id`*`:apps/`*`project-id`*`/campaigns/`*`campaign-id`*

`mobiletargeting:GetCampaign`

Retrieve information about a specific campaign.

- URI – `/apps/`*`project-id`*`/campaigns/`*`campaign-id`*
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`account-id`*`:apps/`*`project-id`*`/campaigns/`*`campaign-id`*

`mobiletargeting:GetCampaignActivities`

Retrieve information about the activities performed by a campaign.

- URI – `/apps/`*`project-id`*`/campaigns/`*`campaign-id`*`/activities`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`account-id`*`:apps/`*`project-id`*`/campaigns/`*`campaign-id`*

`mobiletargeting:GetCampaigns`

Retrieve information about all campaigns for a project.

- URI – `/apps/`*`project-id`*`/campaigns`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`account-id`*`:apps/`*`project-id`*

`mobiletargeting:GetCampaignVersion`

Retrieve information about a specific campaign version.

- URI – `/apps/project-id/campaigns/campaign-id/versions/version-id`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:account-id:apps/project-id/campaigns/campaign-id`

`mobiletargeting:GetCampaignVersions`

Retrieve information about the current and prior versions of a campaign.

- URI – `/apps/project-id/campaigns/campaign-id/versions`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:account-id:apps/project-id/campaigns/campaign-id`

`mobiletargeting:UpdateCampaign`

Update a specific campaign.

- URI – `/apps/project-id/campaigns/campaign-id`
- Method – PUT
- Resource ARN – `arn:aws:mobiletargeting:region:account-id:apps/project-id/campaigns/campaign-id`

Channels

The following permissions are related to managing channels in your Amazon Pinpoint account. In Amazon Pinpoint, *channels* refer to the methods you use to contact your customers, such as by sending email, SMS messages, or push notifications.

`mobiletargeting:DeleteAdmChannel`

Delete the Amazon Device Messaging (ADM) channel for a project.

- URI – `/apps/project-id/channels/adm`
- Method – DELETE
- Resource ARN – `arn:aws:mobiletargeting:region:accountid:apps/project-id/channels/adm`

`mobiletargeting:GetAdmChannel`

Retrieve information about the Amazon Device Messaging (ADM) channel for a project.

- URI – `/apps/project-id/channels/adm`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:accountid:apps/project-id/channels/adm`

`mobiletargeting:UpdateAdmChannel`

Update the Amazon Device Messaging (ADM) channel for a project.

- URI – `/apps/project-id/channels/adm`
- Method – PUT
- Resource ARN – `arn:aws:mobiletargeting:region:accountid:apps/project-id/channels/adm`

`mobiletargeting:DeleteApnsChannel`

Delete the APNs channel for a project.

- URI – `/apps/project-id/channels/apns`
- Method – DELETE
- Resource ARN – `arn:aws:mobiletargeting:region:account-id:apps/project-id/channels/apns`

`mobiletargeting:GetApnsChannel`

Retrieve information about the APNs channel for a project.

- URI – `/apps/project-id/channels/apns`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:account-id:apps/project-id/channels/apns`

`mobiletargeting:UpdateApnsChannel`

Update the Apple Push Notification service (APNs) certificate and private key, which allow Amazon Pinpoint to send push notifications to your iOS app.

- URI – `/apps/project-id/channels/apns`
- Method – PUT
- Resource ARN – `arn:aws:mobiletargeting:region:account-id:apps/project-id/channels/apns`

`mobiletargeting:DeleteApnsSandboxChannel`

Delete the APNs sandbox channel for a project.

- URI – `/apps/project-id/channels/apns_sandbox`
- Method – DELETE
- Resource ARN – `arn:aws:mobiletargeting:region:account-id:apps/project-id/channels/apns_sandbox`

`mobiletargeting:GetApnsSandboxChannel`

Retrieve information about the APNs sandbox channel for a project.

- URI – `/apps/project-id/channels/apns_sandbox`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:account-id:apps/project-id/channels/apns_sandbox`

`mobiletargeting:UpdateApnsSandboxChannel`

Update the APNs sandbox channel for a project.

- URI – `/apps/project-id/channels/apns_sandbox`
- Method – PUT
- Resource ARN – `arn:aws:mobiletargeting:region:account-id:apps/project-id/channels/apns_sandbox`

`mobiletargeting:DeleteApnsVoipChannel`

Delete the APNs VoIP channel for a project.

- URI – `/apps/project-id/channels/apns_voip`
- Method – DELETE
- Resource ARN – `arn:aws:mobiletargeting:region:account-id:apps/project-id/channels/apns_voip`

`mobiletargeting:GetApnsVoipChannel`

Retrieve information about the APNs VoIP channel for a project.

- URI – `/apps/project-id/channels/apns_voip`

- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:account-id:apps/project-id/channels/apns_voip`

`mobiletargeting:UpdateApnsVoipChannel`

Update the APNs VoIP channel for a project.

- URI – `/apps/project-id/channels/apns_voip`
- Method – PUT
- Resource ARN – `arn:aws:mobiletargeting:region:account-id:apps/project-id/channels/apns_voip`

`mobiletargeting:DeleteApnsVoipChannel`

Delete the APNs VoIP sandbox channel for a project.

- URI – `/apps/project-id/channels/apns_voip_sandbox`
- Method – DELETE
- Resource ARN – `arn:aws:mobiletargeting:region:account-id:apps/project-id/channels/apns_voip_sandbox`

`mobiletargeting:GetApnsVoipChannel`

Retrieve information about the APNs VoIP sandbox channel for a project.

- URI – `/apps/project-id/channels/apns_voip_sandbox`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:account-id:apps/project-id/channels/apns_voip_sandbox`

`mobiletargeting:UpdateApnsVoipChannel`

Update the APNs VoIP sandbox channel for a project.

- URI – `/apps/project-id/channels/apns_voip_sandbox`
- Method – PUT
- Resource ARN – `arn:aws:mobiletargeting:region:account-id:apps/project-id/channels/apns_voip_sandbox`

`mobiletargeting:DeleteBaiduChannel`

Delete the Baidu channel for a project.

- URI – `/apps/project-id/channels/baidu`
- Method – DELETE
- Resource ARN – `arn:aws:mobiletargeting:region:account-id:apps/project-id/channels/baidu`

`mobiletargeting:GetBaiduChannel`

Retrieve information about the Baidu channel for a project.

- URI – `/apps/project-id/channels/baidu`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:account-id:apps/project-id/channels/baidu`

`mobiletargeting:UpdateBaiduChannel`

Update the Baidu channel for a project.

- URI – `/apps/project-id/channels/baidu`
- Method – PUT

- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`account-id`*`:apps/`*`project-id`*`/channels/baidu`

`mobiletargeting:DeleteEmailChannel`

Delete the email channel in a project.

- URI – `/apps/`*`project-id`*`/channels/email`
- Method – DELETE
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`account-id`*`:apps/`*`project-id`*`/channels/email`

`mobiletargeting:GetEmailChannel`

Obtain information about the email channel in a project.

- URI – `/apps/`*`project-id`*`/channels/email`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`account-id`*`:apps/`*`project-id`*`/channels/email`

`mobiletargeting:UpdateEmailChannel`

Update the email channel in a project.

- URI – `/apps/`*`project-id`*`/channels/email`
- Method – PUT
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`account-id`*`:apps/`*`project-id`*`/channels/email`

`mobiletargeting:DeleteGcmChannel`

Delete the GCM channel for a project.

- URI – `/apps/`*`project-id`*`/channels/gcm`
- Method – DELETE
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`account-id`*`:apps/`*`project-id`*`/channels/gcm`

`mobiletargeting:GetGcmChannel`

Retrieve information about the GCM channel for a project.

- URI – `/apps/`*`project-id`*`/channels/gcm`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`account-id`*`:apps/`*`project-id`*`/channels/gcm`

`mobiletargeting:UpdateGcmChannel`

Update the Firebase Cloud Messaging (FCM) or Google Cloud Messaging (GCM) API key, which allows Amazon Pinpoint to send push notifications to your Android app.

- URI – `/apps/`*`project-id`*`/channels/gcm`
- Method – PUT
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`account-id`*`:apps/`*`project-id`*`/channels/gcm`

`mobiletargeting:DeleteSmsChannel`

Delete the SMS channel in a project.

- URI – `/apps/`*`project-id`*`/channels/sms`
- Method – DELETE
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`account-id`*`:apps/`*`project-id`*`/channels/sms`

`mobiletargeting:GetSmsChannel`

Obtain information about the SMS channel in a project.

- URI – `/apps/project-id/channels/sms`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:account-id:apps/project-id/channels/sms`

`mobiletargeting:UpdateSmsChannel`

Update the SMS channel in a project.

- URI – `/apps/project-id/channels/sms`
- Method – PUT
- Resource ARN – `arn:aws:mobiletargeting:region:account-id:apps/project-id/channels/sms`

Endpoints

The following permissions are related to managing endpoints in your Amazon Pinpoint account. In Amazon Pinpoint, an *endpoint* is a single destination for your messages. For example, an endpoint could be a customer's email address, telephone number, or mobile device token.

`mobiletargeting:DeleteEndpoint`

Delete an endpoint.

- URI – `/apps/project-id/endpoints/endpoint-id`
- Method – DELETE
- Resource ARN – `arn:aws:mobiletargeting:region:account-id:apps/project-id/endpoints/endpoint-id`

`mobiletargeting:GetEndpoint`

Retrieve information about a specific endpoint.

- URI – `/apps/project-id/endpoints/endpoint-id`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:account-id:apps/project-id/endpoints/endpoint-id`

`mobiletargeting:UpdateEndpoint`

Create an endpoint or update the information for an endpoint.

- URI – `/apps/project-id/endpoints/endpoint-id`
- Method – PUT
- Resource ARN – `arn:aws:mobiletargeting:region:account-id:apps/project-id/endpoints/endpoint-id`

`mobiletargeting:UpdateEndpointsBatch`

Create or update endpoints as a batch operation.

- URI – `/apps/project-id/endpoints`
- Method – PUT
- Resource ARN – `arn:aws:mobiletargeting:region:account-id:apps/project-id`

Event Streams

The following permissions are related to managing campaigns in your Amazon Pinpoint account.

`mobiletargeting:DeleteEventStream`

Delete the event stream for a project.

- URI – `/apps/project-id/eventstream/`
- Method – DELETE
- Resource ARN – `arn:aws:mobiletargeting:region:account-id:apps/project-id/eventstream`

`mobiletargeting:GetEventStream`

Retrieve information about the event stream for a project.

- URI – `/apps/project-id/eventstream/`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:account-id:apps/project-id/eventstream`

`mobiletargeting:PutEventStream`

Create or update an event stream for a project.

- URI – `/apps/project-id/eventstream/`
- Method – PUT
- Resource ARN – `arn:aws:mobiletargeting:region:account-id:apps/project-id/eventstream`

Export Jobs

The following permissions are related to managing export jobs in your Amazon Pinpoint account. In Amazon Pinpoint, you create *export jobs* to send information about endpoints to an Amazon S3 bucket for storage or analysis.

`mobiletargeting:CreateExportJobs`

Create an export job for exporting endpoint definitions to Amazon S3.

- URI – `/apps/project-id/jobs/export`
- Method – POST
- Resource ARN – `arn:aws:mobiletargeting:region:account-id:apps/project-id/jobs/export`

`mobiletargeting:GetExportJob`

Obtain information a specific export job.

- URI – `/apps/project-id/jobs/export/job-id`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:account-id:apps/project-id/jobs/export/job-id`

`mobiletargeting:GetExportJobs`

Retrieve a list of all of the export jobs for a project.

- URI – `/apps/project-id/jobs/export`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:account-id:apps/project-id/jobs/export`

Import Jobs

The following permissions are related to managing import jobs in your Amazon Pinpoint account. In Amazon Pinpoint, you create *import jobs* to create segments based on endpoint definitions stored in an Amazon S3 bucket.

`mobiletargeting:CreateImportJob`

Import endpoint definitions from Amazon S3 to create a segment.

- URI – `/apps/project-id/jobs/import`
- Method – POST
- Resource ARN – `arn:aws:mobiletargeting:region:account-id:apps/project-id`

`mobiletargeting:GetImportJob`

Retrieve information about a specific import job.

- URI – `/apps/project-id/jobs/import/job-id`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:account-id:apps/project-id/jobs/import/job-id`

`mobiletargeting:GetImportJobs`

Retrieve information about all import jobs for a project.

- URI – `/apps/project-id/jobs/import`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:account-id:apps/project-id`

Messages

The following permissions are related to sending SMS messages and push notifications from your Amazon Pinpoint account. You can use the `SendMessages` and `SendUsersMessages` operations to send messages to specific endpoints without creating segments and campaigns first.

`mobiletargeting:SendMessages`

Send an SMS message or push notification to specific endpoints.

- URI – `/apps/project-id/messages`
- Method – POST
- Resource ARN – `arn:aws:mobiletargeting:region:account-id:apps/project-id/messages`

`mobiletargeting:SendUsersMessages`

Send an SMS message or push notification to all endpoints that are associated a specific user ID.

- URI – `/apps/project-id/users-messages`
- Method – POST
- Resource ARN – `arn:aws:mobiletargeting:region:account-id:apps/project-id/users-messages`

Phone Number Validate

The following permissions are related to using the Phone Number Validate feature in Amazon Pinpoint.

`mobiletargeting:PhoneNumberValidate`

Obtain information about a phone number.

- URI – `/phone/number/validate`
- Method – POST
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`account-id`*`:phone/number/validate`

Projects

The following permissions are related to managing projects in your Amazon Pinpoint account. Originally, projects were referred to as *applications*. For the purposes of these operations, a Amazon Pinpoint application is the same as a Amazon Pinpoint project.

`mobiletargeting:CreateApp`

Create a project.

- URI – `/apps`
- Method – POST
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`account-id`*`:apps`

`mobiletargeting:DeleteApp`

Delete a project.

- URI – `/apps/`*`project-id`*
- Method – DELETE
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`account-id`*`:apps/`*`project-id`*

`mobiletargeting:GetApp`

Retrieve information about a specific project in your Amazon Pinpoint account.

- URI – `/apps/`*`project-id`*
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`account-id`*`:apps/`*`project-id`*

`mobiletargeting:GetApps`

Retrieve a list of projects in your Amazon Pinpoint account.

- URI – `/apps`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`account-id`*`:apps`

`mobiletargeting:GetApplicationSettings`

Retrieve the default settings for an Amazon Pinpoint project.

- URI – `/apps/`*`project-id`*`/settings`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`account-id`*`:apps/`*`project-id`*

`mobiletargeting:UpdateApplicationSettings`

Retrieve the default settings for an Amazon Pinpoint project.

- URI – `/apps/`*`project-id`*`/settings`
- Method – PUT
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`account-id`*`:apps/`*`project-id`*

Reports

The following permission is related to retrieving reports and metrics related to your Amazon Pinpoint account.

`mobiletargeting:GetReports`

View analytics in the Amazon Pinpoint console.

- URI – Not applicable
- Method – Not applicable
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`account-id`*`:reports`

Segments

The following permissions are related to managing segments in your Amazon Pinpoint account. In Amazon Pinpoint, *segments* are groups of recipients for your campaigns that share certain attributes that you define.

`mobiletargeting:CreateSegment`

Create a segment. To allow a user to create a segment by importing endpoint data from outside of Amazon Pinpoint, allow the `mobiletargeting:CreateImportJob` action.

- URI – `/apps/`*`project-id`*`/segments`
- Method – POST
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`account-id`*`:apps/`*`project-id`*

`mobiletargeting:DeleteSegment`

Delete a segment.

- URI – `/apps/`*`project-id`*`/segments/`*`segment-id`*
- Method – DELETE
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`account-id`*`:apps/`*`project-id`*`/segments/`*`segment-id`*

`mobiletargeting:GetSegment`

Retrieve information about a specific segment.

- URI – `/apps/`*`project-id`*`/segments/`*`segment-id`*
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`account-id`*`:apps/`*`project-id`*`/segments/`*`segment-id`*

`mobiletargeting:GetSegmentExportJobs`

Retrieve information about jobs that create segments by importing endpoint definitions from Amazon S3.

- URI – `/apps/`*`project-id`*`/segments/`*`segment-id`*`/jobs/export`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`account-id`*`:apps/`*`project-id`*`/segments/`*`segment-id`*`/jobs/export`

`mobiletargeting:GetSegments`

Retrieve information about the segments for a project.

- URI – `/apps/`*`project-id`*`/segments`

- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`account-id`*`:apps/`*`project-id`*

`mobiletargeting:GetSegmentImportJobs`

Retrieve information about jobs that create segments by importing endpoint definitions from Amazon S3.

- URI – `/apps/`*`project-id`*`/segments/`*`segment-id`*`/jobs/import`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`account-id`*`:apps/`*`project-id`*`/segments/`*`segment-id`*

`mobiletargeting:GetSegmentVersion`

Retrieve information about a specific segment version.

- URI – `/apps/`*`project-id`*`/segments/`*`segment-id`*`/versions/`*`version-id`*
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`account-id`*`:apps/`*`project-id`*`/segments/`*`segment-id`*

`mobiletargeting:GetSegmentVersions`

Retrieve information about the current and prior versions of a segment.

- URI – `/apps/`*`project-id`*`/segments/`*`segment-id`*`/versions`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`account-id`*`:apps/`*`project-id`*`/segments/`*`segment-id`*

`mobiletargeting:UpdateSegment`

Update a specific segment.

- URI – `/apps/`*`project-id`*`/segments/`*`segment-id`*
- Method – PUT
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`account-id`*`:apps/`*`project-id`*`/segments/`*`segment-id`*

Users

The following permissions are related to managing users in your Amazon Pinpoint account. In Amazon Pinpoint, *users* correspond to individuals who receive messages from you. A single user might be associated with more than one endpoint.

`mobiletargeting:DeleteUser`

Delete all of the endpoints that are associated with a user ID.

- URI – `/apps/`*`project-id`*`/users/`*`user-id`*
- Method – DELETE
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`account-id`*`:apps/`*`project-id`*`/users/`*`user-id`*

`mobiletargeting:GetUser`

Retrieve information about the endpoints that are associated with a user ID.

- URI – `/apps/`*`project-id`*`/users/`*`user-id`*
- Method – GET

- Resource ARN – `arn:aws:mobiletargeting:region:account-id:apps/project-id/users/user-id`

Amazon Pinpoint SMS and Voice API Actions

This section contains API actions that you can add to the IAM policies in your AWS account. By adding these policies to an IAM user account, you can specify which features of the Amazon Pinpoint SMS and Voice a user is allowed to use.

To learn more about the Amazon Pinpoint SMS and Voice API, see the Amazon Pinpoint SMS and Voice API Reference.

`sms-voice:CreateConfigurationSet`

Create a configuration set for sending voice messages.

- URI – `/sms-voice/configuration-sets`
- Method – POST
- Resource ARN – not available; use `*`

`sms-voice:DeleteConfigurationSet`

Delete a voice message configuration set.

- URI – /sms-voice/configuration-sets/*ConfigurationSetName*
- Method – DELETE
- Resource ARN – not available; use `*`

`sms-voice:GetConfigurationSetEventDestinations`

Get information about a configuration set and the event destinations that it contains.

- URI – /sms-voice/configuration-sets/*ConfigurationSetName*/event-destinations
- Method – GET
- Resource ARN – not available; use `*`

`sms-voice:CreateConfigurationSetEventDestination`

Create an event destination for voice events.

- URI – /sms-voice/configuration-sets/*ConfigurationSetName*/event-destinations
- Method – POST
- Resource ARN – not available; use `*`

`sms-voice:UpdateConfigurationSetEventDestination`

Update an event destination for voice events.

- URI – /sms-voice/configuration-sets/*ConfigurationSetName*/event-destinations/*EventDestinationName*
- Method – PUT
- Resource ARN – not available; use `*`

`sms-voice:DeleteConfigurationSetEventDestination`

Delete an event destination for voice events.

- URI – /sms-voice/configuration-sets/`ConfigurationSetName`/event-destinations/`EventDestinationName`
- Method – DELETE
- Resource ARN – not available; use `*`

`sms-voice:SendVoiceMessage`

Create and send voice messages.

- URI – /sms-voice/voice/message
- Method – POST
- Resource ARN – not available; use `*`

User Authentication in Amazon Pinpoint Apps

To integrate with Amazon Pinpoint, your app must authenticate users to register endpoints and report usage data. When you add your app to Amazon Pinpoint by creating a project in AWS Mobile Hub, Mobile Hub automatically provisions the following AWS resources to help you implement user authentication:

Amazon Cognito identity pool

Amazon Cognito creates unique identities for your users and provides credentials that grant temporary access to the backend AWS resources for your app. An identity pool is a store of user identity data for your app users.

Amazon Cognito provides credentials for authenticated and unauthenticated users. Authenticated users include those who sign in to your app through a public identity provider, such as Facebook, Amazon, or Google. Unauthenticated users are those who do not sign in to your app, such as guest users.

You control your users' access to AWS resources with separate AWS Identity and Access Management (IAM) roles for authenticated and unauthenticated users. These roles must be assigned to the identity pool.

IAM role for unauthenticated users

Includes permissions policies that delegate limited access to AWS resources for unauthenticated users. You can customize the role as needed. By default, this role is assigned to the Amazon Cognito identity pool.

If your app requires users to authenticate with a public identity provider, you must create an IAM role for authenticated users and assign this role to the identity pool. To support Amazon Pinpoint, the permissions in your authenticated role must include the same permissions as those in the unauthenticated role created by Mobile Hub.

For more information about IAM roles, see IAM Roles in the *IAM User Guide*.

Your app code uses Amazon Cognito and IAM to authenticate users as follows:

1. Your app code constructs an Amazon Cognito credentials provider.
2. Your app code passes the provider as a parameter when it initializes the Amazon Pinpoint client.
3. The Amazon Pinpoint client uses the provider to get credentials for the user's identity in the identity pool. New users are assigned a new identity.
4. The user gains the permissions granted by the IAM roles that are associated with the identity pool.

For more information about how Amazon Cognito supports user authentication, see Amazon Cognito Identity: Using Federated Identities in the *Amazon Cognito Developer Guide*.

Unauthenticated Role

The unauthenticated role created by Mobile Hub allows your app users to send data to Amazon Pinpoint. The role name includes "`unauth_MOBILEHUB`"; for example, in the IAM console, you will see a role with a name similar to `MySampleApp_unauth_MOBILEHUB_1234567890`.

IAM roles delegate permissions with two types of policies:

- Permissions policy – Grants the user of the role permission to take the specified actions on the specified resources.
- Trust policy – Specifies which entities are allowed to assume the role and gain its permissions.

Permissions Policies

The unauthenticated role includes two permissions policies. The following permissions policy allows your app users to register with Amazon Pinpoint and report app usage events. Mobile Hub assigns the policy a name that includes "`mobileanalytics_MOBILEHUB`".

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "mobileanalytics:PutEvents"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "mobiletargeting:UpdateEndpoint"
      ],
      "Resource": [
        "arn:aws:mobiletargeting:*:*:apps/*"
      ]
    }
  ]
}
```

After your app is integrated with Amazon Pinpoint, your app registers an endpoint with Amazon Pinpoint when a new user starts an app session. Your app sends updated endpoint data to Amazon Pinpoint each time the user starts a new session.

The following permissions policy allows your app users to establish an identity with the Amazon Cognito identity pool for your app. Mobile Hub assigns the policy a name that includes "`signin_MOBILEHUB`".

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cognito-identity:GetId"
      ],
      "Resource": [
        "arn:aws:cognito-identity:*:*:identityPool/us-
east-1:1a2b3c4d-5e6f-7g8h-9i0j-1k2l3m4n5o6p"
      ]
    }
  ]
}
```

Trust Policy

To allow Amazon Cognito to assume the role for unauthenticated users in your identity pool, Mobile Hub adds the following trust policy to the role:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Federated": "cognito-identity.amazonaws.com"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "cognito-identity.amazonaws.com:aud": "us-
east-1:1a2b3c4d-5e6f-7g8h-9i0j-1k2l3m4n5o6p"
        },
        "ForAnyValue:StringLike": {
          "cognito-identity.amazonaws.com:amr": "unauthenticated"
        }
      }
    }
  ]
}
```

For an example of a trust policy assigned to an authenticated role, see Role-Based Access Control in the *Amazon Cognito Developer Guide*.

AWS Mobile Hub Service Role

AWS Mobile Hub creates an AWS Identity and Access Management (IAM) role in your AWS account when you agree to a one-time request in the Mobile Hub console to manage AWS resources and services for you. This role, called `MobileHub_Service_Role`, allows Mobile Hub to create and modify your AWS resources and services for your Mobile Hub project.

For more information about the Mobile Hub service role, see Mobile Hub Service Role and Policies Used on Your Behalf in the *AWS Mobile Hub Developer Guide*.

To add an app to Amazon Pinpoint, you create a Mobile Hub project and configure it to include the User Engagement feature. To support this feature, Mobile Hub adds the following permissions to the Mobile Hub service role:

```
{
  "Effect": "Allow",
  "Action": [
    "mobiletargeting:UpdateApnsChannel",
    "mobiletargeting:UpdateApnsSandboxChannel",
    "mobiletargeting:UpdateGcmChannel",
    "mobiletargeting:DeleteApnsChannel",
    "mobiletargeting:DeleteApnsSandboxChannel",
    "mobiletargeting:DeleteGcmChannel"
  ],
  "Resource": [
    "arn:aws:mobiletargeting:*:*:apps/*/channels/*"
  ]
}
```

These permissions allow Mobile Hub to manage the channels that Amazon Pinpoint uses to deliver messages to the push notification services for iOS and Android. Mobile Hub creates or updates a channel when you provide your credentials for Apple Push Notification service, Firebase Cloud Messaging, or Google Cloud Messaging. You provide your credentials by using the Mobile Hub console or Amazon Pinpoint console.

IAM Role for Importing Endpoints or Segments

With Amazon Pinpoint, you define a user segment by importing endpoint definitions from an Amazon S3 bucket in your AWS account. Before you import, you must delegate the required permissions to Amazon Pinpoint. Create an AWS Identity and Access Management (IAM) role and attach the following policies to the role:

- The `AmazonS3ReadOnlyAccess` AWS managed policy. This policy is created and managed by AWS, and it grants read-only access to your Amazon S3 bucket.
- A *trust policy* that allows Amazon Pinpoint to assume the role.

For more information about IAM roles, see IAM Roles in the *IAM User Guide*.

After you create the role, you can use Amazon Pinpoint to import segments. For an example of how to import a segment by using the AWS SDK for Java, see Importing Segments (p. 81). For information about creating the Amazon S3 bucket, creating endpoint files, and importing a segment by using the console, see Importing Segments in the *Amazon Pinpoint User Guide*.

Trust Policy

To allow Amazon Pinpoint to assume the IAM role and perform the actions allowed by the `AmazonS3ReadOnlyAccess` policy, attach the following trust policy to the role:

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

Creating the IAM Role (AWS CLI)

Complete the following steps to create the IAM role by using the AWS Command Line Interface (AWS CLI).

If you have not installed the AWS CLI, see Getting Set Up with the AWS Command Line Interface in the *AWS Command Line Interface User Guide*.

To create the IAM role by using the AWS CLI

1. Create a JSON file that contains the trust policy for your role, and save the file locally. You can copy the trust policy provided in this topic.
2. Use the `create-role` command to create the role and attach the trust policy:

   ```
   aws iam create-role --role-name PinpointSegmentImport --assume-role-policy-document
    file://PinpointImportTrustPolicy.json
   ```

   Following the `file://` prefix, specify the path to the JSON file that contains the trust policy.

   When you run this command, the AWS CLI prints the following output in your terminal:

   ```
   {
       "Role": {
           "AssumeRolePolicyDocument": {
               "Version": "2012-10-17",
               "Statement": [
                   {
                       "Action": "sts:AssumeRole",
                       "Effect": "Allow",
                       "Principal": {
                           "Service": "pinpoint.amazonaws.com"
                       }
                   }
               ]
           },
           "RoleId": "AIDACKCEVSQ6C2EXAMPLE",
           "CreateDate": "2016-12-20T00:44:37.406Z",
           "RoleName": "PinpointSegmentImport",
           "Path": "/",
           "Arn": "arn:aws:iam::111122223333:role/PinpointSegmentImport"
       }
   }
   ```
3. Use the `attach-role-policy` command to attach the `AmazonS3ReadOnlyAccess` AWS managed policy to the role:

   ```
   aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess
    --role-name PinpointSegmentImport
   ```

IAM Role for Exporting Endpoints or Segments

You can obtain a list of endpoints by creating an export job. When you create an export job, you have to specify a project ID, and you can optionally specify a segment ID. Amazon Pinpoint then exports a list of the endpoints associated with the project or segment to an Amazon Simple Storage Service (Amazon S3) bucket. The resulting file contains a JSON-formatted list of endpoints and their attributes (such as channel, address, opt-in/opt-out status, creation date, and endpoint ID).

To create an export job, you have to configure an IAM role that allows Amazon Pinpoint to write to an Amazon S3 bucket. The process of configuring the role consists of two steps:

1. Create an IAM policy that allows an entity (in this case, Amazon Pinpoint) to write to a specific Amazon S3 bucket.

2. Create an IAM role and attach the policy to it.

This section contains procedures for completing both of these steps. These procedures assume that you've already created an Amazon S3 bucket, as well as a folder within that bucket, for storing exported segments. For more information about creating buckets, see Create a Bucket in the *Amazon Simple Storage Service Getting Started Guide*.

These procedures also assume that you've already installed and configured the AWS CLI. For more information about setting up the AWS CLI, see Installing the AWS Command Line Interface in the *AWS Command Line Interface User Guide*.

Step 1: Create the IAM Policy

An IAM policy defines the permissions for an entity, such as an identity or resource. To create a role for exporting Amazon Pinpoint endpoints, you have to create a policy that grants permission to write to a specific folder in a specific Amazon S3 bucket. The following policy example follows the security practice of granting least privilege—that is, it only grants the permissions that are required to perform a single task.

To create the IAM policy

1. In a text editor, create a new file. Paste the following code into the file:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowUserToSeeBucketListInTheConsole",
            "Action": [
                "s3:ListAllMyBuckets",
                "s3:GetBucketLocation"
            ],
            "Effect": "Allow",
            "Resource": [ "arn:aws:s3:::*" ]
        },
        {
            "Sid": "AllowRootAndHomeListingOfBucket",
            "Action": [
                "s3:ListBucket"
            ],
            "Effect": "Allow",
            "Resource": [ "arn:aws:s3:::example-bucket" ],
            "Condition": {
                "StringEquals": {
                    "s3:delimiter": [ "/" ],
                    "s3:prefix": [
                        "",
                        "Exports/"
                    ]
                }
            },
        },
        {
            "Sid": "AllowListingOfUserFolder",
            "Action": [
                "s3:ListBucket"
            ],
            "Effect": "Allow",
            "Resource": [ "arn:aws:s3:::example-bucket" ],
            "Condition": {
                "StringLike": {
                    "s3:prefix": [
                        "Exports/*"
                    ]
                }
            }
        },
        {
            "Sid": "AllowAllS3ActionsInUserFolder",
            "Action": [ "s3:*" ],
            "Effect": "Allow",
            "Resource": [ "arn:aws:s3:::example-bucket/Exports/*" ]
        }
    ]
}
```

In the preceding code, replace all instances of `example-bucket` with the name of the Amazon S3 bucket that contains the folder that you want to export the segment information into. Also, replace all instances of `Example` with the name of the folder itself.

When you finish, save the file as `s3policy.json`.

2. At the command line, navigate to the directory where `s3policy.json` is located. Then type the following command to create the policy:

```
aws iam create-policy --policy-name s3ExportPolicy --policy-document
 file://s3policy.json
```

If the policy was created successfully, you see output similar to the following:

```
{
    "Policy": {
        "CreateDate": "2018-04-11T18:44:34.805Z",
        "IsAttachable": true,
        "DefaultVersionId": "v1",
        "AttachmentCount": 0,
        "PolicyId": "ANPAJ2YJQRJCG3EXAMPLE",
        "UpdateDate": "2018-04-11T18:44:34.805Z",
        "Arn": "arn:aws:iam::123456789012:policy/s3ExportPolicy",
        "PolicyName": "s3ExportPolicy",
        "Path": "/"
    }
}
```

Copy the Amazon Resource Name (ARN) of the policy (`arn:aws:iam::123456789012:policy/s3ExportPolicy` in the preceding example). In the next section, you must supply this ARN when you create the role.

Note
If you see a message stating that your account isn't authorized to perform the CreatePolicy operation, then you need to attach a policy to your user account that lets you create new IAM policies and roles. For more information, see Attaching IAM Policies (Console) in the *IAM User Guide*.

Step 2: Create the IAM Role

Now that you've created an IAM policy, you can create a role and attach the policy to it. Each IAM role contains a *trust policy*—a set of rules that specifies which entities are allowed to assume the role. In this section, you create a trust policy that allows Amazon Pinpoint to assume the role. Next, you create the role itself, and then attach the policy that you created in the previous section.

To create the IAM role

1. In a text editor, create a new file. Paste the following code into the file:

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

   Save the file as `trustpolicy.json`.

2. At the command line, navigate to the directory where `trustpolicy.json` is located. Then type the following command to create a new role:

```
aws iam create-role --role-name s3ExportRole --assume-role-policy-document
 file://trustpolicy.json
```

   If the command runs successfully, you see output similar to the following:

```
{
    "Role": {
        "RoleName": "s3ExportRole",
        "AssumeRolePolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Principal": {
                        "Service": "pinpoint.amazonaws.com"
                    },
                    "Action": "sts:AssumeRole"
                }
            ]
        },
        "RoleId": "AROAICPO353GIPEXAMPLE",
        "Arn": "arn:aws:iam::123456789012:role/s3ExportRole",
        "CreateDate": "2018-04-11T18:52:36.712Z",
        "Path": "/"
    }
}
```

3. At the command line, type the following command to attach the policy that you created in the previous section to the role you just created:

```
aws iam attach-role-policy --policy-arn arn:aws:iam::123456789012:policy/s3ExportPolicy
 --role-name s3ExportRole
```

   In the preceding command, replace *`arn:aws:iam::123456789012:policy/s3ExportPolicy`* with the ARN of the policy that you created in the previous section.

IAM Role for Streaming Events to Kinesis

Amazon Pinpoint can automatically send app usage data, or *event data*, from your app to a Kinesis stream or Amazon Kinesis Data Firehose delivery stream in your AWS account. Before Amazon Pinpoint can begin streaming the event data, you must delegate the required permissions to Amazon Pinpoint.

If you use the console to set up event streaming, Amazon Pinpoint automatically creates an AWS Identity and Access Management (IAM) role with the required permissions. For more information, see Streaming Amazon Pinpoint Events to Amazon Kinesis in the *Amazon Pinpoint User Guide*.

If you want to create the role manually, attach the following policies to the role:

- A permissions policy that allows Amazon Pinpoint to send records to your stream.
- A trust policy that allows Amazon Pinpoint to assume the role.

For more information about IAM roles, see IAM Roles in the *IAM User Guide*.

After you create the role, you can configure Amazon Pinpoint to automatically send events to your stream. For more information, see Streaming Amazon Pinpoint Events to Kinesis (p. 93).

Permissions Policies

To allow Amazon Pinpoint to send event data to your stream, attach one of the following policies to the role.

Amazon Kinesis Data Streams

The following policy allows Amazon Pinpoint to send event data to a Kinesis stream.

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Action": [
            "kinesis:PutRecords",
            "kinesis:DescribeStream"
        ],
        "Effect": "Allow",
        "Resource": [
            "arn:aws:kinesis:region:account-id:stream/stream-name"
        ]
    }
}
```

Amazon Kinesis Data Firehose

The following policy allows Amazon Pinpoint to send event data to a Kinesis Data Firehose delivery stream.

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": [
         "firehose:PutRecordBatch",
         "firehose:DescribeDeliveryStream"
        ],
        "Resource": [
         "arn:aws:firehose:region:account-id:deliverystream/delivery-stream-name"
      ]
    }
}
```

Trust Policy

To allow Amazon Pinpoint to assume the IAM role and perform the actions allowed by the permissions policy, attach the following trust policy to the role:

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

Creating the IAM Role (AWS CLI)

Complete the following steps to create the IAM role by using the AWS Command Line Interface (AWS CLI).

If you have not installed the AWS CLI, see Getting Set Up with the AWS Command Line Interface in the *AWS Command Line Interface User Guide*.

To create the role by using the IAM console, see Setting up Event Streaming in the *Amazon Pinpoint User Guide*.

To create the IAM role by using the AWS CLI

1. Create a JSON file that contains the trust policy for your role, and save the file locally. You can copy the trust policy provided in this topic.
2. Use the `create-role` command to create the role and attach the trust policy:

   ```
   aws iam create-role --role-name PinpointEventStreamRole --assume-role-policy-document
    file://PinpointEventStreamTrustPolicy.json
   ```

   Following the `file://` prefix, specify the path to the JSON file that contains the trust policy.

   When you run this command, the AWS CLI prints the following output in your terminal:

   ```
   {
       "Role": {
           "AssumeRolePolicyDocument": {
               "Version": "2012-10-17",
               "Statement": [
                   {
                       "Action": "sts:AssumeRole",
                       "Effect": "Allow",
                       "Principal": {
                           "Service": "pinpoint.amazonaws.com"
   ```

```
                    }
                }
            ]
        },
        "RoleId": "AIDACKCEVSQ6C2EXAMPLE",
        "CreateDate": "2017-02-28T18:02:48.220Z",
        "RoleName": "PinpointEventStreamRole",
        "Path": "/",
        "Arn": "arn:aws:iam::111122223333:role/PinpointEventStreamRole"
    }
}
```

3. Create a JSON file that contains the permissions policy for your role, and save the file locally. You can copy one of the policies provided in the Permissions Policies (p. 132) section.
4. Use the `put-role-policy` command to attach the permissions policy to the role:

```
aws iam put-role-policy --role-name PinpointEventStreamRole --
policy-name PinpointEventStreamPermissionsPolicy --policy-document
 file://PinpointEventStreamPermissionsPolicy.json
```

   Following the `file://` prefix, specify the path to the JSON file that contains the permissions policy.

Limits in Amazon Pinpoint

The following sections describe limits within Amazon Pinpoint.

Topics

General Limits

The following limits affect general use of Amazon Pinpoint.

Resource	Default Limit	Eligible for Increase
API request payload size	7 MB per request	No
Apps	100 per account	No

Endpoint Limits

The following limits apply to the Endpoints resource in the Amazon Pinpoint API.

Resource	Default Limit	Eligible for Increase
Attributes assigned to the `Attributes`, `Metrics`, and `UserAttributes` parameters collectively	40 per app	No
Attributes assigned to the `Attributes` parameter	40 per app	No
Attributes assigned to the `Metrics` parameter	40 per app	No
Attributes assigned to the `UserAttributes` parameter	40 per app	No
Attribute name length	50 characters	No

Resource	Default Limit	Eligible for Increase
Attribute value length	100 characters	No
`EndpointBatchItem` objects in an `EndpointBatchRequest` payload	100 per payload. The payload size can't exceed 7 MB.	No
Endpoints with the same user ID	10 unique endpoints per user ID	No
Values assigned to `Attributes` parameter attributes	50 per attribute	No
Values assigned to `UserAttributes` parameter attributes	50 per attribute	No

Endpoint Import Limits

The following limits apply when you import endpoints into Amazon Pinpoint.

Resource	Default Limit	Eligible for Increase
Concurrent import jobs	2 per account	Yes (p. 141)
Import size	1 GB per import job (For example, if each endpoint is 4 KB or less, you can import 250,000 endpoints.)	Yes (p. 141)

Segment Limits

The following limits apply to the Segments resource in the Amazon Pinpoint API.

Resource	Default Limit	Eligible for Increase
Maximum number of segments	100 per app	No
Maximum number of dimensions that can be used to create a segment	100 per segment	No

Campaign Limits

The following limits apply to the Campaigns resource in the Amazon Pinpoint API.

Resource	Default Limit	Eligible for Increase
Active campaigns	200 per account	Yes (p. 141)

Resource	Default Limit	Eligible for Increase
	Note An *active campaign* is a campaign that hasn't completed or failed. Active campaigns have a status of `SCHEDULED`, `EXECUTING`, or `PENDING_NEXT_RUN`.	
Message sends	100 million per campaign activity	Yes (p. 141)

Mobile Push Limits

The following limits apply to messages that Amazon Pinpoint delivers through mobile push channels.

Resource	Default Limit	Eligible for Increase
Maximum number of mobile push notifications that can be sent per second	25,000 notifications per second	No
Amazon Device Messaging (ADM) message payload size	6 KB per message	No
Apple Push Notification service (APNs) message payload size	4 KB per message	No
APNs sandbox message payload size	4 KB per message	No
Baidu Cloud Push message payload size	4 KB per message	No
Firebase Cloud Messaging (FCM) or Google Cloud Messaging (GCM) message payload size	4 KB per message	No

Email Limits

The limits in the following sections apply to the Email channel.

Email Sending Limits

Resource	Default Limit	Eligible for Increase
Number of emails that can be sent per 24-hour period (*sending quota*)	If your account is in the sandbox: 200 emails per 24-hour period. If your account is out of the sandbox, the quota varies based on your specific use case.	Yes (p. 141)

Resource	Default Limit	Eligible for Increase
	Note This quota is based on the number of recipients, as opposed to the number of unique messages sent. A *recipient* is any email address on the To: line.	
Number of emails that can be sent each second (*sending rate*)	If your account is in the sandbox: 1 email per second. If your account is out of the sandbox, the rate varies based on your specific use case. **Note** This rate is based on the number of recipients, as opposed to the number of unique messages sent. A *recipient* is any email address on the To: line.	Yes (p. 141)

Email Message Limits

Resource	Default Limit	Eligible for Increase
Maximum message size (including attachments)	10 MB per message.	No
Number of verified identities	10,000 identities. **Note** *Identities* refers to email addresses or domains, or any combination of the two. Every email you send using Amazon Pinpoint must be sent from a verified identity.	No

Email Sender and Recipient Limits

Resource	Default Limit	Eligible for Increase
Sender address	All sending addresses or domains must be verified.	No
Recipient address	If your account is still in the sandbox, all recipient email	Yes (p. 141)

Resource	Default Limit	Eligible for Increase
	addresses or domains must be verified. If your account is out of the sandbox, you can send to any valid address.	
Number of recipients per message	50 recipients per message.	No
Number of identities that you can verify	10,000 identities per AWS Region. **Note** *Identities* refers to email addresses or domains, or any combination of the two. Every email you send using Amazon Pinpoint must be sent from a verified identity.	No

SMS Limits

The following limits apply to the SMS channel.

Resource	Default Limit	Eligible for Increase
Account spend threshold	USD$1.00 per account.	Yes (p. 141)
Number of SMS messages that can be sent each second (*sending rate#*)	20 messages per second.	No
Number of Amazon SNS topics for two-way SMS	100,000 per account.	Yes (p. 141)

Voice Limits

The following limits apply to the Voice channel.

Resource	Default Limit	
Number of voice messages that can be sent in a 24-hour period	If your account is in the sandbox: 20 messages. If your account is out of the sandbox: unlimited.	
Number of voice messages that can be sent to a single recipient in a 24-hour period	5 messages.	

Resource	**Default Limit**	
Number of voice messages that can be sent per minute	If your account is in the sandbox: 5 calls per minute. If your account is out of the sandbox: 20 calls per minute.	
Number of voice messages that can be sent from a single originating phone number per second	1 message per second.	
Voice message length	If your account is in the sandbox: 30 seconds. If your account is out of the sandbox: 5 minutes.	
Ability to send voice messages to international phone numbers	If your account is in the sandbox, you can only send messages to recipients in the following countries: • Australia • Canada • China • Germany • Hong Kong • Israel • Japan • Mexico • Singapore • Sweden • United States • United Kingdom If your account is out of the sandbox, you can send messages to recipients in any country. **Note** International calls are subject to additional fees, which vary by destination country or region.	
Number of characters in a voice message	3,000 billable characters (characters in words that are spoken) 6,000 characters total (including billable characters and SSML tags)	

Resource	Default Limit	
Number of configuration sets	10,000 voice configuration sets per AWS Region.	

Event Ingestion Limits

The following limits apply to the ingestion of events using the AWS Mobile SDKs and the Amazon Pinpoint Events API.

Resource	Default Limit	Eligible for Increase
Maximum number of custom event types	1,500 per app	No
Maximum number of custom attribute keys	500 per app	No
Maximum number of custom attribute values per attribute key	100,000	No
Maximum number of characters per attribute key	50	No
Maximum number of characters per attribute value	200	No
Maximum number of custom metric keys	500 per app	No
Maximum number events in a request	100 per request	No
Maximum size of a request	4 MB	No
Maximum size of an individual event	1,000 KB	No
Maximum number of attribute keys and metric keys for each event	40 per request	No

Requesting a Limit Increase

If the value in the **Eligible for Increase** column in any of the tables above is **Yes**, you can request a change to that limit.

To request a limit increase

1. Sign in to the AWS Management Console at https://console.aws.amazon.com/.
2. Create a new Support case at https://console.aws.amazon.com/support/home#/case/create.
3. On the **Create Case** page, make the following selections:

- For **Regarding**, choose **Service Limit Increase**.
- For **Limit Type**, choose one of the following options:
  - Choose **Amazon Pinpoint** for limit increases related to Amazon Pinpoint campaigns and imports.
  - Choose **Amazon Pinpoint Email** for limit increases related to the email channel.
  - Choose **Amazon Pinpoint SMS** for limit increases related to the SMS channel.

4. For **Use Case Description**, explain why you are requesting the limit increase.
5. For **Support Language**, choose the language you prefer to use when communicating with the AWS Support team.
6. For **Contact Method**, choose your preferred method of communicating with the AWS Support team.
7. Choose **Submit**.

Creating Custom Channels with AWS Lambda

This is prerelease documentation for a feature in public beta release. It is subject to change.

Amazon Pinpoint supports messaging channels for mobile push, email, and SMS. However, some messaging use cases might require unsupported channels. For example, you might want to send a message to an instant messaging service, such as Facebook Messenger, or you might want to display a notification within your web application. In such cases, you can use AWS Lambda to create a custom channel that performs the message delivery outside of Amazon Pinpoint.

AWS Lambda is a compute service that you can use to run code without provisioning or managing servers. You package your code and upload it to Lambda as *Lambda functions*. Lambda runs a function when the function is invoked, which might be done manually by you or automatically in response to events.

For more information, see Lambda Functions in the *AWS Lambda Developer Guide*.

To create a custom channel, you define a Lambda function that handles the message delivery for an Amazon Pinpoint campaign. Then, you assign the function to a campaign by defining the campaign's `CampaignHook` settings. These settings include the Lambda function name and the `CampaignHook` mode. By setting the mode to `DELIVERY`, you specify that the Lambda function handles the message delivery instead of Amazon Pinpoint.

A Lambda function that you assign to a campaign is referred to as an Amazon Pinpoint *extension*.

With the `CampaignHook` settings defined, Amazon Pinpoint automatically invokes the Lambda function when it runs the campaign, without sending the campaign's message to a standard channel. Instead, Amazon Pinpoint sends *event data* about the message delivery to the function, and it allows the function to handle the delivery. The event data includes the message body and the list of endpoints to which the message should be delivered.

After Amazon Pinpoint successfully invokes the function, it generates a successful send event for the campaign.

Note
You can also use the `CampaignHook` settings to assign a Lambda function that modifies and returns a campaign's segment before Amazon Pinpoint delivers the campaign's message. For more information, see Customizing Segments with AWS Lambda (p. 84).

To create a custom channel with AWS Lambda, first create a function that accepts the event data sent by Amazon Pinpoint and handles the message delivery. Then, authorize Amazon Pinpoint to invoke the function by assigning a Lambda function policy. Finally, assign the function to one or more campaigns by defining `CampaignHook` settings.

Event Data

When Amazon Pinpoint invokes your Lambda function, it provides the following payload as the event data:

```
{
  "MessageConfiguration": {Message configuration}
  "ApplicationId": ApplicationId,
  "CampaignId": CampaignId,
  "TreatmentId": TreatmentId,
  "ActivityId": ActivityId,
  "ScheduledTime": Scheduled Time,
  "Endpoints": {
    EndpointId: {Endpoint definition}
    . . .
  }
}
```

The event data provides the following attributes:

- `MessageConfiguration` – Has the same structure as the `DirectMessageConfiguration` in the `Messages` resource in the Amazon Pinpoint API.
- `ApplicationId` – The ID of the Amazon Pinpoint project to which the campaign belongs.
- `CampaignId` – The ID of the Amazon Pinpoint project for which the function is invoked.
- `TreatmentId` – The ID of a campaign variation used for A/B testing.
- `ActivityId` – The ID of the activity being performed by the campaign.
- `ScheduledTime` – The schedule time at which the campaign's messages are delivered in ISO 8601 format.
- `Endpoints` – A map that associates endpoint IDs with endpoint definitions. Each event data payload contains up to 50 endpoints. If the campaign segment contains more than 50 endpoints, Amazon Pinpoint invokes the function repeatedly, with up to 50 endpoints at a time, until all endpoints have been processed.

Creating a Lambda Function

To create a Lambda function, refer to Building Lambda Functions in the *AWS Lambda Developer Guide*.

Example Lambda Function

The following example Lambda function receives event data when Amazon Pinpoint runs a campaign, and it sends the campaign's message to Facebook Messenger:

```
"use strict";

var https = require("https");
var q = require("q");

var VERIFY_TOKEN = "my_token";
var PAGE_ACCESS_TOKEN = "EAF...DZD";
/* this constant can be put in a constants file and shared between this function and your
 Facebook Messenger webhook code */
var FACEBOOK_MESSENGER_PSID_ATTRIBUTE_KEY = "facebookMessengerPsid";

exports.handler = function(event, context, callback) {

    var deliverViaMessengerPromises = [];

    if (event.Message && event.Endpoints) {
        for (var endpoint in event.Endpoints) {
            if (isFbookMessengerActive(event.Endpoints[endpoint])) {
                deliverViaMessengerPromises.push(deliverViaMessenger(event.Message,
 event.Endpoints[endpoint].User));
```

```
            }
        }
    }

    /* default OK response */
    var response = {
        body: "ok",
        statusCode: 200
    };

    if (deliverViaMessengerPromises.length > 0) {
        q.all(deliverViaMessengerPromises).done(function() {
            callback(null, response);
        });
    } else {
        callback(null, response);
    }

}

/**
Example Pinpoint Endpoint User object where we've added custom attribute
 facebookMessengerPsid to store the PSID needed by Facebook's API
{
    "UserId": "7a9870b7-493c-4521-b0ca-08bbbc36e595",
    "UserAttributes": {
        "facebookMessengerPsid": [ "1667566386619741" ]
    }
}
**/
function isFbookMessengerActive(endpoint) {
    return endpoint.User && endpoint.User.UserAttributes &&
 endpoint.User.UserAttributes[FACEBOOK_MESSENGER_PSID_ATTRIBUTE_KEY];
}

/**
Sample message object from Pinpoint. This sample was an SMS so it has "smsmessage"
 attribute but this will vary for each messaging channel
{
    "smsmessage": {
        "body": "This message should be intercepted by a campaign hook."
    }
}
**/
function deliverViaMessenger(message, user) {
    var deferred = q.defer();

    var messageText = message["smsmessage"]["body"];
    var pinpointUserId = user.UserId;
    var facebookPsid = user.UserAttributes[FACEBOOK_MESSENGER_PSID_ATTRIBUTE_KEY][0];
    console.log("Sending message for user %s and page %s with message:", pinpointUserId,
 facebookPsid, messageText);

    var messageData = {
        recipient: {
            id: facebookPsid
        },
        message: {
            text: messageText
        }
    };

    var body = JSON.stringify(messageData);
    var path = "/v2.6/me/messages?access_token=" + PAGE_ACCESS_TOKEN;
    var options = {
        host: "graph.facebook.com",
        path: path,
        method: "POST",
        headers: {
            "Content-Type": "application/json"
        }
    };

    var req = https.request(options, httpsCallback);

    req.on("error", function(e) {
        console.log("Error posting to Facebook Messenger: " + e);
        deferred.reject(e);
    });

    req.write(body);
    req.end();

    return deferred.promise;

    function httpsCallback(response) {
        var str = "";
        response.on("data", function(chunk) {
            str += chunk;
        });
        response.on("end", function() {
            console.log(str);
            deferred.resolve(response);
        });
    }
}
```

Assigning a Lambda Function Policy

Before you can use your Lambda function to process your endpoints, you must authorize Amazon Pinpoint to invoke your Lambda function. To grant invocation permission, assign a *Lambda function policy* to the function. A Lambda function policy is a resource-based permissions policy that designates which entities can use your function and what actions those entities can take.

For more information, see Using Resource-Based Policies for AWS Lambda (Lambda Function Policies) in the *AWS Lambda Developer Guide*.

Example Function Policy

The following policy grants permission to the Amazon Pinpoint service principal to use the `lambda:InvokeFunction` action:

```
{
  "Sid": "sid",
  "Effect": "Allow",
  "Principal": {
    "Service": "pinpoint.us-east-1.amazonaws.com"
  },
  "Action": "lambda:InvokeFunction",
  "Resource": "{arn:aws:lambda:us-east-1:account-id:function:function-name}",
  "Condition": {
    "ArnLike": {
      "AWS:SourceArn": "arn:aws:mobiletargeting:us-east-1:account-id:/apps/application-id/
campaigns/campaign-id"
    }
  }
}
```

Your function policy requires a `Condition` block that includes an `AWS:SourceArn` key. This code states which Amazon Pinpoint campaign is allowed to invoke the function. In this example, the policy grants permission to only a single campaign ID. To write a more generic policy, use multi-character match wildcards (*). For example, you can use the following `Condition` block to allow any Amazon Pinpoint campaign in your AWS account to invoke the function:

```
"Condition": {
  "ArnLike": {
    "AWS:SourceArn": "arn:aws:mobiletargeting:us-east-1:account-id:/apps/*/campaigns/*"
  }
}
```

Granting Amazon Pinpoint Invocation Permission

You can use the AWS Command Line Interface (AWS CLI) to add permissions to the Lambda function policy assigned to your Lambda function. To allow Amazon Pinpoint to invoke a function, use the Lambda `add-permission` command, as shown by the following example:

```
$ aws lambda add-permission \
> --function-name function-name \
> --statement-id sid \
> --action lambda:InvokeFunction \
> --principal pinpoint.us-east-1.amazonaws.com \
> --source-arn arn:aws:mobiletargeting:us-east-1:account-id:/apps/application-id/
campaigns/campaign-id
```

If you want to provide a campaign ID for the `--source-arn` parameter, you can look up your campaign IDs by using the Amazon Pinpoint `get-campaigns` command with the AWS CLI. This command requires an `--application-id` parameter. To look up your application IDs, sign in to the Amazon Pinpoint console at https://console.aws.amazon.com/pinpoint/, and go to the **Projects** page. The console shows an **ID** for each project, which is the project's application ID.

When you run the Lambda `add-permission` command, AWS Lambda returns the following output:

```
{
  "Statement": "{\"Sid\":\"sid\",
    \"Effect\":\"Allow\",
    \"Principal\":{\"Service\":\"pinpoint.us-east-1.amazonaws.com\"},
    \"Action\":\"lambda:InvokeFunction\",
    \"Resource\":\"arn:aws:lambda:us-east-1:111122223333:function:function-name\",
    \"Condition\":
      {\"ArnLike\":
        {\"AWS:SourceArn\":
         \"arn:aws:mobiletargeting:us-east-1:111122223333:/apps/application-id/campaigns/
campaign-id\"}}}"
}
```

The `Statement` value is a JSON string version of the statement added to the Lambda function policy.

Assigning a Lambda Function to a Campaign

You can assign a Lambda function to an individual Amazon Pinpoint campaign. Or, you can set the Lambda function as the default used by all campaigns for a project, except for those campaigns to which you assign a function individually.

To assign a Lambda function to an individual campaign, use the Amazon Pinpoint API to create or update a `Campaign` object, and define its `CampaignHook` attribute. To set a Lambda function as the default for all campaigns in a project, create or update the `Settings` resource for that project, and define its `CampaignHook` object.

In both cases, set the following `CampaignHook` attributes:

- `LambdaFunctionName` – The name or ARN of the Lambda function that Amazon Pinpoint invokes to send messages for the campaign.
- `Mode` – Set to `DELIVERY`. With this mode, Amazon Pinpoint uses the function to deliver the messages for a campaign, and it doesn't attempt to send the messages through the standard channels.

Document History for Amazon Pinpoint

The following table describes the documentation for this release of Amazon Pinpoint.

- **Latest documentation update:** November 15, 2018

Change	Description	Date
Voice channel	You can use the new Amazon Pinpoint voice channel to create voice messages and deliver them to your customers over the phone. Currently, you can only send voice messages by using the Amazon Pinpoint SMS and Voice API.	November 15, 2018
EU (Ireland) Availability	Amazon Pinpoint is now available in the EU (Ireland) AWS Region.	October 25, 2018
Events API	Use the Amazon Pinpoint API to record events (p. 18) and associate them with endpoints.	August 7, 2018
Code examples for defining and looking up endpoints	Code examples are added that show you how to define, update, delete, and look up endpoints. Examples are provided for the AWS CLI, AWS SDK for Java, and the Amazon Pinpoint API. For more information, see Defining Your Audience to Amazon Pinpoint (p. 40) and Accessing Audience Data in Amazon Pinpoint (p. 65).	August 7, 2018
Endpoint export permissions	Configure an IAM policy (p. 128) that allows you to export Amazon Pinpoint endpoints to an Amazon S3 bucket.	May 1, 2018
Updated topics for Amazon Pinpoint integration	Integrate Amazon Pinpoint (p. 3) with your Android, iOS, or JavaScript application by using AWS SDKs or libraries.	March 23, 2018
AWS CloudTrail logging	Added information about logging Amazon Pinpoint API calls with CloudTrail (p. 100).	February 6, 2018

Change	Description	Date
Updated service limits	Updated *Limits* (p. 135) with additional information about email limits.	January 19, 2018
Public beta for Amazon Pinpoint extensions	Use AWS Lambda functions to customize segments (p. 84) or create custom messaging channels (p. 143).	November 28, 2017
External ID removed from IAM trust policies	The external ID key is removed from the example trust policy (p. 127) and example Java code (p. 81) for importing segments.	October 26, 2017
Push notification payload limits	The limits include payload sizes for mobile push messages (p. 137).	October 25, 2017
Updated service limits	Added SMS and email channel information to *Limits* (p. 135).	October 9, 2017
ADM and Baidu mobile push	Update your app code to handle push notifications from the Baidu (p. 34) and ADM (p. 33) mobile push channels.	September 27, 2017
User IDs and authentication events with Amazon Cognito user pools.	If you use Amazon Cognito user pools to manage user sign-in in your mobile apps, Amazon Cognito assigns user IDs to endpoints, and it reports authentication events to Amazon Pinpoint.	September 26, 2017
User IDs	Assign user IDs to endpoints to monitor app usage from individual users. Examples are provided for the AWS Mobile SDKs (p. 9) and SDK for Java (p. 45).	August 31, 2017
Authentication events	Report authentication events to learn how frequently users authenticate with your app. Examples are provided in Reporting Events in Your Application (p. 16).	August 31, 2017
Updated sample events	The example events (p. 94) include events that Amazon Pinpoint streams for email and SMS activity.	June 08, 2017
Android session management	Manage sessions in Android apps by using a class provided by the AWS Mobile Hub sample app.	April 20, 2017

Change	Description	Date
Updated monetization event samples	The sample code is updated for reporting monetization events. .	March 31, 2017
Event streams	You can configure Amazon Pinpoint to send your app and campaign events to an Kinesis stream (p. 93).	March 24, 2017
Permissions	See Permissions (p. 109) for information about granting access to Amazon Pinpoint for AWS users in your account and users of your mobile app.	January 12, 2017
Amazon Pinpoint general availability	This release introduces Amazon Pinpoint.	December 1, 2016