Amazon Pinpoint

Developer Guide

Amazon Pinpoint: Developer Guide

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. 108).

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 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 and the tasks that you can perform by using them.

Define audience segments

Reach the right audience for your messages by defining audience segments (p. 151). 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 by using another service or application.

Engage your audience with messaging campaigns

Engage your audience by creating a messaging campaign (p. 160). 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— directly 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, demographics, and more. You can also monitor your message traffic by viewing metrics such as the total number of messages that were sent or opened for a campaign or application. Through the Amazon Pinpoint API, your application can report custom data, which Amazon Pinpoint makes available for analysis, and you can query analytics data for certain standard metrics.

To analyze or store analytics data outside Amazon Pinpoint, you can configure Amazon Pinpoint to stream the data (p. 206) to Amazon Kinesis.

Regional availability

Amazon Pinpoint is available in several AWS Regions in North America, Europe, Asia, and Oceania. In each Region, AWS maintains multiple Availability Zones. These Availability Zones are physically isolated from each other, but are united by private, low-latency, high-throughput, and highly redundant network connections. These Availability Zones enable us to provide very high levels of availability and redundancy, while also minimizing latency.

To learn more about AWSRegions, see Managing AWSRegions in the *Amazon Web Services General Reference*. For a list of all the Regions where Amazon Pinpoint is currently available, see AWS service endpoints in the *Amazon Web Services General Reference*. To learn more about the number of Availability Zones that are available in each Region, see AWS global infrastructure.

Tutorials

The tutorials in this section are intended to show new Amazon Pinpoint users how to complete several important tasks. If you're new to Amazon Pinpoint, or just unfamiliar with certain features, these tutorials are a good place to start.

The tutorials in this guide include tasks that are oriented toward a developer or system administrator audience. These tutorials show you how to perform tasks by using the Amazon Pinpoint API, the AWS SDKs, and the AWS CLI. If you mainly interact with Amazon Pinpoint by using the web-based console, see the Tutorials section of the Amazon Pinpoint User Guide.

Tutorials

Tutorial: Using Postman with the Amazon Pinpoint API

Postman is a popular tool for testing APIs in an easy-to-use graphical environment. You can use Postman to send API requests to any REST API, and to receive responses to your requests. Using Postman is a convenient way to test and troubleshoot the calls that you make to the Amazon Pinpoint API. This tutorial includes procedures for setting up and using Postman with Amazon Pinpoint.

Note
Postman is developed by a third-party company. It isn't developed or supported by Amazon Web Services (AWS). To learn more about using Postman, or for assistance with issues related to Postman, see the Support center on the Postman website.

About this tutorial

This section contains an overview of this tutorial.

Intended Audience

This tutorial is intended for developers and system implementers. You don't have to be familiar with Amazon Pinpoint or Postman to complete the steps in this tutorial. You should be comfortable managing IAM policies and modifying JSON code examples.

The procedures in this tutorial were designed to prevent new users from using API operations that can permanently delete Amazon Pinpoint resources. Advanced users can remove this restriction by modifying the policy that's associated with their IAM users.

Features Used

This tutorial includes usage examples for the following Amazon Pinpoint feature:

- Interacting with the Amazon Pinpoint API by using Postman

Time Required

It should take about 15 minutes to complete this tutorial.

Regional Restrictions

There are no regional restrictions associated with using this solution.

Resource Usage Costs

There's no charge for creating an AWS account. However, by implementing this solution, you might incur AWS usage costs if you use Postman to do any of the following:

- Send email, SMS, mobile push, or voice messages
- Create and send campaigns
- Use the phone number validation feature

For more information about the charges that are associated with using Amazon Pinpoint, see Amazon Pinpoint pricing.

Prerequisites

Before you begin this tutorial, you have to complete the following prerequisites:

- You have to have an AWS account. To create an AWS account, go to https://console.aws.amazon.com/ and choose **Create a new AWS account**.
- The account that you use to sign in to the AWSManagement Console has to be able to create new IAM policies and roles.
- You have to download and install Postman on your computer. You can download Postman from the Postman website.
- After you install Postman on your computer, you have to create a Postman account. When you first start the Postman application, you're prompted to log in or create a new account. Complete the instructions shown on the screen to log in to your account (if you already have one), or to create an account (if you don't already have one).

Step 1: Create IAM policies and roles

The first step in using Postman to test the Amazon Pinpoint API is to create an IAM user. In this section, you create a policy that provides users with the ability to interact with all the Amazon Pinpoint resources. You then create a user account and attach the policy directly to the user account.

Step 1.1: Create an IAM policy

This section shows you how to create an IAM policy. Users and roles that use this policy are able to interact with all of the resources in the Amazon Pinpoint API. It also provides access to resources that are associated with the Email API, as well as the SMS and Voice API.

To create the policy

1. Sign in to the AWSManagement Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies**, and then choose **Create policy**.
3. On the **JSON** tab, paste the following code.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "mobiletargeting:Update*",
                "mobiletargeting:Get*",
                "mobiletargeting:Send*",
                "mobiletargeting:Put*",
                "mobiletargeting:Create*"
            ],
            "Resource": [
                "arn:aws:mobiletargeting:*:123456789012:apps/*",
                "arn:aws:mobiletargeting:*:123456789012:apps/*/campaigns/*",
                "arn:aws:mobiletargeting:*:123456789012:apps/*/segments/*"
            ]
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": [
                "mobiletargeting:TagResource",
                "mobiletargeting:PhoneNumberValidate",
                "mobiletargeting:ListTagsForResource",
                "mobiletargeting:CreateApp"
            ],
            "Resource": "arn:aws:mobiletargeting:*:123456789012:*"
        },
        {
            "Sid": "VisualEditor2",
            "Effect": "Allow",
            "Action": [
                "ses:TagResource",
                "ses:Send*",
                "ses:Create*",
                "ses:Get*",
                "ses:List*",
                "ses:Put*",
                "ses:Update*",
                "sms-voice:SendVoiceMessage",
                "sms-voice:List*",
                "sms-voice:Create*",
                "sms-voice:Get*",
                "sms-voice:Update*"
            ],
            "Resource": "*"
        }
    ]
}
```

In the preceding example, replace *123456789012* with the unique ID for your AWS account.

Note

To protect the data in your Amazon Pinpoint account, this policy only includes permissions that allow you to read, create, and modify resources. It doesn't include permissions that allow you to delete resources. You can modify this policy by using the visual editor in the IAM console. For more information, see Managing IAM policies in the IAM User Guide. You can also use the CreatePolicyVersion operation in the IAM API to update this policy.

Also note that this policy includes permissions that allow you to interact with the `ses` and `sms-voice` services, in addition to the `mobiletargeting` service. The `ses` and `sms-voice` permissions allow you to interact with the Email API and SMS and Voice API, respectively. The `mobiletargeting` permissions allow you to interact with the Amazon Pinpoint API.

4. Choose **Review policy**.
5. For **Name**, enter a name for the policy, such as **`PostmanAccessPolicy`**. Choose **Create policy**.

Step 1.2: Create an IAM user

After you create the policy, you can create an IAM user and attach the policy to it. When you create the user, IAM provides you with a set of credentials that you can use to allow Postman to execute Amazon Pinpoint API operations.

To create the user

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. On the IAM console, in the navigation pane, choose **Users**, and then choose **Add user**.
3. Under **Set user details**, for **User name**, enter a name that identifies the user account, such as **`PostmanUser`**.
4. Under **Select AWS access type**, for **Access type**, choose **Programmatic access**. Then choose **Next: Permissions**.
5. Under **Set permissions**, choose **Attach existing policies directly**. In the list of policies, choose the policy that you created in Step 1.1 (p. 4). Then choose **Next: Tags**.
6. On the **Add tags** page, optionally add tags that help you identify the user. For more information about using tags, see Tagging IAM users and roles in the *IAM User Guide*. Then choose **Next: Review**.
7. On the **Review** page, review and confirm the settings for the user. When you're ready to create the user, choose **Create user**.
8. On the **Success** page, copy the credentials that are shown in the **Access key ID** and **Secret access key** columns.

   Note
   You need to provide both the access key ID and the secret access key later in this tutorial. This is the only time that you're able to view the secret access key, so you should copy it and save it in a safe location.

Step 2: Set up Postman

Now that you've created an IAM user account that's able to access the Amazon Pinpoint API, you can set up Postman. In this section, you create one or more environments in Postman. Next, you import a collection that contains a request template for each of the operations in the Amazon Pinpoint API.

Step 2.1: Create Postman environments

In Postman, an *environment* is a set of variables that are stored as key-value pairs. You can use environments to quickly change the configuration of the requests that you make through Postman, without having to change the API requests themselves.

In this section, you create at least one environment to use with Amazon Pinpoint. Each environment that you create contains a set of variables that are specific to your account in a single AWS Region. If you use the procedures in this section to create more than one environment, you can easily change between Regions by choosing a different environment from the **Environment** menu in Postman.

To create an environment

1. In Postman, on the **File** menu, choose **New**.
2. On the **Create New** window, choose **Environment**.
3. On the **MANAGE ENVIRONMENTS** window, for **Environment Name**, enter **`Amazon Pinpoint - `*`Region Name`***. Replace *`Region Name`* with one of the following values:

   - US East (N. Virginia)
   - US West (Oregon)

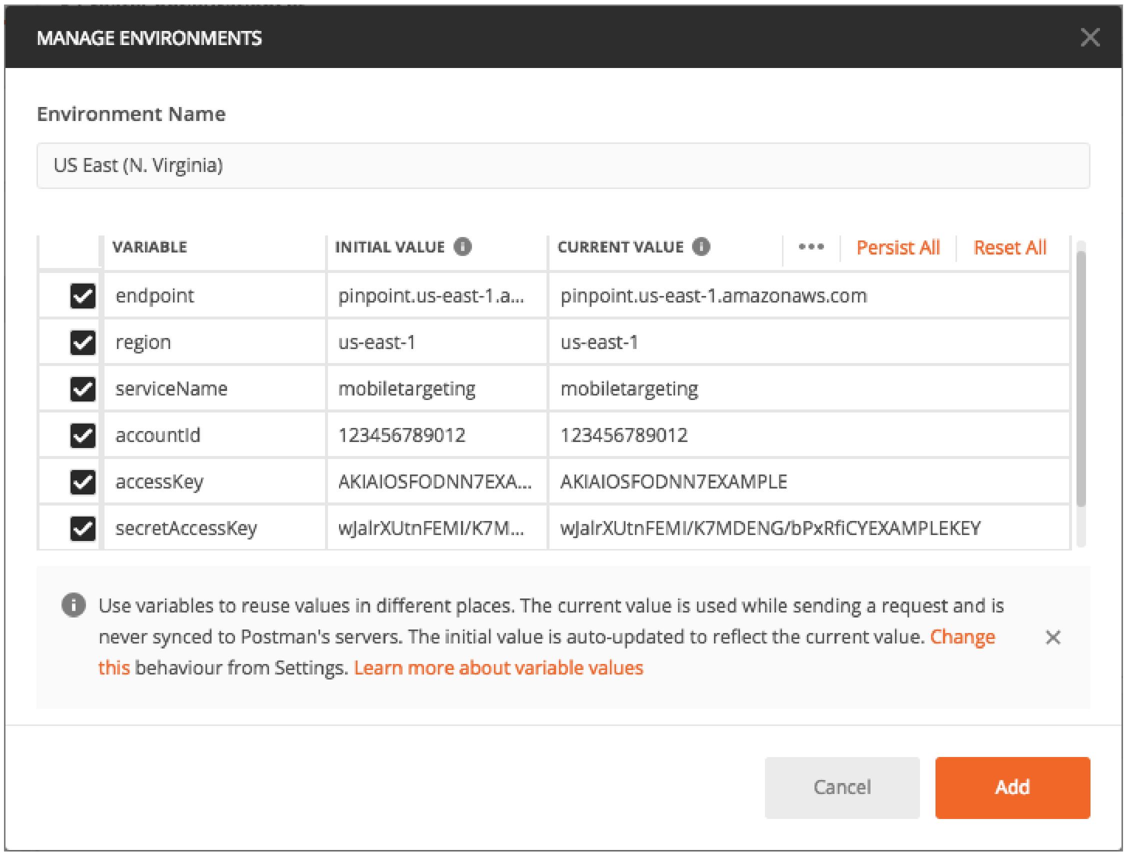
- Asia Pacific (Mumbai)
- Asia Pacific (Sydney)
- Europe (Frankfurt)
- Europe (Ireland)

4. Create six new variables: `endpoint`, `region`, `serviceName`, `accountId`, `accessKey`, and `secretAccessKey`. Use the following table to determine which value to enter in the **Initial Value** column for each variable.

Region	Variable	Initial value
US East (N. Virginia)	`endpoint`	**`pinpoint.us-east-1.amazonaws.com`**
	`region`	**`us-east-1`**
	`serviceName`	**`mobiletargeting`**
	`accountId`	*(your AWS account ID)*
	`accessKey`	*(your IAM access key ID)*
	`secretAccessKey`	*(your IAM secret access key)*
US West (Oregon)	`endpoint`	**`pinpoint.us-west-2.amazonaws.com`**
	`region`	**`us-west-2`**
	`serviceName`	**`mobiletargeting`**
	`accountId`	*(your AWS account ID)*
	`accessKey`	*(your IAM access key ID)*
	`secretAccessKey`	*(your IAM secret access key)*
Asia Pacific (Mumbai)	`endpoint`	**`pinpoint.ap-south-1.amazonaws.com`**
	`region`	**`ap-south-1`**
	`serviceName`	**`mobiletargeting`**
	`accountId`	*(your AWS account ID)*
	`accessKey`	*(your IAM access key ID)*
	`secretAccessKey`	*(your IAM secret access key)*
Asia Pacific (Sydney)	`endpoint`	**`pinpoint.ap-southeast-2.amazonaws.com`**
	`region`	**`ap-southeast-2`**

Region	Variable	Initial value
	`serviceName`	**`mobiletargeting`**
	`accountId`	*(your AWS account ID)*
	`accessKey`	*(your IAM access key ID)*
	`secretAccessKey`	*(your IAM secret access key)*
Europe (Frankfurt)	`endpoint`	**`pinpoint.eu-central-1.amazonaws.com`**
	`region`	**`eu-central-1`**
	`serviceName`	**`mobiletargeting`**
	`accountId`	*(your AWS account ID)*
	`accessKey`	*(your IAM access key ID)*
	`secretAccessKey`	*(your IAM secret access key)*
Europe (Ireland)	`endpoint`	**`pinpoint.eu-west-1.amazonaws.com`**
	`region`	**`eu-west-1`**
	`serviceName`	**`mobiletargeting`**
	`accountId`	*(your AWS account ID)*
	`accessKey`	*(your IAM access key ID)*
	`secretAccessKey`	*(your IAM secret access key)*

After you create these variables, the **MANAGE ENVIRONMENTS** window resembles the example shown in the following image.



Environment Name

US East (N. Virginia)

	VARIABLE	INITIAL VALUE	CURRENT VALUE
✓	endpoint	pinpoint.us-east-1.a...	pinpoint.us-east-1.amazonaws.com
✓	region	us-east-1	us-east-1
✓	serviceName	mobiletargeting	mobiletargeting
✓	accountId	123456789012	123456789012
✓	accessKey	AKIAIOSFODNN7EXA...	AKIAIOSFODNN7EXAMPLE
✓	secretAccessKey	wJalrXUtnFEMI/K7M...	wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY

Use variables to reuse values in different places. The current value is used while sending a request and is never synced to Postman's servers. The initial value is auto-updated to reflect the current value. Change this behaviour from Settings. Learn more about variable values

When you finish, choose **Add**.

Important
The access keys shown in the preceding image are fictitious. Never share your IAM access keys with others.
Postman includes features that enable you to share and export environments. If you use these features, be careful to not share your access key ID and secret access key with anybody who shouldn't have access to these credentials.
For more information, see IAM best practices in the *IAM User Guide*.

5. (Optional) Repeat steps 1–4 for each additional environment that you want to create.

   Tip
   In Postman, you can create as many environments as you need. You can use environments in several ways. For example, you can do all of the following:

   - Create a separate environment for every Region where you need to test the Amazon Pinpoint API.
   - Create environments that are associated with different AWS accounts.
   - Create environments that use credentials that are associated with other IAM users.

6. When you finish creating environments, proceed to the next section.

Step 2.2: Create an Amazon Pinpoint collection in Postman

In Postman, a *collection* is a group of API requests. Requests in a collection are typically united by a common purpose. In this section, you create a new collection that contains a request template for each operation in the Amazon Pinpoint API.

To create the Amazon Pinpoint collection

1. In Postman, on the **File** menu, choose **Import**.
2. On the **Import** window, choose **Import From Link**, and then enter the following URL: https://raw.githubusercontent.com/awsdocs/amazon-pinpoint-developer-guide/master/Amazon%20Pinpoint.postman_collection.json.

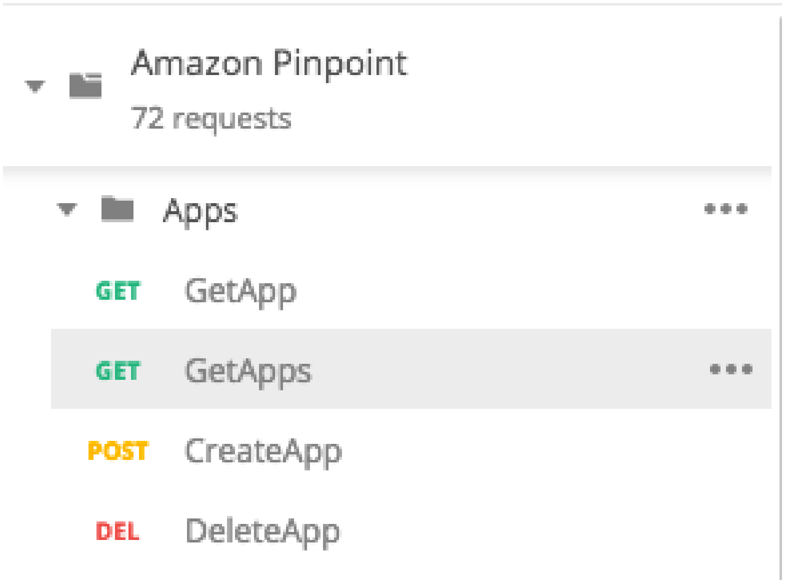
   Choose **Import**. Postman imports the Amazon Pinpoint collection, which contains 120 example requests.

Step 2.3: Test your Postman configuration

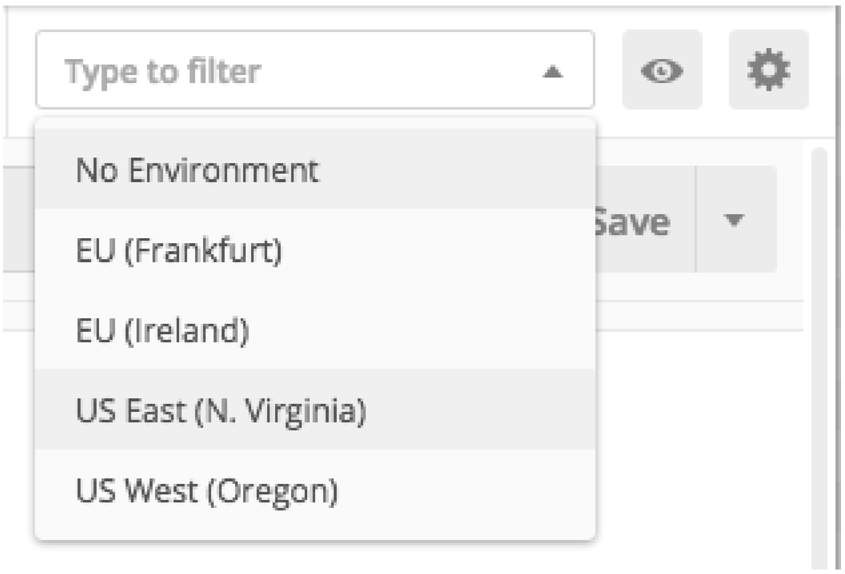
After you import the Amazon Pinpoint collection, you should perform a quick test to make sure that all of the components are properly configured. You can test your configuration by submitting a `GetApps` request. This request returns a list of all of the projects that exist in your Amazon Pinpoint account in the current AWS Region. This request doesn't require any additional configuration, so it's a good way to quickly test your configuration.

To test the configuration of the Amazon Pinpoint collection

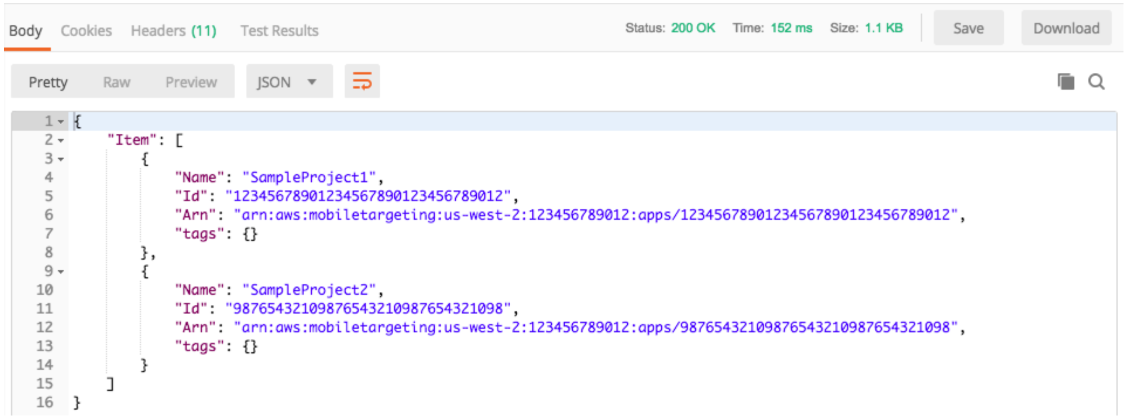
1. In the navigation pane, expand the **Amazon Pinpoint** collection, and then expand the **Apps** folder.
2. In the list of requests, choose **GetApps**.



3. Use the **Environment** selector to choose the environment that you created in Step 2.1 (p. 6), as shown in the following image.



4. Choose **Send**. If the request is sent successfully, the response pane shows a status of `200 OK`. You see a response that resembles the example in the following image.



```
{
    "Item": [
        {
            "Name": "SampleProject1",
            "Id": "12345678901234567890123456789012",
            "Arn": "arn:aws:mobiletargeting:us-west-2:123456789012:apps/12345678901234567890123456789012",
            "tags": {}
        },
        {
            "Name": "SampleProject2",
            "Id": "98765432109876543210987654321098",
            "Arn": "arn:aws:mobiletargeting:us-west-2:123456789012:apps/98765432109876543210987654321098",
            "tags": {}
        }
    ]
}
```

This response shows a list of all of the Amazon Pinpoint projects that exist in your account in the Region that you chose in step 3.

Troubleshooting

When you submit your request, you might see an error. See the following list for several common errors that you might encounter, and for steps that you can take to resolve them.

Error message	Problem	Resolution
Could not get any response There was an error connecting to https://%7B%7Bendpoint%7D%7D/v1/apps.	There is no current value for the `{{endpoint}}` variable, which is set when you choose an environment.	Use the environment selector to choose an environment.
The security token included in the request is invalid.	Postman wasn't able to find the current value of your access key ID or secret access key.	Choose the gear icon near the environment selector, and then choose the current environment.

Error message	Problem	Resolution
		Make sure that the `accessKey` and `secretAccessKey` values appear in both the **INITIAL VALUE** and **CURRENT VALUE** columns, and that you entered the credentials correctly.
"Message": "User: arn:aws:iam::123456789012:user/PinpointPostmanUser is not authorized to perform: mobiletargeting:GetApps on resource: arn:aws:mobiletargeting:us-west-2:123456789012:*"	The IAM policy associated with your user doesn't include the appropriate permissions.	Make sure that your IAM user has the permissions that are described in Step 1.1 (p. 4), and that you provided the correct credentials when you created the environment in Step 2.1 (p. 6).

Step 3: Send additional requests

When you finish configuring and testing Postman, you can start sending additional requests to the Amazon Pinpoint API. This section includes information that you need to know before you start sending requests. It also includes two sample requests that help you understand how to use the Amazon Pinpoint collection.

Important
When you complete the procedures in this section, you submit requests to the Amazon Pinpoint API. These requests are capable of creating new resources in your Amazon Pinpoint account, modifying existing resources, sending messages, changing the configuration of your Amazon Pinpoint projects, and using other Amazon Pinpoint features. Use caution when you execute these requests.

About the examples in the Amazon Pinpoint Postman collection

You have to configure most of the operations in the Amazon Pinpoint Postman collection before you can use them. For `GET` and `DELETE` operations, you typically only need to modify the variables that are set on the **Pre-request Script** tab.

Note
When you use the IAM policy that's shown in Step 1.1 (p. 4), you can't execute any of the `DELETE` requests that are included in this collection.

For example, the `GetCampaign` operation requires you to specify a `projectId` and a `campaignId`. On the **Pre-request Script** tab, both of these variables are present, and are populated with example values. Delete the example values and replace them with the appropriate values for your Amazon Pinpoint project and campaign.

Of these variables, the most commonly used is the `projectId` variable. The value for this variable should be the unique identifier for the project that your request applies to. To get a list of these identifiers for your projects, you can refer to the response to the `GetApps` request that you sent in the preceding step of this tutorial. (In Amazon Pinpoint, a "project" is the same thing as an "app" or "application.") In that response, the `Id` field provides the unique identifier for a project. To learn more about the `GetApps` operation and the meaning of each field in the response, see Apps in the *Amazon Pinpoint API Reference*.

For `POST` and `PUT` operations, you also need to modify the request body to include the values that you want to send to the API. For example, when you submit a `CreateApp` request (which is a `POST` request), you have to specify a name for the project that you create. You can modify the request on the **Body** tab. In this example, replace the value next to `"Name"` with the name of the project. If you want to add tags to the project, you can specify them in the `tags` object. Or, if you don't want to add tags, you can delete the entire `tags` object.

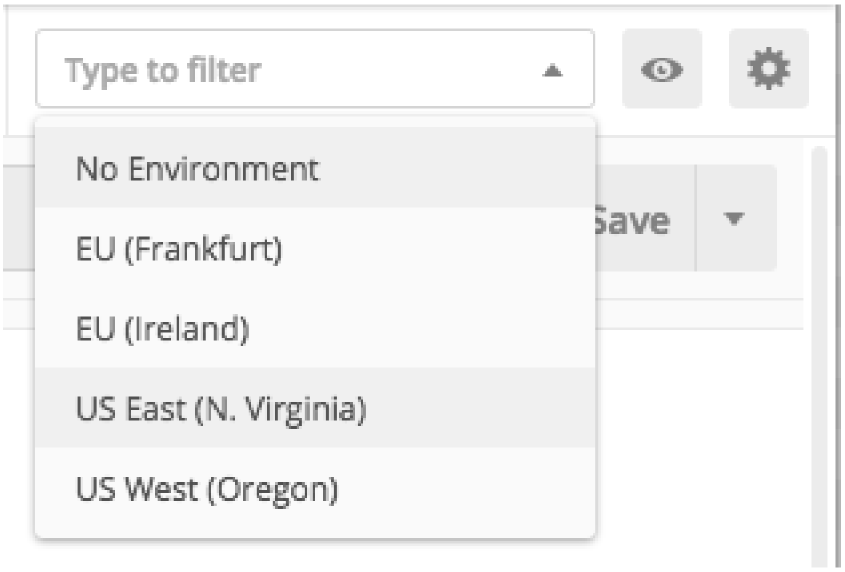
> **Note**
> The `UntagResource` operation also requires you to specify URL parameters. You can specify these parameters on the **Params** tab. Replace the values in the **VALUE** column with the tags that you want to delete for the specified resource.

Example request: Creating a project by using the `CreateApp` operation

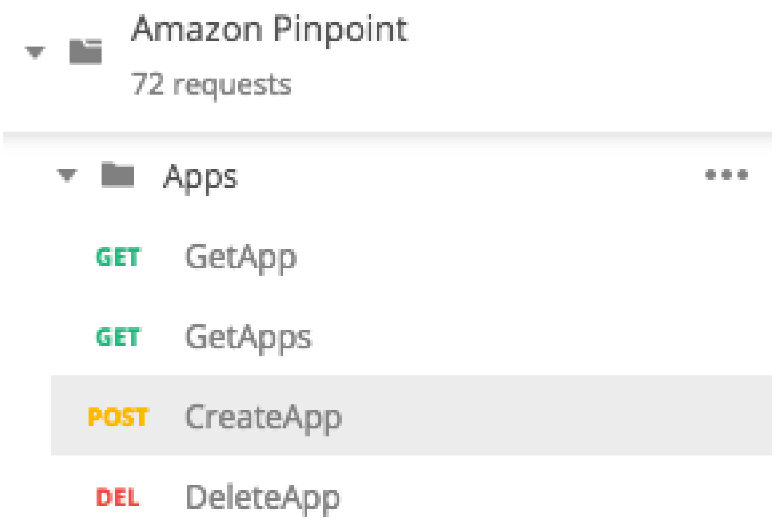
Before you create segments and campaigns in Amazon Pinpoint, you first have to create a project. In Amazon Pinpoint, a *project* consists of segments, campaigns, configurations, and data that are united by a common purpose. For example, you could use a project to contain all of the content that's related to a particular app, or to a specific brand or marketing initiative. When you add customer information to Amazon Pinpoint, that information is associated with a project.

To create a project by sending a CreateApp API request

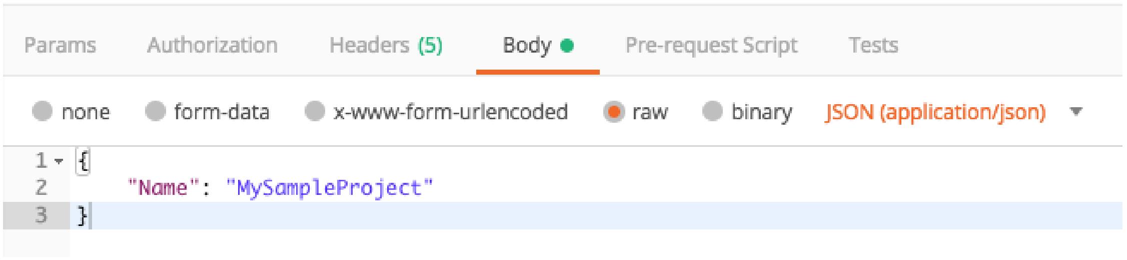
1. On the **Environments** menu, choose the AWS Region that you want to create the project in, as shown in the following image.



2. In the **Apps** folder, choose the **CreateApp** operation, as shown in the following image.

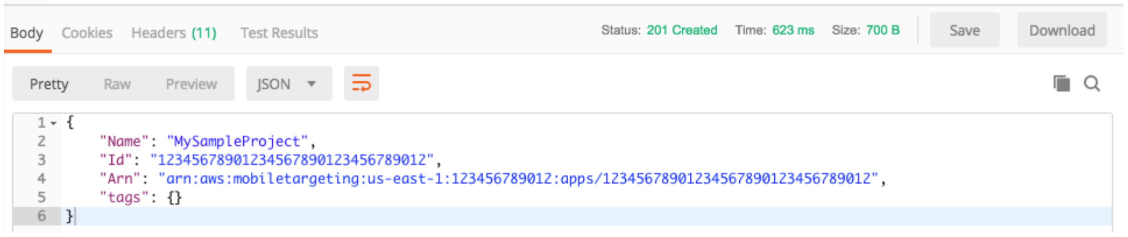


3. On the **Body** tab, next to `"Name"`, replace the placeholder value (`"string"`) with a name for the campaign, such as **`"MySampleProject"`**.
4. Delete the comma that after the campaign name, and then delete the entire `tags` object on lines 3 through 5. When you finish, your request should resemble the example that's shown in the following image.



```
{
    "Name": "MySampleProject"
}
```

5. Choose **Send**. If the campaign is created successfully, the response pane shows a status of `201 Created`. You see a response that resembles the example in the following image.



```
{
    "Name": "MySampleProject",
    "Id": "12345678901234567890123456789012",
    "Arn": "arn:aws:mobiletargeting:us-east-1:123456789012:apps/12345678901234567890123456789012",
    "tags": {}
}
```

Example: Sending an email by using the `SendMessages` operation

It's very common to use the Amazon Pinpoint `SendMessages` API to send transactional messages. One advantage to sending messages by using the `SendMessages` API (as opposed to creating campaigns), is that you can use the `SendMessages` API to send messages to any address (such as an email address, phone number, or device token). The address that you send messages to doesn't have to exist in your Amazon Pinpoint account already. Compare this to sending messages by creating campaigns. Before you send a campaign in Amazon Pinpoint, you have to add endpoints to your Amazon Pinpoint account, create segments, create the campaign, and execute the campaign.

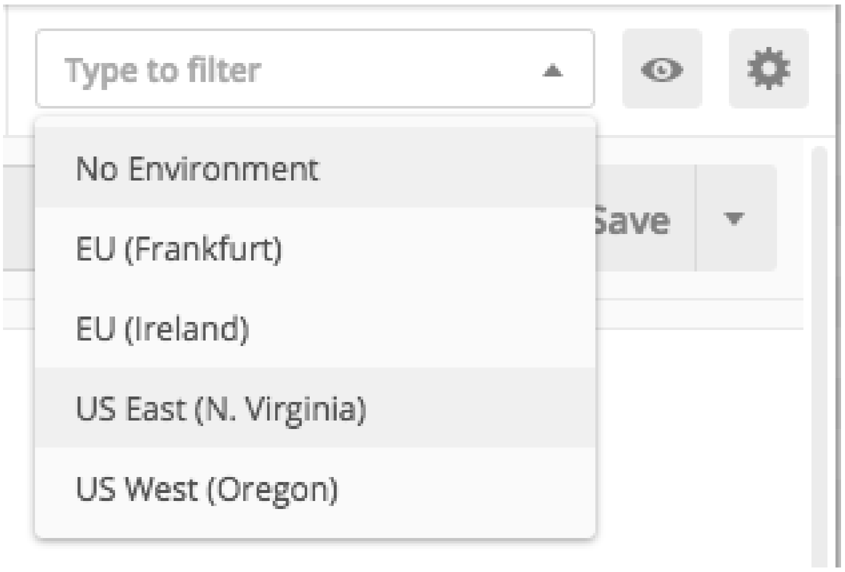
The example in this section shows you how to send a transactional email message directly to a specific email address. You can modify this request to send messages through other channels, such as SMS, mobile push, or voice.

To send an email message by submitting a SendMessages request

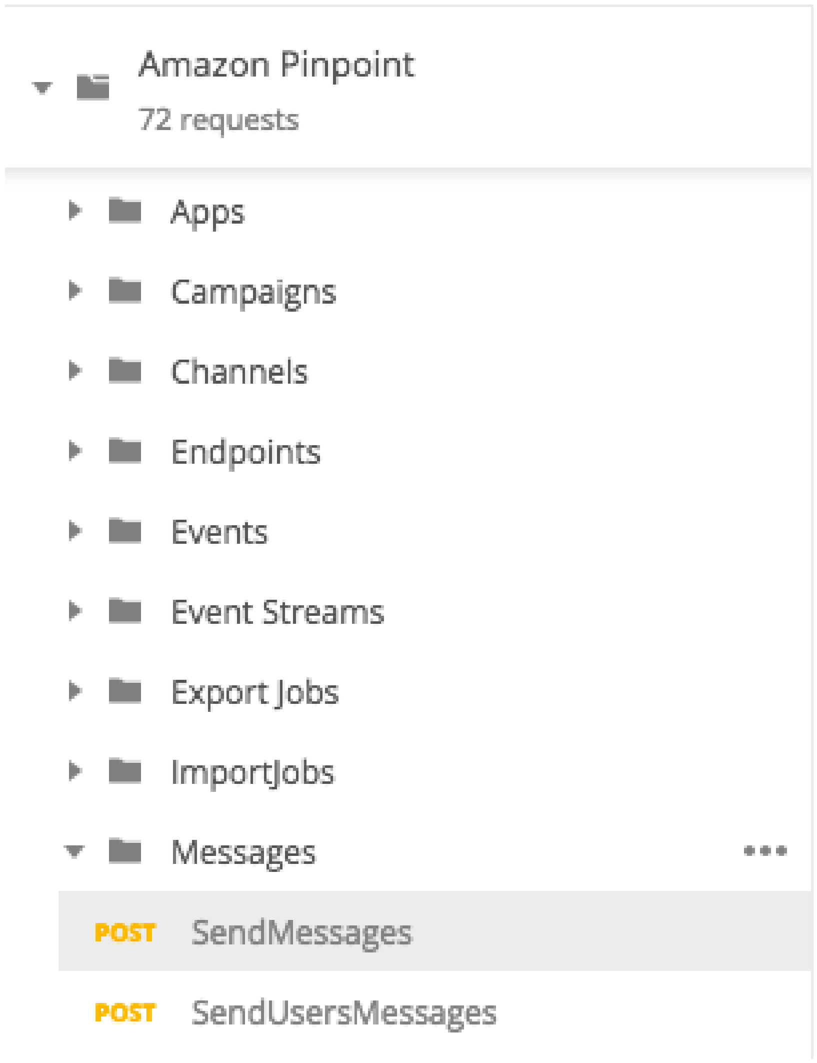
1. Verify the email address or domain that you want to use to send the message. For more information, see Verifying email identities in the *Amazon Pinpoint User Guide*.

   Note
   In Amazon Pinpoint, you can only send email from addresses or domains that you've verified. You won't be able to complete the procedure in this section until you verify an email address.

2. On the **Environments** menu, choose the AWS Region that you want to send the message from, as shown in the following image.



3. In the **Messages** folder, choose the **SendMessages** operation.



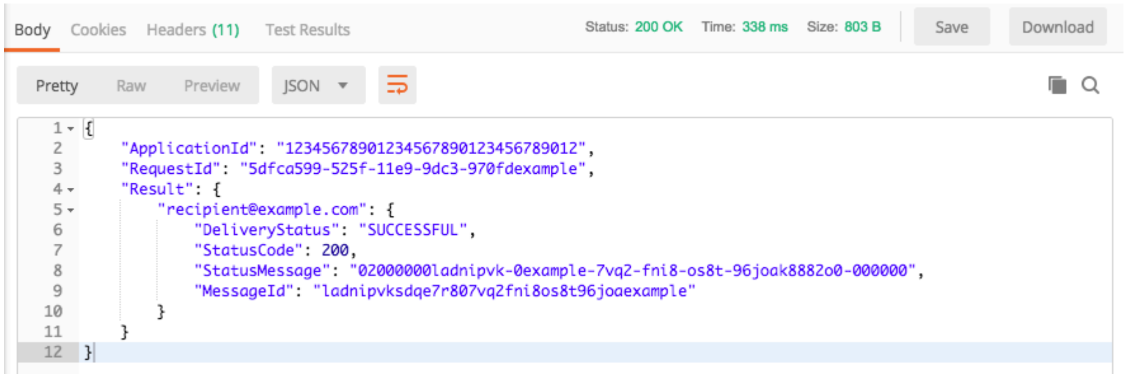
4. On the **Pre-request Script** tab, replace the value of the `projectId` variable with the ID of a project that already exists in the Region that you selected in step 2 of this section.
5. On the **Body** tab, delete the example request that's shown in the request editor. Paste the following code:

```
{
    "MessageConfiguration":{
        "EmailMessage":{
            "FromAddress":"sender@example.com",
            "SimpleEmail":{
                "Subject":{
                    "Data":"Sample Amazon Pinpoint message"
                },
                "HtmlPart":{
                    "Data":"<h1>Test message</h1><p>This is a sample message sent from
 <a href=\"https://aws.amazon.com/pinpoint\">Amazon Pinpoint</a> using the SendMessages
 API.</p>"
                },
                "TextPart":{
                    "Data":"This is a sample message sent from Amazon Pinpoint using
 the SendMessages API."
                }
            }
        }
    },
    "Addresses":{
        "recipient@example.com": {
            "ChannelType": "EMAIL"
        }
    }
}
```

6. In the preceding code, replace *sender@example.com* with your verified email address. Replace *recipient@example.com* with the address that you want to send the message to.

   Note
   If your account is still in the Amazon Pinpoint email sandbox, you can only send email to addresses or domains that are verified in your Amazon Pinpoint account. For more information about having your account removed from the sandbox, see Requesting production access for email in the *Amazon Pinpoint User Guide*.

7. Choose **Send**. If the message is sent successfully, the response pane shows a status of `200 OK`. You see a response that resembles the example in the following image.



```
{
    "ApplicationId": "12345678901234567890123456789012",
    "RequestId": "5dfca599-525f-11e9-9dc3-970fdexample",
    "Result": {
        "recipient@example.com": {
            "DeliveryStatus": "SUCCESSFUL",
            "StatusCode": 200,
            "StatusMessage": "02000000ladnipvk-0example-7vq2-fni8-os8t-96joak888200-000000",
            "MessageId": "ladnipvksdqe7r807vq2fni8os8t96joaexample"
        }
    }
}
```

Tutorial: Importing data into Amazon Pinpoint from external sources

Before you can create segments and send campaign messages, you have to create endpoints. In Amazon Pinpoint, *endpoints* are destinations that you send messages to—such as email addresses, mobile device identifiers, or mobile phone numbers. There are several ways to add endpoints to Amazon Pinpoint. For example, your web or mobile apps can use an AWS Mobile SDK to automatically write endpoint data to Amazon Pinpoint.

Alternatively, you can import lists of existing customers into Amazon Pinpoint by using the console, or by using the CreateImportJob API operation. However, when you create an import job, the files that you import have to be formatted in a way that Amazon Pinpoint can interpret. Additionally, if the data that you want to import includes multiple contact methods (such as email addresses and phone numbers) for each customer, you have to create separate endpoints for each contact method. You can then unite those endpoints by applying a common user ID attribute to each one.

The solution that's described in this tutorial is intended to simplify the process of bringing customer information into Amazon Pinpoint from an external system, such as Salesforce, Segment, Braze, or Adobe Marketing Cloud. This tutorial includes a small sample file that's based on data that was originally exported from Salesforce.

Note

AWS and Amazon Pinpoint aren't associated with any of the products or services that are mentioned in the preceding paragraph. To learn more about exporting data from these systems, see their official documentation:

- Salesforce: See Creating a custom report on the Salesforce website.
- Segment: See What are my data export options? on the Segment website.
- Braze: See Exporting to CSV in the User Guide on the Braze website.
- Adobe Marketing Cloud: See Exporting data using segment export on the Adobe Experience Cloud Help website.

After you implement this solution, you can easily send customer data to Amazon Pinpoint by uploading your source file to a specific Amazon S3 bucket. When you add a new source file to the Amazon S3 bucket, Amazon S3 triggers an AWS Lambda function. This function splits the source file into several smaller files, and then moves these files to a second Amazon S3 bucket. When files are added to the second Amazon S3 bucket, it triggers another Lambda function.

The second function processes each of the smaller input files by doing the following:

- It creates a separate record for each endpoint (email address, phone number) that it finds in the file.
- It changes the headers in the incoming spreadsheet to the record names that Amazon Pinpoint expects.
- It uses the phone number validation feature of Amazon Pinpoint to properly format the phone numbers that it finds. If the phone number validation service discovers that a phone number can receive SMS messages, then it creates the endpoint as an SMS endpoint. Otherwise, it creates the endpoint as a voice endpoint.
- It changes some data to use the format that Amazon Pinpoint expects. For example, Amazon Pinpoint expects you to specify country names in ISO 3166 alpha-2 format. If your input includes country names, this function automatically converts them to the correct format. Additionally, Amazon Pinpoint expects you to specify phone numbers in E.164 format. It obtains this value from the phone number validation step.
- It stores the processed files in a third Amazon S3 bucket.

Because this function only processes a small section of your data, it can run quickly each time it's invoked. Also, each instance of the function runs concurrently, which makes it possible to process large amounts of data in a relatively short amount of time.

When files are added to the third and final Amazon S3 bucket, an additional Lambda function is triggered. This function takes the processed files and uses them to create import jobs in Amazon Pinpoint. When these import jobs are complete, you can start sending campaigns to the imported endpoints.

The following diagram shows the flow of data in this solution.

Intended audience

This tutorial is intended for developers and system implementers. You don't have to be familiar with Amazon Pinpoint to complete the steps in this tutorial.

To implement the solution that's described in this tutorial, you have to install Python and the PIP package manager on your computer. This enables you to download some of the required packages. This tutorial includes all of the code that you need in order to complete it. However, to make this solution work for your specific use case, you might have to modify some of the Python code that's included in the tutorial. Finally, you have to be able to create new IAM policies and users.

Features used

This tutorial shows you how to import customer data by using custom Lambda functions to process your data. In addition to modifying the format of some of the data in your source files, the solution described in this tutorial also uses the phone number validation service that's built into Amazon Pinpoint.

After the data is processed, a Lambda function imports your customer data into Amazon Pinpoint by using the `CreateImportJob` API operation.

Time required

It should take 60–90 minutes to complete the procedures in this tutorial. However, you should plan to spend additional time customizing this solution to fit your unique use case.

Regional restrictions

There are no regional restrictions associated with using this solution.

Resource usage costs

There's no charge for creating an AWS account. However, by implementing this solution, you might incur some or all of the costs that are listed in the following table.

Description	Cost (US dollars)
Lambda usage	This solution uses three Lambda functions that interact with the Amazon Pinpoint API. When you call a Lambda function, you're charged based on the number of requests for your functions, for the time it takes your code to execute, and for the amount of memory that your functions use. The number of requests, amount of compute time, and amount of memory required depends on the number of records that you're importing. Your usage of Lambda in implementing this solution might be included in the AWS Free Tier. For more information, see AWS Lambda pricing.
Amazon S3 usage	You pay $0.023–0.025 per GB of data that you store in Amazon S3, depending on which AWS Region you use. You also pay $0.005–0.0055, depending on the Region, for every 1,000 `PUT` requests that you make to Amazon S3. Your usage of Amazon S3 in implementing this solution might be included in the AWS Free Tier. For more information, see Amazon Simple Storage Service pricing.
Phone number validation usage	The solution in this tutorial uses the phone number validation feature in Amazon Pinpoint to verify that each phone number in your incoming data is valid and properly formatted. You pay $0.006 for each phone number validation request. This tutorial includes a sample file that contains 10 endpoints. Each time you execute the solution that's shown in this tutorial, you

Description	Cost (US dollars)
	pay $0.06 for your use of the phone number validation service.

Prerequisites

Before you begin this tutorial, complete the following prerequisites:

- Create an AWS account, if you don't have one already. To create an AWS account, go to https://console.aws.amazon.com/, and choose **Create a new AWS account**.
- Make sure that the account that you use to sign in to the AWSManagement Console is able to create new IAM policies and roles. If you're unsure, ask your system administrator.
- Download and install Python and the PIP package manager on your computer. This solution was tested using Python version 3.7.3 and PIP version 19.0.3. For more information about downloading and installing Python and PIP, see the Beginner's guide to Python on the Python website.
- Create a new project in your Amazon Pinpoint account, or identify an existing project that you want to import segments into. For more information about creating projects, see Getting started with Amazon Pinpoint in the *Amazon Pinpoint User Guide*.

Step 1: Create an Amazon S3 bucket

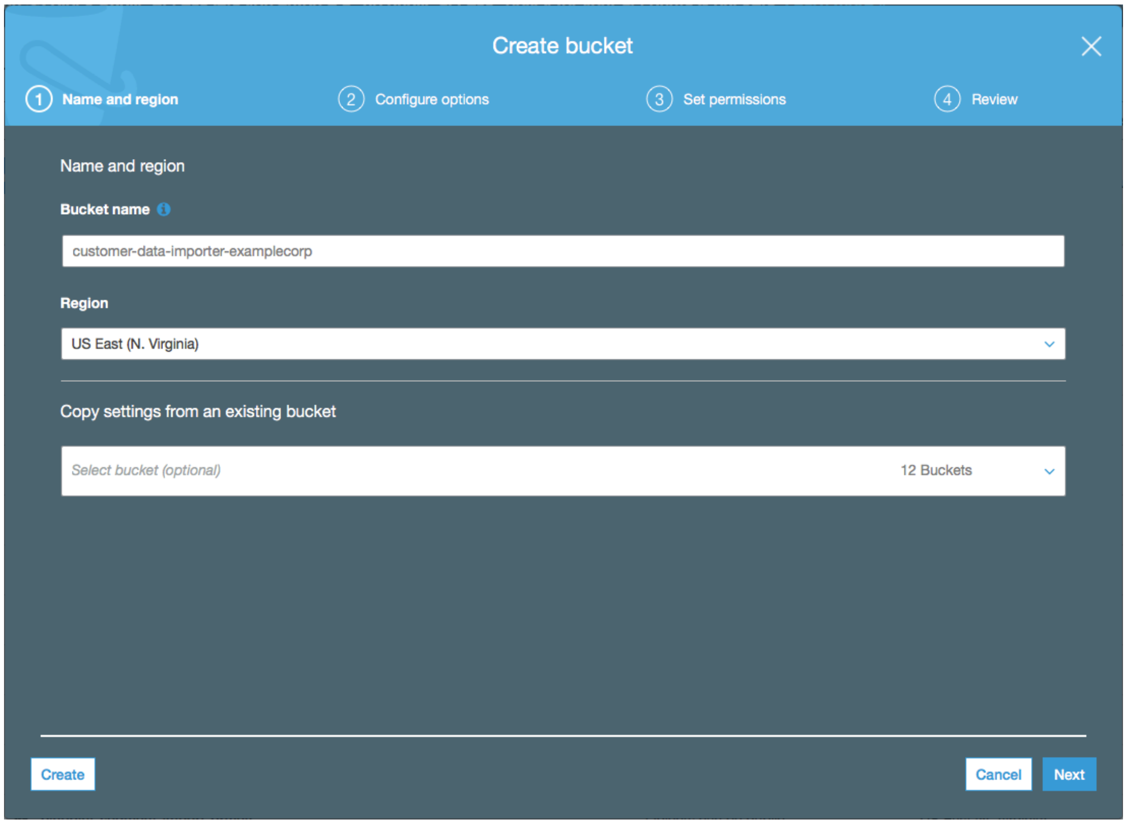
In this solution, you upload files that you want to import into an `input` folder in an Amazon S3 bucket. When you upload a file into this folder, Amazon S3 triggers a Lambda function. This function moves the input file into an `archive` folder. It also creates several smaller files in a `to_process` folder. The first step in implementing this solution is to create an Amazon S3 bucket. Next, you create an `input` folder in that bucket.

Step 1.1: Create an Amazon S3 bucket and input folder

Complete the following procedure to create a new Amazon S3 bucket that contains a folder named `input`.

To create an Amazon S3 bucket

1. Open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. Choose **Create bucket**.
3. For **Bucket name**, enter a unique name for the bucket, as shown in the following image.



Name and region

Bucket name: customer-data-importer-examplecorp

Region: US East (N. Virginia)

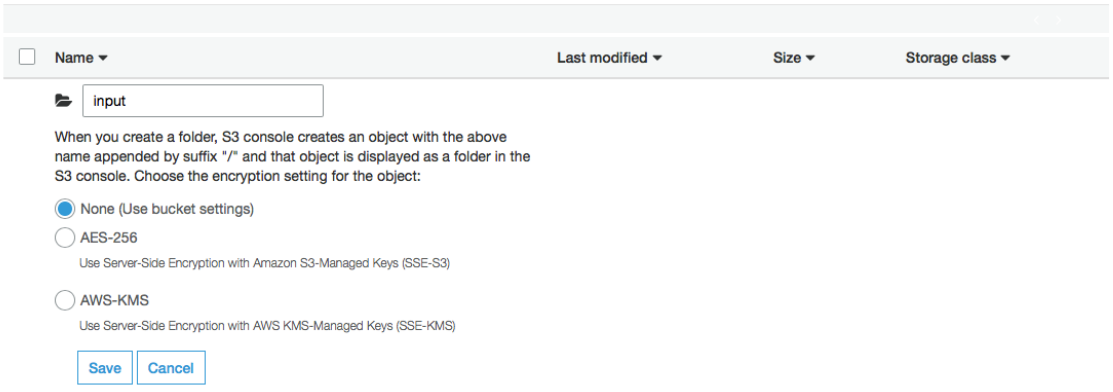
Copy settings from an existing bucket

Select bucket (optional) 12 Buckets

Tip
The name that you specify has to meet all of the following requirements:

- It has to be unique across all of AWS.
- It has to contain at least 3 characters, and no more than 63 characters.
- It can only contain lowercase ASCII letters (a–z), numbers (0–9), periods (.), and dashes (-).
- The first character in the bucket name has to be a letter or a number.
- The name of the bucket can't be formatted like an IP address (such as 192.0.2.0).

4. Choose **Create**.
5. In the list of buckets, choose the bucket that you just created.
6. Choose **Create folder**.
7. For the folder name, enter `input`, as shown in the following image. Choose **Save**.



input

When you create a folder, S3 console creates an object with the above name appended by suffix "/" and that object is displayed as a folder in the S3 console. Choose the encryption setting for the object:

● None (Use bucket settings)

○ AES-256
Use Server-Side Encryption with Amazon S3-Managed Keys (SSE-S3)

○ AWS-KMS
Use Server-Side Encryption with AWS KMS-Managed Keys (SSE-KMS)

Step 2: Create IAM roles

The next step in implementing this solution is to configure policies and roles in AWS Identity and Access Management (IAM). For this solution, you need to create the following roles and policies:

- A role that can read and write from a specific set of Amazon S3 buckets.
- A role that can be passed to Amazon Pinpoint that allows Amazon Pinpoint to import segment data from your Amazon S3 bucket when you create an import job.
- A policy that can perform certain actions in your Amazon Pinpoint account.

The policies that you create in this section use the principal of granting *least privilege*. In other words, they only grant the specific permissions that are required to complete a specific task, and no more.

Step 2.1: Create a policy and role for reading and writing from Amazon S3

The first policy that you need to create is one that allows Lambda to view the contents of a folder in an Amazon S3 bucket. It also allows Lambda to read files in that bucket, and to move those files to other folders in the same bucket.

To create the policy

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies**, and then choose **Create policy**.
3. On the **JSON** tab, paste the following code:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "logs:CreateLogStream",
                "s3:ListBucket",
                "s3:DeleteObject",
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:*:*:*",
                "arn:aws:s3:::bucket-name/*",
                "arn:aws:s3:::bucket-name"
            ]
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": "logs:CreateLogGroup",
            "Resource": "arn:aws:logs:*:*:*"
        },
        {
            "Sid": "VisualEditor2",
            "Effect": "Allow",
            "Action": [
                "s3:GetAccountPublicAccessBlock",
                "s3:ListAllMyBuckets",
                "s3:HeadBucket"
            ],
            "Resource": "*"
        }
    ]
}
```

In the preceding example, replace *bucket-name* with the name of the bucket that you created in Step 1 (p. 20) of this tutorial.

4. Choose **Review policy**.
5. For **Name**, enter **`ImporterS3Policy`**. Choose **Create policy**.

When you finish creating the policy, you can create a role that uses it.

To create the role

1. In the navigation pane, choose **Roles**, and then choose **Create role**.
2. Under **Choose the service that will use this role**, choose **Lambda**, and then choose **Next: Permissions**.
3. Under **Attach permissions policies**, choose **ImporterS3Policy**, and then choose **Next: Tags**.
4. Choose **Next: Review**.
5. On the **Review** page, for **Name**, enter **`ImporterS3Role`**. Choose **Create role**.

Step 2.2: Create a policy and role for importing from Amazon S3

To use the `CreateImportJob` API operation, you have to provide an IAM role that allows Amazon Pinpoint to read and write from your Amazon S3 bucket. This role has to be passed from your user role to Amazon Pinpoint. To enable the role to be passed to Amazon Pinpoint, you have to add a *trust policy* to the role.

To create the policy

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies**, and then choose **Create policy**.
3. On the **JSON** tab, paste the following code:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "s3:PutAccountPublicAccessBlock",
                "s3:GetAccountPublicAccessBlock",
                "s3:ListAllMyBuckets",
                "s3:HeadBucket"
            ],
            "Resource": "*"
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": "s3:*",
            "Resource": [
                "arn:aws:s3:::bucket-name/*",
                "arn:aws:s3:::bucket-name"
            ]
        }
    ]
}
```

In the preceding example, replace *bucket-name* with the name of the bucket that you created in Step 1 (p. 20) of this tutorial.

4. Choose **Review policy**.
5. For **Name**, enter **`ImporterS3PassthroughPolicy`**. Choose **Create policy**.

Next, you create a role that uses this policy, and then attach a trust policy to the role.

To create the role and trust policy

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**, and then choose **Create role**.
3. Under **Choose the service that will use this role**, choose **Lambda**, and then choose **Next: Permissions**.
4. Under **Attach permissions policies**, choose **ImporterS3PassthroughPolicy**, and then choose **Next: Tags**.
5. Choose **Next: Review**.
6. On the **Review** page, for **Name**, enter `PinpointSegmentImport`. Choose **Create role**.
7. In the list of roles, choose the **PinpointSegmentImport** role that you just created.
8. On the **Trust relationships** tab, choose **Edit trust relationship**.
9. Remove the code in the policy editor, and then paste the following code:

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

10. Choose **Update trust policy**.

Step 2.3: Create a policy and role for using Amazon Pinpoint resources

The next policy that you need to create is one that allows Lambda to interact with Amazon Pinpoint. Specifically, this policy allows Lambda to call the Amazon Pinpoint phone number validation service, and to create import jobs in Amazon Pinpoint. It also allows Amazon Pinpoint to read from your Amazon S3 bucket when you create an import job.

To create the policy

1. In the navigation pane, choose **Policies**, and then choose **Create policy**.
2. On the **JSON** tab, paste the following code:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "mobiletargeting:CreateImportJob",
                "mobiletargeting:GetImportJobs",
                "iam:GetRole",
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:mobiletargeting:us-
east-1:123456789012:apps/01234567890123456789012345678901",
                "arn:aws:mobiletargeting:us-
east-1:123456789012:apps/01234567890123456789012345678901/*",
                "arn:aws:iam::123456789012:role/PinpointSegmentImport"
            ]
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": "mobiletargeting:PhoneNumberValidate",
            "Resource": "arn:aws:mobiletargeting:us-east-1:123456789012:phone/number/
validate"
        }
    ]
}
```

In the preceding example, do the following:

- Replace *`us-east-1`* with the AWS Region that you use Amazon Pinpoint in.
- Replace *`123456789012`* with your AWS account ID.
- Replace *`01234567890123456789012345678901`* with the Amazon Pinpoint project ID that you want to import contacts into. The application ID that you specify has to exist in your Amazon Pinpoint account in the specified Region.

Note
You can use wildcards for any of these values. Using wildcards makes this policy more flexible, but also reduces the security of the policy because it doesn't strictly conform to the principle of least privilege.

3. Choose **Review policy**.
4. For **Name**, enter **`ImporterPinpointPolicy`**. Choose **Create policy**.

When you finish creating the policy, you can create a role that uses it. You also add the `ImporterS3Policy` to the role.

To create the role

1. In the navigation pane, choose **Roles**, and then choose **Create role**.
2. Under **Choose the service that will use this role**, choose **Lambda**, and then choose **Next: Permissions**.
3. Under **Attach permissions policies**, choose **ImporterS3Policy** and **ImporterPinpointPolicy**, and then choose **Next: Tags**.
4. Choose **Next: Review**.

5. On the **Review** page, for **Name**, enter `ImporterPinpointRole`. Choose **Create role**.

Step 3: Create a package that contains the required Python libraries

The solution that's documented in this tutorial uses several libraries that aren't included with the standard Python package that Lambda uses. To use these libraries, you first have to download them on your computer. Next, you create a `.zip` archive that contains all of the libraries. Finally, you upload this archive in Lambda so that you can call the libraries from your functions.

> **Note**
> This procedure assumes that you've already installed Python. Python is included by default with most recent Linux, macOS, or Unix distributions. If you use Windows, you can download Python from the Python releases for Windows page on the Python website. This solution was tested using Python version 3.7.3 on macOS High Sierra, Ubuntu 18.04 LTS, and Windows Server 2019.

To download the libraries that are required for this tutorial

1. At the command line, enter the following command to install the `virtualenv` package:

   ```
   python -m pip install virtualenv
   ```

2. Enter the following command to create a new directory for the virtual environment:

   Linux, macOS, or Unix

   ```
   mkdir ~/pinpoint-importer
   ```

   Windows PowerShell

   ```
   new-item pinpoint-importer -itemtype directory
   ```

3. Enter the following command to change to the `pinpoint-importer` directory:

   ```
   cd pinpoint-importer
   ```

4. Enter the following command to create and initialize a virtual environment called `venv`:

   ```
   python -m virtualenv venv
   ```

5. Enter the following command to activate the virtual environment:

   Linux, macOS, or Unix

   ```
   source venv/bin/activate
   ```

   Windows PowerShell

   ```
   .\venv\Scripts\activate
   ```

   Your command prompt changes to show that the virtual environment is active.

6. Enter the following command to install the packages that are required for this tutorial:

```
pip install s3fs jmespath s3transfer six python-dateutil docutils
```

7. Enter the following command to deactivate the virtual environment:

```
deactivate
```

8. Enter the following command to create an archive that contains the necessary libraries:

   Linux, macOS, or Unix

   ```
   cd venv/lib/python3.7/site-packages && \
   zip -r ~/pinpoint-importer/pinpoint-importer.zip dateutil docutils jmespath \
   s3fs s3transfer fsspec six.py && cd -
   ```

   In the preceding command, replace `3.7` with the version of Python that's installed on your computer.

   Windows PowerShell

   ```
   cd .\venv\Lib\site-packages\
   Compress-Archive -Path dateutil, docutils, jmespath, s3fs, s3transfer, six.py `
   -DestinationPath ..\..\..\pinpoint-importer.zip ; cd ..\..\..
   ```

Step 4: Create the Lambda function that splits input data

The solution that's described in this tutorial uses three Lambda functions. The first Lambda function is triggered when you upload a file to a specific Amazon S3 bucket. This function reads the content of the input file, and then breaks it into smaller parts. Other functions (which you create later in this tutorial) process these incoming files concurrently. Processing the files concurrently reduces the amount of time that it takes to import all of the endpoints into Amazon Pinpoint.

Step 4.1: Create the function

To create the first Lambda function for this tutorial, you first have to upload the .zip file that you created in Step 3 (p. 26). Next, you set up the function itself.

To create the Lambda function

1. Open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
2. Choose **Create function**.
3. Choose **Author from scratch**. Under **Basic information**, do the following:
   - For **Function name**, enter **CustomerImport_ReadIncomingAndSplit**.
   - For **Runtime**, choose **Python 3.7**.
   - For **Execution role**, choose **Use an existing role**.
   - For **Existing role**, choose **ImporterS3Role**.
   - Choose **Create function**.
4. Under **Function code**, for **Code entry type**, choose **Upload a .zip file**. Under **Function package**, choose **Upload**. Choose the `pinpoint-importer.zip` file that you created in Step 3 (p. 26). After you select the file, choose **Save**.

> **Note**
> After you choose **Save**, you receive an error message stating that Lambda couldn't open the file `lambda_function.py`. Dismiss this error—you create this file in the next step.

5. In the function editor, on the **File** menu, choose **New File**. The editor creates a new file named `Untitled1`.
6. Paste the following code in the editor:

```
import os
import boto3
import s3fs
from botocore.exceptions import ClientError

input_archive_folder = "input_archive"
to_process_folder = "to_process"
file_row_limit = 50
file_delimiter = ','

# S3 bucket info
s3 = s3fs.S3FileSystem(anon=False)

def lambda_handler(event, context):
    print("Received event: \n" + str(event))
    for record in event['Records']:
        # Assign some variables that make it easier to work with the data in the
        # event record.
        bucket = record['s3']['bucket']['name']
        key = record['s3']['object']['key']
        input_file = os.path.join(bucket,key)
        archive_path = os.path.join(bucket,input_archive_folder,os.path.basename(key))
        folder =  os.path.split(key)[0]
        s3_url = os.path.join(bucket,folder)
        output_file_template = os.path.splitext(os.path.basename(key))[0] + "__part"
        output_path = os.path.join(bucket,to_process_folder)

        # Set a variable that contains the number of files that this Lambda
        # function creates after it runs.
        num_files = file_count(s3.open(input_file, 'r'), file_delimiter,
 file_row_limit)

        # Split the input file into several files, each with 50 rows.
        split(s3.open(input_file, 'r'), file_delimiter, file_row_limit,
 output_file_template, output_path, True, num_files)

        # Send the unchanged input file to an archive folder.
        archive(input_file,archive_path)

# Determine the number of files that this Lambda function will create.
def file_count(file_handler, delimiter, row_limit):
    import csv
    reader = csv.reader(file_handler, delimiter=delimiter)
    # Figure out the number of files this function will generate.
    row_count = sum(1 for row in reader) - 1
    # If there's a remainder, always round up.
    file_count = int(row_count // row_limit) + (row_count % row_limit > 0)
    return file_count

# Split the input into several smaller files.
def split(filehandler, delimiter, row_limit, output_name_template, output_path,
 keep_headers, num_files):
    import csv
    reader = csv.reader(filehandler, delimiter=delimiter)

    current_piece = 1
    current_out_path = os.path.join(
         output_path,
         output_name_template + str(current_piece) + "__of" + str(num_files) + ".csv"
    )
    current_out_writer = csv.writer(s3.open(current_out_path, 'w'),
 delimiter=delimiter)
    current_limit = row_limit
    if keep_headers:
        headers = next(reader)
        current_out_writer.writerow(headers)
    for i, row in enumerate(reader):
        if i + 1 > current_limit:
            current_piece += 1
            current_limit = row_limit * current_piece
            current_out_path = os.path.join(
               output_path,
               output_name_template + str(current_piece) + "__of" + str(num_files) +
 ".csv"
            )
            current_out_writer = csv.writer(s3.open(current_out_path, 'w'),
 delimiter=delimiter)
            if keep_headers:
                current_out_writer.writerow(headers)
        current_out_writer.writerow(row)

# Move the original input file into an archive folder.
def archive(input_file, archive_path):
    s3.copy_basic(input_file,archive_path)
    print("Moved " + input_file + " to " + archive_path)
    s3.rm(input_file)
```

7. In the function editor, on the **File** menu, choose **Save As**. Save the file as `lambda_function.py` in the root directory of the function.
8. At the top of the page, choose **Save**.

Step 4.2: Test the function

After you create the function, you should test it to make sure that it's set up correctly.

To test the Lambda function

1. In a text editor, create a new file. In the file, paste the following text:

```
Salutation,First Name,Last Name,Title,Mailing Street,Mailing City,Mailing State/
Province,Mailing Zip/Postal Code,Mailing Country,Phone,Email,Contact Record
 Type,Account Name,Account Owner,Lead Source
Mr.,Alejandro,Rosales,Operations Manager,414 Main Street,Anytown,AL,95762,United
 States,+18705550156,arosales@example.com,Customer,Example Corp.,Richard Roe,Website
Ms.,Ana Carolina,Silvia,Customer Service Representative,300 First Avenue,Any
 Town,AK,65141,United States,(727) 555-0128,asilvia@example.com,Qualified
 Lead,AnyCompany,Jane Doe,Seminar
Mrs.,Juan,Li,Auditor,717 Kings Street,Anytown,AZ,18162,United
 States,768.555.0122,jli@example.com,Unqualified Lead,Example Corp.,Richard Roe,Phone
Dr.,Arnav,Desai,Senior Analyst,782 Park Court,Anytown,AR,27084,United States,
+17685550162,adesai@example.com,Customer,Example Corp.,Richard Roe,Website
Mr.,Mateo,Jackson,Sales Representative,372 Front Street,Any Town,CA,83884,United
 States,(781) 555-0169,mjackson@example.com,Customer,AnyCompany,Jane Doe,Seminar
Mr.,Nikhil,Jayashankar,Executive Assistant,468 Fifth Avenue,Anytown,CO,75376,United
 States,384.555.0178,njayashankar@example.com,Qualified Lead,Example Corp.,Jane
 Doe,Website
Mrs.,Shirley,Rodriguez,Account Manager,287 Park Avenue,Any Town,CT,26715,United States,
+12455550188,srodriguez@example.com,Qualified Lead,Example Corp.,Richard Roe,Seminar
```

```
Ms.,Xiulan,Wang,Information Architect,107 Queens Place,Anytown,DE,70710,United States,
(213) 555-0192,xwang@example.com,Unqualified Lead,Example Corp.,Richard Roe,Phone
Miss,Saanvi,Sarkar,Director of Finance,273 Sample Boulevard,Any Town,FL,85431,United
 States,237.555.0121,ssarkar@example.com,Customer,AnyCompany,Richard Roe,Referral
Mr.,Wei,Zhang,Legal Counsel,15 Third Avenue,Any Town,GA,82387,United States,
+18065550179,wzhang@example.com,Customer,AnyCompany,Jane Doe,Website
```

Save the file as `testfile.csv`.

Note
This file contains fictitious contact records. You only use it to test the Lambda function that you create in this tutorial. Later, you can delete the segments that contain this fictitious data.
For now, don't add or remove any columns to the file. After you implement the solution that's shown in this tutorial, you can modify it to meet your needs.

2. Open the Amazon S3 console at https://console.aws.amazon.com/s3/.
3. In the list of buckets, choose the bucket that you created in Step 1 (p. 20), and then choose the `input` folder.
4. Choose **Upload**. Upload the `testfile.csv` file that you just created.
5. Open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
6. In the list of functions, choose the **CustomerImport_ReadIncomingAndSplit** function that you created earlier.
7. Choose **Test**. On the **Configure test event** window, for **Event name**, enter **`TestEvent`**. Then, in the editor, paste the following code:

```
{
  "Records": [
    {
      "s3": {
        "bucket": {
          "name": "bucket-name",
          "arn": "arn:aws:s3:::bucket-name"
        },
        "object": {
          "key": "input/testfile.csv"
        }
      }
    }
  ]
}
```

In the preceding example, replace *`bucket-name`* with the name of the Amazon S3 bucket that you created in Step 1 (p. 20). When you finish, choose **Create**.

8. Choose **Test** again. The function executes with the test event that you provided.

   If the function runs as expected, proceed to the next step.

   If the function fails to complete, do the following:

   - Make sure that you specified the correct bucket name in the IAM policy that you created in Step 2: Create IAM roles (p. 22).
   - Make sure that the Lambda test event that you created in step 7 of this section refers to the correct bucket and file name.
   - If you named the input file something other than `testfile.csv`, make sure that the file name doesn't contain any spaces.
9. Return to the Amazon S3 console. Choose the bucket that you created in the section called “Step 1: Create an Amazon S3 bucket” (p. 20).

Open the folders in the bucket and take note of the contents of each one. If all of the following statements are true, then the Lambda function worked as expected:

- The `input` folder doesn't contain any files.
- The `input_archive` folder contains the file that you uploaded in step 4 of this section.
- The `to_process` folder contains a file named `testfile__part1__of1.csv`.

Don't delete any of the newly generated files. The Lambda function that you create in the next step uses the files in the `to_process` folder.

Step 5: Create the Lambda function that processes the incoming records

The next Lambda function that you create processes the records in the incoming files that were created by the function that you created in Step 4 (p. 27). Specifically, it does the following things:

- Changes the headers in the incoming file to values that Amazon Pinpoint expects to see.
- Converts some of the contact information in the input spreadsheet into a format that Amazon Pinpoint expects. For example, the value "United States" in the `Mailing Country` column is converted to "US" in the `Location.Country` column of the output file. This is because Amazon Pinpoint expects countries to be represented in ISO-3166-1 format.
- Sends every phone number that it finds to the phone number validation service. This step ensures that all phone numbers are converted to E.164 format. It also determines the phone number type. Mobile phone numbers are created as SMS endpoints, while all other phone numbers are created as voice endpoints.
- Writes separate rows for each endpoint that it finds. For example, if one row in the input file contains a phone number and an email address, then the output file contains a separate row for each of those endpoints. However, the two records are united by a common user ID.
- Checks to see if endpoint IDs in the incoming file already exist in the Amazon Pinpoint project. If they do, the Lambda function updates the existing records rather than creating new ones.

When the function finishes processing the input files, it sends them to a folder in the `processed` directory.

Step 5.1: Create the function

The process of creating this function is similar to the process that you completed in Step 4 (p. 27) of this tutorial. First, you upload the .zip file that contains the necessary libraries. Next, you create two Python files.

To create the Lambda function

1. Open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
2. Choose **Create function**.
3. Choose **Author from scratch**. Under **Basic information**, do the following:
   - For **Function name**, enter **CustomerImport_ProcessInput**.
   - For **Runtime**, choose **Python 3.7**.
   - For **Execution role**, choose **Use an existing role**.
   - For **Existing role**, choose **ImporterPinpointRole**.

- Choose **Create function**.

4. Under **Function code**, for **Code entry type**, choose **Upload a .zip file**. Under **Function package**, choose **Upload**. Choose the `pinpoint-importer.zip` file that you created in Step 3 (p. 26). After you select the file, choose **Save**.

   Note
   After you choose **Save**, you receive an error message stating that Lambda couldn't open the file `lambda_function.py`. Dismiss this error—you create this file in the next step.
5. In the function editor, on the **File** menu, choose **New File**. The editor creates a new file named `Untitled1`.
6. Paste the following code in the editor:

```
import io
import os
import csv
import time
import uuid
import boto3
import s3fs
import input_internationalization as i18n
from datetime import datetime
from botocore.exceptions import ClientError

# You might need to change some things to fit your specific needs.

# If incoming data doesn't specify a country, you have to pass a default value.
# Specify a default country code in ISO 3166-1 alpha-2 format.
defaultCountry = "US"

# The column header names that are applied to the output file. You might need to
# change the order of the items in this list to suit the data that's in the file
# that you want to import. Column numbers are included as comments here to make it
# easier to align the columns.
# If you add columns here, you also need to add them in the process_incoming_file
# function below. Specifically, you need to add them to the lists that the
# Filewriter object uses to write the processed files. See the sections that
# begin at lines 137 and 175 below.
header =    [                                           #Col num in input
                'ChannelType',                          #not in input
                'Address',                              #9 (phone), 10 (email)
                'Id',                                   #9 (phone), 10 (email)
                'User.UserAttributes.City',             #5
                'User.UserAttributes.Region',           #6
                'User.UserAttributes.PostalCode',       #7
                'Location.Country',                     #8
                'User.UserAttributes.Salutation',       #0
                'User.UserAttributes.FirstName',        #1
                'User.UserAttributes.LastName',         #2
                'User.UserAttributes.Title',            #3
                'User.UserAttributes.StreetAddress',    #4
                'User.UserAttributes.ContactRecordType',    #11
                'User.UserAttributes.AccountName',      #12
                'User.UserAttributes.AccountOwner',     #13
                'User.UserAttributes.LeadSource',       #14
                'User.UserId'                           #not in input
            ]

# You probably don't need to change any variables below this point.
AWS_REGION = os.environ['region']
projectId = os.environ['projectId']
processed_folder = "processed"
startTime = datetime.now()
s3 = s3fs.S3FileSystem(anon=False)
```

```
def lambda_handler(event, context):
    print("Received event: " + str(event))

    for record in event['Records']:
        # Create some variables that make it easier to work with the data in the
        # event record.
        bucket = record['s3']['bucket']['name']
        key = record['s3']['object']['key']
        input_file = os.path.join(bucket,key)
        output_file_name = os.path.splitext(os.path.basename(input_file))[0] +
 "_processed.csv"
        processed_subfolder = os.path.basename(input_file).split("__part",1)[0]
        output_fullpath =
 os.path.join(bucket,processed_folder,processed_subfolder,output_file_name)
        # Start the function that processes the incoming data.
        process_incoming_file(input_file, output_fullpath)

# Check the current project to see if an endpoint ID already exists.
def check_endpoint_exists(endpointId):
    client = boto3.client('pinpoint',region_name=AWS_REGION)

    try:
        response = client.get_endpoint(
            ApplicationId=projectId,
            EndpointId=endpointId
        )
    except ClientError as e:
        endpointInfo = [ False, "" ]
    else:
        userId = response['EndpointResponse']['User']['UserId']
        endpointInfo = [ True, userId ]

    return endpointInfo

# Change the column names, validate and reformat some of the input, and then
# write to output files.
def process_incoming_file(input_file, output_file):
    # Counters for tracking the number of records and endpoints processed.
    line_count = 0
    create_count = 0
    update_count = 0

    folder = os.path.split(output_file)[0]

    with s3.open(output_file, 'w', newline='', encoding='utf-8-sig') as outFile:
        fileWriter = csv.writer(outFile)
        with s3.open(input_file, 'r', newline='', encoding='utf-8-sig') as inFile:
            fileReader = csv.reader(inFile)

            for row in fileReader:
                # Sleep to prevent throttling errors.
                time.sleep(.025)
                # Write the header row.
                if (line_count == 0):
                    fileWriter.writerow(header)
                    line_count += 1
                # Write the rest of the data.
                else:
                    # Generate a new UUID.
                    userId = str(uuid.uuid4())

                    # Varibles that make things easier to read.
                    inputEmail = row[10]
                    inputPhone = row[9]
                    inputCountry = row[8]
                    # If a country is included in the incoming record, create a
                    # variable that contains the ISO 3166-1 alpha-2 country code.
                    if inputCountry:
                        country = i18n.get_country_code(inputCountry, defaultCountry)
                    # If no country code is provided, create a variable that contains a
                    # default country code. Change this if you want to use a different
                    # default value.
                    elif not inputCountry:
                        country = defaultCountry

                    if inputEmail:

                        emailEndpointInfo = check_endpoint_exists(inputEmail)

                        if emailEndpointInfo[0]:
                            userId = emailEndpointInfo[1]
                            update_count += 1
                        else:
                            create_count += 1

                        fileWriter.writerow([
                                                "EMAIL",
                                                row[10],
                                                row[10],
                                                row[5],     #City
                                                row[6],     #Region
                                                row[7],     #Postal code
                                                country,    #Country
                                                row[0],     #Salutation
                                                row[1],     #First name
                                                row[2],     #Last name
                                                row[3],     #Title
                                                row[4],     #Street address
                                                row[11],    #Contact record type
                                                row[12],    #Account name
                                                row[13],    #Account owner
                                                row[14],    #Lead source
                                                userId
                                            ])

                    if inputPhone:
                        phoneInfo = i18n.check_phone_number(country, inputPhone,
AWS_REGION)
                        cleansedPhone = phoneInfo[0]
                        phoneType = phoneInfo[1]

                        if (phoneType == "MOBILE") or (phoneType == "PREPAID"):
                            phoneType = "SMS"
                        else:
                            phoneType = "VOICE"

                        phoneEndpointInfo = check_endpoint_exists(cleansedPhone)

                        if phoneEndpointInfo[0]:
                            userId = phoneEndpointInfo[1]
                            update_count += 1
                        else:
                            create_count += 1

                        fileWriter.writerow([
                                                phoneType,
                                                cleansedPhone,
                                                cleansedPhone,
                                                row[5],     #City
                                                row[6],     #Region
                                                row[7],     #Postal code
                                                country,    #Country
                                                row[0],     #Salutation
                                                row[1],     #First name
                                                row[2],     #Last name
                                                row[3],     #Title
                                                row[4],     #Street address
                                                row[11],    #Contact record type
                                                row[12],    #Account name
                                                row[13],    #Account owner
                                                row[14],    #Lead source
                                                userId
                                            ])
                    line_count += 1

    # Calculate the amount of time the script ran.
    duration = datetime.now() - startTime

    # Print the number of records processed. Subtract 1 to account for the header.
    print("Processed " + str(line_count - 1) + " records in " + str(duration)
            + ". ", end="")

    # Print the numbers of endpoints created and updated.
    print("Found " + str(create_count) + " new endpoints and "
            + str(update_count) + " existing endpoints.")

    s3.rm(input_file)
```

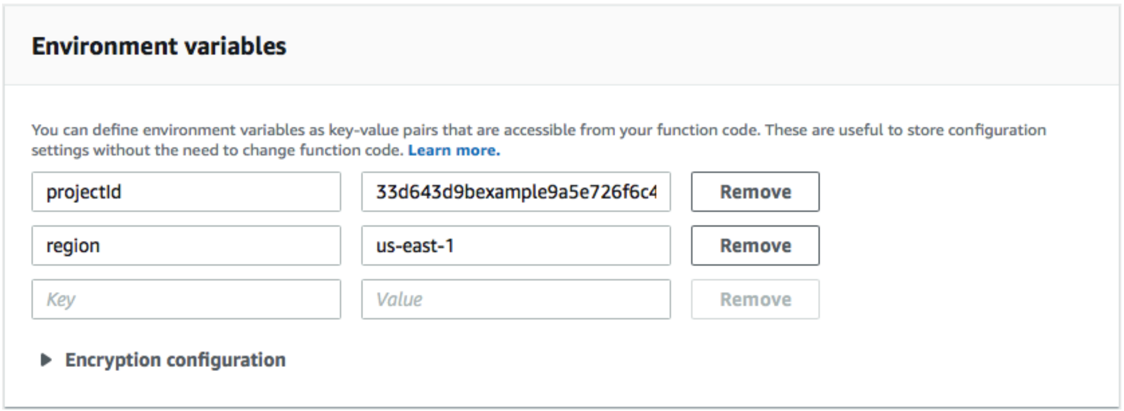
7. In the function editor, on the **File** menu, choose **Save As**. Save the file as `lambda_function.py` in the root directory of the function.
8. In the function editor, on the **File** menu, choose **New File**. The editor creates a new file named `Untitled1`.
9. Paste the following code in the editor:

```
import re
import boto3
from botocore.exceptions import ClientError

def get_country_code(country, default):
    # The Location.Country attribute expects an ISO 3166-1 Alpha 2 formatted
    # country name. This function attempts to identify several possible
    # variations of each country name and convert them to the required format.
    # This function exists so that the program doesn't need to rely on non-
    # standard libraries. Update this section to suit the data in your existing
    # content management system.
    # This section includes common aliases for the 10 most populous countries
    # in the world, and some additional countries where Amazon Pinpoint is often used
 to
    # send SMS messages. If necessary, expand this section to include
    # additional countries, or additional aliases for the countries that are
    # already listed.
    country = country.strip().lower()

    bangladeshAliases = [ "bd", "bgd", "bangladesh", "########" ]
    brazilAliases     = [ "br", "bra", "brazil", "brasil",
                          "república federativa do brasil" ]
    canadaAliases     = [ "ca", "can", "canada" ]
    chinaAliases      = [ "cn", "chn", "prc", "proc", "china",
                          "people's republic of china", "#######", "##" ]
    indiaAliases      = [ "in", "ind", "india", "republic of india" ]
    indonesiaAliases  = [ "id", "idn", "indonesia", "republic of indonesia",
                          "republik indonesia" ]
    irelandAliases    = [ "ie", "irl", "ireland", "éire" ]
    japanAliases      = [ "jp", "jpn", "japan", "##" ]
    mexicoAliases     = [ "mx", "mex", "mexico", "méxico",
                          "estados unidos mexicanos" ]
    nigeriaAliases    = [ "ng", "nga", "nigeria" ]
    pakistanAliases   = [ "pk", "pak", "pakistan", "#######" ]
    russiaAliases     = [ "ru", "rus", "russian federation", "#######",
                          "########### ##########" ]
    ukAliases         = [ "uk", "gb", "gbr", "united kingdom", "britain",
                          "england", "wales", "scotland", "northern ireland" ]
    usAliases         = [ "us", "usa", "united states",
                          "united states of america" ]

    if country in bangladeshAliases:
        countryISO3166 = "BD"
    elif country in brazilAliases:
        countryISO3166 = "BR"
    elif country in canadaAliases:
        countryISO3166 = "CA"
    elif country in chinaAliases:
        countryISO3166 = "CN"
    elif country in indiaAliases:
        countryISO3166 = "IN"
    elif country in indonesiaAliases:
        countryISO3166 = "ID"
    elif country in irelandAliases:
        countryISO3166 = "IE"
    elif country in japanAliases:
        countryISO3166 = "JP"
    elif country in mexicoAliases:
        countryISO3166 = "MX"
    elif country in nigeriaAliases:
        countryISO3166 = "NG"
    elif country in pakistanAliases:
        countryISO3166 = "PK"
    elif country in russiaAliases:
        countryISO3166 = "RU"
    elif country in ukAliases:
        countryISO3166 = "GB"
    elif country in usAliases:
        countryISO3166 = "US"
    else:
        countryISO3166 = default

    return countryISO3166

def check_phone_number(country, phone, region):

    client = boto3.client('pinpoint',region_name=region)

    # Phone number validation can generate an E.164-compliant phone number as
    # long as you provide it with the correct country code. This function looks
    # for the appropriate country code at the beginning of the phone number. If
    # the country code isn't present, it adds it to the beginning of the phone
    # number that was provided to the function, and then sends it to the phone
    # number validation service. The phone number validation service performs
    # additional cleansing of the phone number, removing things like
    # unnecessary leading digits. It also provides metadata, such as the phone
    # number type (mobile, landline, etc.).
    # Add more countries and regions to this function if necessary.
    phone =  re.sub("[^0-9]", "", phone)

    if (country == 'BD') and not phone.startswith('880'):
        phone = "+880" + phone
    elif (country == 'BR') and not phone.startswith('55'):
        phone = "+55" + phone
    elif (country == 'CA' or country == 'US') and not phone.startswith('1'):
        # US and Canada (country code 1) area codes and phone numbers can't use
        # 1 as their first digit, so it's fine to search for a 1 at the
        # beginning of the phone number to determine whether or not the number
        # contains a country code.
        phone = "+1" + phone
    elif (country == 'CN') and not phone.startswith('86'):
        phone = "+86" + phone
    elif (country == 'IN') and not phone.startswith('91'):
        phone = "+91" + phone
    elif (country == 'ID') and not phone.startswith('62'):
        phone = "+62" + phone
    elif (country == 'IE') and not phone.startswith('353'):
        phone = "+353" + phone
    elif (country == 'JP') and not phone.startswith('81'):
        phone = "+81" + phone
    elif (country == 'MX') and not phone.startswith('52'):
        phone = "+52" + phone
    elif (country == 'NG') and not phone.startswith('234'):
        phone = "+234" + phone
    elif (country == 'PK') and not phone.startswith('92'):
        phone = "+92" + phone
    elif (country == 'RU') and not phone.startswith('7'):
        # No area codes in Russia begin with 7. However, Kazakhstan also uses
        # country code 7, and Kazakh area codes can begin with 7. If your
        # contact database contains Kazakh phone numbers, you might have to
        # use some additional logic to identify them.
        phone = "+7" + phone
    elif (country == 'GB') and not phone.startswith('44'):
        phone = "+44" + phone

    try:
        response = client.phone_number_validate(
            NumberValidateRequest={
                'IsoCountryCode': country,
                'PhoneNumber': phone
            }
        )
    except ClientError as e:
        print(e.response)
    else:
        returnValues = [
            response['NumberValidateResponse']['CleansedPhoneNumberE164'],
 response['NumberValidateResponse']['PhoneType']
        ]
        return returnValues
```

10. In the function editor, on the **File** menu, choose **Save As**. Save the file as `input_internationalization.py` in the root directory of the function.
11. Under **Environment variables**, do the following:

    - In the first row, create a variable with a key of **`projectId`**. Next, set the value to the unique ID of the project that you specified in the IAM policy in Step 2.2 (p. 24).
    - In the second row, create a variable with a key of **`region`**. Next, set the value to the AWS Region that you specified in the IAM policy in Step 2.2 (p. 24).

    When you finish, the **Environment Variables** section should resemble the example shown in the following image.



Environment variables

You can define environment variables as key-value pairs that are accessible from your function code. These are useful to store configuration settings without the need to change function code. **Learn more.**

projectId	33d643d9bexample9a5e726f6c4	Remove
region	us-east-1	Remove
Key	Value	Remove

▶ Encryption configuration

12. Under **Basic settings**, set the **Timeout** value to 6 minutes.

    Note
    Each call to the phone number validation service typically takes about 0.5 seconds to complete, but it can occasionally take longer. If you don't increase the **Timeout** value, the function might time out before it finishes processing the incoming records. This setting gives the Lambda function plenty of time to finish running.
13. Under **Concurrency**, choose **Reserve concurrency**, and then set the value to `20`.

    Note
    A higher concurrency value might cause the files to be processed faster. However, if the concurrency value is too high, you start to generate a number of PUT events that exceeds Amazon S3 quotas. As a result, the import process doesn't capture all of the data in your input spreadsheet. This is because many of the requests that are generated by this function end in failure.
14. At the top of the page, choose **Save**.

Step 5.2: Test the function

After you create the function, you should test it to make sure that it's set up correctly.

To test the Lambda function

1. Choose **Test**. On the **Configure test event** window, for **Event name**, enter **`TestEvent`**. Then, in the editor, paste the following code:

```
{
  "Records": [
    {
      "s3": {
        "bucket": {
          "name": "bucket-name",
          "arn": "arn:aws:s3:::bucket-name"
        },
        "object": {
          "key": "to_process/testfile__part1__of1.csv"
        }
      }
    }
  ]
}
```

   In the preceding example, replace *`bucket-name`* with the name of the Amazon S3 bucket that you created in Step 1 (p. 22). When you finish, choose **Create**.

2. Choose **Test** again. The function executes with the test event that you provided.

   If the function runs as expected, proceed to the next step.

   If the function fails to complete, do the following:

   - Make sure that you specified the correct bucket name in the IAM policy that you created in Step 1: Create an Amazon S3 bucket (p. 20).
   - Make sure that the Lambda test event that you created in step 1 of this section refers to the correct bucket and file name.
3. Open the Amazon S3 console at https://console.aws.amazon.com/s3/.

   Choose the bucket that you created in the section called "Step 1: Create an Amazon S3 bucket" (p. 20).

   Open the folders in the bucket and take note of the contents of each one. If all of the following statements are true, then the Lambda function worked as expected:

   - The `to_process` folder doesn't contain any files.
   - The processed folder contains a subfolder named `testfile`. Within the `testfile` folder, there is a file named `testfile__part1__of1_processed.csv`.

   Don't delete any of the newly generated files. The Lambda function that you create in the next step uses the files in the `to_process` folder.

Step 6: Create the Lambda function that imports records into Amazon Pinpoint

The final Lambda function that you create for this tutorial creates an import job in Amazon Pinpoint. The import job imports the files in the `processed` folder of your Amazon S3 bucket. Although the `processed` folder might contain several files, the import job creates a single segment that contains all of the endpoints within those files.

This function uses a test to make sure that all of the files that were generated by the previous Lambda functions are present before starting the import job. It also checks to see if you've created any import jobs that refer to the same folder, in order to prevent the creation of duplicate segments.

Step 6.1: Create the function

The process of creating the final Lambda function is much simpler than the process for the previous two functions. This is because you don't need to import any external libraries.

To create the function

1. Open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
2. Choose **Create function**.
3. Choose **Author from scratch**. Under **Basic information**, do the following:

   - For **Function name**, enter **CustomerImport_CreateJob**.
   - For **Runtime**, choose **Python 3.7**.
   - For **Execution role**, choose **Use an existing role**.
   - For **Existing role**, choose **ImporterPinpointRole**.
   - Choose **Create function**.

4. Erase the example in the code editor, and then paste the following code into the editor:

```
import os
import time
import boto3
from botocore.exceptions import ClientError

AWS_REGION = os.environ['region']
projectId = os.environ['projectId']
importRoleArn = os.environ['importRoleArn']

def lambda_handler(event, context):
    print("Received event: " + str(event))
    for record in event['Records']:
        # Assign some variables to make it easier to work with the data in the
        # event recordi
        bucket = record['s3']['bucket']['name']
        key = record['s3']['object']['key']
        folder =  os.path.split(key)[0]
        folder_path = os.path.join(bucket, folder)
        full_path = os.path.join(bucket, key)
        s3_url = "s3://" + folder_path

        # Check to see if all file parts have been processed.
        if all_files_processed(bucket, folder, full_path):
            # If you haven't recently run an import job that uses a file stored in
            # the specified S3 bucket, then create a new import job. This prevents
            # the creation of duplicate segments.
            if not (check_import_jobs(bucket, folder, s3_url)):
                create_import_job(s3_url)
            else:
                print("Import job found with URL s3://"
                        + os.path.join(bucket,folder) + ". Aborting.")
        else:
            print("Parts haven't finished processing yet.")

# Determine if all of the file parts have been processed.
def all_files_processed(bucket, folder, full_path):
    # Use the "__ofN" part of the file name to determine how many files there
    # should be.
    number_of_parts = int((full_path.split("__of")[1]).split("_processed")[0])

    # Figure out how many keys contain the prefix for the current batch of
    # folders (basically, how many files are in the appropriate "folder").
    client = boto3.client('s3')
    objs = client.list_objects_v2(Bucket=bucket,Prefix=folder)
    file_count = objs['KeyCount']

    ready_for_import = False

    if file_count == number_of_parts:
        ready_for_import = True

    return ready_for_import

# Check Amazon Pinpoint to see if any import jobs have been created by using
# the same S3 folder.
def check_import_jobs(bucket, folder, s3_url):
    url_list = []

    # Retrieve a list of import jobs for the current project ID.
    client = boto3.client('pinpoint')
    try:
        client_response = client.get_import_jobs(
            ApplicationId=projectId
        )
    except ClientError as e:
        print(e.response['Error']['Message'])
    else:
        segment_response = client_response['ImportJobsResponse']['Item']
        # Parse responses. Add all S3Url values to a list.
        for item in segment_response:
            s3_url_existing = item['Definition']['S3Url']
            url_list.append(s3_url_existing)

    # Search for the current S3 URL in the list.
    if s3_url in url_list:
        found = True
    else:
        found = False

    return found

# Create the import job in Amazon Pinpoint.
def create_import_job(s3_url):
    client = boto3.client('pinpoint')

    segment_name = s3_url.split('/')[4]

    try:
        response = client.create_import_job(
            ApplicationId=projectId,
            ImportJobRequest={
                'DefineSegment': True,
                'Format': 'CSV',
                'RegisterEndpoints': True,
                'RoleArn': importRoleArn,
                'S3Url': s3_url,
                'SegmentName': segment_name
            }
        )
    except ClientError as e:
        print(e.response['Error']['Message'])
    else:
        print("Import job " + response['ImportJobResponse']['Id'] + " "
                + response['ImportJobResponse']['JobStatus'] + ".")

        print("Segment ID: "
                + response['ImportJobResponse']['Definition']['SegmentId'])

        print("Application ID: " + projectId)
```

5. Under **Environment variables**, do the following:

   - In the first row, create a variable with a key of **`projectId`**. Next, set the value to the unique ID of the project that you specified in the IAM policy in Step 2.2 (p. 24).
   - In the second row, create a variable with a key of **`region`**. Next, set the value to the AWS Region that you specified in the IAM policy in Step 2.2 (p. 24).
   - In the third row, create a variable with a key of **`importRoleArn`**. Next, set the value to the Amazon Resource Name (ARN) of the IAM role that you created in Step 2.2 (p. 23).

6. Under **Basic settings**, set the **Timeout** value to 7 seconds.

   Note
   You might be able to use the default timeout value of 3 seconds. However, it might require slightly more time for larger jobs to be created.

7. At the top of the page, choose **Save**.

Step 6.2: Test the function

After you create the function, you should test it to make sure that it's set up correctly.

To test the Lambda function

1. Choose **Test**. On the **Configure test event** page, for **Event name**, enter **`TestEvent`**. Then, in the editor, paste the following code:

```
{
  "Records": [
    {
      "s3": {
        "bucket": {
          "name": "bucket-name",
          "arn": "arn:aws:s3:::bucket-name"
        },
        "object": {
          "key": "processed/testfile/testfile__part1_processed.csv"
        }
      }
    }
  ]
}
```

   In the preceding example, replace *`bucket-name`* with the name of the Amazon S3 bucket that you created in Step 1 (p. 22). When you finish, choose **Create**.
2. Choose **Test** again. The function executes with the test event that you provided.
3. Open the Amazon S3 console at https://console.aws.amazon.com/s3/.

   Choose the bucket that you created in the section called "Step 1: Create an Amazon S3 bucket" (p. 20).

   Open the folders in the bucket and take note of the contents of each one. If all of the following statements are true, then the Lambda function worked as expected:

   - The `to_process` folder doesn't contain any files.
   - The processed folder contains a subfolder named `testfile`. Within the `testfile` folder, there is a file named `testfile__part1__of1_processed.csv`.

   Don't delete any of the newly generated files. The Lambda function that you create in the next step uses the files in the `to_process` folder.

Step 7: Set up Amazon S3 events

In this section, you set up your Amazon S3 bucket so that it triggers your Lambda functions when you add files to the folders in the bucket. You also test the entire function to make sure that the triggers work as expected.

By using event triggers in Amazon S3, you make the process of executing the Lambda functions automatic. When you upload a file to the `input` folder, Amazon S3 automatically sends a notification to the `CustomerImport_ReadIncomingAndSplit` function. When that function runs, it sends files to the `to_process` folder. When files are added to the `to_process` folder, Amazon S3 triggers the `CustomerImport_ProcessInput` function. When that function runs, it adds files to the `processed` folder, which triggers the `CustomerImport_CreateJob` function.

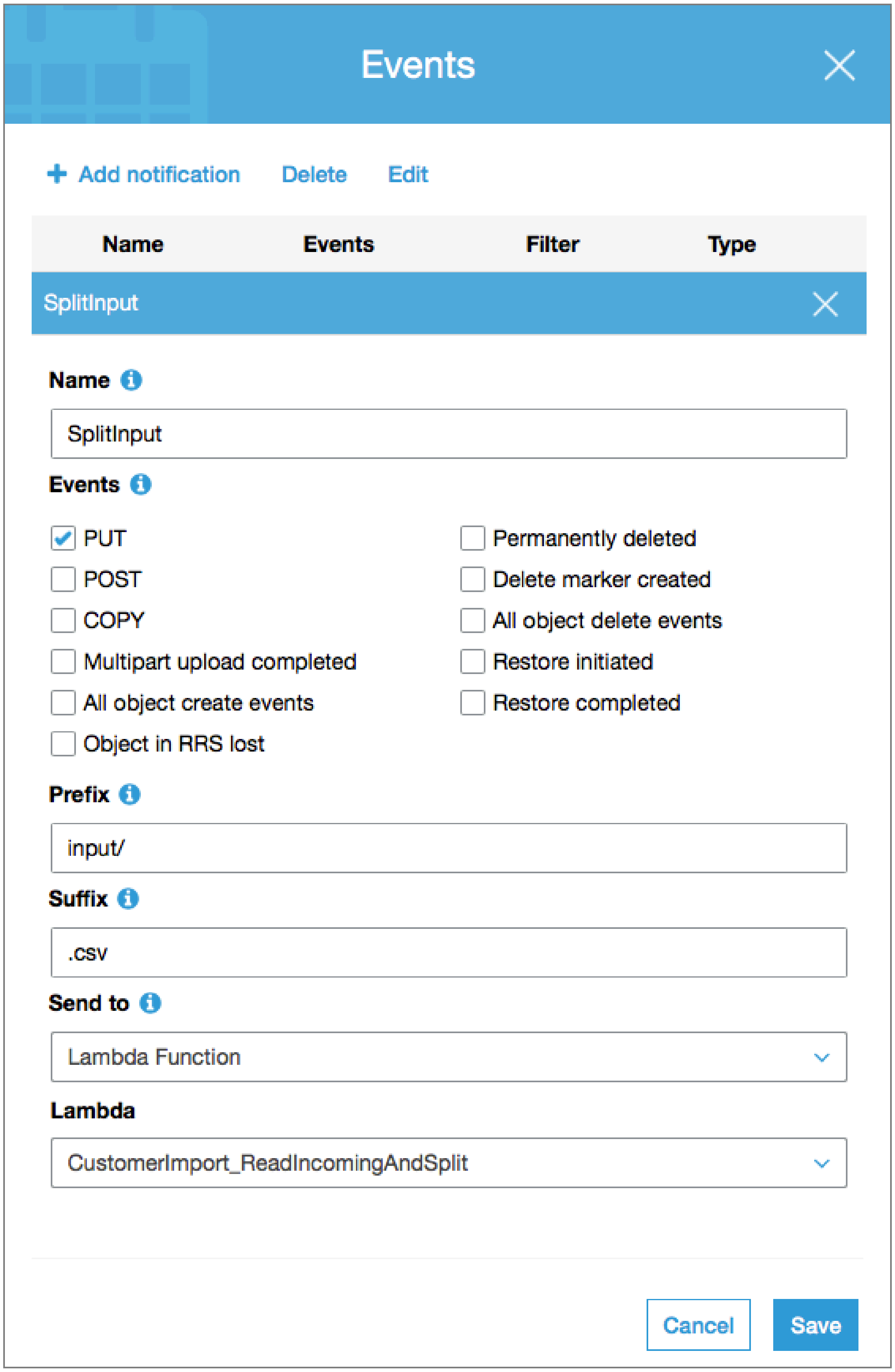
Step 7.1: Set up event notifications

In this section, you configure three event notifications for your Amazon S3 bucket. These notifications automatically trigger your Lambda functions when files are added to specific folders in your bucket.

To configure the event triggers

1. Open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. In the list of buckets, choose the bucket that you created in Step 1 (p. 20).
3. On the **Properties** tab, choose **Events**. Next, do the following:
   - Choose **Add notification**, as shown in the following image.
   - For **Name**, enter **`SplitInput`**.
   - For **Events**, choose **PUT**.
   - For **Prefix**, enter **`input/`**.
   - For **Suffix**, enter **`.csv`**.
   - For **Send to**, choose **Lambda Function**.
   - For **Lambda**, choose **CustomerImport_ReadIncomingAndSplit**.

   When you finish, the **Events** section resembles the example that's shown in the following image.



Name	Events	Filter	Type

SplitInput

Name

SplitInput

Events

[✓] PUT	[ ] Permanently deleted
[ ] POST	[ ] Delete marker created
[ ] COPY	[ ] All object delete events
[ ] Multipart upload completed	[ ] Restore initiated
[ ] All object create events	[ ] Restore completed
[ ] Object in RRS lost	

Prefix

input/

Suffix

.csv

Send to

Lambda Function

Lambda

CustomerImport_ReadIncomingAndSplit

- Choose **Save**.

4. Choose **Events**, and then choose **Add notification** again. Do the following:

   - Choose **Add notification**, as shown in the following image.
   - For **Name**, enter **`ProcessInput`**.
   - For **Events**, choose **PUT**.
   - For **Prefix**, enter **`to_process/`**.
   - For **Suffix**, enter **`.csv`**.
   - For **Send to**, choose **Lambda Function**.
   - For **Lambda**, choose **CustomerImport_ProcessInput**.
   - Choose **Save**.

5. Choose **Events**, and then choose **Add notification** again. Do the following:

   - Choose **Add notification**, as shown in the following image.
   - For **Name**, enter **`CreateImportJob`**.
   - For **Events**, choose **PUT**.
   - For **Prefix**, enter **`processed/`**.
   - For **Suffix**, enter **`.csv`**.
   - For **Send to**, choose **Lambda Function**.
   - For **Lambda**, choose **CustomerImport_CreateJob**.
   - Choose **Save**.

Step 7.2: Test the triggers

The last step in setting up the solution that's discussed in this tutorial is to upload a file to the `input` folder in your Amazon S3 bucket. Uploading a file triggers the Lambda function that you created in Step 4 (p. 27). When this function finishes running, it creates files in the other folders that you configured event notifications for in the preceding section. After a few minutes, the entire sequence of Lambda functions finishes running, and your Amazon Pinpoint project contains a new segment.

To test the event triggers

1. On your computer, locate the `testfile.csv` file that you created in Step 4.2 (p. 29). Change the name of the file to `testfile1.csv`.
2. Open the Amazon S3 console at https://console.aws.amazon.com/s3/.
3. In the list of buckets, choose the bucket that you created in Step 1 (p. 20), and then choose the `input` folder.
4. Upload `testfile1.csv` to the `input` folder. After the file finishes uploading, wait for several minutes.
5. Open the Amazon Pinpoint console at https://console.aws.amazon.com/pinpoint/.
6. In the list of projects, choose the project that you're importing segments into.
7. In the navigation pane, choose **Segments**. On the **Segments** tab, look for a segment named **testfile1**. If the segment isn't listed, check the **Scheduled imports** tab to see if the import job is still in progress.

   If you don't see a new segment on either tab, wait a few minutes longer, and then refresh the **Segments** page. If you still don't see the segment, do the following:

   - Make sure that the Amazon S3 notification events are configured exactly as described in the preceding section.

- Check the logs for all three of the Lambda functions in CloudWatch Logs. If any of the functions ended in error, fix the issues that caused the error.

Next steps

By completing this tutorial, you've done the following:

- Created an Amazon S3 bucket to contain the files that you import into Amazon Pinpoint.
- Created several IAM roles and policies that follow the principle of least privilege.
- Created Lambda functions that split your input file into smaller parts, process those files, and then use them to create an import job in Amazon Pinpoint.
- Set up your Amazon S3 bucket to initiate Lambda functions when it detects files in certain folders.

This section discusses a few ways that you can customize this solution to fit your unique use case.

Perform cleanup tasks

After you complete this tutorial, you can optionally delete the `input_archive`, `to_process`, and `processed` folders entirely. These folders the functions in this tutorial automatically create these folders again when they're needed.

This solution doesn't automatically delete the files in the `processed` folder. If an import job fails, you can try to import the files again by creating a new import job using the console or the API.

Over time, this folder might accumulate a large number of files that you don't need anymore. You can create a script that periodically removes old content from this folder. If you do, you should include logic that checks to see if there are any ongoing import jobs before deleting any files.

You can also optionally delete the segments that were created when you ran this tutorial. Alternatively, you can delete all of the example endpoints, as well as the segment that they belong to, by deleting the entire project that you imported the endpoints into.

Modify the internationalization function to suit your customer database

One of the Lambda functions that you create in Step 5 performs some simple tests to normalize the country and phone number data that it processes. This function includes tests for the most populous countries in the world, as well as some other countries that Amazon Pinpoint customers commonly send messages to. You can expand these tests to include additional countries and regions.

Modify the input processing function to suit your external systems

The other Lambda function that you create in Step 5 creates the column names that Amazon Pinpoint expects to see. It maps them to the columns that are in your input spreadsheet. You can change this mapping by making some small changes to the function. The comments in the included code tell you specifically what you need to change.

Automatically synchronize data with external systems

If you regularly use data from external systems to send campaigns in Amazon Pinpoint, you might be able to set up the external system to regularly export data to the input folder in your Amazon S3 bucket. If you do, make sure that each file that you export has a unique name. If you don't supply unique names, the Lambda function that creates import jobs fails. This is because the function has some simple logic to prevent the creation of duplicate import jobs.

Tutorial: Setting up an email preference management system

In many jurisdictions around the world, email senders are required to include a mechanism for opting out of email communications in each email that they send.

A common way to allow customers to specify their preferences is to host a page that customers can use to choose the specific types of messages that they want to receive. Typically, customers can also use this same page to completely opt out of all email communications that you send.

This tutorial shows you how to set up a web form that you can use to capture your customers' email preferences. The web form sends this information to Amazon Pinpoint. Amazon Pinpoint then creates or modifies attributes in the customer's endpoint record that indicate the topics they want to receive information about. You can use these attributes when you create segments in Amazon Pinpoint.

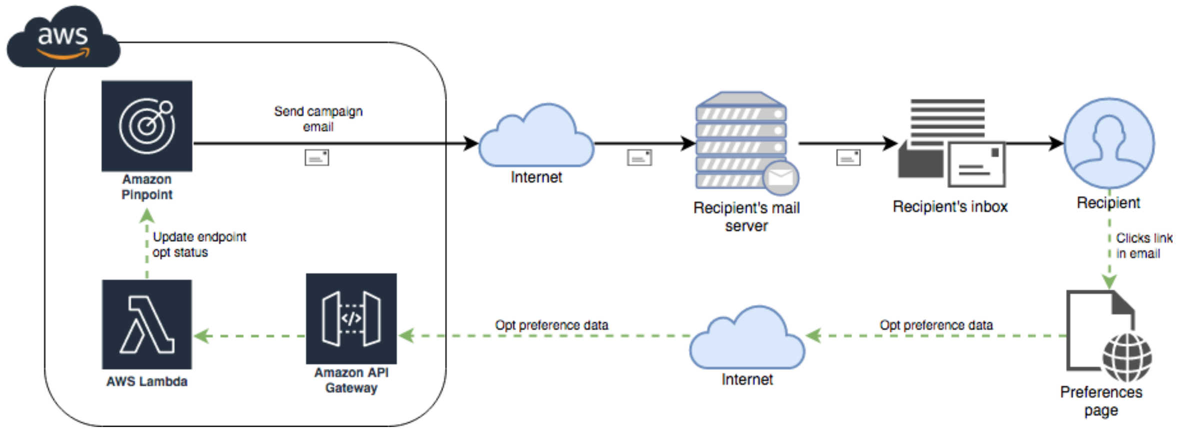
The web form also includes an option that customers can choose to opt themselves out of all email communications. When customers choose this option, they're automatically opted out of all topics. Additionally, the solution in this tutorial modifies their endpoint records so that they don't appear in any future segments in your Amazon Pinpoint project.

Architecture

When you use the solution that's described in this tutorial, you send a campaign email to your customers. This email contains a link to a preference management page. The link contains several attribute tags that identify each recipient.

When customers receive the email from you, they can choose the link. When they do, they're taken to a web form that they can use to opt into or out of various topics. The form sends this data to an API that's hosted by API Gateway. This API triggers a Lambda function, which makes changes to the customer's endpoint record.

The following diagram shows the flow of information in this solution.



About this solution

This section contains information about the solution that you're building in this tutorial.

Intended Audience

This tutorial is intended for developers and system implementers. You don't have to be familiar with Amazon Pinpoint to complete the steps in this tutorial. You should be comfortable managing IAM policies, creating Lambda functions in Node.js, and deploying web content. However, you don't need to write any code—this tutorial includes complete example code that you can implement without having to make any changes.

Features Used

This tutorial includes usage examples for the following Amazon Pinpoint features:

- Creating dynamic segments
- Sending email campaigns that contain personalized content
- Interacting with the Amazon Pinpoint API by using AWS Lambda

Time Required

It should take about 30–45 minutes to complete this tutorial. After you implement this solution, there are additional steps that you can take to refine the solution to suit your unique use case.

Regional Restrictions

There are no regional restrictions associated with using this solution. However, you should make sure that the email campaigns that you send include all of the information that's required in each recipient's jurisdiction.

Resource Usage Costs

There's no charge for creating an AWS account. However, by implementing this solution, you might incur some or all of the costs that are listed in the following table.

Description	Cost (US dollars)
Message sending costs	You pay $0.0001 for each email that you send through Amazon Pinpoint.
Monthly targeted audience	You pay $0 for the first 5,000 endpoints that you target in Amazon Pinpoint each month. After that, you pay $0.0012 per endpoint that you target.
Lambda usage	The Lambda function in this tutorial uses about 80–90 MB of memory and about 100–300 milliseconds of compute time each time it's executed. Your usage of AWS Lambda in implementing this solution might be included in the Free Tier. For more information, see AWS Lambda pricing.
API Gateway usage	You pay $0.0000035–$0.0000037 per API request, depending on which AWS Region you use. For more information, see Amazon API Gateway pricing.
Web hosting costs	The price that you pay varies depending on your web hosting provider.

Next: Prerequisites (p. 49)

Prerequisites

Before you begin this tutorial, you have to complete the following prerequisites:

- You have to have an AWS account. To create an AWS account, go to https://console.aws.amazon.com/ and choose **Create a new AWS account**.
- The account that you use to sign in to the AWSManagement Console has to be able to perform the following tasks:
  - Create new IAM policies and roles
  - Create new Amazon Pinpoint projects
  - Create new Lambda functions
  - Create new APIs in API Gateway
- You have to have a method of hosting webpages, and you should know how to publish webpages. Although you can use AWS services to host your webpages, you aren't required to.

  Tip
  To learn more about hosting webpages by using AWS services, see Host a static webpage.

Next: Set Up Amazon Pinpoint (p. 49)

Step 1: Set Up Amazon Pinpoint

The first step in implementing this solution is to set up Amazon Pinpoint. In this section, you do the following:

- Create an Amazon Pinpoint project
- Enable the email channel and verify an identity
- Set up endpoint attributes

Before you begin this tutorial, you should review the prerequisites (p. 49).

Step 1.1: Create an Amazon Pinpoint Project

To get started, you need to create an Amazon Pinpoint project. In Amazon Pinpoint, a project consists of segments, campaigns, configurations, and data that are united by a common purpose. For example, you could use a project to contain all of the content that's related to a particular app, or to a specific brand or marketing initiative. When you add customer information to Amazon Pinpoint, that information is associated with a project.

The steps involved in creating a new project differ depending on whether you've created a project in Amazon Pinpoint previously.

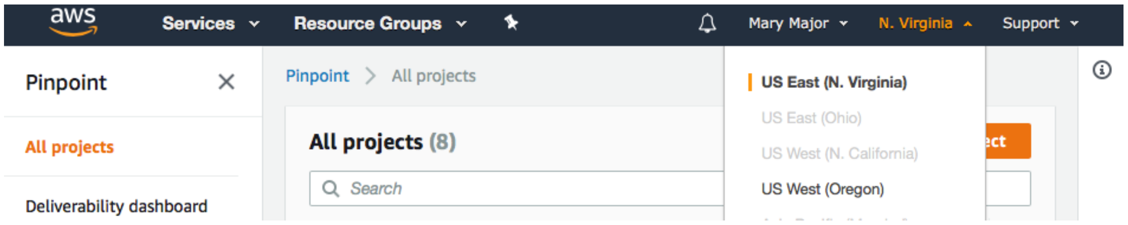
Creating a Project (New Amazon Pinpoint Users)

These steps describe the process of creating a new Amazon Pinpoint project if you've never created a project in the current AWS Region.

To create a project

1. Sign in to the AWSManagement Console and open the Amazon Pinpoint console at https://console.aws.amazon.com/pinpoint/.

2. Use the Region selector to choose the AWS Region that you want to use, as shown in the following image. If you're unsure, choose the Region that's located closest to you.



3. Under **Get started**, for **Name**, enter a name for the campaign, and then choose **Create project**.
4. On the **Configure features** page, choose **Skip this step**.
5. In the navigation pane, choose **All projects**.
6. On the **All projects** page, next to the project you just created, copy the value that's shown in the **Project ID** column.

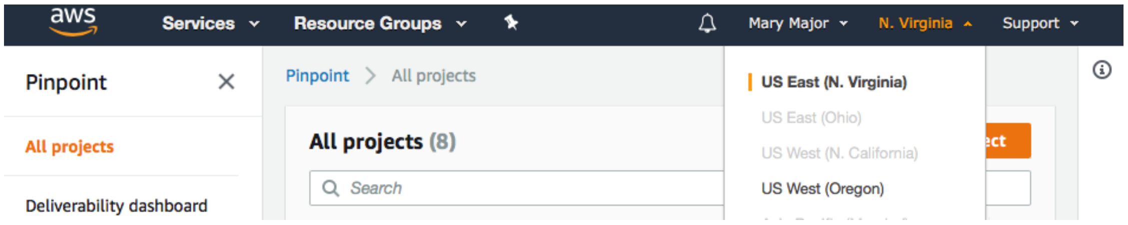
   Tip
   You need to use this ID in a few different places in this tutorial. Keep the project ID in a convenient place so that you can copy it later.

Creating a Project (Existing Amazon Pinpoint Users)

These steps describe the process of creating a new Amazon Pinpoint project if you've already created projects in the current AWS Region.

To create a project

1. Sign in to the AWSManagement Console and open the Amazon Pinpoint console at https://console.aws.amazon.com/pinpoint/.
2. Use the Region selector to choose the AWS Region that you want to use, as shown in the following image. If you're unsure, choose the Region that's located closest to you.



3. On the **All projects** page, choose **Create a project**.
4. In the **Create a project** window, for **Project name**, enter a name for the project. Choose **Create**.
5. On the **Configure features** page, choose **Skip this step**.
6. In the navigation pane, choose **All projects**.
7. On the **All projects** page, next to the project that you just created, copy the value that's shown in the **Project ID** column.

   Tip
   You need to use this ID in a few different places in this tutorial. Keep the project ID in a convenient place so that you can copy it later.

Step 1.2: Enable the Email Channel

After you create a project, you can start to configure features within that project. In this section, you enable the email channel.

1. In the navigation pane, under **Settings**, choose **Email**.
2. Next to **Identity details**, choose **Edit**.
3. On the **Edit email** page, under **Identity details**, choose **Enable the email channel for this project**.
4. Complete the steps in the next section to verify an identity.

Step 1.3: Verify an Identity

In this section, you verify an identity. In Amazon Pinpoint, an *identity* is an email address or domain that you use for sending email. When you verify an email address, you can send email from that address. When you verify a domain, you can send email from any address on that domain.

This section includes procedures for verifying an email address, as well as procedures for verifying an identity. You only need to verify one type of identity. In most cases, the process of verifying an email address is easier and faster. However, verifying a domain gives you more flexibility, because it allows you to send email using any address from that domain. For example, if you want to use a Reply-To address that's different from the From address, you might find it easier to verify the entire domain.

Verifying an Email Address

This section includes procedures for verifying an email address in Amazon Pinpoint. You don't need to complete these steps if you've already verified the domain (p. 52) that you want to use for sending email.

1. Under **Identity type**, choose **Email address**, and then choose **Verify a new email address**.
2. For **Email address**, enter the email address that you want to verify. The email address must be an address that you can access, and it has to be able to receive mail.
3. Choose **Verify email address**.
4. Choose **Save**.
5. Check the inbox of the address that you entered and look for an email from *no-reply-aws@amazon.com*. Open the email and click the link in the email to complete the verification process for the email address.

   Note
   You should receive the verification email within five minutes. If you don't receive the email, do the following:

   - Make sure you typed the address that you want to verify correctly.
   - Make sure the email address that you're attempting to verify is able to receive email. You can test this by using another email address to send a test email to the address that you want to verify.
   - Check your junk mail folder.

   The link in the verification email expires after 24 hours. To resend the verification email, choose **Send verification email again**.

When you verify an email address, consider the following:

- Amazon Pinpoint has endpoints in multiple AWS Regions. The verification status of an email address is separate for each Region. If you want to send email from the same identity in more than one Region, you must verify that identity in each Region. You can verify as many as 10,000 identities (email addresses and domains, in any combination) in each AWS Region.
- The *local part* of the email address, which is the part that precedes the at sign (@), is case sensitive. For example, if you verify *user@example.com*, you can't send email from *USER@example.com* unless you verify that address too.

- Domain names are case insensitive. For example, if you verify *user@example.com*, you can also send email from *user@EXAMPLE.com*.
- You can apply labels to verified email addresses by adding a plus sign (+) followed by a string of text after the local part of the address and before the at sign (@). For example, to apply *label1* to the address *user@example.com*, use *user+label1@example.com*. You can use as many labels as you want for each verified address. You can also use labels in the "From" and "Return-Path" fields to implement Variable Envelope Return Path (VERP).

  Note
  When you verify an unlabeled address, you are verifying all addresses that could be formed by adding a label to the address. However, if you verify a labeled address, you can't use other labels with that address.

Verifying a Domain

This section includes procedures for verifying a domain in Amazon Pinpoint. You don't need to complete these steps if you've already verified the email address (p. 51) that you want to use for sending email.

Note
To complete the steps in this section, you need to be able to modify the DNS settings for your domain. The exact steps required for modifying the DNS settings vary depending on which DNS provider you use. If you're unable to change the DNS settings for your domain, or you're not comfortable making these changes, contact your system administrator.

1. Complete the steps in the previous section to enable the email channel.
2. Under **Identity type**, choose **Domain**.
3. Choose **Verify a new domain**. Then, for **Domain**, enter the name of the domain that you want to verify.
4. For **Default sender address**, enter the default address that you want to use to send email from this domain.

   When you send email from this domain, you can send it from any address on the domain. However, if you don't specify a sender address, Amazon Pinpoint sends the email from the address that you specify in this section.
5. Choose **Verify domain**. Copy the three DNS name and record values that are shown under **Record set**. Alternatively, you can choose **Download record set** to download a spreadsheet that contains these records.

   When you finish, choose **Save**.
6. Log in to the management console for your DNS or web hosting provider, and then create three new CNAME records that contain the values that you saved in the previous step. See the next section for links to the documentation for several common providers.

   It usually takes 24–48 hours for DNS settings to propagate. As soon as Amazon Pinpoint detects all three of these CNAME records in the DNS configuration of your domain, the verification process is complete.

   Note
   You can't send email from a domain until the verification process is complete.

Instructions for Configuring DNS Records for Various Providers

The procedures for updating the DNS records for a domain vary depending on which DNS or web hosting provider you use. The following table lists links to the documentation for several common providers. This list isn't exhaustive, and inclusion in this list isn’t an endorsement or recommendation of any company’s products or services. If your provider isn't listed in the table, you can probably use the domain with Amazon Pinpoint.

DNS/Hosting Provider	Documentation Link
Amazon Route53	Creating Records by Using the Amazon Route53 Console
GoDaddy	Add a CNAME record (external link)
Dreamhost	How do I add custom DNS records? (external link)
Cloudflare	Managing DNS records in Cloudflare (external link)
HostGator	Manage DNS Records with HostGator/eNom (external link)
Namecheap	How do I add TXT/SPF/DKIM/DMARC records for my domain? (external link)
Names.co.uk	Changing your domains DNS Settings (external link)
Wix	Adding or Updating CNAME Records in Your Wix Account (external link)

Domain Verification Tips and Troubleshooting

If you completed the preceding steps but your domain isn't verified after 72 hours, check the following:

- Make sure that you entered the values for the DNS records in the correct fields. Some providers refer to the **Name/host** field as *Host* or *Hostname*. In addition, some providers refer to the **Record value** field as *Points to* or *Result*.
- Make sure that your provider didn't automatically append your domain name to the **Name/host** value that you entered in the DNS record. Some providers append the domain name without indicating that they've done so. If your provider appended your domain name to the **Name/host** value, remove the domain name from the end of the value. You can also try adding a period to the end of the value in the DNS record. This period indicates to the provider that the domain name is fully qualified.
- The underscore character (_) is required in the **Name/host** value of each DNS record. If your provider doesn't allow underscores in DNS record names, contact the provider's customer support department for additional assistance.
- The validation records that you have to add to the DNS configuration for your domain are different for each AWS Region. If you want to use a domain to send email from multiple AWS Regions, you have to verify the domain in each of those Regions.

Step 2: Add or configure endpoints

When you send campaigns in Amazon Pinpoint, you send them to endpoints. An endpoint represents a single method of contacting a customer. For example, a customer's phone number, their email address, and their unique Apple Push Notification Service (APNs) token are three separate endpoints. In this example, these three endpoints represent three different ways of communicating with a customer—in this case, by sending SMS messages, emails, or push notifications.

If you're new to Amazon Pinpoint, your Amazon Pinpoint project doesn't contain any endpoints. There are a few ways that you can add endpoints to Amazon Pinpoint:

- Import from a CSV or JSON file.
- Use the UpdateEndpoint API operation in the Amazon Pinpoint API.

- Export event data from an app that uses the AWS Mobile SDK or the AWS Amplify JavaScript library to send analytics data to Amazon Pinpoint.

To complete this tutorial, your Amazon Pinpoint project has to contain at least one email endpoint. If your Amazon Pinpoint project already contains email endpoints, you can use those. You can also use the sample CSV file in this section to import a list of test endpoints.

Import test endpoints

This section includes a file that you can use as a source file for importing a test endpoint into Amazon Pinpoint. This method of adding endpoints is helpful if you don't have an application that sends endpoint information to Amazon Pinpoint, if you don't want to use the Amazon Pinpoint API to create endpoints, or if you don't want to modify the endpoints that already exist in your Amazon Pinpoint project.

Create the source file

First, create the CSV file that contains information about your endpoints.

To create the CSV file

1. In a text editor, create a new file.
2. Paste the following text into the file.

```
ChannelType,Address,EndpointStatus,OptOut,Id,Attributes.Email,Attributes.EndpointId,User.UserAttrib
EMAIL,recipient@example.com,ACTIVE,NONE,12345,recipient
%40example.com,12345,Carlos,Salazar
```

3. In the text from the previous step, make the following changes:
   - Replace *`recipient@example.com`* with your email address.
   - Replace *`recipient%40example.com`* with your email address, but use `%40` instead of the at sign (@).
   - Replace *`Carlos`* and *`Salazar`* with your first and last name, respectively.
   - (Optional) Replace both occurrences of *`12345`* with an endpoint ID value. The value that you use has to be exactly the same in both locations.
4. Save the file as `testImports.csv`.

Upload the file

Amazon Pinpoint requires you to store source files in Amazon S3. Complete the procedure below to upload your CSV file to a folder in an Amazon S3 bucket.

To upload the file to an Amazon S3 bucket

1. Open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. Choose **Create bucket**.
3. On the **Create bucket** window, for **Bucket name**, enter a name for the bucket that you want to store the file in. The name that you specify has to be unique. Also, there are several restrictions related to the bucket name. For more information about these restrictions, see Bucket restrictions in the *Amazon Simple Storage Service Developer Guide*. Choose **Create**.
4. In the list of buckets, choose the bucket that you just created.
5. Choose **Create folder**. Name the folder `Imports`. Choose **Save**.

6. Choose the **Imports** folder that you just created.
7. Choose **Upload**.
8. Choose **Add files**. Locate the `testImports.csv` file that you created in the previous section (p. 54). Choose **Upload**.

Import the file into Amazon Pinpoint

After you upload your CSV file into an Amazon S3 bucket, you can import it into your Amazon Pinpoint project.

To import the file into Amazon Pinpoint

1. Open the Amazon Pinpoint console at https://console.aws.amazon.com/pinpoint/.
2. On the **All projects** page, choose the project that you created in Step 1.1 (p. 49).
3. In the navigation pane, choose **Segments**.
4. Choose **Create a segment**.
5. On the **Create a segment** page, choose **Import a segment**.
6. Under **Specifications**, do the following:
   - For **Name**, enter `EmailRegistrationTestRecipients`.
   - For **Amazon Simple Storage Service URL**, enter the address of the Amazon S3 bucket and folder that you created in the previous section. For example, if your bucket is named email-registration-tutorial, enter the following URL: `s3://email-registration-tutorial/Imports`.
   - Under **IAM role**, choose **Automatically create a role**. Then, enter a name for the IAM role, such as `SegmentImportRole`.
   - Under **What type of file are you importing**, choose **Comma-Separated Values**. Choose **Create segment**.
7. After you submit the segment, you see the **Scheduled imports** tab on the **Segments** page. Wait for 30 seconds after you submit the segment, and then refresh the page. The **Import status** column should indicate that the segment import is complete. If it indicates that the import is still pending, wait an additional 30 seconds, and then refresh the page again. Repeat this process until the status of the import is **Completed**.
8. On the **Segments** tab, choose the **EmailRegistrationTestRecipients** segment. In the **Import details** section, confirm that the value under **Number of records** is accurate. If you used the sample CSV file in this section, the correct value is **1**.

Use existing endpoints (for advanced users)

If your Amazon Pinpoint project already contains email endpoints, you have to modify those endpoints slightly before you can use them in this tutorial. First, the endpoints have to include two endpoint Attribute values. These values are listed in the following table.

Attribute name	Value
`Attributes.Email`	A URL-encoded version of the existing `Address` attribute for the endpoint.
`Attributes.EndpointId`	Equal to the existing `Id` attribute for the endpoint. If the endpoint ID value contains non-alphanumeric characters, then you should URL-encode this value.

Attribute name	Value
	Note For more information about URL encoding, see the HTML URL encoding reference on the W3Schools website.

The endpoints should also contain two User Attribute values. These values aren't strictly required for the solution to work. These values are listed in the following table.

Attribute name	Value
`User.UserAttributes.FirstName`	The recipient's first name.
`User.UserAttributes.LastName`	The recipient's last name.

If your Amazon Pinpoint project contains a large number of existing endpoints, you can use the CreateExportJob API operation to export a list of all the endpoints for a segment or project to an Amazon S3 bucket. After that, you can write a script that uses the UpdateEndpoint operation to programmatically updates endpoints to include these attributes.

Step 3: Create IAM policies and roles

The next step in implementing the email preference management solution is to configure a policy and a role in AWS Identity and Access Management (IAM). For this solution, you need to create a policy that provides access to certain resources that are related to Amazon Pinpoint. You then create a role and attach the policy to it. Later in this tutorial, you create an AWS Lambda function that uses this role to call certain operations in the Amazon Pinpoint API.

Step 3.1: Create an IAM policy

This section shows you how to create an IAM policy. Users and roles that use this policy are able to view, create, and update Amazon Pinpoint endpoints.

In this tutorial, you want to give Lambda the ability to perform these tasks. However, for added security, this policy uses the principal of granting *least privilege*. In other words, it grants only the permissions that are required to complete this solution, and no more. You can only use this policy to view, create, or update endpoints that are associated with a specific Amazon Pinpoint project.

To create the policy

1. Sign in to the AWSManagement Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies**, and then choose **Create policy**.
3. On the **JSON** tab, paste the following code.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:CreateLogGroup"
            ],
            "Resource": "arn:aws:logs:*:*:*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "mobiletargeting:UpdateEndpoint",
                "mobiletargeting:GetEndpoint"
            ],
            "Resource": "arn:aws:mobiletargeting:region:accountId:apps/projectId/
endpoints/*"
        }
    ]
}
```

In the preceding example, do the following:

- Replace *region* with the AWS Region that you use Amazon Pinpoint in, such as `us-east-1` or `eu-central-1`.

  Tip
  For a complete list of AWS Regions where Amazon Pinpoint is available, see AWS regions and endpoints in the *AWS General Reference*.
- Replace *accountId* with the unique ID for your AWS account.
- Replace *projectId* with the unique ID of the project that you created in Step 1.1 (p. 49) of this tutorial.

  Note
  The `logs` actions enable Lambda to log its output in CloudWatch Logs.

4. Choose **Review policy**.
5. For **Name**, enter a name for the policy, such as **`EmailPreferencesPolicy`**. Choose **Create policy**.

Step 3.2: Create an IAM role

After you create the policy, you can create a role and attach the policy to it. The Lambda function that you create in Step 3 (p. 56) uses this role in order to gain access to the necessary AWS resources.

To create the role

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the IAM console, in the navigation pane, choose **Roles**, and then choose **Create role**.
3. Under **Choose the service that will use this role**, choose **Lambda**, and then choose **Next: Permissions**.

   Note
   The service that you choose in this step isn't important—regardless of the service that you choose, you apply your own policy in the next step.
4. Under **Attach permissions policies**, choose the policy that you created in the previous section, and then choose **Next: Tags**.
5. Choose **Next: Review**.
6. Under **Review**, for **Name**, enter a name for the role, such as **`EmailPreferencesForm`**. Choose **Create role**.

Step 4: Create the Lambda function

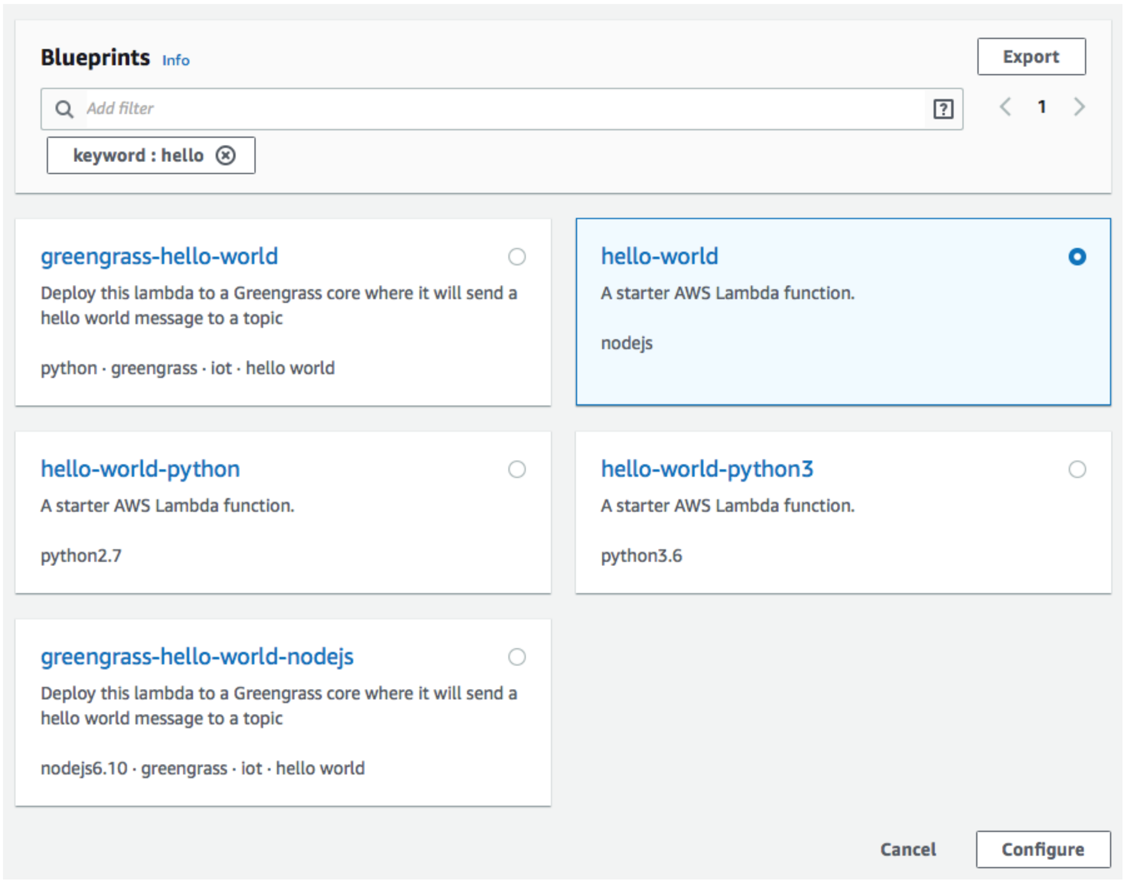
This solution uses a Lambda function to update customers' endpoint records based on the preferences that they provide on the web form. This section shows you how to create, configure, and test this function. Later, you set up API Gateway and Amazon Pinpoint to execute this function.

Step 4.1: Create the function that updates endpoints

The first function takes input from the preference management web form, which it receives from Amazon API Gateway. It uses this information to update the customer's endpoint record according to the preferences that they provided on the form.

To create the Lambda function

1. Open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
2. Choose **Create function**.
3. Under **Create a function**, choose **Blueprints**.
4. In the search field, enter **`hello`**, and then press Enter. In the list of results, choose the `hello-world` Node.js function, as shown in the following image. Choose **Configure**.



greengrass-hello-world
Deploy this lambda to a Greengrass core where it will send a hello world message to a topic
python · greengrass · iot · hello world

hello-world
A starter AWS Lambda function.
nodejs

hello-world-python
A starter AWS Lambda function.
python2.7

hello-world-python3
A starter AWS Lambda function.
python3.6

greengrass-hello-world-nodejs
Deploy this lambda to a Greengrass core where it will send a hello world message to a topic
nodejs6.10 · greengrass · iot · hello world

5. Under **Basic information**, do the following:
   - For **Name**, enter a name for the function, such as **`EmailPreferences`**.
   - For **Role**, select **Choose an existing role**.
   - For **Existing role**, choose the **EmailPreferencesForm** role that you created in Step 3.2 (p. 57).

When you finish, choose **Create function**.

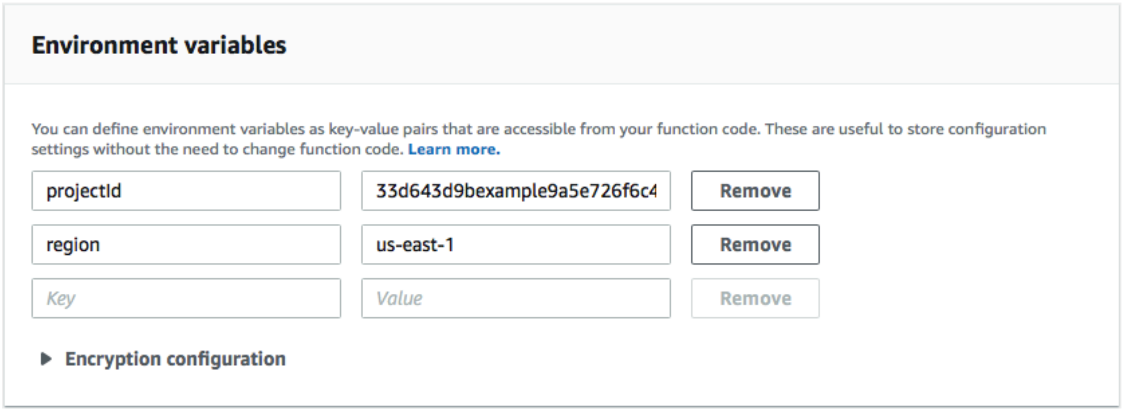
6. Delete the example code in the code editor, and then paste the following code:

```
var AWS = require('aws-sdk');
var pinpoint = new AWS.Pinpoint({region: process.env.region});
var projectId = process.env.projectId;

exports.handler = (event, context, callback) => {
  console.log('Received event:', event);
  var optOutAll = event.optOutAll;

  if (optOutAll === false) {
    updateOpt(event);
  } else if (optOutAll === true) {
    unsubAll(event);
  }
};

function unsubAll(event) {

  var params = {
    ApplicationId: projectId,
    EndpointId: event.endpointId,
    EndpointRequest: {
      ChannelType: 'EMAIL',
      OptOut: 'ALL',
      Attributes: {
        SpecialOffersOptStatus: [
          "OptOut"
        ],
        NewProductsOptStatus: [
          "OptOut"
        ],
        ComingSoonOptStatus: [
          "OptOut"
        ],
        DealOfTheDayOptStatus: [
          "OptOut"
        ],
        OptStatusLastChanged: [
          event.optTimestamp
        ],
        OptSource: [
          event.source
        ]
      }
    }
  };
  pinpoint.updateEndpoint(params, function(err,data) {
    if (err) {
      console.log(err, err.stack);
    }
    else {
      console.log(data);
    }
  });


}

function updateOpt(event) {
  var endpointId = event.endpointId,
      firstName = event.firstName,
      lastName = event.lastName,
      source = event.source,
      specialOffersOptStatus = event.topic1,
      newProductsOptStatus = event.topic2,
      comingSoonOptStatus = event.topic3,
      dealOfTheDayOptStatus = event.topic4,
      optTimestamp = event.optTimestamp;

  var params = {
    ApplicationId: projectId,
    EndpointId: endpointId,
    EndpointRequest: {
      ChannelType: 'EMAIL',
      OptOut: 'NONE',
      Attributes: {
        SpecialOffersOptStatus: [
          specialOffersOptStatus
        ],
        NewProductsOptStatus: [
          newProductsOptStatus
        ],
        ComingSoonOptStatus: [
          comingSoonOptStatus
        ],
        DealOfTheDayOptStatus: [
          dealOfTheDayOptStatus
        ],
        OptStatusLastChanged: [
          optTimestamp
        ],
        OptSource: [
          source
        ]
      },
      User: {
        UserAttributes: {
          FirstName: [
            firstName
          ],
          LastName: [
            lastName
          ]
        }
      }
    }
  };
  pinpoint.updateEndpoint(params, function(err,data) {
    if (err) {
      console.log(err, err.stack);
    }
    else {
      console.log(data);
    }
  });
}
```

7. Under **Environment variables**, do the following:

   - In the first row, create a variable with a key of **`projectId`**. Next, set the value to the unique ID of the project that you created in Step 1.1 (p. 49).
   - In the second row, create a variable with a key of **`region`**. Next, set the value to the Region that you use Amazon Pinpoint in, such as **`us-east-1`** or **`us-west-2`**.

When you finish, the **Environment Variables** section should resemble the example shown in the following image.


Environment variables

You can define environment variables as key-value pairs that are accessible from your function code. These are useful to store configuration settings without the need to change function code. Learn more.

projectId	33d643d9bexample9a5e726f6c4	Remove
region	us-east-1	Remove
Key	Value	Remove

8. At the top of the page, choose **Save**.

Step 4.1.1: Test the function

After you create the function, you should test it to make sure that it's configured properly. Also, make sure that the IAM role you created has the appropriate permissions.

To test the function

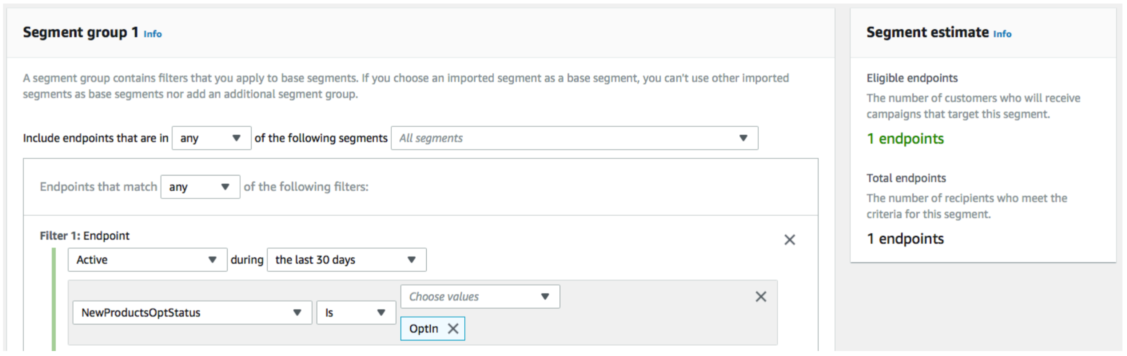
1. Choose **Test**.
2. On the **Configure test event** window, do the following:
   - Choose **Create new test event**.
   - For **Event name**, enter a name for the test event, such as **`MyTestEvent`**.
   - Erase the example code in the code editor. Paste the following code:

```
{
  "endpointId": "12345",
  "firstName": "Carlos",
  "lastName": "Salazar",
  "topic1": "OptOut",
  "topic2": "OptIn",
  "topic3": "OptOut",
  "topic4": "OptIn",
  "source": "Lambda test",
  "optTimestamp": "Tue Mar 05 2019 14:35:07 GMT-0800 (PST)",
  "optOutAll": false
}
```

   - In the preceding code example, replace the values of the `endpointId`, `firstName`, and `lastName` attributes with the values for the endpoint that you imported in Step 2 (p. 54).
   - Choose **Create**.
3. Choose **Test** again.
4. Under **Execution result: succeeded**, choose **Details**. In the **Log output** section, review the output of the function. Make sure that the function ran without errors.
5. Open the Amazon Pinpoint console at https://console.aws.amazon.com/pinpoint/.
6. On the **All projects** page, choose the project that you created in Step 1.1 (p. 49).
7. In the navigation pane, choose **Segments**. On the **Segments page**, choose **Create a segment**.

8. In **Segment group 1**, under **Add filters to refine your segment**, choose **Filter by endpoint**.
9. For **Choose an endpoint attribute**, choose **NewProductsOptStatus**. Then, for **Choose values**, choose **OptIn**.

   The **Segment estimate** section should show that there is one eligible endpoint, and one total endpoint, as shown in the following image.



Segment group 1 Info

A segment group contains filters that you apply to base segments. If you choose an imported segment as a base segment, you can't use other imported segments as base segments nor add an additional segment group.

Include endpoints that are in: any — of the following segments: All segments

Endpoints that match: any — of the following filters:

Filter 1: Endpoint
Active during the last 30 days
NewProductsOptStatus Is OptIn

Segment estimate Info

Eligible endpoints
The number of customers who will receive campaigns that target this segment.
1 endpoints

Total endpoints
The number of recipients who meet the criteria for this segment.
1 endpoints

Step 5: Set up Amazon API Gateway

In this section, you create a new API by using Amazon API Gateway. The registration form that you deploy in this solution calls this API. API Gateway then passes the information that's captured on the email preferences page to the Lambda function you created in Step 4 (p. 58).

Step 5.1: Create the API

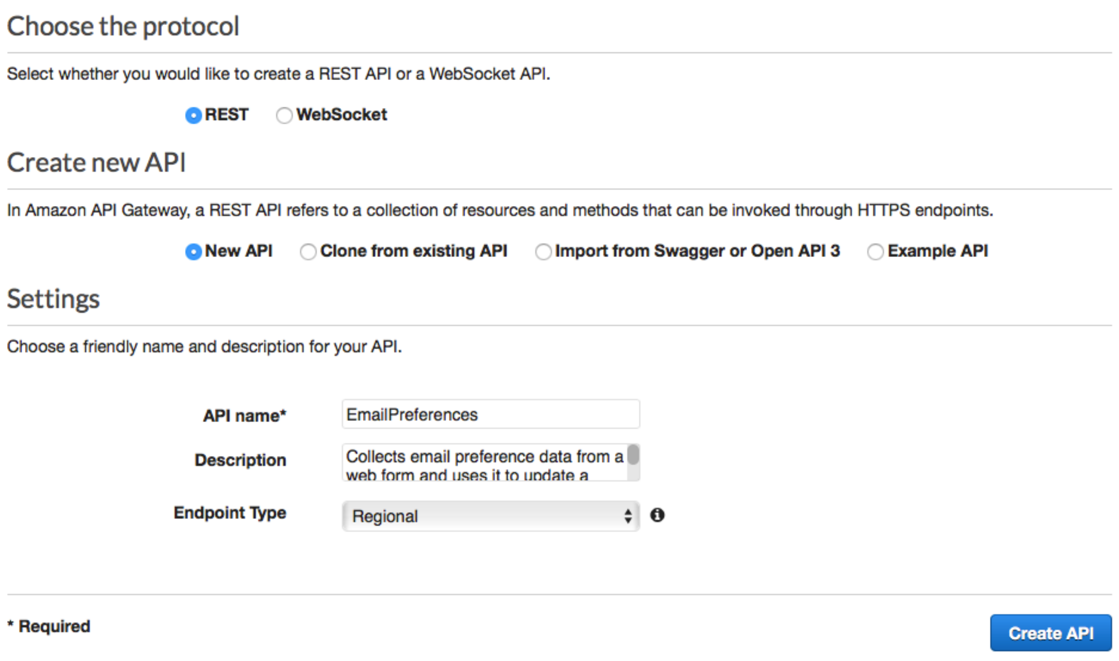
First, you have to create a new API in API Gateway. The following procedures show you how to create a new REST API.

To create a new API

1. Open the API Gateway console at https://console.aws.amazon.com/apigateway/.
2. Choose **Create API**. Make the following selections:

   - Under **Choose the protocol**, choose **REST**.
   - Under **Create new API**, choose **New API**.
   - Under **Settings**, for **Name**, enter a name, such as `EmailPreferences`. For **Description**, optionally enter some text that describes the purpose of the API. For **Endpoint Type**, choose **Regional**. Then, choose **Create API**.

   An example of these settings is shown in the following image.


Choose the protocol

Select whether you would like to create a REST API or a WebSocket API.

[✓ REST] WebSocket

Create new API

In Amazon API Gateway, a REST API refers to a collection of resources and methods that can be invoked through HTTPS endpoints.

[✓ New API] Clone from existing API | Import from Swagger or Open API 3 | Example API

Settings

Choose a friendly name and description for your API.

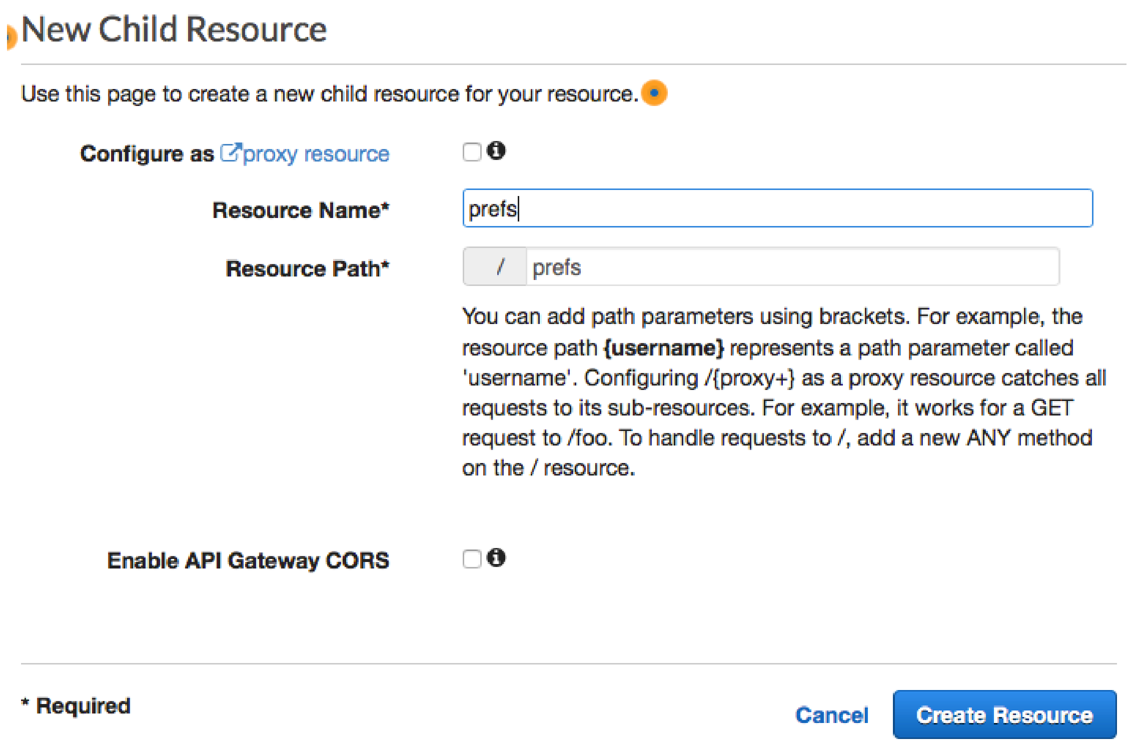
API name*	EmailPreferences
Description	Collects email preference data from a web form and uses it to update a
Endpoint Type	Regional

* Required

Step 5.2: Create a resource

Now that you've created an API, you can start to add resources to it. After that, you add a POST method to the resource, and tell API Gateway to pass the data that you receive from this method to your Lambda function.

1. On the **Actions** menu, choose **Create Resource**. In the **New Child Resource** pane, for **Resource Name**, enter **`prefs`**, as shown in the following image. Choose **Create Resource**.



New Child Resource

Use this page to create a new child resource for your resource.

Configure as proxy resource ☐

Resource Name* prefs

Resource Path* / prefs

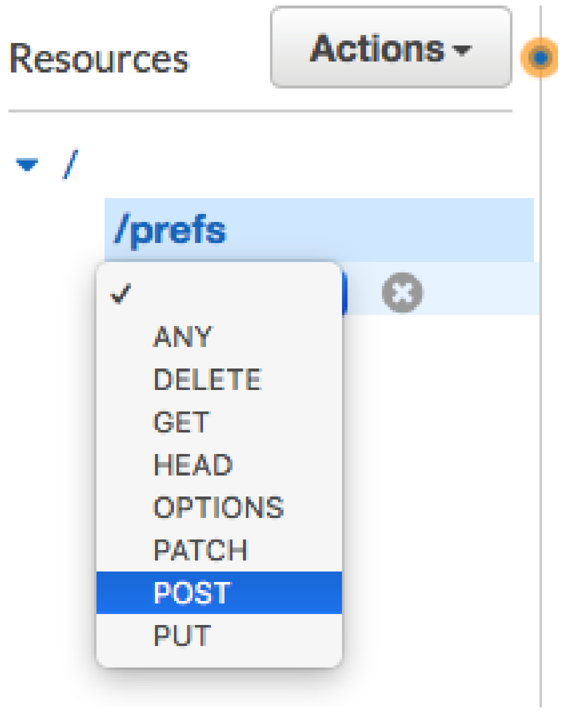
You can add path parameters using brackets. For example, the resource path **{username}** represents a path parameter called 'username'. Configuring /{proxy+} as a proxy resource catches all requests to its sub-resources. For example, it works for a GET request to /foo. To handle requests to /, add a new ANY method on the / resource.

Enable API Gateway CORS ☐

* Required

Cancel | Create Resource

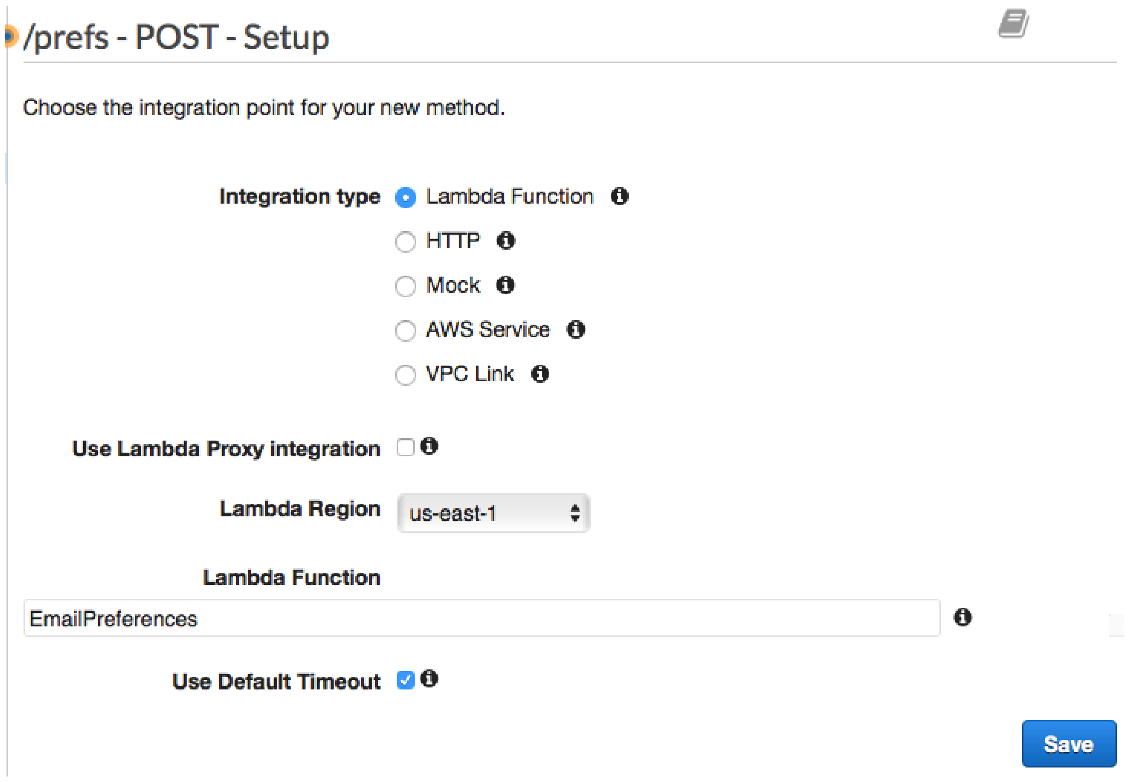
2. On the **Actions** menu, choose **Create Method**. From the menu that appears, choose **POST**, as shown in the following image. Then choose the **check mark** (✓) button.



3. In the **/prefs - POST - Setup** pane, make the following selections:

- For **Integration type**, choose **Lambda Function**.
- Choose **Use Lambda Proxy Integration**.
- For **Lambda Region**, choose the Region that you created the Lambda function in.
- For **Lambda Function**, choose the `EmailPreferences` function that you created in Step 4 (p. 58).

An example of these settings is shown in the following image.



/prefs - POST - Setup

Choose the integration point for your new method.

Integration type: [✓ Lambda Function] HTTP Mock AWS Service VPC Link

Use Lambda Proxy integration: ☐

Lambda Region: us-east-1

Lambda Function: EmailPreferences

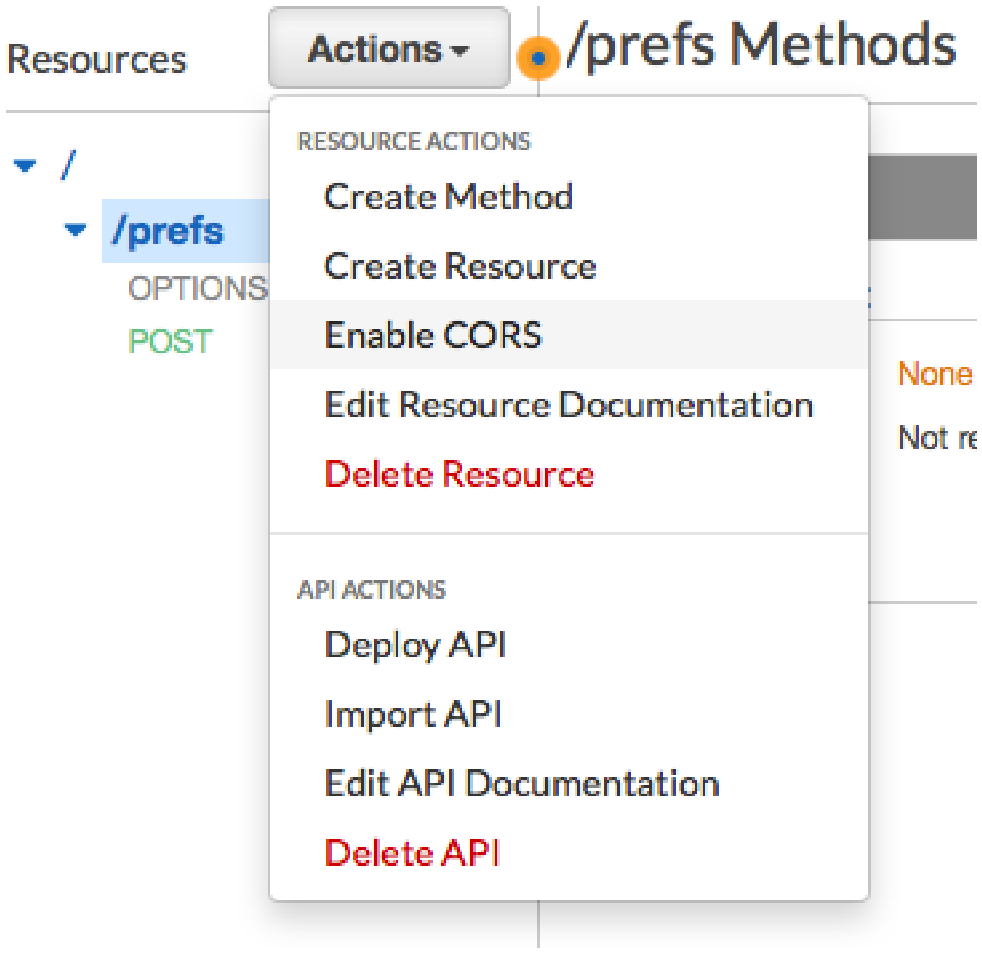
Use Default Timeout: ✓

Choose **Save**. On the window that appears, choose **OK** to give API Gateway permission to execute your Lambda function.

Step 5.3: Deploy the API

The API is now ready to use. At this point, you have to deploy it in order to create a publicly accessible endpoint.

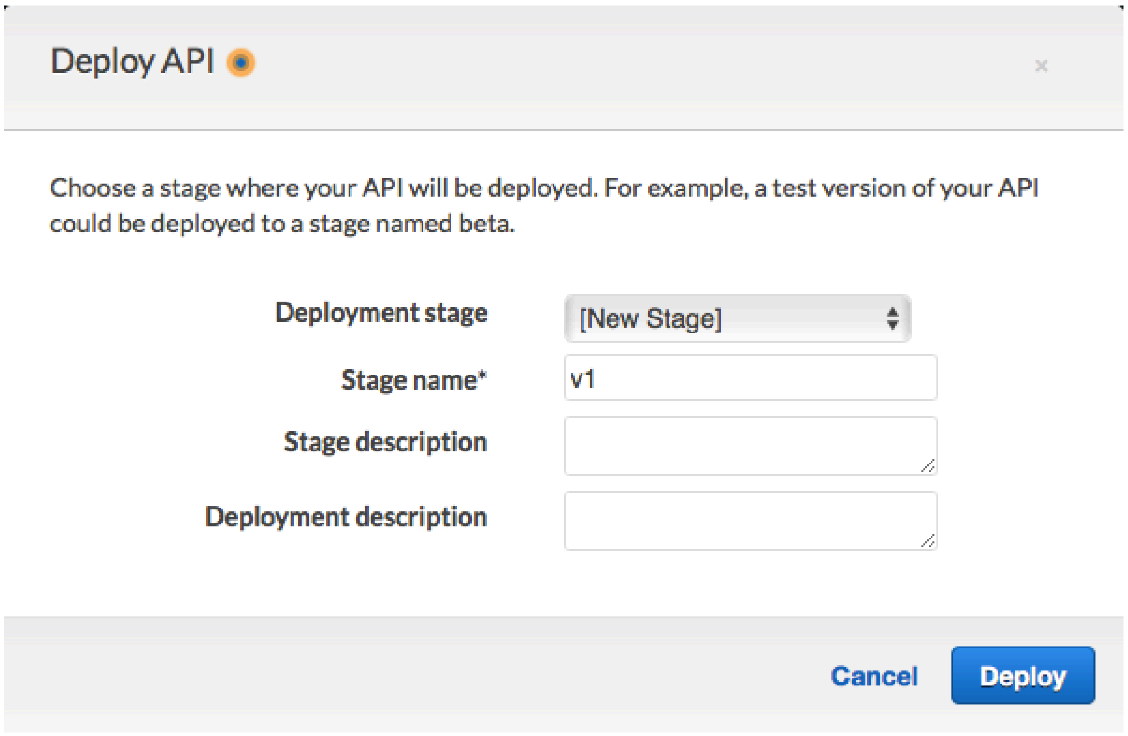
1. In the navigation pane, choose **Resources**. In the list of resources, choose the **/prefs** resource. Finally, on the **Actions** menu, choose **Enable CORS**, as shown in the following image.



2. On the **Enable CORS** pane, choose **Enable CORS and replace existing CORS headers**.
3. On the **Actions** menu, choose **Deploy API**. On the **Deploy API** window, make the following selections:

   - For **Deployment stage**, choose **[New Stage]**.
   - For **Stage name**, enter **v1**.
   - Choose **Deploy**.

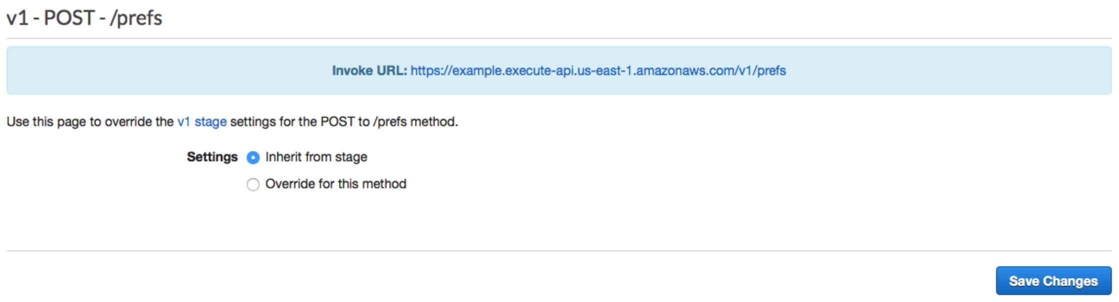
   An example of these selections is shown in the following image.



Choose a stage where your API will be deployed. For example, a test version of your API could be deployed to a stage named beta.

Deployment stage	[New Stage]
Stage name*	v1
Stage description	
Deployment description	

4. In the **v1 Stage Editor** pane, choose the **/prefs** resource, and then choose the **POST** method. Copy the address that's shown next to **Invoke URL**, as shown in the following image.



v1 - POST - /prefs

Invoke URL: https://example.execute-api.us-east-1.amazonaws.com/v1/prefs

Use this page to override the v1 stage settings for the POST to /prefs method.

Settings [✓ Inherit from stage] Override for this method

Step 6: Create and deploy the web form

All of the components of this solution that use AWS services are now in place. The last step is to create and deploy the web form that captures customer's data.

Step 6.1: Create the JavaScript form handler

In this section, you create a JavaScript function that parses the content of the web form that you create in the next section. After parsing the content, this function sends the data to the API that you created in Part 4 (p. 58).

To create the form handler

1. In a text editor, create a new file.
2. In the editor, paste the following code.

```
$(document).ready(function() {
 // Function that parses URL parameters.
 function getParameterByName(name) {
   name = name.replace(/[\[]/, "\\[").replace(/[\]]/, "\\]");
   var regex = new RegExp("[\\?&]" + name + "=([^&#]*)"),
     results = regex.exec(location.search);
   return results == null ? "" : decodeURIComponent(results[1].replace(/\+/g, " "));
 }

 // Pre-fill form data.
 $("#firstName").val(getParameterByName("firstName"));
 $("#lastName").val(getParameterByName("lastName"));
 $("#email").val(getParameterByName("email"));
 $("#endpointId").val(getParameterByName("endpointId"));
 if (getParameterByName("t1") == "OptIn") {
   $("#topic1In").prop('checked', true);
 }
 if (getParameterByName("t2") == "OptIn") {
   $("#topic2In").prop('checked', true);
 }
 if (getParameterByName("t3") == "OptIn") {
   $("#topic3In").prop('checked', true);
 }
 if (getParameterByName("t4") == "OptIn") {
   $("#topic4In").prop('checked', true);
 }

 // Handle form submission.
 $("#submit").click(function(e) {

   // Get endpoint ID from URL parameter.
   var endpointId = getParameterByName("endpointId");

   var firstName = $("#firstName").val(),
     lastName = $("#lastName").val(),
     email = $("#email").val(),
     source = window.location.pathname,
     optTimestamp = undefined,
     utcSeconds = Date.now() / 1000,
     timestamp = new Date(0);

   var topicOptIn = [
     $('#topic1In').is(':checked'),
     $('#topic2In').is(':checked'),
     $('#topic3In').is(':checked'),
     $('#topic4In').is(':checked')
   ];

   e.preventDefault();

   $('#submit').prop('disabled', true);
   $('#submit').html('<span class="spinner-border spinner-border-sm" role="status"
aria-hidden="true"></span>Saving your preferences</button>');

   // If customer unchecks a box, or leaves a box unchecked, set the opt
   // status to "OptOut".
   for (i = 0; i < topicOptIn.length; i++) {
     if (topicOptIn[i] == true) {
       topicOptIn[i] = "OptIn";
     } else {
       topicOptIn[i] = "OptOut";
     }
   }

   timestamp.setUTCSeconds(utcSeconds);
```

```
    var data = JSON.stringify({
      'endpointId': endpointId,
      'firstName': firstName,
      'lastName': lastName,
      'topic1': topicOptIn[0],
      'topic2': topicOptIn[1],
      'topic3': topicOptIn[2],
      'topic4': topicOptIn[3],
      'source': source,
      'optTimestamp': timestamp.toString(),
      'optOutAll': false
    });

    $.ajax({
      type: 'POST',
      url: 'https://example.execute-api.us-east-1.amazonaws.com/v1/prefs',
      contentType: 'application/json',
      data: data,
      success: function(res) {
        $('#form-response').html('<div class="mt-3 alert alert-success"
role="alert">Your preferences have been saved!</div>');
        $('#submit').prop('hidden', true);
        $('#unsubAll').prop('hidden', true);
        $('#submit').text('Preferences saved!');
      },
      error: function(jqxhr, status, exception) {
        $('#form-response').html('<div class="mt-3 alert alert-danger" role="alert">An
error occurred. Please try again later.</div>');
        $('#submit').text('Save preferences');
        $('#submit').prop('disabled', false);
      }
    });
  });

  // Handle the case when the customer clicks the "Unsubscribe from all"
  // button.
  $("#unsubAll").click(function(e) {
    var firstName = $("#firstName").val(),
      lastName = $("#lastName").val(),
      source = window.location.pathname,
      optTimestamp = undefined,
      utcSeconds = Date.now() / 1000,
      timestamp = new Date(0);

    // Get endpoint ID from URL parameter.
    var endpointId = getParameterByName("endpointId");

    e.preventDefault();

    $('#unsubAll').prop('disabled', true);
    $('#unsubAll').html('<span class="spinner-border spinner-border-sm" role="status"
aria-hidden="true"></span>Saving your preferences</button>');

    // Uncheck all boxes to give user a visual representation of the opt-out action.
    $("#topic1In").prop('checked', false);
    $("#topic2In").prop('checked', false);
    $("#topic3In").prop('checked', false);
    $("#topic4In").prop('checked', false);

    timestamp.setUTCSeconds(utcSeconds);

    var data = JSON.stringify({
      'endpointId': endpointId,
      'source': source,
      'optTimestamp': timestamp.toString(),
      'optOutAll': true
    });

    $.ajax({
      type: 'POST',
      url: 'https://example.execute-api.us-east-1.amazonaws.com/v1/prefs',
      contentType: 'application/json',
      data: data,
      success: function(res) {
        $('#form-response').html('<div class="mt-3 alert alert-info"
 role="alert">Successfully opted you out of all future email communications.</div>');
        $('#submit').prop('hidden', true);
        $('#unsubAll').prop('hidden', true);
      },
      error: function(jqxhr, status, exception) {
        $('#form-response').html('<div class="mt-3 alert alert-danger" role="alert">An
 error occurred. Please try again later.</div>');
      }
    });
  });
});
```

3. In the preceding example, replace *`https://example.execute-api.us-east-1.amazonaws.com/v1/prefs`* with the **Invoke URL** that you obtained in Step 5.3 (p. 65).
4. Save the file.

Step 6.2: Create the form file

In this section, you create an HTML file that contains the form that customers use to manage their email preferences. This file uses the JavaScript form handler that you created in the previous section to transmit the form data to your Lambda function.

Important
When a user submits this form, it triggers a Lambda function that updates endpoints in your Amazon Pinpoint project. Malicious users could launch an attack on your form that could impact your data or cause a large number of requests to be made. If you plan to use this solution for a production use case, you should secure it by using a system such as Google reCAPTCHA.

To create the form

1. In a text editor, create a new file.
2. In the editor, paste the following code.

```
<!doctype html>
<html lang="en">

<head>
  <!-- Required meta tags -->
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-
fit=no">
  <!-- Bootstrap CSS -->
  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/
css/bootstrap.min.css" integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/
iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">
  <!-- Bootstrap and AJAX scripts -->
  <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-
q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"
 crossorigin="anonymous"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/
popper.js/1.14.7/umd/popper.min.js" integrity="sha384-
UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1"
 crossorigin="anonymous"></script>
```

```
  <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"
 integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM"
 crossorigin="anonymous"></script>
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></
script>

  <!-- Reference to JavaScript form handler. -->
  <script type="text/javascript" src="formHandler.js"></script>

  <title>Manage your email preferences</title>
</head>

<body>
  <div class="container">
    <h1>Manage your email subscriptions</h1>
    <h2 class="mt-3">Contact information</h2>
    <form>

      <div class="form-row mt-3">
        <div class="form-group col-md-6">
          <label for="email" class="font-weight-bold">Email address</label>
          <input type="email" class="form-control-plaintext" id="email" readonly>
        </div>
        <div class="form-group col-md-6">
          <label for="email" class="font-weight-bold">ID</label>
          <input type="email" class="form-control-plaintext" id="endpointId" readonly>
        </div>
      </div>
      <div class="form-row">
        <div class="form-group col-md-6">
          <label for="firstName" class="font-weight-bold">First name</label>
          <input type="text" class="form-control" id="firstName">
        </div>
        <div class="form-group col-md-6">
          <label for="lastName" class="font-weight-bold">Last name</label>
          <input type="text" class="form-control" id="lastName">
        </div>
      </div>

      <div class="form-row">
        <div class="col-md-12">
          <p class="font-weight-bold mt-3">Subscriptions</p>
        </div>
      </div>

      <!-- Change topic names here, if necessary -->
      <div class="form-row">
        <div class="col-md-6">
          <div class="custom-control custom-switch">
            <input type="checkbox" class="custom-control-input" id="topic1In">
            <label class="custom-control-label font-weight-bold" for="topic1In">Special
 offers</label>
            <p>Get exclusive offers available only to subscribers.</p>
          </div>
        </div>
        <div class="col-md-6">
          <div class="custom-control custom-switch">
            <input type="checkbox" class="custom-control-input" id="topic2In">
            <label class="custom-control-label font-weight-bold" for="topic2In">Coming
 soon</label>
            <p>Learn about our newest products before anyone else!</p>
          </div>
        </div>
      </div>

      <div class="form-row mt-3">
```

```
        <div class="col-md-6">
          <div class="custom-control custom-switch">
            <input type="checkbox" class="custom-control-input" id="topic3In">
            <label class="custom-control-label font-weight-bold" for="topic3In">New
 products</label>
            <p>Get the inside scoop on products that haven't been released yet.</p>
          </div>
        </div>
        <div class="col-md-6">
          <div class="custom-control custom-switch">
            <input type="checkbox" class="custom-control-input" id="topic4In">
            <label class="custom-control-label font-weight-bold" for="topic4In">Deal of
 the Day</label>
            <p>Get special deals in your inbox every day!</p>
          </div>
        </div>
      </div>

      <div id="form-response"></div>

      <div class="row mt-3">
        <div class="col-md-12 text-center">
          <button type="submit" id="submit" class="btn btn-primary">Update
 preferences</button>
        </div>
      </div>

      <div class="row mt-3">
        <div class="col-md-12 text-center">
          <button type="button" class="btn btn-link" id="unsubAll">Unsubscribe from all
 email communications</button>
        </div>
      </div>
    </form>

    <div class="row mt-3">
      <div class="col-md-12 text-center">
        <small class="text-muted">Copyright © 2019, ExampleCorp or its affiliates.</
small>
      </div>
    </div>

  </div>
</body>

</html>
```

3. In the preceding example, replace *`formHandler.js`* with the full path to the form handler JavaScript file that you created in the previous section.
4. Save the file.

Step 6.3: Upload the form files

Now that you've created the HTML form and the JavaScript form handler, the last step is to publish these files to the internet. This section assumes that you have an existing web hosting provider. If you don't have an existing hosting provider, you can launch a website by using Amazon Route53, Amazon Simple Storage Service (Amazon S3), and Amazon CloudFront. For more information, see Host a static website.

If you use another web hosting provider, consult the provider's documentation for more information about publishing webpages.

Step 6.4: Test the form

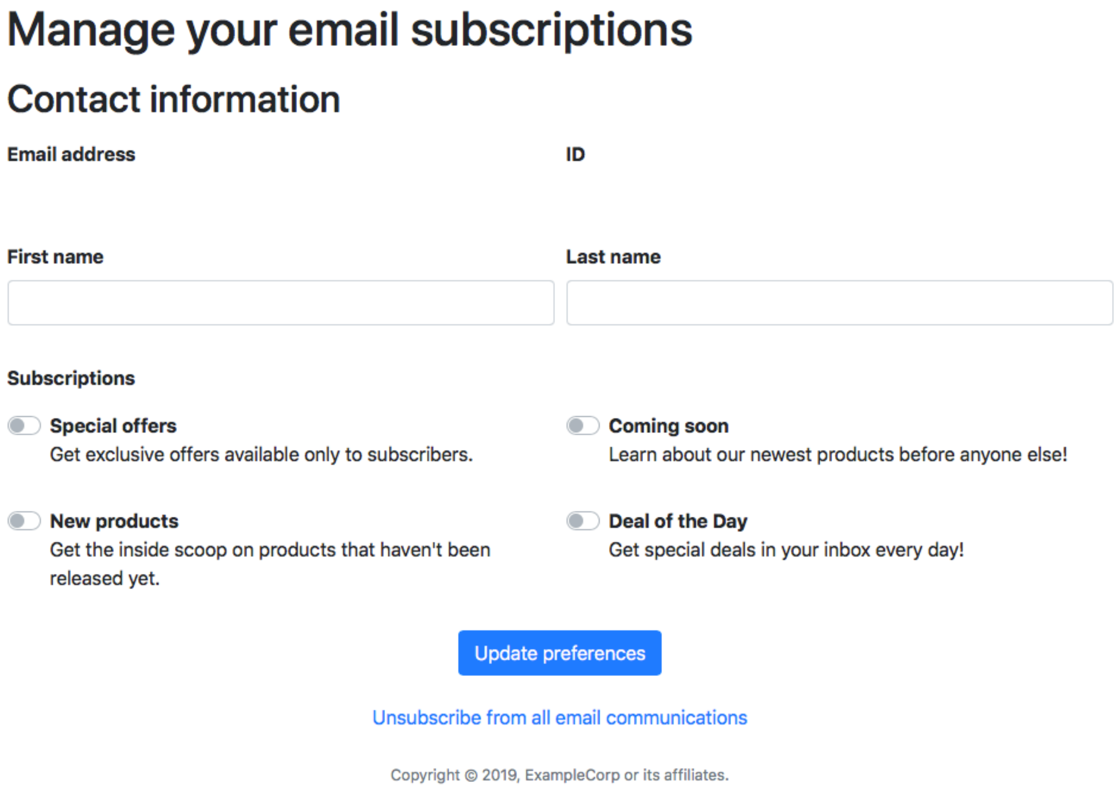
After you publish the form, you should test it to confirm that it works properly.

Step 6.4.1: Use the form to submit test data

The first step in testing the preferences form is to use it to submit some test data. In this case, you opt in to all of the subscription topics that are listed on the form.

To submit test data

1. In a web browser, go to the location where you uploaded the preferences page. If you used the code example from Step 6.2 (p. 70), you see a form that resembles the example in the following image.



Manage your email subscriptions

Contact information

Email address

ID

First name

Last name

Subscriptions

Special offers
Get exclusive offers available only to subscribers.

Coming soon
Learn about our newest products before anyone else!

New products
Get the inside scoop on products that haven't been released yet.

Deal of the Day
Get special deals in your inbox every day!

Update preferences

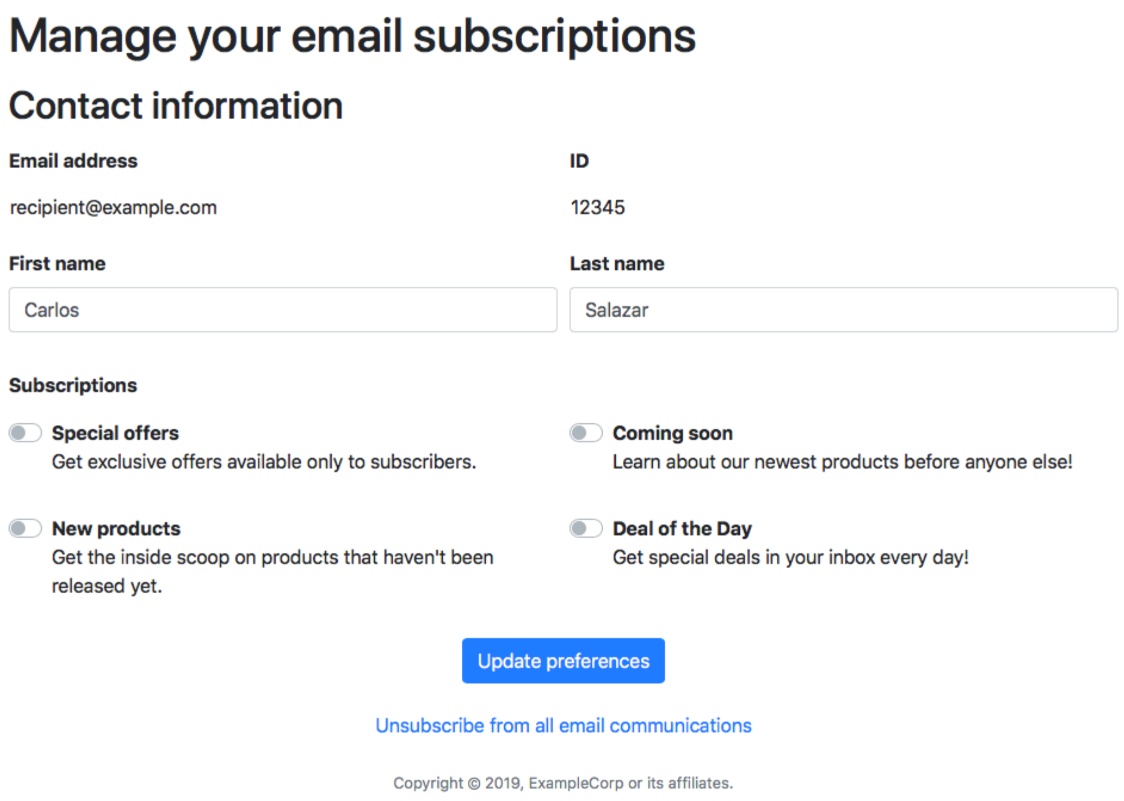
Unsubscribe from all email communications

Copyright © 2019, ExampleCorp or its affiliates.

2. Add the following string to the end of the URL. Replace the values for *firstName*, *lastName*, and *endpointId* with the values that you specified in Step 2.

```
?firstName=Carlos&lastName=Salazar&email=recipient%40example.com&endpointId=12345
```

When you add these attributes to the end of the URL and press Enter, the page updates to include the information in the attribute string. The form should resemble the example in the following image.



Manage your email subscriptions

Contact information

Email address: recipient@example.com

ID: 12345

First name: Carlos

Last name: Salazar

Subscriptions

Special offers
Get exclusive offers available only to subscribers.

Coming soon
Learn about our newest products before anyone else!

New products
Get the inside scoop on products that haven't been released yet.

Deal of the Day
Get special deals in your inbox every day!

Update preferences
Unsubscribe from all email communications
Copyright © 2019, ExampleCorp or its affiliates.

3. On the form, use the toggle switches to opt the endpoint into all of the topics, and then choose **Update preferences**.
4. Wait for approximately one minute, and then proceed to the next section.

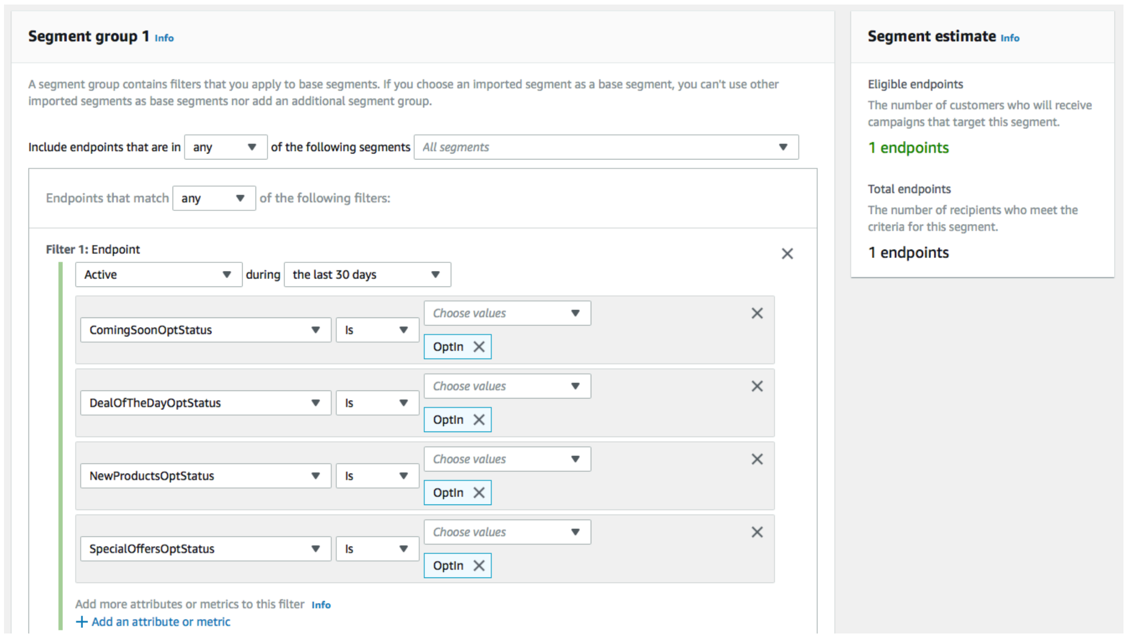
Step 6.4.2: Check the opt status of the test endpoint

Now that you've submitted some test data, you can use the segmentation tool in the Amazon Pinpoint console to make sure that the test data was handled properly.

To check the endpoint opt status

1. Open the Amazon Pinpoint console at https://console.aws.amazon.com/pinpoint/.
2. On the **All projects** page, choose the project that you created in Step 1.1 (p. 49).
3. In the navigation pane, choose **Segments**.
4. Choose **Create a segment**.
5. Under **Segment group 1**, choose **Add a filter**, and then choose **Filter by endpoint**.
6. Choose **Choose an endpoint attribute**, and then choose **ComingSoonOptStatus**. Set the value of the filter to **OptIn**.
7. Choose **Add an attribute or metric**. Add the **DealOfTheDay** attribute to the filter, and set the value to **OptIn**.
8. Repeat the previous step for the two remaining opt topics: **NewProductsOptStatus** and **SpecialOffersOptStatus**.

   When you finish, the **Segment estimate** section should indicate that the number of **Eligible endpoints** and **Total endpoints** are both 1. The page should resemble the example shown in the following image.


Segment group 1 Info

A segment group contains filters that you apply to base segments. If you choose an imported segment as a base segment, you can't use other imported segments as base segments nor add an additional segment group.

Include endpoints that are in any of the following segments All segments

Endpoints that match any of the following filters:

Filter 1: Endpoint

Active during the last 30 days

ComingSoonOptStatus Is OptIn

DealOfTheDayOptStatus Is OptIn

NewProductsOptStatus Is OptIn

SpecialOffersOptStatus Is OptIn

Add more attributes or metrics to this filter Info

+ Add an attribute or metric

Segment estimate Info

Eligible endpoints

The number of customers who will receive campaigns that target this segment.

1 endpoints

Total endpoints

The number of recipients who meet the criteria for this segment.

1 endpoints

Step 6.4.3: Test the "unsubscribe from all" functionality

Also make sure that the **Unsubscribe from all email communications** button on the preferences form works properly. When a customer chooses this button, their opt status for all topics is set to "OptOut". Additionally, the OptOut attribute for the customer's endpoint record is set to "ALL". This change prevents the endpoint from being included in segments that you create in this project in the future.

To test the unsubscribe button

1. In a web browser, go to the location where you uploaded the preferences page.
2. Add the following string to the end of the URL. Replace the values for *firstName*, *lastName*, and *endpointId* with the values that you specified in Step 2.

```
?firstName=Carlos&lastName=Salazar&email=recipient
%40example.com&endpointId=12345&t1=OptIn&t2=OptIn&t3=OptIn&t4=OptIn
```

   When you add these attributes to the end of the URL and press Enter, the page updates to include the information in the attribute string.
3. On the form, choose **Unsubscribe from all email communications**.
4. Wait for approximately one minute, and then proceed to the next section.

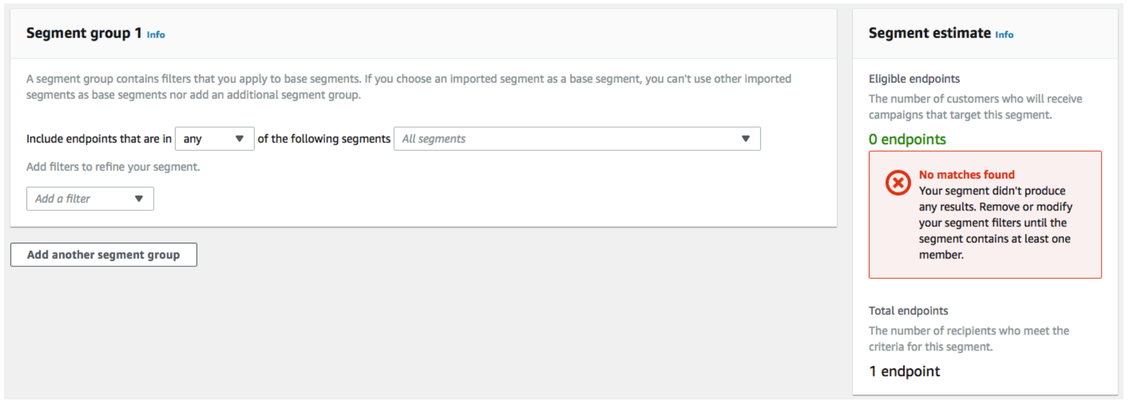
Step 6.3.4: Check the opt status of the test endpoint

The final step in testing the preference form is to confirm that unsubscribe requests are handled properly. In this section, you use the segmentation tool in the Amazon Pinpoint console to ensure that your test endpoint is opted out of email communications that are sent from the current Amazon Pinpoint project.

To check the endpoint opt status

1. Open the Amazon Pinpoint console at https://console.aws.amazon.com/pinpoint/.
2. On the **All projects** page, choose the project that you created in Step 1.1 (p. 49).

3. In the navigation pane, choose **Segments**.
4. Choose **Create a segment**.
5. Note the values in the **Segment estimate** section. This section should indicate that the number of **Eligible endpoints** is 0, and the number of **Total endpoints** is 1. The page should resemble the example shown in the following image.


Segment group 1 Info

A segment group contains filters that you apply to base segments. If you choose an imported segment as a base segment, you can't use other imported segments as base segments nor add an additional segment group.

Include endpoints that are in: any — of the following segments: All segments

Add filters to refine your segment.

Segment estimate Info

Eligible endpoints
The number of customers who will receive campaigns that target this segment.
0 endpoints

No matches found
Your segment didn't produce any results. Remove or modify your segment filters until the segment contains at least one member.

Total endpoints
The number of recipients who meet the criteria for this segment.
1 endpoint

Troubleshooting the form

If you perform these test steps, but the data in your Amazon Pinpoint project doesn't change, there are a few steps that you can take to troubleshoot the issue:

- In the Amazon Pinpoint console, make sure that you're using the correct AWS Region. Segments and endpoints aren't shared between Regions.
- In the Lambda console, open the function that you created in Step 4 (p. 58). On the **Monitoring** tab, choose **View logs in CloudWatch**.

  Under **Log Streams**, choose the most recently logged event. Check the output of the event for more information about the issue. For example, if you see an "AccessDeniedException" error in the logs, make sure that the Amazon Pinpoint project ID and AWS Region that you entered in Step 4 (p. 58) are equal to the project ID and Region values that you specified in the IAM policy in Step 3.1 (p. 56). If you recently changed these values, wait for a few minutes, and then try again.
- If CloudWatch doesn't contain any logged information for the function, make sure that CORS is enabled in API Gateway, and that the API is deployed. If you're unsure, repeat the steps in Step 5.3 (p. 65).

Step 7: Create and send Amazon Pinpoint campaigns

The email preference management solution is now set up and ready to use. Now you can start sending campaign emails. This section shows you how to send campaign emails that contain a special link that recipients can use to manage their subscription preferences.

Step 7.1: Create a segment

To send a campaign email, you first have to create the segment that you want to send the campaign to. For the purpose of this tutorial, you should use a segment that only contains your own endpoints, or those of internal recipients within your organization. After you confirm that the solution works as you expect it to work, you can start sending messages to external recipients.

To create a segment, repeat the procedures in Step 2 (p. 53) to import a segment of internal recipients.

Step 7.2: Create the campaign

After you create a segment of recipients, you can create a campaign that targets the segment.

To create the campaign

1. Open the Amazon Pinpoint console at https://console.aws.amazon.com/pinpoint/.
2. On the **All projects** page, choose the project that you created in Step 1.1 (p. 49).
3. In the navigation pane, choose **Campaigns**.
4. Choose **Create a campaign**.
5. On the **Create a campaign** page, do the following:
   - For **Campaign name**, enter a name for the campaign.
   - For **Campaign type**, choose **Standard campaign**.
   - Choose **Next**.
6. On the **Choose a segment** page, do the following:
   - Choose **Use an existing segment**.
   - Under **Segment details**, for **Segment**, choose the segment that you created in Step 7.1 (p. 76).
   - Choose **Next**.
7. On the **Create a message** page, do the following:
   - Under **Specifications**, for **Choose a channel for this campaign**, choose **Email**.
   - Under **Email details**, for **Sender email address**, enter the email address that you want to send the email from. The email address that you specify must be verified.
   - Under **Message content**, choose **Create a new email**.
   - For **Subject**, enter the subject line for the email.
   - For **Message**, enter the message that you want to send. In the body of the email, include an "Unsubscribe" or "Change your email preferences" link. Use the following URL as the destination for the link:

```
https://www.example.com/prefs.html
  ?firstName={{User.UserAttributes.FirstName}}
  &lastName={{User.UserAttributes.LastName}}
  &email={{Attributes.Email}}
  &endpointId={{Attributes.EndpointId}}
  &t1={{Attributes.SpecialOffersOptStatus}}
  &t2={{Attributes.NewProductsOptStatus}}
  &t3={{Attributes.ComingSoonOptStatus}}
  &t4={{Attributes.DealOfTheDayOptStatus}}
```

     In the preceding example, replace *`https://www.example.com/prefs.html`* with the location where you uploaded the form in Step 6.3 (p. 72).

     Important
     The URL in the preceding example includes line breaks and spaces in order to make it easier to read. Remove all of the line breaks and spaces from this example before you paste it into the email editor.
   - Choose **Next**.
8. On the **Choose when to send the campaign** page, do the following:
   - For **Choose when the campaign should be sent**, choose **At a specific time**.

  - If you want to send the message as soon as you finish creating the campaign, choose **Immediately**. If you want to send the message at a specific time, choose **Once**, and then specify the date and time when the message should be sent.
  - Choose **Next**.

9. On the **Review and launch** page, confirm the settings for the campaign. If the campaign settings are correct, choose **Launch campaign**.
10. When you receive the campaign email, choose the "Unsubscribe" or "Manage your email preferences" link that you specified in the message. Confirm that the form loads properly, and that it's filled in with the appropriate values for **Email address**, **ID**, **First name**, and **Last name**. Also, make sure that the selections in the **Subscriptions** section correspond with the opt-in values that you specified when you created the endpoint.

Step 7.3: Create production segments

After you complete the procedures in the preceding sections and confirmed that the preference page is working as expected, you're ready to start creating segments of customers who've opted into your various topics.

To create production segments for your topics

1. Open the Amazon Pinpoint console at https://console.aws.amazon.com/pinpoint/.
2. On the **All projects** page, choose the project that you created in Step 1.1 (p. 49).
3. In the navigation pane, choose **Segments**.
4. On the **Create a segment** page, choose **Build a segment**.
5. Under **Specifications**, for **Name**, enter a name for the segment.
6. In **Segment group 1**, add a **Filter by endpoint**.
7. For **Choose an endpoint attribute**, choose **ComingSoonOptStatus**. Then, for **Choose values**, choose **OptIn**.
8. Choose **Create segment**.
9. Repeat steps 4–8 for each of the remaining opt topics (**DealOfTheDayOptStatus**, **NewProductsOptStatus**, and **SpecialOffersOptStatus**). When you complete this step, you have a separate segment for each of your opt topics. When you want to send an email campaign to customers who subscribe to a specific topic, choose the appropriate segment when you create the campaign.
10. (Optional) Create additional segments that further refine your audience. When you create additional segments, use the **Include endpoints that are in any of the following segments** menu to choose the appropriate base segment of opted-in endpoints.

    To learn more about creating segments, see Customizing segments with AWS Lambda (p. 154).

Next steps

By completing this tutorial, you've done the following:

- Created an Amazon Pinpoint project, enabled the email channel, and verified an identity for sending email.
- Created a IAM policy that uses the principal of least privilege to grant access rights, and have associated that policy with a role.
- Created a Lambda function that uses the UpdateEndpoint operation in the Amazon Pinpoint API.
- Created a REST API using API Gateway.
- Created and deployed a web-based form that collects email recipients' subscription preferences.

- Created an email campaign that contains a link to the web form that is personalized for each recipient.
- Created segments that contain endpoints that are opted in to your topics.
- Performed tests on the solution to make sure it works as expected.

This section discusses a few ways that you can use the information that you collect by using this solution. It also includes some suggestions of ways that you can customize this solution to fit your unique use case.

Use the form to collect additional information

You can modify this solution to collect additional information on the registration form. For example, you could ask the customer to provide their address, and then use the address data to populate the `Location.City`, `Location.Country`, `Location.Region`, and `Location.PostalCode` fields in the `Endpoint` resource. If you collect this data, you can use it to create targeted segments.

For example, if you offer different products to customers based on their country, you could create a segment of users who are opted in to a topic, and who are located in a specific country. After that, you could send a campaign to that segment that contains products that are tailored to that country or region. To make this change, you need to add the appropriate fields to the web form. You also have to modify the JavaScript code in the form handler to pass the new values. Finally, you have to modify the Lambda function that updates the endpoint to handle the new incoming information.

You can also modify the form so that it collects contact information in other channels. For example, you could use the form to collect the customers' phone numbers so that you can send them SMS alerts. To make this change, you need to modify the HTML for the web form, and the JavaScript code in the form handler. You also have to modify the Lambda function so that it creates or updates two separate endpoints (one for the email endpoint, and one for the SMS endpoint). Finally, you should modify the Lambda function so that it generates a unique value for the `User.UserId` attribute, and then associates that value with both endpoints.

Record additional attributes for auditing purposes

This solution records two valuable attributes when it creates and updates endpoints. First, when the first Lambda function updates the endpoint, it records the location of the form in the `Attributes.OptSource` attribute. When a customer submits the form, the Lambda function creates or updates the `Attributes.OptStatusLastChanged` attribute. This attribute contains the exact date and time when the customer last updated the opt status for all of your topics.

Both of these fields can be useful if you're ever asked an email provider or regulatory agency to provide evidence of a customer's consent. You can retrieve this information at any time by using the GetEndpoint API operation.

You can also modify the Lambda functions to record additional data that might be useful for auditing purposes, such as the IP address that the registration request was submitted from.

Collect new customer information

The solution in this tutorial uses the UpdateEndpoint API operation, which creates new endpoints and also updates existing ones. You can use the web form to capture information about new customers. If you do, you have to generate a unique endpoint ID for each customer. When a new endpoint enters your system in this way, you should set the OptOut value for it to "ALL". Next, you should send a message to the endpoint that asks the customer to click a link to confirm their subscription. When the customer confirms their subscription, you can change the OptOut value to "NONE".

This practice is called "double opt-in." By using a double opt-in system, you can prevent malicious users from registering endpoints that aren't their own. Implementing this kind of system helps to protect your reputation as a sender. Additionally, capturing this type of explicit opt information is required in several countries and regions around the world.

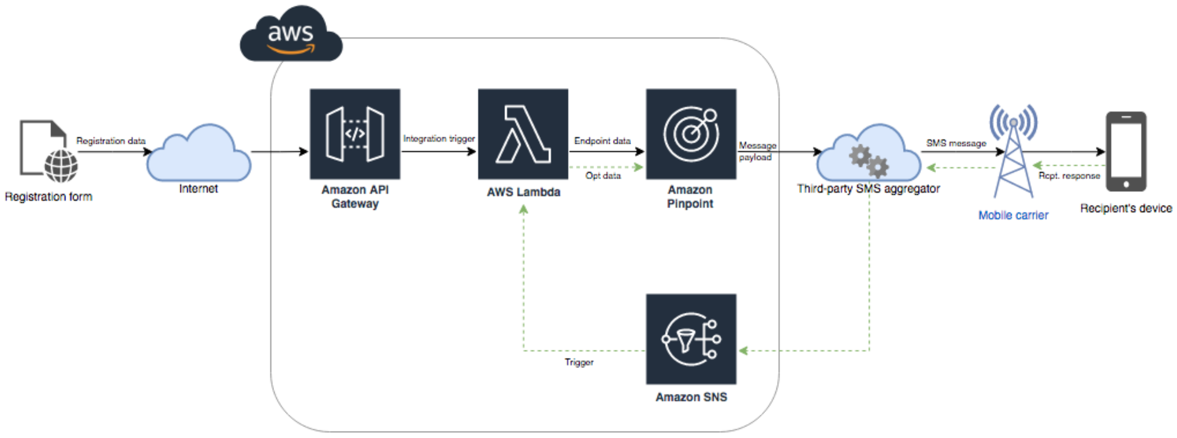
Tutorial: Setting up an SMS registration system

SMS messages (text messages) are a great way to send time-sensitive messages to your customers. These days, many people keep their phones nearby at all times. Also, SMS messages tend to capture people's attention more than push notifications, emails, or phone calls.

A common way to capture customers' mobile phone numbers is to use a web-based form. After you verify the customer's phone number and confirm their subscription, you can start sending promotional, transactional, and informational SMS messages to that customer.

This tutorial shows you how to set up a web form to capture customers' contact information. The web form sends this information to Amazon Pinpoint. Next, Amazon Pinpoint verifies that the phone number is valid, and captures other metadata that's related to the phone number. After that, Amazon Pinpoint sends the customer a message asking them to confirm their subscription. After the customer confirms their subscription, Amazon Pinpoint opts them in to receiving your messages.

The following architecture diagram shows the flow of data in this solution.



About double opt-in

This tutorial shows you how to set up a double opt-in system in Amazon Pinpoint that uses two-way SMS messaging.

In an SMS double opt-in system, a customer provides you with their phone number by submitting it in a web form or within your app. When you receive the request from the customer, you create a new endpoint in Amazon Pinpoint. The new endpoint should be opted out of your communications. Next, you send a message to that phone number. In your message, you ask the recipient to confirm their subscription by replying with a specific word or phrase (such as "Yes" or "Confirm"). If the customer responds to the message with the word or phrase that you specified, you change the endpoint's status to opted-in. Otherwise, if the customer doesn't respond or they respond with a different word or phrase, you can leave the endpoint with a status of opted-out.

About this solution

This section contains information about the solution that you're building in this tutorial.

Intended Audience

This tutorial is intended for developer and system implementer audiences. You don't have to be familiar with Amazon Pinpoint to complete the steps in this tutorial. However, you should be comfortable managing IAM policies, creating Lambda functions in Node.js, and deploying web content.

Features Used

This tutorial includes usage examples for the following Amazon Pinpoint features:

- Sending transactional SMS messages
- Obtaining information about phone numbers by using phone number validation
- Receiving incoming SMS messages by using two-way SMS messaging
- Creating dynamic segments
- Creating campaigns
- Interacting with the Amazon Pinpoint API by using AWS Lambda

Time Required

It should take about one hour to complete this tutorial. After you implement this solution, there are additional steps that you can take to refine the solution to suit your unique use case.

Regional Restrictions

This tutorial requires you to lease a long code by using the Amazon Pinpoint console. You can use the Amazon Pinpoint console to lease dedicated long codes that are based in several countries. However, only long codes that are based in the United States or Canada can be used to send SMS messages. (You can use long codes that are based in other countries and regions to send voice messages.)

We developed the code examples in this tutorial with this restriction in mind. For example, the code examples assume that the recipient's phone number always has 10 digits, and a country code of 1. If you implement this solution in countries or regions other than the United States or Canada, you have to modify the code examples appropriately.

Resource Usage Costs

There's no charge for creating an AWS account. However, by implementing this solution, you might incur the following costs:

- **Long code lease costs** – To complete this tutorial, you have to lease a long code. Long codes that are based in the United States (excluding US Territories) and Canada cost $1.00 per month.
- **Phone number validation usage** – The solution in this tutorial uses the phone number validation feature of Amazon Pinpoint to verify that each number you receive is valid and properly formatted, and to obtain additional information about the phone number. You pay $0.006 for each phone number validation request.
- **Message sending costs** – The solution in this tutorial sends outbound SMS messages. You pay for each message that you send through Amazon Pinpoint. The price that you pay for each message depends on the country or region of the recipient. If you send messages to recipients in the United States (excluding US Territories), you pay $0.00645 per message. If you send messages to recipients in Canada, you pay between $0.00109–$0.02, depending on the recipient's carrier and location.
- **Message receiving costs** – This solution also receives and processes incoming SMS messages. You pay for each incoming message that's sent to phone numbers that are associated with your Amazon Pinpoint account. The price that you pay depends on where the receiving phone number is based. If your receiving number is based in the United States (excluding US Territories), you pay $0.0075 per incoming message. If your number is based in Canada, you pay $0.00155 per incoming message.
- **Lambda usage** – This solution uses two Lambda functions that interact with the Amazon Pinpoint API. When you call a Lambda function, you're charged based on the number of requests for your functions, for the time that it takes for your code to execute, and for the amount of memory that your functions use. The functions in this tutorial use very little memory, and typically run for 1–3 seconds. Some or all of your usage of this solution might fall under the Lambda free usage tier. For more information, see Lambda pricing.
- **API Gateway usage** – The web form in this solution calls an API that's managed by API Gateway. For every million calls to API Gateway, you pay $3.50–$3.70, depending on which AWS Region you use Amazon Pinpoint in. For more information, see API Gateway pricing.

- **Web hosting costs** – This solution includes a web-based form that you have to host on your website. The price that you pay for hosting this content depends on your web hosting provider.

> **Note**
> All prices shown in this list are in US Dollars (USD).

Next: Prerequisites (p. 82)

Prerequisites

Before you begin this tutorial, you have to complete the following prerequisites:

- You have to have an AWS account. To create an AWS account, go to https://console.aws.amazon.com/ and choose **Create a new AWS account**.
- The account that you use to sign in to the AWSManagement Console has to be able to perform the following tasks:
  - Create new IAM policies and roles
  - Create new Amazon Pinpoint projects
  - Create new Lambda functions
  - Create new APIs in API Gateway
- You have to have a method of hosting webpages, and you should know how to publish webpages. Although you can use AWS services to host your webpages, you aren't required to.

  > **Tip**
  > To learn more about hosting webpages using AWS services, see Host a static webpage.

Next: Set Up Amazon Pinpoint (p. 82)

Step 1: Set up Amazon Pinpoint

The first step in implementing this solution is to set up Amazon Pinpoint. In this section, you do the following:

- Create an Amazon Pinpoint project
- Enable the SMS channel and lease a long code
- Configure two-way SMS messaging

Before you begin with this tutorial, you should review the prerequisites (p. 82).

Step 1.1: Create an Amazon Pinpoint project

To get started, you need to create an Amazon Pinpoint project. In Amazon Pinpoint, a *project* consists of segments, campaigns, configurations, and data that are united by a common purpose. For example, you could use a project to contain all of the content that's related to a particular app, or to a specific brand or marketing initiative. When you add customer information to Amazon Pinpoint, that information is associated with a project.

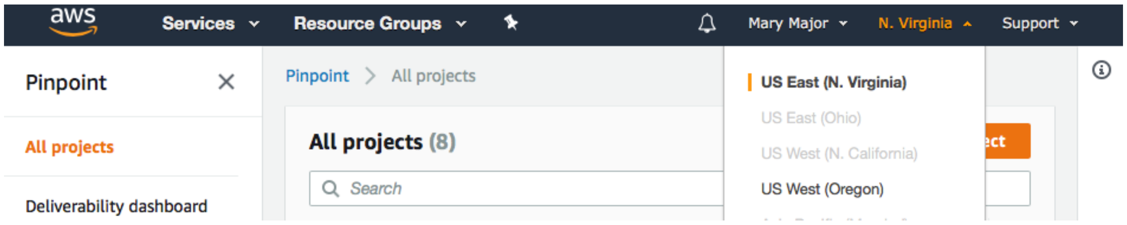
The steps involved in creating a new project differ depending on whether you've created a project in Amazon Pinpoint previously.

Creating a project (new Amazon Pinpoint users)

These steps describe the process of creating a new Amazon Pinpoint project if you've never created a project in the current AWS Region.

To create a project

1. Sign in to the AWSManagement Console and open the Amazon Pinpoint console at https://console.aws.amazon.com/pinpoint/.
2. Use the Region selector to choose the AWS Region that you want to use, as shown in the following image. If you're unsure, choose the Region that's located closest to you.



3. Under **Get started**, for **Name**, enter a name for the campaign (such as `SMSRegistration`), and then choose **Create project**.
4. On the **Configure features** page, choose **Skip this step**.
5. In the navigation pane, choose **All projects**.
6. On the **All projects** page, next to the project you just created, copy the value that's shown in the **Project ID** column.

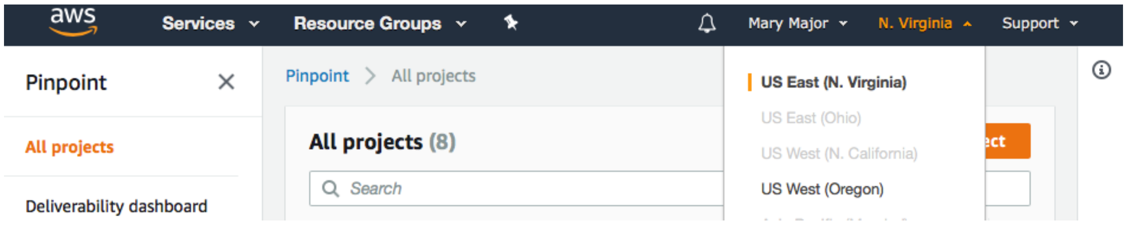
   Tip
   You need to use this ID in a few different places in this tutorial. Keep the project ID in a convenient place so that you can copy it later.

Creating a project (existing Amazon Pinpoint users)

These steps describe the process of creating a new Amazon Pinpoint project if you've already created projects in the current AWS Region.

To create a project

1. Sign in to the AWSManagement Console and open the Amazon Pinpoint console at https://console.aws.amazon.com/pinpoint/.
2. Use the Region selector to choose the AWS Region that you want to use, as shown in the following image. If you're unsure, choose the Region that's located closest to you.



3. On the **All projects** page, choose **Create a project**.
4. On the **Create a project** window, for **Project name**, enter a name for the project (such as `SMSRegistration`). Choose **Create**.
5. On the **Configure features** page, choose **Skip this step**.
6. In the navigation pane, choose **All projects**.
7. On the **All projects** page, next to the project you just created, copy the value that's shown in the **Project ID** column.

   Tip
   You need to use this ID in a few different places in this tutorial. Keep the project ID in a convenient place so that you can copy it later.

Step 1.2: Obtain a dedicated long code

After you create a project, you can start to configure features within that project. In this section, you enable the SMS channel, and obtain a dedicated long code to use when sending SMS messages.

> **Note**
> This section assumes that you're leasing a long code that's based in the United States or Canada. If you follow the procedures in this section, but choose a country other than the United States or Canada, you won't be able to use that number to send SMS messages. To learn more about leasing SMS-capable long codes in countries other than the United States or Canada, see Requesting dedicated long codes for SMS messaging with Amazon Pinpoint in the *Amazon Pinpoint User Guide*.

1. In the navigation pane, under **Settings**, choose **SMS and voice**.
2. Next to **SMS settings**, choose **Edit**.
3. Under **General settings**, choose **Enable the SMS channel for this project**, and then choose **Save changes**.
4. Next to **Number settings**, choose **Request long codes**.
5. Under **Long code specifications**, do the following:
    - For **Target country or region**, choose **United States** or **Canada**.
    - For **Default call type**, choose **Transactional**.
    - For **Quantity**, choose **1**.
6. Choose **Request long code**.

Step 1.3: Enable two-way SMS

Now that you have a dedicated phone number, you can set up two-way SMS. Enabling two-way SMS makes it possible for your customers to respond to the SMS messages that you send them. In this solution, you use two-way SMS to give your customers a way to confirm that they want to subscribe to your SMS program.

1. On the **SMS and voice** settings page, under **Number settings**, choose the long code that you received in the previous section.
2. Under **Required keywords**, do the following:
    - Next to the **HELP** keyword, for **Response message**, enter the message that you want Amazon Pinpoint to automatically send recipients when they send the keyword "HELP" (or its variations) to this long code.
    - Next to the **STOP** keyword, for **Response message**, enter the message that you want Amazon Pinpoint to automatically send recipients when they send the keyword "STOP" (or its variations) to this long code.

        > **Note**
        > When a recipient sends the keyword "STOP" to one of your long codes, Amazon Pinpoint automatically opts that recipient out of all future SMS messages that are sent from this project.
3. Under **Registered keyword**, for **Keyword**, enter the word that customers can send you to register to receive messages from you. Then, for **Response message**, enter the message that Amazon Pinpoint automatically sends when a customer sends the keyword to this long code.

    > **Note**
    > For the purposes of this tutorial, the value that you enter in this section isn't important. In this scenario, customers register for your SMS messaging program by completing a registration form, rather than by sending you a message directly.

4. Under **Two-Way SMS**, choose **Enable two-way SMS**.
5. Under **Incoming message destination**, choose **Create a new SNS topic**. Enter an Amazon SNS topic name, such as **`SMSRegistrationFormTopic`**.
6. Under **Two-way SMS keywords**, for **Keyword**, enter the word that customers send you to confirm their subscriptions (such as **`Yes`** or **`Confirm`**).

   Note
   This value isn't case sensitive.
7. For **Response message**, enter the message that Amazon Pinpoint automatically sends to customers when they send you the keyword that you specified in the previous step.
8. Choose **Save**.

Next: Create IAM Policies and Roles (p. 85)

Step 2: Create IAM policies and roles

The next step in implementing the SMS registration solution is to configure a policy and a role in AWS Identity and Access Management (IAM). For this solution, you need to create a policy that provides access to certain resources that are related to Amazon Pinpoint. You then create a role and attach the policy to it. Later in this tutorial, you create an AWS Lambda function that uses this role to call certain operations in the Amazon Pinpoint API.

Step 2.1: Create an IAM policy

This section shows you how to create an IAM policy. Users and roles that use this policy are able to do the following:

- Use the Phone Number Validate feature
- View, create, and update Amazon Pinpoint endpoints
- Send messages to Amazon Pinpoint endpoints

In this tutorial, you want to give Lambda the ability to perform these tasks. However, for added security, this policy uses the principal of granting *least privilege*. In other words, it grants only the permissions that are required to complete this solution, and no more. This policy is restricted in the following ways:

- You can only use it to call the Phone Number Validate API in a specific Region.
- You can only use it to view, create, or update endpoints that are associated with a specific Amazon Pinpoint project.
- You can only use it to send messages to endpoints that are associated with a specific Amazon Pinpoint project.

To create the policy

1. Sign in to the AWSManagement Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies**, and then choose **Create policy**.
3. On the **JSON** tab, paste the following code.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:CreateLogGroup"
            ],
            "Resource": "arn:aws:logs:*:*:*"
        },
        {
            "Effect": "Allow",
            "Action": "mobiletargeting:SendMessages",
            "Resource": "arn:aws:mobiletargeting:region:accountId:apps/projectId/*"
        },
        {
            "Effect": "Allow",
            "Action": [
              "mobiletargeting:GetEndpoint",
              "mobiletargeting:UpdateEndpoint",
              "mobiletargeting:PutEvents"
            ],
            "Resource": "arn:aws:mobiletargeting:region:accountId:apps/projectId/
endpoints/*"
        },
        {
          "Effect": "Allow",
          "Action": "mobiletargeting:PhoneNumberValidate",
          "Resource": "arn:aws:mobiletargeting:region:accountId:phone/number/validate"
        }
    ]
}
```

In the preceding example, do the following:

- Replace *`region`* with the AWS Region that you use Amazon Pinpoint in, such as `us-east-1` or `eu-central-1`.

  Tip
  For a complete list of AWS Regions where Amazon Pinpoint is available, see AWS regions and endpoints in the *AWS General Reference*.
- Replace *`accountId`* with the unique ID for your AWS account.
- Replace *`projectId`* with the unique ID of the project that you created in Step 1.1 (p. 82) of this tutorial.

  Note
  The `logs` actions enable Lambda to log its output in CloudWatch Logs.

4. Choose **Review policy**.
5. For **Name**, enter a name for the policy, such as **`RegistrationFormPolicy`**. Choose **Create policy**.

Step 2.2: Create an IAM role

To create the role

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the IAM console, in the navigation pane, choose **Roles**, and then choose **Create role**.
3. Under **Choose the service that will use this role**, choose **Lambda**, and then choose **Next: Permissions**.

   Note
   The service that you choose in this step isn't important—regardless of the service that you choose, you apply your own policy in the next step.

4. Under **Attach permissions policies**, choose the policy that you created in the previous section, and then choose **Next: Tags**.
5. Choose **Next: Review**.
6. Under **Review**, for **Name**, enter a name for the role, such as **`SMSRegistrationForm`**. Choose **Create role**.

Next: Create Lambda functions (p. 87)

Step 3: Create Lambda functions

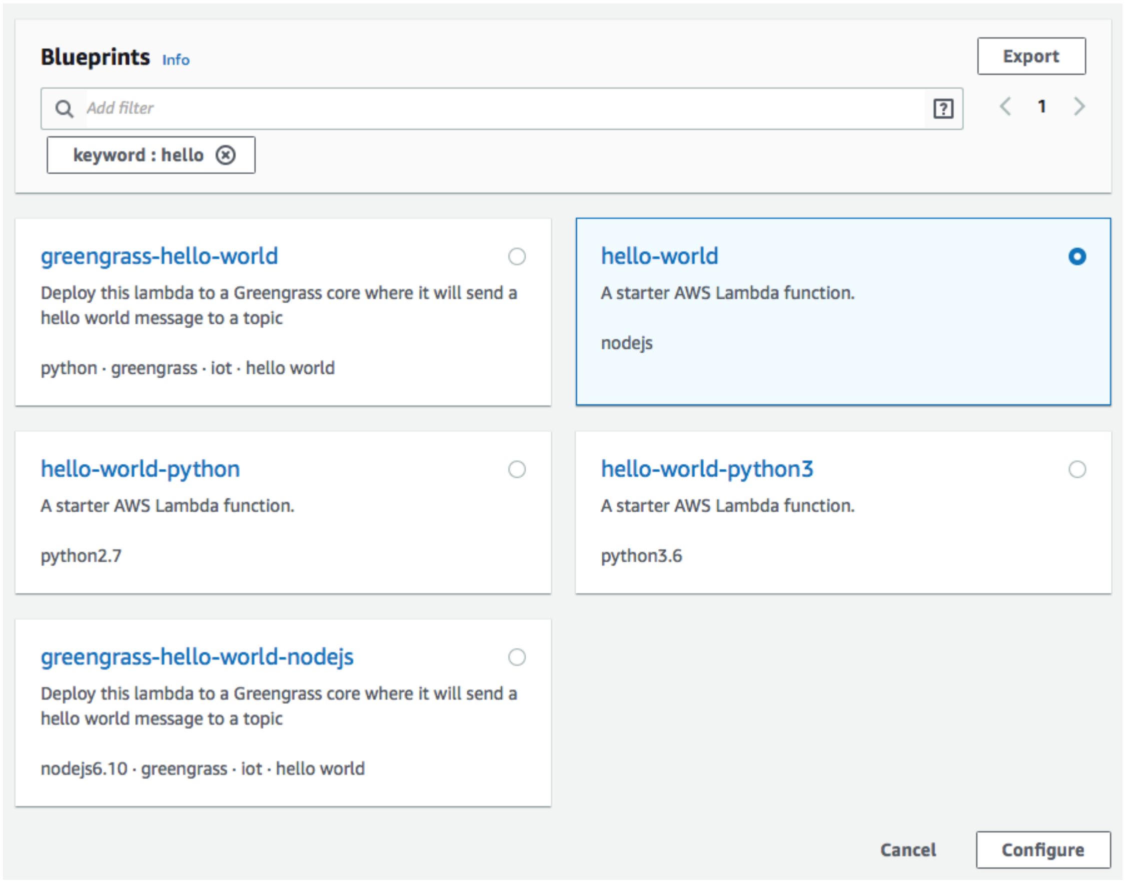
This solution uses two Lambda functions. This section shows you how to create and configure these functions. Later, you set up API Gateway and Amazon Pinpoint to execute these functions when certain events occur. Both of these functions create and update endpoints in the Amazon Pinpoint project that you specify. The first function also uses the phone number validation feature.

Step 3.1: Create the function that validates customer information and creates endpoints

The first function takes input from your registration form, which it receives from Amazon API Gateway. It uses this information to obtain information about the customer's phone number by using the phone number validation (p. 164) feature of Amazon Pinpoint. The function then uses the validated data to create a new endpoint in the Amazon Pinpoint project that you specify. By default, the endpoint that the function creates is opted out of future communications from you, but this status can be changed by the second function. Finally, this function sends the customer a message asking them to verify that they want to receive SMS communications from you.

To create the Lambda function

1. Open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
2. Choose **Create function**.
3. Under **Create a function**, choose **Blueprints**.
4. In the search field, enter **`hello`**, and then press Enter. In the list of results, choose the `hello-world` Node.js function, as shown in the following image. Choose **Configure**.



greengrass-hello-world
Deploy this lambda to a Greengrass core where it will send a hello world message to a topic
python · greengrass · iot · hello world

hello-world
A starter AWS Lambda function.
nodejs

hello-world-python
A starter AWS Lambda function.
python2.7

hello-world-python3
A starter AWS Lambda function.
python3.6

greengrass-hello-world-nodejs
Deploy this lambda to a Greengrass core where it will send a hello world message to a topic
nodejs6.10 · greengrass · iot · hello world

5. Under **Basic information**, do the following:

   - For **Name**, enter a name for the function, such as **`RegistrationForm`**.
   - For **Role**, select **Choose an existing role**.
   - For **Existing role**, choose the **SMSRegistrationForm** role that you created in Step 2.2 (p. 86).

   When you finish, choose **Create function**.

6. Delete the sample function in the code editor, and then paste the following code:

```
var AWS = require('aws-sdk');
var pinpoint = new AWS.Pinpoint({region: process.env.region});

// Make sure the SMS channel is enabled for the projectId that you specify.
// See: https://docs.aws.amazon.com/pinpoint/latest/userguide/channels-sms-setup.html
var projectId = process.env.projectId;

// You need a dedicated long code in order to use two-way SMS.
// See: https://docs.aws.amazon.com/pinpoint/latest/userguide/channels-voice-
manage.html#channels-voice-manage-request-phone-numbers
var originationNumber = process.env.originationNumber;

// This message is spread across multiple lines for improved readability.
var message = "ExampleCorp: Reply YES to confirm your subscription. 2 msgs per "
            + "month. No purchase req'd. Msg&data rates may apply. Terms: "
            + "example.com/terms-sms";

var messageType = "TRANSACTIONAL";
```

```
exports.handler = (event, context, callback) => {
  console.log('Received event:', event);
  validateNumber(event);
};

function validateNumber (event) {
  var destinationNumber = event.destinationNumber;
  if (destinationNumber.length == 10) {
    destinationNumber = "+1" + destinationNumber;
  }
  var params = {
    NumberValidateRequest: {
      IsoCountryCode: 'US',
      PhoneNumber: destinationNumber
    }
  };
  pinpoint.phoneNumberValidate(params, function(err, data) {
    if (err) {
      console.log(err, err.stack);
    }
    else {
      console.log(data);
      //return data;
      if (data['NumberValidateResponse']['PhoneTypeCode'] == 0) {
        createEndpoint(data, event.firstName, event.lastName, event.source);
      } else {
        console.log("Received a phone number that isn't capable of receiving "
                   +"SMS messages. No endpoint created.");
      }
    }
  });
}

function createEndpoint(data, firstName, lastName, source) {
  var destinationNumber = data['NumberValidateResponse']['CleansedPhoneNumberE164'];
  var endpointId = data['NumberValidateResponse']
['CleansedPhoneNumberE164'].substring(1);

  var params = {
    ApplicationId: projectId,
    // The Endpoint ID is equal to the cleansed phone number minus the leading
    // plus sign. This makes it easier to easily update the endpoint later.
    EndpointId: endpointId,
    EndpointRequest: {
      ChannelType: 'SMS',
      Address: destinationNumber,
      // OptOut is set to ALL (that is, endpoint is opted out of all messages)
      // because the recipient hasn't confirmed their subscription at this
      // point. When they confirm, a different Lambda function changes this
      // value to NONE (not opted out).
      OptOut: 'ALL',
      Location: {
        PostalCode:data['NumberValidateResponse']['ZipCode'],
        City:data['NumberValidateResponse']['City'],
        Country:data['NumberValidateResponse']['CountryCodeIso2'],
      },
      Demographic: {
        Timezone:data['NumberValidateResponse']['Timezone']
      },
      Attributes: {
        Source: [
          source
        ]
      },
      User: {
        UserAttributes: {
          FirstName: [
            firstName
          ],
          LastName: [
            lastName
          ]
        }
      }
    }
  };
  pinpoint.updateEndpoint(params, function(err,data) {
    if (err) {
      console.log(err, err.stack);
    }
    else {
      console.log(data);
      //return data;
      sendConfirmation(destinationNumber);
    }
  });
}

function sendConfirmation(destinationNumber) {
  var params = {
    ApplicationId: projectId,
    MessageRequest: {
      Addresses: {
        [destinationNumber]: {
          ChannelType: 'SMS'
        }
      },
      MessageConfiguration: {
        SMSMessage: {
          Body: message,
          MessageType: messageType,
          OriginationNumber: originationNumber
        }
      }
    }
  };

  pinpoint.sendMessages(params, function(err, data) {
    // If something goes wrong, print an error message.
    if(err) {
      console.log(err.message);
    // Otherwise, show the unique ID for the message.
    } else {
      console.log("Message sent! "
          + data['MessageResponse']['Result'][destinationNumber]['StatusMessage']);
    }
  });
}
```

7. Under **Environment variables**, do the following:

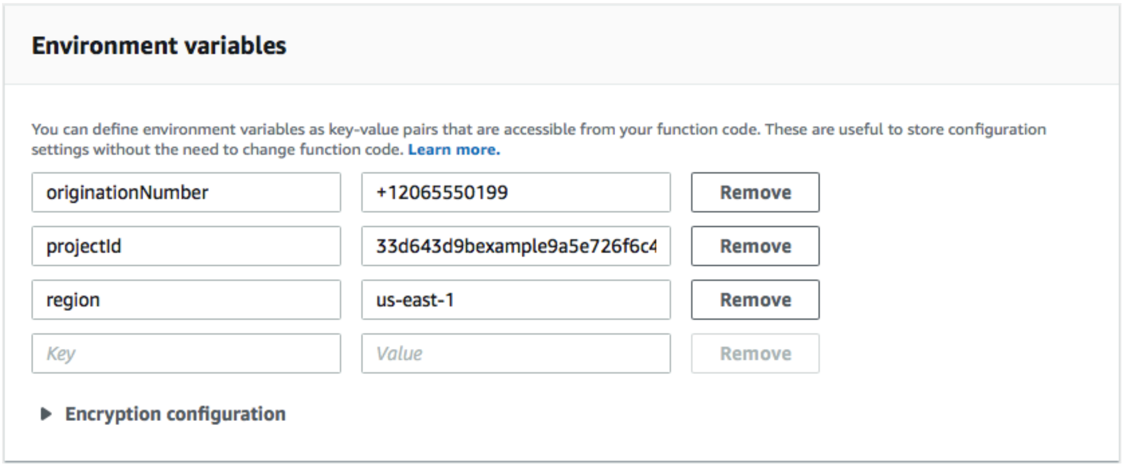
   - In the first row, create a variable with a key of **`originationNumber`**. Next, set the value to the phone number of the dedicated long code that you received in Step 1.2 (p. 84).

     Note
     Be sure to include the plus sign (+) and the country code for the phone number. Don't include any other special characters, such as dashes (-), periods (.), or parentheses.

   - In the second row, create a variable with a key of **`projectId`**. Next, set the value to the unique ID of the project that you created in Step 1.1 (p. 82).

- In the third row, create a variable with a key of **`region`**. Next, set the value to the Region that you use Amazon Pinpoint in, such as **`us-east-1`** or **`us-west-2`**.

When you finish, the **Environment Variables** section should resemble the example shown in the following image.



Environment variables

You can define environment variables as key-value pairs that are accessible from your function code. These are useful to store configuration settings without the need to change function code. Learn more.

originationNumber	+12065550199	Remove
projectId	33d643d9bexample9a5e726f6c4	Remove
region	us-east-1	Remove
Key	Value	Remove

▶ Encryption configuration

8. At the top of the page, choose **Save**.

Step 3.1.1: Test the function

After you create the function, you should test it to make sure that it's configured properly. Also, make sure that the IAM role you created has the appropriate permissions.

To test the function

1. Choose **Test**.
2. On the **Configure test event** window, do the following:
   - Choose **Create new test event**.
   - For **Event name**, enter a name for the test event, such as **`MyPhoneNumber`**.
   - Erase the example code in the code editor. Paste the following code:

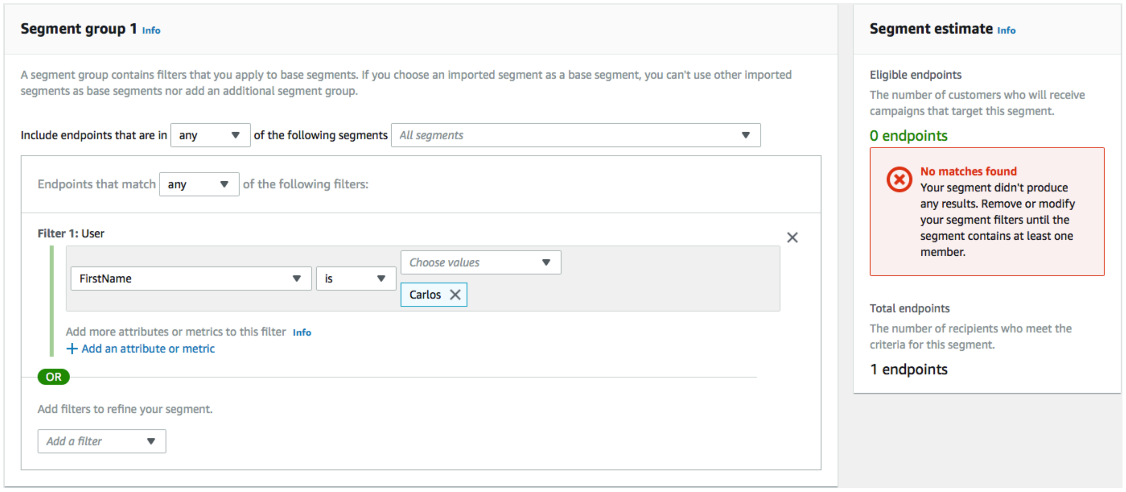
```
{
  "destinationNumber": "2065550142",
  "firstName": "Carlos",
  "lastName": "Salazar",
  "source": "Registration form test"
}
```

   - In the preceding code example, replace the values of the `destinationNumber`, `firstName`, and `lastName` attributes with the values that you want to use for testing, such as your personal contact details. When you test this function, it sends an SMS message to the phone number that you specify in the `destinationNumber` attribute. Make sure that the phone number that you specify is able to receive SMS messages.
   - Choose **Create**.
3. Choose **Test** again.
4. Under **Execution result: succeeded**, choose **Details**. In the **Log output** section, review the output of the function. Make sure that the function ran without errors.

Check the device that's associated with the `destinationNumber` that you specified to make sure that it received the test message.

5. Open the Amazon Pinpoint console at https://console.aws.amazon.com/pinpoint/.
6. On the **All projects** page, choose the project that you created in Step 1.1 (p. 82).
7. In the navigation pane, choose **Segments**. On the **Segments page**, choose **Create a segment**.
8. In **Segment group 1**, under **Add filters to refine your segment**, choose **Filter by user**.
9. For **Choose a user attribute**, choose **FirstName**. Then, for **Choose values**, choose the first name that you specified in the test event.

   The **Segment estimate** section should show that there are zero eligible endpoints, and one total endpoint, as shown in the following image. This result is expected. When the function creates a new endpoint, the endpoint is opted out. Segments in Amazon Pinpoint automatically exclude opted-out endpoints.



Segment group 1 Info

A segment group contains filters that you apply to base segments. If you choose an imported segment as a base segment, you can't use other imported segments as base segments nor add an additional segment group.

Include endpoints that are in any of the following segments All segments

Endpoints that match any of the following filters:

Filter 1: User

FirstName is Choose values Carlos

Add more attributes or metrics to this filter Info

+ Add an attribute or metric

OR

Add filters to refine your segment.

Add a filter

Segment estimate Info

Eligible endpoints

The number of customers who will receive campaigns that target this segment.

0 endpoints

No matches found

Your segment didn't produce any results. Remove or modify your segment filters until the segment contains at least one member.

Total endpoints

The number of recipients who meet the criteria for this segment.

1 endpoints

Step 3.2: Create the function that opts in customers to your communications

The second function is only executed when a customer replies to the message that's sent by the first function. If the customer's reply includes the keyword that you specified in Step 1.3 (p. 84), the function updates their endpoint record to opt them in to future communications. Amazon Pinpoint also automatically responds with the message that you specified in Step 1.3.

If the customer doesn't respond, or responds with anything other than the designated keyword, then nothing happens. The customer's endpoint remains in Amazon Pinpoint, but it can't be targeted by segments.

To create the Lambda function

1. Open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
2. Choose **Create function**.
3. Under **Create function**, choose **Blueprints**.
4. In the search field, enter **`hello`**, and then press Enter. In the list of results, choose the `hello-world` Node.js function, as shown in the following image. Choose **Configure**.
5. Under **Basic information**, do the following:

   - For **Name**, enter a name for the function, such as **`RegistrationForm_OptIn`**.

- For **Role**, select **Choose an existing role**.
- For **Existing role**, choose the SMSRegistrationForm role that you created in Step 2.2 (p. 86).

When you finish, choose **Create function**.

6. Delete the sample function in the code editor, and then paste the following code:

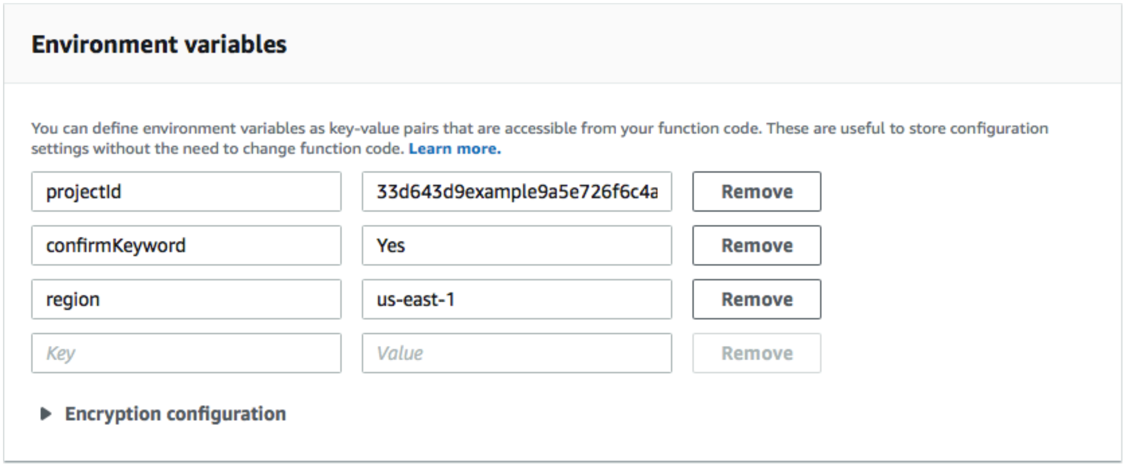
```
var AWS = require('aws-sdk');
var projectId = process.env.projectId;
var confirmKeyword = process.env.confirmKeyword.toLowerCase();
var pinpoint = new AWS.Pinpoint({region: process.env.region});

exports.handler = (event, context) => {
  console.log('Received event:', event);
  var timestamp = event.Records[0].Sns.Timestamp;
  var message = JSON.parse(event.Records[0].Sns.Message);
  var originationNumber = message.originationNumber;
  var response = message.messageBody.toLowerCase();

  if (response.includes(confirmKeyword)) {
    updateEndpointOptIn(originationNumber, timestamp);
  }
};

function updateEndpointOptIn (originationNumber, timestamp) {
  var endpointId = originationNumber.substring(1);

  var params = {
    ApplicationId: projectId,
    EndpointId: endpointId,
    EndpointRequest: {
      Address: originationNumber,
      ChannelType: 'SMS',
      OptOut: 'NONE',
      Attributes: {
        OptInTimestamp: [
          timestamp
        ]
      },
    }
  };
  pinpoint.updateEndpoint(params, function(err, data) {
    if (err) {
      console.log("An error occurred.\n");
      console.log(err, err.stack);
    }
    else {
      console.log("Successfully changed the opt status of endpoint ID " + endpointId);
    }
  });
}
```

7. Under **Environment variables**, do the following:

- In the first row, create a variable with a key of **`projectId`**. Next, set the value to the unique ID of the project that you created in Step 1.1 (p. 82).
- In the second row, create a variable with a key of **`region`**. Next, set the value to the Region that you use Amazon Pinpoint in, such as **`us-east-1`** or **`us-west-2`**.
- In the third row, create a variable with a key of **`confirmKeyword`**. Next, set the value to the confirmation keyword that you created in Step 1.3 (p. 84).

Note
The keyword isn't case sensitive. This function converts the incoming message to lowercase letters.

When you finish, the **Environment Variables** section should resemble the example shown in the following image.



Environment variables

You can define environment variables as key-value pairs that are accessible from your function code. These are useful to store configuration settings without the need to change function code. Learn more.

projectId	33d643d9example9a5e726f6c4a	Remove
confirmKeyword	Yes	Remove
region	us-east-1	Remove
Key	Value	Remove

8. At the top of the page, choose **Save**.

Step 3.2.1: Test the function

After you create the function, you should test it to make sure that it's configured properly. Also, make sure that the IAM role you created has the appropriate permissions.

To test the function

1. Choose **Test**.
2. On the **Configure test event** window, do the following:

   a. Choose **Create new test event**.
   b. For **Event name**, enter a name for the test event, such as **`MyResponse`**.
   c. Erase the example code in the code editor. Paste the following code:

```
{
  "Records":[
    {
      "Sns":{
        "Message":"{\"originationNumber\":\"+12065550142\",\"messageBody\":
\"Yes\"}",
        "Timestamp":"2019-02-20T17:47:44.147Z"
      }
    }
  ]
}
```

   In the preceding code example, replace the values of the `originationNumber` attribute with the phone number that you used when you tested the previous Lambda function. Replace the value of `messageBody` with the two-way SMS keyword that you specified in Step 1.3 (p. 84). Optionally, you can replace the value of `Timestamp` with the current date and time.

   d. Choose **Create**.

3. Choose **Test** again.
4. Under **Execution result: succeeded**, choose **Details**. In the **Log output** section, review the output of the function. Make sure that the function ran without errors.
5. Open the Amazon Pinpoint console at https://console.aws.amazon.com/pinpoint/.
6. On the **All projects** page, choose the project that you created in Step 1.1 (p. 82).
7. In the navigation pane, choose **Segments**. On the **Segments page**, choose **Create a segment**.
8. In **Segment group 1**, under **Add filters to refine your segment**, choose **Filter by user**.
9. For **Choose a user attribute**, choose **FirstName**. Then, for **Choose values**, choose the first name that you specified in the test event.

   The **Segment estimate** section should show that there is one eligible endpoint, and one total endpoint.

Next: Set up Amazon API Gateway (p. 95)

Step 4: Set up Amazon API Gateway

In this section, you create a new API by using Amazon API Gateway. The registration form that you deploy in this solution calls this API. API Gateway then passes the information that's captured on the registration form to the Lambda function you created in Step 3 (p. 87).

Step 4.1: Create the API

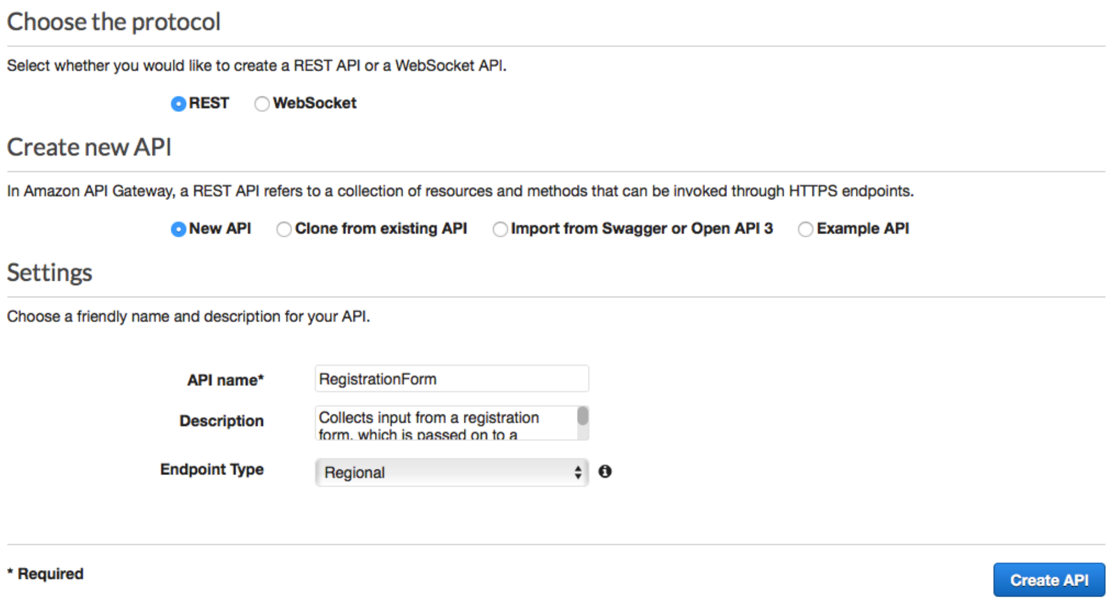
First, you have to create a new API in API Gateway. The following procedures show you how to create a new REST API.

To create a new API

1. Open the API Gateway console at https://console.aws.amazon.com/apigateway/.
2. Choose **Create API**. Make the following selections:

   - Under **Choose the protocol**, choose **REST**.
   - Under **Create new API**, choose **New API**.
   - Under **Settings**, for **Name**, enter a name, such as `RegistrationForm`. For **Description**, optionally enter some text that describes the purpose of the API. For **Endpoint Type**, choose **Regional**. Then, choose **Create API**.

   An example of these settings is shown in the following image.


Choose the protocol

Select whether you would like to create a REST API or a WebSocket API.

[✓ REST] WebSocket

Create new API

In Amazon API Gateway, a REST API refers to a collection of resources and methods that can be invoked through HTTPS endpoints.

[✓ New API] Clone from existing API Import from Swagger or Open API 3 Example API

Settings

Choose a friendly name and description for your API.

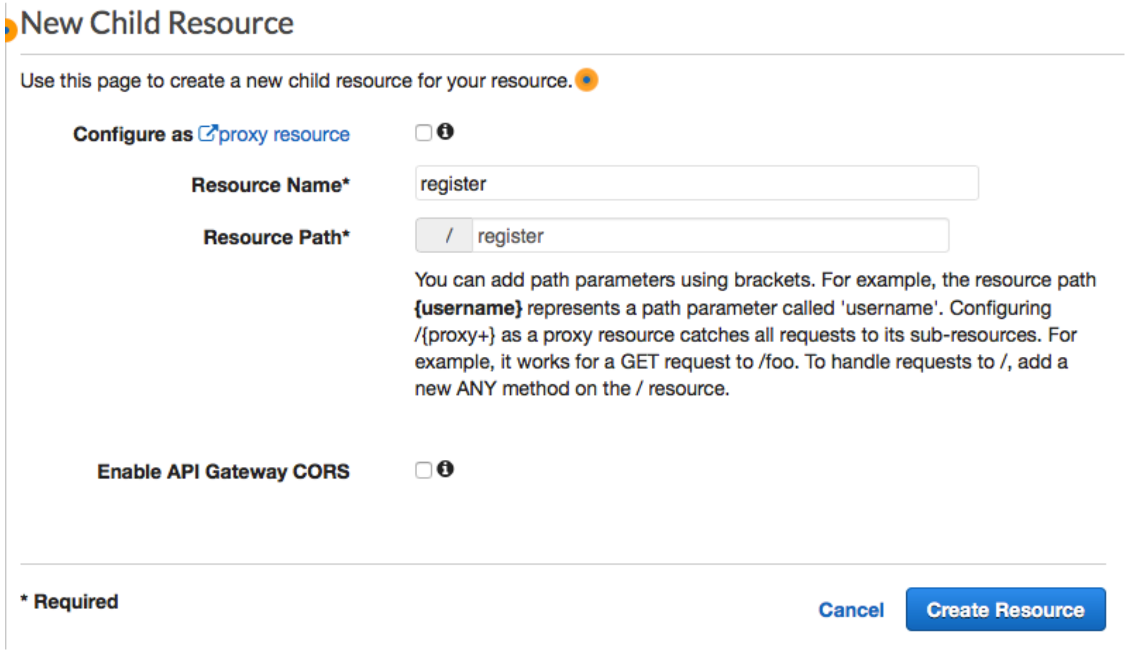
API name*	RegistrationForm
Description	Collects input from a registration form, which is passed on to a
Endpoint Type	Regional

* Required

Step 4.2: Create a resource

Now that you've created an API, you can start to add resources to it. After that, you add a POST method to the resource, and tell API Gateway to pass the data that you receive from this method to your Lambda function.

1. On the **Actions** menu, choose **Create Resource**. In the **New Child Resource** pane, for **Resource Name**, enter **`register`**, as shown in the following image. Choose **Create Resource**.


New Child Resource

Use this page to create a new child resource for your resource.

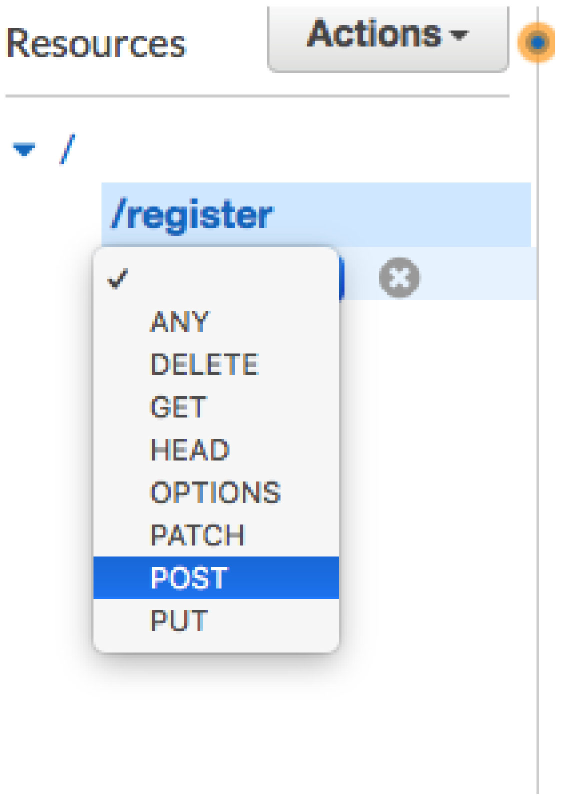
Configure as proxy resource	☐
Resource Name*	register
Resource Path*	/ register
Enable API Gateway CORS	☐

You can add path parameters using brackets. For example, the resource path **{username}** represents a path parameter called 'username'. Configuring /{proxy+} as a proxy resource catches all requests to its sub-resources. For example, it works for a GET request to /foo. To handle requests to /, add a new ANY method on the / resource.

* Required

2. On the **Actions** menu, choose **Create Method**. From the menu that appears, choose **POST**, as shown in the following image. Then choose the **check mark**

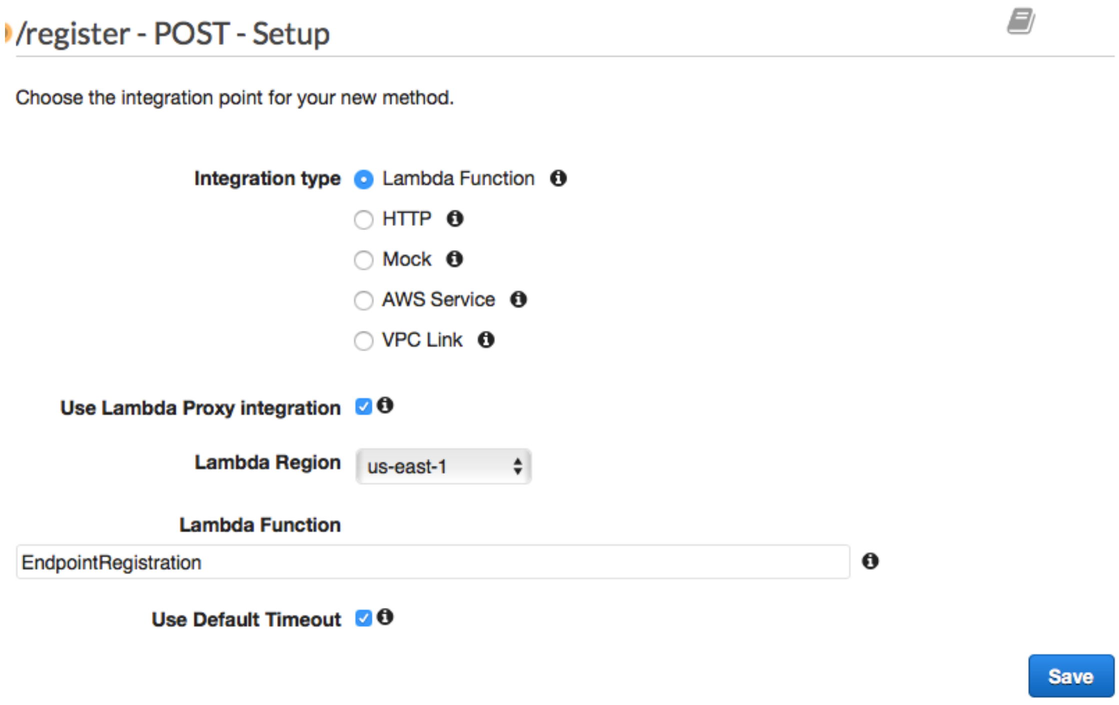
(✓)
button.



3. In the **/register - POST - Setup** pane, make the following selections:

   - For **Integration type**, choose **Lambda Function**.
   - Choose **Use Lambda Proxy Integration**.
   - For **Lambda Region**, choose the Region that you created the Lambda function in.
   - For **Lambda Function**, choose the RegisterEndpoint function that you created in Step 3 (p. 87).

   An example of these settings is shown in the following image.



/register - POST - Setup

Choose the integration point for your new method.

Integration type: [✓ Lambda Function] HTTP Mock AWS Service VPC Link

Use Lambda Proxy integration ✓

Lambda Region: us-east-1

Lambda Function: EndpointRegistration

Use Default Timeout ✓

Choose **Save**. On the window that appears, choose **OK** to give API Gateway permission to execute your Lambda function.

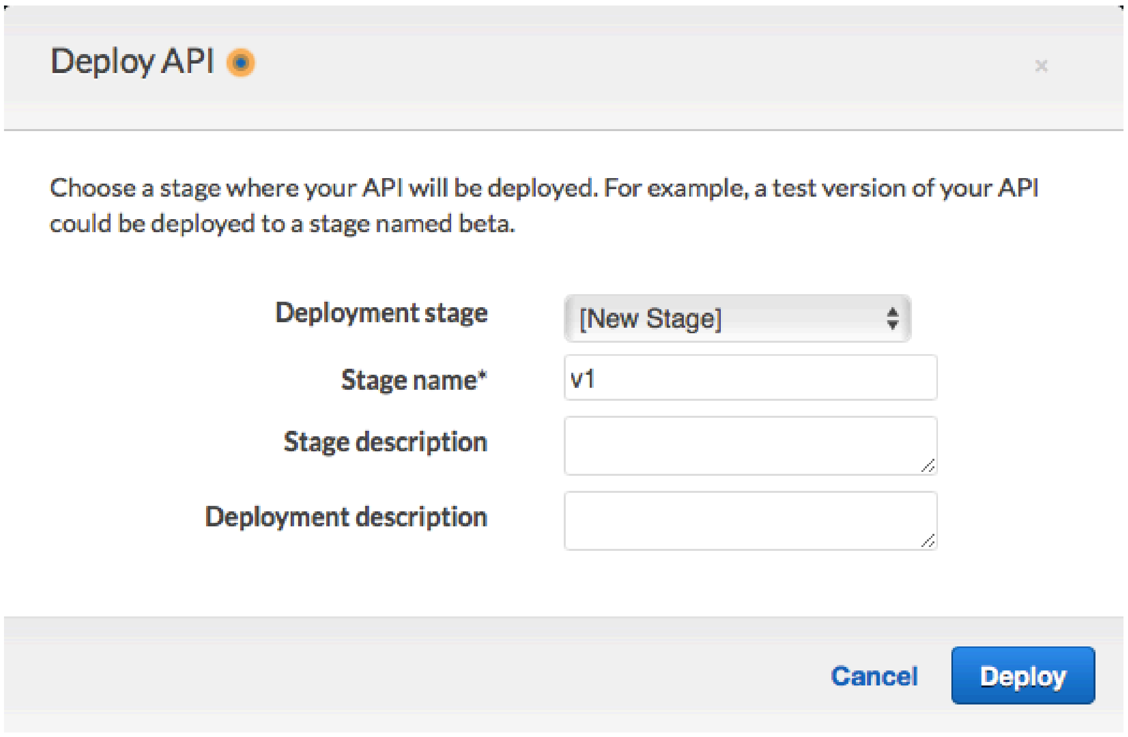
Step 4.3: Deploy the API

The API is now ready to use. At this point, you have to deploy it in order to create a publicly accessible endpoint.

1. On the **Actions** menu, choose **Deploy API**. On the **Deploy API** window, make the following selections:

   - For **Deployment stage**, choose **[New Stage]**.
   - For **Stage name**, enter `v1`.
   - Choose **Deploy**.

   An example of these selections is shown in the following image.

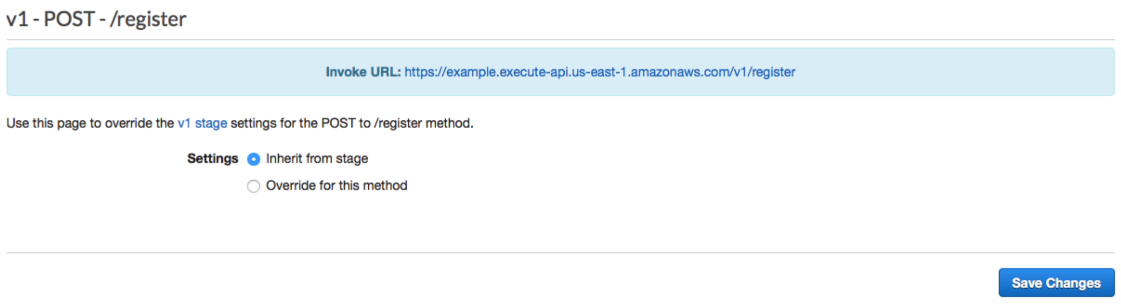


Deploy API

Choose a stage where your API will be deployed. For example, a test version of your API could be deployed to a stage named beta.

Deployment stage: [New Stage]
Stage name*: v1
Stage description:
Deployment description:

2. In the **v1 Stage Editor** pane, choose the **/register** resource, and then choose the **POST** method. Copy the address that's shown next to **Invoke URL**, as shown in the following image.



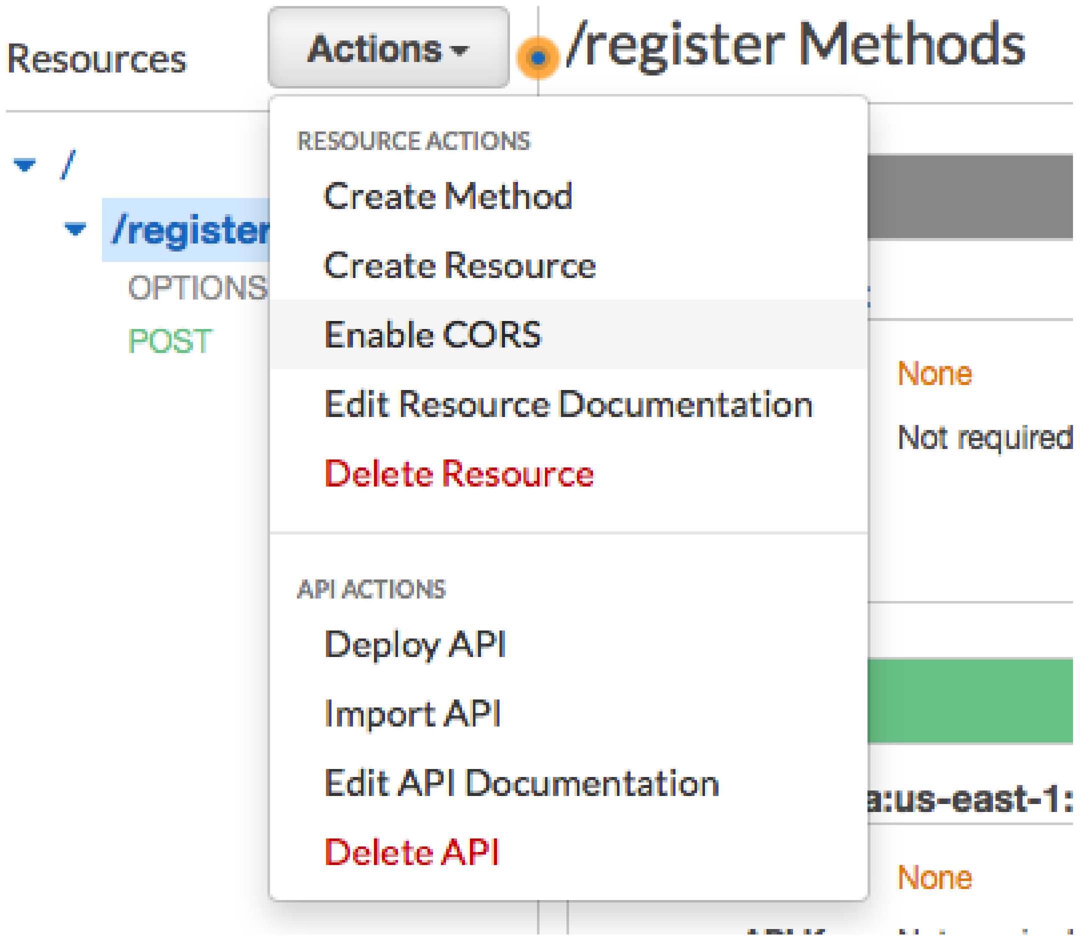
v1 - POST - /register

Invoke URL: https://example.execute-api.us-east-1.amazonaws.com/v1/register

Use this page to override the v1 stage settings for the POST to /register method.

Settings [✓ Inherit from stage] Override for this method

3. In the navigation pane, choose **Resources**. In the list of resources, choose the **/register** resource. Finally, on the **Actions** menu, choose **Enable CORS**, as shown in the following image.



4. On the **Enable CORS** pane, choose **Enable CORS and replace existing CORS headers**.

Next: Create and deploy the web form (p. 100)

Step 5: Create and deploy the web form

All of the components of this solution that use AWS services are now in place. The last step is to create and deploy the web form that captures customer's data.

Step 5.1: Create the JavaScript form handler

In this section, you create a JavaScript function that parses the content of the web form that you create in the next section. After parsing the content, this function sends the data to the API that you created in Part 4 (p. 95).

To create the form handler

1. In a text editor, create a new file.
2. In the editor, paste the following code.

```
$(document).ready(function() {

  // Handle form submission.
  $("#submit").click(function(e) {
    var firstName = $("#firstName").val(),
        lastName = $("#lastName").val(),
        source = window.location.pathname,
        optTimestamp = undefined,
        utcSeconds = Date.now() / 1000,
        timestamp = new Date(0),
        phone = $("#areaCode").val()
              + $("#phone1").val()
              + $("#phone2").val();

    e.preventDefault();

    if (firstName == "") {
      $('#form-response').html('<div class="mt-3 alert alert-info" role="alert">Please
enter your first name.</div>');
    } else if (lastName == "") {
      $('#form-response').html('<div class="mt-3 alert alert-info" role="alert">Please
enter your last name.</div>');
    } else if (phone.match(/[^0-9]/gi)) {
      $('#form-response').html('<div class="mt-3 alert alert-info" role="alert">Your
phone number contains invalid characters. Please check the phone number that you
supplied.</div>');
    } else if (phone.length < 10) {
      $('#form-response').html('<div class="mt-3 alert alert-info" role="alert">Please
enter your phone number.</div>');
    } else if (phone.length > 10) {
      $('#form-response').html('<div class="mt-3 alert alert-info" role="alert">Your
phone number contains too many digits. Please check the phone number that you
supplied.</div>');
    } else {
      $('#submit').prop('disabled', true);
      $('#submit').html('<span class="spinner-border spinner-border-sm" role="status"
aria-hidden="true"></span>Saving your preferences</button>');

      timestamp.setUTCSeconds(utcSeconds);

      var data = JSON.stringify({
        'destinationNumber': phone,
        'firstName': firstName,
        'lastName': lastName,
        'source': source,
        'optTimestamp': timestamp.toString()
      });

      $.ajax({
        type: 'POST',
        url: 'https://example.execute-api.us-east-1.amazonaws.com/v1/register',
        contentType: 'application/json',
        data: data,
        success: function(res) {
          $('#form-response').html('<div class="mt-3 alert alert-success"
role="alert"><p>Congratulations! You&apos;ve successfully registered for SMS Alerts
from ExampleCorp.</p><p>We just sent you a message. Follow the instructions in the
message to confirm your subscription. We won&apos;t send any additional messages
until we receive your confirmation.</p><p>If you decide you don&apos;t want to receive
any additional messages from us, just reply to one of our messages with the keyword
STOP.</p></div>');
          $('#submit').prop('hidden', true);
          $('#unsubAll').prop('hidden', true);
          $('#submit').text('Preferences saved!');
        },
        error: function(jqxhr, status, exception) {
          $('#form-response').html('<div class="mt-3 alert alert-danger"
role="alert">An error occurred. Please try again later.</div>');
          $('#submit').text('Save preferences');
          $('#submit').prop('disabled', false);
        }
      });
    }
  });
});
```

3. In the preceding example, replace *https://example.execute-api.us-east-1.amazonaws.com/v1/register* with the Invoke URL that you obtained in Step 4.3 (p. 98).
4. Save the file.

Step 5.2: Create the form file

In this section, you create an HTML file that contains the form that customers use to register for your SMS program. This file uses the JavaScript form handler that you created in the previous section to transmit the form data to your Lambda function.

> **Important**
> When a user submits this form, it triggers a Lambda function that calls several Amazon Pinpoint API operations. Malicious users could launch an attack on your form that could cause a large number of requests to be made. If you plan to use this solution for a production use case, you should secure it by using a system such as Google reCAPTCHA.

To create the form

1. In a text editor, create a new file.
2. In the editor, paste the following code.

```
<!doctype html>
<html lang="en">

<head>
  <!-- Meta tags required by Bootstrap -->
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-
fit=no">

  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/
css/bootstrap.min.css" integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/
iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">
  <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-
q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"
 crossorigin="anonymous"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/
popper.js/1.14.7/umd/popper.min.js" integrity="sha384-
UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1"
 crossorigin="anonymous"></script>
  <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"
 integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM"
 crossorigin="anonymous"></script>
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></
script>

  <script type="text/javascript" src="SMSFormHandler.js"></script>
  <title>SMS Registration Form</title>
</head>

<body>
  <div class="container">
    <div class="row justify-content-center mt-3">
```

```
      <div class="col-md-6">
        <h1>Register for SMS Alerts</h1>
        <p>Enter your phone number below to sign up for PromotionName messages from
 ExampleCorp.</p>
        <p>We don't share your contact information with anyone else. For more
 information, see our <a href="http://example.com/privacy">Privacy Policy</a>.</p>
        <p>ExampleCorp alerts are only available to recipients in the United States.</
p>
      </div>
    </div>
    <div class="row justify-content-center">
      <div class="col-md-6">
        <form>
          <div class="form-group">
            <label for="firstName" class="font-weight-bold">First name</label>
            <input type="text" class="form-control" id="firstName" placeholder="Your
 first name" required>
          </div>
          <div class="form-group">
            <label for="lastName" class="font-weight-bold">Last name</label>
            <input type="text" class="form-control" id="lastName" placeholder="Your
 last name" required>
          </div>
          <label for="areaCode" class="font-weight-bold">Phone number</label>
          <div class="input-group">
            <span class="h3">(&nbsp;</span>
            <input type="tel" class="form-control" id="areaCode" placeholder="Area
 code" required>
            <span class="h3">&nbsp;)&nbsp;</span>
            <input type="tel" class="form-control" id="phone1" placeholder="555"
 required>
            <span class="h3">&nbsp;-&nbsp;</span>
            <input type="tel" class="form-control" id="phone2" placeholder="0199"
 required>
          </div>
          <div id="form-response"></div>
          <button id="submit" type="submit" class="btn btn-primary btn-block
 mt-3">Submit</button>
        </form>
      </div>
    </div>
    <div class="row mt-3">
      <div class="col-md-12 text-center">
        <small class="text-muted">Copyright © 2019, ExampleCorp or its affiliates.</
small>
      </div>
    </div>
  </div>
</body>

</html>
```

3. In the preceding example, replace `SMSFormHandler.js` with the full path to the form handler JavaScript file that you created in the previous section.
4. Save the file.

Step 5.2: Upload the form files

Now that you've created the HTML form and the JavaScript form handler, the last step is to publish these files to the internet. This section assumes that you have an existing web hosting provider. If you don't have an existing hosting provider, you can launch a website by using Amazon Route53, Amazon Simple Storage Service (Amazon S3), and Amazon CloudFront. For more information, see Host a static website.

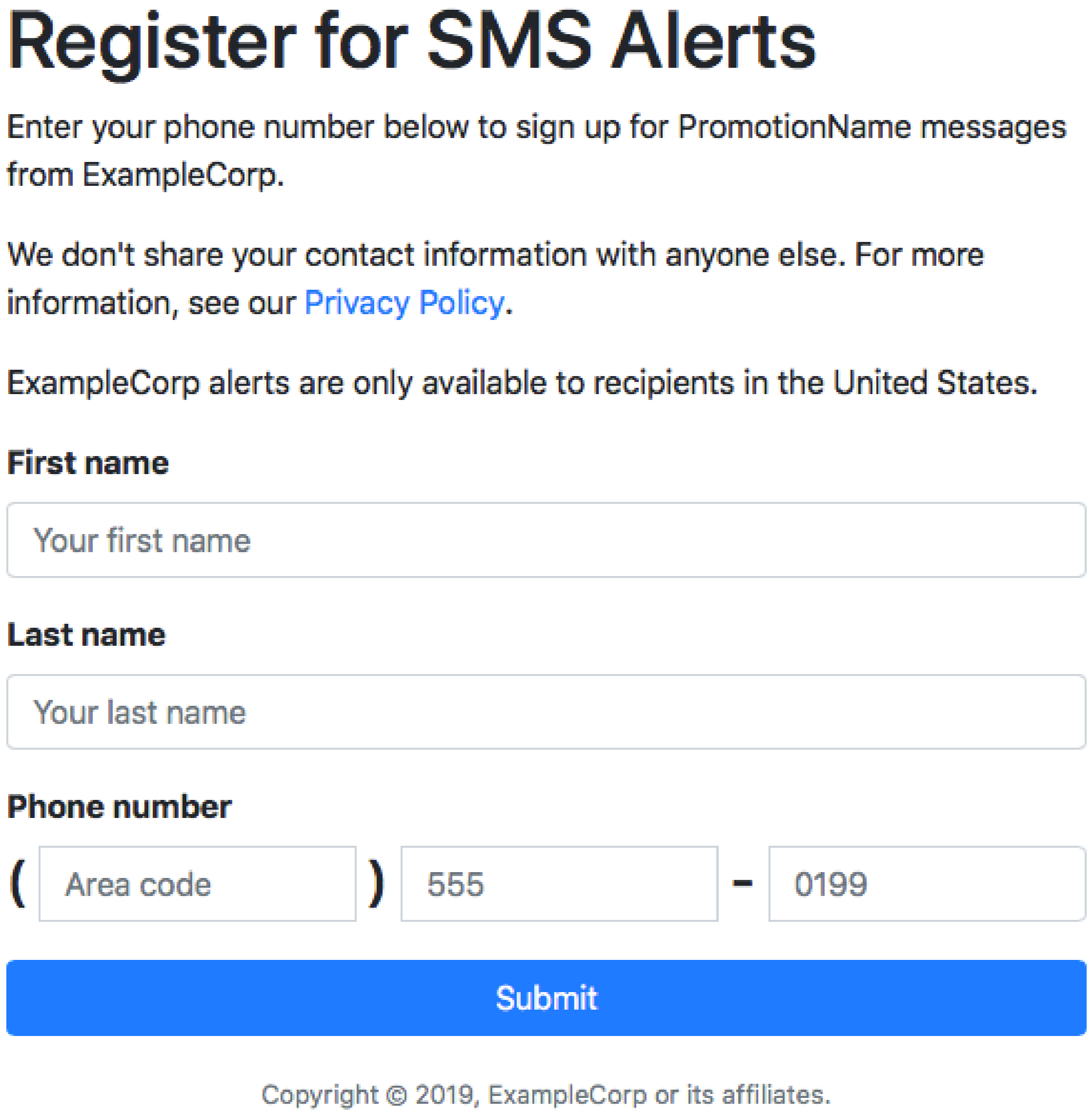
If you use another web hosting provider, consult the provider's documentation for more information about publishing webpages.

Step 5.3: Test the form

After you publish the form, you should submit some test events to make sure that it works as expected.

To test the registration form

1. In a web browser, go to the location where you uploaded the registration form. If you used the code example from Step 5.1 (p. 102), you see a form that resembles the example in the following image.



Register for SMS Alerts

Enter your phone number below to sign up for PromotionName messages from ExampleCorp.

We don't share your contact information with anyone else. For more information, see our Privacy Policy.

ExampleCorp alerts are only available to recipients in the United States.

First name

Your first name

Last name

Your last name

Phone number

(Area code) 555 - 0199

Submit

Copyright © 2019, ExampleCorp or its affiliates.

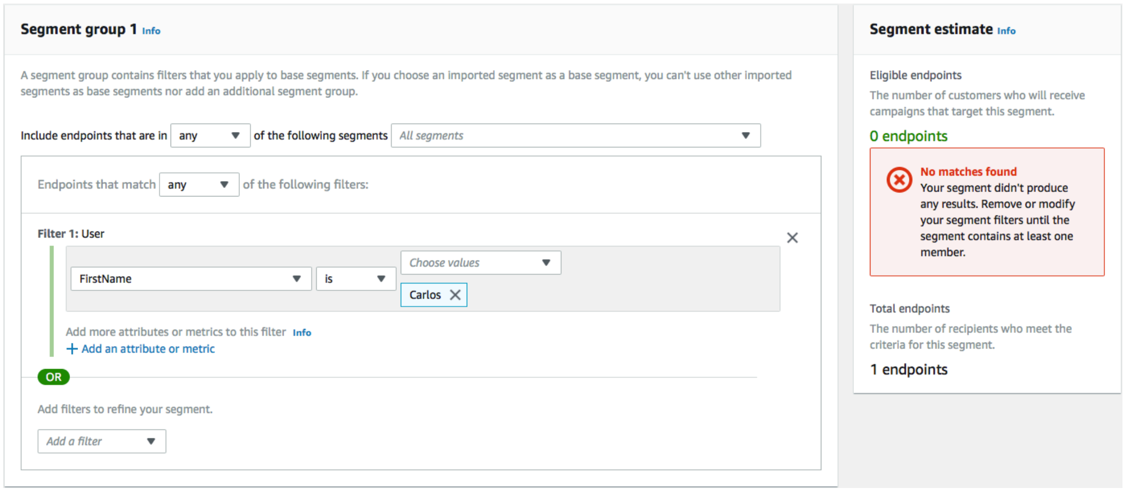
2. Enter your contact information in the **First name**, **Last name**, and **Phone number** fields.

   Note
   When you submit the form, Amazon Pinpoint attempts to send a message to the phone number that you specified. Because of this functionality, you should use a real phone number to test the solution from beginning to end.

If you tested the Lambda function in Step 3 (p. 87), your Amazon Pinpoint project already contains at least one endpoint. When you test this form, you should either submit a different phone number on the form, or delete the existing endpoint by using the DeleteEndpoint API operation.

3. Check the device that's associated with the phone number that you specified to make sure that it received the message.
4. Open the Amazon Pinpoint console at https://console.aws.amazon.com/pinpoint/.
5. On the **All projects** page, choose the project that you created in Step 1.1 (p. 82).
6. In the navigation pane, choose **Segments**. On the **Segments page**, choose **Create a segment**.
7. In **Segment group 1**, under **Add filters to refine your segment**, choose **Filter by user**.
8. For **Choose a user attribute**, choose **FirstName**. Then, for **Choose values**, choose the first name that you specified when you submitted the form.

   The **Segment estimate** section should show that there are zero eligible endpoints, and one endpoint (under Total endpoints), as shown in the following example. This result is expected. When the Lambda function creates a new endpoint, the endpoint is opted out by default.


Segment group 1 Info

A segment group contains filters that you apply to base segments. If you choose an imported segment as a base segment, you can't use other imported segments as base segments nor add an additional segment group.

Include endpoints that are in: any, of the following segments: All segments

Endpoints that match: any, of the following filters:

Filter 1: User

FirstName is Carlos

Segment estimate Info

Eligible endpoints
The number of customers who will receive campaigns that target this segment.
0 endpoints

No matches found
Your segment didn't produce any results. Remove or modify your segment filters until the segment contains at least one member.

Total endpoints
The number of recipients who meet the criteria for this segment.
1 endpoints

Choose values
Add more attributes or metrics to this filter Info
+ Add an attribute or metric
OR
Add filters to refine your segment.
Add a filter

9. On the device that received the message, reply to the message with the two-way SMS keyword that you specified in Step 1.3 (p. 84). Amazon Pinpoint sends a response message immediately.
10. In the Amazon Pinpoint console, repeat steps 4 through 8. This time, when you create the segment, you see one eligible endpoint, and one total endpoint. This result is expected, because the endpoint is now opted in.

Next steps

By completing this tutorial, you've done the following:

- Created an Amazon Pinpoint project, configured the SMS channel, and obtained a dedicated long code.
- Created a IAM policy that uses the principal of least privilege to grant access rights, and associated that policy with a role.
- Created two Lambda functions that use the PhoneNumberValidate, UpdateEndpoint, and SendMessages operations in the Amazon Pinpoint API.
- Created a REST API using API Gateway.
- Created and deployed a web-based form that collects customers' contact information.
- Performed tests on the solution to make sure it works.

This section discusses a few ways that you can use the customer information that you collect by using this solution. It also includes some suggestions of ways that you can customize this solution to fit your unique use case.

Create customer segments

All of the customer details that you collect through this form are stored as endpoints. This solution creates endpoints that contain several attributes that you can use for segmentation purposes.

For example, this solution captures an endpoint attribute called `Source`. This attribute contains the full path to the location where the form was hosted. When you create a segment, you can filter the segment by endpoint, and then further refine the filter by choosing a `Source` attribute.

Creating segments based on the `Source` attribute can be useful in several ways. First, it allows you to quickly create a segment of customers who signed up to receive SMS messages from you. Additionally, the segmentation tool in Amazon Pinpoint automatically excludes endpoints that aren't opted in to receive messages.

The `Source` attribute is also useful if you decide to host the registration form in several different locations. For example, your marketing material could refer to a form that's hosted in one location, while customers who encounter the form while browsing your website could see a version that's hosted somewhere else. When you do this, the Source attributes for customers who complete the form after seeing your marketing materials are different from those who complete the form after finding it on your website. You can use this difference to create distinct segments, and then send tailored communications to each of those audiences.

Send personalized campaign messages

After you create segments, you can start sending campaigns to those segments. When you create campaign messages, you can personalize them by specifying which endpoint attributes you want to include in the message. For example, the web form used in this solution requires the customer to enter their first and last names. These values are stored in the user record that's associated with the endpoint.

For example, if you use the `GetEndpoint` API operation to retrieve information about an endpoint that was created using this solution, you see a section that resembles the following example:

```
...
"User": {
  "UserAttributes": {
    "FirstName": [
      "Carlos"
    ],
    "LastName": [
      "Salazar"
    ]
  }
}
...
```

If you want to include the values of these attributes in your campaign message, you can use dot notation to refer to the attribute. Then enclose the entire reference in double curly braces. For example, to include each recipient's first name in a campaign message, include the following string in the message: `{{User.UserAttributes.FirstName}}`. When Amazon Pinpoint sends the message, it replaces the string with the value of the `FirstName` attribute.

Use the form to collect additional information

You can modify this solution to collect additional information on the registration form. For example, you could ask the customer to provide their address, and then use the address data to populate the

`Location.City`, `Location.Country`, `Location.Region`, and `Location.PostalCode` fields in the `Endpoint` resource. Collecting address information on the registration form might result in the endpoint containing more accurate information. To make this change, you need to add the appropriate fields to the web form. You also have to modify the JavaScript code for the form to pass the new values. Finally, you have to modify the Lambda function that creates the endpoint to handle the new incoming information.

You can also modify the form so that it collects contact information in other channels. For example, you could use the form to collect the customer's email address in addition to their phone number. To make this change, you need to modify the HTML and JavaScript for the web form. You also have to modify the Lambda function that creates the endpoint so that it creates two separate endpoints (one for the email endpoint, and one for the SMS endpoint). You should also modify the Lambda function so that it generates a unique value for the `User.UserId` attribute, and then associates that value with both endpoints.

Record additional attributes for auditing purposes

This solution records two valuable attributes when it creates and updates endpoints. First, when the first Lambda function initially creates the endpoint, it records the URL of the form itself in the `Attributes.Source` attribute. If the customer responds to the message, the second Lambda function creates an `Attributes.OptInTimestamp` attribute. This attribute contains the exact date and time when the customer provided their consent to receive messages from you.

Both of these fields can be useful if you're ever asked by a mobile carrier or regulatory agency to provide evidence of a customer's consent. You can retrieve this information at any time by using the GetEndpoint API operation.

You can also modify the Lambda functions to record additional data that might be useful for auditing purposes, such as the IP address that the registration request was submitted from.

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. 108) or the Amazon Pinpoint API to import endpoints, export event data, define customer segments, create and run campaigns, and more.

To start integrating your apps, see the section called “Integrating the mobile SDKs or JS library” (p. 109).

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 JavaScript
- AWS Amplify JavaScript library for react native
- AWS SDK for JavaScript version 2.7.10 or later
- AWS SDK for Java version 2.15.x 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 (Boto3) 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.

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

Use AWS Amplify to integrate your app with AWS. For iOS apps, see Getting started in the iOS SDK documentation. For Android apps, see Getting started in the Android SDK documentation. These topics help you:

- Create a project with AWS Amplify.
- Connect your app to the AWS features and resources that AWS Amplify supports.

Integrating the AWS Amplify JavaScript library

To integrate AWS Amplify with your JavaScript application, see Getting started (JavaScript) or Getting started (React) in the AWS Amplify React documentation. These topics help you:

- Use command line tools and AWS Amplify to create a project.
- 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.

Next step

You've integrated AWS Amplify with your application. Next, update your code to register your users' devices as endpoints. See Registering endpoints in your application (p. 110).

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. 109).

Registering endpoints with the AWS mobile SDKs for Android or iOS

You can use the AWS Mobile SDKs for Android or iOS to register and customize endpoints. For more information, and to view code examples, see the following documents:

- Registering endpoints in your application in the Android SDK documentation.
- Registering endpoints in your application in the iOS SDK documentation.

Registering endpoints with the AWS Amplify JavaScript library

You can use the AWS Amplify JavaScript library to register and update endpoints in your apps. For more information, and to view code examples, see Update endpoint in the AWS Amplify JavaScript documentation.

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, complete the steps at Reporting events in your application (p. 111) to update your app to report usage data.

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. 206).

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 your app with AWS Amplify. See Integrating the AWS mobile SDKs or JavaScript library with your application (p. 109).

- Update your application to register endpoints. See Registering endpoints in your application (p. 110).

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.

For more information about updating your app to record and submit events to Amazon Pinpoint, see the following pages in the AWS Amplify documentation:

- Analytics in the iOS SDK documentation
- Analytics in the Android SDK documentation

Reporting events with the AWS Amplify JavaScript library

You can enable JavaScript and React Native apps to report application usage events to Amazon Pinpoint by using the AWS Amplify JavaScript library. For more information about updating your app to submit events to Amazon Pinpoint, see Analytics in the AWS Amplify JavaScript documentation.

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. 112).

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

In order to set up Amazon Pinpoint so that it can send push notifications to your apps, you first have to provide the credentials that authorize Amazon Pinpoint to send messages to your app. The credentials that you provide depend on which push notification system you use:

- 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 the Apple Push Notification service.
- For Android apps, you provide a Web API Key, which you obtain from the Firebase console. These credentials authorize Amazon Pinpoint to send messages to your app through Firebase Cloud Messaging.

After you obtain the credentials for a push notification channel, you have to create a project in Amazon Pinpoint and provide it with the credentials for the push notification service.

Topics

Setting up iOS push notifications

Push notifications for iOS apps are sent using the 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. You can find more information about completing these steps in Setting up push notification services in the iOS SDK documentation.

Working with APNs tokens

As a best practice, you should develop your app so that your customers' device tokens are regenerated when the app is reinstalled.

If a recipient upgrades their device to a new major version of iOS (for example, from iOS 12 to iOS 13), and later reinstalls your app, the app generates a new token. If your app doesn't refresh the token, the older token is used to send the notification. As a result, Apple Push Notification service (APNs) rejects the notification, because the token is now invalid. When you attempt to send the notification, you receive a message failure notification from APNs.

Setting up Android push notifications

Push notifications for Android apps are sent using Firebase Cloud Messaging (FCM), which replaces Google Cloud Messaging (GCM). Before you can send push notifications to Android devices, you must obtain FCM credentials. You can then use those credentials to create an Android project and launch a sample app that can receive push notifications. You can find more information about completing these steps in the Push notifications section of the Android SDK documentation.

Create a project in Amazon Pinpoint

In Amazon Pinpoint, a *project* is a collection of settings, data, campaigns, and segments that all share a common purpose. In the Amazon Pinpoint API, projects are also referred to as *applications*. This section uses the word "project" exclusively when referring to this concept.

To start sending push notifications in Amazon Pinpoint, you have to create a project. Next, you have to enable the push notification channels that you want to use by providing the appropriate credentials.

You can create new projects and set up push notification channels by using the Amazon Pinpoint console. For more information, see Setting up Amazon Pinpoint push notification channels in the *Amazon Pinpoint User Guide*.

You can also create and set up projects by using the Amazon Pinpoint API, an AWS SDK, or the AWS Command Line Interface (AWS CLI). To create a project, use the `Apps` resource. To configure push notification channels, use the following resources:

- APNs channel to send messages to users of iOS devices by using the Apple Push Notification service.
- ADM channel to send messages to users of Amazon Kindle Fire devices.
- Baidu channel to send messages to Baidu users.
- GCM channel to send messages to Android devices by using Firebase Cloud Messaging (FCM), which replaces Google Cloud Messaging (GCM).

Handling push notifications

After you obtain the credentials that are required to send push notifications, you can update your apps so that they're able to receive push notifications. For more information, see the following sections in the AWS Amplify documentation:

- **Firebase Cloud Messaging**: Handling FCM/GCM push notifications
- **Apple Push Notification service**: Add Amazon Pinpoint targeted and campaign push messaging
- **Amazon Device Messaging**: Handling amazon device messaging push notifications
- **Baidu Push**: Handling baidu push notifications

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. 110) 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. 110).

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 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 information about the quotas that apply to endpoints, such as the number of attributes that you can assign, see the section called “Endpoint quotas” (p. 377).

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 learn about adding users with Amazon Cognito, see Using amazon pinpoint analytics with amazon cognito user pools in the *Amazon Cognito Developer Guide*.

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":{
            "FirstName":["Wang"],
            "LastName":["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 software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.pinpoint.PinpointClient;
import software.amazon.awssdk.services.pinpoint.model.EndpointRequest;
import software.amazon.awssdk.services.pinpoint.model.EndpointUser;
import software.amazon.awssdk.services.pinpoint.model.ChannelType;
import software.amazon.awssdk.services.pinpoint.model.UpdateEndpointRequest;
import software.amazon.awssdk.services.pinpoint.model.UpdateEndpointResponse;
import software.amazon.awssdk.services.pinpoint.model.PinpointException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
```

```
   public static void updatePinpointEndpoint(PinpointClient pinpoint,String
 applicationId, String endPointId) {

        try{
            List<String> wangXiList = new ArrayList<String>();
            wangXiList.add("cooking");
            wangXiList.add("running");
            wangXiList.add("swimming");

            Map myMapWang = new HashMap<String, List>();
            myMapWang.put("interests", wangXiList);

            List<String> myNameWang = new ArrayList<String>();
            myNameWang.add("Wang ");
            myNameWang.add("Xiulan");

            Map wangName = new HashMap<String, List>();
            wangName.put("name",myNameWang );

            EndpointUser wangMajor = EndpointUser.builder()
                .userId("example_user_10")
                .userAttributes(wangName)
                .build();

            // Create an EndpointBatchItem object for Mary Major
            EndpointRequest wangXiulanEndpoint = EndpointRequest.builder()
                .channelType(ChannelType.EMAIL)
                .address("wang_xiulan@example.com")
                .attributes(myMapWang)
                .user(wangMajor)
                .build();

            // Adds multiple endpoint definitions to a single request object.
            UpdateEndpointRequest endpointList = UpdateEndpointRequest.builder()
               .applicationId(applicationId)
                .endpointRequest(wangXiulanEndpoint)
                .endpointId(endPointId)
                .build();

            UpdateEndpointResponse result = pinpoint.updateEndpoint(endpointList);

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

    } catch (PinpointException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
```

For the full SDK example, see AddExampleUser.java on GitHub.

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":{
            "FirstName":"Wang",
            "LastName":"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 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 learn how to define a segment by importing a list of user IDs, see Importing segments in the *Amazon Pinpoint User Guide*.

For information about sending a direct message to up to 100 user IDs, see Users messages in the *Amazon Pinpoint API Reference*.

For information about the quotas that apply to endpoints, including the number of user attributes that you can assign, see the section called “Endpoint quotas” (p. 377).

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. 128) 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": {
                    "FirstName": "Richard",
                    "LastName": "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": {
                    "FirstName": "Mary",
                    "LastName": "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 software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.pinpoint.PinpointClient;
import software.amazon.awssdk.services.pinpoint.model.UpdateEndpointsBatchResponse;
import software.amazon.awssdk.services.pinpoint.model.EndpointUser;
import software.amazon.awssdk.services.pinpoint.model.EndpointBatchItem;
import software.amazon.awssdk.services.pinpoint.model.ChannelType;
import software.amazon.awssdk.services.pinpoint.model.EndpointBatchRequest;
import software.amazon.awssdk.services.pinpoint.model.PinpointException;
import software.amazon.awssdk.services.pinpoint.model.UpdateEndpointsBatchRequest;
import java.util.Map;
import java.util.List;
import java.util.ArrayList;
import java.util.HashMap;
```

```
    public static void updateEndpointsViaBatch( PinpointClient pinpoint, String
 applicationId) {

        try {
            List<String> myList = new ArrayList<String>();
            myList.add("music");
            myList.add("books");

            Map myMap = new HashMap<String, List>();
            myMap.put("attributes", myList);

            List<String> myNames = new ArrayList<String>();
            myList.add("Richard");
            myList.add("Roe");

            Map myMap2 = new HashMap<String, List>();
            myMap2.put("name",myNames );

            EndpointUser richardRoe = EndpointUser.builder()
                .userId("example_user_1")
                .userAttributes(myMap2)
                .build();

            // Create an EndpointBatchItem object for Richard Roe
            EndpointBatchItem richardRoesEmailEndpoint = EndpointBatchItem.builder()
                .channelType(ChannelType.EMAIL)
                .address("richard_roe@example.com")
                .id("example_endpoint_1")
                .attributes(myMap)
                .user(richardRoe)
                .build();

            List<String> myListMary = new ArrayList<String>();
            myListMary.add("cooking");
            myListMary.add("politics");
            myListMary.add("finance");

            Map myMapMary = new HashMap<String, List>();
            myMapMary.put("interests", myListMary);

            List<String> myNameMary = new ArrayList<String>();
            myNameMary.add("Mary ");
            myNameMary.add("Major");

            Map maryName = new HashMap<String, List>();
            myMapMary.put("name",myNameMary );

            EndpointUser maryMajor = EndpointUser.builder()
                .userId("example_user_2")
                .userAttributes(maryName)
                .build();

            // Create an EndpointBatchItem object for Mary Major
            EndpointBatchItem maryMajorsSmsEndpoint = EndpointBatchItem.builder()
                .channelType(ChannelType.SMS)
                .address("+16145550100")
                .id("example_endpoint_2")
                .attributes(myMapMary)
                .user(maryMajor)
                .build();

            // Adds multiple endpoint definitions to a single request object.
            EndpointBatchRequest endpointList = EndpointBatchRequest.builder()
                .item( richardRoesEmailEndpoint)
                .item( maryMajorsSmsEndpoint)
                .build();

            // Create the UpdateEndpointsBatchRequest
            UpdateEndpointsBatchRequest batchRequest = 
UpdateEndpointsBatchRequest.builder()
                .applicationId(applicationId)
```

```
                 .endpointBatchRequest(endpointList)
                 .build();

             //  Updates the endpoints with Amazon Pinpoint
             UpdateEndpointsBatchResponse result =
 pinpoint.updateEndpointsBatch(batchRequest);
             System.out.format("Update endpoints batch result: %s\n",
                 result.messageBody().message());

      } catch (PinpointException e) {
         System.err.println(e.awsErrorDetails().errorMessage());
         System.exit(1);
     }
```

For the full SDK example, see AddExampleEndpoints.java on GitHub.

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": {
                    "FirstName": "Richard",
                    "LastName": "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": {
                    "FirstName": "Mary",
                    "LastName": "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 quota increase request to AWS Support. For more information, see Requesting a quota increase (p. 384).

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. 355).

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 or double quote in a value, enclose the value in double quotes, as in `"aaa,bbb"`.

Line breaks are not supported within a value in the CSV.

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 software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.pinpoint.PinpointClient;
import software.amazon.awssdk.services.pinpoint.model.DeleteEndpointRequest;
import software.amazon.awssdk.services.pinpoint.model.DeleteEndpointResponse;
import software.amazon.awssdk.services.pinpoint.model.PinpointException;
```

```
    public static void deletePinEncpoint(PinpointClient pinpoint, String appId, String
 endpointId ) {

        try {

            DeleteEndpointRequest appRequest = DeleteEndpointRequest.builder()
                    .applicationId(appId)
                    .endpointId(endpointId)
                    .build();

            DeleteEndpointResponse result = pinpoint.deleteEndpoint(appRequest);
            String id = result.endpointResponse().id();
            System.out.println("The deleted endpoint id  " + id);
        } catch (PinpointException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        System.out.println("Done");
```

For the full SDK example, see DeleteEndpoint.java on GitHub.

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. 115).

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. 143).

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",
                "FirstName": "Wang",
                "LastName": "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.google.gson.FieldNamingPolicy;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.pinpoint.PinpointClient;
import software.amazon.awssdk.services.pinpoint.model.EndpointResponse;
import software.amazon.awssdk.services.pinpoint.model.GetEndpointResponse;
import software.amazon.awssdk.services.pinpoint.model.PinpointException;
import software.amazon.awssdk.services.pinpoint.model.GetEndpointRequest;
```

```
    public static void lookupPinpointEndpoint(PinpointClient pinpoint, String appId,
 String endpoint ) {

        try {
            GetEndpointRequest appRequest = GetEndpointRequest.builder()
                    .applicationId(appId)
                    .endpointId(endpoint)
                    .build();

            GetEndpointResponse result = pinpoint.getEndpoint(appRequest);
            EndpointResponse endResponse = result.endpointResponse();

            // 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(endResponse);
            System.out.println(endpointJson);

        } catch (PinpointException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        System.out.println("Done");
```

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

For the full SDK example, see LookupEndpoint.java on GitHub.

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. 356).

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 software.amazon.awssdk.core.ResponseBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.pinpoint.PinpointClient;
import software.amazon.awssdk.services.pinpoint.model.ExportJobRequest;
import software.amazon.awssdk.services.pinpoint.model.PinpointException;
import software.amazon.awssdk.services.pinpoint.model.CreateExportJobRequest;
import software.amazon.awssdk.services.pinpoint.model.CreateExportJobResponse;
import software.amazon.awssdk.services.pinpoint.model.GetExportJobResponse;
import software.amazon.awssdk.services.pinpoint.model.GetExportJobRequest;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.GetObjectRequest;
import software.amazon.awssdk.services.s3.model.ListObjectsV2Request;
import software.amazon.awssdk.services.s3.model.ListObjectsV2Response;
import software.amazon.awssdk.services.s3.model.S3Object;
import software.amazon.awssdk.services.s3.model.GetObjectResponse;
import software.amazon.awssdk.services.s3.model.S3Exception;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import java.util.concurrent.TimeUnit;
import java.util.stream.Collectors;
```

```
    public static void exportAllEndpoints(PinpointClient pinpoint,
                                          S3Client s3Client,
                                          String applicationId,
                                          String s3BucketName,
                                          String path,
                                          String iamExportRoleArn) {
```

```
        try {

            List<String> objectKeys = exportEndpointsToS3(pinpoint, s3Client,
 s3BucketName, iamExportRoleArn, applicationId);
            List<String> endpointFileKeys = objectKeys.stream().filter(o ->
 o.endsWith(".gz")).collect(Collectors.toList());
            downloadFromS3(s3Client, path, s3BucketName, endpointFileKeys);

        } catch ( PinpointException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static List<String> exportEndpointsToS3(PinpointClient pinpoint, S3Client
 s3Client, String s3BucketName, String iamExportRoleArn, String applicationId) {

        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 + "/";
        List<String> objectKeys = new ArrayList<>();
        String key ="" ;

        try {
            // Defines the export job that Amazon Pinpoint runs
            ExportJobRequest jobRequest = ExportJobRequest.builder()
                    .roleArn(iamExportRoleArn)
                    .s3UrlPrefix(s3UrlPrefix)
                    .build();

            CreateExportJobRequest exportJobRequest =  CreateExportJobRequest.builder()
                    .applicationId(applicationId)
                    .exportJobRequest(jobRequest)
                    .build();

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

            CreateExportJobResponse exportResult =
 pinpoint.createExportJob(exportJobRequest);
            String jobId = exportResult.exportJobResponse().id();
            System.out.println(jobId);
            printExportJobStatus(pinpoint, applicationId, jobId);

            ListObjectsV2Request v2Request = ListObjectsV2Request.builder()
                    .bucket(s3BucketName)
                    .prefix(endpointsKeyPrefix)
                    .build();

            // Create a list of object keys
            ListObjectsV2Response v2Response = s3Client.listObjectsV2(v2Request);
            List<S3Object> objects = v2Response.contents();
            for (S3Object object: objects) {
                key = object.key();
                objectKeys.add(key);
            }

            return objectKeys;

        } catch ( PinpointException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return null;
    }

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

        GetExportJobResponse getExportJobResult;
        String status = "";

        try {
            // Checks the job status until the job completes or fails
            GetExportJobRequest exportJobRequest = GetExportJobRequest.builder()
                    .jobId(jobId)
                    .applicationId(applicationId)
                    .build();

            do {
                getExportJobResult = pinpointClient.getExportJob(exportJobRequest);
                status =
getExportJobResult.exportJobResponse().jobStatus().toString().toUpperCase();
                System.out.format("Export job %s . . .\n", status);
                TimeUnit.SECONDS.sleep(3);

            } while (!status.equals("COMPLETED") && !status.equals("FAILED"));

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

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

    // Downloads files from an Amazon S3 bucket and writes them to the path location
    public static void downloadFromS3(S3Client s3Client, String path, String
s3BucketName, List<String> objectKeys) {

        try {
            for (String key : objectKeys) {

                GetObjectRequest objectRequest = GetObjectRequest.builder()
                        .bucket(s3BucketName)
                        .key(key)
                        .build();

                ResponseBytes<GetObjectResponse> objectBytes =
s3Client.getObjectAsBytes(objectRequest);
                byte[] data = objectBytes.asByteArray();

                // Write the data to a local file
                String fileSuffix = new SimpleDateFormat("yyyyMMddHHmmss").format(new
Date());
                path = path+fileSuffix+".gz";
                File myFile = new File(path );
                OutputStream os = new FileOutputStream(myFile);
                os.write(data);
            }

            System.out.println("Download finished.");
        } catch (S3Exception | NullPointerException | IOException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
```

For the full SDK example, see ExportEndpoints.java on GitHub.

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. 143). 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 software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.pinpoint.PinpointClient;
import software.amazon.awssdk.services.pinpoint.model.EndpointResponse;
import software.amazon.awssdk.services.pinpoint.model.GetUserEndpointsRequest;
import software.amazon.awssdk.services.pinpoint.model.GetUserEndpointsResponse;
import software.amazon.awssdk.services.pinpoint.model.PinpointException;
import java.util.List;
```

```
    public static void listAllEndpoints(PinpointClient pinpoint,
                                        String applicationId,
                                       String userId ) {

        try {
            GetUserEndpointsRequest endpointsRequest = GetUserEndpointsRequest.builder()
                    .userId(userId)
                    .applicationId(applicationId)
                    .build();

            GetUserEndpointsResponse response =
 pinpoint.getUserEndpoints(endpointsRequest);
            List<EndpointResponse> endpoints = response.endpointsResponse().item();

            // Display the results
            for (EndpointResponse endpoint: endpoints) {
                System.out.println("The channel type is: "+endpoint.channelType());
                System.out.println("The address is  "+endpoint.address());
            }

        } catch ( PinpointException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
```

For the full SDK example, see ListEndpointIs.java on GitHub.

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 30 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 software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.pinpoint.PinpointClient;
import software.amazon.awssdk.services.pinpoint.model.EndpointResponse;
import software.amazon.awssdk.services.pinpoint.model.EndpointRequest;
import software.amazon.awssdk.services.pinpoint.model.UpdateEndpointRequest;
import software.amazon.awssdk.services.pinpoint.model.UpdateEndpointResponse;
import software.amazon.awssdk.services.pinpoint.model.GetEndpointRequest;
import software.amazon.awssdk.services.pinpoint.model.GetEndpointResponse;
import software.amazon.awssdk.services.pinpoint.model.PinpointException;
import software.amazon.awssdk.services.pinpoint.model.EndpointDemographic;
import software.amazon.awssdk.services.pinpoint.model.EndpointLocation;
import software.amazon.awssdk.services.pinpoint.model.EndpointUser;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.List;
import java.util.UUID;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
import java.util.Date;
```

```
    public static EndpointResponse createEndpoint(PinpointClient client, String appId) {
        String endpointId = UUID.randomUUID().toString();
        System.out.println("Endpoint ID: " + endpointId);
        try {

            EndpointRequest endpointRequest = createEndpointRequestData();

            UpdateEndpointRequest updateEndpointRequest = UpdateEndpointRequest.builder()
                    .applicationId(appId)
                    .endpointId(endpointId)
                    .endpointRequest(endpointRequest)
                    .build();

            UpdateEndpointResponse updateEndpointResponse =
client.updateEndpoint(updateEndpointRequest);
            System.out.println("Update Endpoint Response: " +
updateEndpointResponse.messageBody());

            GetEndpointRequest getEndpointRequest = GetEndpointRequest.builder()
                    .applicationId(appId)
                    .endpointId(endpointId)
                    .build();
            GetEndpointResponse getEndpointResponse =
client.getEndpoint(getEndpointRequest);

            System.out.println(getEndpointResponse.endpointResponse().address());
            System.out.println(getEndpointResponse.endpointResponse().channelType());
            System.out.println(getEndpointResponse.endpointResponse().applicationId());
            System.out.println(getEndpointResponse.endpointResponse().endpointStatus());
            System.out.println(getEndpointResponse.endpointResponse().requestId());
            System.out.println(getEndpointResponse.endpointResponse().user());

            return getEndpointResponse.endpointResponse();

        } catch (PinpointException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return null;
    }

    private static EndpointRequest createEndpointRequestData() {

        try {
            List<String> favoriteTeams = new ArrayList<>();
            favoriteTeams.add("Lakers");
            favoriteTeams.add("Warriors");
            HashMap<String, List<String>> customAttributes = new HashMap<>();
            customAttributes.put("team", favoriteTeams);

            EndpointDemographic demographic = EndpointDemographic.builder()
                    .appVersion("1.0")
                    .make("apple")
                    .model("iPhone")
                    .modelVersion("7")
                    .platform("ios")
                    .platformVersion("10.1.1")
                    .timezone("America/Los_Angeles")
                    .build();

            EndpointLocation location = EndpointLocation.builder()
                    .city("Los Angeles")
                    .country("US")
                    .latitude(34.0)
                    .longitude(-118.2)
                    .postalCode("90068")
                    .region("CA")
                    .build();
            Map<String,Double> metrics = new HashMap<>();
            metrics.put("health", 100.00);
            metrics.put("luck", 75.00);

            EndpointUser user = EndpointUser.builder()
                    .userId(UUID.randomUUID().toString())
                    .build();

            DateFormat df = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm'Z'"); // Quoted "Z" to
indicate UTC, no timezone offset
            String nowAsISO = df.format(new Date());

            EndpointRequest endpointRequest = EndpointRequest.builder()
                    .address(UUID.randomUUID().toString())
                    .attributes(customAttributes)
                    .channelType("APNS")
                    .demographic(demographic)
                    .effectiveDate(nowAsISO)
                    .location(location)
                    .metrics(metrics)
                    .optOut("NONE")
                    .requestId(UUID.randomUUID().toString())
                    .user(user)
                    .build();

            return endpointRequest;

        } catch (PinpointException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }

        return null;
    }
```

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

```
Segment ID: 09cb2967a82b4a2fbab38fead8d1f4c4
```

For the full SDK example, see CreateSegment.java on GitHub.

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. 355). 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 software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.pinpoint.PinpointClient;
import software.amazon.awssdk.services.pinpoint.model.CreateImportJobRequest;
import software.amazon.awssdk.services.pinpoint.model.ImportJobResponse;
import software.amazon.awssdk.services.pinpoint.model.ImportJobRequest;
import software.amazon.awssdk.services.pinpoint.model.Format;
import software.amazon.awssdk.services.pinpoint.model.CreateImportJobResponse;
import software.amazon.awssdk.services.pinpoint.model.PinpointException;
```

```
    public static ImportJobResponse createImportSegment(PinpointClient client,
                                                         String appId,
                                                         String bucket,
                                                         String key,
                                                         String roleArn) {

        try {
             ImportJobRequest importRequest = ImportJobRequest.builder()
                    .defineSegment(true)
                    .registerEndpoints(true)
                    .roleArn(roleArn)
                    .format(Format.JSON)
                    .s3Url("s3://" + bucket + "/" + key)
                    .build();

            CreateImportJobRequest jobRequest = CreateImportJobRequest.builder()
                    .importJobRequest(importRequest)
                    .applicationId(appId)
                    .build();

            CreateImportJobResponse jobResponse = client.createImportJob(jobRequest);

            return jobResponse.importJobResponse();

        } catch (PinpointException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return null;
    }
```

For the full SDK example, see ImportingSegments.java on GitHub.

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 the moment when Amazon Pinpoint sends 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 the AWS Lambda Developer Guide.

To assign a Lambda function to a campaign, you define the campaign's `CampaignHook` settings by using the Campaign resource in the Amazon Pinpoint API. These settings include the Lambda function name. They also include the `CampaignHook` mode, which specifies 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 sends 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 when campaign messages are sent. Filtering segments enables you to engage more narrowly defined subsets of your segments. Adding or updating endpoint attributes also enables you to make new data available for message variables.

> **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 in Amazon Pinpoint (p. 199).

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}
    . . .
  }
}
```

AWS Lambda passes the event data to your function code. The event data provides the following attributes:

- `MessageConfiguration` – Has the same structure as the `DirectMessageConfiguration` object of 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 campaign 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, in ISO 8601 format, when the campaign's messages will be delivered.
- `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 learn how to create a Lambda function, see Getting started in the *AWS Lambda Developer Guide*. When you create your function, remember that message delivery fails in the following conditions:

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

Amazon Pinpoint only accepts endpoint definitions in the function's return value. The function can't 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.

Optionally, you can include the `TitleOverride` and `BodyOverride` attributes in the endpoint definitions that you return.

Note
When you use this solution to send messages, Amazon Pinpoint honors the `TitleOverride` and `BodyOverride` attributes only for endpoints where the value of the `ChannelType` attribute is one of the following: `ADM`, `APNS`, `APNS_SANDBOX`, `APNS_VOIP`, `APNS_VOIP_SANDBOX`, `BAIDU`, `GCM`, or `SMS`.
Amazon Pinpoint **doesn't** honor these attributes for endpoints where the value of the `ChannelType` attribute is `EMAIL`.

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 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 for a specific campaign (*`campaign-id`*):

```
{
  "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.

To write a more generic policy, use a multicharacter match wildcard (*). For example, you can use the following `Condition` block to allow any campaign in a specific Amazon Pinpoint project (*`application-id`*) to invoke the function:

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

If you want the Lambda function to be the default function that's used by all the campaigns for a project, we recommend that you configure the `Condition` block for the policy in the preceding way. For information about setting a Lambda function as the default for all campaigns in a project, see *Assigning a Lambda Function to a Campaign* later in this topic.

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 for a specific campaign, use the Lambda `add-permission` command, as shown in 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 **All projects** page. The console shows a **Project 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 that was 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 software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.pinpoint.PinpointClient;
import software.amazon.awssdk.services.pinpoint.model.CampaignResponse;
import software.amazon.awssdk.services.pinpoint.model.Message;
import software.amazon.awssdk.services.pinpoint.model.Schedule;
import software.amazon.awssdk.services.pinpoint.model.Action;
import software.amazon.awssdk.services.pinpoint.model.MessageConfiguration;
import software.amazon.awssdk.services.pinpoint.model.WriteCampaignRequest;
import software.amazon.awssdk.services.pinpoint.model.CreateCampaignResponse;
import software.amazon.awssdk.services.pinpoint.model.CreateCampaignRequest;
import software.amazon.awssdk.services.pinpoint.model.PinpointException;
```

```
    public static void createPinCampaign(PinpointClient pinpoint, String appId, String
 segmentId) {


        CampaignResponse result = createCampaign(pinpoint, appId, segmentId);
        System.out.println("Campaign " + result.name() + " created.");
        System.out.println(result.description());

    }


    public static CampaignResponse createCampaign(PinpointClient client, String appID,
 String segmentID) {

        try {
            Schedule schedule = Schedule.builder()
                    .startTime("IMMEDIATE")
                    .build();

            Message defaultMessage = Message.builder()
                    .action(Action.OPEN_APP)
                    .body("My message body.")
                    .title("My message title.")
                    .build();

            MessageConfiguration messageConfiguration = MessageConfiguration.builder()
                    .defaultMessage(defaultMessage)
                    .build();

            WriteCampaignRequest request = WriteCampaignRequest.builder()
                    .description("My description")
                    .schedule(schedule)
                    .name("MyCampaign")
                    .segmentId(segmentID)
                    .messageConfiguration(messageConfiguration)
                    .build();

            CreateCampaignResponse result = client.createCampaign(
                    CreateCampaignRequest.builder()
                            .applicationId(appID)
                            .writeCampaignRequest(request).build()
            );

            System.out.println("Campaign ID: " + result.campaignResponse().id());

            return result.campaignResponse();

        } catch (PinpointException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }

        return null;
    }
```

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

```
Campaign ID: b1c3de717aea4408a75bb3287a906b46
```

For the full SDK example, see CreateCampaign.java on GitHub.

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
```

Validating phone numbers in Amazon Pinpoint

Amazon Pinpoint includes a phone number validation service that you can use to determine if a phone number is valid, and to obtain additional information about the phone number itself. For example, when you use the phone number validation service, it returns the following information:

- The phone number in E.164 format.
- The phone number type (such as mobile, landline, or VoIP).
- The city and country where the phone number is based.
- The service provider that's associated with the phone number.

There is an additional charge for using the phone number validation service. For more information, see Amazon Pinpoint pricing.

Phone number validation use cases

You can use the phone number validation service to enable several use cases, including the following:

- **Verifying phone numbers provided on a web form** – If you use web-based forms to collect contact information for your customers, you validate the phone numbers that customers provide before submitting the form. Use your website's backend to validate the number by using the Amazon Pinpoint API. The API response states whether the number is invalid—for example, if the phone number is formatted incorrectly. If you determine that the phone number that the customer provided is invalid, your web form can prompt the customer to provide a different number.
- **Cleansing your existing contact database** – If you have a database of customer phone numbers, you can validate each phone number, and then update your database based on your findings. For example, if you find endpoints with phone numbers that aren't capable of receiving SMS messages, you can change the `ChannelType` property for the endpoint from `SMS` to `VOICE`.
- **Choosing the right channel before you send a message** – If you intend to send an SMS message but you determine that the destination number is invalid, you can send a message to the recipient through a different channel. For example, if the endpoint isn't able to receive SMS messages, you can send a voice message instead.

Using the phone number validation service

To validate a number, send an HTTP POST request to the `/v1/phone/number/validate/` URI in the Amazon Pinpoint API. The request in the following example includes the required HTTP headers and a simple JSON body. The body specifies the number to validate with the `PhoneNumber` parameter.

```
POST /v1/phone/number/validate/ HTTP/1.1
Host: pinpoint.us-east-1.amazonaws.com
Content-Type: application/json
X-Amz-Date: 20190805T031042Z
Authorization: AWS4-HMAC-SHA256 Credential=AKIAIOSFODNN7EXAMPLE/20190805/us-east-1/
mobiletargeting/aws4_request, SignedHeaders=content-length;content-type;host;x-amz-date,
 Signature=39df573629ddb283aea1fa2f7eef4106c0fb4826edf72e9934f03cf77example
```

```
Cache-Control: no-cache

{
 "PhoneNumber": "+12065550100"
}
```

For information about supported methods, parameters, and schemas, see Phone number validate in the *Amazon Pinpoint API Reference*.

You can also use the AWS CLI to quickly validate individual phone numbers.

To use the phone number validation service by using the AWS CLI

- At the command line, enter the following command:

  ```
  aws pinpoint phone-number-validate --number-validate-request PhoneNumber=+442079460881
  ```

  In the preceding command, replace *+442079460881* with the phone number that you want to validate.

  > **Note**
  > When you provide a phone number to the phone number validation service, you should always include the country code. If you don't include the country code, the service might return information for a phone number in a different country.

Phone number validation responses

The information that the phone number validation service provides varies slightly based on the data that's available for the phone number that you provide. This section contains examples of the responses that the phone number validation service returns.

> **Note**
> The data that's provided by the phone number validation service is based on information provided by telecommunication providers and other entities around the world. Providers in some countries might update this information less frequently than providers in other countries do. For example, if you issue a request to validate a mobile phone number, and the number that you provided was ported from one mobile carrier to another, the response from the phone number validation service might include the name of the original carrier, as opposed to the current one.

Valid mobile phone numbers

When you send a request to the phone number validation service, and the phone number is a valid mobile phone number, it returns information that resembles the following example:

```
{
    "NumberValidateResponse": {
        "Carrier": "ExampleCorp Mobile",
        "City": "Seattle",
        "CleansedPhoneNumberE164": "+12065550142",
        "CleansedPhoneNumberNational": "2065550142",
        "Country": "United States",
        "CountryCodeIso2": "US",
        "CountryCodeNumeric": "1",
        "OriginalPhoneNumber": "+12065550142",
        "PhoneType": "MOBILE",
        "PhoneTypeCode": 0,
        "Timezone": "America/Los_Angeles",
```

```
        "ZipCode": "98101"
    }
}
```

Valid landline phone numbers

If your request contains a valid landline phone number, the phone number validation service returns information that resembles the following example:

```
{
    "CountryCodeIso2": "US",
    "CountryCodeNumeric": "1",
    "Country": "United States",
    "City": "Santa Clara",
    "ZipCode": "95037",
    "Timezone": "America/Los_Angeles",
    "CleansedPhoneNumberNational": "4085550101",
    "CleansedPhoneNumberE164": "14085550101",
    "Carrier": "AnyCompany",
    "PhoneTypeCode": 1,
    "PhoneType": "LANDLINE",
    "OriginalPhoneNumber": "+14085550101"
}
```

Valid VoIP phone numbers

If your request contains a valid Voice over Internet Protocol (VoIP) phone number, the phone number validation service returns information that resembles the following example:

```
{
    "NumberValidateResponse": {
        "Carrier": "ExampleCorp",
        "City": "Countrywide",
        "CleansedPhoneNumberE164": "+441514960001",
        "CleansedPhoneNumberNational": "1514960001",
        "Country": "United Kingdom",
        "CountryCodeIso2": "GB",
        "CountryCodeNumeric": "44",
        "OriginalPhoneNumber": "+441514960001",
        "PhoneType": "VOIP",
        "PhoneTypeCode": 2
    }
}
```

Invalid phone numbers

If your request contains an invalid phone number, the phone number validation service returns information that resembles the following example:

```
{
    "NumberValidateResponse": {
        "CleansedPhoneNumberE164": "+44163296076",
        "CleansedPhoneNumberNational": "163296076",
        "Country": "United Kingdom",
        "CountryCodeIso2": "GB",
        "CountryCodeNumeric": "44",
        "OriginalPhoneNumber": "+440163296076",
        "PhoneType": "PREPAID",
        "PhoneTypeCode": 3
    }
}
```

Note that the `PhoneType` property in this response indicates that this phone number is `INVALID`, and that it doesn't include information about the carrier or location associated with the phone number. You should avoid sending SMS or voice messages to phone numbers where the `PhoneType` is `INVALID`, because these numbers are unlikely to belong to actual recipients.

Other phone numbers

Occasionally, the response from the phone number validation service includes a `PhoneType` value of `OTHER`. The service might return this kind of response in the following situations:

- The phone number is a toll-free (freephone) number.
- The phone number is reserved for use in TV shows and movies, such as North American phone numbers that begin with *555*.
- The phone number includes an area code that is not currently in use, such as the *999* area code in North America.
- The phone number is reserved for some other purpose.

The following example shows the response that the phone number validation services provides when your request includes a fictitious North American phone number:

```
{
    "NumberValidateResponse": {
        "Carrier": "Multiple OCN Listing",
        "CleansedPhoneNumberE164": "+14255550199",
        "CleansedPhoneNumberNational": "4255550199",
        "Country": "United States",
        "CountryCodeIso2": "US",
        "CountryCodeNumeric": "1",
        "OriginalPhoneNumber": "+14255550199",
        "PhoneType": "OTHER",
        "PhoneTypeCode": 4,
        "Timezone": "America/Los_Angeles"
    }
}
```

Prepaid phone numbers

If your request contains a valid prepaid phone number, the phone number validation service returns information that resembles the following example:

```
{
    "NumberValidateResponse": {
        "Carrier": "ExampleCorp",
        "City": "Countrywide",
        "CleansedPhoneNumberE164": "+14255550199",
        "CleansedPhoneNumberNational": "4255550199",
        "Country": "United States",
        "CountryCodeIso2": "US",
        "CountryCodeNumeric": "1",
        "OriginalPhoneNumber": "+14255550199",
        "PhoneType": "PREPAID",
        "PhoneTypeCode": 5
    }
}
```

For more information about the information that's contained in these responses, see Phone number validate in the *Amazon Pinpoint API Reference*.

Sending transactional messages from your apps

You can use the Amazon Pinpoint API and the AWS SDKs to send *transactional messages* directly from your apps. Transactional messages are messages that you send to specific recipients, as opposed to messages that you send to segments. There are several reasons that you might want to send transactional messages rather than campaign-based messages. For example, you can send an order confirmation by email when a customer places an order. You could also send a one-time password by SMS or voice that a customer can use to complete the process of creating an account for your service.

This section includes example code in several programming languages that you can use to start sending transactional emails, SMS messages, and voice messages.

Topics in this section:

Send transactional email messages

This section provides complete code samples that you can use to send transactional email messages through Amazon Pinpoint. There are two methods that you can use to send transactional email messages:

- By using the SendMessages operation in the Amazon Pinpoint API (p. 169): You can use the `SendMessages` operation in the Amazon Pinpoint API to send messages in all of the channels that Amazon Pinpoint supports, including the push notification, SMS, voice, and email channels.

  The advantage of using this operation is that the request syntax for sending messages is very similar across all channels. This makes it easier to repurpose your existing code. The `SendMessages` operation also lets you to substitute content in your email messages, and lets you send email to Amazon Pinpoint endpoint IDs rather than to specific email addresses.
- By using the Amazon Pinpoint SMTP interface (p. 176): You can use the Amazon Pinpoint SMTP interface to send email messages only.

  The advantage to using the SMTP interface is that you can use email-sending applications and libraries to send email. For example, if you use a ticketing system that sends emails to your customers, you can use the SMTP interface to configure the system to send emails from your domain. You can also use email-sending libraries in several programming languages to send email through the SMTP interface.

This section includes example code in several programming languages that you can use to start sending transactional emails.

Topics in this section:

Choosing an email-sending method

The best method to use for sending transactional email depends on your use case. For example, if you need to send email by using a third-party application, or if there isn't an AWS SDK available for your programming language, you might have to use the SMTP interface. If you want to send messages in other channels that Amazon Pinpoint supports, and you want to use consistent code for making those requests, you should use the `SendMessages` operation in the Amazon Pinpoint API.

Choosing between Amazon Pinpoint and Amazon Simple Email Service (SES)

If you send a large number of transactional emails, such as purchase confirmations or password reset messages, consider using Amazon SES. Amazon SES has an API and an SMTP interface, both of which are well suited to sending email from your applications or services. It also offers additional email features, including email receiving capabilities, configuration sets, and sending authorization capabilities.

Amazon SES also includes an SMTP interface that you can integrate with your existing third-party applications, including customer relationship management (CRM) services such as Salesforce. For more information about sending email using Amazon SES, Amazon Simple Email Service Developer Guide for more information.

Send email by using the Amazon Pinpoint API

This section contains complete code examples that you can use to send email through the Amazon Pinpoint API by using an AWS SDK.

C#

Use this example to send email by using the AWS SDK for .NET. This example assumes that you've already installed and configured the AWS SDK for .NET. For more information, see Getting started with the AWS SDK for .NET in the *AWS SDK for .NET Developer Guide*.

This example assumes that you're using a shared credentials file to specify the Access Key and Secret Access Key for an existing IAM user. For more information, see Configuring AWS credentials in the *AWS SDK for .NET Developer Guide*.

This code example was tested using the AWS SDK for .NET version 3.3.29.13 and .NET Core runtime version 2.1.2.

```
using System;
using System.Collections.Generic;
using Amazon;
using Amazon.Pinpoint;
using Amazon.Pinpoint.Model;

namespace PinpointEmailSendMessageAPI
{
    class MainClass
    {
        // The AWS Region that you want to use to send the email. For a list of
        // AWS Regions where the Amazon Pinpoint API is available, see
        // https://docs.aws.amazon.com/pinpoint/latest/apireference/
        static string region = "us-west-2";

        // The "From" address. This address has to be verified in Amazon Pinpoint
        // in the region you're using to send email.
        static string senderAddress = "sender@example.com";
        // The address on the "To" line. If your Amazon Pinpoint account is in
        // the sandbox, this address also has to be verified.
        static string toAddress = "recipient@example.com";

        // The Amazon Pinpoint project/application ID to use when you send this
 message.
        // Make sure that the SMS channel is enabled for the project or application
        // that you choose.
        static string appId = "ce796be37f32f178af652b26eexample";

        // The subject line of the email.
        static string subject = "Amazon Pinpoint Email test";

        // The body of the email for recipients whose email clients don't
        // support HTML content.
        static string textBody = @"Amazon Pinpoint Email Test (.NET)
---------------------------------
This email was sent using the Amazon Pinpoint API using the AWS SDK for .NET.";

        // The body of the email for recipients whose email clients support
        // HTML content.
        static string htmlBody = @"<html>
<head></head>
<body>
  <h1>Amazon Pinpoint Email Test (AWS SDK for .NET)</h1>
  <p>This email was sent using the
    <a href='https://aws.amazon.com/pinpoint/'>Amazon Pinpoint</a> API
    using the <a href='https://aws.amazon.com/sdk-for-net/'>
      AWS SDK for .NET</a>.</p>
</body>
</html>";

        // The character encoding the you want to use for the subject line and
        // message body of the email.
        static string charset = "UTF-8";

        public static void Main(string[] args)
        {
            using (var client = new
 AmazonPinpointClient(RegionEndpoint.GetBySystemName(region)))
            {
                var sendRequest = new SendMessagesRequest
                {
                    ApplicationId = appId,
                    MessageRequest = new MessageRequest
                    {
                        Addresses = new Dictionary<string, AddressConfiguration>
                        {
                            {
                                toAddress,
                                new AddressConfiguration
                                {
                                    ChannelType = "EMAIL"
                                }
                            }
                        },
                        MessageConfiguration = new DirectMessageConfiguration
                        {
                            EmailMessage = new EmailMessage
                            {
                                FromAddress = senderAddress,
                                SimpleEmail = new SimpleEmail
                                {
                                    HtmlPart = new SimpleEmailPart
                                    {
                                        Charset = charset,
                                        Data = htmlBody
                                    },
                                    TextPart = new SimpleEmailPart
                                    {
                                        Charset = charset,
                                        Data = textBody
                                    },
                                    Subject = new SimpleEmailPart
                                    {
                                        Charset = charset,
                                        Data = subject
                                    }
                                }
                            }
                        }
                    }
                };
                try
                {
                    Console.WriteLine("Sending message...");
                    SendMessagesResponse response = client.SendMessages(sendRequest);
                    Console.WriteLine("Message sent!");
                }
                catch (Exception ex)
                {
                    Console.WriteLine("The message wasn't sent. Error message: " +
 ex.Message);
                }
            }
        }
    }
}
```

Java

Use this example to send email by using the AWS SDK for Java. This example assumes that you've already installed and configured the AWS SDK for Java 2.x. For more information, see Getting started in the *AWS SDK for Java 2.x Developer Guide*.

This example assumes that you're using a shared credentials file to specify the Access Key and Secret Access Key for an existing IAM user. For more information, see Set default credentials and Region in the *AWS SDK for Java Developer Guide*.

This code example was tested using the AWS SDK for Java version 2.3.1 and OpenJDK version 11.0.1.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.pinpoint.PinpointClient;
import software.amazon.awssdk.services.pinpoint.model.AddressConfiguration;
import software.amazon.awssdk.services.pinpoint.model.ChannelType;
import software.amazon.awssdk.services.pinpoint.model.SimpleEmailPart;
import software.amazon.awssdk.services.pinpoint.model.SimpleEmail;
import software.amazon.awssdk.services.pinpoint.model.EmailMessage;
import software.amazon.awssdk.services.pinpoint.model.DirectMessageConfiguration;
import software.amazon.awssdk.services.pinpoint.model.MessageRequest;
import software.amazon.awssdk.services.pinpoint.model.SendMessagesRequest;
import software.amazon.awssdk.services.pinpoint.model.PinpointException;
import java.util.HashMap;
import java.util.Map;
```

```
    public static void sendEmail(PinpointClient pinpoint,
                                 String subject,
                                  String appId,
                                  String senderAddress,
                                  String toAddress) {

        try {

            Map<String,AddressConfiguration> addressMap = new
HashMap<String,AddressConfiguration>();
            AddressConfiguration configuration =  AddressConfiguration.builder()
                    .channelType(ChannelType.EMAIL)
                    .build();

            addressMap.put(toAddress, configuration);
            SimpleEmailPart emailPart = SimpleEmailPart.builder()
                    .data(htmlBody)
                    .charset(charset)
                    .build() ;

            SimpleEmailPart subjectPart = SimpleEmailPart.builder()
                    .data(subject)
                    .charset(charset)
                    .build() ;

            SimpleEmail simpleEmail = SimpleEmail.builder()
                    .htmlPart(emailPart)
                    .subject(subjectPart)
                    .build();

            EmailMessage emailMessage =  EmailMessage.builder()
                    .body(htmlBody)
                    .fromAddress(senderAddress)
                    .simpleEmail(simpleEmail)
                    .build();

            DirectMessageConfiguration directMessageConfiguration =
DirectMessageConfiguration.builder()
                    .emailMessage(emailMessage)
                    .build();

            MessageRequest messageRequest = MessageRequest.builder()
                    .addresses(addressMap)
                    .messageConfiguration(directMessageConfiguration)
                    .build();

            SendMessagesRequest messagesRequest = SendMessagesRequest.builder()
                    .applicationId(appId)
                    .messageRequest(messageRequest)
                    .build();

            pinpoint.sendMessages(messagesRequest);


        } catch (PinpointException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
```

For the full SDK example, see SendEmailMessage.java on GitHub.

JavaScript (Node.js)

Use this example to send email by using the AWS SDK for JavaScript in Node.js. This example assumes that you've already installed and configured the SDK for JavaScript in Node.js. For more information, see Getting started in the *AWS SDK for JavaScript in Node.js Developer Guide*.

This example assumes that you're using a shared credentials file to specify the Access Key and Secret Access Key for an existing IAM user. For more information, see Setting credentials in the *AWS SDK for JavaScript in Node.js Developer Guide*.

This code example was tested using the SDK for JavaScript in Node.js version 2.388.0 and Node.js version 11.7.0.

```
'use strict';

const AWS = require('aws-sdk');

// The AWS Region that you want to use to send the email. For a list of
// AWS Regions where the Amazon Pinpoint API is available, see
// https://docs.aws.amazon.com/pinpoint/latest/apireference/
const aws_region = "us-west-2"

// The "From" address. This address has to be verified in Amazon Pinpoint
// in the region that you use to send email.
const senderAddress = "sender@example.com";

// The address on the "To" line. If your Amazon Pinpoint account is in
// the sandbox, this address also has to be verified.
var toAddress = "recipient@example.com";

// The Amazon Pinpoint project/application ID to use when you send this message.
// Make sure that the SMS channel is enabled for the project or application
// that you choose.
const appId = "ce796be37f32f178af652b26eexample";

// The subject line of the email.
var subject = "Amazon Pinpoint (AWS SDK for JavaScript in Node.js)";

// The email body for recipients with non-HTML email clients.
var body_text = `Amazon Pinpoint Test (SDK for JavaScript in Node.js)
----------------------------------------------------
This email was sent with Amazon Pinpoint using the AWS SDK for JavaScript in Node.js.
For more information, see https:\/\/aws.amazon.com/sdk-for-node-js/`;

// The body of the email for recipients whose email clients support HTML content.
var body_html = `<html>
<head></head>
<body>
  <h1>Amazon Pinpoint Test (SDK for JavaScript in Node.js)</h1>
  <p>This email was sent with
    <a href='https://aws.amazon.com/pinpoint/'>the Amazon Pinpoint API</a> using the
    <a href='https://aws.amazon.com/sdk-for-node-js/'>
      AWS SDK for JavaScript in Node.js</a>.</p>
</body>
</html>`;

// The character encoding the you want to use for the subject line and
// message body of the email.
var charset = "UTF-8";

// Specify that you're using a shared credentials file.
var credentials = new AWS.SharedIniFileCredentials({profile: 'default'});
AWS.config.credentials = credentials;

// Specify the region.
AWS.config.update({region:aws_region});

//Create a new Pinpoint object.
var pinpoint = new AWS.Pinpoint();
```

```
// Specify the parameters to pass to the API.
var params = {
  ApplicationId: appId,
  MessageRequest: {
    Addresses: {
      [toAddress]:{
        ChannelType: 'EMAIL'
      }
    },
    MessageConfiguration: {
      EmailMessage: {
        FromAddress: senderAddress,
        SimpleEmail: {
          Subject: {
            Charset: charset,
            Data: subject
          },
          HtmlPart: {
            Charset: charset,
            Data: body_html
          },
          TextPart: {
            Charset: charset,
            Data: body_text
          }
        }
      }
    }
  }
};

//Try to send the email.
pinpoint.sendMessages(params, function(err, data) {
  // If something goes wrong, print an error message.
  if(err) {
    console.log(err.message);
  } else {
    console.log("Email sent! Message ID: ", data['MessageResponse']['Result']
[toAddress]['MessageId']);
  }
});
```

Python

Use this example to send email by using the AWS SDK for Python (Boto3). This example assumes that you've already installed and configured the SDK for Python (Boto3). For more information, see Quickstart in the *AWS SDK for Python (Boto3) API Reference*.

This example assumes that you're using a shared credentials file to specify the Access Key and Secret Access Key for an existing IAM user. For more information, see Credentials in the *AWS SDK for Python (Boto3) API Reference*.

This code example was tested using the SDK for Python (Boto3) version 1.9.62 and Python version 3.6.7.

```
import boto3
from botocore.exceptions import ClientError

# The AWS Region that you want to use to send the email. For a list of
# AWS Regions where the Amazon Pinpoint API is available, see
# https://docs.aws.amazon.com/pinpoint/latest/apireference/
AWS_REGION = "us-west-2"
```

```
# The "From" address. This address has to be verified in
# Amazon Pinpoint in the region you're using to send email.
SENDER = "Mary Major <sender@example.com>"

# The addresses on the "To" line. If your Amazon Pinpoint account is in
# the sandbox, these addresses also have to be verified.
TOADDRESS = "recipient@example.com"

# The Amazon Pinpoint project/application ID to use when you send this message.
# Make sure that the email channel is enabled for the project or application
# that you choose.
APPID = "ce796be37f32f178af652b26eexample"

# The subject line of the email.
SUBJECT = "Amazon Pinpoint Test (SDK for Python (Boto 3))"

# The body of the email for recipients whose email clients don't support HTML
# content.
BODY_TEXT = """Amazon Pinpoint Test (SDK for Python)
-------------------------------------
This email was sent with Amazon Pinpoint using the AWS SDK for Python (Boto 3).
For more information, see https:#aws.amazon.com/sdk-for-python/
            """

# The body of the email for recipients whose email clients can display HTML
# content.
BODY_HTML = """<html>
<head></head>
<body>
  <h1>Amazon Pinpoint Test (SDK for Python)</h1>
  <p>This email was sent with
    <a href='https:#aws.amazon.com/pinpoint/'>Amazon Pinpoint</a> using the
    <a href='https:#aws.amazon.com/sdk-for-python/'>
      AWS SDK for Python (Boto 3)</a>.</p>
</body>
</html>
            """

# The character encoding that you want to use for the subject line and message
# body of the email.
CHARSET = "UTF-8"

# Create a new client and specify a region.
client = boto3.client('pinpoint',region_name=AWS_REGION)
try:
    response = client.send_messages(
        ApplicationId=APPID,
        MessageRequest={
            'Addresses': {
                TOADDRESS: {
                     'ChannelType': 'EMAIL'
                }
            },
            'MessageConfiguration': {
                'EmailMessage': {
                    'FromAddress': SENDER,
                    'SimpleEmail': {
                        'Subject': {
                            'Charset': CHARSET,
                            'Data': SUBJECT
                        },
                        'HtmlPart': {
                            'Charset': CHARSET,
                            'Data': BODY_HTML
                        },
                        'TextPart': {
                            'Charset': CHARSET,
                            'Data': BODY_TEXT
                        }
                    }
                }
            }
        }
    )
except ClientError as e:
    print(e.response['Error']['Message'])
else:
    print("Message sent! Message ID: "
            + response['MessageResponse']['Result'][TOADDRESS]['MessageId'])
```

Send email by using the Amazon Pinpoint SMTP interface

This section contains complete code examples that you can use to send email from your apps by using the Amazon Pinpoint SMTP interface. These examples use standard email-sending libraries whenever possible.

These examples assume that you've already created an Amazon Pinpoint SMTP user name and password. For more information, see Obtaining SMTP credentials in the *Amazon Pinpoint User Guide*.

C#

Use this example to send email by using classes in the .NET System.Net.Mail namespace.

```

using System;
using System.Net;
using System.Net.Mail;

namespace PinpointEmailSMTP
{
    class MainClass
    {
        // If you're using Amazon Pinpoint in a region other than US West (Oregon),
        // replace email-smtp.us-west-2.amazonaws.com with the Amazon Pinpoint SMTP
        // endpoint in the appropriate AWS Region.
        static string smtpEndpoint = "email-smtp.us-west-2.amazonaws.com";

        // The port to use when connecting to the SMTP server.
        static int port = 587;

        // Replace sender@example.com with your "From" address.
        // This address must be verified with Amazon Pinpoint.
        static string senderName = "Mary Major";
        static string senderAddress = "sender@example.com";

        // Replace recipient@example.com with a "To" address. If your account
        // is still in the sandbox, this address must be verified.
        static string toAddress = "recipient@example.com";

        // CC and BCC addresses. If your account is in the sandbox, these
        // addresses have to be verified.
        static string ccAddress = "cc-recipient@example.com";
        static string bccAddress = "bcc-recipient@example.com";
        // Replace smtp_username with your Amazon Pinpoint SMTP user name.
        static string smtpUsername = "AKIAIOSFODNN7EXAMPLE";

        // Replace smtp_password with your Amazon Pinpoint SMTP password.
        static string smtpPassword = "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY";

        // (Optional) the name of a configuration set to use for this message.
        static string configurationSet = "ConfigSet";

        // The subject line of the email
        static string subject =
            "Amazon Pinpoint test (SMTP interface accessed using C#)";

        // The body of the email for recipients whose email clients don't
        // support HTML content.
        static AlternateView textBody = AlternateView.
            CreateAlternateViewFromString("Amazon Pinpoint Email Test (.NET)\r\n"
                    + "This email was sent using the Amazon Pinpoint SMTP "
                    + "interface.", null, "text/plain");

        // The body of the email for recipients whose email clients support
        // HTML content.
        static AlternateView htmlBody = AlternateView.
            CreateAlternateViewFromString("<html><head></head><body>"
                    + "<h1>Amazon Pinpoint SMTP Interface Test</h1><p>This "
                    + "email was sent using the "
                    + "<a href='https://aws.amazon.com/pinpoint/'>Amazon Pinpoint"
                    + "</a> SMTP interface.</p></body></html>", null, "text/html");

        // The message tags that you want to apply to the email.
        static string tag0 = "key0=value0";
        static string tag1 = "key1=value1";

        public static void Main(string[] args)
        {
            // Create a new MailMessage object
            MailMessage message = new MailMessage();

            // Add sender and recipient email addresses to the message
            message.From = new MailAddress(senderAddress,senderName);
            message.To.Add(new MailAddress(toAddress));
            message.CC.Add(new MailAddress(ccAddress));
            message.Bcc.Add(new MailAddress(bccAddress));

            // Add the subject line, text body, and HTML body to the message
            message.Subject = subject;
            message.AlternateViews.Add(textBody);
            message.AlternateViews.Add(htmlBody);

            // Add optional headers for configuration set and message tags to the
message
            message.Headers.Add("X-SES-CONFIGURATION-SET", configurationSet);
            message.Headers.Add("X-SES-MESSAGE-TAGS", tag0);
            message.Headers.Add("X-SES-MESSAGE-TAGS", tag1);

            using (var client = new System.Net.Mail.SmtpClient(smtpEndpoint, port))
            {
                // Create a Credentials object for connecting to the SMTP server
                client.Credentials =
                    new NetworkCredential(smtpUsername, smtpPassword);

                client.EnableSsl = true;

                // Send the message
                try
                {
                    Console.WriteLine("Attempting to send email...");
                    client.Send(message);
                    Console.WriteLine("Email sent!");
                }
                // Show an error message if the message can't be sent
                catch (Exception ex)
                {
                    Console.WriteLine("The email wasn't sent.");
                    Console.WriteLine("Error message: " + ex.Message);
                }
            }
        }
    }
}
```

Java

Use this example to send email by using the JavaMail API.

```
import java.util.Properties;

import javax.mail.Message;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;

public class SendEmail {

    // If you're using Amazon Pinpoint in a region other than US West (Oregon),
    // replace email-smtp.us-west-2.amazonaws.com with the Amazon Pinpoint SMTP
    // endpoint in the appropriate AWS Region.
    static final String smtpEndpoint = "email-smtp.us-west-2.amazonaws.com";

    // The port to use when connecting to the SMTP server.
    static final int port = 587;

    // Replace sender@example.com with your "From" address.
    // This address must be verified with Amazon Pinpoint.
    static final String senderName= "Mary Major";
    static final String senderAddress = "sender@example.com";

    // Replace recipient@example.com with a "To" address. If your account
    // is still in the sandbox, this address must be verified. To specify
    // multiple addresses, separate each address with a comma.
    static final String toAddresses = "recipient@example.com";

    // CC and BCC addresses. If your account is in the sandbox, these
    // addresses have to be verified. To specify multiple addresses, separate
    // each address with a comma.
    static final String ccAddresses = "cc-recipient0@example.com,cc-
recipient1@example.com";
    static final String bccAddresses = "bcc-recipient@example.com";

    // Replace smtp_username with your Amazon Pinpoint SMTP user name.
    static final String smtpUsername = "AKIAIOSFODNN7EXAMPLE";

    // Replace smtp_password with your Amazon Pinpoint SMTP password.
    static final String smtpPassword = "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY";

    // (Optional) the name of a configuration set to use for this message.
    static final String configurationSet = "ConfigSet";
    // The subject line of the email
    static final String subject = "Amazon Pinpoint test (SMTP interface accessed using
Java)";

    // The body of the email for recipients whose email clients don't
    // support HTML content.
    static final String htmlBody = String.join(
         System.getProperty("line.separator"),
         "<h1>Amazon Pinpoint SMTP Email Test</h1>",
         "<p>This email was sent with Amazon Pinpoint using the ",
         "<a href='https://github.com/javaee/javamail'>Javamail Package</a>",
         " for <a href='https://www.java.com'>Java</a>."
    );

    // The message tags that you want to apply to the email.
    static final String tag0 = "key0=value0";
    static final String tag1 = "key1=value1";

    public static void main(String[] args) throws Exception {

        // Create a Properties object to contain connection configuration information.
     Properties props = System.getProperties();
     props.put("mail.transport.protocol", "smtp");
     props.put("mail.smtp.port", port);
     props.put("mail.smtp.starttls.enable", "true");
     props.put("mail.smtp.auth", "true");

        // Create a Session object to represent a mail session with the specified
properties.
     Session session = Session.getDefaultInstance(props);

        // Create a message with the specified information.
        MimeMessage msg = new MimeMessage(session);
        msg.setFrom(new InternetAddress(senderAddress,senderName));
        msg.setRecipients(Message.RecipientType.TO,
InternetAddress.parse(toAddresses));
        msg.setRecipients(Message.RecipientType.CC,
InternetAddress.parse(ccAddresses));
        msg.setRecipients(Message.RecipientType.BCC,
InternetAddress.parse(bccAddresses));

        msg.setSubject(subject);
        msg.setContent(htmlBody,"text/html");

        // Add headers for configuration set and message tags to the message.
        msg.setHeader("X-SES-CONFIGURATION-SET", configurationSet);
        msg.setHeader("X-SES-MESSAGE-TAGS", tag0);
        msg.setHeader("X-SES-MESSAGE-TAGS", tag1);

        // Create a transport.
        Transport transport = session.getTransport();

        // Send the message.
        try {
            System.out.println("Sending...");

            // Connect to Amazon Pinpoint using the SMTP username and password you
specified above.
            transport.connect(smtpEndpoint, smtpUsername, smtpPassword);

            // Send the email.
            transport.sendMessage(msg, msg.getAllRecipients());
            System.out.println("Email sent!");
        }
        catch (Exception ex) {
            System.out.println("The email wasn't sent. Error message: "
                + ex.getMessage());
        }
        finally {
            // Close the connection to the SMTP server.
            transport.close();
        }
    }
}
```

JavaScript (Node.js)

Use this example to send email by using the Nodemailer module for Node.js.

```
/*
This code uses callbacks to handle asynchronous function responses.
It currently demonstrates using an async-await pattern.
AWS supports both the async-await and promises patterns.
For more information, see the following:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/
async_function
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Using_promises
https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/calling-services-
asynchronously.html
https://docs.aws.amazon.com/lambda/latest/dg/nodejs-prog-model-handler.html
*/

"use strict";
const nodemailer = require("nodemailer");

// If you're using Amazon Pinpoint in a region other than US West (Oregon),
// replace email-smtp.us-west-2.amazonaws.com with the Amazon Pinpoint SMTP
// endpoint in the appropriate AWS Region.
const smtpEndpoint = "email-smtp.us-west-2.amazonaws.com";

// The port to use when connecting to the SMTP server.
const port = 587;

// Replace sender@example.com with your "From" address.
// This address must be verified with Amazon Pinpoint.
const senderAddress = "Mary Major <sender@example.com>";

// Replace recipient@example.com with a "To" address. If your account
// is still in the sandbox, this address must be verified. To specify
// multiple addresses, separate each address with a comma.
var toAddresses = "recipient@example.com";

// CC and BCC addresses. If your account is in the sandbox, these
// addresses have to be verified. To specify multiple addresses, separate
// each address with a comma.
var ccAddresses = "cc-recipient0@example.com,cc-recipient1@example.com";
var bccAddresses = "bcc-recipient@example.com";

// Replace smtp_username with your Amazon Pinpoint SMTP user name.
const smtpUsername = "AKIAIOSFODNN7EXAMPLE";

// Replace smtp_password with your Amazon Pinpoint SMTP password.
const smtpPassword = "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY";

// (Optional) the name of a configuration set to use for this message.
var configurationSet = "ConfigSet";

// The subject line of the email
var subject = "Amazon Pinpoint test (Nodemailer)";
```

```
// The email body for recipients with non-HTML email clients.
var body_text = `Amazon Pinpoint Test (Nodemailer)
---------------------------------
This email was sent through the Amazon Pinpoint SMTP interface using Nodemailer.
`;

// The body of the email for recipients whose email clients support HTML content.
var body_html = `<html>
<head></head>
<body>
  <h1>Amazon Pinpoint Test (Nodemailer)</h1>
  <p>This email was sent with <a href='https://aws.amazon.com/pinpoint/'>Amazon
 Pinpoint</a>
        using <a href='https://nodemailer.com'>Nodemailer</a> for Node.js.</p>
</body>
</html>`;

// The message tags that you want to apply to the email.
var tag0 = "key0=value0";
var tag1 = "key1=value1";

async function main(){

  // Create the SMTP transport.
  let transporter = nodemailer.createTransport({
    host: smtpEndpoint,
    port: port,
    secure: false, // true for 465, false for other ports
    auth: {
      user: smtpUsername,
      pass: smtpPassword
    }
  });

  // Specify the fields in the email.
  let mailOptions = {
    from: senderAddress,
    to: toAddresses,
    subject: subject,
    cc: ccAddresses,
    bcc: bccAddresses,
    text: body_text,
    html: body_html,
    // Custom headers for configuration set and message tags.
    headers: {
      'X-SES-CONFIGURATION-SET': configurationSet,
      'X-SES-MESSAGE-TAGS': tag0,
      'X-SES-MESSAGE-TAGS': tag1
    }
  };

  // Send the email.
  let info = await transporter.sendMail(mailOptions)

  console.log("Message sent! Message ID: ", info.messageId);
}

main().catch(console.error);
```

Python

Use this example to send email by using the Python email and smtplib libraries.

```
import smtplib
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText

# If you're using Amazon Pinpoint in a region other than US West (Oregon),
# replace email-smtp.us-west-2.amazonaws.com with the Amazon Pinpoint SMTP
# endpoint in the appropriate AWS Region.
HOST = "email-smtp.us-west-2.amazonaws.com"

# The port to use when connecting to the SMTP server.
PORT = 587

# Replace sender@example.com with your "From" address.
# This address must be verified.
SENDER = 'Mary Major <sender@example.com>'

# Replace recipient@example.com with a "To" address. If your account
# is still in the sandbox, this address has to be verified.
RECIPIENT  = 'recipient@example.com'

# CC and BCC addresses. If your account is in the sandbox, these
# addresses have to be verified.
CCRECIPIENT = "cc_recipient@example.com"
BCCRECIPIENT = "bcc_recipient@example.com"

# Replace smtp_username with your Amazon Pinpoint SMTP user name.
USERNAME_SMTP = "AKIAIOSFODNN7EXAMPLE"

# Replace smtp_password with your Amazon Pinpoint SMTP password.
PASSWORD_SMTP = "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"

# (Optional) the name of a configuration set to use for this message.
# If you comment out this line, you also need to remove or comment out
# the "X-Pinpoint-CONFIGURATION-SET:" header below.
CONFIGURATION_SET = "ConfigSet"

# The subject line of the email.
SUBJECT = 'Amazon Pinpoint Test (Python smtplib)'

# The email body for recipients with non-HTML email clients.
BODY_TEXT = ("Amazon Pinpoint Test\r\n"
             "This email was sent through the Amazon Pinpoint SMTP "
             "Interface using the Python smtplib package."
            )

# Create a MIME part for the text body.
textPart = MIMEText(BODY_TEXT, 'plain')

# The HTML body of the email.
BODY_HTML = """<html>
<head></head>
<body>
  <h1>Amazon Pinpoint SMTP Email Test</h1>
  <p>This email was sent with Amazon Pinpoint using the
    <a href='https://www.python.org/'>Python</a>
    <a href='https://docs.python.org/3/library/smtplib.html'>
    smtplib</a> library.</p>
</body>
</html>
            """

# Create a MIME part for the HTML body.
htmlPart = MIMEText(BODY_HTML, 'html')

# The message tags that you want to apply to the email.
TAG0 = "key0=value0"
```

```
TAG1 = "key1=value1"

# Create message container. The correct MIME type is multipart/alternative.
msg = MIMEMultipart('alternative')

# Add sender and recipient addresses to the message
msg['From'] = SENDER
msg['To'] = RECIPIENT
msg['Cc'] = CCRECIPIENT
msg['Bcc'] = BCCRECIPIENT

# Add the subject line, text body, and HTML body to the message.
msg['Subject'] = SUBJECT
msg.attach(textPart)
msg.attach(htmlPart)

# Add  headers for configuration set and message tags to the message.
msg.add_header('X-SES-CONFIGURATION-SET',CONFIGURATION_SET)
msg.add_header('X-SES-MESSAGE-TAGS',TAG0)
msg.add_header('X-SES-MESSAGE-TAGS',TAG1)

# Open a new connection to the SMTP server and begin the SMTP conversation.
try:
    with smtplib.SMTP(HOST, PORT) as server:
        server.ehlo()
        server.starttls()
        #stmplib docs recommend calling ehlo() before and after starttls()
        server.ehlo()
        server.login(USERNAME_SMTP, PASSWORD_SMTP)
        #Uncomment the next line to send SMTP server responses to stdout
        #server.set_debuglevel(1)
        server.sendmail(SENDER, RECIPIENT, msg.as_string())
except Exception as e:
    print ("Error: ", e)
else:
    print ("Email sent!")
```

Send SMS messages

You can use the Amazon Pinpoint API to send SMS messages (text messages) to specific phone numbers or endpoint IDs. This section contains complete code examples that you can use to send SMS messages through the Amazon Pinpoint API by using an AWS SDK.

C#

Use this example to send an SMS message by using the AWS SDK for .NET. This example assumes that you've already installed and configured the AWS SDK for .NET. For more information, see Getting started in the *AWS SDK for .NET Developer Guide*.

This example assumes that you're using a shared credentials file to specify the Access Key and Secret Access Key for an existing IAM user. For more information, see Configuring AWS credentials in the *AWS SDK for .NET Developer Guide*.

```
using System;
using System.Collections.Generic;
using Amazon;
using Amazon.Pinpoint;
using Amazon.Pinpoint.Model;
```

```
namespace SendMessage
{
    class MainClass
    {
        // The AWS Region that you want to use to send the message. For a list of
        // AWS Regions where the Amazon Pinpoint API is available, see
        // https://docs.aws.amazon.com/pinpoint/latest/apireference/
        private static readonly string region = "us-east-1";

        // The phone number or short code to send the message from. The phone number
        // or short code that you specify has to be associated with your Amazon
 Pinpoint
        // account. For best results, specify long codes in E.164 format.
        private static readonly string originationNumber = "+12065550199";

        // The recipient's phone number.  For best results, you should specify the
        // phone number in E.164 format.
        private static readonly string destinationNumber = "+14255550142";

        // The content of the SMS message.
        private static readonly string message = "This message was sent through Amazon
 Pinpoint"
                + "using the AWS SDK for .NET. Reply STOP to opt out.";

        // The Pinpoint project/application ID to use when you send this message.
        // Make sure that the SMS channel is enabled for the project or application
        // that you choose.
        private static readonly string appId = "ce796be37f32f178af652b26eexample";

        // The type of SMS message that you want to send. If you plan to send
        // time-sensitive content, specify TRANSACTIONAL. If you plan to send
        // marketing-related content, specify PROMOTIONAL.
        private static readonly string messageType = "TRANSACTIONAL";

        // The registered keyword associated with the originating short code.
        private static readonly string registeredKeyword = "myKeyword";

        // The sender ID to use when sending the message. Support for sender ID
        // varies by country or region. For more information, see
        // https://docs.aws.amazon.com/pinpoint/latest/userguide/channels-sms-
countries.html
        private static readonly string senderId = "mySenderId";

        public static void Main(string[] args)
        {
            using (AmazonPinpointClient client = new
 AmazonPinpointClient(RegionEndpoint.GetBySystemName(region)))
            {
                SendMessagesRequest sendRequest = new SendMessagesRequest
                {
                    ApplicationId = appId,
                    MessageRequest = new MessageRequest
                    {
                        Addresses = new Dictionary<string, AddressConfiguration>
                        {
                            {
                                destinationNumber,
                                new AddressConfiguration
                                {
                                    ChannelType = "SMS"
                                }
                            }
                        },
                        MessageConfiguration = new DirectMessageConfiguration
                        {
                            SMSMessage = new SMSMessage
                            {
                                Body = message,
                                MessageType = messageType,
                                OriginationNumber = originationNumber,
                                SenderId = senderId,
                                Keyword = registeredKeyword
                            }
                        }
                    }
                };
                try
                {
                    Console.WriteLine("Sending message...");
                    SendMessagesResponse response = client.SendMessages(sendRequest);
                    Console.WriteLine("Message sent!");
                }
                catch (Exception ex)
                {
                    Console.WriteLine("The message wasn't sent. Error message: " +
 ex.Message);
                }
            }
        }
    }
}
```

Java

Use this example to send an SMS message by using the AWS SDK for Java. This example assumes that you've already installed and configured the SDK for Java. For more information, see Getting started in the *AWS SDK for Java Developer Guide*.

This example assumes that you're using a shared credentials file to specify the Access Key and Secret Access Key for an existing IAM user. For more information, see Set default credentials and Region in the *AWS SDK for Java Developer Guide*.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.pinpoint.PinpointClient;
import software.amazon.awssdk.services.pinpoint.model.DirectMessageConfiguration;
import software.amazon.awssdk.services.pinpoint.model.SMSMessage;
import software.amazon.awssdk.services.pinpoint.model.AddressConfiguration;
import software.amazon.awssdk.services.pinpoint.model.ChannelType;
import software.amazon.awssdk.services.pinpoint.model.MessageRequest;
import software.amazon.awssdk.services.pinpoint.model.SendMessagesRequest;
import software.amazon.awssdk.services.pinpoint.model.SendMessagesResponse;
import software.amazon.awssdk.services.pinpoint.model.MessageResponse;
import software.amazon.awssdk.services.pinpoint.model.PinpointException;
import java.util.HashMap;
import java.util.Map;
```

```
    public static void sendSMSMessage(PinpointClient pinpoint, String message, String
 appId, String originationNumber, String destinationNumber) {

    try {

        Map<String, AddressConfiguration> addressMap =
                new HashMap<String, AddressConfiguration>();

        AddressConfiguration addConfig = AddressConfiguration.builder()
                .channelType(ChannelType.SMS)
                .build();
        addressMap.put(destinationNumber, addConfig);

        SMSMessage smsMessage = SMSMessage.builder()
                .body(message)
                .messageType(messageType)
                .originationNumber(originationNumber)
                .senderId(senderId)
                .keyword(registeredKeyword)
                .build();

        // Create a DirectMessageConfiguration object
        DirectMessageConfiguration direct = DirectMessageConfiguration.builder()
                .smsMessage(smsMessage)
                .build();

        MessageRequest msgReq = MessageRequest.builder()
                .addresses(addressMap)
                .messageConfiguration(direct)
                .build();

        // create a  SendMessagesRequest object
        SendMessagesRequest request = SendMessagesRequest.builder()
                .applicationId(appId)
                .messageRequest(msgReq)
                .build();

        SendMessagesResponse response= pinpoint.sendMessages(request);

        MessageResponse msg1 = response.messageResponse();
        Map map1 = msg1.result();

        //Write out the result of sendMessage
        map1.forEach((k, v) -> System.out.println((k + ":" + v)));

    } catch (PinpointException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
  }
```

For the full SDK example, see SendMessage.java on GitHub.

JavaScript (Node.js)

Use this example to send an SMS message by using the AWS SDK for JavaScript in Node.js. This example assumes that you've already installed and configured the SDK for JavaScript in Node.js. For more information, see Getting started in the *AWS SDK for JavaScript in Node.js Developer Guide*.

This example assumes that you're using a shared credentials file to specify the Access Key and Secret Access Key for an existing IAM user. For more information, see Setting credentials in the *AWS SDK for JavaScript in Node.js Developer Guide*.

```

'use strict';

var AWS = require('aws-sdk');

// The AWS Region that you want to use to send the message. For a list of
// AWS Regions where the Amazon Pinpoint API is available, see
// https://docs.aws.amazon.com/pinpoint/latest/apireference/.
var aws_region = "us-east-1";

// The phone number or short code to send the message from. The phone number
// or short code that you specify has to be associated with your Amazon Pinpoint
// account. For best results, specify long codes in E.164 format.
var originationNumber = "+12065550199";

// The recipient's phone number.  For best results, you should specify the
// phone number in E.164 format.
var destinationNumber = "+14255550142";

// The content of the SMS message.
var message = "This message was sent through Amazon Pinpoint "
            + "using the AWS SDK for JavaScript in Node.js. Reply STOP to "
            + "opt out.";

// The Amazon Pinpoint project/application ID to use when you send this message.
// Make sure that the SMS channel is enabled for the project or application
// that you choose.
var applicationId = "ce796be37f32f178af652b26eexample";

// The type of SMS message that you want to send. If you plan to send
// time-sensitive content, specify TRANSACTIONAL. If you plan to send
// marketing-related content, specify PROMOTIONAL.
var messageType = "TRANSACTIONAL";

// The registered keyword associated with the originating short code.
var registeredKeyword = "myKeyword";

// The sender ID to use when sending the message. Support for sender ID
// varies by country or region. For more information, see
// https://docs.aws.amazon.com/pinpoint/latest/userguide/channels-sms-countries.html
var senderId = "MySenderID";

// Specify that you're using a shared credentials file, and optionally specify
// the profile that you want to use.
var credentials = new AWS.SharedIniFileCredentials({profile: 'default'});
AWS.config.credentials = credentials;

// Specify the region.
AWS.config.update({region:aws_region});

//Create a new Pinpoint object.
var pinpoint = new AWS.Pinpoint();

// Specify the parameters to pass to the API.
var params = {
  ApplicationId: applicationId,
  MessageRequest: {
    Addresses: {
      [destinationNumber]: {
        ChannelType: 'SMS'
      }
    },
    MessageConfiguration: {
      SMSMessage: {
        Body: message,
        Keyword: registeredKeyword,
        MessageType: messageType,
        OriginationNumber: originationNumber,
        SenderId: senderId,
      }
    }
  }
};

//Try to send the message.
pinpoint.sendMessages(params, function(err, data) {
  // If something goes wrong, print an error message.
  if(err) {
    console.log(err.message);
  // Otherwise, show the unique ID for the message.
  } else {
    console.log("Message sent! "
        + data['MessageResponse']['Result'][destinationNumber]['StatusMessage']);
  }
});
```

Python

Use this example to send an SMS message by using the AWS SDK for Python (Boto3). This example assumes that you've already installed and configured the SDK for Python. For more information, see Quickstart in *AWS SDK for Python (Boto3) Getting Started*.

This example assumes that you're using a shared credentials file to specify the Access Key and Secret Access Key for an existing IAM user. For more information, see Credentials in the *AWS SDK for Python (Boto3) API Reference*.

```
import boto3
from botocore.exceptions import ClientError

# The AWS Region that you want to use to send the message. For a list of
# AWS Regions where the Amazon Pinpoint API is available, see
# https://docs.aws.amazon.com/pinpoint/latest/apireference/
region = "us-east-1"

# The phone number or short code to send the message from. The phone number
# or short code that you specify has to be associated with your Amazon Pinpoint
# account. For best results, specify long codes in E.164 format.
originationNumber = "+12065550199"

# The recipient's phone number.  For best results, you should specify the
# phone number in E.164 format.
destinationNumber = "+14255550142"

# The content of the SMS message.
message = ("This is a sample message sent from Amazon Pinpoint by using the "
           "AWS SDK for Python (Boto 3).")

# The Amazon Pinpoint project/application ID to use when you send this message.
# Make sure that the SMS channel is enabled for the project or application
# that you choose.
applicationId = "ce796be37f32f178af652b26eexample"

# The type of SMS message that you want to send. If you plan to send
# time-sensitive content, specify TRANSACTIONAL. If you plan to send
# marketing-related content, specify PROMOTIONAL.
messageType = "TRANSACTIONAL"

# The registered keyword associated with the originating short code.
registeredKeyword = "myKeyword"

# The sender ID to use when sending the message. Support for sender ID
# varies by country or region. For more information, see
# https://docs.aws.amazon.com/pinpoint/latest/userguide/channels-sms-countries.html
senderId = "MySenderID"

# Create a new client and specify a region.
client = boto3.client('pinpoint',region_name=region)
try:
    response = client.send_messages(
        ApplicationId=applicationId,
        MessageRequest={
            'Addresses': {
                destinationNumber: {
                    'ChannelType': 'SMS'
                }
            },
            'MessageConfiguration': {
                'SMSMessage': {
                    'Body': message,
                    'Keyword': registeredKeyword,
                    'MessageType': messageType,
                    'OriginationNumber': originationNumber,
                    'SenderId': senderId
                }
            }
        }
    )

except ClientError as e:
    print(e.response['Error']['Message'])
else:
    print("Message sent! Message ID: "
            + response['MessageResponse']['Result'][destinationNumber]['MessageId'])
```

Send voice messages

You can use the Amazon Pinpoint API to send voice messages to specific phone numbers. This section contains complete code examples that you can use to send voice messages through the Amazon Pinpoint SMS and Voice API by using an AWS SDK.

Java

Use this example to send a voice message by using the AWS SDK for Java. This example assumes that you've already installed and configured the SDK for Java. For more information, see Getting started in the AWS SDK for Java Developer Guide.

This example assumes that you're using a shared credentials file to specify the Access Key and Secret Access Key for an existing IAM user. For more information, see Set up AWS credentials and Region for development in the *AWS SDK for Java Developer Guide*.

```
import software.amazon.awssdk.core.client.config.ClientOverrideConfiguration;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.pinpointsmsvoice.PinpointSmsVoiceClient;
import software.amazon.awssdk.services.pinpointsmsvoice.model.SSMLMessageType;
import software.amazon.awssdk.services.pinpointsmsvoice.model.VoiceMessageContent;
import software.amazon.awssdk.services.pinpointsmsvoice.model.SendVoiceMessageRequest;
import
 software.amazon.awssdk.services.pinpointsmsvoice.model.PinpointSmsVoiceException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
```

```
    public static void sendVoiceMsg(PinpointSmsVoiceClient client, String
 originationNumber, String destinationNumber ) {

        try {
            SSMLMessageType ssmlMessageType = SSMLMessageType.builder()
                    .languageCode(languageCode)
                    .text(ssmlMessage)
                     .voiceId(voiceName)
                     .build();

            VoiceMessageContent content = VoiceMessageContent.builder()
                    .ssmlMessage(ssmlMessageType)
                    .build();

            SendVoiceMessageRequest voiceMessageRequest =
 SendVoiceMessageRequest.builder()
                    .destinationPhoneNumber(destinationNumber)
                    .originationPhoneNumber(originationNumber)
                    .content(content)
                    .build();

            client.sendVoiceMessage(voiceMessageRequest);
            System.out.println("The message was sent successfully.");

        } catch (PinpointSmsVoiceException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
```

For the full SDK example, see SendVoiceMessage.java on GitHub.

JavaScript (Node.js)

Use this example to send a voice message by using the AWS SDK for JavaScript in Node.js. This example assumes that you've already installed and configured the SDK for JavaScript in Node.js.

This example assumes that you're using a shared credentials file to specify the Access Key and Secret Access Key for an existing IAM user. For more information, see Setting credentials in the *AWS SDK for JavaScript in Node.js Developer Guide*.

```

'use strict'

var AWS = require('aws-sdk');

// The AWS Region that you want to use to send the voice message. For a list of
// AWS Regions where the Amazon Pinpoint SMS and Voice API is available, see
// https://docs.aws.amazon.com/pinpoint-sms-voice/latest/APIReference/
var aws_region = "us-east-1";

// The phone number that the message is sent from. The phone number that you
// specify has to be associated with your Amazon Pinpoint account. For best results,
 you
// should specify the phone number in E.164 format.
var originationNumber = "+12065550110";

// The recipient's phone number. For best results, you should specify the phone
// number in E.164 format.
var destinationNumber = "+12065550142";

// The language to use when sending the message. For a list of supported
// languages, see https://docs.aws.amazon.com/polly/latest/dg/SupportedLanguage.html
var languageCode = "en-US";

// The Amazon Polly voice that you want to use to send the message. For a list
// of voices, see https://docs.aws.amazon.com/polly/latest/dg/voicelist.html
var voiceId = "Matthew";
```

```
// The content of the message. This example uses SSML to customize and control
// certain aspects of the message, such as the volume or the speech rate.
// The message can't contain any line breaks.
var ssmlMessage = "<speak>"
    + "This is a test message sent from <emphasis>Amazon Pinpoint</emphasis> "
    + "using the <break strength='weak'/>AWS SDK for JavaScript in Node.js. "
    + "<amazon:effect phonation='soft'>Thank you for listening."
    + "</amazon:effect>"
    + "</speak>";

// The phone number that you want to appear on the recipient's device. The phone
// number that you specify has to be associated with your Amazon Pinpoint account.
var callerId = "+12065550199";

// The configuration set that you want to use to send the message.
var configurationSet = "ConfigSet";

// Specify that you're using a shared credentials file, and optionally specify
// the profile that you want to use.
var credentials = new AWS.SharedIniFileCredentials({profile: 'default'});
AWS.config.credentials = credentials;

// Specify the region.
AWS.config.update({region:aws_region});

//Create a new Pinpoint object.
var pinpointsmsvoice = new AWS.PinpointSMSVoice();

var params = {
  CallerId: callerId,
  ConfigurationSetName: configurationSet,
  Content: {
    SSMLMessage: {
      LanguageCode: languageCode,
      Text: ssmlMessage,
      VoiceId: voiceId
    }
  },
  DestinationPhoneNumber: destinationNumber,
  OriginationPhoneNumber: originationNumber
};

//Try to send the message.
pinpointsmsvoice.sendVoiceMessage(params, function(err, data) {
  // If something goes wrong, print an error message.
  if(err) {
    console.log(err.message);
  // Otherwise, show the unique ID for the message.
  } else {
    console.log("Message sent! Message ID: " + data['MessageId']);
  }
});
```

Python

Use this example to send a voice message by using the AWS SDK for Python (Boto3). This example assumes that you've already installed and configured the SDK for Python (Boto3).

This example assumes that you're using a shared credentials file to specify the Access Key and Secret Access Key for an existing IAM user. For more information, see Credentials in the *AWS SDK for Python (Boto3) API Reference*.

```
import boto3
from botocore.exceptions import ClientError

# The AWS Region that you want to use to send the voice message. For a list of
# AWS Regions where the Amazon Pinpoint SMS and Voice API is available, see
# https://docs.aws.amazon.com/pinpoint-sms-voice/latest/APIReference/
region = "us-east-1"

# The phone number that the message is sent from. The phone number that you
# specify has to be associated with your Amazon Pinpoint account. For best results, you
# should specify the phone number in E.164 format.
originationNumber = "+12065550110"

# The recipient's phone number. For best results, you should specify the phone
# number in E.164 format.
destinationNumber = "+12065550142"

# The language to use when sending the message. For a list of supported
# languages, see https://docs.aws.amazon.com/polly/latest/dg/SupportedLanguage.html
languageCode = "en-US"

# The Amazon Polly voice that you want to use to send the message. For a list
# of voices, see https://docs.aws.amazon.com/polly/latest/dg/voicelist.html
voiceId = "Matthew"

# The content of the message. This example uses SSML to customize and control
# certain aspects of the message, such as the volume or the speech rate.
# The message can't contain any line breaks.
ssmlMessage = ("<speak>"
               "This is a test message sent from <emphasis>Amazon Pinpoint</emphasis> "
               "using the <break strength='weak'/>AWS SDK for Python. "
               "<amazon:effect phonation='soft'>Thank you for listening."
               "</amazon:effect>"
               "</speak>")

# The phone number that you want to appear on the recipient's device. The phone
# number that you specify has to be associated with your Amazon Pinpoint account.
callerId = "+12065550199"

# The configuration set that you want to use to send the message.
configurationSet = "ConfigSet"

# Create a new SMS and Voice client and specify an AWS Region.
client = boto3.client('sms-voice',region_name=region)

try:
    response = client.send_voice_message(
        DestinationPhoneNumber = destinationNumber,
        OriginationPhoneNumber = originationNumber,
        CallerId = callerId,
        ConfigurationSetName = configurationSet,
        Content={
            'SSMLMessage':{
                'LanguageCode': languageCode,
                'VoiceId': voiceId,
                'Text': ssmlMessage
            }
        }
    )
# Display an error message if something goes wrong.
except ClientError as e:
    print(e.response['Error']['Message'])
# If the message is sent successfully, show the message ID.
else:
    print("Message sent!"),
    print("Message ID: " + response['MessageId'])
```

Send push notifications

The Amazon Pinpoint API can send transactional push notifications to specific device identifiers. This section contains complete code examples that you can use to send push notifications through the Amazon Pinpoint API by using an AWS SDK.

You can use these examples to send push notifications through any push notification service that Amazon Pinpoint supports. Currently, Amazon Pinpoint supports the following channels: Firebase Cloud Messaging (FCM), Apple Push Notification Service (APNs), Baidu Cloud Push, and Amazon Device Messaging (ADM).

Note
When you send push notifications through the Firebase Cloud Messaging (FCM) service, use the service name `GCM` in your call to the Amazon Pinpoint API. The Google Cloud Messaging (GCM) service was discontinued by Google on April 10, 2018. However, the Amazon Pinpoint API uses the `GCM` service name for messages that it sends through the FCM service in order to maintain compatibility with API code that was written prior to the discontinuation of the GCM service.

JavaScript (Node.js)

Use this example to send push notifications by using the AWS SDK for JavaScript in Node.js. This example assumes that you've already installed and configured the SDK for JavaScript in Node.js.

This example also assumes that you're using a shared credentials file to specify the Access Key and Secret Access Key for an existing IAM user. For more information, see Setting credentials in the *AWS SDK for JavaScript in Node.js Developer Guide*.

```
'use strict';

const AWS = require('aws-sdk');

// The AWS Region that you want to use to send the message. For a list of
// AWS Regions where the Amazon Pinpoint API is available, see
// https://docs.aws.amazon.com/pinpoint/latest/apireference/
const region = 'us-east-1';

// The title that appears at the top of the push notification.
var title = 'Test message sent from Amazon Pinpoint.';

// The content of the push notification.
var message = 'This is a sample message sent from Amazon Pinpoint by using the '
            + 'AWS SDK for JavaScript in Node.js';

// The Amazon Pinpoint project ID that you want to use when you send this
// message. Make sure that the push channel is enabled for the project that
// you choose.
var applicationId = 'ce796be37f32f178af652b26eexample';

// An object that contains the unique token of the device that you want to send
// the message to, and the push service that you want to use to send the message.
var recipient = {
  'token': 'a0b1c2d3e4f5g6h7i8j9k0l1m2n3o4p5q6r7s8t9u0v1w2x3y4z5a6b7c8d8e9f0',
  'service': 'GCM'
  };

// The action that should occur when the recipient taps the message. Possible
// values are OPEN_APP (opens the app or brings it to the foreground),
// DEEP_LINK (opens the app to a specific page or interface), or URL (opens a
// specific URL in the device's web browser.)
var action = 'URL';

// This value is only required if you use the URL action. This variable contains
// the URL that opens in the recipient's web browser.
var url = 'https://www.example.com';

// The priority of the push notification. If the value is 'normal', then the
// delivery of the message is optimized for battery usage on the recipient's
// device, and could be delayed. If the value is 'high', then the notification is
// sent immediately, and might wake a sleeping device.
var priority = 'normal';

// The amount of time, in seconds, that the push notification service provider
// (such as FCM or APNS) should attempt to deliver the message before dropping
// it. Not all providers allow you specify a TTL value.
var ttl = 30;

// Boolean that specifies whether the notification is sent as a silent
// notification (a notification that doesn't display on the recipient's device).
var silent = false;

function CreateMessageRequest() {
  var token = recipient['token'];
  var service = recipient['service'];
  if (service == 'GCM') {
    var messageRequest = {
      'Addresses': {
        [token]: {
          'ChannelType' : 'GCM'
        }
      },
      'MessageConfiguration': {
        'GCMMessage': {
          'Action': action,
          'Body': message,
          'Priority': priority,
          'SilentPush': silent,
          'Title': title,
          'TimeToLive': ttl,
          'Url': url
        }
      }
    };
  } else if (service == 'APNS') {
    var messageRequest = {
      'Addresses': {
        [token]: {
          'ChannelType' : 'APNS'
        }
      },
      'MessageConfiguration': {
        'APNSMessage': {
          'Action': action,
          'Body': message,
          'Priority': priority,
          'SilentPush': silent,
          'Title': title,
          'TimeToLive': ttl,
          'Url': url
        }
      }
    };
  } else if (service == 'BAIDU') {
    var messageRequest = {
      'Addresses': {
        [token]: {
          'ChannelType' : 'BAIDU'
        }
      },
      'MessageConfiguration': {
        'BaiduMessage': {
          'Action': action,
          'Body': message,
          'SilentPush': silent,
          'Title': title,
          'TimeToLive': ttl,
          'Url': url
        }
      }
    };
  } else if (service == 'ADM') {
    var messageRequest = {
      'Addresses': {
        [token]: {
          'ChannelType' : 'ADM'
        }
      },
      'MessageConfiguration': {
        'ADMMessage': {
          'Action': action,
          'Body': message,
          'SilentPush': silent,
          'Title': title,
          'Url': url
        }
      }
    };
  }

  return messageRequest
}

function ShowOutput(data){
  if (data["MessageResponse"]["Result"][recipient["token"]]["DeliveryStatus"]
      == "SUCCESSFUL") {
    var status = "Message sent! Response information: ";
  } else {
    var status = "The message wasn't sent. Response information: ";
  }
  console.log(status);
  console.dir(data, { depth: null });
}

function SendMessage() {
  var token = recipient['token'];
  var service = recipient['service'];
  var messageRequest = CreateMessageRequest();

  // Specify that you're using a shared credentials file, and specify the
  // IAM profile to use.
  var credentials = new AWS.SharedIniFileCredentials({ profile: 'default' });
  AWS.config.credentials = credentials;

  // Specify the AWS Region to use.
  AWS.config.update({ region: region });

  //Create a new Pinpoint object.
  var pinpoint = new AWS.Pinpoint();
  var params = {
    "ApplicationId": applicationId,
    "MessageRequest": messageRequest
  };

  // Try to send the message.
  pinpoint.sendMessages(params, function(err, data) {
    if (err) console.log(err);
    else     ShowOutput(data);
  });
}

SendMessage()
```

Python

Use this example to send push notifications by using the AWS SDK for Python (Boto3). This example assumes that you've already installed and configured the SDK for Python (Boto3).

This example also assumes that you're using a shared credentials file to specify the Access Key and Secret Access Key for an existing IAM user. For more information, see Credentials in the *AWS SDK for Python (Boto3) API Reference*.

```
import json
import boto3
from botocore.exceptions import ClientError

# The AWS Region that you want to use to send the message. For a list of
# AWS Regions where the Amazon Pinpoint API is available, see
# https://docs.aws.amazon.com/pinpoint/latest/apireference/
region = "us-east-1"

# The title that appears at the top of the push notification.
title = "Test message sent from Amazon Pinpoint."

# The content of the push notification.
message = ("This is a sample message sent from Amazon Pinpoint by using the "
           "AWS SDK for Python (Boto3).")

# The Amazon Pinpoint project/application ID to use when you send this message.
# Make sure that the push channel is enabled for the project or application
# that you choose.
application_id = "ce796be37f32f178af652b26eexample"

# A dictionary that contains the unique token of the device that you want to send the
# message to, and the push service that you want to use to send the message.
recipient = {
    "token": "a0b1c2d3e4f5g6h7i8j9k0l1m2n3o4p5q6r7s8t9u0v1w2x3y4z5a6b7c8d8e9f0",
    "service": "GCM"
    }

# The action that should occur when the recipient taps the message. Possible
# values are OPEN_APP (opens the app or brings it to the foreground),
# DEEP_LINK (opens the app to a specific page or interface), or URL (opens a
# specific URL in the device's web browser.)
action = "URL"

# This value is only required if you use the URL action. This variable contains
# the URL that opens in the recipient's web browser.
url = "https://www.example.com"

# The priority of the push notification. If the value is 'normal', then the
# delivery of the message is optimized for battery usage on the recipient's
# device, and could be delayed. If the value is 'high', then the notification is
# sent immediately, and might wake a sleeping device.
priority = "normal"
```

```
# The amount of time, in seconds, that the push notification service provider
# (such as FCM or APNS) should attempt to deliver the message before dropping
# it. Not all providers allow you specify a TTL value.
ttl = 30

# Boolean that specifies whether the notification is sent as a silent
# notification (a notification that doesn't display on the recipient's device).
silent = False

# Set the MessageType based on the values in the recipient variable.
def create_message_request():

    token = recipient["token"]
    service = recipient["service"]

    if service == "GCM":
        message_request = {
            'Addresses': {
                token: {
                    'ChannelType': 'GCM'
                }
            },
            'MessageConfiguration': {
                'GCMMessage': {
                    'Action': action,
                    'Body': message,
                    'Priority' : priority,
                    'SilentPush': silent,
                    'Title': title,
                    'TimeToLive': ttl,
                    'Url': url
                }
            }
        }
    elif service == "APNS":
        message_request = {
            'Addresses': {
                token: {
                    'ChannelType': 'APNS'
                }
            },
            'MessageConfiguration': {
                'APNSMessage': {
                    'Action': action,
                    'Body': message,
                    'Priority' : priority,
                    'SilentPush': silent,
                    'Title': title,
                    'TimeToLive': ttl,
                    'Url': url
                }
            }
        }
    elif service == "BAIDU":
        message_request = {
            'Addresses': {
                token: {
                    'ChannelType': 'BAIDU'
                }
            },
            'MessageConfiguration': {
                'BaiduMessage': {
                    'Action': action,
                    'Body': message,
                    'SilentPush': silent,
                    'Title': title,
                    'TimeToLive': ttl,
                'Url': url
                }
            }
        }
    elif service == "ADM":
        message_request = {
            'Addresses': {
                token: {
                    'ChannelType': 'ADM'
                }
            },
            'MessageConfiguration': {
                'ADMMessage': {
                    'Action': action,
                    'Body': message,
                    'SilentPush': silent,
                    'Title': title,
                    'Url': url
                }
            }
        }
    else:
        message_request = None

    return message_request

# Show a success or failure message, and provide the response from the API.
def show_output(response):
    if response['MessageResponse']['Result'][recipient["token"]]['DeliveryStatus'] ==
 "SUCCESSFUL":
        status = "Message sent! Response information:\n"
    else:
        status = "The message wasn't sent. Response information:\n"
    print(status, json.dumps(response,indent=4))

# Send the message through the appropriate channel.
def send_message():

    token = recipient["token"]
    service = recipient["service"]
    message_request = create_message_request()

    client = boto3.client('pinpoint',region_name=region)

    try:
        response = client.send_messages(
            ApplicationId=application_id,
            MessageRequest=message_request
        )
    except ClientError as e:
        print(e.response['Error']['Message'])
    else:
        show_output(response)

send_message()
```

Creating custom channels in Amazon Pinpoint

Amazon Pinpoint includes built-in support for sending messages through the push notification, email, SMS, and voice channels. You can also configure Amazon Pinpoint to send messages through other channels by creating custom channels. Custom channels in Amazon Pinpoint allow you to send messages through any service that has an API, including third-party services. You can interact with APIs by using a webhook, or by calling an AWS Lambda function.

The segments that you send custom channel campaigns to can contain endpoints of all types (that is, endpoints where the value of the `ChannelType` attribute is EMAIL, VOICE, SMS, CUSTOM, or one of the various push notification endpoint types).

Creating a campaign that sends messages through a custom channel

To assign a Lambda function or webhook to an individual campaign, use the Amazon Pinpoint API to create or update a Campaign object.

The `MessageConfiguration` object in the campaign must also contain a `CustomMessage` object. This object has one member, `Data`. The value of `Data` is a JSON string that contains the message payload that you want to send to the custom channel.

The campaign has to contain a `CustomDeliveryConfiguration` object. Within the `CustomDeliveryConfiguration` object, specify the following:

- `EndpointTypes` – An array that contains all of the endpoint types that the custom channel campaign should be sent to. It can contain any or all of the following channel types:
  - `ADM`
  - `APNS`
  - `APNS_SANDBOX`
  - `APNS_VOIP`
  - `APNS_VOIP_SANDBOX`
  - `BAIDU`
  - `CUSTOM`
  - `EMAIL`
  - `GCM`
  - `SMS`
  - `VOICE`
- `DeliveryUri` – The destination that endpoints are sent to. You can specify only one of the following:
  - The Amazon Resource Name (ARN) of a Lambda function that you want to execute when the campaign runs.
  - The URL of the webhook that you want to send endpoint data to when the campaign runs.

Note
The `Campaign` object can also contain a `Hook` object. This object is only used to create segments that are customized by a Lambda function when a campaign is executed. For more information, see Customizing segments with AWS Lambda (p. 154).

Understanding the event data that Amazon Pinpoint sends to custom channels

Before you create a Lambda function that sends messages over a custom channel, you should familiarize yourself with the data that Amazon Pinpoint emits. When a Amazon Pinpoint campaign sends messages over a custom channel, it sends a payload to the target Lambda function that resembles the following example:

```
{
  "Message":{},
  "Data":"The payload that's provided in the CustomMessage object in MessageConfiguration",
  "ApplicationId":"3a9b1f4e6c764ba7b031e7183example",
  "CampaignId":"13978104ce5d6017c72552257example",
  "TreatmentId":"0",
  "ActivityId":"575cb1929d5ba43e87e2478eeexample",
  "ScheduledTime":"2020-04-08T19:00:16.843Z",
  "Endpoints":{
    "1dbcd396df28ac6cf8c1c2b7fexample":{
      "ChannelType":"EMAIL",
      "Address":"mary.major@example.com",
      "EndpointStatus":"ACTIVE",
      "OptOut":"NONE",
      "Location":{
        "City":"Seattle",
        "Country":"USA"
      },
      "Demographic":{
        "Make":"OnePlus",
        "Platform":"android"
      },
      "EffectiveDate":"2020-04-01T01:05:17.267Z",
      "Attributes":{
        "CohortId":[
          "42"
        ]
      },
      "CreationDate":"2020-04-01T01:05:17.267Z"
    }
  }
}
```

The event data provides the following attributes:

- `ApplicationId` – The ID of the Amazon Pinpoint project that the campaign belongs to.
- `CampaignId` – The ID of the Amazon Pinpoint project that invoked the Lambda function.
- `TreatmentId` – The ID of the campaign variant. If you created a standard campaign, this value is always 0. If you created an A/B test campaign, this value is an integer between 0 and 4.
- `ActivityId` – The ID of the activity being performed by the campaign.
- `ScheduledTime` – The time when Amazon Pinpoint executed the campaign, shown in ISO 8601 format.
- `Endpoints` – A list of the endpoints that were targeted by the campaign. Each payload can contain up to 50 endpoints. If the segment that the campaign was sent to 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.

You can use this sample data when creating and testing your custom channel Lambda function.

Configuring webhooks

If you use a webhook to send custom channel messages, the URL of the webhook has to begin with "https://". The webhook URL can only contain alphanumeric characters, plus the following symbols: hyphen (-), period (.), underscore (_), tilde (~), question mark (?), slash or solidus (/), pound or hash sign (#), and semicolon (:). The URL has to comply with RFC3986.

When you create a campaign that specifies a webhook URL, Amazon Pinpoint issues an HTTP `HEAD` to that URL. The response to the `HEAD` request must contain a header called `X-Amz-Pinpoint-AccountId`. The value of this header must equal your AWS account ID.

Configuring Lambda functions

This section provides an overview of the steps that you need to take when you create a Lambda function that sends messages over a custom channel. First, you create the function. After that, you add an execution policy to the function. This policy allows Amazon Pinpoint to execute the policy when a campaign runs.

For an introduction to creating Lambda functions, see Building Lambda functions in the *AWS Lambda Developer Guide*.

Example Lambda function

The following code example processes the payload and logs the number of endpoints of each endpoint type in CloudWatch.

```
import boto3
import random
import pprint
import json
import time

cloudwatch = boto3.client('cloudwatch')

def lambda_handler(event, context):
    customEndpoints = 0
    smsEndpoints = 0
    pushEndpoints = 0
    emailEndpoints = 0
    voiceEndpoints = 0
    numEndpoints = len(event['Endpoints'])

    print("Payload:\n", event)
    print("Endpoints in payload: " + str(numEndpoints))

    for key in event['Endpoints'].keys():
        if event['Endpoints'][key]['ChannelType'] == "CUSTOM":
            customEndpoints += 1
        elif event['Endpoints'][key]['ChannelType'] == "SMS":
            smsEndpoints += 1
        elif event['Endpoints'][key]['ChannelType'] == "EMAIL":
            emailEndpoints += 1
        elif event['Endpoints'][key]['ChannelType'] == "VOICE":
            voiceEndpoints += 1
        else:
            pushEndpoints += 1

    response = cloudwatch.put_metric_data(
        MetricData = [
            {
                'MetricName': 'EndpointCount',
                'Dimensions': [
                    {
                        'Name': 'CampaignId',
                        'Value': event['CampaignId']
                    },
                    {
                        'Name': 'ApplicationId',
                        'Value': event['ApplicationId']
                    }
                ],
                'Unit': 'None',
                'Value': len(event['Endpoints'])
            },
            {
                'MetricName': 'CustomCount',
                'Dimensions': [
                    {
                        'Name': 'CampaignId',
                        'Value': event['CampaignId']
                    },
                    {
                        'Name': 'ApplicationId',
                        'Value': event['ApplicationId']
                    }
                ],
                'Unit': 'None',
                'Value': customEndpoints
            },
            {
                'MetricName': 'SMSCount',
                'Dimensions': [
                    {
                        'Name': 'CampaignId',
                        'Value': event['CampaignId']
                    },
                    {
                        'Name': 'ApplicationId',
                        'Value': event['ApplicationId']
                    }
                ],
                'Unit': 'None',
                'Value': smsEndpoints
            },
            {
                'MetricName': 'EmailCount',
                'Dimensions': [
                    {
                        'Name': 'CampaignId',
                        'Value': event['CampaignId']
                    },
                    {
                        'Name': 'ApplicationId',
                        'Value': event['ApplicationId']
                    }
                ],
                'Unit': 'None',
                'Value': emailEndpoints
            },
            {
                'MetricName': 'VoiceCount',
                'Dimensions': [
                    {
                        'Name': 'CampaignId',
                        'Value': event['CampaignId']
                    },
                    {
                        'Name': 'ApplicationId',
                        'Value': event['ApplicationId']
                    }
                ],
                'Unit': 'None',
                'Value': voiceEndpoints
            },
            {
                'MetricName': 'PushCount',
                'Dimensions': [
                    {
                        'Name': 'CampaignId',
                        'Value': event['CampaignId']
                    },
                    {
                        'Name': 'ApplicationId',
                        'Value': event['ApplicationId']
                    }
                ],
                'Unit': 'None',
                'Value': pushEndpoints
            },
            {
                'MetricName': 'EndpointCount',
                'Dimensions': [
                ],
                'Unit': 'None',
                'Value': len(event['Endpoints'])
            },
            {
                'MetricName': 'CustomCount',
                'Dimensions': [
                ],
                'Unit': 'None',
                'Value': customEndpoints
            },
            {
                'MetricName': 'SMSCount',
                'Dimensions': [
                ],
                'Unit': 'None',
                'Value': smsEndpoints
            },
            {
                'MetricName': 'EmailCount',
                'Dimensions': [
                ],
                'Unit': 'None',
                'Value': emailEndpoints
            },
            {
                'MetricName': 'VoiceCount',
                'Dimensions': [
                ],
                'Unit': 'None',
                'Value': voiceEndpoints
            },
            {
                'MetricName': 'PushCount',
                'Dimensions': [
                ],
                'Unit': 'None',
                'Value': pushEndpoints
            }
        ],
        Namespace = 'PinpointCustomChannelExecution'
    )
    print("cloudwatchResponse:\n",response)
```

When an Amazon Pinpoint campaign executes this Lambda function, Amazon Pinpoint sends the function a list of segment members. The function counts the number of endpoints of each `ChannelType`. It then sends that data to Amazon CloudWatch. You can view these metrics in the **Metrics** section of the CloudWatch console. The metrics are available in the **PinpointCustomChannelExecution** namespace.

You can modify this code example so that it also connects to the API of an external service in order to send messages through that service.

Granting Amazon Pinpoint permission to invoke the Lambda function

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 myFunction \
--statement-id sid0 \
--action lambda:InvokeFunction \
--principal pinpoint.us-east-1.amazonaws.com \
--source-arn arn:aws:mobiletargeting:us-east-1:111122223333:apps/*
```

In the preceding command, do the following:

- Replace *`myFunction`* with the name of the Lambda function.
- Replace *`us-east-1`* with the AWS Region where you use Amazon Pinpoint.
- Replace *`111122223333`* with your AWS account ID.

When you run the `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:myFunction\",
    \"Condition\":
      {\"ArnLike\":
        {\"AWS:SourceArn\":
         \"arn:aws:mobiletargeting:us-east-1:111122223333:apps/*\"}}}"
}
```

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

Further restricting the execution policy

You can modify the execution policy by restricting it to a specific Amazon Pinpoint project. To do this, replace the `*` in the preceding example with the unique ID of the project. You can further restrict the policy by limiting it to a specific campaign. For example, to restrict the policy to only allow a campaign with the campaign ID `95fee4cd1d7f5cd67987c1436example` in a project with the project ID `dbaf6ec2226f0a9a8615e3ea5example`, use the following value for the `source-arn` attribute:

```
arn:aws:mobiletargeting:us-east-1:111122223333:apps/dbaf6ec2226f0a9a8615e3ea5example/
campaigns/95fee4cd1d7f5cd67987c1436example
```

Note
If you do restrict execution of the Lambda function to a specific campaign, you first have to create the function with a less restrictive policy. Next, you have to create the campaign in Amazon Pinpoint and choose the function. Finally, you have to update the execution policy to refer to the specified campaign.

Streaming Amazon Pinpoint events to Kinesis

In Amazon Pinpoint, an *event* is an action that occurs when a user interacts with one of your applications, when you send a message from a campaign or journey, or when you send a transactional SMS or email message. For example, if you send an email message, several events occur:

- When you send the message, a *send* event occurs.
- When the message reaches the recipient's inbox, a *delivered* event occurs.
- When the recipient opens the message, an *open* event occurs.

You can configure Amazon Pinpoint to send information about events to Amazon Kinesis. The Kinesis platform offers services that you can use to collect, process, and analyze data from AWS services in real time. Amazon Pinpoint can send event data to Kinesis Data Firehose, which streams this data to AWS data stores such as Amazon S3 or Amazon Redshift. Amazon Pinpoint can also stream data to Kinesis Data Streams, which ingests and stores multiple data streams for processing by analytics applications.

The Amazon Pinpoint event stream includes information about user interactions with applications (apps) that you connect to Amazon Pinpoint. It also includes information about all the messages that you send from campaigns, through any channel, and from journeys. This can also include any custom events that you've defined. Finally, it includes information about all the transactional email and SMS messages that you send.

Note
Amazon Pinpoint doesn't stream information about transactional push notifications or voice messages.

This chapter provides information about setting up Amazon Pinpoint to stream event data to Kinesis. It also contains examples of the event data that Amazon Pinpoint streams.

Topics

Setting up event streaming

You can set up Amazon Pinpoint to send event data to an Amazon Kinesis stream or an Amazon Kinesis Data Firehose delivery stream. Amazon Pinpoint can send event data for campaigns, journeys, and transactional email and SMS messages.

This section includes information about setting up event streaming programmatically. You can also use the Amazon Pinpoint console to set up event streaming. For information about setting up event streaming by using the Amazon Pinpoint console, see Event stream settings in the *Amazon Pinpoint User Guide*.

Prerequisites

The examples in this section require the following input:

- The application ID of an application that's integrated with Amazon Pinpoint and reporting events. For information about how to integrate, see Integrating Amazon Pinpoint with your application (p. 108).
- The Amazon Resource Name (ARN) of a Kinesis stream or Kinesis Data Firehose delivery stream in your AWS account. For information about creating these resources, see Creating and updating 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. 363).

AWS CLI

The following AWS CLI example uses the put-event-stream command. This command configures Amazon Pinpoint to send events to a Kinesis stream:

```
aws pinpoint put-event-stream \
--application-id projectId \
--write-event-stream DestinationStreamArn=streamArn,RoleArn=roleArn
```

AWS SDK for Java

The following Java example configures Amazon Pinpoint to send events to a 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 application. The `PutEventStreamRequest` object is passed to the `putEventStream` method of the Amazon Pinpoint client.

You can assign a Kinesis stream to multiple applications. If you do this, Amazon Pinpoint sends event data encoded in base64 from each application to the stream, which enables you to analyze the data as a collection. The following example method accepts a list of application (app) IDs, and it uses the previous example method, `createEventStream`, to assign a stream to each application:

```
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());
}
```

Although you can assign one stream to multiple applications, you can't assign multiple streams to one application.

Disabling event streaming

If you assign a Kinesis stream to an application, you can disable event streaming for that application. Amazon Pinpoint stops streaming the events to Kinesis, but you can view event analytics by using 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));
```

App events

After you integrate your application (app) with Amazon Pinpoint, Amazon Pinpoint can stream event data about user activity, custom events, and message deliveries for the app.

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-e4967example",
    "start_timestamp": 1487973202531,
    "stop_timestamp": 1487973802507
  },
  "attributes": {},
  "metrics": {}
}
```

App event attributes

This section defines the attributes that are included in the app event stream.

Attribute	Description
`event_type`	The type of event. Possible values are: • **_session.start** – The endpoint began a new session. • **_session.stop** – The endpoint ended a session. • **_userauth.sign_in** – The endpoint logged in to your app. • **_userauth.sign_up** – A new endpoint completed the registration process in your app. • **_userauth.auth_fail** – The endpoint attempted to sign in to your app, but wasn't able to complete the process. • **_monetization.purchase** – The endpoint made a purchase in your app. • **_session.pause** – The endpoint paused a session. Paused sessions can be resumed so that you can continue to collect metrics without starting an entirely new session. • **_session.resume** – The endpoint resumed a session.
`event_timestamp`	The time when the event was reported, shown as Unix time in milliseconds.

Attribute	Description
`arrival_timestamp`	The time when the event was received by Amazon Pinpoint, shown as Unix time in milliseconds.
`event_version`	The version of the event JSON schema. **Tip** Check this version in your event-processing application so that you know when to update the application in response to a schema update.
`application`	Information about the Amazon Pinpoint project that's associated with the event. For more information, see the Application (p. 210) table.
`client`	Information about the endpoint that reported the event. For more information, see the Client (p. 211) table.
`device`	Information about the device that reported the event. For more information, see the Device (p. 211) table.
`session`	Information about the session that generated the event. For more information, see the Session (p. 212) table.
`attributes`	Attributes that are associated with the event. For events that are reported by your apps, this object includes custom attributes that you define.
`metrics`	Metrics that are related to the event. You can optionally configure your apps to send custom metrics to Amazon Pinpoint.

Application

Includes information about the Amazon Pinpoint project that the event is associated with.

Attribute	Description
`app_id`	The unique ID of the Amazon Pinpoint project that reported the event.
`cognito_identity_pool_id`	The ID of the Amazon Cognito Identity Pool that the endpoint is associated with.
`package_name`	The name of the app package, such as `com.example.my_app`.
`sdk`	Information about the SDK that was used to report the event. For more information, see the SDK (p. 211) table.
`title`	The name of the app.
`version_name`	The name of the version of the app, such as `V2.5`.

Attribute	Description
`version_code`	The version number of the app, such as `3`.

SDK

Includes information about the SDK that was used to report the event.

Attribute	Description
`name`	The name of the SDK that was used to report the event.
`version`	The version of the SDK.

Client

Includes information about the endpoint that generated the event.

Attribute	Description
`client_id`	The ID of the endpoint.
`cognito_id`	The Amazon Cognito ID token that's associated with the endpoint.

Device

Includes information about the device of the endpoint that generated the event.

Attribute	Description
`locale`	Information about the language and region settings for the endpoint's device. For more information, see the Locale (p. 211) table.
`make`	The manufacturer of the endpoint's device.
`model`	The model identifier of the endpoint's device.
`platform`	Information about the operating system on the endpoint's device. For more information, see the Platform (p. 212) table.

Locale

Includes information about the language and region settings for the endpoint's device.

Attribute	Description
`code`	The locale identifier that's associated with the device.

Attribute	Description
`country`	The country or region that's associated with the device's locale.
`language`	The language that's associated with the device's locale.

Platform

Includes information about the operating system on the endpoint's device.

Attribute	Description
`name`	The name of the operating system on the device.
`version`	The version of the operating system on the device.

Session

Includes information about the session that generated the event.

Attribute	Description
`session_id`	A unique ID that identifies the session.
`start_timestamp`	The date and time when the session began, shown as Unix time in milliseconds.
`stop_timestamp`	The date and time when the session ended, shown as Unix time in milliseconds.

Campaign events

If you use Amazon Pinpoint to send campaigns through any channel, Amazon Pinpoint can stream event data about those campaigns. This includes event data for any email or SMS messages that you send from a campaign. For detailed information about the data that Amazon Pinpoint streams for those types of messages, see the section called "Email events" (p. 219) and the section called "SMS events" (p. 228).

Sample event

The JSON object for a campaign event contains the data shown in the following sample.

```
{
  "event_type": "_campaign.send",
  "event_timestamp": 1562109497426,
  "arrival_timestamp": 1562109497494,
  "event_version": "3.1",
  "application": {
    "app_id": "a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6",
    "sdk": {}
  },
  "client": {
    "client_id": "d8dcf7c5-e81a-48ae-8313-f540cexample"
  },
  "device": {
    "platform": {}
  },
  "session": {},
  "attributes": {
    "treatment_id": "0",
    "campaign_activity_id": "5473285727f04865bc673e527example",
    "delivery_type": "GCM",
    "campaign_id": "4f8d6097c2e8400fa3081d875example",
    "campaign_send_status": "SUCCESS"
  },
  "client_context": {
    "custom": {
      "endpoint": "{\"ChannelType\":\"GCM\",\"EndpointStatus\":\"ACTIVE\",
          #\"OptOut\":\"NONE\",\"RequestId\":\"ec229696-9d1e-11e9-8bf1-85d0aexample\",
          #\"EffectiveDate\":\"2019-07-02T23:12:54.836Z\",\"User\":{}}"
    }
  },
  "awsAccountId": "123456789012"
}
```

Campaign event attributes

This section defines the attributes that are included in the campaign event stream.

Attribute	Description
`event_type`	The type of event. Possible values are: • **_campaign.send** – Amazon Pinpoint executed the campaign. • **_campaign.opened_notification** – For push notification campaigns, this event type indicates that the recipient tapped the notification to open it. • **_campaign.received_foreground** – For push notification campaigns, this event type indicates that the recipient received the message as a foreground notification. • **_campaign.received_background** – For push notification campaigns, this event type indicates that the recipient received the message as a background notification. **Note** **_campaign.opened_notification**, **_campaign.received_foreground**, and **_campaign.received_background** are returned only if you use AWS Amplify. For more information on integrating your app with AWS Amplify. See Integrating the AWS mobile SDKs or JavaScript library with your application (p. 109).
`event_timestamp`	The time when the event was reported, shown as Unix time in milliseconds.

Attribute	Description
`arrival_timestamp`	The time when the event was received by Amazon Pinpoint, shown as Unix time in milliseconds.
`event_version`	The version of the event JSON schema. **Tip** Check this version in your event-processing application so that you know when to update the application in response to a schema update.
`application`	Information about the Amazon Pinpoint project that's associated with the event. For more information, see the Application (p. 214) table.
`client`	Information about the endpoint that the event is associated with. For more information, see the Client (p. 216) table.
`device`	Information about the device that reported the event. For campaign and transactional messages, this object is empty.
`session`	Information about the session that generated the event. For campaigns, this object is empty.
`attributes`	Attributes that are associated with the event. For events that are reported by one of your apps, this object can include custom attributes that are defined by the app. For events that are created when you send a campaign, this object contains attributes that are associated with the campaign. For events that are generated when you send transactional messages, this object contains information that's related to the message itself. For more information, see the Attributes (p. 215) table.
`client_context`	Contains a `custom` object, which contains an `endpoint` property. The `endpoint` property contains the contents of the endpoint record for the endpoint that the campaign was sent to.
`awsAccountId`	The ID of the AWS account that was used to send the message.

Application

Includes information about the Amazon Pinpoint project that the event is associated with.

Attribute	Description
`app_id`	The unique ID of the Amazon Pinpoint project that reported the event.

Attribute	Description
`sdk`	The SDK that was used to report the event.

Attributes

Includes information about the campaign that produced the event.

Attribute	Description
`treatment_id`	If the message was sent using an A/B test campaign, this value represents the treatment number of the message. For standard campaigns, this value is `0`.
`campaign_activity_id`	The unique ID that Amazon Pinpoint generates when the event occurs.
`delivery_type`	The delivery method for the campaign. This attribute should not be confused with the `ChannelType` field specified under the `endpoint` property of `client_context`. `ChannelType` is typically based on the endpoint that the message is being sent to. For channels that support only one endpoint type — for example, the email channel — the `delivery_type` and `ChannelType` fields have the same value, EMAIL. However, this may not be true for channels that support different endpoint types, such as custom channels. Since a custom channel can be used for different endpoints — EMAIL, SMS, CUSTOM, etc. — the `delivery_type` identifies a custom delivery event, CUSTOM, while the `ChannelType` specifies the type of the endpoint that the campaign was sent to — EMAIL, SMS, CUSTOM, etc. For more information on creating custom channels, see *Creating custom channels* (p. 199). Possible values are: • **EMAIL** • **SMS** • **ADM** • **APNS** • **APNS_SANDBOX** • **APNS_VOIP** • **APNS_VOIP_SANDBOX** • **VOICE** • **GCM** • **BAIDU** • **PUSH** • **CUSTOM**
`campaign_id`	The unique ID of the campaign that the message was sent from.

Attribute	Description
`campaign_send_status`	Indicates the status of the campaign for the target endpoint. Possible values include: • **SUCCESS** – The campaign was successfully sent to the endpoint. • **FAILURE** – The campaign wasn't sent to the endpoint. • **DAILY_CAP** – The campaign wasn't sent to the endpoint because the maximum number of daily messages have already been sent to the endpoint. • **EXPIRED** – The campaign wasn’t sent to the endpoint because sending it would exceed the maximum duration or sending rate settings for the campaign. • **QUIET_TIME** – The campaign wasn't sent to the endpoint due to quiet time restrictions. • **HOLDOUT** – The campaign wasn't sent to the endpoint because the endpoint was a member of the holdout group.

Client

Includes information about the endpoint that was targeted by the campaign.

Attribute	Description
`client_id`	The ID of the endpoint that the campaign was sent to.

Journey events

If you publish a journey, Amazon Pinpoint can stream event data about the journey. This includes event data for any email, SMS, push, or custom messages that you send from the journey.

See the following for information about the data that Amazon Pinpoint streams:

- For email messages, see the section called “Email events” (p. 219).
- For SMS messages, see SMS events.

Sample event

The JSON object for a journey event contains the data shown in the following sample.

```
{
   "event_type":"_journey.send",
   "event_timestamp":1572989078843,
   "arrival_timestamp":1572989078843,
   "event_version":"3.1",
   "application":{
      "app_id":"a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6",
      "sdk":{

      }
   },
   "client":{
      "client_id":"d8dcf7c5-e81a-48ae-8313-f540cexample"
   },
   "device":{
      "platform":{

      }
   },
   "session":{

   },
   "attributes":{
      "journey_run_id":"edc9a0b577164d1daf72ebd15example",
      "journey_send_status":"SUCCESS",
      "journey_id":"546401670c5547b08811ac6a9example",
      "journey_activity_id":"0yKexample",
      "journey_activity_type": "EMAIL"
   },
   "client_context":{
      "custom":{
         "endpoint":"{\"ChannelType\":\"EMAIL\",\"EndpointStatus\":\"ACTIVE\",\"OptOut\":
\"NONE\",\"Demographic\":{\"Timezone\":\"America/Los_Angeles\"}}"
      }
   },
   "awsAccountId":"123456789012"
}
```

Journey event attributes

This section defines the attributes that are included in the event stream data that Amazon Pinpoint generates for a journey.

Attribute	Description
`event_type`	The type of event. For journey events, the value for this attribute is always `_journey.send`, which indicates that Amazon Pinpoint executed the journey.
`event_timestamp`	The time when the event was reported, shown as Unix time in milliseconds.
`arrival_timestamp`	The time when the event was received by Amazon Pinpoint, shown as Unix time in milliseconds.
`event_version`	The version of the event JSON schema. **Tip** Check this version in your event-processing application so that you know when to update the application in response to a schema update.
`application`	Information about the Amazon Pinpoint project that's associated with the event. For more information, see the Application (p. 218) table.
`client`	Information about the endpoint that's associated with the event. For more information, see the Client (p. 218) table.

Attribute	Description
`device`	Information about the device that reported the event. For journeys, this object is empty.
`session`	Information about the session that generated the event. For journeys, this object is empty.
`attributes`	Attributes that are associated with the journey and journey activity that generated the event. For more information, see the Attributes (p. 218) table.
`client_context`	Contains a `custom` object, which contains an `endpoint` property. The `endpoint` property contains the contents of the endpoint record for the endpoint that's associated with the event.
`awsAccountId`	The ID of the AWS account that was used to execute the journey.

Application

Includes information about the Amazon Pinpoint project that's associated with the event.

Attribute	Description
`app_id`	The unique ID of the Amazon Pinpoint project that reported the event.
`sdk`	The SDK that was used to report the event.

Client

Includes information about the endpoint that's associated with the event.

Attribute	Description
`client_id`	The ID of the endpoint.

Attributes

Includes information about the journey that generated the event.

Attribute	Description
`journey_run_id`	The unique ID of the journey run that generated the event. Amazon Pinpoint generates and assigns this ID automatically to each new run of a journey.
`journey_send_status`	Indicates the delivery status of the message that's associated with the event. Possible values include: • **SUCCESS** – The message was successfully sent to the endpoint.

Attribute	Description
	• **FAILURE** – The message wasn't sent to the endpoint because an error occurred. • **DAILY_CAP** – The message wasn't sent to the endpoint because sending the message would exceed the maximum number of messages that the journey or project can send to a single endpoint during a 24-hour period. • **QUIET_TIME** – The message wasn't sent because of quiet-time restrictions for the journey or project.
`journey_id`	The unique ID of the journey that generated the event.
`journey_activity_id`	The unique ID of the journey activity that generated the event.
`journey_activity_type`	The event's journey activity type. This can be **EMAIL**, **SMS**, **PUSH**, or **CUSTOM**. **Note** **VOICE** is not a supported journey activity type.

Email events

When you send email messages, Amazon Pinpoint can stream data that provides additional information about the following types of events for those messages:

- Sends
- Deliveries
- Bounces
- Complaints
- Opens
- Clicks
- Rejections
- Unsubscribes
- Rendering failures

The event types in the preceding list are explained in detail in Email event attributes (p. 224).

Depending on the API and settings that you use to send email messages, you might see additional event types or different data. For example, if you send messages using configuration sets that publish event data to Amazon Kinesis, such as those provided by Amazon Simple Email Service (Amazon SES), the data can also include events for template-rendering failures. For information about that data, see Monitoring using Amazon SES event publishing in the *Amazon Simple Email Service Developer Guide*.

Sample events

Email send

The JSON object for an *email send* event contains the data shown in the following example.

```
{
  "event_type": "_email.send",
  "event_timestamp": 1564618621380,
  "arrival_timestamp": 1564618622025,
  "event_version": "3.1",
  "application": {
    "app_id": "a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6",
    "sdk": {}
  },
  "client": {
    "client_id": "9a311b17-6f8e-4093-be61-4d0bbexample"
  },
  "device": {
    "platform": {}
  },
  "session": {},
  "attributes": {
    "feedback": "received"
  },
  "awsAccountId": "123456789012",
  "facets": {
    "email_channel": {
      "mail_event": {
        "mail": {
          "message_id": "0200000073rnbmd1-mbvdg3uo-q8ia-m3ku-ibd3-ms77kexample-000000",
          "message_send_timestamp": 1564618621380,
          "from_address": "sender@example.com",
          "destination": ["recipient@example.com"],
          "headers_truncated": false,
          "headers": [{
            "name": "From",
            "value": "sender@example.com"
          }, {
            "name": "To",
            "value": "recipient@example.com"
          }, {
            "name": "Subject",
            "value": "Amazon Pinpoint Test"
          }, {
            "name": "MIME-Version",
            "value": "1.0"
          }, {
            "name": "Content-Type",
            "value": "multipart/alternative;  boundary=\"----=_Part_314159_271828\""
          }],
          "common_headers": {
            "from": "sender@example.com",
            "to": ["recipient@example.com"],
            "subject": "Amazon Pinpoint Test"
          }
        },
        "send": {}
      }
    }
  }
}
```

Email delivered

The JSON object for an *email delivered* event contains the data shown in the following example.

```
{
  "event_type": "_email.delivered",
  "event_timestamp": 1564618621380,
  "arrival_timestamp": 1564618622690,
  "event_version": "3.1",
  "application": {
    "app_id": "a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6",
    "sdk": {}
  },
  "client": {
    "client_id": "e9a3000d-daa2-40dc-ac47-1cd34example"
  },
  "device": {
    "platform": {}
  },
  "session": {},
  "attributes": {
    "feedback": "delivered"
  },
  "awsAccountId": "123456789012",
  "facets": {
    "email_channel": {
      "mail_event": {
        "mail": {
          "message_id": "0200000073rnbmd1-mbvdg3uo-q8ia-m3ku-ibd3-ms77kexample-000000",
          "message_send_timestamp": 1564618621380,
          "from_address": "sender@example.com",
          "destination": ["recipient@example.com"],
          "headers_truncated": false,
          "headers": [{
            "name": "From",
            "value": "sender@example.com"
          }, {
            "name": "To",
            "value": "recipient@example.com"
          }, {
            "name": "Subject",
            "value": "Amazon Pinpoint Test"
          }, {
            "name": "MIME-Version",
            "value": "1.0"
          }, {
            "name": "Content-Type",
            "value": "multipart/alternative;  boundary=\"----=_Part_314159_271828\""
          }],
          "common_headers": {
            "from": "sender@example.com",
            "to": ["recipient@example.com"],
            "subject": "Amazon Pinpoint Test"
          }
        },
        "delivery": {
          "smtp_response": "250 ok:  Message 82080542 accepted",
          "reporting_mta": "a8-53.smtp-out.amazonses.com",
          "recipients": ["recipient@example.com"],
          "processing_time_millis": 1310
        }
      }
    }
  }
}
```

Email click

The JSON object for an *email click* event contains the data shown in the following example.

```
{
  "event_type": "_email.click",
  "event_timestamp": 1564618621380,
  "arrival_timestamp": 1564618713751,
  "event_version": "3.1",
  "application": {
    "app_id": "a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6",
    "sdk": {}
  },
  "client": {
    "client_id": "49c1413e-a69c-46dc-b1c4-6470eexample"
  },
  "device": {
    "platform": {}
  },
  "session": {},
  "attributes": {
    "feedback": "https://aws.amazon.com/pinpoint/"
  },
  "awsAccountId": "123456789012",
  "facets": {
    "email_channel": {
      "mail_event": {
        "mail": {
          "message_id": "0200000073rnbmd1-mbvdg3uo-q8ia-m3ku-ibd3-ms77kexample-000000",
          "message_send_timestamp": 1564618621380,
          "from_address": "sender@example.com",
          "destination": ["recipient@example.com"],
          "headers_truncated": false,
          "headers": [{
            "name": "From",
            "value": "sender@example.com"
          }, {
            "name": "To",
            "value": "recipient@example.com"
          }, {
            "name": "Subject",
            "value": "Amazon Pinpoint Test"
          }, {
            "name": "MIME-Version",
            "value": "1.0"
          }, {
            "name": "Content-Type",
            "value": "multipart/alternative;  boundary=\"----=_Part_314159_271828\""
          }, {
            "name": "Message-ID",
            "value": "null"
          }],
          "common_headers": {
            "from": "sender@example.com",
            "to": ["recipient@example.com"],
            "subject": "Amazon Pinpoint Test"
          }
        },
        "click": {
          "ip_address": "72.21.198.67",
          "user_agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_6)
 AppleWebKit/605.1.15 (KHTML, like Gecko) Version/12.1.2 Safari/605.1.15",
          "link": "https://aws.amazon.com/pinpoint/"
        }
      }
    }
  }
}
```

Email open

The JSON object for an *email open* event contains the data shown in the following example.

```
{
  "event_type": "_email.open",
  "event_timestamp": 1564618621380,
  "arrival_timestamp": 1564618712316,
  "event_version": "3.1",
  "application": {
    "app_id": "a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6",
    "sdk": {}
  },
  "client": {
    "client_id": "8dc1f651-b3ec-46fc-9b67-2a050example"
  },
  "device": {
    "platform": {}
  },
  "session": {},
  "attributes": {
    "feedback": "opened"
  },
  "awsAccountId": "123456789012",
  "facets": {
    "email_channel": {
      "mail_event": {
        "mail": {
          "message_id": "0200000073rnbmd1-mbvdg3uo-q8ia-m3ku-ibd3-ms77kexample-000000",
          "message_send_timestamp": 1564618621380,
          "from_address": "sender@example.com",
          "destination": ["recipient@example.com"],
          "headers_truncated": false,
          "headers": [{
            "name": "From",
            "value": "sender@example.com"
          }, {
            "name": "To",
            "value": "recipient@example.com"
          }, {
            "name": "Subject",
            "value": "Amazon Pinpoint Test"
          }, {
            "name": "MIME-Version",
            "value": "1.0"
          }, {
            "name": "Content-Type",
            "value": "multipart/alternative;  boundary=\"----=_Part_314159_271828\""
          }, {
            "name": "Message-ID",
            "value": "null"
          }],
          "common_headers": {
            "from": "sender@example.com",
            "to": ["recipient@example.com"],
            "subject": "Amazon Pinpoint Test"
          }
        },
        "open": {
          "ip_address": "72.21.198.67",
          "user_agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_6)
 AppleWebKit/605.1.15 (KHTML, like Gecko)"
        }
      }
    }
  }
}
```

Email event attributes

This section defines the attributes that are included in the event stream data that Amazon Pinpoint generates when you send email messages.

Attribute	Description
`event_type`	The type of event. Possible values are: • **_email.send** – Amazon Pinpoint accepted the message and attempted to deliver it to the recipient. • **_email.delivered** – The message was delivered to the recipient. • **_email.rejected** – Amazon Pinpoint determined that the message contained malware and didn't attempt to send it. • **_email.hardbounce** – A permanent issue prevented Amazon Pinpoint from delivering the message. Amazon Pinpoint won't attempt to deliver the message again. • **_email.softbounce** – A temporary issue prevented Amazon Pinpoint from delivering the message. Amazon Pinpoint will attempt to deliver the message again for a certain amount of time. If the message still can't be delivered, no more retries will be attempted. The final state of the email will then be SOFTBOUNCE. • **_email.complaint** – The recipient received the message, and then reported the message to their email provider as spam (for example, by using the "Report Spam" feature of their email client). • **_email.open** – The recipient received the message and opened it. • **_email.click** – The recipient received the message and clicked a link in it. • **_email.unsubscribe** – The recipient received the message and clicked an unsubscribe link in it. • **_email.rendering_failure** – The email was not sent due to a rendering failure. This can occur when template data is missing or when there is a mismatch between template parameters and data.
`event_timestamp`	The time when the message was sent, shown as Unix time in milliseconds. This value is typically the same for all the events that are generated for a message.
`arrival_timestamp`	The time when the event was received by Amazon Pinpoint, shown as Unix time in milliseconds.
`event_version`	The version of the event JSON schema. **Tip** Check this version in your event-processing application so that you know when to update

Attribute	Description
	the application in response to a schema update.
`application`	Information about the Amazon Pinpoint project that's associated with the event. See the *Application* table for more information.
`client`	Information about the app client that's installed on the device that reported the event. For more information, see the *Client* table.
`device`	Information about the device that reported the event. For more information, see the *Device* table. For email events, this object is empty.
`session`	For email events, this object is empty.
`attributes`	Attributes that are associated with the event. For more information, see the *Attributes* table. For events that are reported by one of your apps, this object can include custom attributes that are defined by the app. For events that are created when you send a message from a campaign or journey, this object contains attributes that are associated with the campaign or journey. For events that are generated when you send transactional messages, this object contains information that's related to the message itself.
`client_context`	For email events, this object contains a `custom` object, which contains a `legacy_identifier` attribute. The value for the `legacy_identifier` attribute is the ID of the project that the message was sent from.
`facets`	Additional information about the message, such as the email headers. See the *Facets* table for more information.
`awsAccountId`	The ID of the AWS account that was used to send the message.

Application

Includes information about the Amazon Pinpoint project that the event is associated with.

Attribute	Description
`app_id`	The unique ID of the Amazon Pinpoint project that reported the event.
`sdk`	The SDK that was used to report the event. If you send a transactional email message by calling the Amazon Pinpoint API directly or by using the Amazon Pinpoint console, this object is empty.

Attributes

Includes information about the campaign or journey that produced the event.

Campaign

Includes information about the campaign that produced the event.

Attribute	Description
`feedback`	For `_email.click` events, the value for this attribute is the URL of the link that the recipient clicked in the message to generate the event. For other events, this value represents the event type, such as `received`, `opened`, or `clicked`.
`treatment_id`	If the message was sent using an A/B test campaign, this value represents the treatment number of the message. For standard campaigns and transactional email messages, this value is `0`.
`campaign_activity_id`	The unique ID that Amazon Pinpoint generates when the event occurs.
`campaign_id`	The unique ID of the campaign that sent the message.

Journey

Includes information about the journey that produced the event.

Attribute	Description
`journey_run_id`	The unique ID of the journey run that sent the message. Amazon Pinpoint generates and assigns this ID automatically to each new run of a journey.
`feedback`	For `_email.click` events, the value for this attribute is the URL of the link that the recipient clicked in the message to generate the event. For other events, this value represents the event type, such as `received`, `delivered`, or `opened`.
`journey_id`	The unique ID of the journey that sent the message.
`journey_activity_id`	The unique ID of the journey activity that sent the message.

Client

Includes information about the endpoint that was targeted by the campaign or journey.

Attribute	Description
`client_id`	The email address of the endpoint that reported the event.

Facets

Includes information about the message and the event type.

Attribute	Description
`email_channel`	Contains a `mail_event` object, which contains two objects: `mail`, and an object that corresponds with the event type.

Mail

Includes information about the content of the email message, and metadata about the message.

Attribute	Description
`message_id`	The unique ID of the message. Amazon Pinpoint automatically generates this ID when it accepts the message.
`message_send_timestamp`	The date and time when the message was sent, in the format specified in RFC 822.
`from_address`	The email address that the message was sent from.
`destination`	An array that contains the email addresses that the message was sent to.
`headers_truncated`	A Boolean value that indicates whether the email headers were truncated.
`headers`	An object that contains several name-value pairs that correspond to the headers in the message. This object typically contains information about the following headers: • `From` – The sender's email address. • `To` – The recipient's email address. • `Subject` – The subject line of the email. • `MIME-Version` – Indicates that the message is in MIME format. If this header is present, the value is always `1.0`. • `Content-Type` – The MIME media type of the message content.
`common_headers`	Contains information about several common headers for email messages. The information can include the date when the message was sent, and the to, from, and subject lines of the message.

SMS events

If the SMS channel is enabled for a project, Amazon Pinpoint can stream event data about SMS message deliveries for the project.

Example

The JSON object for an SMS event contains the data shown in the following example.

```
{
  "event_type": "_SMS.SUCCESS",
  "event_timestamp": 1553104954322,
  "arrival_timestamp": 1553104954064,
  "event_version": "3.1",
  "application": {
    "app_id": "a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6",
    "sdk": {}
  },
  "client": {
    "client_id": "123456789012"
  },
  "device": {
    "platform": {}
  },
  "session": {},
  "attributes": {
    "sender_request_id": "565d4425-4b3a-11e9-b0a5-example",
    "campaign_activity_id": "cbcfc3c5e3bd48a8ae2b9cb41example",
    "origination_phone_number": "+12065550142",
    "destination_phone_number": "+14255550199",
    "record_status": "DELIVERED",
    "iso_country_code": "US",
    "treatment_id": "0",
    "number_of_message_parts": "1",
    "message_id": "1111-2222-3333",
    "message_type": "Transactional",
    "campaign_id": "52dc44b35c4742c98c5935269example"
  },
  "metrics": {
    "price_in_millicents_usd": 645.0
  },
  "awsAccountId": "123456789012"
}
```

SMS event attributes

This section defines the attributes that are included in the event stream data that Amazon Pinpoint generates when you send SMS messages.

Event

Attribute	Description
`event_type`	The type of event. Possible values are: • **_SMS.BUFFERED** – The message is still in the process of being delivered to the recipient. • **_SMS.SUCCESS** – The message was successfully accepted by the carrier/delivered to the recipient.

Attribute	Description
	• **_SMS.FAILURE** – Amazon Pinpoint wasn't able to deliver the message to the recipient. To learn more about the error that prevented the message from being delivered, see `attributes.record_status`. • **_SMS.OPTOUT** – The customer received the message and replied by sending the opt-out keyword (usually "STOP").
`event_timestamp`	The time when the event was reported, shown as Unix time in milliseconds.
`arrival_timestamp`	The time when the event was received by Amazon Pinpoint, shown as Unix time in milliseconds.
`event_version`	The version of the event JSON schema. **Tip** Check this version in your event-processing application so that you know when to update the application in response to a schema update.
`application`	Information about the Amazon Pinpoint project that's associated with the event. For more information, see the Application (p. 230) table.
`client`	Information about the app client installed on the device that reported the event. For more information, see the Client (p. 232) table.
`device`	Information about the device that reported the event. For more information, see the Device (p. 232) table. For SMS events, this object is empty.
`session`	For SMS events, this object is empty.
`attributes`	Attributes that are associated with the event. For events that are reported by one of your apps, this object can include custom attributes that are defined by the app. For events that are created when you send a campaign, this object contains attributes that are associated with the campaign. For events that are generated when you send transactional messages, this object contains information that's related to the message itself. For more information, see the Attributes (p. 230) table.
`metrics`	Additional metrics that are associated with the event. For more information, see the Metrics (p. 232) table.
`awsAccountId`	The ID of the AWS account that was used to send the message.

Application

Includes information about the Amazon Pinpoint project that the event is associated with and, if applicable, the SDK that was used to report the event.

Attribute	Description
`app_id`	The unique ID of the Amazon Pinpoint project that reported the event.
`sdk`	The SDK that was used to report the event. If you send a transactional SMS message by calling the Amazon Pinpoint API directly or by using the Amazon Pinpoint console, this object is empty.

Attributes

Includes information about the attributes that are associated with the event.

Attribute	Description
`sender_request_id`	A unique ID that's associated with the request to send the SMS message.
`campaign_activity_id`	The unique ID of the activity within the campaign.
`origination_phone_number`	The phone number that the message was sent from.
`destination_phone_number`	The phone number that you attempted to send the message to.
`record_status`	Additional information about the status of the message. Possible values include: • **SUCCESSFUL** – The message was accepted by the carrier. • **DELIVERED** – The message was delivered to the recipient's device. • **PENDING** – The message hasn't yet been delivered to the recipient's device. • **INVALID** – The destination phone number is invalid. • **UNREACHABLE** – The recipient's device is currently unreachable or unavailable. For example, the device might be powered off, or might be disconnected from the network. You can try to send the message again later. • **UNKNOWN** – An error occurred that prevented the delivery of the message. This error is usually transient, and you can attempt to send the message again later. • **BLOCKED** – The recipient's device is blocking SMS messages from the origination number. • **CARRIER_UNREACHABLE** – An issue with the mobile network of the recipient prevented the message from

Attribute	Description
	being delivered. This error is usually transient, and you can attempt to send the message again later. • **SPAM** – The recipient's mobile carrier identified the contents of the message as spam and blocked delivery of the message. • **INVALID_MESSAGE** – The body of the SMS message is invalid and can't be delivered. • **CARRIER_BLOCKED** – The recipient's carrier has blocked delivery of this message. This often occurs when the carrier identifies the contents of the message as unsolicited or malicious. • **TTL_EXPIRED** – The SMS message couldn't be delivered within a certain time frame. This error is usually transient, and you can attempt to send the message again later. • **MAX_PRICE_EXCEEDED** – Sending the message would have resulted in a charge that exceeded the monthly SMS spending quota for your account. You can request an increase to this quota by completing the procedure in Requesting increases to your monthly SMS spending quota in the *Amazon Pinpoint User Guide*.
`iso_country_code`	The country that's associated with the recipient's phone number, shown in ISO 3166-1 alpha-2 format.
`treatment_id`	The ID of the message treatment, if the message was sent in an A/B campaign.
`treatment_id`	If the message was sent using an A/B test campaign, this value represents the treatment number of the message. For transactional SMS messages, this value is 0.
`number_of_message_parts`	The number of message parts that Amazon Pinpoint created in order to send the message. Generally, SMS messages can contain only 160 GSM-7 characters or 67 non-GSM characters, although these limits can vary by country . If you send a message that exceeds these limits, Amazon Pinpoint automatically splits the message into smaller parts. We bill you based on the number of message parts that you send.
`message_id`	The unique ID that Amazon Pinpoint generates when it accepts the message.
`message_type`	The type of message. Possible values are **Promotional** and **Transactional**. You specify this value when you create a campaign, or when you send transactional messages by using the SendMessages operation in the Amazon Pinpoint API.
`campaign_id`	The unique ID of the Amazon Pinpoint campaign that sent the message.

Client

Includes information about the app client that's installed on the device that reported the event.

Attribute	Description
`client_id`	For events that are generated by apps, this value is the unique ID of the app client that's installed on the device. This ID is automatically generated by the AWS Mobile SDK for iOS and the AWS Mobile SDK for Android. For events that are generated when you send campaigns and transactional messages, this value is equal to the ID of the endpoint that you sent the message to.
`cognito_id`	The unique ID assigned to the app client in the Amazon Cognito identity pool used by your app.

Device

Includes information about the device that reported the event.

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`.

Metrics

Includes information about metrics that are associated with the event.

Attribute	Description
`price_in_millicents_usd`	The amount that we charged you to send the message. This price is shown in thousandths of a United States cent. For example, if the value of this attribute is `645`, then we charged you 0.645¢ to send the message (645 / 1000 = 0.645¢ = $0.00645). **Note** This property doesn't appear for messages with an `event_type` of **_SMS.BUFFERED**.

Querying Amazon Pinpoint analytics data

In addition to using the analytics pages on the Amazon Pinpoint console, you can use Amazon Pinpoint Analytics APIs to query analytics data for a subset of standard metrics that provide insight into trends related to user engagement, campaign outreach, and more. These metrics, also referred to as *key performance indicators (KPIs)*, are measurable values that can help you monitor and assess the performance of your projects, campaigns, and journeys.

If you use the APIs to query analytics data, you can analyze the data by using the reporting tool of your choice, without having to sign in to the Amazon Pinpoint console or analyze raw event data from sources such as Amazon Kinesis streams. For example, you can build a custom dashboard that displays weekly campaign results or provides in-depth analytics about delivery rates for your campaigns.

You can query the data by using the Amazon Pinpoint REST API, the AWS Command Line Interface (AWS CLI), or an AWS SDK. To query the data, you send a request to the Amazon Pinpoint API and use supported parameters to specify the data that you want and any filters that you want to apply. After you submit your query, Amazon Pinpoint returns the query results in a JSON response. You can then pass the results to another service or application for deeper analysis, storage, or reporting.

Supported metrics

Amazon Pinpoint provides programmatic access to analytics data for several types of standard metrics:

- **Application metrics** – These metrics provide insight into trends for all the campaigns and transactional messages that are associated with a project. For example, you can use an application metric to get a breakdown of the number of messages that were opened by recipients for each campaign that's associated with a project. To access the data for an application metric, use the Application metrics resource of the Amazon Pinpoint API.
- **Campaign metrics** – These metrics provide insight into the performance of individual campaigns. For example, you can use a campaign metric to determine how many endpoints a campaign message was sent to. To access the data for a campaign metric, use the Campaign metrics resource of the Amazon Pinpoint API.
- **Journey engagement metrics** – These metrics provide insight into the performance of individual journeys. For example, you can use a journey engagement metric to get a breakdown of the number of messages that were opened by participants in each activity of a journey. To access the data for a journey engagement metric, use the Journey engagement metrics resource of the Amazon Pinpoint API.
- **Journey execution metrics** – These metrics provide insight into participation trends for individual journeys. For example, you can use a journey execution metric to determine how many participants are proceeding through the activities in the journey. To access the data for a journey execution metric, use the Journey Execution Metrics resource of the Amazon Pinpoint API.
- **Journey activity execution metrics** – These metrics provide insight into participation trends for individual activities in a journey. For example, you can use a journey activity execution metric to determine how many participants completed an activity. To access the data for a journey activity execution metric, use the Journey activity execution metrics resource of the Amazon Pinpoint API.

For a complete list of standard metrics that you can query programmatically, see Standard metrics (p. 237).

Amazon Pinpoint automatically collects and aggregates data for all supported metrics, for all of your projects, campaigns, and journeys. In addition, the data is updated continuously, resulting in a data latency time frame that's limited to approximately two hours. Note, however, that there may be additional data latency for certain metrics. This is because the data for some metrics is based on information that we receive from recipients' email providers. Some providers send us this information immediately, while others send it less frequently.

Amazon Pinpoint stores the data for 90 days. To store the data for more than 90 days or to access raw analytics data in real time, you can configure an Amazon Pinpoint project to stream event data to Amazon Kinesis Data Streams or Amazon Kinesis Data Firehose. For information about configuring event streams, see Streaming Amazon Pinpoint events to Kinesis (p. 206).

Query basics

To query the data for a metric, you send a `get` request to the appropriate metrics resource of the Amazon Pinpoint API. In your request, you define your query by using supported parameters for the following query components:

- **Project** – Specify a project by providing the project ID as the value for the `application-id` parameter. This parameter is required for all metrics.
- **Campaign** – Specify a campaign by providing the campaign ID as the value for the `campaign-id` parameter. This parameter is required only for campaign metrics.
- **Journey** – Specify a journey by providing the journey ID as the value for the `journey-id` parameter. This parameter is required only for journey engagement and execution metrics, and journey activity execution metrics.
- **Journey activity** – Specify a journey activity by providing the journey activity ID as the value for the `journey-activity-id` parameter. This parameter is required only for journey activity execution metrics.
- **Date range** – To optionally filter the data by date range, provide the first and last date and time of the date range by using supported start and end time parameters. The values should be in extended ISO 8601 format and use Coordinated Universal Time (UTC)—for example, `2019-07-19T20:00:00Z` for 8:00 PM UTC July 19, 2019.

  Date ranges are inclusive and must be limited to 31 or fewer calendar days. In addition, the first date and time must be fewer than 90 days from the current day. If you don't specify a date range, Amazon Pinpoint returns the data for the preceding 31 calendar days. Date range parameters are supported by all metrics except journey execution metrics and journey activity execution metrics.
- **Metric** – Specify the metric by providing the name of the metric as the value for the `kpi-name` parameter. This value describes the associated metric and consists of two or more terms, which are comprised of lowercase alphanumeric characters, separated by a hyphen. Examples are `email-open-rate` and `successful-delivery-rate`. This parameter is required for all metrics except journey execution metrics and journey activity execution metrics. For a complete list of supported metrics and the `kpi-name` value to use for each one, see Standard metrics (p. 237).

After you send your query, Amazon Pinpoint returns the query results in a JSON response. In the response, the structure of the results varies depending on the metric that you queried.

Some metrics provide only one value—for example, the number of messages that were delivered by a campaign. Other metrics provide multiple values and typically group those values by a relevant field —for example, the number of messages that were delivered by each run of a campaign, grouped by campaign run. If a metric provides and groups multiple values, the JSON response includes a field that indicates which field was used to group the data. To learn more about the structure of query results, see Using query results (p. 279).

IAM policies for querying Amazon Pinpoint analytics data

By using the Amazon Pinpoint API, you can query analytics data for a subset of standard metrics, also referred to as *key performance indicators (KPIs)* that apply to Amazon Pinpoint projects, campaigns, and journeys. These metrics can help you monitor and assess the performance of projects, campaigns, and journeys.

To manage access to this data, you can create AWS Identity and Access Management (IAM) policies that define permissions for IAM roles or users who are authorized to access the data. To support granular control of access to this data, Amazon Pinpoint provides several distinct actions that you can specify in IAM policies. There is a distinct action for viewing analytics data on the Amazon Pinpoint console (`mobiletargeting:GetReports`), and there are other actions for accessing analytics data programmatically by using the Amazon Pinpoint API.

To create IAM policies that manage access to analytics data, you can use the AWSManagement Console, the AWS CLI, or the IAM API. Note that the **Visual editor** tab on the AWSManagement Console doesn't currently include actions for viewing or querying Amazon Pinpoint analytics data. However, you can add the necessary actions to IAM policies manually by using the **JSON** tab on the console.

For example, the following policy allows programmatic access to all the analytics data for all of your projects, campaigns, and journeys in all AWS Regions:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "QueryAllAnalytics",
            "Effect": "Allow",
            "Action": [
                "mobiletargeting:GetApplicationDateRangeKpi",
                "mobiletargeting:GetCampaignDateRangeKpi",
                "mobiletargeting:GetJourneyDateRangeKpi",
                "mobiletargeting:GetJourneyExecutionMetrics",
                "mobiletargeting:GetJourneyExecutionActivityMetrics"
            ],
            "Resource": [
                "arn:aws:mobiletargeting:*:accountId:apps/*/kpis/*",
                "arn:aws:mobiletargeting:*:accountId:apps/*/campaigns/*/kpis/*",
                "arn:aws:mobiletargeting:*:accountId:apps/*/journeys/*/kpis/*",
                "arn:aws:mobiletargeting:*:accountId:apps/*/journeys/*/execution-metrics",
                "arn:aws:mobiletargeting:*:accountId:apps/*/journeys/*/activities/*/
execution-metrics"
            ]
        }
    ]
}
```

Where *`accountId`* is your AWS account ID.

However, as a best practice, you should create policies that follow the principle of *least privilege*. In other words, you should create policies that include only the permissions that are required to perform a specific task. To support this practice and implement more granular control, you can restrict programmatic access to the analytics data for only a particular project in a specific AWS Region, for example:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "QueryProjectAnalytics",
            "Effect": "Allow",
            "Action": [
                "mobiletargeting:GetApplicationDateRangeKpi",
                "mobiletargeting:GetCampaignDateRangeKpi",
                "mobiletargeting:GetJourneyDateRangeKpi",
                "mobiletargeting:GetJourneyExecutionMetrics",
                "mobiletargeting:GetJourneyExecutionActivityMetrics"
            ],
            "Resource": [
                "arn:aws:mobiletargeting:region:accountId:apps/projectId/kpis/*",
                "arn:aws:mobiletargeting:region:accountId:apps/projectId/campaigns/*/kpis/
*",
                "arn:aws:mobiletargeting:region:accountId:apps/projectId/journeys/*/kpis/
*",
                "arn:aws:mobiletargeting:region:accountId:apps/projectId/journeys/*/
execution-metrics",
                "arn:aws:mobiletargeting:region:accountId:apps/projectId/journeys/*/
activities/*/execution-metrics"
            ]
        }
    ]
}
```

Where:

- *region* is the name of the AWS Region that hosts the project.
- *accountId* is your AWS account ID.
- *projectId* is the identifier for the project that you want to provide access to.

Similarly, the following example policy allows programmatic access to the analytics data for only a particular campaign:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "QueryCampaignAnalytics",
            "Effect": "Allow",
            "Action": "mobiletargeting:GetCampaignDateRangeKpi",
            "Resource": "arn:aws:mobiletargeting:region:accountId:apps/projectId/
campaigns/campaignId/kpis/*"
        }
    ]
}
```

Where:

- *region* is the name of the AWS Region that hosts the project.
- *accountId* is your AWSaccount ID.
- *projectId* is the identifier for the project that's associated with the campaign.
- *campaignId* is the identifier for the campaign that you want to provide access to.

And the following example policy allows programmatic access to all the analytics data, both engagement and execution data, for a particular journey and the activities that comprise that journey:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "QueryJourneyAnalytics",
            "Effect": "Allow",
            "Action": [
                "mobiletargeting:GetJourneyDateRangeKpi",
                "mobiletargeting:GetJourneyExecutionMetrics",
                "mobiletargeting:GetJourneyExecutionActivityMetrics"
            ],
            "Resource": [
                "arn:aws:mobiletargeting:region:accountId:apps/projectId/
journeys/journeyId/kpis/*",
                "arn:aws:mobiletargeting:region:accountId:apps/projectId/
journeys/journeyId/execution-metrics",
                "arn:aws:mobiletargeting:region:accountId:apps/projectId/
journeys/journeyId/activities/*/execution-metrics"
            ]
        }
    ]
}
```

Where:

- *region* is the name of the AWS Region that hosts the project.
- *accountId* is your AWS account ID.
- *projectId* is the identifier for the project that's associated with the journey.
- *journeyId* is the identifier for the journey that you want to provide access to.

For a complete list of Amazon Pinpoint API actions that you can use in IAM policies, see Amazon Pinpoint actions for IAM policies (p. 326). For detailed information about creating and managing IAM policies, see the IAM User Guide.

Standard Amazon Pinpoint analytics metrics

You can use Amazon Pinpoint Analytics APIs to query analytics data for a subset of standard metrics that apply to Amazon Pinpoint projects, campaigns, and journeys. These metrics, also referred to as a *key performance indicators (KPIs)*, are measurable values that can help you monitor and assess the performance of projects, campaigns, and journeys.

Amazon Pinpoint provides programmatic access to analytics data for several types of standard metrics:

- **Application metrics** – These metrics provide insight into trends for all the campaigns and transactional messages that are associated with a project, also referred to as an *application*. For example, you can use an application metric to get a breakdown of the number of messages that were opened by recipients for each campaign that's associated with a project.
- **Campaign metrics** – These metrics provide insight into the performance of individual campaigns. For example, you can use a campaign metric to determine how many endpoints a campaign message was sent to or how many of those messages were delivered to endpoints.
- **Journey engagement metrics** – These metrics provide insight into the performance of individual journeys. For example, you can use a journey engagement metric to get a breakdown of the number of messages that were opened by participants in each activity of a journey.
- **Journey execution metrics** – These metrics provide insight into participation trends for individual journeys. For example, you can use a journey execution metric to determine how many participants started a journey.
- **Journey activity execution metrics** – These metrics provide insight into participation trends for individual activities in a journey. For example, you can use a journey activity execution metric to

determine how many participants started an activity and how many participants completed each path in an activity.

The topics in this section list and describe the individual metrics that you can query for each type of metric.

Topics

Application metrics for campaigns

The following table lists and describes standard application metrics that you can query to assess the performance of all the campaigns that are associated with an Amazon Pinpoint project. To query data for these metrics, use the Application metrics resource of the Amazon Pinpoint API. The **kpi-name** column in the table indicates the value to use for the `kpi-name` parameter in a query.

Metric	Kpi-name	Description
Delivery rate	`successful-delivery-rate`	For all the campaigns that are associated with a project, the percentage of messages that were delivered to recipients. This metric is calculated as the number of messages that were sent by all the campaigns for a project and delivered to recipients, divided by the number of messages that were sent by all of those campaigns.
Delivery rate, grouped by date	`successful-delivery-rate-grouped-by-date`	For all the campaigns that are associated with a project, the percentage of messages that were delivered to recipients, for each day in the specified date range. This metric is calculated as the number of messages that were sent by all the campaigns for a project and delivered to recipients, divided by the number of messages that were sent by all of those campaigns, for each day in the specified date range.

Metric	Kpi-name	Description
		The query results for this metric are grouped by calendar day, in extended ISO 8601 format.
Email open rate	`email-open-rate`	For all the campaigns that are associated with a project, the percentage of email messages that were opened by recipients. This metric is calculated as the number of email messages that were sent by all the campaigns for a project and opened by recipients, divided by the number of email messages that were sent by all of those campaigns and delivered to recipients.
Email open rate, grouped by campaign	`email-open-rate-grouped-by-campaign`	For each campaign that's associated with a project, the percentage of email messages that were opened by recipients. This metric is calculated as the number of email messages that were sent by a campaign and opened by recipients, divided by the number of email messages that were sent by the campaign and delivered to recipients. The query results for this metric are grouped by campaign ID (`CampaignId`), which is a string that uniquely identifies a campaign.
Endpoint deliveries	`unique-deliveries`	For all the campaigns that are associated with a project, the number of unique endpoints that messages were delivered to.
Endpoint deliveries, grouped by campaign	`unique-deliveries-grouped-by-campaign`	For each campaign that's associated with a project, the number of unique endpoints that messages were delivered to. The query results for this metric are grouped by campaign ID (`CampaignId`), which is a string that uniquely identifies a campaign.

Metric	Kpi-name	Description
Endpoint deliveries, grouped by date	`unique-deliveries-grouped-by-date`	For all the campaigns that are associated with a project, the number of unique endpoints that messages were delivered to, for each day in the specified date range. The query results for this metric are grouped by calendar day, in extended ISO 8601 format.
Messages delivered, grouped by campaign	`successful-deliveries-grouped-by-campaign`	For each campaign that's associated with a project, the number of messages that were delivered to recipients. This metric is calculated as the number of messages that were sent by a campaign, minus the number of messages that were sent by the campaign and couldn't be delivered to recipients due to a hard bounce. The query results for this metric are grouped by campaign ID (`CampaignId`), which is a string that uniquely identifies a campaign.
Push open rate	`push-open-rate`	For all the campaigns that are associated with a project, the percentage of push notifications that were opened by recipients. This metric is calculated as the number of push notifications that were sent by all the campaigns for a project and opened by recipients, divided by the number of push notifications that were sent by all of those campaigns and delivered to recipients.

Metric	Kpi-name	Description
Push open rate, grouped by campaign	`push-open-rate-grouped-by-campaign`	For each campaign that's associated with a project, the percentage of push notifications that were opened by recipients. This metric is calculated as the number of push notifications that were sent by a campaign and opened by recipients, divided by the number of push notifications that were sent by the campaign and delivered to recipients. The query results for this metric are grouped by campaign ID (`CampaignId`), which is a string that uniquely identifies a campaign.

Application metrics for transactional email messages

The following table lists and describes standard application metrics that you can query to monitor trends for all the transactional email messages that are associated with an Amazon Pinpoint project. To query data for these metrics, use the Application metrics resource of the Amazon Pinpoint API. The **kpi-name** column in the table indicates the value to use for the `kpi-name` parameter in a query.

Note that these metrics don't provide data about email messages that were sent by campaigns. They provide data about transactional email messages only. To query data for messages that were sent by one or more campaigns, use a campaign metric (p. 250) or an application metric for campaigns (p. 238).

Metric	Kpi-name	Description
Clicks	`txn-emails-clicked`	The number of times that recipients clicked links in the messages. If a single recipient clicked multiple links in a message, or clicked the same link more than once, each click is included in the count.
Clicks, grouped by date	`txn-emails-clicked-grouped-by-date`	The number of times that recipients clicked links in the messages, for each day in the specified date range. If a single recipient clicked multiple links in a message, or clicked the same link more than once, each click is included in the count. The query results for this metric are grouped by calendar day, in extended ISO 8601 format.

Metric	Kpi-name	Description
Complaint rate	`txn-emails-complaint-rate`	The percentage of messages that were reported by recipients as unsolicited or unwanted email. This metric is calculated as the number of messages that were reported by recipients as unsolicited or unwanted email, divided by the number of messages that were sent.
Complaint rate, grouped by date	`txn-emails-complaint-rate-grouped-by-date`	The percentage of messages that were reported by recipients as unsolicited or unwanted email, for each day in the specified date range. This metric is calculated as the number of messages that were reported by recipients as unsolicited or unwanted email, divided by the number of messages that were sent, for each day in the specified date range. The query results for this metric are grouped by calendar day, in extended ISO 8601 format.
Complaints	`txn-emails-with-complaints`	The number of messages that were reported by recipients as unsolicited or unwanted email.
Complaints, grouped by date	`txn-emails-with-complaints-grouped-by-date`	The number of messages that were reported by recipients as unsolicited or unwanted email, for each day in the specified date range. The query results for this metric are grouped by calendar day, in extended ISO 8601 format.

Metric	Kpi-name	Description
Deliveries	`txn-emails-delivered`	The number of messages that were delivered to recipients. This metric is calculated as the number of messages that were sent, minus the number of messages that couldn't be delivered due to a soft or hard bounce or because they were rejected. A message is rejected if Amazon Pinpoint determines that the message contains malware. Amazon Pinpoint doesn't attempt to send rejected messages.
Deliveries, grouped by date	`txn-emails-delivered-grouped-by-date`	The number of messages that were delivered to recipients, for each day in the specified date range. This metric is calculated as the number of messages that were sent, minus the number of messages that couldn't be delivered due to a soft or hard bounce or because they were rejected, for each day in the specified date range. A message is rejected if Amazon Pinpoint determines that the message contains malware. Amazon Pinpoint doesn't attempt to send rejected messages. The query results for this metric are grouped by calendar day, in extended ISO 8601 format.
Delivery rate	`txn-emails-delivery-rate`	The percentage of messages that were delivered to recipients. This metric is calculated as the number of messages that were sent and delivered to recipients, divided by the number of messages that were sent.

Metric	Kpi-name	Description
Delivery rate, grouped by date	`txn-emails-delivery-rate-grouped-by-date`	The percentage of messages that were delivered to recipients, for each day in the specified date range. This metric is calculated as the number of messages that were sent and delivered to recipients, divided by the number of messages that were sent, for each day in the specified date range. The query results for this metric are grouped by calendar day, in extended ISO 8601 format.
Hard bounces	`txn-emails-hard-bounced`	The number of messages that couldn't be delivered to recipients due to a hard bounce. A hard bounce occurs if a persistent issue prevents a message from being delivered—for example, if a recipient's email address doesn't exist.
Hard bounces, grouped by date	`txn-emails-hard-bounced-grouped-by-date`	The number of messages that couldn't be delivered to recipients due to a hard bounce, for each day in the specified date range. A hard bounce occurs if a persistent issue prevents a message from being delivered—for example, if a recipient's email address doesn't exist. The query results for this metric are grouped by calendar day, in extended ISO 8601 format.
Opens	`txn-emails-opened`	The number of messages that were opened by recipients.
Opens, grouped by date	`txn-emails-opened-grouped-by-date`	The number of messages that were opened by recipients, for each day in the specified date range. The query results for this metric are grouped by calendar day, in extended ISO 8601 format.
Sends	`txn-emails-sent`	The number of messages that were sent.

Metric	Kpi-name	Description
Sends, grouped by date	`txn-emails-sent-grouped-by-date`	The number of messages that were sent, for each day in the specified date range. The query results for this metric are grouped by calendar day, in extended ISO 8601 format.
Soft bounces	`txn-emails-soft-bounced`	The number of messages that couldn't be delivered to recipients due to a soft bounce. A soft bounce occurs if a temporary issue prevents a message from being delivered—for example, if a recipient's inbox is full or the receiving server is temporarily unavailable.
Soft bounces, grouped by date	`txn-emails-soft-bounced-grouped-by-date`	The number of messages that couldn't be delivered to recipients due to a soft bounce, for each day in the specified date range. A soft bounce occurs if a temporary issue prevents a message from being delivered—for example, if a recipient's inbox is full or the receiving server is temporarily unavailable. The query results for this metric are grouped by calendar day, in extended ISO 8601 format.
Unique user click events	`txn-emails-unique-clicks`	The number of unique recipients (endpoints) who clicked links in messages. Unlike the **Clicks** metric, this metric reports the number of unique recipients who clicked links, not the number of click events that occurred. For example, if a single recipient clicked multiple links in the same message, or clicked the same link more than once, this metric reports only one click event for that recipient.

Metric	Kpi-name	Description
Unique user click events, grouped by date	`txn-emails-unique-clicks-grouped-by-date`	The number of unique recipients (endpoints) who clicked links in messages, for each day in the specified date range. Unlike the **Clicks, grouped by date** metric, this metric reports the number of unique recipients who clicked links, not the number of click events that occurred. For example, if a single recipient clicked multiple links in the same message, or clicked the same link more than once, this metric reports only one click event for that recipient. The query results for this metric are grouped by calendar day, in extended ISO 8601 format.
Unique user open events	`txn-emails-unique-opens`	The number of unique recipients (endpoints) who opened messages. Unlike the **Opens** metric, this metric reports the number of unique recipients who opened messages, not the number of open events that occurred. For example, if a single recipient opens the same message multiple times, this metric reports only one open event for that recipient.
Unique user open events, grouped by date	`txn-emails-unique-opens-grouped-by-date`	The number of unique recipients (endpoints) who opened messages, for each day in the specified date range. Unlike the **Opens, grouped by date** metric, this metric reports the number of unique recipients who opened messages, not the number of open events that occurred. For example, if a single recipient opens the same message multiple times, this metric reports only one open event for that recipient. The query results for this metric are grouped by calendar day, in extended ISO 8601 format.

Application metrics for transactional SMS messages

The following table lists and describes standard application metrics that you can query to monitor trends for all the transactional SMS messages that are associated with an Amazon Pinpoint project. To query data for these metrics, use the Application metrics resource of the Amazon Pinpoint API. The **kpi-name** column in the table indicates the value to use for the `kpi-name` parameter in a query.

Note that these metrics don't provide data about SMS messages that were sent by campaigns. They provide data about transactional SMS messages only. To query data for messages that were sent by one or more campaigns, use a campaign metric (p. 250) or an application metric for campaigns (p. 238).

Metric	Kpi-name	Description
Average price per message, grouped by country	`txn-sms-average-price-grouped-by-country`	The average cost, in US Dollars, of sending each message, for each country or region that messages were sent to. This metric is calculated as the total cost of all the messages that were sent to recipients in each country or region, divided by the number of messages that were sent to recipients in each of those countries and regions. The query results for this metric are grouped by country or region, in ISO 3166-1 alpha-2 format.
Average price per message part, grouped by country	`txn-sms-average-price-by-parts-grouped-by-country`	The average cost, in US Dollars, of sending each message part, for each country or region that messages were sent to. A message part is a portion of an SMS message. This metric is calculated as the total cost of all the message parts that were sent to recipients in each country or region, divided by the number of message parts that were sent to recipients in each of those countries and regions. The query results for this metric are grouped by country or region, in ISO 3166-1 alpha-2 format.
Deliveries	`txn-sms-delivered`	The number of messages that were delivered to recipients.
Deliveries, grouped by country	`txn-sms-delivered-grouped-by-country`	The number of messages that were delivered to recipients,

Metric	Kpi-name	Description
		for each country or region that messages were sent to. The query results for this metric are grouped by country or region, in ISO 3166-1 alpha-2 format.
Deliveries, grouped by date	`txn-sms-delivered-grouped-by-date`	The number of messages that were delivered to recipients, for each day in the specified date range. The query results for this metric are grouped by calendar day, in extended ISO 8601 format.
Delivery errors	`txn-sms-error-distribution`	The number of times that an error occurred while attempting to deliver the messages, for each type of error that occurred. The query results for this metric are grouped by error code, for each type of error that occurred.
Delivery rate	`txn-sms-delivery-rate`	The percentage of messages that were delivered to recipients. This metric is calculated as the number of messages that were sent and delivered to recipients, divided by the number of messages that were sent.
Delivery rate, grouped by date	`txn-sms-delivery-rate-grouped-by-date`	The percentage of messages that were delivered to recipients, for each day in the specified date range. This metric is calculated as the number of messages that were sent and delivered to recipients, divided by the number of messages that were sent, for each day in the specified date range. The query results for this metric are grouped by calendar day, in extended ISO 8601 format.

Metric	Kpi-name	Description
Message parts delivered	`txn-sms-delivered-by-parts`	The number of message parts that were delivered. A *message part* is a portion of an SMS message. If an SMS message contains more characters than the SMS protocol allows, Amazon Pinpoint splits the message into as many message parts as necessary to send the message to a recipient.
Message parts delivered, grouped by country	`txn-sms-delivered-by-parts-grouped-by-country`	The number of message parts that were delivered, for each country or region that messages were sent to. A *message part* is a portion of an SMS message. The query results for this metric are grouped by country or region, in ISO 3166-1 alpha-2 format.
Message parts sent	`txn-sms-sent-by-parts`	The number of message parts that were sent. A *message part* is a portion of an SMS message. If an SMS message contains more characters than the SMS protocol allows, Amazon Pinpoint splits the message into as many message parts as necessary to send the message to a recipient.
Message parts sent, grouped by country	`txn-sms-sent-by-parts-grouped-by-country`	The number of message parts that were sent, for each country or region that messages were sent to. A *message part* is a portion of an SMS message. The query results for this metric are grouped by country or region, in ISO 3166-1 alpha-2 format.
Messages sent	`txn-sms-sent`	The number of messages that were sent.
Messages sent, grouped by country	`txn-sms-sent-grouped-by-country`	The number of messages that were sent, for each country or region that messages were sent to. The query results for this metric are grouped by country or region, in ISO 3166-1 alpha-2 format.

Metric	Kpi-name	Description
Messages sent, grouped by date	`txn-sms-sent-grouped-by-date`	The number of messages that were sent, for each day in the specified date range. The query results for this metric are grouped by calendar day, in extended ISO 8601 format.
Total price, grouped by country	`txn-sms-total-price-grouped-by-country`	The total cost, in US Dollars, of sending the messages, for each country or region that messages were sent to. The query results for this metric are grouped by country or region, in ISO 3166-1 alpha-2 format.
Total SMS spend	`sms-spend`	For all campaigns, the total amount of money, in milicents, spent on sending SMS.

Campaign metrics

The following table lists and describes standard campaign metrics that you can query to assess the performance of an individual campaign. To query data for these metrics, use the Campaign metrics resource of the Amazon Pinpoint API. The **kpi-name** column in the table indicates the value to use for the `kpi-name` parameter in your query.

Metric	Kpi-name	Description
Bounce rate	`hard-bounce-rate`	For all campaign runs, the percentage of email messages that couldn't be delivered to recipients. This metric measures only hard bounces—that is, messages in which the recipient's email address had a permanent issue that prevented the message from being delivered. This metric is calculated as the number of bounced email messages that were sent by all the campaign runs, divided by the number of email messages that were sent by all of those campaign runs.
Bounce rate, grouped by campaign run	`hard-bounce-rate-grouped-by-campaign-activity`	For each campaign run, the percentage of email messages that couldn't be delivered to recipients. This metric

Metric	Kpi-name	Description
		measures only hard bounces —that is, messages in which the recipient's email address had a permanent issue that prevented the message from being delivered. This metric is calculated as the number of bounced email messages that were sent by a campaign run, divided by the number of email messages that were sent by the campaign run. The query results for this metric are grouped by campaign activity ID (`CampaignActivityId`), which is a string that uniquely identifies a campaign run.
Delivery rate	`successful-delivery-rate`	For all campaign runs, the percentage of messages that were delivered to recipients. This metric is calculated as the number of messages that were sent by all the campaign runs and delivered to recipients, divided by the number of messages that were sent by all of those campaign runs.
Delivery rate, grouped by campaign run	`successful-delivery-rate-grouped-by-campaign-activity`	For each campaign run, the percentage of messages that were delivered to recipients. This metric is calculated as the number of messages that were sent by a campaign run and delivered to recipients, divided by the number of messages that were sent by the campaign run. The query results for this metric are grouped by campaign activity ID (`CampaignActivityId`), which is a string that uniquely identifies a campaign run.

Metric	Kpi-name	Description
Delivery rate, grouped by date	`successful-delivery-rate-grouped-by-date`	For all campaign runs, the percentage of messages that were delivered to recipients during each day in the specified date range. This metric is calculated as the number of messages that were sent by all the campaign runs and delivered to recipients, divided by the number of messages that were sent by all of those campaign runs, for each day in the specified date range. The query results for this metric are grouped by calendar day, in extended ISO 8601 format.
Email open rate	`email-open-rate`	For all campaign runs, the percentage of email messages that were opened by recipients. This metric is calculated as the number of email messages that were sent by all the campaign runs and opened by recipients, divided by the number of email messages that were sent by all of those campaign runs and delivered to recipients.
Email open rate, grouped by campaign run	`email-open-rate-grouped-by-campaign-activity`	For each campaign run, the percentage of email messages that were opened by recipients. This metric is calculated as the number of email messages that were sent by a campaign run and opened by recipients, divided by the number of email messages that were sent by the campaign run and delivered to recipients. The query results for this metric are grouped by campaign activity ID (`CampaignActivityId`), which is a string that uniquely identifies a campaign run.

Metric	Kpi-name	Description
Emails opened, grouped by campaign run	`direct-email-opens-grouped-by-campaign-activity`	For each campaign run, the number of email messages that were opened by recipients. The query results for this metric are grouped by campaign activity ID (`CampaignActivityId`), which is a string that uniquely identifies a campaign run.
Endpoint deliveries	`unique-deliveries`	For all campaign runs, the number of unique endpoints that messages were delivered to.
Endpoint deliveries, grouped by campaign run	`unique-deliveries-grouped-by-campaign-activity`	For each campaign run, the number of unique endpoints that messages were delivered to. The query results for this metric are grouped by campaign activity ID (`CampaignActivityId`), which is a string that uniquely identifies a campaign run.
Endpoint deliveries, grouped by date	`unique-deliveries-grouped-by-date`	For all campaign runs, the number of unique endpoints that messages were delivered to, for each day in the specified date range. The query results for this metric are grouped by calendar day, in extended ISO 8601 format.
Links clicked, grouped by campaign run	`clicks-grouped-by-campaign-activity`	For each campaign run, the number of times that recipients clicked links in the email message. If a single recipient clicked multiple links in the message, or clicked the same link more than once, each click is included in the count. The query results for this metric are grouped by campaign activity ID (`CampaignActivityId`), which is a string that uniquely identifies a campaign run.

Metric	Kpi-name	Description
Messages delivered, grouped by campaign run	`successful-deliveries-grouped-by-campaign-activity`	For each campaign run, the number of messages that were delivered to recipients. This metric is calculated as the number of messages that were sent by a campaign run, minus the number of messages that couldn't be delivered to recipients of the run due to a hard bounce. The query results for this metric are grouped by campaign activity ID (`CampaignActivityId`), which is a string that uniquely identifies a campaign run.
Messages sent, grouped by campaign run	`attempted-deliveries-grouped-by-campaign-activity`	For each campaign run, the number of messages that were sent. The query results for this metric are grouped by campaign activity ID (`CampaignActivityId`), which is a string that uniquely identifies a campaign run.
Push open rate	`push-open-rate`	For all campaign runs, the percentage of push notifications that were opened by recipients. This metric is calculated as the number of push notifications that were sent by all the campaign runs and opened by recipients, divided by the number of push notifications that were sent by all of those campaign runs and delivered to recipients.

Metric	Kpi-name	Description
Push open rate, grouped by campaign run	`push-open-rate-grouped-by-campaign-activity`	For each campaign run, the percentage of push notifications that were opened by recipients. This metric is calculated as the number of push notifications that were sent by a campaign run and opened by recipients, divided by the number of push notifications that were sent by the campaign run and delivered to recipients. The query results for this metric are grouped by campaign activity ID (`CampaignActivityId`), which is a string that uniquely identifies a campaign run.
Total push opened, grouped by campaign run	`direct-push-opens-grouped-by-campaign-activity`	For each campaign run, the number of push notifications that were opened by recipients. The query results for this metric are grouped by campaign activity ID (`CampaignActivityId`), which is a string that uniquely identifies a campaign run.

Journey engagement metrics

The following table lists and describes standard journey engagement metrics that you can query to monitor trends for all the email messages that were sent by an Amazon Pinpoint journey. To query data for these metrics, use the Journey engagement metrics resource of the Amazon Pinpoint API. The **kpi-name** column in the table indicates the value to use for the `kpi-name` parameter in a query.

Metric	Kpi-name	Description
Clicks	`journey-emails-clicked`	The number of times that participants clicked links in the messages. If a single participant clicked multiple links in a message, or clicked the same link more than once, each click is included in the count.
Clicks, grouped by activity	`emails-clicked-grouped-by-journey-activity`	For each activity in the journey, the number of times that participants clicked links in the messages. If a single participant clicked multiple links in a message, or clicked the same

Metric	Kpi-name	Description
		link more than once, each click is included in the count. The query results for this metric are grouped by activity ID (`JourneyActivityId`), which is a string that uniquely identifies an activity.
Complaints	`journey-emails-complained`	The number of messages that were reported by participants as unsolicited or unwanted email.
Complaints, grouped by activity	`emails-complained-grouped-by-journey-activity`	For each activity in the journey, the number of messages that were reported by participants as unsolicited or unwanted email. The query results for this metric are grouped by activity ID (`JourneyActivityId`), which is a string that uniquely identifies an activity.
Deliveries	`journey-emails-delivered`	The number of messages that were delivered to participants. This metric is calculated as the number of messages that were sent, minus the number of messages that couldn't be delivered due to a soft or hard bounce, or because they were rejected.
Deliveries, grouped by activity	`emails-delivered-grouped-by-journey-activity`	For each activity in the journey, the number of messages that were delivered to participants. This metric is calculated as the number of messages that were sent, minus the number of messages that couldn't be delivered due to a soft or hard bounce, or because they were rejected, for each activity in the journey. The query results for this metric are grouped by activity ID (`JourneyActivityId`), which is a string that uniquely identifies an activity.

Metric	Kpi-name	Description
Hard bounces	`journey-emails-hardbounced`	The number of messages that couldn't be delivered to participants due to a hard bounce. A hard bounce occurs if a persistent issue prevents a message from being delivered—for example, if a participant's email address doesn't exist.
Hard bounces, grouped by activity	`emails-hardbounced-grouped-by-journey-activity`	For each activity in the journey, the number of messages that couldn't be delivered to participants due to a hard bounce. A hard bounce occurs if a persistent issue prevents a message from being delivered—for example, if a participant's email address doesn't exist. The query results for this metric are grouped by activity ID (`JourneyActivityId`), which is a string that uniquely identifies an activity.
Opens	`journey-emails-opened`	The number of messages that were opened by participants.
Opens, grouped by activity	`emails-opened-grouped-by-journey-activity`	For each activity in the journey, the number of messages that were opened by participants. The query results for this metric are grouped by activity ID (`JourneyActivityId`), which is a string that uniquely identifies an activity.
Rejections	`journey-emails-rejected`	The number of messages that weren't sent to participants because they were rejected. A message is rejected if Amazon Pinpoint determines that the message contains malware. Amazon Pinpoint doesn't attempt to send rejected messages.

Metric	Kpi-name	Description
Rejections, grouped by activity	`emails-rejected-grouped-by-journey-activity`	For each activity in the journey, the number of messages that weren't sent to participants because they were rejected. A message is rejected if Amazon Pinpoint determines that the message contains malware. Amazon Pinpoint doesn't attempt to send rejected messages. The query results for this metric are grouped by activity ID (`JourneyActivityId`), which is a string that uniquely identifies an activity.
Sends	`journey-emails-sent`	The number of messages that were sent.
Sends, grouped by activity	`emails-sent-grouped-by-journey-activity`	For each activity in the journey, the number of messages that were sent. The query results for this metric are grouped by activity ID (`JourneyActivityId`), which is a string that uniquely identifies an activity.
Soft bounces	`journey-emails-softbounced`	The number of messages that couldn't be delivered to participants due to a soft bounce. A soft bounce occurs if a temporary issue prevents a message from being delivered—for example, if a participant's inbox is full or the receiving server is temporarily unavailable.

Metric	Kpi-name	Description
Soft bounces, grouped by activity	`emails-softbounced-grouped-by-journey-activity`	For each activity in the journey, the number of messages that couldn't be delivered to participants due to a soft bounce. A soft bounce occurs if a temporary issue prevents a message from being delivered—for example, if a participant's inbox is full or the receiving server is temporarily unavailable. The query results for this metric are grouped by activity ID (`JourneyActivityId`), which is a string that uniquely identifies an activity.
Unsubscribes	`journey-emails-unsubscribed`	The number of times that participants clicked unsubscribe links in the messages. If a single participant clicked the same unsubscribe link more than once, each click is included in the count.
Unsubscribes, grouped by activity	`emails-unsubscribed-grouped-by-journey-activity`	For each activity in the journey, the number of times that participants clicked unsubscribe links in the messages. If a single participant clicked the same unsubscribe link more than once, each click is included in the count. The query results for this metric are grouped by activity ID (`JourneyActivityId`), which is a string that uniquely identifies an activity.

Journey execution metrics

The following table lists and describes standard execution metrics that you can query to assess the status of participants in an Amazon Pinpoint journey. To query data for these metrics, use the Journey execution metrics resource of the Amazon Pinpoint API. The **Field** column in the table identifies the name of the field that appears in the query results for each metric.

Metric	Field	Description
Active participants	`ENDPOINT_ACTIVE`	The number of participants who are actively proceeding through the activities in the journey.

Metric	Field	Description
		This metric is calculated as the number of participants who started the journey, minus the number of participants who left the journey and the number of participants who were removed from the journey.
Participant cancellations	`CANCELLED`	The number of participants who didn't complete the journey because the journey was cancelled.
Participant departures	`ENDPOINT_LEFT`	The number of participants who left the journey.
Participant entries	`ENDPOINT_ENTERED`	The number of participants who started the journey.
Participant exceptions, reentry limits	`REENTRY_CAP_EXCEEDED`	The number of participants who didn't complete the journey because they would have exceeded the maximum number of times that a single participant can re-enter the journey.
Participant exceptions, rejections	`ACTIVE_ENDPOINT_REJECTED`	The number of participants who can't start the journey because they are already active participants in the journey. A participant is rejected if they start a journey and you subsequently update their endpoint definition in a way that affects their inclusion in a segment (based on segment criteria) or the journey (based on activity conditions).

Journey activity execution metrics

The following table lists and describes standard execution metrics that you can query to assess the status of participants in each type of individual activity for an Amazon Pinpoint journey. To query data for these metrics, use the Journey activity execution metrics resource of the Amazon Pinpoint API. The **Metrics** column in the table lists the fields that appear in the query results for each type of activity. It also provides a brief description of each field.

Activity type	Metrics
Yes/No split (`CONDITIONAL_SPLIT`)	The metrics are: • `Branch_FALSE` – The number of participants who didn't meet the activity's conditions and proceeded to the activity on the "No" path.

Activity type	Metrics
	• `Branch_TRUE` – The number of participants who met the activity's conditions and proceeded to the activity on the "Yes" path. Additional metrics are available for the activity on each path. For information about those metrics, see the row in this table for that type of activity.
Holdout (`HOLDOUT`)	The metrics are: • `HOLDOUT` – The number of participants who were removed from the journey as part of the holdout percentage for the activity. • `PASSED` – The number of participants who proceeded to the next activity in the journey.
Email (`MESSAGE`)	The metrics are: • `DAILY_CAP_EXCEEDED` – The number of messages that weren't sent because they would have exceeded the maximum number of messages that a single participant can receive during a 24-hour period. • `FAILURE_PERMANENT` – The number of messages that weren't sent due to a permanent issue. • `QUIET_TIME` – The number of messages that weren't sent because they would have been delivered during quiet time for the participant's time zone. • `SERVICE_FAILURE` – The number of messages that weren't sent due to an issue with Amazon Pinpoint. • `SUCCESS` – The number of messages that were successfully delivered to participants. • `THROTTLED` – The number of messages that weren't sent because sending them would exceed the sending quotas for your Amazon Pinpoint account. • `TRANSIENT_FAILURE` – The number of messages that weren't sent due to a temporary issue. • `UNKNOWN` – The number of messages that weren't sent due to an unknown issue.

Activity type	Metrics
Multivariate split (`MULTI_CONDITIONAL_SPLIT`)	For each path of the activity, the number of participants who proceeded to the activity on the path. The query results for this metric are grouped by path, `Branch_#` where `#` is the numeric identifier for a path—for example, `Branch_1` for the first path of the activity. Additional metrics are available for the activity on each path. For information about those metrics, see the row in this table for that type of activity.
Random split (`RANDOM_SPLIT`)	For each path of the activity, the number of participants who proceeded to the activity on the path. The query results for this metric are grouped by path, `Branch_#` where `#` is the numeric identifier for a path—for example, `Branch_1` for the first path of the activity. Additional metrics are available for the activity on each path. For information about those metrics, see the row in this table for that type of activity.
Wait (`WAIT`)	The metrics are: • `WAIT_FINISHED` – The number of participants who finished waiting for the specified amount of time. • `WAIT_SKIPPED` – The number of participants who didn't wait for the specified amount of time, typically because they started the activity or journey after the scheduled end time for the activity. • `WAIT_STARTED` – The number of participants who started waiting, and haven't skipped or finished waiting for the specified amount of time.

Querying Amazon Pinpoint analytics data for campaigns

In addition to using the analytics pages on the Amazon Pinpoint console, you can use Amazon Pinpoint Analytics APIs to query analytics data for a subset of standard metrics that provide insight into delivery and engagement trends for campaigns.

Each of these metrics is a measurable value, also referred to as a *key performance indicator (KPI)*, that can help you monitor and assess the performance of one or more campaigns. For example, you can use a metric to find out how many endpoints a campaign message was sent to, or how many of those messages were delivered to the intended endpoints.

Amazon Pinpoint automatically collects and aggregates this data for all of your campaigns. It stores the data for 90 days. If you integrated a mobile app with Amazon Pinpoint by using an AWS Mobile SDK, Amazon Pinpoint extends this support to include additional metrics, such as the percentage of push notifications that were opened by recipients. For information about integrating a mobile app, see Integrating Amazon Pinpoint with your application (p. 108).

If you use Amazon Pinpoint Analytics APIs to query data, you can choose various options that define the scope, data, grouping, and filters for your query. You do this by using parameters that specify the project, campaign, and metric that you want to query, in addition to any date-based filters that you want to apply.

This topic explains and provides examples of how to choose these options and query the data for one or more campaigns.

Prerequisites

Before you query analytics data for one or more campaigns, it helps to gather the following information, which you use to define your query:

- **Project ID** – The unique identifier for the project that's associated with the campaign or campaigns. In the Amazon Pinpoint API, this value is stored in the `application-id` property. On the Amazon Pinpoint console, this value is displayed as the **Project ID** on the **All projects** page.
- **Campaign ID** – The unique identifier for the campaign, if you want to query the data for only one campaign. In the Amazon Pinpoint API, this value is stored in the `campaign-id` property. This value is not displayed on the console.
- **Date range** – Optionally, the first and last date and time of the date range to query data for. Date ranges are inclusive and must be limited to 31 or fewer calendar days. In addition, they must start fewer than 90 days from the current day. If you don't specify a date range, Amazon Pinpoint automatically queries the data for the preceding 31 calendar days.
- **Metric type** – The type of metric to query. There are two types, *application metrics* and *campaign metrics*. An *application metric* provides data for all the campaigns that are associated with a project, also referred to as an *application*. A *campaign metric* provides data for only one campaign.
- **Metric** – The name of the metric to query—more specifically, the `kpi-name` value for the metric. For a complete list of supported metrics and the `kpi-name` value for each one, see Standard metrics (p. 237).

It also helps to determine whether you want to group the data by a relevant field. If you do, you can simplify your analysis and reporting by choosing a metric that's designed to group data for you automatically. For example, Amazon Pinpoint provides several standard metrics that report the percentage of messages that were delivered to recipients of a campaign. One of these metrics automatically groups the data by date (`successful-delivery-rate-grouped-by-date`). Another metric automatically groups the data by campaign run (`successful-delivery-rate-grouped-by-campaign-activity`). A third metric simply returns a single value—the percentage of messages that were delivered to recipients by all campaign runs (`successful-delivery-rate`).

If you can't find a standard metric that groups data the way that you want, you can develop a series of queries that return the data that you want. You can then manually break down or combine the query results into custom groups that you design.

Finally, it's important to verify that you're authorized to access the data that you want to query. For more information, see IAM policies for querying Amazon Pinpoint analytics data (p. 235).

Querying data for one campaign

To query the data for one campaign, you use the Campaign Metrics API and specify values for the following required parameters:

- **application-id** – The project ID, which is the unique identifier for the project that's associated with the campaign. In Amazon Pinpoint, the terms *project* and *application* have the same meaning.
- **campaign-id** – The unique identifier for the campaign.
- **kpi-name** – The name of the metric to query. This value describes the associated metric and consists of two or more terms, which are comprised of lowercase alphanumeric characters, separated by a hyphen. For a complete list of supported metrics and the `kpi-name` value for each one, see Standard metrics (p. 237).

You can also apply a filter that queries the data for a specific date range. If you don't specify a date range, Amazon Pinpoint returns the data for the preceding 31 calendar days. To filter the data by different dates, use the supported date range parameters to specify the first and last date and time of the date range. The values should be in extended ISO 8601 format and use Coordinated Universal Time (UTC)—for example, `2019-07-19T20:00:00Z` for 8:00 PM UTC July 19, 2019. Date ranges are inclusive and must be limited to 31 or fewer calendar days. In addition, the first date and time must be fewer than 90 days from the current day.

The following examples show how to query analytics data for a campaign by using the Amazon Pinpoint REST API, the AWS CLI, and the AWS SDK for Java. You can use any supported AWS SDK to query analytics data for a campaign. The AWS CLI examples are formatted for Microsoft Windows. For Unix, Linux, and macOS, replace the caret (^) line-continuation character with a backslash (\).

REST API

To query analytics data for a campaign by using the Amazon Pinpoint REST API, send an HTTP(S) GET request to the Campaign Metrics URI. In the URI, specify the appropriate values for the required path parameters:

```
https://endpoint/v1/apps/application-id/campaigns/campaign-id/kpis/daterange/kpi-name
```

Where:

- *`endpoint`* is the Amazon Pinpoint endpoint for the AWS Region that hosts the project associated with the campaign.
- *`application-id`* is the unique identifier for the project that's associated with the campaign.
- *`campaign-id`* is the unique identifier for the campaign.
- *`kpi-name`* is the `kpi-name` value for the metric to query.

All the parameters should be URL encoded.

To apply a filter that queries the data for a specific date range, append the `start-time` and `end-time` query parameters and values to the URI. By using these parameters, you can specify the first and last date and time, in extended ISO 8601 format, of an inclusive date range to retrieve the data for. Use an ampersand (&) to separate the parameters.

For example, the following request retrieves the number of unique endpoints that messages were delivered to, by all runs of a campaign, from July 19, 2019 through July 26, 2019:

```
https://pinpoint.us-east-1.amazonaws.com/v1/apps/1234567890123456789012345example/
campaigns/80b8efd84042ff8d9c96ce2f8example/kpis/daterange/unique-deliveries?start-
time=2019-07-19T00:00:00Z&end-time=2019-07-26T23:59:59Z
```

Where:

- `pinpoint.us-east-1.amazonaws.com` is the Amazon Pinpoint endpoint for the AWS Region that hosts the project.

- `12345678901234567890123455example` is the unique identifier for the project that's associated with the campaign.
- `80b8efd84042ff8d9c96ce2f8example` is the unique identifier for the campaign.
- `unique-deliveries` is the `kpi-name` value for the *endpoint deliveries* campaign metric, which is the metric that reports the number of unique endpoints that messages were delivered to, by all runs of a campaign.
- `2019-07-19T00:00:00Z` is the first date and time to retrieve data for, as part of an inclusive date range.
- `2019-07-26T23:59:59Z` is the last date and time to retrieve data for, as part of an inclusive date range.

AWS CLI

To query analytics data for a campaign by using the AWS CLI, use the **get-campaign-date-range-kpi** command and specify the appropriate values for the required parameters:

```
C:\> aws pinpoint get-campaign-date-range-kpi ^
    --application-id application-id ^
    --campaign-id campaign-id ^
    --kpi-name kpi-name
```

Where:

- *`application-id`* is the unique identifier for the project that's associated with the campaign.
- *`campaign-id`* is the unique identifier for the campaign.
- *`kpi-name`* is the `kpi-name` value for the metric to query.

To apply a filter that queries the data for a specific date range, add the `start-time` and `end-time` parameters and values to your query. By using these parameters, you can specify the first and last date and time, in extended ISO 8601 format, of an inclusive date range to retrieve the data for. For example, the following request retrieves the number of unique endpoints that messages were delivered to, by all runs of a campaign, from July 19, 2019 through July 26, 2019:

```
C:\> aws pinpoint get-campaign-date-range-kpi ^
    --application-id 12345678901234567890123455example ^
    --campaign-id 80b8efd84042ff8d9c96ce2f8example ^
    --kpi-name unique-deliveries ^
    --start-time 2019-07-19T00:00:00Z ^
    --end-time 2019-07-26T23:59:59Z
```

Where:

- `12345678901234567890123455example` is the unique identifier for the project that's associated with the campaign.
- `80b8efd84042ff8d9c96ce2f8example` is the unique identifier for the campaign.
- `unique-deliveries` is the `kpi-name` value for the *endpoint deliveries* campaign metric, which is the metric that reports the number of unique endpoints that messages were delivered to, by all runs of a campaign.
- `2019-07-19T00:00:00Z` is the first date and time to retrieve data for, as part of an inclusive date range.
- `2019-07-26T23:59:59Z` is the last date and time to retrieve data for, as part of an inclusive date range.

SDK for Java

To query analytics data for a campaign by using the AWS SDK for Java, use the **GetCampaignDateRangeKpiRequest** method of the Campaign Metrics API. Specify the appropriate values for the required parameters:

```
GetCampaignDateRangeKpiRequest request = new GetCampaignDateRangeKpiRequest()
        .withApplicationId("applicationId")
        .withCampaignId("campaignId")
        .withKpiName("kpiName")
```

Where:

- *`applicationId`* is the unique identifier for the project that's associated with the campaign.
- *`campaignId`* is the unique identifier for the campaign.
- *`kpiName`* is the `kpi-name` value for the metric to query.

To apply a filter that queries the data for a specific date range, include the `startTime` and `endTime` parameters and values in your query. By using these parameters, you can specify the first and last date and time, in extended ISO 8601 format, of an inclusive date range to retrieve the data for. For example, the following request retrieves the number of unique endpoints that messages were delivered to, by all runs of a campaign, from July 19, 2019 through July 26, 2019:

```
GetCampaignDateRangeKpiRequest request = new GetCampaignDateRangeKpiRequest()
        .withApplicationId("1234567890123456789012345example")
        .withCampaignId("80b8efd84042ff8d9c96ce2f8example")
        .withKpiName("unique-deliveries")
        .withStartTime(Date.from(Instant.parse("2019-07-19T00:00:00Z")))
        .withEndTime(Date.from(Instant.parse("2019-07-26T23:59:59Z")));
```

Where:

- `1234567890123456789012345example` is the unique identifier for the project that's associated with the campaign.
- `80b8efd84042ff8d9c96ce2f8example` is the unique identifier for the campaign.
- `unique-deliveries` is the `kpi-name` value for the *endpoint deliveries* campaign metric, which is the metric that reports the number of unique endpoints that messages were delivered to, by all runs of a campaign.
- `2019-07-19T00:00:00Z` is the first date and time to retrieve data for, as part of an inclusive date range.
- `2019-07-26T23:59:59Z` is the last date and time to retrieve data for, as part of an inclusive date range.

After you send your query, Amazon Pinpoint returns the query results in a JSON response. The structure of the results varies depending on the metric that you queried. Some metrics return only one value. For example, the *endpoint deliveries* (`unique-deliveries`) campaign metric, which is used in the preceding examples, returns one value—the number of unique endpoints that messages were delivered to, by all runs of a campaign. In this case, the JSON response is the following:

```
{
    "CampaignDateRangeKpiResponse":{
        "ApplicationId":"1234567890123456789012345example",
        "CampaignId":"80b8efd84042ff8d9c96ce2f8example",
        "EndTime":"2019-07-26T23:59:59Z",
        "KpiName":"unique-deliveries",
        "KpiResult":{
            "Rows":[
                {
                    "Values":[
                        {
                            "Key":"UniqueDeliveries",
                            "Type":"Double",
                            "Value":"123.0"
                        }
                    ]
                }
            ]
        },
        "StartTime":"2019-07-19T00:00:00Z"
    }
}
```

Other metrics return multiple values, and group the values by a relevant field. If a metric returns multiple values, the JSON response includes a field that indicates which field was used to group the data.

To learn more about the structure of query results, see Using query results (p. 279).

Querying data for multiple campaigns

There are two ways to query the data for multiple campaigns. The best way depends on whether you want to query the data for campaigns that are all associated with the same project. If you do, it also depends on whether you want to query the data for all or only or subset of those campaigns.

To query the data for campaigns that are associated with different projects or for only a subset of the campaigns that are associated with the same project, the best approach is to create and run a series of individual queries, one for each campaign that you want to query the data for. The preceding section explains how to query the data for only one campaign.

To query the data for all the campaigns that are associated with the same project, you can use the Application Metrics API. Specify values for the following required parameters:

- **application-id** – The project ID, which is the unique identifier for the project. In Amazon Pinpoint, the terms *project* and *application* have the same meaning.
- **kpi-name** – The name of the metric to query. This value describes the associated metric and consists of two or more terms, which are comprised of lowercase alphanumeric characters, separated by a hyphen. For a complete list of supported metrics and the `kpi-name` value for each one, see Standard metrics (p. 237).

You can also filter the data by date range. If you don't specify a date range, Amazon Pinpoint returns the data for the preceding 31 calendar days. To filter the data by different dates, use the supported date range parameters to specify the first and last date and time of the date range. The values should be in extended ISO 8601 format and use Coordinated Universal Time (UTC)—for example, `2019-07-19T20:00:00Z` for 8:00 PM UTC July 19, 2019. Date ranges are inclusive and must be limited to 31 or fewer calendar days. In addition, the first date and time must be fewer than 90 days from the current day.

The following examples show how to query analytics data for a campaign by using the Amazon Pinpoint REST API, the AWS CLI, and the AWS SDK for Java. You can use any supported AWS SDK to query analytics data for a campaign. The AWS CLI examples are formatted for Microsoft Windows. For Unix, Linux, and macOS, replace the caret (^) line-continuation character with a backslash (\).

REST API

To query analytics data for multiple campaigns by using the Amazon Pinpoint REST API, send an HTTP(S) GET request to the Application Metrics URI. In the URI, specify the appropriate values for the required path parameters:

```
https://endpoint/v1/apps/application-id/kpis/daterange/kpi-name
```

Where:

- *endpoint* is the Amazon Pinpoint endpoint for the AWS Region that hosts the project associated with the campaigns.
- *application-id* is the unique identifier for the project that's associated with the campaigns.
- *kpi-name* is the `kpi-name` value for the metric to query.

All the parameters should be URL encoded.

To apply a filter that retrieves the data for a specific date range, append the `start-time` and `end-time` query parameters and values to the URI. By using these parameters, you can specify the first and last date and time, in extended ISO 8601 format, of an inclusive date range to retrieve the data for. Use an ampersand (&) to separate the parameters.

For example, the following request retrieves the number of unique endpoints that messages were delivered to, by each of a project's campaigns, from July 19, 2019 through July 26, 2019:

```
https://pinpoint.us-east-1.amazonaws.com/v1/apps/12345678901234567890123 45example/kpis/
daterange/unique-deliveries-grouped-by-campaign?start-time=2019-07-19T00:00:00Z&end-
time=2019-07-26T23:59:59Z
```

Where:

- `pinpoint.us-east-1.amazonaws.com` is the Amazon Pinpoint endpoint for the AWS Region that hosts the project.
- `1234567890123456789012345example` is the unique identifier for the project that's associated with the campaigns.
- `unique-deliveries-grouped-by-campaign` is the `kpi-name` value for the *endpoint deliveries, grouped by campaign* application metric, which is the metric that returns the number of unique endpoints that messages were delivered to, by each campaign.
- `2019-07-19T00:00:00Z` is the first date and time to retrieve data for, as part of an inclusive date range.
- `2019-07-26T23:59:59Z` is the last date and time to retrieve data for, as part of an inclusive date range.

AWS CLI

To query analytics data for multiple campaigns by using the AWS CLI, use the **get-application-date-range-kpi** command and specify the appropriate values for the required parameters:

```
C:\> aws pinpoint get-application-date-range-kpi ^
    --application-id application-id ^
    --kpi-name kpi-name
```

Where:

- *application-id* is the unique identifier for the project that's associated with the campaigns.

- *`kpi-name`* is the `kpi-name` value for the metric to query.

To apply a filter that retrieves the data for a specific date range, include the `start-time` and `end-time` parameters and values in your query. By using these parameters, you can specify the first and last date and time, in extended ISO 8601 format, of an inclusive date range to retrieve the data for. For example, the following request retrieves the number of unique endpoints that messages were delivered to, by each of a project's campaigns, from July 19, 2019 through July 26, 2019:

```
C:\> aws pinpoint get-application-date-range-kpi ^
    --application-id 1234567890123456789012345example ^
    --kpi-name unique-deliveries-grouped-by-campaign ^
    --start-time 2019-07-19T00:00:00Z ^
    --end-time 2019-07-26T23:59:59Z
```

Where:

- `1234567890123456789012345example` is the unique identifier for the project that's associated with the campaign.
- `unique-deliveries-grouped-by-campaign` is the `kpi-name` value for the *endpoint deliveries, grouped by campaign* application metric, which is the metric that returns the number of unique endpoints that messages were delivered to, by each campaign.
- `2019-07-19T00:00:00Z` is the first date and time to retrieve data for, as part of an inclusive date range.
- `2019-07-26T23:59:59Z` is the last date and time to retrieve data for, as part of an inclusive date range.

SDK for Java

To query analytics data for multiple campaigns by using the AWS SDK for Java, use the **GetApplicationDateRangeKpiRequest** method of the Application Metrics API. Specify the appropriate values for the required parameters:

```
GetApplicationDateRangeKpiRequest request = new GetApplicationDateRangeKpiRequest()
        .withApplicationId("applicationId")
        .withKpiName("kpiName")
```

Where:

- *`applicationId`* is the unique identifier for the project that's associated with the campaigns.
- *`kpiName`* is the `kpi-name` value for the metric to query.

To apply a filter that retrieves the data for a specific date range, include the `startTime` and `endTime` parameters and values in your query. By using these parameters, you can specify the first and last date and time, in extended ISO 8601 format, of an inclusive date range to retrieve the data for. For example, the following request retrieves the number of unique endpoints that messages were delivered to, by each of a project's campaigns, from July 19, 2019 through July 26, 2019:

```
GetApplicationDateRangeKpiRequest request = new GetApplicationDateRangeKpiRequest()
        .withApplicationId("1234567890123456789012345example")
        .withKpiName("unique-deliveries-grouped-by-campaign")
        .withStartTime(Date.from(Instant.parse("2019-07-19T00:00:00Z")))
        .withEndTime(Date.from(Instant.parse("2019-07-26T23:59:59Z")));
```

Where:

- `12345678901234567890123455example` is the unique identifier for the project that’s associated with the campaigns.
- `unique-deliveries-grouped-by-campaign` is the `kpi-name` value for the *endpoint deliveries, grouped by campaign* application metric, which is the metric that returns the number of unique endpoints that messages were delivered to, by each campaign.
- `2019-07-19T00:00:00Z` is the first date and time to retrieve data for, as part of an inclusive date range.
- `2019-07-26T23:59:59Z` is the last date and time to retrieve data for, as part of an inclusive date range.

After you send your query, Amazon Pinpoint returns the query results in a JSON response. The structure of the results varies depending on the metric that you queried. Some metrics return only one value. Other metrics return multiple values, and those values are grouped by a relevant field. If a metric returns multiple values, the JSON response includes a field that indicates which field was used to group the data.

For example, the *endpoint deliveries, grouped by campaign* (`unique-deliveries-grouped-by-campaign`) application metric, which is used in the preceding examples, returns multiple values—the number of unique endpoints that messages were delivered to, for each campaign that's associated with a project. In this case, the JSON response is the following:

```
{
    "ApplicationDateRangeKpiResponse":{
        "ApplicationId":"12345678901234567890123455example",
        "EndTime":"2019-07-26T23:59:59Z",
        "KpiName":"unique-deliveries-grouped-by-campaign",
        "KpiResult":{
            "Rows":[
                {
                    "GroupedBys":[
                        {
                            "Key":"CampaignId",
                            "Type":"String",
                            "Value":"80b8efd84042ff8d9c96ce2f8example"
                        }
                    ],
                    "Values":[
                        {
                            "Key":"UniqueDeliveries",
                            "Type":"Double",
                            "Value":"123.0"
                        }
                    ]
                },
                {
                    "GroupedBys":[
                        {
                            "Key":"CampaignId",
                            "Type":"String",
                            "Value":"810c7aab86d42fb2b56c8c966example"
                        }
                    ],
                    "Values":[
                        {
                            "Key":"UniqueDeliveries",
                            "Type":"Double",
                            "Value":"456.0"
                        }
                    ]
                },
                {
                    "GroupedBys":[
                        {
                            "Key":"CampaignId",
                            "Type":"String",
                            "Value":"42d8c7eb0990a57ba1d5476a3example"
                        }
                    ],
                    "Values":[
                        {
                            "Key":"UniqueDeliveries",
                            "Type":"Double",
                            "Value":"789.0"
                        }
                    ]
                }
            ]
        },
        "StartTime":"2019-07-19T00:00:00Z"
    }
}
```

In this case, the `GroupedBys` field indicates that the values are grouped by campaign ID (`CampaignId`).

To learn more about the structure of query results, see Using query results (p. 279).

Querying Amazon Pinpoint analytics data for transactional messages

In addition to using the analytics pages on the Amazon Pinpoint console, you can use Amazon Pinpoint Analytics APIs to query analytics data for a subset of standard metrics that provide insight into delivery and engagement trends for the transactional messages that were sent for a project.

Each of these metrics is a measurable value, also referred to as a *key performance indicator (KPI)*, that can help you monitor and assess the performance of transactional messages. For example, you can use a metric to find out how many transactional email or SMS messages you sent, or how many of those messages were delivered to recipients. Amazon Pinpoint automatically collects and aggregates this data for all the transactional email and SMS messages that you send for a project. It stores the data for 90 days.

If you use Amazon Pinpoint Analytics APIs to query data, you can choose various options that define the scope, data, grouping, and filters for your query. You do this by using parameters that specify the project and metric that you want to query, in addition to any date-based filters that you want to apply.

This topic explains and provides examples of how to choose these options and query transactional messaging data for a project.

Prerequisites

Before you query analytics data for transactional messages, it helps to gather the following information, which you use to define your query:

- **Project ID** – The unique identifier for the project that the messages were sent from. In the Amazon Pinpoint API, this value is stored in the `application-id` property. On the Amazon Pinpoint console, this value is displayed as the **Project ID** on the **All projects** page.
- **Date range** – Optionally, the first and last date and time of the date range to query data for. Date ranges are inclusive and must be limited to 31 or fewer calendar days. In addition, they must start fewer than 90 days from the current day. If you don't specify a date range, Amazon Pinpoint automatically queries the data for the preceding 31 calendar days.

- **Metric** – The name of the metric to query—more specifically, the `kpi-name` value for the metric. For a complete list of supported metrics and the `kpi-name` value for each one, see Standard metrics (p. 237).

It also helps to determine whether you want to group the data by a relevant field. If you do, you can simplify your analysis and reporting by choosing a metric that's designed to group data for you automatically. For example, Amazon Pinpoint provides several standard metrics that report the number of transactional SMS messages that were delivered to recipients. One of these metrics automatically groups the data by date (`txn-sms-delivered-grouped-by-date`). Another metric automatically groups the data by country or region (`txn-sms-delivered-grouped-by-country`). A third metric simply returns a single value—the number of messages that were delivered to recipients (`txn-sms-delivered`). If you can't find a standard metric that groups data the way that you want, you can develop a series of queries that return the data that you want. You can then manually break down or combine the query results into custom groups that you design.

Finally, it's important to verify that you're authorized to access the data that you want to query. For more information, see IAM policies for querying Amazon Pinpoint analytics data (p. 235).

Querying data for transactional email messages

To query the data for transactional email messages that were sent for a project, you use the Application Metrics API and specify values for the following required parameters:

- **application-id** – The project ID, which is the unique identifier for the project. In Amazon Pinpoint, the terms *project* and *application* have the same meaning.
- **kpi-name** – The name of the metric to query. This value describes the associated metric and consists of two or more terms, which are comprised of lowercase alphanumeric characters, separated by a hyphen. For a complete list of supported metrics and the `kpi-name` value for each one, see Standard metrics (p. 237).

You can also apply a filter that queries the data for a specific date range. If you don't specify a date range, Amazon Pinpoint returns the data for the preceding 31 calendar days. To filter the data by different dates, use the supported date range parameters to specify the first and last date and time of the date range. The values should be in extended ISO 8601 format and use Coordinated Universal Time (UTC)—for example, `2019-09-06T20:00:00Z` for 8:00 PM UTC September 6, 2019. Date ranges are inclusive and must be limited to 31 or fewer calendar days. In addition, the first date and time must be fewer than 90 days from the current day.

The following examples show how to query analytics data for transactional email messages by using the Amazon Pinpoint REST API, the AWS CLI, and the AWS SDK for Java. You can use any supported AWS SDK to query analytics data for transactional messages. The AWS CLI examples are formatted for Microsoft Windows. For Unix, Linux, and macOS, replace the caret (^) line-continuation character with a backslash (\).

REST API

To query analytics data for transactional email messages by using the Amazon Pinpoint REST API, send an HTTP(S) GET request to the Application Metrics URI. In the URI, specify the appropriate values for the required path parameters:

```
https://endpoint/v1/apps/application-id/kpis/daterange/kpi-name
```

Where:

- *`endpoint`* is the Amazon Pinpoint endpoint for the AWS Region that hosts the project.
- *`application-id`* is the unique identifier for the project.

- *kpi-name* is the `kpi-name` value for the metric to query.

All the parameters should be URL encoded.

To apply a filter that queries the data for a specific date range, append the `start-time` and `end-time` query parameters and values to the URI. By using these parameters, you can specify the first and last date and time, in extended ISO 8601 format, of an inclusive date range to retrieve the data for. Use an ampersand (&) to separate the parameters.

For example, the following request retrieves the number of transactional email messages that were sent for a project from September 6, 2019 through September 13, 2019:

```
https://pinpoint.us-east-1.amazonaws.com/v1/apps/1234567890123456789012345example/kpis/
daterange/txn-emails-sent?start-time=2019-09-06T00:00:00Z&end-time=2019-09-13T23:59:59Z
```

Where:

- `pinpoint.us-east-1.amazonaws.com` is the Amazon Pinpoint endpoint for the AWS Region that hosts the project.
- `1234567890123456789012345example` is the unique identifier for the project.
- `txn-emails-sent` is the `kpi-name` value for the *sends* application metric, which is the metric that reports the number of transactional email messages that were sent for a project.
- `2019-09-06T00:00:00Z` is the first date and time to retrieve data for, as part of an inclusive date range.
- `2019-09-13T23:59:59Z` is the last date and time to retrieve data for, as part of an inclusive date range.

AWS CLI

To query analytics data for transactional email messages by using the AWS CLI, use the **get-application-date-range-kpi** command, and specify the appropriate values for the required parameters:

```
C:\> aws pinpoint get-application-date-range-kpi ^
    --application-id application-id ^
    --kpi-name kpi-name
```

Where:

- *application-id* is the unique identifier for the project.
- *kpi-name* is the `kpi-name` value for the metric to query.

To apply a filter that queries the data for a specific date range, add the `start-time` and `end-time` parameters and values to your query. By using these parameters, you can specify the first and last date and time, in extended ISO 8601 format, of an inclusive date range to retrieve the data for. For example, the following request retrieves the number of transactional email messages that were sent for a project from September 6, 2019 through September 13, 2019:

```
C:\> aws pinpoint get-application-date-range-kpi ^
    --application-id 1234567890123456789012345example ^
    --kpi-name txn-emails-sent ^
    --start-time 2019-09-06T00:00:00Z ^
    --end-time 2019-09-13T23:59:59Z
```

Where:

- `1234567890123456789012345example` is the unique identifier for the project.
- `txn-emails-sent` is the `kpi-name` value for the *sends* application metric, which is the metric that reports the number of transactional email messages that were sent for a project.
- `2019-09-06T00:00:00Z` is the first date and time to retrieve data for, as part of an inclusive date range.
- `2019-09-13T23:59:59Z` is the last date and time to retrieve data for, as part of an inclusive date range.

SDK for Java

To query analytics data for transactional email messages by using the AWS SDK for Java, use the **GetApplicationDateRangeKpiRequest** method of the Application Metrics API. Specify the appropriate values for the required parameters:

```
GetApplicationDateRangeKpiRequest request = new GetApplicationDateRangeKpiRequest()
        .withApplicationId("applicationId")
        .withKpiName("kpiName")
```

Where:

- *`applicationId`* is the unique identifier for the project.
- *`kpiName`* is the `kpi-name` value for the metric to query.

To apply a filter that queries the data for a specific date range, include the `startTime` and `endTime` parameters and values in your query. By using these parameters, you can specify the first and last date and time, in extended ISO 8601 format, of an inclusive date range to retrieve the data for. For example, the following request retrieves the number of transactional email messages that were sent for a project from September 6, 2019 through September 13, 2019:

```
GetApplicationDateRangeKpiRequest request = new GetApplicationDateRangeKpiRequest()
        .withApplicationId("1234567890123456789012345example")
        .withKpiName("txn-emails-sent")
        .withStartTime(Date.from(Instant.parse("2019-09-06T00:00:00Z")))
        .withEndTime(Date.from(Instant.parse("2019-09-13T23:59:59Z")));
```

Where:

- `1234567890123456789012345example` is the unique identifier for the project.
- `txn-emails-sent` is the `kpi-name` value for the *sends* application metric, which is the metric that reports the number of transactional email messages that were sent for a project.
- `2019-09-06T00:00:00Z` is the first date and time to retrieve data for, as part of an inclusive date range.
- `2019-09-13T23:59:59Z` is the last date and time to retrieve data for, as part of an inclusive date range.

After you send your query, Amazon Pinpoint returns the query results in a JSON response. The structure of the results varies depending on the metric that you queried. Some metrics return only one value. For example, the *sends* (`txn-emails-sent`) application metric, which is used in the preceding examples, returns one value—the number of transactional email messages that were sent from a project. In this case, the JSON response is the following:

```
{
    "ApplicationDateRangeKpiResponse":{
        "ApplicationId":"1234567890123456789012345example",
        "EndTime":"2019-09-13T23:59:59Z",
        "KpiName":"txn-emails-sent",
        "KpiResult":{
            "Rows":[
                {
                    "Values":[
                        {
                            "Key":"TxnEmailsSent",
                            "Type":"Double",
                            "Value":"62.0"
                        }
                    ]
                }
            ]
        },
        "StartTime":"2019-09-06T00:00:00Z"
    }
}
```

Other metrics return multiple values and group the values by a relevant field. If a metric returns multiple values, the JSON response includes a field that indicates which field was used to group the data.

To learn more about the structure of query results, see Using query results (p. 279).

Querying data for transactional SMS messages

To query the data for transactional SMS messages that were sent for a project, you use the Application Metrics API and specify values for the following required parameters:

- **application-id** – The project ID, which is the unique identifier for the project. In Amazon Pinpoint, the terms *project* and *application* have the same meaning.
- **kpi-name** – The name of the metric to query. This value describes the associated metric and consists of two or more terms, which are comprised of lowercase alphanumeric characters, separated by a hyphen. For a complete list of supported metrics and the `kpi-name` value for each one, see Standard metrics (p. 237).

You can also apply a filter that queries the data for a specific date range. If you don't specify a date range, Amazon Pinpoint returns the data for the preceding 31 calendar days. To filter the data by different dates, use the supported date range parameters to specify the first date and time and the last date and time of the date range. The values should be in extended ISO 8601 format and use Coordinated Universal Time (UTC)—for example, `2019-09-06T20:00:00Z` for 8:00 PM UTC September 6, 2019. Date ranges are inclusive and must be limited to 31 or fewer calendar days. In addition, the first date and time must be fewer than 90 days from the current day.

The following examples show how to query analytics data for transactional SMS messages by using the Amazon Pinpoint REST API, the AWS CLI, and the AWS SDK for Java. You can use any supported AWS SDK to query analytics data for transactional messages. The AWS CLI examples are formatted for Microsoft Windows. For Unix, Linux, and macOS, replace the caret (^) line-continuation character with a backslash (\).

REST API

To query analytics data for transactional SMS messages by using the Amazon Pinpoint REST API, send an HTTP(S) GET request to the Application Metrics URI. In the URI, specify the appropriate values for the required path parameters:

```
https://endpoint/v1/apps/application-id/kpis/daterange/kpi-name
```

Where:

- *`endpoint`* is the Amazon Pinpoint endpoint for the AWS Region that hosts the project.
- *`application-id`* is the unique identifier for the project.
- *`kpi-name`* is the `kpi-name` value for the metric to query.

All the parameters should be URL encoded.

To apply a filter that retrieves the data for a specific date range, append the `start-time` and `end-time` query parameters and values to the URI. By using these parameters, you can specify the first and last date and time, in extended ISO 8601 format, of an inclusive date range to retrieve the data for. Use an ampersand (&) to separate the parameters.

For example, the following request retrieves the number of transactional SMS messages that were sent each day from September 6, 2019 through September 8, 2019:

```
https://pinpoint.us-east-1.amazonaws.com/v1/apps/1234567890123456789012345example/
kpis/daterange/txn-sms-sent-grouped-by-date?start-time=2019-09-06T00:00:00Z&end-
time=2019-09-08T23:59:59Z
```

Where:

- `pinpoint.us-east-1.amazonaws.com` is the Amazon Pinpoint endpoint for the AWS Region that hosts the project.
- `1234567890123456789012345example` is the unique identifier for the project.
- `txn-sms-sent-grouped-by-date` is the `kpi-name` value for the *sends, grouped by date* application metric, which is the metric that returns the number of transactional SMS messages that were sent during each day of the date range.
- `2019-09-06T00:00:00Z` is the first date and time to retrieve data for, as part of an inclusive date range.
- `2019-09-08T23:59:59Z` is the last date and time to retrieve data for, as part of an inclusive date range.

AWS CLI

To query analytics data for transactional SMS messages by using the AWS CLI, use the **get-application-date-range-kpi** command, and specify the appropriate values for the required parameters:

```
C:\> aws pinpoint get-application-date-range-kpi ^
    --application-id application-id ^
    --kpi-name kpi-name
```

Where:

- *`application-id`* is the unique identifier for the project.
- *`kpi-name`* is the `kpi-name` value for the metric to query.

To apply a filter that retrieves the data for a specific date range, include the `start-time` and `end-time` parameters and values in your query. By using these parameters, you can specify the first and last date and time, in extended ISO 8601 format, of an inclusive date range to retrieve the data for. For example, the following request retrieves the number of transactional SMS messages that were sent each day from September 6, 2019 through September 8, 2019:

```
C:\> aws pinpoint get-application-date-range-kpi ^
    --application-id 1234567890123456789012345example ^
    --kpi-name txn-sms-sent-grouped-by-date ^
    --start-time 2019-09-06T00:00:00Z ^
    --end-time 2019-09-08T23:59:59Z
```

Where:

- `1234567890123456789012345example` is the unique identifier for the project.
- `txn-sms-sent-grouped-by-date` is the `kpi-name` value for the *sends, grouped by date* application metric, which is the metric that returns the number of transactional SMS messages that were sent during each day of the date range.
- `2019-09-06T00:00:00Z` is the first date and time to retrieve data for, as part of an inclusive date range.
- `2019-09-08T23:59:59Z` is the last date and time to retrieve data for, as part of an inclusive date range.

SDK for Java

To query analytics data for transactional SMS messages by using the AWS SDK for Java, use the **GetApplicationDateRangeKpiRequest** method of the Application Metrics API, and specify the appropriate values for the required parameters:

```
GetApplicationDateRangeKpiRequest request = new GetApplicationDateRangeKpiRequest()
        .withApplicationId("applicationId")
        .withKpiName("kpiName")
```

Where:

- *`applicationId`* is the unique identifier for the project.
- *`kpiName`* is the `kpi-name` value for the metric to query.

To apply a filter that retrieves the data for a specific date range, include the `startTime` and `endTime` parameters and values in your query. By using these parameters, you can specify the first and last date and time, in extended ISO 8601 format, of an inclusive date range to retrieve the data for. For example, the following request retrieves the number of transactional SMS messages that were sent each day from September 6, 2019 through September 8, 2019:

```
GetApplicationDateRangeKpiRequest request = new GetApplicationDateRangeKpiRequest()
        .withApplicationId("1234567890123456789012345example")
        .withKpiName("txn-sms-sent-grouped-by-date")
        .withStartTime(Date.from(Instant.parse("2019-09-06T00:00:00Z")))
        .withEndTime(Date.from(Instant.parse("2019-09-08T23:59:59Z")));
```

Where:

- `1234567890123456789012345example` is the unique identifier for the project.
- `txn-sms-sent-grouped-by-date` is the `kpi-name` value for the *sends, grouped by date* application metric, which is the metric that returns the number of transactional SMS messages that were sent during each day of the date range.
- `2019-09-06T00:00:00Z` is the first date and time to retrieve data for, as part of an inclusive date range.
- `2019-09-08T23:59:59Z` is the last date and time to retrieve data for, as part of an inclusive date range.

After you send your query, Amazon Pinpoint returns the query results in a JSON response. The structure of the results varies depending on the metric that you queried. Some metrics return only one value. Other metrics return multiple values and group those values by a relevant field. If a metric returns multiple values, the JSON response includes a field that indicates which field was used to group the data.

For example, the *sends, grouped by date* (`txn-sms-sent-grouped-by-date`) application metric, which is used in the preceding examples, returns multiple values—the number of transactional SMS messages that were sent during each day of the specified date range. In this case, the JSON response is the following:

```
{
    "ApplicationDateRangeKpiResponse":{
        "ApplicationId":"1234567890123456789012345example",
        "EndTime":"2019-09-08T23:59:59Z",
        "KpiName":"txn-sms-sent-grouped-by-date",
        "KpiResult":{
            "Rows":[
                {
                    "GroupedBys":[
                        {
                            "Key":"Date",
                            "Type":"String",
                            "Value":"2019-09-06"
                        }
                    ],
                    "Values":[
                        {
                            "Key":"TxnSmsSent",
                            "Type":"Double",
                            "Value":"29.0"
                        }
                    ]
                },
                {
                    "GroupedBys":[
                        {
                            "Key":"Date",
                            "Type":"String",
                            "Value":"2019-09-07"
                        }
                    ],
                    "Values":[
                        {
                            "Key":"TxnSmsSent",
                            "Type":"Double",
                            "Value":"35.0"
                        }
                    ]
                },
                {
                    "GroupedBys":[
                        {
                            "Key":"Date",
                            "Type":"String",
                            "Value":"2019-09-08"
                        }
                    ],
                    "Values":[
                        {
                            "Key":"TxnSmsSent",
                            "Type":"Double",
                            "Value":"10.0"
                        }
                    ]
                }
            ]
        },
        "StartTime":"2019-09-06T00:00:00Z"
    }
}
```

In this case, the `GroupedBys` field indicates that the values are grouped by calendar day (`Date`). This means that:

- 29 messages were sent on September 6, 2019.
- 35 messages were sent on September 7, 2019.
- 10 messages were sent on September 8, 2019.

To learn more about the structure of query results, see Using query results (p. 279).

Using Amazon Pinpoint analytics query results

When you use Amazon Pinpoint Analytics APIs to query analytics data, Amazon Pinpoint returns the results in a JSON response. For application metrics, campaign metrics, and journey engagement metrics, the data in the response adheres to a standard JSON schema for reporting Amazon Pinpoint analytics data.

This means that you can use the programming language or tool of your choice to implement a custom solution that queries the data for one or more of these metrics, captures the results of each query, and then writes the results to a table, object, or other location. You can then work with the query results in that location by using another service or application.

For example, you can:

- Build a custom dashboard that queries a set of metrics on a regular basis and displays the results by using your preferred data visualization framework.
- Create a report that tracks engagement rates by querying the appropriate metrics and displaying the results in a chart or other type of report that you design.
- Parse and write analytics data to a particular storage format, and then port the results to a long-term storage solution.

Note that Amazon Pinpoint Analytics APIs aren't designed to create or store any persistent objects that you can subsequently read or use in an Amazon Pinpoint project or your Amazon Pinpoint account. Instead, the APIs are designed to help you retrieve analytics data and transfer that data to other services and applications for further analysis, storage, or reporting. They do this partly by using the same JSON response structure and schema for all the analytics data that you can query programmatically for application metrics, campaign metrics, and journey engagement metrics.

This topic explains the structure, objects, and fields in a JSON response to a query for an application metric, campaign metric, or journey engagement metric. For information about the fields in a JSON response to a query for a journey execution metric or journey activity execution metric, see Standard Amazon Pinpoint analytics metrics (p. 237).

JSON structure

To help you parse and use query results, Amazon Pinpoint Analytics APIs use the same JSON response structure for all Amazon Pinpoint analytics data that you can query programmatically for application metrics, campaign metrics, and journey engagement metrics. Each JSON response specifies the values that defined the query, such as the project ID (`ApplicationId`). The response also includes one (and only one) `KpiResult` object. The `KpiResult` object contains the overall result set for a query.

Each `KpiResult` object contains a `Rows` object. This is an array of objects that contain query results and relevant metadata about the values in those results. The structure and content of a `Rows` object has the following general characteristics:

- Each row of query results is a separate JSON object, named `Values`, in the `Rows` object. For example, if a query returns three values, the `Rows` object contains three `Values` objects. Each `Values` object contains an individual result for the query.
- Each column of query results is a property of the `Values` object that it applies to. The name of the column is stored in the `Key` field of the `Values` object.
- For grouped query results, each `Values` object has an associated `GroupedBys` object. The `GroupedBys` object indicates which field was used to group the results. It also provides the grouping value for the associated `Values` object.
- If the query results for a metric is null, the `Rows` object is empty.

Beyond these general characteristics, the structure and contents of the `Rows` object varies depending on the metric. This is because Amazon Pinpoint supports two kinds of metrics, *single-value metrics* and *multiple-value metrics*.

A *single-value metric* provides only one cumulative value. An example is the percentage of messages that were delivered to recipients by all runs of a campaign. A *multiple-value metric* provides more than one value and groups those values by a relevant field. An example is the percentage of messages that were delivered to recipients for each run of a campaign, grouped by campaign run.

You can quickly determine whether a metric is a single-value metric or multiple-value metric by referring to the name of the metric. If the name doesn't contain `grouped-by`, it's a single-value metric. If it does, it's a multiple-value metric. For a complete list of metrics that you can query programmatically, see Standard Amazon Pinpoint analytics metrics (p. 237).

Single-value metrics

For a single-value metric, the `Rows` object contains a `Values` object that:

- Specifies the friendly name of the metric that was queried.
- Provides the value for the metric that was queried.
- Identifies the data type of the value that was returned.

For example, the following JSON response contains the query results for a single-value metric. This metric reports the number of unique endpoints that messages were delivered to by all the campaigns that are associated with a project, from August 1, 2019 through August 31, 2019:

```
{
    "ApplicationDateRangeKpiResponse":{
        "ApplicationId":"1234567890123456789012345example",
        "EndTime":"2019-08-31T23:59:59Z",
        "KpiName":"unique-deliveries",
        "KpiResult":{
            "Rows":[
                {
                    "Values":[
                        {
                            "Key":"UniqueDeliveries",
                            "Type":"Double",
                            "Value":"1368.0"
                        }
                    ]
                }
            ]
        },
        "StartTime":"2019-08-01T00:00:00Z"
    }
}
```

In this example, the response indicates that all the project's campaigns delivered messages to 1,368 unique endpoints from August 1, 2019 through August 31, 2019, where:

- `Key` is the friendly name of the metric whose value is specified in the `Value` field (`UniqueDeliveries`).
- `Type` is the data type of the value specified in the `Value` field (`Double`).
- `Value` is the actual value for the metric that was queried, including any filters that were applied (`1368.0`).

If the query results for a single-value metric is null (not greater than or equal to zero), the `Rows` object is empty. Amazon Pinpoint returns a null value for a metric if there isn't any data to return for the metric. For example:

```
{
    "ApplicationDateRangeKpiResponse":{
        "ApplicationId":"23456789012345678901234567example",
        "EndTime":"2019-08-31T23:59:59Z",
        "KpiName":"unique-deliveries",
        "KpiResult":{
            "Rows":[

            ]
        },
        "StartTime":"2019-08-01T00:00:00Z"
    }
}
```

Multiple-value metrics

The structure and contents of the `Rows` object for a multiple-value metric are mostly the same as a single-value metric. The `Rows` object for a multiple-value metric also contains a `Values` object. The `Values` object specifies the friendly name of the metric that was queried, provides the value for that metric, and identifies the data type of that value.

However, the `Rows` object for a multiple-value metric also contains one or more `GroupedBy` objects. There is one `GroupedBy` object for each `Values` object in the query results. The `GroupedBy` object indicates which field was used to group the data in the results and the data type of that field. It also indicates the grouping value for that field (for the associated `Values` object).

For example, the following JSON response contains the query results for a multiple-value metric that reports the number of unique endpoints that messages were delivered to, for each campaign that's associated with a project, from August 1, 2019 through August 31, 2019:

```
{
    "ApplicationDateRangeKpiResponse":{
        "ApplicationId":"12345678901234567890123456example",
        "EndTime":"2019-08-31T23:59:59Z",
        "KpiName":"unique-deliveries-grouped-by-campaign",
        "KpiResult":{
            "Rows":[
                {
                    "GroupedBys":[
                        {
                            "Key":"CampaignId",
                            "Type":"String",
                            "Value":"80b8efd84042ff8d9c96ce2f8example"
                        }
                    ],
                    "Values":[
                        {
                            "Key":"UniqueDeliveries",
                            "Type":"Double",
                            "Value":"123.0"
                        }
                    ]
                },
                {
                    "GroupedBys":[
                        {
                            "Key":"CampaignId",
                            "Type":"String",
                            "Value":"810c7aab86d42fb2b56c8c966example"
                        }
                    ],
                    "Values":[
                        {
                            "Key":"UniqueDeliveries",
                            "Type":"Double",
                            "Value":"456.0"
                        }
                    ]
                },
                {
                    "GroupedBys":[
                        {
                            "Key":"CampaignId",
                            "Type":"String",
                            "Value":"42d8c7eb0990a57ba1d5476a3example"
                        }
                    ],
                    "Values":[
                        {
                            "Key":"UniqueDeliveries",
                            "Type":"Double",
                            "Value":"789.0"
                        }
                    ]
                }
            ]
        },
        "StartTime":"2019-08-01T00:00:00Z"
    }
}
```

In this example, the response indicates that three of the project's campaigns delivered messages to unique endpoints from August 1, 2019 through August 31, 2019. For each of those campaigns, the breakdown of delivery counts is:

- Campaign `80b8efd84042ff8d9c96ce2f8example` delivered messages to 123 unique endpoints.
- Campaign `810c7aab86d42fb2b56c8c966example` delivered messages to 456 unique endpoints.
- Campaign `42d8c7eb0990a57ba1d5476a3example` delivered messages to 789 unique endpoints.

Where the general structure of the objects and fields is:

- `GroupedBys.Key` – The name of the property or field that stores the grouping value specified in the `GroupedBys.Value` field (`CampaignId`).
- `GroupedBys.Type` – The data type of the value specified in the `GroupedBys.Value` field (`String`).
- `GroupedBys.Value` – The actual value for the field that was used to group the data, as specified in the `GroupedBys.Key` field (campaign ID).
- `Values.Key` – The friendly name of the metric whose value is specified in the `Values.Value` field (`UniqueDeliveries`).
- `Values.Type` – The data type of the value specified in the `Values.Value` field (`Double`).
- `Values.Value` – The actual value for the metric that was queried, including any filters that were applied.

If the query results for a multiple-value metric is null (not greater than or equal to zero) for a specific project, campaign, or other resource, Amazon Pinpoint doesn't return any objects or fields for the resource. If the query results for a multiple-value metric is null for all resources, Amazon Pinpoint returns an empty `Rows` object.

JSON objects and fields

In addition to specifying the values that defined a query, such as the project ID (`ApplicationId`), each JSON response to a query for an application metric, campaign metric, or journey engagement metric includes a `KpiResult` object. This object contains the overall result set for a query, which you can parse to send analytics data to another service or application. Each `KpiResult` object contains some or all of the following standard objects and fields, depending on the metric.

Object or field	Description
`Rows`	An array of objects that contains the result set for a query.
`Rows.GroupedBys`	For a multiple-value metric, an array of fields that defines the field and values that were used to group data in query results.
`Rows.GroupedBys.Key`	For a multiple-value metric, the name of the property or field that stores the value specified in the `GroupedBys.Value` field.
`Rows.GroupedBys.Type`	For a multiple-value metric, the data type of the value specified in the `GroupedBys.Value` field.
`Rows.GroupedBys.Value`	For a multiple-value metric, the actual value for the field that was used to group data in query results. This value correlates to an associated `Values` object.
`Rows.Values`	An array of fields that contains query results.
`Rows.Values.Key`	The friendly name of the metric that was queried. The metric's value is specified in the `Values.Value` field.
`Rows.Values.Type`	The data type of the value specified in the `Values.Value` field.
`Rows.Values.Value`	The actual value for the metric that was queried, including any filters that were applied.

For information about the fields in a JSON response to a query for a journey execution metric or journey activity execution metric, see Standard Amazon Pinpoint analytics metrics (p. 237).

Logging Amazon Pinpoint API calls with AWS CloudTrail

Amazon Pinpoint is integrated with AWS CloudTrail, which is a service that provides a record of actions taken by a user, role, or AWS service in Amazon Pinpoint. CloudTrail captures API calls for Amazon Pinpoint as events. The calls that are captured include calls from the Amazon Pinpoint console and code calls to Amazon Pinpoint API operations.

If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon Simple Storage Service (Amazon S3) bucket, including events for Amazon Pinpoint. If you don't configure a trail, you can still view the most recent events by using **Event history** on the CloudTrail console. Using the information collected by CloudTrail, you can determine the request that was made to Amazon Pinpoint, the IP address that the request was made from, 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:

- 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 CloudTrail userIdentity element.

You can create a trail and store your log files in your Amazon S3 bucket for as long as you want. Also, you can 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. 286)
- Amazon Pinpoint SMS and Voice API (p. 290)

Amazon Pinpoint API actions that can be logged by CloudTrail

The Amazon Pinpoint API supports logging the following actions as events in CloudTrail log files:

- CreateApp
- CreateCampaign
- CreateEmailTemplate
- CreateExportJob
- CreateImportJob
- CreateJourney
- CreatePushTemplate
- CreateRecommenderConfiguration
- CreateSegment
- CreateSmsTemplate
- CreateVoiceTemplate
- DeleteAdmChannel
- DeleteApnsChannel
- DeleteApnsSandboxChannel
- DeleteApnsVoipChannel
- DeleteApnsVoipSandboxChannel
- DeleteApp
- DeleteBaiduChannel
- DeleteCampaign
- DeleteEmailChannel
- DeleteEmailTemplate
- DeleteEndpoint
- DeleteEventStream
- DeleteGcmChannel
- DeleteJourney

- DeletePushTemplate
- DeleteRecommenderConfiguration
- DeleteSegment
- DeleteSmsChannel
- DeleteSmsTemplate
- DeleteUserEndpoints
- DeleteVoiceChannel
- DeleteVoiceTemplate
- GetAdmChannel
- GetApnsChannel
- GetApnsSandboxChannel
- GetApnsVoipChannel
- GetApnsVoipSandboxChannel
- GetApp
- GetApplicationDateRangeKpi
- GetApplicationSettings
- GetApps
- GetBaiduChannel
- GetCampaign
- GetCampaignActivities
- GetCampaignDateRangeKpi
- GetCampaignVersion
- GetCampaignVersions
- GetCampaigns
- GetChannels
- GetEmailChannel
- GetEmailTemplate
- GetEndpoint
- GetEventStream
- GetExportJob
- GetExportJobs
- GetGcmChannel
- GetImportJob
- GetImportJobs
- GetJourney
- GetJourneyDateRangeKpi
- GetJourneyExecutionActivityMetrics
- GetJourneyExecutionMetrics
- GetPushTemplate
- GetRecommenderConfiguration
- GetRecommenderConfigurations
- GetSegment
- GetSegmentExportJobs

- GetSegmentImportJobs
- GetSegmentVersion
- GetSegmentVersions
- GetSegments
- GetSmsChannel
- GetSmsTemplate
- GetUserEndpoints
- GetVoiceChannel
- GetVoiceTemplate
- ListJourneys
- ListTagsForResource
- ListTemplates
- ListTemplateVersions
- PhoneNumberValidate
- PutEventStream
- RemoveAttributes
- TagResource
- UntagResource
- UpdateAdmChannel
- UpdateApnsChannel
- UpdateApnsSandboxChannel
- UpdateApnsVoipChannel
- UpdateApnsVoipSandboxChannel
- UpdateApplicationSettings
- UpdateBaiduChannel
- UpdateCampaign
- UpdateEmailChannel
- UpdateEmailTemplate
- UpdateEndpoint
- UpdateEndpointsBatch
- UpdateGcmChannel
- UpdateJourney
- UpdateJourneyState
- UpdatePushTemplate
- UpdateRecommenderConfiguration
- UpdateSegment
- UpdateSmsChannel
- UpdateSmsTemplate
- UpdateTemplateActiveVersion
- UpdateVoiceChannel
- UpdateVoiceTemplate

The following Amazon Pinpoint API actions **aren't** logged in CloudTrail:

- PutEvents
- SendMessages
- SendUsersMessages

Amazon Pinpoint email API 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

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
- CreateConfigurationSetEventDestination
- DeleteConfigurationSet
- DeleteConfigurationSetEventDestination
- GetConfigurationSetEventDestinations
- UpdateConfigurationSetEventDestination

The following Amazon Pinpoint SMS and Voice API action **isn't** logged in CloudTrail:

- SendVoiceMessage

Examples: 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. It 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"
    }
  ]
}
```

Tagging Amazon Pinpoint resources

A *tag* is a label that you optionally define and associate with AWS resources, including certain types of Amazon Pinpoint resources. Tags can help you categorize and manage resources in different ways, such as by purpose, owner, environment, or other criteria. For example, you can use tags to apply policies or automation, or to identify resources that are subject to certain compliance requirements. You can add tags to the following types of Amazon Pinpoint resources:

- Campaigns
- Message templates
- Projects (applications)
- Segments

A resource can have as many as 50 tags.

Managing tags

Each tag consists of a required *tag key* and an optional *tag value*, both of which you define. A *tag key* is a general label that acts as a category for more specific tag values. A *tag value* acts as a descriptor for a tag key. For example, if you have two versions of an Amazon Pinpoint project (one for internal testing and another for external use), you might assign a `Stack` tag key to both projects. The value of the `Stack` tag key might be `Test` for one version of the project and `Production` for the other version.

A tag key can contain as many as 128 characters. A tag value can contain as many as 256 characters. The characters can be Unicode letters, numbers, white space, or one of the following symbols: _ . : / = + -. The following additional restrictions apply to tags:

- Tag keys and values are case sensitive.
- For each associated resource, each tag key must be unique and it can have only one value.
- The `aws:` prefix is reserved for use by AWS; you can't use it in any tag keys or values that you define. In addition, you can't edit or remove tag keys or values that use this prefix. Tags that use this prefix don't count against the quota of 50 tags per resource.
- You can't update or delete a resource based only on its tags. You must also specify the Amazon Resource Name (ARN) or resource ID, depending on the operation that you use.
- You can associate tags with public or shared resources. However, the tags are available only for your AWS account, not any other accounts that share the resource. In addition, the tags are available only for resources that are located in the specified AWS Region for your AWS account.

To add, display, update, and remove tag keys and values from Amazon Pinpoint resources, you can use the AWS Command Line Interface (AWS CLI), the Amazon Pinpoint API, the AWS Resource Groups Tagging API, or an AWS SDK. To manage tag keys and values across all the AWS resources that are located in a specific AWS Region for your AWS account (including Amazon Pinpoint resources), use the AWS Resource Groups Tagging API.

Using tags in IAM policies

After you start implementing tags, you can apply tag-based, resource-level permissions to AWS Identity and Access Management (IAM) policies and API operations. This includes operations that support adding tags to resources when resources are created. By using tags in this way, you can implement granular control of which groups and users in your AWS account have permission to create and tag resources, and which groups and users have permission to create, update, and remove tags more generally.

For example, you can create a policy that allows a user to have full access to all the Amazon Pinpoint resources where their name is a value in the `Owner` tag for the resource:

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "ModifyResourceIfOwner",
         "Effect": "Allow",
         "Action": "mobiletargeting:*",
         "Resource": "*",
         "Condition": {
            "StringEqualsIgnoreCase": {
               "aws:ResourceTag/Owner": "${aws:username}"
            }
         }
      }
   ]
}
```

If you define tag-based, resource-level permissions, the permissions take effect immediately. This means that your resources are more secure as soon as they're created, and you can quickly start enforcing the use of tags for new resources. You can also use resource-level permissions to control which tag keys and values can be associated with new and existing resources. For more information, see Controlling Access Using Tags in the *AWS IAM User Guide*.

Adding tags to resources

The following examples show how to add a tag to an Amazon Pinpoint resource by using the AWS CLI and the Amazon Pinpoint REST API. The AWS CLI examples are formatted for Microsoft Windows. For Unix, Linux, and macOS, replace the caret (^) line-continuation character with a backslash (\). You can also use any supported AWS SDK to add a tag to a resource.

To add a tag to multiple Amazon Pinpoint resources in a single operation, use the resource groups tagging operations of the AWS CLI or the AWS Resource Groups Tagging API.

AWS CLI

To create a new resource and add a tag to it by using the AWS CLI, use the appropriate `create` command for the resource. Include the `tags` parameter and values. For example, the following command creates a new project named `ExampleCorp` and adds a `Stack` tag key with a `Test` tag value to the project:

```
C:\> aws pinpoint create-app ^
     --create-application-request ^
     --Name=ExampleCorp ^
     --tags={Stack=Test}
```

For information about the commands that you can use to create an Amazon Pinpoint resource, see the AWS CLI Command Reference.

To add a tag to an existing resource, use the `tag-resource` command and specify the appropriate values for the required parameters:

```
C:\> aws pinpoint tag-resource ^
     --resource-arn resource-arn ^
     --tags-model tags={key=value}
```

Where:

- *resource-arn* is the Amazon Resource Name (ARN) of the resource that you want to add a tag to.
- *key* is the tag key that you want to add to the resource. The `key` argument is required.
- *value* is the optional tag value that you want to add for the specified tag key (`key`). The `value` argument is required. If you don't want the resource to have a specific tag value, don't specify a value for the `value` argument. Amazon Pinpoint sets the value to an empty string.

REST API

To create a new resource and add a tag to it by using the Amazon Pinpoint REST API, send a POST request to the appropriate resource URI. Include the `tags` parameter and values in the body of the request. For example, the following request creates a new project named `ExampleCorp` and adds a `Stack` tag key with a `Test` tag value to the project:

```
POST /v1/apps HTTP/1.1
Host: pinpoint.us-east-1.amazonaws.com
Content-Type: application/x-www-form-urlencoded
Accept: application/json
Cache-Control: no-cache

{
   "Name":"ExampleCorp",
   "tags":{
      "Stack":"Test"
   }
}
```

To add a tag to an existing resource, send a POST request to the Tags URI. Include the Amazon Resource Name (ARN) of the resource in the URI. The ARN should be URL encoded. In the body of the request, include the `tags` parameter and values. For example, the following request adds a `Stack` tag key with a `Test` tag value to the specified project (*resource-arn*):

```
POST /v1/tags/resource-arn HTTP/1.1
Host: pinpoint.us-east-1.amazonaws.com
Content-Type: application/json
Accept: application/json
Cache-Control: no-cache

{
   "tags":{
      "Stack":"Test"
   }
}
```

Alternatively, you can send a PUT request to the appropriate resource URI and include the `tags` parameter and values in the body of the request. For example, the following request adds a `Stack` tag key with a `Test` tag value to the specified campaign:

```
PUT /v1/apps/application-id/campaigns/campaign-id HTTP/1.1
Host: pinpoint.us-east-1.amazonaws.com
Content-Type: application/x-www-form-urlencoded
Accept: application/json
```

```
Cache-Control: no-cache

{
   "tags":{
      "Stack":"Test"
   }
}
```

Where:

- *application-id* is the ID of the project that contains the campaign.
- *campaign-id* is the ID of the campaign.

Note that the Amazon Pinpoint API currently doesn't support PUT requests for projects. Therefore, you have to use the Tags resource to add a tag to an existing project.

Displaying tags for resources

The following examples show how to use the AWS CLI and the Amazon Pinpoint REST API to display a list of all the tags (keys and values) that are associated with an Amazon Pinpoint resource. The AWS CLI examples are formatted for Microsoft Windows. For Unix, Linux, and macOS, replace the caret (^) line-continuation character with a backslash (\). You can also use any supported AWS SDK to display all the tags that are associated with a resource.

AWS CLI

To use the AWS CLI to display a list of the tags that are associated with a specific resource, run the `list-tags-for-resource` command and specify the Amazon Resource Name (ARN) of the resource for the `resource-arn` parameter:

```
C:\> aws pinpoint list-tags-for-resource ^
     --resource-arn resource-arn
```

To display a list of all the Amazon Pinpoint resources that have tags, and all the tags that are associated with each of those resources, use the `get-resources` command of the AWS Resource Groups Tagging API. Set the `resource-type-filters` parameter to `mobiletargeting`. For example:

```
C:\> aws resourcegroupstaggingapi get-resources ^
     --resource-type-filters "mobiletargeting"
```

The output of the command is a list of ARNs for all the Amazon Pinpoint resources that have tags. The list includes all the tag keys and values that are associated with each resource.

REST API

To use the Amazon Pinpoint REST API to display all the tags that are associated with a specific resource, send a GET request to the Tags URI and include the Amazon Resource Name (ARN) of the resource in the URI. The ARN should be URL encoded. For example, the following request retrieves all the tags that are associated with a specified campaign (*resource-arn*):

```
GET /v1/tags/resource-arn HTTP/1.1
Host: pinpoint.us-east-1.amazonaws.com
Content-Type: application/json
Accept: application/json
Cache-Control: no-cache
```

The JSON response to the request includes a `tags` object. The `tags` object lists all the tag keys and values that are associated with the campaign.

To display all the tags that are associated with more than one resource of the same type, send a GET request to the appropriate URI for that type of resource. For example, the following request retrieves information about all the campaigns in the specified project (*`application-id`*):

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

The JSON response to the request lists all the campaigns in the project. The `tags` object of each campaign lists all the tag keys and values that are associated with the campaign.

Updating tags for resources

There are several ways to update (overwrite) a tag for an Amazon Pinpoint resource. The best way to update a tag depends on:

- The type of resource that you want to update tags for.
- Whether you want to update a tag for one or multiple resources at the same time.
- Whether you want to update a tag key, a tag value, or both.

To update a tag for an Amazon Pinpoint project or for multiple resources at the same time, use the resource groups tagging operations of the AWS CLI or the AWS Resource Groups Tagging API. The Amazon Pinpoint API currently doesn't provide direct support for either of those tasks.

To update a tag key for one resource, you can remove the current tag (p. 299) and add a new tag (p. 295) by using the Amazon Pinpoint API.

To update a tag value (of a tag key) for only one resource, you can use the Amazon Pinpoint API. The following examples show how to do this by using the AWS CLI and the Amazon Pinpoint REST API. The AWS CLI examples are formatted for Microsoft Windows. For Unix, Linux, and macOS, replace the caret (^) line-continuation character with a backslash (\). You can also use any supported AWS SDK to update a tag value for a resource.

AWS CLI

You can use the AWS CLI to update a tag value for a resource.

To do this by specifying a resource ID, use the appropriate `update` command for the resource type. Include the `tags` parameter and arguments. For example, the following command changes the tag value from `Test` to `Production` for the `Stack` tag key that's associated with the specified campaign:

```
C:\> aws pinpoint update-campaign ^
     --application-id application-id ^
     --campaign-id campaign-id ^
     --write-campaign-request tags={Stack=Production}
```

Where:

- *`application-id`* is the ID of the project that contains the campaign.
- *`campaign-id`* is the ID of the campaign whose tag you want to update.

For information about the commands that you can use to update an Amazon Pinpoint resource, see the AWS CLI Command Reference.

To update a tag value by specifying an Amazon Resource Name (ARN), run the `tag-resource` command and include the `tags-model` parameter and arguments:

```
C:\> aws pinpoint tag-resource ^
     --resource-arn resource-arn ^
     --tags-model tags={key=value}
```

Where:

- *`resource-arn`* is the ARN of the resource whose tag you want to update. To get a list of ARNs for Amazon Pinpoint resources, you can display a list of Amazon Pinpoint resources that have tags (p. 297).
- *`key`* is the tag key. The `key` argument is required.
- *`value`* is the new tag value to use for the specified tag key (`key`). The `value` argument is required. To remove the tag value, don't specify a value for this argument. Amazon Pinpoint sets the value to an empty string.

REST API

You can use the Amazon Pinpoint REST API to update a tag value for a resource. To do this, send a PUT request to the appropriate URI for the type of resource whose tag you want to update. In the body of the request, include the `tags` parameter and values. For the `tags` parameter, specify the corresponding tag key for the `tag` property. For the `value` property, do one of the following:

- To use a new value, specify the value.
- To remove the value, don't specify a value. Amazon Pinpoint sets the value to an empty string.

For example, the following request changes the tag value from `Test` to `Production` for the `Stack` tag key that's associated with the specified campaign:

```
PUT /v1/apps/application-id/campaigns/campaign-id HTTP/1.1
Host: pinpoint.us-east-1.amazonaws.com
Content-Type: application/json
Accept: application/json
Cache-Control: no-cache

{
    "tags": {
        "Stack": "Production"
    }
}
```

Where:

- *`application-id`* is the ID of the project that contains the campaign.
- *`campaign-id`* is the ID of the campaign whose tag you want to update.

Removing tags from resources

The following examples show how to remove a tag (both the key and value) from an Amazon Pinpoint resource by using the AWS CLI and the Amazon Pinpoint REST API. The AWS CLI examples are formatted for Microsoft Windows. For Unix, Linux, and macOS, replace the caret (^) line-continuation character with a backslash (\). You can also use any supported AWS SDK to remove a tag from a resource.

To remove a tag from multiple Amazon Pinpoint resources in a single operation, use the resource groups tagging operations of the AWS CLI or the AWS Resource Groups Tagging API. To remove only a specific tag value (not a tag key) from a resource, update the tag for the resource (p. 298).

AWS CLI

To remove a tag from a resource by using the AWS CLI, run the `untag-resource` command. Include the `tag-keys` parameter and argument:

```
C:\> aws pinpoint untag-resource ^
     --resource-arn resource-arn ^
     --tag-keys key
```

Where:

- *`resource-arn`* is the Amazon Resource Name (ARN) of the resource that you want to remove a tag from. To get a list of ARNs for Amazon Pinpoint resources, you can display a list of all Amazon Pinpoint resources that have tags (p. 297).
- *`key`* is the tag that you want to remove from the resource. The `key` argument is required.

To remove multiple tags from a resource, add each additional key as an argument for the `tag-keys` parameter:

```
C:\> aws pinpoint untag-resource ^
     --resource-arn resource-arn ^
     --tag-keys key1 key2
```

Where:

- *`resource-arn`* is the ARN of the resource that you want to remove tags from.
- *`key#`* is each tag that you want to remove from the resource.

REST API

To remove a tag from a resource by using the Amazon Pinpoint REST API, send a DELETE request to the Tags URI. In the URI, include the Amazon Resource Name (ARN) of the resource that you want to remove a tag from, followed by the `tagKeys` parameter and the tag to remove. For example:

```
https://endpoint/v1/tags/resource-arn?tagKeys=key
```

Where:

- *`endpoint`* is the Amazon Pinpoint endpoint for the AWS Region that hosts the resource.
- *`resource-arn`* is the ARN of the resource that you want to remove a tag from.
- *`key`* is the tag that you want to remove from the resource.

All the parameters should be URL encoded.

To remove multiple tag keys and their associated values from a resource, append the `tagKeys` parameter and argument for each additional tag to remove, separated by an ampersand (&). For example:

```
https://endpoint/v1/tags/resource-arn?tagKeys=key1&tagKeys=key2
```

All the parameters should be URL encoded.

Related information

For more information about the CLI commands that you can use to manage Amazon Pinpoint resources, see the Amazon Pinpoint section of the AWS CLI Command Reference.

For more information about resources in the Amazon Pinpoint API, including supported HTTP(S) methods, parameters, and schemas, see the Amazon Pinpoint API Reference.

Customizing recommendations with AWS Lambda

In Amazon Pinpoint, you can retrieve personalized recommendations from a recommender model and add them to messages that you send from campaigns and journeys. A *recommender model* is a type of machine learning (ML) model that finds patterns in data and generates predictions and recommendations based on the patterns that it finds. It predicts what a particular user will prefer from a given set of products or items, and it provides that information as a set of recommendations for the user.

By using recommender models with Amazon Pinpoint, you can send personalized recommendations to message recipients based on each recipient's attributes and behavior. With AWS Lambda, you can also customize and enhance these recommendations. For example, you can dynamically transform a recommendation from a single text value (such as a product name or ID) to more sophisticated content (such as a product name, description, and image). And you can do it in real time, when Amazon Pinpoint sends the message.

This feature is available in the following AWS Regions: US East (N. Virginia); US West (Oregon); Asia Pacific (Mumbai); Asia Pacific (Sydney); and, Europe (Ireland).

Topics

Using recommendations in messages

To use a recommender model with Amazon Pinpoint, you start by creating an Amazon Personalize solution and deploying that solution as an Amazon Personalize campaign. Then, you create a configuration for the recommender model in Amazon Pinpoint. In the configuration, you specify settings that determine how to retrieve and process recommendation data from the Amazon Personalize campaign. This includes whether to invoke an AWS Lambda function to perform additional processing of the data that's retrieved.

Amazon Personalize is an AWS service that's designed to help you create ML models that provide real-time, personalized recommendations for customers who use your applications. Amazon Personalize guides you through the process of creating and training an ML model, and then preparing and deploying the model as an Amazon Personalize campaign. You can then retrieve real-time, personalized recommendations from the campaign. To learn more about Amazon Personalize, see the Amazon Personalize Developer Guide.

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 AWS Lambda as a *Lambda function*. AWS Lambda then runs the function when the function is invoked. A function can be invoked manually by you, automatically in response to events, or in response to requests from applications or services, including Amazon Pinpoint. For information about creating and invoking Lambda functions, see the AWS Lambda Developer Guide.

After you create an Amazon Pinpoint configuration for a recommender model, you can add recommendations from the model to messages that you send from campaigns and journeys. You do this by using message templates that contain message variables for recommended attributes. A *recommended attribute* is a dynamic endpoint or user attribute that's designed to store recommendation data. You define these attributes when you create the configuration for a recommender model.

You can use variables for recommended attributes in the following types of message templates:

- Email templates, for email messages that you send from campaigns or journeys.
- Push notification templates, for push notifications that you send from campaigns.
- SMS templates, for SMS text messages that you send from campaigns.

For more information about using recommender models with Amazon Pinpoint, see Machine Learning Models in the *Amazon Pinpoint User Guide*.

If you configure Amazon Pinpoint to invoke a Lambda function that processes recommendation data, Amazon Pinpoint performs the following general tasks each time it sends personalized recommendations in a message for a campaign or journey:

1. Evaluates and processes the configuration settings and contents of the message and message template.
2. Determines that the message template is connected to a recommender model.
3. Evaluates the configuration settings for connecting to and using the model. These are defined by the Recommender Model resource for the model.
4. Detects one or more message variables for recommended attributes that are defined by the configuration settings for the model.
5. Retrieves recommendation data from the Amazon Personalize campaign that's specified in the configuration settings for the model. It uses the GetRecommendations operation of the Amazon Personalize Runtime API to perform this task.
6. Adds the appropriate recommendation data to a dynamic recommended attribute (`RecommendationItems`) for each message recipient.
7. Invokes your Lambda function and sends the recommendation data for each recipient to that function for processing.

   The data is sent as a JSON object that contains the endpoint definition for each recipient. Each endpoint definition includes a `RecommendationItems` field that contains an ordered array of 1–5 values. The number of values in the array depends on the configuration settings for the model.
8. Waits for your Lambda function to process the data and return the results.

   The results are a JSON object that contains an updated endpoint definition for each recipient. Each updated endpoint definition contains a new `Recommendations` object. This object contains 1–10 fields, one for each custom recommended attribute that you defined in the configuration settings for the model. Each of these fields stores enhanced recommendation data for the endpoint.
9. Uses the updated endpoint definition for each recipient to replace each message variable with the appropriate value for that recipient.
10. Sends a version of the message that contains the personalized recommendations for each message recipient.

To customize and enhance recommendations in this way, start by creating a Lambda function that processes the endpoint definitions sent by Amazon Pinpoint, and returns updated endpoint definitions. Next, assign a Lambda function policy to the function and authorize Amazon Pinpoint to invoke the function. Then, configure the recommender model in Amazon Pinpoint. When you configure the model, specify the function to invoke and define the recommended attributes to use.

Creating the Lambda function

To learn how to create a Lambda function, see Getting Started in the *AWS Lambda Developer Guide*. When you design and develop your function, keep the following requirements and guidelines in mind.

Input event data

When Amazon Pinpoint invokes a Lambda function for a recommender model, it sends a payload that contains the configuration and other settings for the campaign or journey that's sending the message. The payload includes an `Endpoints` object, which is a map that associates endpoint IDs with endpoint definitions for message recipients.

The endpoint definitions use the structure defined by the Endpoint resource of the Amazon Pinpoint API. However, they also include a field for a dynamic recommended attribute named `RecommendationItems`. The `RecommendationItems` field contains one or more recommended items for the endpoint, as returned from the Amazon Personalize campaign. The value for this field is an ordered array of 1–5 recommended items (as strings). The number of items in the array depends on the number of recommended items that you configured Amazon Pinpoint to retrieve for each endpoint or user.

For example:

```
"Endpoints": {
    "endpointIDexample-1":{
        "ChannelType":"EMAIL",
        "Address":"sofiam@example.com",
        "EndpointStatus":"ACTIVE",
        "OptOut":"NONE",
        "EffectiveDate":"2020-02-26T18:56:24.875Z",
        "Attributes":{
            "AddressType":[
                "primary"
            ]
        },
        "User":{
            "UserId":"SofiaMartínez",
            "UserAttributes":{
                "LastName":[
                    "Martínez"
                ],
                "FirstName":[
                    "Sofia"
                ],
                "Neighborhood":[
                    "East Bay"
                ]
            }
        },
        "RecommendationItems":[
            "1815",
            "2009",
            "1527"
        ],
        "CreationDate":"2020-02-26T18:56:24.875Z"
    },
    "endpointIDexample-2":{
        "ChannelType":"EMAIL",
        "Address":"alejandror@example.com",
        "EndpointStatus":"ACTIVE",
        "OptOut":"NONE",
        "EffectiveDate":"2020-02-26T18:56:24.897Z",
        "Attributes":{
            "AddressType":[
                "primary"
            ]
        },
        "User":{
            "UserId":"AlejandroRosalez",
            "UserAttributes":{
                "LastName ":[
                    "Rosalez"
                ],
                "FirstName":[
                    "Alejandro"
                ],
                "Neighborhood":[
                    "West Bay"
                ]
            }
        },
        "RecommendationItems":[
            "1210",
            "6542",
            "4582"
        ],
        "CreationDate":"2020-02-26T18:56:24.897Z"
    }
}
```

In the preceding example, the relevant Amazon Pinpoint settings are:

- The recommender model is configured to retrieve three recommended items for each endpoint or user. (The value for the `RecommendationsPerMessage` property is set to `3`.) With this setting, Amazon Pinpoint retrieves and adds only the first, second, and third recommended items for each endpoint or user.
- The project is configured to use custom user attributes that store each user's first name, last name, and the neighborhood where they live. (The `UserAttributes` object contains the values for these attributes.)
- The project is configured to use a custom endpoint attribute (`AddressType`) that indicates whether the endpoint is the user's preferred address (channel) for receiving messages from the project. (The `Attributes` object contains the value for this attribute.)

When Amazon Pinpoint invokes the Lambda function and sends this payload as the event data, AWS Lambda passes the data to the Lambda function for processing.

Each payload can contain data for up to 50 endpoints. If a segment contains more than 50 endpoints, Amazon Pinpoint invokes the function repeatedly, for up to 50 endpoints at a time, until the function processes all the data.

Response data and requirements

As you design and develop your Lambda function, keep the quotas for machine learning models (p. 379) in mind. If the function doesn't meet the conditions defined by these quotas, Amazon Pinpoint won't be able to process and send the message.

Also keep the following requirements in mind:

- The function must return updated endpoint definitions in the same format that was provided by the input event data.

- Each updated endpoint definition can contain 1–10 custom recommended attributes for the endpoint or user. The names of these attributes must match the attribute names that you specify when you configure the recommender model in Amazon Pinpoint.
- All custom recommended attributes have to be returned in a single `Recommendations` object for each endpoint or user. This requirement helps ensure that naming conflicts don't occur. You can add the `Recommendations` object to any location in an endpoint definition.
- The value for each custom recommended attribute has to be a string (single value) or an array of strings (multiple values). If the value is an array of strings, we recommend that you maintain the order of the recommended items that Amazon Personalize returned, as indicated in the `RecommendationItems` field. Otherwise, your content might not reflect the model's predictions for an endpoint or user.
- The function shouldn't modify other elements in the event data, including other attribute values for an endpoint or user. It should only add and return values for custom recommended attributes. Amazon Pinpoint won't accept updates to any other values in the function's response.
- The function has to be hosted in the same AWS Region as the Amazon Pinpoint project that's invoking the function. If the function and the project aren't in the same Region, Amazon Pinpoint can't send event data to the function.

If any of the preceding requirements isn't met, Amazon Pinpoint won't be able to process and send the message to one or more endpoints. This might cause a campaign or journey activity to fail.

Finally, we recommend that you reserve 256 concurrent executions for the function.

Overall, your Lambda function should process the event data that's sent by Amazon Pinpoint and return modified endpoint definitions. It can do this by iterating through each endpoint in the `Endpoints` object and, for each endpoint, creating and setting values for the custom recommended attributes that you want to use. The following example handler, written in Python and continuing with the preceding example of input event data, shows this:

```
import json
from random import randrange
import random
import string

def lambda_handler(event, context):
    print("Received event: " + json.dumps(event))
    print("Received context: " +  str(context))
    segment_endpoints = event["Endpoints"]
    new_segment = dict()
    for endpoint_id in segment_endpoints.keys():
        endpoint = segment_endpoints[endpoint_id]
        if supported_endpoint(endpoint):
            new_segment[endpoint_id] = add_recommendation(endpoint)

    print("Returning endpoints: " + json.dumps(new_segment))
    return new_segment

def supported_endpoint(endpoint):
    return True

def add_recommendation(endpoint):
    endpoint["Recommendations"] = dict()

    customTitleList = list()
    customGenreList = list()
    for i,item in enumerate(endpoint["RecommendationItems"]):
        item = int(item)
        if item = 1210:
            customTitleList.insert(i, "Hanna")
            customGenreList.insert(i, "Action")
        elif item = 1527:
            customTitleList.insert(i, "Catastrophe")
            customGenreList.insert(i, "Comedy")
        elif item = 1815:
            customTitleList.insert(i, "Fleabag")
            customGenreList.insert(i, "Comedy")
        elif item = 2009:
            customTitleList.insert(i, "Late Night")
            customGenreList.insert(i, "Drama")
        elif item = 4582:
            customTitleList.insert(i, "Agatha Christie\'s The ABC Murders")
            customGenreList.insert(i, "Crime")
        elif item = 6542:
            customTitleList.insert(i, "Hunters")
            customGenreList.insert(i, "Drama")

    endpoint["Recommendations"]["Title"] = customTitleList
    endpoint["Recommendations"]["Genre"] = customGenreList

    return endpoint
```

In the preceding example, AWS Lambda passes the event data to the handler as the `event` parameter. The handler iterates through each endpoint in the `Endpoints` object and sets values for custom recommended attributes named `Recommendations.Title` and `Recommendations.Genre`. The `return` statement returns each updated endpoint definition to Amazon Pinpoint.

Continuing with the earlier example of input event data, the updated endpoint definitions are:

```
"Endpoints":{
    "endpointIDexample-1":{
        "ChannelType":"EMAIL",
        "Address":"sofiam@example.com",
        "EndpointStatus":"ACTIVE",
        "OptOut":"NONE",
        "EffectiveDate":"2020-02-26T18:56:24.875Z",
        "Attributes":{
            "AddressType":[
                "primary"
            ]
        },
        "User":{
            "UserId":"SofiaMartínez",
            "UserAttributes":{
                "LastName":[
                    "Martínez"
                ],
                "FirstName":[
                    "Sofia"
                ],
                "Neighborhood":[
                    "East Bay"
                ]
            }
        },
        "RecommendationItems":[
            "1815",
            "2009",
            "1527"
        ],
        "CreationDate":"2020-02-26T18:56:24.875Z",
        "Recommendations":{
            "Title":[
                "Fleabag",
                "Late Night",
                "Catastrophe"
            ],
            "Genre":[
                "Comedy",
                "Comedy",
                "Comedy"
            ]
        }
    },
    "endpointIDexample-2":{
        "ChannelType":"EMAIL",
        "Address":"alejandror@example.com",
        "EndpointStatus":"ACTIVE",
        "OptOut":"NONE",
        "EffectiveDate":"2020-02-26T18:56:24.897Z",
        "Attributes":{
            "AddressType":[
                "primary"
            ]
        },
        "User":{
            "UserId":"AlejandroRosalez",
            "UserAttributes":{
                "LastName ":[
                    "Rosalez"
                ],
                "FirstName":[
                    "Alejandro"
                ],
                "Neighborhood":[
                    "West Bay"
                ]
            }
        },
        "RecommendationItems":[
            "1210",
            "6542",
            "4582"
        ],
        "CreationDate":"2020-02-26T18:56:24.897Z",
        "Recommendations":{
            "Title":[
                "Hanna",
                "Hunters",
                "Agatha Christie\'s The ABC Murders"
            ],
            "Genre":[
                "Action",
                "Drama",
                "Crime"
            ]
        }
    }
}
```

In the preceding example, the function modified the `Endpoints` object that it received and returned the results. The `Endpoint` object for each endpoint now contains a new `Recommendations` object, which contains `Title` and `Genre` fields. Each of these fields stores an ordered array of three values (as strings), where each value provides enhanced content for a corresponding recommended item in the `RecommendationItems` field.

Assigning a Lambda function policy

Before you can use your Lambda function to process recommendation data, you must authorize Amazon Pinpoint to invoke the 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 a function and what actions those entities can take. For more information, see Using Resource-Based Policies for AWS Lambda in the *AWS Lambda Developer Guide*.

The following example policy allows the Amazon Pinpoint service principal to use the `lambda:InvokeFunction` action for a particular Amazon Pinpoint campaign (*`campaignId`*) in a particular Amazon Pinpoint project (*`projectId`*):

```
{
  "Sid": "sid",
  "Effect": "Allow",
  "Principal": {
    "Service": "pinpoint.us-east-1.amazonaws.com"
  },
  "Action": "lambda:InvokeFunction",
  "Resource": "{arn:aws:lambda:us-east-1:accountId:function:function-name}",
  "Condition": {
    "ArnLike": {
      "AWS:SourceArn": "arn:aws:mobiletargeting:us-east-1:accountId:recommenders/*"
    }
  }
}
```

The function policy requires a `Condition` block that includes an `AWS:SourceArn` key. This key specifies which resource is allowed to invoke the function. In the preceding example, the policy allows one particular campaign to invoke the function.

You can also write a policy that allows the Amazon Pinpoint service principal to use the `lambda:InvokeFunction` action for all the campaigns and journeys in a specific Amazon Pinpoint project (*`projectId`*). The following example policy shows this:

```
{
  "Sid": "sid",
  "Effect": "Allow",
  "Principal": {
    "Service": "pinpoint.us-east-1.amazonaws.com"
  },
  "Action": "lambda:InvokeFunction",
  "Resource": "{arn:aws:lambda:us-east-1:accountId:function:function-name}",
  "Condition": {
    "ArnLike": {
      "AWS:SourceArn": "arn:aws:mobiletargeting:us-east-1:accountId:recommenders/*"
    }
  }
}
```

Unlike the first example, the `AWS:SourceArn` key in the `Condition` block of this example allows one particular project to invoke the function. This permission applies to all the campaigns and journeys in the project.

To write a more generic policy, you can use a multicharacter match wildcard (*). For example, you can use the following `Condition` block to allow any Amazon Pinpoint project to invoke the function:

```
"Condition": {
  "ArnLike": {
    "AWS:SourceArn": "arn:aws:mobiletargeting:us-east-1:accountId:recommenders/*"
  }
}
```

If you want to use the Lambda function with all the projects for your Amazon Pinpoint account, we recommend that you configure the `Condition` block of the policy in the preceding way. However, as a best practice, you should create policies that include only the permissions that are required to perform a specific action on a specific resource.

Authorizing Amazon Pinpoint to invoke the function

After you assign a Lambda function policy to the function, you can add permissions that allow Amazon Pinpoint to invoke the function for a specific project, campaign, or journey. You can do this using the AWS Command Line Interface (AWS CLI) and the Lambda `add-permission` command. The following example shows how to do this for a specific project (*`projectId`*):

```
$ 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:accountId:recommenders/*
```

The preceding example is formatted for Unix, Linux, and macOS. For Microsoft Windows, replace the backslash (\) line-continuation character with a caret (^).

If the command runs successfully, you see output similar to the following:

```
{
  "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:recommenders/*\"}}}"
}
```

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

Configuring the Recommender Model

To configure Amazon Pinpoint to invoke the Lambda function for a recommender model, specify the following Lambda-specific configuration settings for the model:

- `RecommendationTransformerUri` – This property specifies the name or Amazon Resource Name (ARN) of the Lambda function.
- `Attributes` – This object is a map that defines the custom recommended attributes that the function adds to each endpoint definition. Each of these attributes can be used as a message variable in a message template.

You can specify these settings by using the Recommender Models resource of the Amazon Pinpoint API (when you create the configuration for a model) or the Recommender Model resource of the Amazon Pinpoint API (if you update the configuration for a model). You can also define these settings by using the Amazon Pinpoint console.

For more information about using recommender models with Amazon Pinpoint, see Machine Learning Models in the *Amazon Pinpoint User Guide*.

Deleting data from Amazon Pinpoint

Depending on how you use it, Amazon Pinpoint might store certain data that could be considered personal. For example, an 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 console or the Amazon Pinpoint API to permanently delete personal data. This topic 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, phone number, or other type of destination that you can send messages to. In many jurisdictions, this type of information might be considered personal.

To delete all the data for a specific endpoint, you can use the Amazon Pinpoint API to delete the endpoint. The following procedure shows how to delete an endpoint by using 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 CLI in the *AWS Command Line Interface User Guide*.

To delete an endpoint by using the AWS CLI

- At the command line, enter the following command:

```
aws pinpoint delete-endpoint --application-id 810c7aab86d42fb2b56c8c966example --
endpoint-id ad015a3bf4f1b2b0b82example
```

  In the preceding command, replace `810c7aab86d42fb2b56c8c966example` with the ID of the project that the endpoint is associated with. Also, replace `ad015a3bf4f1b2b0b82example` with the unique ID of the endpoint itself.

To find the endpoint ID for a specific endpoint, determine which segment the endpoint belongs to, and then export the segment from Amazon Pinpoint. The exported data includes the endpoint ID for each endpoint. You can export a segment to a file by using the Amazon Pinpoint console. To learn how, see Exporting Segments in the *Amazon Pinpoint User Guide*. You can also export a segment to an Amazon Simple Storage Service (Amazon S3) bucket by using the Amazon Pinpoint API. To learn how, see Exporting Endpoints (p. 143) in this guide.

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 an endpoint. You can delete these files from Amazon S3.

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?.

Deleting all project data

It's possible to permanently delete all the data that you've stored for an Amazon Pinpoint project. You can do this by deleting the project.

Warning
If you delete a project, Amazon Pinpoint deletes all project-specific settings and data for the project. The information can't be recovered.

When you delete a project, Amazon Pinpoint deletes all project-specific settings for the push notification and two-way SMS messaging channels, and all segments, campaigns, journeys, and project-specific analytics data that's stored in Amazon Pinpoint, such as the following:

- Segments – All segment settings and data. For dynamic segments, this includes segment groups and filters that you defined. For imported segments, this includes endpoints, user IDs, and other data that you imported, and any filters that you applied.
- Campaigns – All messages, message treatments and variables, analytics data, schedules, and other settings.
- Journeys – All activities, analytics data, schedules, and other settings.
- Analytics – Data for all engagement metrics, such as the number of messages sent and delivered for campaigns and journeys, and all journey execution metrics. For mobile and web apps, all event data that wasn’t streamed to another AWS service such as Amazon Kinesis, all funnels, and data for application usage, revenue, and demographic metrics. Before you delete a project, we recommend that you export this data to another location.

You can delete a project by using the Amazon Pinpoint console. To learn more, see Deleting a Project in the *Amazon Pinpoint User Guide*. You can also delete a project programmatically by using the App resource of the Amazon Pinpoint API.

Deleting all AWS data by closing your AWS account

It's also possible to delete all the 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 AWS Regions.

You can close your AWS account by using the AWSManagement Console.

To close your AWS account

1. Open the AWSManagement 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 steps permanently delete all the data that 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**.

Security in Amazon Pinpoint

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS Compliance Programs. To learn about the compliance programs that apply to Amazon Pinpoint, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using Amazon Pinpoint. The following topics show you how to configure Amazon Pinpoint to meet your security and compliance objectives. You also learn how to use other AWS services that help you monitor and secure your Amazon Pinpoint resources.

Topics

Data protection in Amazon Pinpoint

The AWS shared responsibility model applies to data protection in Amazon Pinpoint. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWSCloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. This content includes the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWSaccount credentials and set up individual user accounts with AWS Identity and Access Management (IAM). That way each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We recommend TLS 1.2 or later.

- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing personal data that is stored in Amazon S3.
- If you require FIPS 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-2.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form fields such as a **Name** field. This includes when you work with Amazon Pinpoint or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Depending on how you configure and use the service, Amazon Pinpoint might store the following types of personal data for you or about your customers:

Configuration data

This includes project configuration data such as credentials and settings that define how and when Amazon Pinpoint sends messages through supported channels, and the user segments that it sends messages to. To send messages, this data can include dedicated IP addresses for email messages, short codes and sender IDs for SMS text messages, and credentials for communicating with push notification services such as the Apple Push Notification service (APNs) and Firebase Cloud Messaging (FCM).

User and endpoint data

This includes standard and custom attributes that you use to store and manage data about users and endpoints for an Amazon Pinpoint project. An attribute can store information about a specific user (such as a user's name) or a specific endpoint for a user (such as a user's email address, mobile phone number, or mobile device token). This data can also include external user IDs that correlate users for an Amazon Pinpoint project with users in an external system, such as a customer relationship management system. For more information about what this data can include, see the User and Endpoint schemas in the *Amazon Pinpoint API Reference*.

Analytics data

This includes data for metrics, also referred to as *key performance indicators (KPIs)*, that provide insight into the performance of an Amazon Pinpoint project for areas such as user engagement and purchase activity. This also includes data for metrics that provide insight into user demographics for a project. The data can derive from standard and custom attributes for users and endpoints, such as the city where a user lives. It can also derive from events, such as open and click events for the email messages that you send for a project.

Imported data

This includes any user, segmentation, and analytics data that you add or import from external sources and use in Amazon Pinpoint. An example is a JSON file that you import into Amazon Pinpoint (through the console directly or from an Amazon S3 bucket) to build a static segment. Other examples are endpoint data that you add programmatically to build a dynamic segment, endpoint addresses that you send direct messages to, and events that you configure an app to report to Amazon Pinpoint.

Topics

Data encryption

Amazon Pinpoint data is encrypted in transit and at rest. When you submit data to Amazon Pinpoint, it encrypts the data as it receives and stores it. When you retrieve data from Amazon Pinpoint, it transmits the data to you by using current security protocols.

Encryption at rest

Amazon Pinpoint encrypts all the data that it stores for you. This includes configuration data, user and endpoint data, analytics data, and any data that you add or import into Amazon Pinpoint. To encrypt your data, Amazon Pinpoint uses internal AWS Key Management Service (AWS KMS) keys that the service owns and maintains on your behalf. We rotate these keys on a regular basis. For information about AWS KMS, see the AWS Key Management Service Developer Guide.

Encryption in transit

Amazon Pinpoint uses HTTPS and Transport Layer Security (TLS) 1.0 or later to communicate with your clients and applications. To communicate with other AWS services, Amazon Pinpoint uses HTTPS and TLS 1.2. In addition, when you create and manage Amazon Pinpoint resources by using the console, an AWS SDK, or the AWS Command Line Interface, all communications are secured using HTTPS and TLS 1.2.

Key management

To encrypt your Amazon Pinpoint data, Amazon Pinpoint uses internal AWS KMS keys that the service owns and maintains on your behalf. We rotate these keys on a regular basis. You can't provision and use your own AWS KMS or other keys to encrypt data that you store in Amazon Pinpoint.

Internetwork traffic privacy

Internetwork traffic privacy refers to securing connections and traffic between Amazon Pinpoint and your on-premises clients and applications, and between Amazon Pinpoint and other AWS resources in the same AWS Region. The following features and practices can help you ensure internetwork traffic privacy for Amazon Pinpoint.

Traffic between Amazon Pinpoint and on-premises clients and applications

To establish a private connection between Amazon Pinpoint and clients and applications on your on-premises network, you can use AWS Direct Connect. This enables you to link your network to an AWS Direct Connect location by using a standard, fiber-optic Ethernet cable. One end of the cable is connected to your router. The other end is connected to an AWS Direct Connect router. For more information, see What is AWS Direct Connect? in the *AWS Direct Connect User Guide*.

To help secure access to Amazon Pinpoint through published APIs, we recommend that you comply with Amazon Pinpoint requirements for API calls. Amazon Pinpoint requires clients to use Transport Layer Security (TLS) 1.0 or later. (We recommend TLS 1.2 or later.) Clients must also support cipher suites with perfect forward secrecy (PFS), such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Diffie-Hellman Ephemeral (ECDHE). Most modern systems such as Java 7 and later support these modes.

In addition, requests must be signed using an access key ID and a secret access key that's associated with an AWS Identity and Access Management (IAM) principal for your AWS account. Alternatively, you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

Traffic between Amazon Pinpoint and other AWS resources

To secure communications between Amazon Pinpoint and other AWS resources in the same AWS Region, Amazon Pinpoint uses HTTPS and TLS 1.2 by default.

Identity and access management for Amazon Pinpoint

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use Amazon Pinpoint resources. IAM is an AWS service that you can use with no additional charge.

Topics

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in Amazon Pinpoint.

Service user – If you use the Amazon Pinpoint service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more Amazon Pinpoint features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in Amazon Pinpoint, see Troubleshooting Amazon Pinpoint identity and access management (p. 368).

Service administrator – If you're in charge of Amazon Pinpoint resources at your company, you probably have full access to Amazon Pinpoint. It's your job to determine which Amazon Pinpoint features and resources your employees should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with Amazon Pinpoint, see How Amazon Pinpoint works with IAM (p. 322).

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to Amazon Pinpoint. To view example Amazon Pinpoint identity-based policies that you can use in IAM, see Amazon Pinpoint identity-based policy examples (p. 347).

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. For more information about signing in using the AWSManagement Console, see Signing in to the AWSManagement Console as an IAM user or root user in the *IAM User Guide*.

You must be *authenticated* (signed in to AWS) as the AWSaccount root user, an IAM user, or by assuming an IAM role. You can also use your company's single sign-on authentication or even sign in using Google or Facebook. In these cases, your administrator previously set up identity federation using IAM roles. When you access AWS using credentials from another company, you are assuming a role indirectly.

To sign in directly to the AWSManagement Console, use your password with your root user email address or your IAM user name. You can access AWS programmatically using your root user or IAM users access keys. AWS provides SDK and command line tools to cryptographically sign your request using your credentials. If you don't use AWS tools, you must sign the request yourself. Do this using *Signature Version 4*, a protocol for authenticating inbound API requests. For more information about authenticating requests, see Signature Version 4 signing process in the *AWS General Reference*.

Regardless of the authentication method that you use, you might also be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Using multi-factor authentication (MFA) in AWS in the *IAM User Guide*.

AWSaccount root user

When you first create an AWSaccount, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWSaccount *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the best practice of using the root user only to create your first IAM user. Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.

IAM users and groups

An *IAM user* is an identity within your AWSaccount that has specific permissions for a single person or application. An IAM user can have long-term credentials such as a user name and password or a set of access keys. To learn how to generate access keys, see Managing access keys for IAM users in the *IAM User Guide*. When you generate access keys for an IAM user, make sure you view and securely save the key pair. You cannot recover the secret access key in the future. Instead, you must generate a new access key pair.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see When to create an IAM user (instead of a role) in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity within your AWSaccount that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWSManagement Console by switching roles. You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Using IAM roles in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Temporary IAM user permissions** – An IAM user can assume an IAM role to temporarily take on different permissions for a specific task.

- **Federated user access** – Instead of creating an IAM user, you can use existing identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as *federated users*. AWS assigns a role to a federated user when access is requested through an identity provider. For more information about federated users, see Federated users and roles in the *IAM User Guide*.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
  - **Principal permissions** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional dependent actions in a policy, see Actions, Resources, and Condition Keys for Amazon Pinpoint in the *Service Authorization Reference*.
  - **Service role** – A service role is an IAM role that a service assumes to perform actions on your behalf. Service roles provide access only within your account and cannot be used to grant access to services in other accounts. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.
  - **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Using an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*.

To learn whether to use IAM roles or IAM users, see When to create an IAM role (instead of a user) in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to IAM identities or AWS resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. You can sign in as the root user or an IAM user, or you can assume an IAM role. When you then make a request, AWS evaluates the related identity-based or resource-based policies. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

Every IAM entity (user or role) starts with no permissions. In other words, by default, users can do nothing, not even change their own password. To give a user permission to do something, an administrator must attach a permissions policy to a user. Or the administrator can add the user to a group that has the intended permissions. When an administrator gives permissions to a group, all users in that group are granted those permissions.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWSManagement Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWSaccount. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choosing between managed policies and inline policies in the *IAM User Guide*.

Amazon Pinpoint supports the use of identity-based policies to control access to Amazon Pinpoint resources.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Amazon Pinpoint supports the use of resource-based policies to control access to Amazon Pinpoint resources.

Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

Amazon Pinpoint doesn't support the use of ACLs to control access to Amazon Pinpoint resources.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role).

You can set a permissions boundary for an entity. The resulting permissions are the intersection of entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the *IAM User Guide*.

- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWSaccounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWSaccount root user. For more information about Organizations and SCPs, see How SCPs work in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session policies in the *IAM User Guide*.

Amazon Pinpoint supports the use of these types of policies to control access to Amazon Pinpoint resources.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

How Amazon Pinpoint works with IAM

To use Amazon Pinpoint, users in your AWS account require permissions that allow them to view analytics data, create projects, define user segments, deploy campaigns, and more. If you integrate a mobile or web app with Amazon Pinpoint, users of your app also require access to Amazon Pinpoint. This access enables your app to register endpoints and report usage data to Amazon Pinpoint. To grant access to Amazon Pinpoint features, create AWS Identity and Access Management (IAM) policies that allow Amazon Pinpoint actions for IAM identities or Amazon Pinpoint resources.

IAM is a service that helps administrators securely control access to AWS resources. IAM policies include statements that allow or deny specific actions by specific users or for specific resources. Amazon Pinpoint provides a set of actions (p. 326) that you can use in IAM policies to specify granular permissions for Amazon Pinpoint users and resources. This means that you can grant the appropriate level of access to Amazon Pinpoint without creating overly permissive policies that might expose important data or compromise your resources. For example, you can grant unrestricted access to an Amazon Pinpoint administrator, and grant read-only access to individuals who need access to only a specific project.

Before you use IAM to manage access to Amazon Pinpoint, you should understand what IAM features are available for use with Amazon Pinpoint. To get a high-level view of how Amazon Pinpoint and other AWS services work with IAM, see AWS services that work with IAM in the *IAM User Guide*.

Topics

Amazon Pinpoint identity-based policies

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. Amazon Pinpoint supports specific actions, resources, and condition keys. To learn about all the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Actions

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

This means that policy actions control what users can do on the Amazon Pinpoint console. They also control what users can do programmatically by using the AWS SDKs, the AWS Command Line Interface (AWS CLI), or the Amazon Pinpoint APIs directly.

Policy actions in Amazon Pinpoint use the following prefixes:

- **`mobiletargeting`** – For actions that derive from the Amazon Pinpoint API, which is the primary API for Amazon Pinpoint.
- **`sms-voice`** – For actions that derive from the SMS and Voice API, which is a supplemental API that provides advanced options for using and managing the SMS and voice channels in Amazon Pinpoint.

For example, to grant someone permission to view information about all the segments for a project, which is an action that corresponds to the `GetSegments` operation in the Amazon Pinpoint API, include the `mobiletargeting:GetSegments` action in their policy. Policy statements must include either an `Action` or `NotAction` element. Amazon Pinpoint defines its own set of actions that describe the tasks that users can perform with it.

To specify multiple actions in a single statement, separate them with commas:

```
"Action": [
      "mobiletargeting:action1",
      "mobiletargeting:action2"
```

You can also specify multiple actions by using wildcards (*). For example, to specify all actions that begin with the word `Get`, include the following action:

```
"Action": "mobiletargeting:Get*"
```

However, as a best practice, you should create policies that follow the principle of *least privilege*. In other words, you should create policies that include only the permissions that are required to perform a specific action.

For a list of Amazon Pinpoint actions that you can use in IAM policies, see Amazon Pinpoint actions for IAM policies (p. 326).

Resources

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its Amazon Resource Name (ARN). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

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

For example, the `mobiletargeting:GetSegments` action retrieves information about all the segments that are associated with a specific Amazon Pinpoint project. You identify a project with an ARN in the following format:

```
arn:aws:mobiletargeting:${Region}:${Account}:apps/${projectId}
```

For more information about the format of ARNs, see Amazon Resource Names (ARNs) in the *AWS General Reference*.

In IAM policies, you can specify ARNs for the following types of Amazon Pinpoint resources:

- Campaigns
- Journeys
- Message templates (referred to as *templates* in some contexts)
- Projects (referred to as *apps* or *applications* in some contexts)
- Recommender models (referred to as *recommenders* in some contexts)
- Segments

For example, to create a policy statement for the project that has the project ID `810c7aab86d42fb2b56c8c966example`, use the following ARN:

```
"Resource": "arn:aws:mobiletargeting:us-
east-1:123456789012:apps/810c7aab86d42fb2b56c8c966example"
```

To specify all the projects that belong to a specific account, use the wildcard (*):

```
"Resource": "arn:aws:mobiletargeting:us-east-1:123456789012:apps/*"
```

Some Amazon Pinpoint actions, such as certain actions for creating resources, can't be performed on a specific resource. In those cases, you must use the wildcard (*):

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

Some Amazon Pinpoint API actions involve multiple resources. For example, the `TagResource` action can add a tag to multiple projects. To specify multiple resources in a single statement, separate the ARNs with commas:

```
"Resource": [
      "resource1",
      "resource2"
```

To see a list of Amazon Pinpoint resource types and their ARNs, see Resources Defined by Amazon Pinpoint in the *IAM User Guide*. To learn which actions you can specify with the ARN of each resource type, see Actions Defined by Amazon Pinpoint in the *IAM User Guide*.

Condition keys

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element (or `Condition` *block*) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical `AND` operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical `OR` operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM policy elements: variables and tags in the *IAM User Guide*.

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

Amazon Pinpoint defines its own set of condition keys and also supports some global condition keys. To see a list of all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*. To see a list of Amazon Pinpoint condition keys, see Condition Keys for Amazon Pinpoint in the *IAM User Guide*. To learn which actions and resources you can use a condition key with, see Actions Defined by Amazon Pinpoint in the *IAM User Guide*.

Examples

To view examples of Amazon Pinpoint identity-based policies, see Amazon Pinpoint identity-based policy examples (p. 347).

Amazon Pinpoint resource-based policies

Resource-based policies are JSON policy documents that specify what actions a specified principal can perform on an Amazon Pinpoint resource and under what conditions. Amazon Pinpoint supports resource-based permissions policies for campaigns, journeys, message templates (*templates*), recommender models (*recommenders*), projects (*apps*), and segments. Resource-based policies let you grant usage permission to other accounts on a per-resource basis. You can also use a resource-based policy to allow another AWS service to access these types of Amazon Pinpoint resources.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. Adding a cross-account principal to a resource-based policy is only half of establishing the trust relationship. When the principal and the resource are in different AWS accounts, you must also grant the principal entity permission to access the resource. Grant permission by attaching an identity-based policy to the entity. However, if a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required. For more information, see How IAM roles differ from resource-based policies in the *IAM User Guide*.

Examples

To view examples of Amazon Pinpoint resource-based policies, see the section called “Identity-based policy examples” (p. 347).

Authorization based on Amazon Pinpoint tags

You can associate tags with certain types of Amazon Pinpoint resources or pass tags in a request to Amazon Pinpoint. To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/${TagKey}`, `aws:RequestTag/${TagKey}`, or `aws:TagKeys` condition keys.

For information about tagging Amazon Pinpoint resources, including an example IAM policy, see Tagging Amazon Pinpoint resources (p. 294).

Amazon Pinpoint IAM roles

An IAM role is an entity within your AWS account that has specific permissions.

Using temporary credentials with Amazon Pinpoint

You can use temporary credentials to sign in with federation, assume an IAM role, or assume a cross-account role. You obtain temporary security credentials by calling AWS Security Token Service (AWSSTS) API operations such as AssumeRole or GetFederationToken.

Amazon Pinpoint supports using temporary credentials.

Service-linked roles

Service-linked roles allow AWS services to access resources in other services to complete an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view but not edit the permissions for service-linked roles.

Amazon Pinpoint doesn't use service-linked roles.

Service roles

This feature allows a service to assume a service role on your behalf. This role allows the service to access resources in other services to complete an action on your behalf. Service roles appear in your IAM account and are owned by the account. This means that an IAM administrator can change the permissions for this role. However, doing so might break the functionality of the service.

Amazon Pinpoint supports using service roles.

Amazon Pinpoint actions for IAM policies

To manage access to Amazon Pinpoint resources in your AWS account, you can add Amazon Pinpoint actions to AWS Identity and Access Management (IAM) policies. By using actions in policies, you can control what users can do on the Amazon Pinpoint console. You can also control what users can do programmatically by using the AWS SDKs, the AWS Command Line Interface (AWS CLI), or the Amazon Pinpoint APIs directly.

In a policy, you specify each action with the appropriate Amazon Pinpoint namespace followed by a colon and the name of the action, such as `GetSegments`. Most actions correspond to a request to the Amazon Pinpoint 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 retrieve information about all the segments for a project by submitting an HTTP GET request to the `/apps/projectId/segments` URI. This policy also allows the user to view that information on the console, and retrieve that information 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 project, which you identify with the ARN, `arn:aws:mobiletargeting:region:accountId:apps/projectId`.

This topic identifies Amazon Pinpoint actions that you can add to IAM policies for your AWS account. To see examples that demonstrate how you can use actions in policies to manage access to Amazon Pinpoint resources, see Amazon Pinpoint identity-based policy examples (p. 347).

Topics

Amazon Pinpoint API actions

This section identifies actions for features that are available from the Amazon Pinpoint API, which is the primary API for Amazon Pinpoint. To learn more about this API, see the Amazon Pinpoint API Reference.

Categories:

Analytics and metrics

The following permissions are related to viewing analytics data on the Amazon Pinpoint console. They're also related to retrieving (querying) aggregated data for standard metrics, also referred to as *key performance indicators (KPIs)*, that apply to projects, campaigns, and journeys.

`mobiletargeting:GetReports`

View analytics data on the Amazon Pinpoint console.

- URI – Not applicable
- Method – Not applicable
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`accountId`*`:*`

`mobiletargeting:GetApplicationDateRangeKpi`

Retrieve (query) aggregated data for a standard application metric. This is a metric that applies to all the campaigns or transactional messages that are associated with a project.

- URI – `/apps/`*`projectId`*`/kpis/daterange/`*`kpi-name`*
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`accountId`*`:apps/`*`projectId`*`/kpis/daterange/`*`kpi-name`*

`mobiletargeting:GetCampaignDateRangeKpi`

Retrieve (query) aggregated data for a standard campaign metric. This is a metric that applies to an individual campaign.

- URI – `/apps/`*`projectId`*`/campaigns/`*`campaignId`*`/kpis/daterange/`*`kpi-name`*
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`accountId`*`:apps/`*`projectId`*`/campaigns/`*`campaignId`*`/kpis/daterange/`*`kpi-name`*

`mobiletargeting:GetJourneyDateRangeKpi`

Retrieve (query) aggregated data for a standard journey engagement metric. This is an engagement metric that applies to an individual journey—for example, the number of messages that were opened by participants for all the activities in a journey.

- URI – `/apps/`*`projectId`*`/journeys/`*`journeyId`*`/kpis/daterange/`*`kpi-name`*
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`accountId`*`:apps/`*`projectId`*`/journeys/`*`journeyId`*`/kpis/daterange/`*`kpi-name`*

`mobiletargeting:GetJourneyExecutionMetrics`

Retrieve (query) aggregated data for standard execution metrics that apply to an individual journey—for example, the number of participants who are actively proceeding through all the activities in a journey.

- URI – `/apps/`*`projectId`*`/journeys/`*`journeyId`*`/execution-metrics`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`accountId`*`:apps/`*`projectId`*`/journeys/`*`journeyId`*`/execution-metrics`

`mobiletargeting:GetJourneyExecutionActivityMetrics`

Retrieve (query) aggregated data for standard execution metrics that apply to an individual activity in a journey—for example, the number of participants who started or completed an activity.

- URI – `/apps/`*`projectId`*`/journeys/`*`journeyId`*`/activities/`*`journey-activity-id`*`/execution-metrics`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`accountId`*`:apps/`*`projectId`*`/journeys/`*`journeyId`*`/activities/`*`journey-activity-id`*`/execution-metrics`

Campaigns

The following permissions are related to managing campaigns in your Amazon Pinpoint account.

`mobiletargeting:CreateCampaign`

Create a campaign for a project.

- URI – `/apps/projectId/campaigns`
- Method – POST
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/campaigns`

`mobiletargeting:DeleteCampaign`

Delete a specific campaign.

- URI – `/apps/projectId/campaigns/campaignId`
- Method – DELETE
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/campaigns/campaignId`

`mobiletargeting:GetCampaign`

Retrieve information about a specific campaign.

- URI – `/apps/projectId/campaigns/campaignId`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/campaigns/campaignId`

`mobiletargeting:GetCampaignActivities`

Retrieve information about the activities performed by a campaign.

- URI – `/apps/projectId/campaigns/campaignId/activities`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/campaigns/campaignId`

`mobiletargeting:GetCampaigns`

Retrieve information about all campaigns for a project.

- URI – `/apps/projectId/campaigns`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId`

`mobiletargeting:GetCampaignVersion`

Retrieve information about a specific campaign version.

- URI – `/apps/projectId/campaigns/campaignId/versions/versionId`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/campaigns/campaignId`

`mobiletargeting:GetCampaignVersions`

Retrieve information about the current and prior versions of a campaign.

- URI – `/apps/projectId/campaigns/campaignId/versions`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/campaigns/campaignId`

`mobiletargeting:UpdateCampaign`

Update a specific campaign.

- URI – `/apps/projectId/campaigns/campaignId`
- Method – PUT
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/campaigns/campaignId`

Channels

The following permissions are related to managing channels in your Amazon Pinpoint account. In Amazon Pinpoint, *channels* refer to the methods that you use to contact your customers, such as sending email, SMS messages, or push notifications.

`mobiletargeting:DeleteAdmChannel`

Disable the Amazon Device Messaging (ADM) channel for a project.

- URI – `/apps/projectId/channels/adm`
- Method – DELETE
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/channels/adm`

`mobiletargeting:GetAdmChannel`

Retrieve information about the ADM channel for a project.

- URI – `/apps/projectId/channels/adm`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/channels/adm`

`mobiletargeting:UpdateAdmChannel`

Enable or update the ADM channel for a project.

- URI – `/apps/projectId/channels/adm`
- Method – PUT
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/channels/adm`

`mobiletargeting:DeleteApnsChannel`

Disable the Apple Push Notification service (APNs) channel for a project.

- URI – `/apps/projectId/channels/apns`
- Method – DELETE
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/channels/apns`

`mobiletargeting:GetApnsChannel`

Retrieve information about the APNs channel for a project.

- URI – `/apps/projectId/channels/apns`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/channels/apns`

`mobiletargeting:UpdateApnsChannel`

Enable or update the APNs channel for a project.

- URI – `/apps/projectId/channels/apns`

- Method – PUT
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`accountId`*`:apps/`*`projectId`*`/channels/apns`

`mobiletargeting:DeleteApnsSandboxChannel`

Disable the APNs sandbox channel for a project.

- URI – `/apps/`*`projectId`*`/channels/apns_sandbox`
- Method – DELETE
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`accountId`*`:apps/`*`projectId`*`/channels/apns_sandbox`

`mobiletargeting:GetApnsSandboxChannel`

Retrieve information about the APNs sandbox channel for a project.

- URI – `/apps/`*`projectId`*`/channels/apns_sandbox`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`accountId`*`:apps/`*`projectId`*`/channels/apns_sandbox`

`mobiletargeting:UpdateApnsSandboxChannel`

Enable or update the APNs sandbox channel for a project.

- URI – `/apps/`*`projectId`*`/channels/apns_sandbox`
- Method – PUT
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`accountId`*`:apps/`*`projectId`*`/channels/apns_sandbox`

`mobiletargeting:DeleteApnsVoipChannel`

Disable the APNs VoIP channel for a project.

- URI – `/apps/`*`projectId`*`/channels/apns_voip`
- Method – DELETE
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`accountId`*`:apps/`*`projectId`*`/channels/apns_voip`

`mobiletargeting:GetApnsVoipChannel`

Retrieve information about the APNs VoIP channel for a project.

- URI – `/apps/`*`projectId`*`/channels/apns_voip`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`accountId`*`:apps/`*`projectId`*`/channels/apns_voip`

`mobiletargeting:UpdateApnsVoipChannel`

Enable or update the APNs VoIP channel for a project.

- URI – `/apps/`*`projectId`*`/channels/apns_voip`
- Method – PUT
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`accountId`*`:apps/`*`projectId`*`/channels/apns_voip`

`mobiletargeting:DeleteApnsVoipSandboxChannel`

Disable the APNs VoIP sandbox channel for a project.

- URI – `/apps/`*`projectId`*`/channels/apns_voip_sandbox`

- Method – DELETE
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/channels/apns_voip_sandbox`

`mobiletargeting:GetApnsVoipSandboxChannel`

Retrieve information about the APNs VoIP sandbox channel for a project.

- URI – `/apps/projectId/channels/apns_voip_sandbox`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/channels/apns_voip_sandbox`

`mobiletargeting:UpdateApnsVoipSandboxChannel`

Enable or update the APNs VoIP sandbox channel for a project.

- URI – `/apps/projectId/channels/apns_voip_sandbox`
- Method – PUT
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/channels/apns_voip_sandbox`

`mobiletargeting:DeleteBaiduChannel`

Disable the Baidu Cloud Push channel for a project.

- URI – `/apps/projectId/channels/baidu`
- Method – DELETE
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/channels/baidu`

`mobiletargeting:GetBaiduChannel`

Retrieve information about the Baidu Cloud Push channel for a project.

- URI – `/apps/projectId/channels/baidu`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/channels/baidu`

`mobiletargeting:UpdateBaiduChannel`

Enable or update the Baidu Cloud Push channel for a project.

- URI – `/apps/projectId/channels/baidu`
- Method – PUT
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/channels/baidu`

`mobiletargeting:DeleteEmailChannel`

Disable the email channel for a project.

- URI – `/apps/projectId/channels/email`
- Method – DELETE
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/channels/email`

`mobiletargeting:GetEmailChannel`

Retrieve information about the email channel for a project.

- URI – `/apps/projectId/channels/email`
- Method – GET

- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/channels/email`

`mobiletargeting:UpdateEmailChannel`

Enable or update the email channel for a project.

- URI – `/apps/projectId/channels/email`
- Method – PUT
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/channels/email`

`mobiletargeting:DeleteGcmChannel`

Disable the Firebase Cloud Messaging (FCM) channel for a project. This channel allows Amazon Pinpoint to send push notifications to an Android app through the FCM service, which replaces the Google Cloud Messaging (GCM) service.

- URI – `/apps/projectId/channels/gcm`
- Method – DELETE
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/channels/gcm`

`mobiletargeting:GetGcmChannel`

Retrieve information about the FCM channel for a project. This channel allows Amazon Pinpoint to send push notifications to an Android app through the FCM service, which replaces the Google Cloud Messaging (GCM) service.

- URI – `/apps/projectId/channels/gcm`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/channels/gcm`

`mobiletargeting:UpdateGcmChannel`

Enable or update the FCM channel for a project. This channel allows Amazon Pinpoint to send push notifications to an Android app through the FCM service, which replaces the Google Cloud Messaging (GCM) service.

- URI – `/apps/projectId/channels/gcm`
- Method – PUT
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/channels/gcm`

`mobiletargeting:DeleteSmsChannel`

Disable the SMS channel for a project.

- URI – `/apps/projectId/channels/sms`
- Method – DELETE
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/channels/sms`

`mobiletargeting:GetSmsChannel`

Retrieve information about the SMS channel for a project.

- URI – `/apps/projectId/channels/sms`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/channels/sms`

`mobiletargeting:UpdateSmsChannel`

Enable or update the SMS channel for a project.

- URI – `/apps/projectId/channels/sms`
- Method – PUT
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/channels/sms`

`mobiletargeting:GetChannels`

Retrieves information about the history and status of each channel for an application.

- URI – `/apps/application-id/channels`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/channels`

`mobiletargeting:DeleteVoiceChannel`

Disables the voice channel for an application and deletes any existing settings for the channel.

- URI – `/apps/application-id/channels/voice`
- Method – DELETE
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectid/channels/voice`

`mobiletargeting:GetVoiceChannel`

Retrieves information about the status and settings of the voice channel for an application.

- URI – `/apps/application-id/channels/voice`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectid/channels/voice`

`mobiletargeting:UpdateVoiceChannel`

Enables the voice channel for an application or updates the status and settings of the voice channel for an application.

- URI – `/apps/application-id/channels/voice`
- Method – PUT
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectid/channels/voice`

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/projectId/endpoints/endpointId`
- Method – DELETE
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/endpoints/endpointId`

`mobiletargeting:GetEndpoint`

Retrieve information about a specific endpoint.

- URI – `/apps/projectId/endpoints/endpointId`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/endpoints/endpointId`

`mobiletargeting:RemoveAttributes`

Removes one or more attributes, of the same attribute type, from all the endpoints that are associated with an application.

- URI – `apps/application-id/attributes/attribute-type`
- Method – PUT
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/attributes/attribute-type`

`mobiletargeting:UpdateEndpoint`

Create an endpoint or update the information for an endpoint.

- URI – `/apps/projectId/endpoints/endpointId`
- Method – PUT
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/endpoints/endpointId`

`mobiletargeting:UpdateEndpointsBatch`

Create or update endpoints as a batch operation.

- URI – `/apps/projectId/endpoints`
- Method – PUT
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId`

Event streams

The following permissions are related to managing event streams for your Amazon Pinpoint account.

`mobiletargeting:DeleteEventStream`

Delete the event stream for a project.

- URI – `/apps/projectId/eventstream/`
- Method – DELETE
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/eventstream`

`mobiletargeting:GetEventStream`

Retrieve information about the event stream for a project.

- URI – `/apps/projectId/eventstream/`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/eventstream`

`mobiletargeting:PutEventStream`

Create or update an event stream for a project.

- URI – `/apps/projectId/eventstream/`

- Method – POST
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/eventstream`

Events

The following permissions are related to managing events jobs in your Amazon Pinpoint account. In Amazon Pinpoint, you create *import jobs* to create segments based on endpoint definitions that are stored in an Amazon S3 bucket.

`mobiletargeting:PutEvents`

Creates a new event to record for endpoints, or creates or updates endpoint data that existing events are associated with.

- URI – `/apps/application-id/events`
- Method – POST
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/events`

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:CreateExportJob`

Create an export job for exporting endpoint definitions to Amazon S3.

- URI – `/apps/projectId/jobs/export`
- Method – POST
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/jobs/export`

`mobiletargeting:GetExportJob`

Retrieve information about a specific export job for a project.

- URI – `/apps/projectId/jobs/export/jobId`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/jobs/export/jobId`

`mobiletargeting:GetExportJobs`

Retrieve a list of all the export jobs for a project.

- URI – `/apps/projectId/jobs/export`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/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 that are stored in an Amazon S3 bucket.

`mobiletargeting:CreateImportJob`

Import endpoint definitions from Amazon S3 to create a segment.

- URI – `/apps/projectId/jobs/import`
- Method – POST
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId`

`mobiletargeting:GetImportJob`

Retrieve information about a specific import job for a project.

- URI – `/apps/projectId/jobs/import/jobId`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/jobs/import/jobId`

`mobiletargeting:GetImportJobs`

Retrieve information about all the import jobs for a project.

- URI – `/apps/projectId/jobs/import`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId`

Journeys

The following permissions are related to managing journeys in your Amazon Pinpoint account.

`mobiletargeting:CreateJourney`

Create a journey for a project.

- URI – `/apps/projectId/journeys`
- Method – POST
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/journeys`

`mobiletargeting:GetJourney`

Retrieve information about a specific journey.

- URI – `/apps/projectId/journeys/journeyId`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/journeys/journeyId`

`mobiletargeting:ListJourneys`

Retrieve information about all the journeys for a project.

- URI – `/apps/projectId/journeys`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/journeys`

`mobiletargeting:UpdateJourney`

Update the configuration and other settings for a specific journey.

- URI – `/apps/projectId/journeys/journeyId`
- Method – PUT

- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`accountId`*`:apps/`*`projectId`*`/journeys/`*`journeyId`*

`mobiletargeting:UpdateJourneyState`

Cancel an active journey.

- URI – `/apps/`*`projectId`*`/journeys/`*`journeyId`*`/state`
- Method – PUT
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`accountId`*`:apps/`*`projectId`*`/journeys/`*`journeyId`*`/state`

`mobiletargeting:DeleteJourney`

Delete a specific journey.

- URI – `/apps/`*`projectId`*`/journeys/`*`journeyId`*
- Method – DELETE
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`accountId`*`:apps/`*`projectId`*`/journeys/`*`journeyId`*

Message templates

The following permissions are related to creating and managing message templates for your Amazon Pinpoint account. A *message template* is a set of content and settings that you can define, save, and reuse in messages that you send for any of your Amazon Pinpoint projects.

`mobiletargeting:ListTemplates`

Retrieve information about all the message templates that are associated with your Amazon Pinpoint account.

- URI – `/templates`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`accountId`*`:templates`

`mobiletargeting:ListTemplateVersions`

Retrieve information about all the versions of a specific message template.

- URI – `/templates/`*`template-name`*`/`*`template-type`*`/versions`
- Method – GET
- Resource ARN – Not applicable

`mobiletargeting:UpdateTemplateActiveVersion`

Designate a specific version of a message template as the active version of the template.

- URI – `/templates/`*`template-name`*`/`*`template-type`*`/active-version`
- Method – GET
- Resource ARN – Not applicable

`mobiletargeting:GetEmailTemplate`

Retrieve information about a message template for messages that are sent through the email channel.

- URI – `/templates/`*`template-name`*`/email`

- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`accountId`*`:templates/`*`template-name`*`/EMAIL`

`mobiletargeting:CreateEmailTemplate`

Create a message template for messages that are sent through the email channel.

- URI – `/templates/`*`template-name`*`/email`
- Method – POST
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`accountId`*`:templates/`*`template-name`*`/EMAIL`

`mobiletargeting:UpdateEmailTemplate`

Update an existing message template for messages that are sent through the email channel.

- URI – `/templates/`*`template-name`*`/email`
- Method – PUT
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`accountId`*`:templates/`*`template-name`*`/EMAIL`

`mobiletargeting:DeleteEmailTemplate`

Delete a message template for messages that were sent through the email channel.

- URI – `/templates/`*`template-name`*`/email`
- Method – DELETE
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`accountId`*`:templates/`*`template-name`*`/EMAIL`

`mobiletargeting:GetPushTemplate`

Retrieve information about a message template for messages that are sent through a push notification channel.

- URI – `/templates/`*`template-name`*`/push`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`accountId`*`:templates/`*`template-name`*`/PUSH`

`mobiletargeting:CreatePushTemplate`

Create a message template for messages that are sent through a push notification channel.

- URI – `/templates/`*`template-name`*`/push`
- Method – POST
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`accountId`*`:templates/`*`template-name`*`/PUSH`

`mobiletargeting:UpdatePushTemplate`

Update an existing message template for messages that are sent through a push notification channel.

- URI – `/templates/template-name/push`
- Method – PUT
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:templates/template-name/PUSH`

`mobiletargeting:DeletePushTemplate`

Delete a message template for messages that were sent through a push notification channel.

- URI – `/templates/template-name/push`
- Method – DELETE
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:templates/template-name/PUSH`

`mobiletargeting:GetSmsTemplate`

Retrieve information about a message template for messages that are sent through the SMS channel.

- URI – `/templates/template-name/sms`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:templates/template-name/SMS`

`mobiletargeting:CreateSmsTemplate`

Create a message template for messages that are sent through the SMS channel.

- URI – `/templates/template-name/sms`
- Method – POST
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:templates/template-name/SMS`

`mobiletargeting:UpdateSmsTemplate`

Update an existing message template for messages that are sent through the SMS channel.

- URI – `/templates/template-name/sms`
- Method – PUT
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:templates/template-name/SMS`

`mobiletargeting:DeleteSmsTemplate`

Delete a message template for messages that were sent through the SMS channel.

- URI – `/templates/template-name/sms`
- Method – DELETE
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:templates/template-name/SMS`

`mobiletargeting:GetVoiceTemplate`

Retrieve information about a message template for messages that are sent through the voice channel.

- URI – `/templates/template-name/voice`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:templates/template-name/VOICE`

`mobiletargeting:CreateVoiceTemplate`

Create a message template for messages that are sent through the voice channel.

- URI – `/templates/template-name/voice`
- Method – POST
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:templates/template-name/VOICE`

`mobiletargeting:UpdateVoiceTemplate`

Update an existing message template for messages that are sent through the voice channel.

- URI – `/templates/template-name/voice`
- Method – PUT
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:templates/template-name/VOICE`

`mobiletargeting:DeleteVoiceTemplate`

Delete a message template for messages that were sent through the voice channel.

- URI – `/templates/template-name/voice`
- Method – DELETE
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:templates/template-name/VOICE`

Messages

The following permissions are related to sending 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 a message or push notification to specific endpoints.

- URI – `/apps/projectId/messages`
- Method – POST
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/messages`

`mobiletargeting:SendUsersMessages`

Send a message or push notification to all the endpoints that are associated with a specific user ID.

- URI – `/apps/projectId/users-messages`
- Method – POST
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/messages`

Phone number validation

The following permissions are related to using the phone number validation service in Amazon Pinpoint.

`mobiletargeting:PhoneNumberValidate`

Retrieve information about a phone number.

- URI – `/phone/number/validate`
- Method – POST
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`accountId`*`: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, an Amazon Pinpoint application is the same as an Amazon Pinpoint project.

`mobiletargeting:CreateApp`

Create an Amazon Pinpoint project.

- URI – `/apps`
- Method – POST
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`accountId`*`:apps`

`mobiletargeting:DeleteApp`

Delete an Amazon Pinpoint project.

- URI – `/apps/`*`projectId`*
- Method – DELETE
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`accountId`*`:apps/`*`projectId`*

`mobiletargeting:GetApp`

Retrieve information about an Amazon Pinpoint project.

- URI – `/apps/`*`projectId`*
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`accountId`*`:apps/`*`projectId`*

`mobiletargeting:GetApps`

Retrieve information about all the projects that are associated with your Amazon Pinpoint account.

- URI – `/apps`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`accountId`*`:apps`

`mobiletargeting:GetApplicationSettings`

Retrieve the default settings for an Amazon Pinpoint project.

- URI – `/apps/`*`projectId`*`/settings`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`accountId`*`:apps/`*`projectId`*

`mobiletargeting:UpdateApplicationSettings`

Update the default settings for an Amazon Pinpoint project.

- URI – `/apps/`*`projectId`*`/settings`
- Method – PUT
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`accountId`*`:apps/`*`projectId`*

Recommender models

The following permissions are related to managing Amazon Pinpoint configurations for retrieving and processing recommendation data from recommender models. A *recommender model* is a type of machine learning model that predicts and generates personalized recommendations by finding patterns in data.

`mobiletargeting:CreateRecommenderConfiguration`

Create an Amazon Pinpoint configuration for a recommender model.

- URI – `/recommenders`
- Method – POST
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:recommenders`

`mobiletargeting:GetRecommenderConfigurations`

Retrieve information about all the recommender model configurations that are associated with your Amazon Pinpoint account.

- URI – `/recommenders`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:recommenders`

`mobiletargeting:GetRecommenderConfiguration`

Retrieve information about an individual Amazon Pinpoint configuration for a recommender model.

- URI – `/recommenders/recommenderId`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:recommenders/recommenderId`

`mobiletargeting:UpdateRecommenderConfiguration`

Update an Amazon Pinpoint configuration for a recommender model.

- URI – `/recommenders/recommenderId`
- Method – PUT
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:recommenders/recommenderId`

`mobiletargeting:DeleteRecommenderConfiguration`

Delete an Amazon Pinpoint configuration for a recommender model.

- URI – `/recommenders/recommenderId`
- Method – DELETE
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:recommenders/recommenderId`

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 Amazon Pinpoint, allow the `mobiletargeting:CreateImportJob` action.

- URI – `/apps/projectId/segments`
- Method – POST
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId`

`mobiletargeting:DeleteSegment`

Delete a segment.

- URI – `/apps/projectId/segments/segmentId`
- Method – DELETE
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/segments/segmentId`

`mobiletargeting:GetSegment`

Retrieve information about a specific segment.

- URI – `/apps/projectId/segments/segmentId`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/segments/segmentId`

`mobiletargeting:GetSegmentExportJobs`

Retrieve information about jobs that export endpoint definitions for a segment.

- URI – `/apps/projectId/segments/segmentId/jobs/export`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/segments/segmentId/jobs/export`

`mobiletargeting:GetSegments`

Retrieve information about all the segments for a project.

- URI – `/apps/projectId/segments`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId`

`mobiletargeting:GetSegmentImportJobs`

Retrieve information about jobs that create segments by importing endpoint definitions from Amazon S3.

- URI – `/apps/projectId/segments/segmentId/jobs/import`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/segments/segmentId`

`mobiletargeting:GetSegmentVersion`

Retrieve information about a specific segment version.

- URI – `/apps/projectId/segments/segmentId/versions/versionId`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/segments/segmentId`

`mobiletargeting:GetSegmentVersions`

Retrieve information about the current and prior versions of a segment.

- URI – `/apps/projectId/segments/segmentId/versions`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/segments/segmentId`

`mobiletargeting:UpdateSegment`

Update a specific segment.

- URI – `/apps/projectId/segments/segmentId`
- Method – PUT
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/segments/segmentId`

Tags

The following permissions are related to viewing and managing tags for Amazon Pinpoint resources.

`mobiletargeting:ListTagsForResource`

Retrieve information about the tags that are associated with a project, campaign, message template, or segment.

- URI – `/tags/resource-arn`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:*`

`mobiletargeting:TagResource`

Add one or more tags to a project, campaign, message template, or segment.

- URI – `/tags/resource-arn`
- Method – POST
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:*`

`mobiletargeting:UntagResource`

Remove one or more tags from a project, campaign, message template, or segment.

- URI – `/tags/resource-arn`
- Method – DELETE
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:*`

Users

The following permissions are related to managing users. In Amazon Pinpoint, *users* correspond to individuals who receive messages from you. A single user might be associated with more than one endpoint.

`mobiletargeting:DeleteUserEndpoints`

Delete all the endpoints that are associated with a user ID.

- URI – `/apps/projectId/users/userId`
- Method – DELETE

- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/users/userId`

`mobiletargeting:GetUserEndpoints`

Retrieve information about all the endpoints that are associated with a user ID.

- URI – `/apps/projectId/users/userId`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/users/userId`

Amazon Pinpoint SMS and voice API actions

This section identifies actions for features that are available from the SMS and Voice API. This is a supplemental API that provides advanced options for using and managing the SMS and voice channels in Amazon Pinpoint. To learn more about this 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 configuration set for sending voice messages.

- URI – /sms-voice/configuration-sets/*ConfigurationSetName*
- Method – DELETE
- Resource ARN – Not available. Use `*`.

`sms-voice:GetConfigurationSetEventDestinations`

Retrieve 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 `*`.

Amazon Pinpoint identity-based policy examples

By default, IAM users and roles don't have permission to create or modify Amazon Pinpoint resources. They also can't perform tasks using the AWSManagement Console, AWS CLI, or an AWS API. An IAM administrator must create IAM policies that grant users and roles permission to perform specific API operations on the resources that they need. The administrator must then attach those policies to the IAM users or groups that require those permissions.

To learn how to create an IAM identity-based policy using these example JSON policy documents, see Creating policies on the JSON tab in the *IAM User Guide*.

Topics

Policy best practices

Identity-based policies are very powerful. They determine whether someone can create, access, or delete Amazon Pinpoint resources in your account. These actions can incur costs for your AWSaccount. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started using AWS managed policies** – To start using Amazon Pinpoint quickly, use AWS managed policies to give your employees the permissions they need. These policies are already available in your account and are maintained and updated by AWS. For more information, see Get started using permissions with AWS managed policies in the *IAM User Guide*.

- **Grant least privilege** – When you create custom policies, grant only the permissions required to perform a task. Start with a minimum set of permissions and grant additional permissions as necessary. Doing so is more secure than starting with permissions that are too lenient and then trying to tighten them later. For more information, see Grant least privilege in the *IAM User Guide*.
- **Enable MFA for sensitive operations** – For extra security, require IAM users to use multi-factor authentication (MFA) to access sensitive resources or API operations. For more information, see Using multi-factor authentication (MFA) in AWS in the *IAM User Guide*.
- **Use policy conditions for extra security** – To the extent that it's practical, define the conditions under which your identity-based policies allow access to a resource. For example, you can write conditions to specify a range of allowable IP addresses that a request must come from. You can also write conditions to allow requests only within a specified date or time range, or to require the use of SSL or MFA. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.

Using the Amazon Pinpoint console

To access the Amazon Pinpoint console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the Amazon Pinpoint resources in your AWS account. If you create an identity-based policy that applies permissions that are more restrictive than the minimum required permissions, the console won't function as intended for entities (IAM users or roles) with that policy. To ensure that those entities can use the Amazon Pinpoint console, attach a policy to the entities. For more information, see Adding permissions to a user in the *IAM User Guide*.

The following example policy provides read-only access to the Amazon Pinpoint console in a specific AWS Region. It includes read-only access to other services that the Amazon Pinpoint console depends on, such as Amazon Simple Email Service (Amazon SES), IAM, and Amazon Kinesis.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "UseConsole",
            "Effect": "Allow",
            "Action": [
                "mobiletargeting:Get*",
                "mobiletargeting:List*"
             ],
            "Resource": "arn:aws:mobiletargeting:region:accountId:*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "firehose:ListDeliveryStreams",
                "iam:ListRoles",
                "kinesis:ListStreams",
                "s3:List*",
                "ses:Describe*",
                "ses:Get*",
                "ses:List*",
                "sns:ListTopics"
            ],
            "Resource": "*"
        }
    ]
}
```

In the preceding policy example, replace *region* with the name of an AWS Region, and replace *accountId* with your AWS account ID.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that they're trying to perform.

Example: Accessing a single Amazon Pinpoint project

You can also create read-only policies that provide access to only specific projects. The following example policy lets users sign in to the console and view a list of projects. It also lets users view information about related resources for other AWS services that the Amazon Pinpoint console depends on, such as Amazon SES, IAM, and Amazon Kinesis. However, the policy lets users view additional information about only the project that's specified in the policy. You can modify this policy to allow access to additional projects or AWS Regions.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewProject",
            "Effect": "Allow",
            "Action": "mobiletargeting:GetApps",
            "Resource": "arn:aws:mobiletargeting:region:accountId:*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "mobiletargeting:Get*",
                "mobiletargeting:List*"
            ],
            "Resource": [
                "arn:aws:mobiletargeting:region:accountId:apps/projectId",
                "arn:aws:mobiletargeting:region:accountId:apps/projectId/*",
                "arn:aws:mobiletargeting:region:accountId:reports"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ses:Get*",
                "kinesis:ListStreams",
                "firehose:ListDeliveryStreams",
                "iam:ListRoles",
                "ses:List*",
                "sns:ListTopics",
                "ses:Describe*",
                "s3:List*"
            ],
            "Resource": "*"
        }
    ]
}
```

In the preceding example, replace *region* with the name of an AWS Region, replace *accountId* with your AWS account ID, and replace *projectId* with the ID of the Amazon Pinpoint project that you want to provide access to.

Similarly, you can create policies that grant an IAM user in your AWS account with limited write access to one of your Amazon Pinpoint projects, for example the project that has the `810c7aab86d42fb2b56c8c966example` project ID. In this case, you want to allow the user to view, add, and update project components, such as segments and campaigns, but not delete any components.

In addition to granting permissions for `mobiletargeting:Get` and `mobiletargeting:List` actions, create a policy that grants permissions for the following actions: `mobiletargeting:Create`;

`mobiletargeting:Update`; and `mobiletargeting:Put`. These are the additional permissions required to create and manage most project components. For example:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "LimitedWriteProject",
            "Effect": "Allow",
            "Action": "mobiletargeting:GetApps",
            "Resource": "arn:aws:mobiletargeting:region:accountId:*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "mobiletargeting:Get*",
                "mobiletargeting:List*",
                "mobiletargeting:Create*",
                "mobiletargeting:Update*",
                "mobiletargeting:Put*"
            ],
            "Resource": [

 "arn:aws:mobiletargeting:region:accountId:apps/810c7aab86d42fb2b56c8c966example",

 "arn:aws:mobiletargeting:region:accountId:apps/810c7aab86d42fb2b56c8c966example/*",
                "arn:aws:mobiletargeting:region:accountId:reports"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ses:Get*",
                "kinesis:ListStreams",
                "firehose:ListDeliveryStreams",
                "iam:ListRoles",
                "ses:List*",
                "sns:ListTopics",
                "ses:Describe*",
                "s3:List*"
            ],
            "Resource": "*"
        }
    ]
}
```

Example: Viewing Amazon Pinpoint resources based on tags

You can use conditions in an identity-based policy to control access to Amazon Pinpoint resources based on tags. This example policy shows how you might create this kind of policy to allow viewing Amazon Pinpoint resources. However, permission is granted only if the `Owner` resource tag has the value of that user's user name. This policy also grants the permissions necessary to complete this action on the console.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ListResources",
            "Effect": "Allow",
            "Action": [
                "mobiletargeting:Get*",
                "mobiletargeting:List*"
            ],
            "Resource": "*"
        },
        {
            "Sid": "ViewResourceIfOwner",
            "Effect": "Allow",
            "Action": [
                "mobiletargeting:Get*",
                "mobiletargeting:List*"
            ],
            "Resource": "arn:aws:mobiletargeting:*:*:*",
            "Condition": {
                "StringEquals": {"aws:ResourceTag/Owner": "${aws:username}"}
            }
        }
    ]
}
```

You can attach this type of policy to the IAM users in your account. If a user named `richard-roe` attempts to view an Amazon Pinpoint resource, the resource must be tagged `Owner=richard-roe` or `owner=richard-roe`. Otherwise he is denied access. The condition tag key `Owner` matches both `Owner` and `owner` because condition key names are not case-sensitive. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.

Example: Allowing users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

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

Examples: Providing access to Amazon Pinpoint API actions

This section provides example policies that allow access to features that are available from the Amazon Pinpoint API, which is the primary API for Amazon Pinpoint. To learn more about this API, see the Amazon Pinpoint API Reference.

Read-only access

The following example policy allows read-only access to all the resources in your Amazon Pinpoint account in a specific AWS Region.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewAllResources",
            "Effect": "Allow",
            "Action": [
                "mobiletargeting:Get*",
                "mobiletargeting:List*"
            ],
            "Resource": "arn:aws:mobiletargeting:region:accountId:*"
        }
    ]
}
```

In the preceding example, replace *region* with the name of an AWS Region, and replace *accountId* with your AWS account ID.

Administrator access

The following example policy allows full access to all Amazon Pinpoint actions and resources in your Amazon Pinpoint account in all AWS Regions:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "FullAccess",
            "Effect": "Allow",
            "Action": [
                "mobiletargeting:*"
            ],
            "Resource": "arn:aws:mobiletargeting:*:accountId:*"
        }
    ]
}
```

In the preceding example, replace *accountId* with your AWS account ID.

Examples: Providing access to Amazon Pinpoint SMS and voice API actions

This section provides example policies that allow access to features that are available from the SMS and Voice API. This is a supplemental API that provides advanced options for using and managing the SMS and voice channels in Amazon Pinpoint. To learn more about this API, see the Amazon Pinpoint SMS and voice API reference.

Read-only access

The following example policy allows read-only access to all SMS and Voice API actions and resources in your AWS account in all AWS Regions:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewAllResources",
            "Effect": "Allow",
            "Action": [
                "sms-voice:Get*",
                "sms-voice:List*"
            ],

            "Resource": "*"
        }
    ]
}
```

Administrator access

The following example policy allows full access to all SMS and Voice API actions and resources in your AWS account in all AWS Regions:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "FullAccess",
            "Effect": "Allow",
            "Action": [
                "sms-voice:*"
            ],

            "Resource": "*"
        }
    ]
}
```

Example: Restricting Amazon Pinpoint project access to specific IP addresses

The following example policy grants permissions to any user to perform any Amazon Pinpoint action on a specified project (*`projectId`*). However, the request must originate from the range of IP addresses that are specified in the condition.

The condition in this statement identifies the `54.240.143.*` range of allowed Internet Protocol version 4 (IPv4) addresses, with one exception: `54.240.143.188`. The `Condition` block uses the `IpAddress` and `NotIpAddress` conditions and the `aws:SourceIp` condition key, which is an AWS-wide condition key. For more information about these condition keys, see Specifying conditions in a policy *IAM User Guide*. The `aws:SourceIp` IPv4 values use standard CIDR notation. For more information, see IP address condition operators in the *IAM User Guide*.

```
{
  "Version": "2012-10-17",
  "Id": "AMZPinpointPolicyId1",
  "Statement": [
    {
      "Sid": "IPAllow",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "mobiletargeting:*",
      "Resource": [
                "arn:aws:mobiletargeting:*:*:apps/projectId",
                "arn:aws:mobiletargeting:*:*:apps/projectId/*"
                ],
      "Condition": {
         "IpAddress": {"aws:SourceIp": "54.240.143.0/24"},
         "NotIpAddress": {"aws:SourceIp": "54.240.143.188/32"}
      }
    }
  ]
}
```

Example: Restricting Amazon Pinpoint access based on tags

The following example policy grants permissions to perform any Amazon Pinpoint action on a specified project (*projectId*). However, permissions are granted only if the request originates from a user whose name is a value in the `Owner` resource tag for the project, as specified in the condition.

The `Condition` block uses the `StringEquals` condition and the `aws:ResourceTag/${TagKey}` condition key. For more information about conditions and condition keys, see Specifying conditions in a policy in the *IAM User Guide*.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ModifyResourceIfOwner",
            "Effect": "Allow",
            "Action": "mobiletargeting:*",
            "Resource": [
                "arn:aws:mobiletargeting:*:*:apps/projectId",
                "arn:aws:mobiletargeting:*:*:apps/projectId/*"
                ],
            "Condition": {
                "StringEquals": {"aws:ResourceTag/Owner": "${aws:username}"}
            }
        }
    ]
}
```

IAM roles for common Amazon Pinpoint tasks

An IAM role is an AWS Identity and Access Management (IAM) identity that you can create in your AWS account and grant specific permissions. An IAM role is similar to an IAM user, in that it's an AWS identity with permission policies that determine what the identity can and cannot do in AWS. However, instead of being uniquely associated with one person, a role is intended to be assumable by anyone who needs it.

Also, a role doesn't have standard long-term credentials such as a password or access keys associated with it. Instead, it provides temporary security credentials for a session. You can use IAM roles to delegate access to users, apps, applications, or services that don't normally have access to your AWS resources.

For these reasons, you can use IAM roles to integrate Amazon Pinpoint with certain AWS services and resources for your account. For example, you might want to allow Amazon Pinpoint to access endpoint definitions that you store in an Amazon Simple Storage Service (Amazon S3) bucket and want to use for segments. Or you might want to allow Amazon Pinpoint to stream event data to an Amazon Kinesis stream for your account. Similarly, you might want to use IAM roles to allow web or mobile apps to register endpoints or report usage data for Amazon Pinpoint projects, without embedding AWS keys in the apps (where they can be difficult to rotate and users can potentially extract them).

For these scenarios, you can delegate access to Amazon Pinpoint by using IAM roles. This section explains and provides examples of common Amazon Pinpoint tasks that use IAM roles to work with other AWS services. For information about using IAM roles with web and mobile apps more specifically, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.

Topics

IAM role for importing endpoints or segments

With Amazon Pinpoint, you can define a user segment by importing endpoint definitions from an Amazon Simple Storage Service (Amazon S3) bucket in your AWS account. Before you import, you must delegate the required permissions to Amazon Pinpoint. To do this, you 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.

After you create the role, you can use Amazon Pinpoint to import segments from an Amazon S3 bucket. For information about creating the bucket, creating endpoint files, and importing a segment by using the console, see Importing segments in the *Amazon Pinpoint User Guide*. For an example of how to import a segment programmatically by using the AWS SDK for Java, see Importing segments (p. 153) in this guide.

Attaching the 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": [
    {
      "Sid": "AllowUserToImportEndpointDefinitions",
      "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 haven't installed the AWS CLI, see Installing the AWS CLI 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. At the command line, 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.

   After you run this command, you see output that's similar to the following 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 topic contains procedures for completing both of these steps. These procedures assume that you've already created an Amazon S3 bucket, and a folder in that bucket, for storing exported segments. For 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 Command Line Interface (AWS CLI). For information about setting up the AWS CLI, see Installing the AWS CLI 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 grants only 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 *`Exports`* with the name of the folder itself.

When you finish, save the file as `s3policy.json`.

2. By using the AWS CLI, navigate to the directory where the `s3policy.json` file is located. Then enter 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 Adding and removing IAM identity permissions 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. By using the AWS CLI, navigate to the directory where the `trustpolicy.json` file is located. Then enter 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, enter the following command to attach the policy that you created in the previous section to the role that 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 retrieving recommendations from Amazon Personalize

You can configure Amazon Pinpoint to retrieve recommendation data from an Amazon Personalize solution that's been deployed as an Amazon Personalize campaign. You can use this data to send personalized recommendations to message recipients based on each recipient's attributes and behavior. To learn more, see Machine learning models in the *Amazon Pinpoint User Guide*.

Before you can retrieve recommendation data from an Amazon Personalize campaign, you have to create an AWS Identity and Access Management (IAM) role that allows Amazon Pinpoint to retrieve the data from the campaign. Amazon Pinpoint can create this role for you automatically when you use the console to set up a recommender model in Amazon Pinpoint. Or, you can create this role manually.

To create the role manually, use the IAM API to complete the following steps:

1. Create an IAM policy that allows an entity (in this case, Amazon Pinpoint) to retrieve recommendation data from an Amazon Personalize campaign.
2. Create an IAM role and attach the IAM policy to it.

This topic explains how to complete these steps by using the AWS Command Line Interface (AWS CLI). It assumes that you've already created the Amazon Personalize solution and deployed it as an Amazon Personalize campaign. For information about creating and deploying a campaign, see Creating a campaign in the *Amazon Personalize Developer Guide*.

This topic also assumes that you've already installed and configured the AWS CLI. For information about setting up the AWS CLI, see Installing the AWS CLI in the *AWS Command Line Interface User Guide*.

Step 1: Create the IAM policy

An IAM policy defines permissions for an entity, such as an identity or resource. To create a role that allows Amazon Pinpoint to retrieve recommendation data from an Amazon Personalize campaign, you first have to create an IAM policy for the role. This policy needs to allow Amazon Pinpoint to:

- Retrieve configuration information for the solution that's deployed by the campaign (`DescribeSolution`).
- Check the status of the campaign (`DescribeCampaign`).
- Retrieve recommendation data from the campaign (`GetRecommendations`).

In the following procedure, the example policy allows this access for a particular Amazon Personalize solution that was deployed by a particular Amazon Personalize campaign.

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": "RetrieveRecommendationsOneCampaign",
            "Effect": "Allow",
            "Action": [
                "personalize:DescribeSolution",
                "personalize:DescribeCampaign",
                "personalize:GetRecommendations"
```

```
             ],
             "Resource": [
                "arn:aws:personalize:region:accountId:solution/solutionId",
                "arn:aws:personalize:region:accountId:campaign/campaignId"
                ]
         }
     ]
}
```

In the preceding example, replace the *italicized* text with your information:

- *region* – The name of the AWS Region that hosts the Amazon Personalize solution and campaign.
- *accountId* – Your AWSaccount ID.
- *solutionId* – The unique resource ID for the Amazon Personalize solution that's deployed by the campaign.
- *campaignId* – The unique resource ID for the Amazon Personalize campaign to retrieve recommendation data from.

2. When you finish, save the file as `RetrieveRecommendationsPolicy.json`.
3. By using the command line interface, navigate to the directory where you saved the `RetrieveRecommendationsPolicy.json` file.
4. Enter the following command to create a policy and name it `RetrieveRecommendationsPolicy`. To use a different name, change *`RetrieveRecommendationsPolicy`* to the name that you want.

```
aws iam create-policy --policy-name RetrieveRecommendationsPolicy --policy-document
 file://RetrieveRecommendationsPolicy.json
```

If the policy was created successfully, you see output similar to the following:

```
{
    "Policy": {
        "PolicyName": "RetrieveRecommendationsPolicy",
        "PolicyId": "ANPAJ2YJQRJCG3EXAMPLE",
        "Arn": "arn:aws:iam::123456789012:policy/RetrieveRecommendationsPolicy",
        "Path": "/",
        "DefaultVersionId": "v1",
        "AttachmentCount": 0,
        "PermissionBoundaryUsageCount": 0,
        "IsAttachable": true,
        "CreateDate": "2020-03-04T22:23:15Z",
        "UpdateDate": "2020-03-04T22:23:15Z"
    }
}
```

Note
If you receive a message that your account isn't authorized to perform the `CreatePolicy` operation, you need to attach a policy to your user account that lets you create new IAM policies and roles for your account. For more information, see Adding and removing IAM identity permissions in the *IAM User Guide*.

5. Copy the Amazon Resource Name (ARN) of the policy (`arn:aws:iam::123456789012:policy/RetrieveRecommendationsPolicy` in the preceding example). In the next section, you'll need this ARN to create the IAM role.

Step 2: Create the IAM role

After you create the IAM policy, you can create an IAM role and attach the policy to it.

Each IAM role contains a *trust policy*, which is 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. Then, you attach the policy to the role.

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"
        }
    ]
}
```

2. Save the file as `RecommendationsTrustPolicy.json`.
3. By using the command line interface, navigate to the directory where you saved the `RecommendationsTrustPolicy.json` file.
4. Enter the following command to create a new role and name it `PinpointRoleforPersonalize`. To use a different name, change *`PinpointRoleforPersonalize`* to the name that you want.

```
aws iam create-role --role-name PinpointRoleforPersonalize --assume-role-policy-
document file://RecommendationsTrustPolicy.json
```

If the command runs successfully, you see output similar to the following:

```
{
    "Role": {
        "Path": "/",
        "RoleName": "PinpointRoleforPersonalize",
        "RoleId": "AKIAIOSFODNN7EXAMPLE",
        "Arn": "arn:aws:iam::123456789012:role/PinpointRoleforPersonalize",
        "CreateDate": "2020-03-04T22:29:45Z",
        "AssumeRolePolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Principal": {
                        "Service": "pinpoint.amazonaws.com"
                    },
                    "Action": "sts:AssumeRole"
                }
            ]
        }

    }
}
```

5. Enter the following command to attach the policy that you created in the previous section to the role that you just created:

```
aws iam attach-role-policy --policy-arn arn:aws:iam::123456789012:policy/
RetrieveRecommendationsPolicy --role-name PinpointRoleforPersonalize
```

In the preceding command, replace *arn:aws:iam::123456789012:policy/RetrieveRecommendationsPolicy* with the ARN of the policy that you created in the previous section. Also, replace *PinpointRoleforPersonalize* with the name of the role that you specified in step 4, if you specified a different name for the role.

IAM role for streaming events to Kinesis

Amazon Pinpoint can automatically send app usage data, or *event data*, from your app to an Amazon Kinesis data 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 event data to your stream.
- A trust policy that allows Amazon Pinpoint to assume the role.

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. 206) in this guide.

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). To learn how to create the role by using the Amazon Pinpoint console, see Streaming Amazon Pinpoint events to Kinesis in the *Amazon Pinpoint User Guide*.

If you haven't installed the AWS CLI, see Installing the AWS CLI 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 PinpointEventStreamRole --assume-role-policy-document
    file://PinpointEventStreamTrustPolicy.json
   ```

   Following the `file://` prefix, specify the path to the JSON file that contains the trust policy.

   After 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. 363) section of this topic.
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.

IAM role for streaming email events to Kinesis Data Firehose

In the Email API, you can create *configuration sets* that specify how to handle certain email events. For example, you can create a configuration set that sends delivery notifications to a specific *event destination*, such as an Amazon SNS topic or a Kinesis Data Firehose delivery stream. When you send email through the Email API using that configuration set, Amazon Pinpoint sends information about email-related events to the event destination that you specified in the configuration set.

The Email API can deliver information about the following types of email events to the event destinations that you specify:

- **Sends** – The call to Amazon Pinpoint was successful, and Amazon Pinpoint attempted to deliver the email.
- **Deliveries** – Amazon Pinpoint successfully delivered the email to the recipient's mail server.
- **Rejections** – Amazon Pinpoint accepted the email, determined that it contained malware, and rejected it. Amazon Pinpoint didn't attempt to deliver the email to the recipient's mail server.
- **Rendering Failures** – The email wasn't sent because of a template rendering issue. This event type only occurs when you send an email that includes substitution tags. This event type can occur when substitution values are missing. It can also occur when there's a mismatch between the substitution tags that you used in the email and the substitution data that you provided.

  Note
  If you use substitution tags in the emails that you send by using the Email API, you should always create a configuration set that records Rendering Failure events.
- **Bounces** – The recipient's mail server permanently rejected the email.
- **Complaints** – The email was successfully delivered to the recipient, but the recipient used the "Report Spam" (or equivalent) feature of their email client to report the message.
- **Opens** – The recipient received the message and opened it in their email client.
- **Clicks** – The recipient clicked one or more links that were contained in the email.

  Note
  Every time a recipient opens or clicks an email, Amazon Pinpoint generates unique open or click events, respectively. In other words, if a specific recipient opens a message five times, Amazon Pinpoint reports five separate Open events.

If you want to send data about these events to a Kinesis Data Firehose stream, you have to create an IAM role that has the appropriate permissions. The role must use the following policies:

- A trust policy that allows Amazon Pinpoint to assume the role.
- A permissions policy that allows the Email API to send email delivery and response records to your stream.

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. 206).

Trust policy

To allow the Email API 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": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "ses.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "sts:ExternalId": "accountId"
        }
      }
    }
  ]
}
```

In the example above, replace *accountId* with the ID of your AWS account.

Permissions policy

To allow the Email API to send email event data to a Kinesis Data Firehose delivery stream, attach the following permissions policy to a role.

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:accountId:deliverystream/deliveryStreamName"
     ]
    }
}
```

In the example above, replace *region* with the name of the AWS Region that you created the delivery stream in. Replace *accountId* with the ID of your AWS account. Finally, replace *deliveryStreamName* with the name of the delivery stream.

Creating the IAM role (AWS CLI)

Complete the following steps to create the IAM role by using the AWS Command Line Interface (AWS CLI). For information about installing and configuring the AWS CLI, see Installing the AWS CLI 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 then save the file locally. You can copy the trust policy (p. 366) that's provided earlier 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
```

   In the preceding example, replace *PinpointEventStreamTrustPolicy.json* with the full path to the file that contains the trust policy.

   After you run this command, the AWS CLI returns the following output:

```
{
    "Role": {
        "AssumeRolePolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Action": "sts:AssumeRole",
                    "Effect": "Allow",
                    "Principal": {
                        "Service": "ses.amazonaws.com"
                    }
                }
            ]
        },
        "RoleId": "AKIAIOSFODNN7EXAMPLE",
        "CreateDate": "2019-04-10T14:20:42.314Z",
        "RoleName": "PinpointEventStreamRole",
        "Path": "/",
        "Arn": "arn:aws:iam::111122223333:role/PinpointEventStreamRole"
    }
}
```

3. Create a JSON file that contains the permissions policy for your role, and then save the file locally. You can copy the permissions policy (p. 366) that's provided earlier in this topic.
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
```

   In the preceding example, replace *PinpointEventStreamPermissionsPolicy.json* with the full path to the file that contains the permissions policy.

Troubleshooting Amazon Pinpoint identity and access management

Use the following information to diagnose and fix common issues that you might encounter when working with Amazon Pinpoint and IAM.

Topics

I'm not authorized to perform an action in Amazon Pinpoint

If the AWSManagement Console tells you that you're not authorized to perform an action, then you must contact your administrator for assistance. Your administrator is the person who provided you with your user name and password.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a project but doesn't have `mobiletargeting:GetApp` permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 mobiletargeting:GetApp on resource: my-example-project
```

In this case, Mateo asks his administrator to update his policies to allow him to access the `my-example-project` resource using the `mobiletargeting:GetApp` action.

I'm not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password. Ask that person to update your policies to allow you to pass a role to Amazon Pinpoint.

Some AWS services allow you to pass an existing role to that service, instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in Amazon Pinpoint. However, the action requires the service to have permissions granted by a service role. Mary does not have permissions to pass the role to the service.

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

In this case, Mary asks her administrator to update her policies to allow her to perform the `iam:PassRole` action.

I want to view my access keys

After you create your IAM user access keys, you can view your access key ID at any time. However, you can't view your secret access key again. If you lose your secret key, you must create a new access key pair.

Access keys consist of two parts: an access key ID (for example, `AKIAIOSFODNN7EXAMPLE`) and a secret access key (for example, `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`). Like a user name and password, you must use both the access key ID and secret access key together to authenticate your requests. Manage your access keys as securely as you do your user name and password.

Important
Do not provide your access keys to a third party, even to help find your canonical user ID. By doing this, you might give someone permanent access to your account.

When you create an access key pair, you are prompted to save the access key ID and secret access key in a secure location. The secret access key is available only at the time you create it. If you lose your secret access key, you must add new access keys to your IAM user. You can have a maximum of two access keys. If you already have two, you must delete one key pair before creating a new one. To view instructions, see Managing access keys in the *IAM User Guide*.

I'm an administrator and want to allow others to access Amazon Pinpoint

To allow others to access Amazon Pinpoint, you must create an IAM entity (user or role) for the person or application that needs access. They will use the credentials for that entity to access AWS. You must then attach a policy to the entity that grants them the correct permissions in Amazon Pinpoint.

To get started right away, see Creating your first IAM delegated user and group in the *IAM User Guide*.

I want to allow people outside my AWS account to access my Amazon Pinpoint resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether Amazon Pinpoint supports these features, see How Amazon Pinpoint works with IAM (p. 322).
- To learn how to provide access to your resources across AWSaccounts that you own, see Providing access to an IAM user in another AWSaccount that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWSaccounts, see Providing access to AWSaccounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the *IAM User Guide*.

Logging and monitoring in Amazon Pinpoint

Logging and monitoring are an important part of maintaining the reliability, availability, and performance of your Amazon Pinpoint projects and other types of Amazon Pinpoint resources. You should log and collect monitoring data from all parts of your Amazon Pinpoint projects and resources to more easily debug a multipoint failure if one occurs. AWS provides several tools that can help you log and collect this data, and respond to potential incidents:

AWS CloudTrail

Amazon Pinpoint integrates with AWS CloudTrail, which is a service that provides a record of actions that were taken in Amazon Pinpoint by a user, a role, or another AWS service. This includes actions from the Amazon Pinpoint console and programmatic calls to Amazon Pinpoint API operations. By using the information collected by CloudTrail, you can determine which requests were made to Amazon Pinpoint. For each request, you can identify when it was made, the IP address from which it was made, who made it, and additional details. For more information, see Logging Amazon Pinpoint API calls with AWS CloudTrail (p. 285) in this guide.

Amazon CloudWatch

You can use Amazon CloudWatch to collect, view, and analyze several important metrics related to your Amazon Pinpoint account and projects. You can also use CloudWatch to create alarms that notify you if the value for a metric meets certain conditions and is within or exceeds a threshold that you define. If you create an alarm, CloudWatch sends a notification to an Amazon Simple Notification Service (Amazon SNS) topic that you specify. For more information, see Monitoring Amazon Pinpoint with amazon CloudWatch in the *Amazon Pinpoint User Guide*.

AWS Health Dashboards

By using AWS Health dashboards, you can check and monitor the status of your Amazon Pinpoint environment. To check the status of the Amazon Pinpoint service overall, use the AWS Service Health Dashboard. To check, monitor, and view historical data about any events or issues that might affect your AWS environment more specifically, use the AWS Personal Health Dashboard. To learn more about these dashboards, see the AWS Health User Guide.

AWSTrusted Advisor

AWSTrusted Advisor inspects your AWS environment and provides recommendations for opportunities to address security gaps, improve system availability and performance, and save money. All AWS customers have access to a core set of Trusted Advisor checks. Customers who have a Business or Enterprise support plan have access to additional Trusted Advisor checks.

Many of these checks can help you assess the security posture of your Amazon Pinpoint resources as part of your AWS account overall. For example, the core set of Trusted Advisor checks includes the following:

- Logging configurations for your AWS account, for each supported AWS Region.
- Access permissions for your Amazon Simple Storage Service (Amazon S3) buckets, which might contain files that you import into Amazon Pinpoint to build segments.
- Use of AWS Identity and Access Management (IAM) users, groups, and roles to control access to Amazon Pinpoint resources.
- IAM configurations and policy settings that might compromise the security of your AWS environment and Amazon Pinpoint resources.

For more information, see AWSTrusted Advisor in the *AWS Support User Guide*.

Compliance validation for Amazon Pinpoint

Third-party auditors assess the security and compliance of Amazon Pinpoint as part of multiple AWS compliance programs. These include AWS System and Organization Controls (SOC), FedRAMP, HIPAA, ISO/IEC 27001:2013 for security management controls, ISO/IEC 27017:2015 for cloud-specific controls, ISO/IEC 27018:2014 for personal data protection, ISO/IEC 9001:2015 for quality management systems, and others.

For a list of AWS services that are in scope for specific compliance programs, see AWS services in scope by compliance program. For general information, see AWS compliance programs.

You can download third-party audit reports by using AWS Artifact. For more information, see Downloading reports in AWS Artifact.

Your compliance responsibility when using Amazon Pinpoint is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security and compliance quick start guides – These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- Architecting for HIPAA security and compliance whitepaper – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.
- AWS compliance resources – This collection of workbooks and guides might apply to your industry and location.
- Evaluating resources with rules in the *AWS Config Developer Guide* – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

Amazon Pinpoint is an AWS HIPAA eligible service when customers use the proper communication channels. If you wish to use Amazon Pinpoint to run workloads containing Protected Health Information (PHI) as defined by HIPAA and associated legislation and regulations, you should use the email channel, push notification channel, or SMS channel to send messages that contain PHI. If you use the SMS channel to send messages that contain PHI, you should send those messages from a dedicated short code that you requested for your AWS account for the explicit purpose of sending messages that will or may contain PHI. The voice channel is not AWS HIPAA eligible; do not use the voice channel to send messages that contain PHI.

Resilience in Amazon Pinpoint

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS global infrastructure.

Infrastructure security in Amazon Pinpoint

As a managed service, Amazon Pinpoint is protected by the AWS global network security procedures that are described in the Amazon Web Services: Overview of security processes whitepaper.

You use published AWS API calls to access Amazon Pinpoint through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed using an access key ID and a secret access key that's associated with an AWS Identity and Access Management (IAM) principal for your AWS account. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

Although you can make these API calls from any network location, Amazon Pinpoint supports resource-based access policies. These policies can include restrictions based on source IP address. To learn more about this type of policy, see Managing access using policies (p. 320).

In addition, you can configure and use various AWS security features to control access to Amazon Pinpoint resources from any mobile or web apps that you integrate with Amazon Pinpoint. This includes restrictions on API calls for tasks such as adding endpoints, updating endpoint data, submitting event data, and reporting usage data.

To use these features, we recommend that you use the AWS Mobile SDKs or AWS Amplify JavaScript libraries to integrate mobile and web apps with Amazon Pinpoint. For Android or iOS apps, we recommend that you use the AWS Mobile SDK for Android or the AWS Mobile SDK for iOS, respectively. For JavaScript-based mobile or web apps, we recommend that you use the AWS Amplify JavaScript Library for the Web or the AWS Amplify JavaScript Library for React Native. To learn more about these resources, see Getting started with the AWS mobile SDKs, Getting started with the AWS Amplify library for the web, and Getting started with the AWS Amplify library for react native.

Configuration and vulnerability analysis in Amazon Pinpoint

As a managed service, Amazon Pinpoint is protected by the AWS global network security procedures that are described in the Amazon Web Services: Overview of security processes whitepaper. This means that AWS manages and performs basic security tasks and procedures to harden, patch, update, and otherwise maintain the underlying infrastructure for your Amazon Pinpoint account and resources. These procedures have been reviewed and certified by the appropriate third parties.

For more information, see the following resources:

- Compliance validation for Amazon Pinpoint (p. 370)
- Shared responsibility model
- Amazon Web Services: Overview of security processes (whitepaper)

Amazon Pinpoint quotas

The following sections list and describe the quotas, formerly referred to as *limits*, that apply to Amazon Pinpoint resources and operations. Some quotas can be increased, while others cannot. To determine whether you can request an increase for a quota, refer to the **Eligible for Increase** column or statement in each section.

Topics

General quotas

Your account can have up to 100 Amazon Pinpoint projects. This quota is not eligible for increase.

API request quotas

Amazon Pinpoint implements quotas that restrict the size and number of requests that you can make to the Amazon Pinpoint API from your AWS account. These quotas are not eligible for increase.

The maximum size of an invocation (request and response) payload is 7 MB, unless otherwise specified for a particular type of resource. To determine whether a resource has a different quota, see the appropriate section of this topic for that type of resource.

The maximum number of requests varies by quota type and API operation. Amazon Pinpoint implements two types of quotas for API requests:

- **Rate quotas** – Also referred to as *rate limits*, this type of quota defines the maximum number of requests that you can make per second for a particular operation. It controls the rate of requests that are sent or received.
- **Burst quotas** – Also referred to as *burst limits* or *burst capacity*, this type of quota defines the maximum number of requests that you can send at one time for a particular operation. It prevents usage spikes by ensuring that the supported number of requests is distributed evenly across a certain time period.

The following table lists the rate and burst quotas for the Amazon Pinpoint API.

Operation	Default rate quota (requests per second)	Default burst quota (number of requests)
CreateCampaign	25	25
CreateSegment	25	25
DeleteCampaign	25	25
DeleteEndpoint	1,000	1,000
DeleteSegment	25	25
GetEndpoint	7,000	7,000
PhoneNumberValidate	20	20
PutEvents	7,000	7,000
SendMessages	4,000	4,000
SendUsersMessages	6,000	6,000
UpdateCampaign	25	25
UpdateEndpoint	5,000	5,000
UpdateEndpointsBatch	5,000	5,000
UpdateSegment	25	25
All other operations	300	300

If you exceed one of these quotas, Amazon Pinpoint throttles the request—that is, it rejects an otherwise valid request and returns a `TooManyRequests` error. Throttling is based on the total number of requests that you make from your account for a specific operation in a specific AWSRegion. In addition, throttling decisions are calculated independently for each operation. For example, if Amazon Pinpoint throttles a request for the `SendMessages` operation, a concurrent request for the `UpdateEndpoint` operation can complete successfully.

Campaign quotas

The following quotas apply to the Campaigns resource of the Amazon Pinpoint API.

Resource	Default quota	Eligible for increase
Active campaigns	200 per account **Note** An *active campaign* is a campaign that hasn't completed or failed. Active campaigns have a status of `SCHEDULED`,	Yes (p. 384)

Resource	Default quota	Eligible for increase
	`EXECUTING`, or `PENDING_NEXT_RUN`.	
Maximum segment size	For imported segments: 100,000,000 per campaign For dynamic segments: unlimited	No
Event-based campaigns	Each project can include up to 25 campaigns that are sent when events occur. Campaigns that use event-based triggers have to use dynamic segments. They can't use imported segments. If you integrate your app with Amazon Pinpoint by using an AWS Mobile SDK, messages from event-based campaigns are sent only to customers whose apps are running AWS Mobile SDK for Android version 2.7.2 or later, or AWS Mobile SDK for iOS version 2.6.30 or later. If Amazon Pinpoint can't deliver a message from an event-based campaign within five minutes, it drops the message and doesn't attempt to redeliver it.	Yes (p. 384)

Email quotas

The quotas in the following sections apply to the email channel.

Email message quotas

Resource	Default quota	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	No

Resource	Default quota	Eligible for increase
	Pinpoint must be sent from a verified identity.	

Email sender and recipient quotas

Resource	Default quota	Eligible for increase
Sender address	All sending addresses or domains must be verified.	No
Recipient address	If your account is in the sandbox, all recipient email addresses or domains must be verified. If your account is out of the sandbox, you can send to any valid address.	Yes (p. 384)
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

Email sending quotas

The sending quota, sending rate, and sandbox limits are shared between the two services in the same Region. If you use Amazon SES in us-east-1, and you've been removed from the sandbox and had your sending quota/rate increased, then those changes all apply to your Pinpoint account in us-east-1.

Resource	Default quota	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. **Note** This quota is based on the number of recipients, as opposed to the number of	Yes (p. 384)

Resource	Default quota	Eligible for increase
	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. 384)

Endpoint quotas

The following quotas apply to the Endpoints resource of the Amazon Pinpoint API.

The maximum number of attributes supported per endpoint is 250, and the maximum endpoint size is 15 KB. This number of attributes might be limited, however, by the total size of an endpoint, which includes all attributes. If you run into any errors when adding attributes to your template, consider decreasing the amount of data in each attribute or decreasing the number of attributes.

Resource	Default quota	Eligible for increase
Attributes assigned to the `Attributes`, `Metrics`, and `UserAttributes` parameters collectively	250 for all attribute parameters per application and a maximum size of 15 KB for the endpoint.	Yes
Attributes assigned to the `Attributes` parameter	250 for all attribute parameters per application and a maximum size of 15 KB for the endpoint.	Yes
Attributes assigned to the `Metrics` parameter	250 for all attribute parameters per application and a maximum size of 15 KB for the endpoint.	Yes
Attributes assigned to the `UserAttributes` parameter	250 for all attribute parameters per application and a maximum size of 15 KB for the endpoint.	Yes
Attribute name length	50 characters	No
Attribute value length	100 characters	No
`EndpointBatchItem` objects in an `EndpointBatchRequest` payload	100 per payload. The payload size can't exceed 7 MB.	No

Resource	Default quota	Eligible for increase
Endpoints with the same user ID	15 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 quotas

The following quotas apply to importing endpoints into Amazon Pinpoint.

Resource	Default quota	Eligible for increase
Concurrent import jobs	10 per account	Yes (p. 384)
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. 384)

Event ingestion quotas

The following quotas apply to the ingestion of events using the AWS Mobile SDKs and the Events resource of the Amazon Pinpoint API.

Resource	Default quota	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. Any number that exceeds 100,000 is still registered, but won't be available in the Amazon Pinpoint analytics console.	No
Maximum number of characters per attribute key	50	No
Maximum number of characters per attribute value	200. If the number of characters exceeds 200 the event is dropped.	No
Maximum number of custom metric keys	500 per app	No

Resource	Default quota	Eligible for increase
Maximum number of 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

Journey quotas

The following quotas apply to journeys.

Resource	Default quota	Eligible for increase
Maximum number of active journeys	50 per account	Yes (p. 384)
Maximum number of active EventTriggeredJourneys	20 per account	Yes (p. 384)
Maximum number of journey activities	40 per journey	Yes (p. 384)
Maximum segment size	For imported segments: 100,000,000 per journey. For dynamic segments: unlimited	No

Machine learning quotas

The following quotas apply to Amazon Pinpoint configurations for retrieving and processing data from machine learning (ML) models.

Resource	Default quota	Eligible for increase
Maximum number of model configurations	1 per message template 100 per account	No
Maximum number of recommendations	5 per endpoint or user	No
Maximum number of recommended attributes per endpoint or user	1, if the attribute values aren't processed by an AWS Lambda function	No

Resource	Default quota	Eligible for increase
	10, if the attribute values are processed by an AWS Lambda function	
Maximum length of a recommended attribute name	50 characters for an attribute name 25 characters for an attribute display name (the name that appears in the **Attribute finder** on the console)	No
Maximum length of a recommended attribute value that's retrieved from Amazon Personalize	100 characters	No
Maximum size of an invocation payload (request and response) for a Lambda function	6 MB	No
Maximum amount of time to wait for a Lambda function to process data	15 seconds	No
Maximum number of attempts to invoke a Lambda function	3 attempts	No

Depending on how you configure Amazon Pinpoint to use an ML model, additional quotas may apply. To learn about Amazon Personalize quotas, see Quotas in the *Amazon Personalize Developer Guide*. To learn about AWS Lambda quotas, see Quotas in the *AWS Lambda Developer Guide*.

Message template quotas

The following quotas apply to message templates for your Amazon Pinpoint account.

Resource	Default quota	Eligible for increase
Maximum number of message templates	10,000 per account	Yes (p. 384)
Maximum number of versions	5,000 per template	No
Maximum number of characters in an email template	500,000 characters	No
Maximum number of characters in the default template parts of a push notification template	2,000 characters	No
Maximum number of characters in ADM-specific template parts of a push notification template	4,000 characters	No

Resource	Default quota	Eligible for increase
Maximum number of characters in APNs-specific template parts of a push notification template	2,000 characters	No
Maximum number of characters in Baidu-specific template parts of a push notification template	4,000 characters	No
Maximum number of characters in FCM-specific template parts of a push notification template	4,000 characters	No
Maximum number of characters in an SMS template	1,600 characters	No
Maximum number of characters in a voice template	10,000 characters	No

Push notification quotas

The following quotas apply to messages that Amazon Pinpoint sends through push notification channels.

Resource	Default quota	Eligible for increase
Maximum number of push notifications that can be sent per second in a campaign	25,000 notifications per second	Yes (p. 384)
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) message payload size	4 KB per message	No

Segment quotas

The following quotas apply to the Segments resource of the Amazon Pinpoint API.

Resource	Default quota	Eligible for increase
Maximum number of dimensions that can be used to create a segment	100 per segment	No

SMS quotas

The following quotas apply to the SMS channel.

Resource	Default quota	Eligible for increase
Spending threshold	USD $1.00 per account	Yes (p. 384), but spending limits vary by region. You must specify the region(s) in which you require an increase.
Number of SMS messages that can be sent each second (*sending rate*)	20 messages per second **Note** You can request a quota increase if your use case supports it.	Yes (p. 384) if your use case supports an increase; otherwise, No.
Number of SMS messages that can be sent to a single recipient each second	1 message per second	No
Number of Amazon SNS topics for two-way SMS	100,000 per account	Yes (p. 384)

10DLC quotas

The following quotas apply to 10DLC. 10DLC only applies to the U.S.

Resource	Default quota	Eligible for increase
Max 10DLC companies per AWSaccount	25	Yes (p. 384)
Max 10DLC campaigns per 10DLC company	10	Yes (p. 384)
Max 10DLC numbers per 10DLC campaign	49	Yes (p. 384), but may require carrier approval.

Voice quotas

The following quotas apply to the voice channel.

Note
When your account is removed from the sandbox, you automatically qualify for the maximum quotas shown in the following table.

Resource	Default quota	Eligible for increase
Number of voice messages that can be sent during a 24-hour period	If your account is in the sandbox: 20 messages	No

Resource	Default quota	Eligible for increase
Number of voice messages that can be sent to a single recipient during a 24-hour period	5 messages	No
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	No
Number of voice messages that can be sent from a single originating phone number per second	1 message per second	No
Voice message length	If your account is in the sandbox: 30 seconds If your account is out of the sandbox: 5 minutes	No
Ability to send voice messages to international phone numbers	If your account is in the sandbox, you can send messages to recipients in only 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.	No

Resource	Default quota	Eligible for increase
Number of characters in a voice message	3,000 billable characters, in words that are spoken 6,000 characters total, including billable characters and SSML tags	No
Number of configuration sets	10,000 voice configuration sets per AWS Region	No

Requesting a quota increase

If the value in the **Eligible for Increase** column in any of the preceding tables is **Yes**, you can request an increase for that quota.

To request a quota increase

1. Sign in to the AWSManagement Console at https://console.aws.amazon.com/.
2. Create a new AWS Support case at https://console.aws.amazon.com/support/home#/case/create.
3. On the **Create case** page, choose **Service quota increase**.
4. Under **Case classification**, for **Quota type**, choose one of the following options:
   - To request a quota increase that's related to the email channel, choose **Pinpoint Email**.
   - To request a quota increase that's related to the SMS channel, choose **Pinpoint SMS**.
   - To request a quota increase that's related to the voice channel, choose **Pinpoint Voice**.
   - To request an increase that's related to 10DLC, choose **Pinpoint**.
   - To request a quota increase that's related to any other Amazon Pinpoint feature, choose **Pinpoint**.
5. Depending on the option that you chose in the preceding step, enter the requested information in the optional and required fields to explain your use case.
6. Under **Requests**, do one of the following:
   - If your request is related to the SMS channel, for **Resource Type**, choose **General Quotas**.
   - If your request is related to another channel or any other Amazon Pinpoint feature, for **Region**, choose the AWS Region that you want to request the quota increase for. To request an increase to the same quota in an additional Region, choose **Add another request**, and then choose the additional Region.
7. Choose the quota that you want to increase, and then enter the new value that you want for the quota.
8. Under **Case description**, for **Use case description**, explain why you're requesting the quota increase.
9. Under **Contact options**, for **Preferred contact language**, choose the language that you prefer to use when communicating with the AWS Support team.
10. For **Contact method**, choose your preferred method of communicating with the AWS Support team.
11. Choose **Submit**.

The AWS Support team provides an initial response to your request within 24 hours.

In order to prevent our systems from being used to send unsolicited or malicious content, we have to consider each request carefully. If we’re able to do so, we'll grant your request within this 24-hour period.

However, if we need to obtain additional information from you, it might take longer to resolve your request.

We might not be able to grant your request if your use case doesn’t align with our policies.

Document history for Amazon Pinpoint

The following table describes important changes in each release of the *Amazon Pinpoint Developer Guide* after December 2018. For notification about updates to this documentation, you can subscribe to an RSS feed.

- **Latest documentation update:** November 16, 2020

update-history-change	update-history-description	update-history-date
UpdateEndpoint (p. 386)	The Amazon Pinpoint UpdateEndpoint API is now logged by CloudTrail.	November 16, 2020
Custom attributes (p. 386)	Amazon Pinpoint now supports 250 attributes in email messaging templates. See Quotas.	September 18, 2020
Regional availability (p. 386)	Amazon Pinpoint is now available in these Regions: Asia Pacific (Tokyo) Region, Europe (London) Region, and Canada (Central) Region. Note that the Amazon Pinpoint SMS and Voice API is not available in these Regions.	September 10, 2020
Regional availability (p. 386)	Amazon Pinpoint is now available in the Asia Pacific (Tokyo) Region. Note that the Amazon Pinpoint SMS and Voice API does not support Voice in this Region.	September 2, 2020
Campaign events (p. 386)	Added information about a new campaign event `delivery_type` parameter to Campaign events.	August 2, 2020
Regional availability (p. 386)	Amazon Pinpoint is now available in the Asia Pacific (Seoul) Region. Note that the Amazon Pinpoint API does not support Voice or SMS in this Region.	July 31, 2020
Regional availability (p. 386)	Amazon Pinpoint is now available in the AWSGovCloud (US) Region.	April 30, 2020

Custom channels (p. 386)	Updated information about creating custom channels by using Lambda functions or webhooks.	April 23, 2020
Machine learning (p. 386)	Added information about retrieving personalized recommendations from recommender models, and optionally enhancing those recommendations by using AWS Lambda functions.	March 4, 2020
Security (p. 386)	Added a Security chapter, which provides information about various security controls and features of Amazon Pinpoint.	February 4, 2020
Journeys (p. 386)	Added information about using Amazon Pinpoint journeys to develop automated workflows that perform messaging activities for projects. Also added information about querying analytics data for a subset of metrics that apply to journeys.	October 31, 2019
Analytics (p. 386)	Added procedures that explain how to query analytics data for campaigns and transactional messages, and added information about using query results.	October 17, 2019
Analytics (p. 386)	Added information about querying analytics data for a subset of metrics that apply to transactional email and SMS messages.	September 6, 2019
Code examples (p. 386)	Added code examples that you can use to send transactional push notifications using all of the services that Amazon Pinpoint supports.	July 30, 2019
Analytics (p. 386)	Added information about querying analytics data for a subset of metrics that apply to projects (applications) and campaigns.	July 24, 2019
Segments (p. 386)	Added a tutorial that describes a solution for importing customer data into Amazon Pinpoint from external systems, such as Salesforce or Marketo.	May 14, 2019

Regional availability (p. 386)	Amazon Pinpoint is now available in the AWS Asia Pacific (Mumbai) and Asia Pacific (Sydney) Regions.	April 25, 2019
Using postman with Amazon Pinpoint (p. 386)	Added a tutorial that describes how to use Postman to interact with the Amazon Pinpoint API.	April 8, 2019
Tagging (p. 386)	Added information about tagging Amazon Pinpoint resources.	February 27, 2019
SMS registration (p. 386)	Added a Tutorials chapter, and added a tutorial that describes how to create a solution that handles SMS user registration.	February 27, 2019
Code examples (p. 386)	Added code examples in several programming languages that show you how to send email, SMS, and voice messages programmatically.	February 6, 2019

Earlier updates

The following table describes important changes in each release of the *Amazon Pinpoint Developer Guide* through December 2018.

Change	Description	Date
Regional availability	Amazon Pinpoint is now available in the AWS US West (Oregon) and Europe (Frankfurt) Regions.	December 21, 2018
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
Europe (Ireland) Availability	Amazon Pinpoint is now available in the AWS Europe (Ireland) Region.	October 25, 2018
Events API	Use the Amazon Pinpoint API to record events (p. 112) 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.	August 7, 2018

Change	Description	Date
	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. 115) and Accessing audience data in Amazon Pinpoint (p. 139).	
Endpoint export permissions	Configure an IAM policy (p. 356) that allows you to export Amazon Pinpoint endpoints to an Amazon S3 bucket.	May 1, 2018
Phone number verification for SMS	Use the Amazon Pinpoint API to verify a phone number (p. 164) to determine whether it is a valid destination for SMS messages.	April 23, 2018
Updated topics for Amazon Pinpoint integration	Integrate Amazon Pinpoint (p. 108) 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. 285).	February 6, 2018
Updated service quotas	Updated *Quotas* (p. 373) with additional information about email quotas.	January 19, 2018
Public beta for Amazon Pinpoint extensions	Use AWS Lambda functions to customize segments (p. 154) or create custom messaging channels (p. 199).	November 28, 2017
External ID removed from IAM trust policies	The external ID key is removed from the example trust policy (p. 355) and example Java code (p. 153) for importing segments.	October 26, 2017
Push notification payload quotas	The quotas include payload sizes for mobile push messages (p. 381).	October 25, 2017
Updated service quotas	Added SMS and email channel information to *Quotas* (p. 373).	October 9, 2017
ADM and Baidu mobile push	Update your app code to handle push notifications from the Baidu and ADM mobile push channels.	September 27, 2017

Change	Description	Date
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. 110) and SDK for Java (p. 120).	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. 111).	August 31, 2017
Updated sample events	The example events (p. 206) 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
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. 206).	March 24, 2017
Permissions	See How Amazon Pinpoint works with IAM (p. 322) 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