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. 129).

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

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

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

Amazon Pinpoint Features

This section describes the major features of Amazon Pinpoint 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. 176). A segment designates which users receive the messages that are sent from a campaign. You can define dynamic segments based on data that's reported by your application, such as operating system or mobile device type. You can also import static segments that you define outside Amazon Pinpoint.

Engage Your Audience with Messaging Campaigns

Engage your audience by creating a messaging campaign (p. 186). A campaign sends tailored messages on a schedule that you define. You can create campaigns that send mobile push, email, or SMS messages.

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

Send Transactional Messages

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

Analyze User Behavior

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

To analyze or store the analytics data outside of Amazon Pinpoint, you can configure Amazon Pinpoint to stream the data (p. 225) 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.

For a list of all the Regions where Amazon Pinpoint is currently available, see AWS Regions and 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.

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 AWS Management 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 Amazon Pinpoint Email API, as well as the Amazon Pinpoint SMS and Voice API.

To create the policy

1. Sign in to the AWS Management 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 Amazon Pinpoint Email API and Amazon Pinpoint 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. In 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**.
5. Under **Attach permissions policies**, choose the policy that you created in the previous section, and then choose **Next: Tags**.

   Choose **Next: Permissions**.
6. Under **Set permissions**, choose **Attach existing policies directly**. In the list of policies, choose the **PostmanAccessPolicy** that you created in Step 1.1 (p. 4).

   Choose **Next: Tags**.
7. On the **Add tags** page, you can optionally add tags that help you identify the user. For more information about using tags, see Tagging IAM Entities in the *IAM User Guide*.

   Choose **Next: Review**.
8. On the **Review** page, confirm the settings for the user. When you're ready to create the user, choose **Create user**.
9. 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 in a later step 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**.

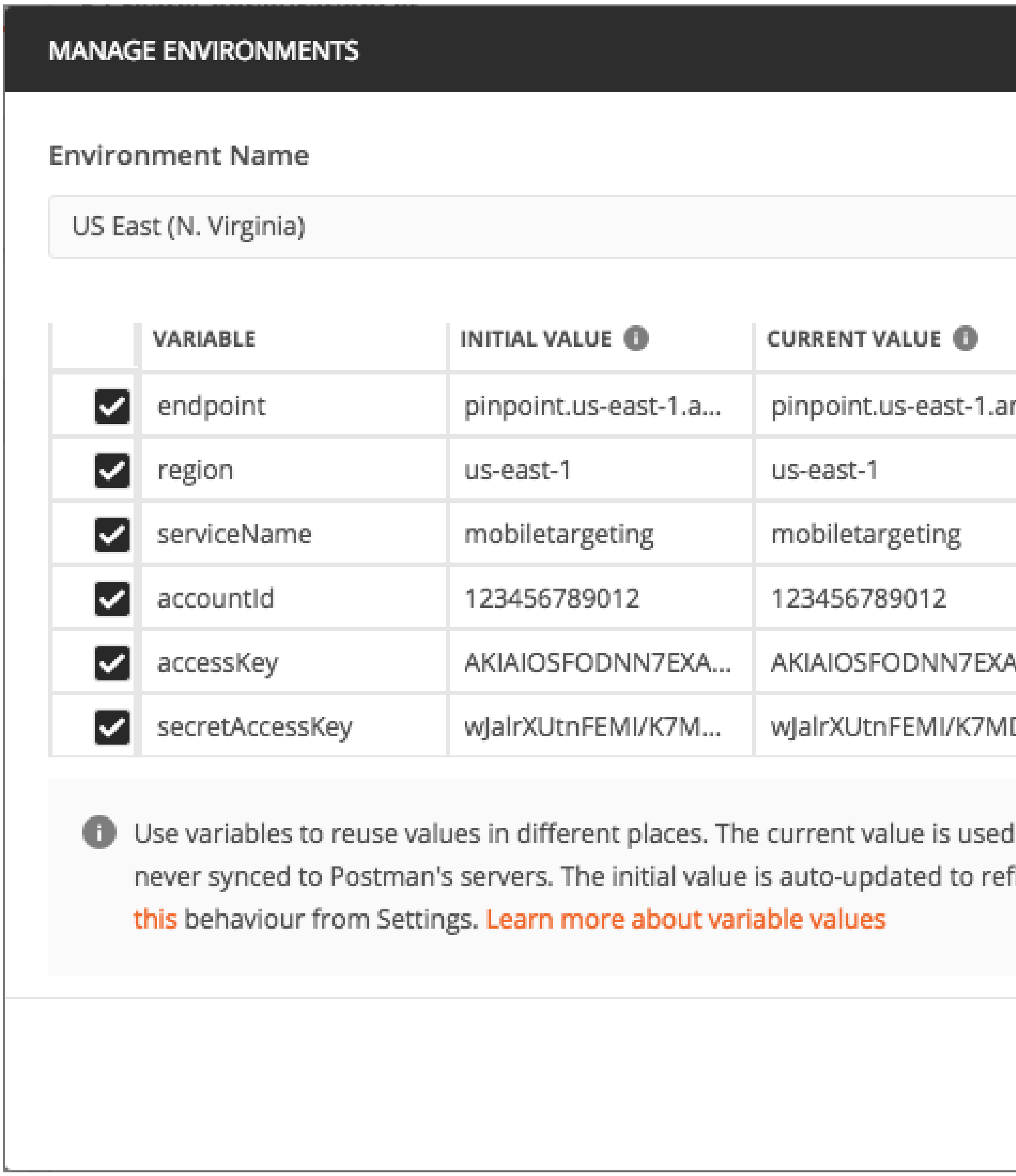
2. On the **Create New** window, choose **Environment**.
3. On the **MANAGE ENVIRONMENTS** window, for **Environment Name**, enter **`Amazon Pinpoint - Region Name`**. Replace *`Region Name`* with one of the following values:

   - US East (N. Virginia)
   - US West (Oregon)
   - Asia Pacific (Mumbai)
   - Asia Pacific (Sydney)
   - EU (Frankfurt)
   - EU (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)*

Region	Variable	Initial Value
Asia Pacific (Sydney)	`endpoint`	**`pinpoint.ap-southeast-2.amazonaws.com`**
	`region`	**`ap-southeast-2`**
	`serviceName`	**`mobiletargeting`**
	`accountId`	*(your AWS account ID)*
	`accessKey`	*(your IAM access key ID)*
	`secretAccessKey`	*(your IAM secret access key)*
EU (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)*
EU (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.ar
✓	region	us-east-1	us-east-1
✓	serviceName	mobiletargeting	mobiletargeting
✓	accountId	123456789012	123456789012
✓	accessKey	AKIAIOSFODNN7EXA...	AKIAIOSFODNN7EXA
✓	secretAccessKey	wJalrXUtnFEMI/K7M...	wJalrXUtnFEMI/K7MD

Use variables to reuse values in different places. The current value is used
never synced to Postman's servers. The initial value is auto-updated to ref
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.

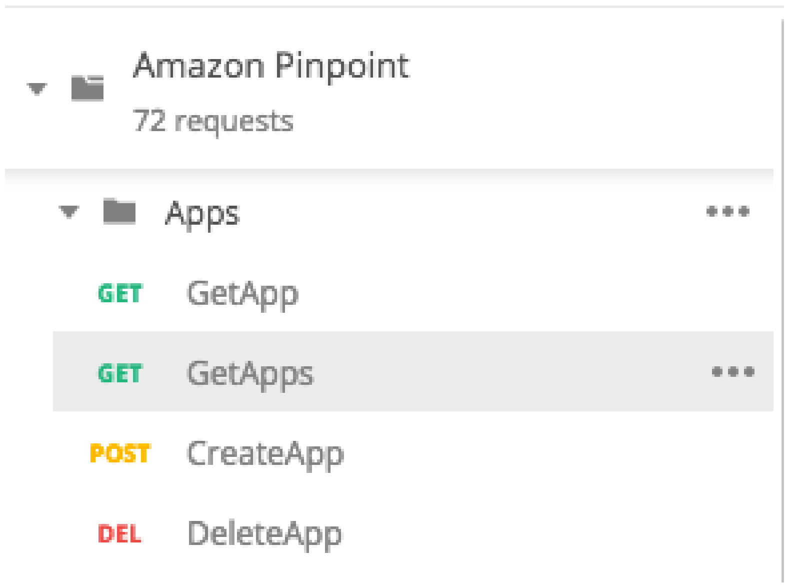
   Choose **Import**. Postman imports the Amazon Pinpoint collection, which contains 120 example requests.

Step 2.3: Test Your Postman Configuration

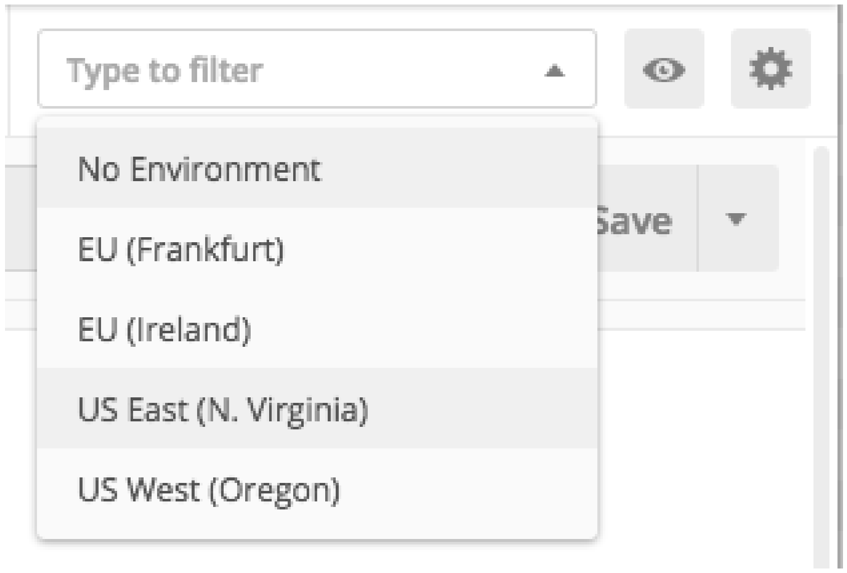
After you import the Amazon Pinpoint collection, you should perform a quick test to make sure that all of the components are properly configured. You can test your configuration by submitting a `GetApps` request. This request returns a list of all of the projects that exist in your Amazon Pinpoint account 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

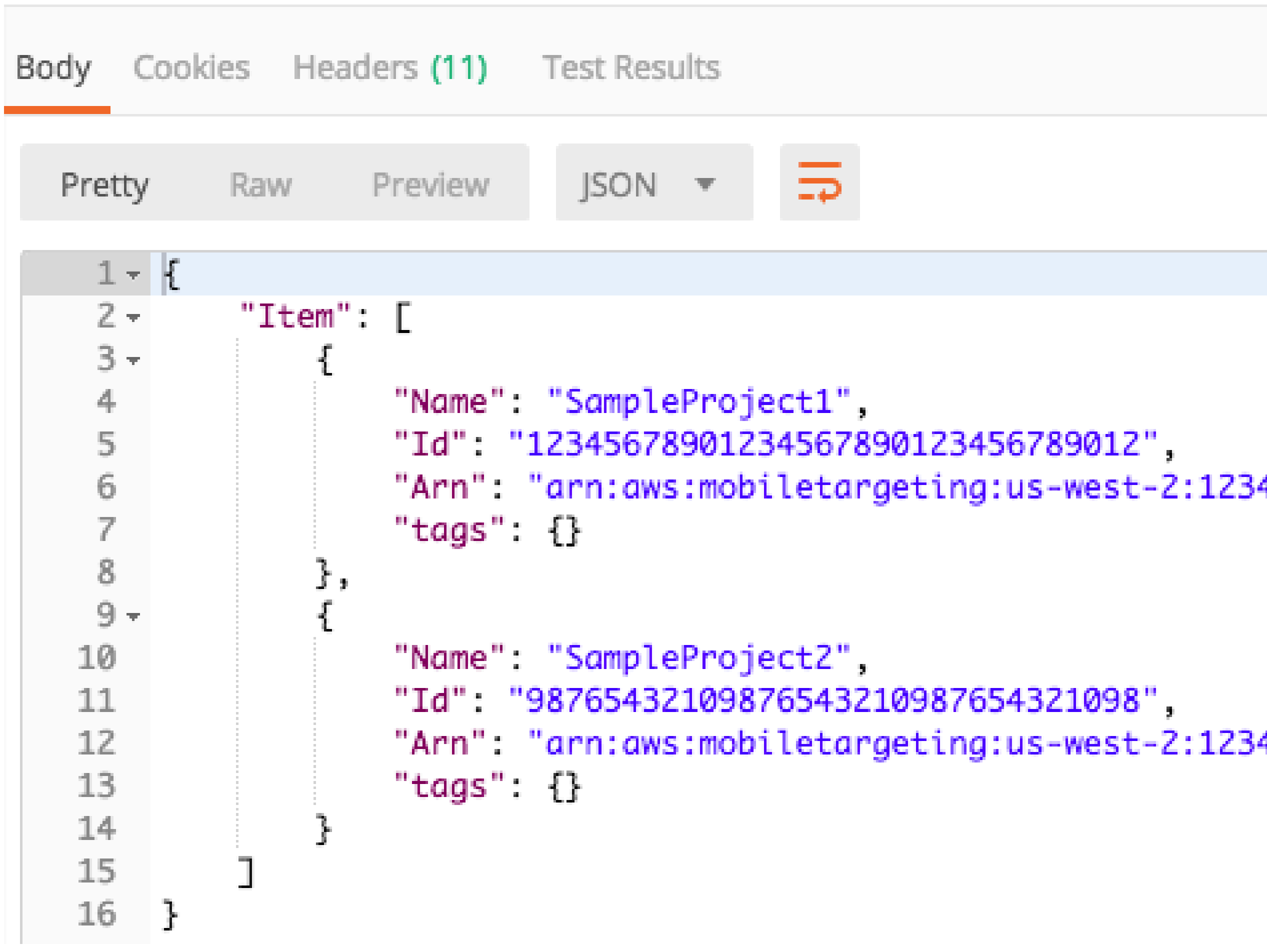
1. In the navigation pane, expand the **Amazon Pinpoint** collection, and then expand the **Apps** folder.
2. In the list of requests, choose **GetApps**.



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-2:1234
            "tags": {}
        },
        {
            "Name": "SampleProject2",
            "Id": "98765432109876543210987654321098",
            "Arn": "arn:aws:mobiletargeting:us-west-2:1234
            "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. Make sure that the `accessKey` and `secretAccessKey` values appear in both the **INITIAL VALUE** and **CURRENT VALUE**

Error Message	Problem	Resolution
		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. These example values are highlighted in the following image. Delete the example values and replace them with the appropriate values for your Amazon Pinpoint project and campaign.

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 (in Amazon Pinpoint, a "project" is the same thing as an "application"). 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.

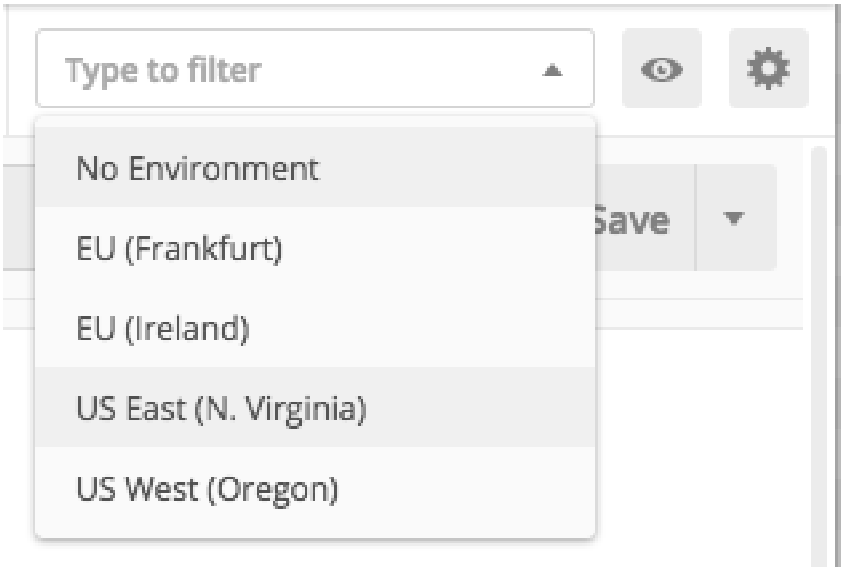
Note
The `UntagResource` operation also requires you to specify URL parameters. You can specify these parameters on the **Params** tab. Replace the values in the **VALUE** column with the tags that you want to delete for the specified resource.

Example Request: Creating a Project by Using the `CreateApp` Operation

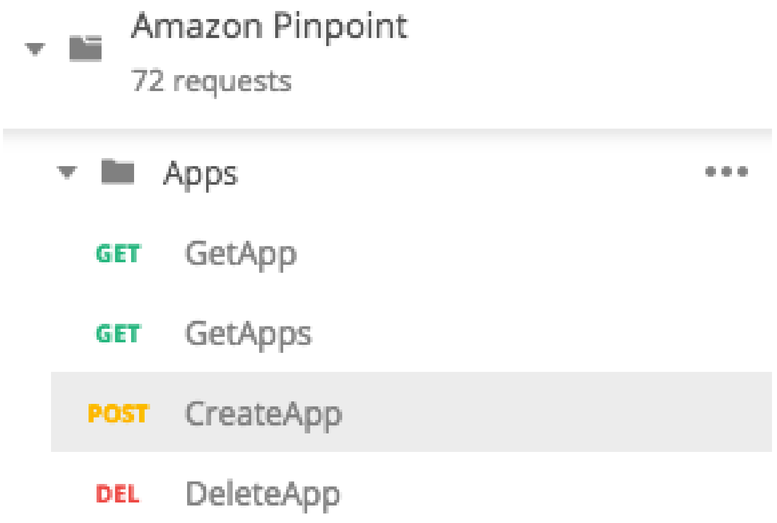
Before you create segments and campaigns in Amazon Pinpoint, you first have to create a project. In Amazon Pinpoint, a *project* consists of segments, campaigns, configurations, and data that are united by a common purpose. For example, you could use a project to contain all of the content that's related 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.

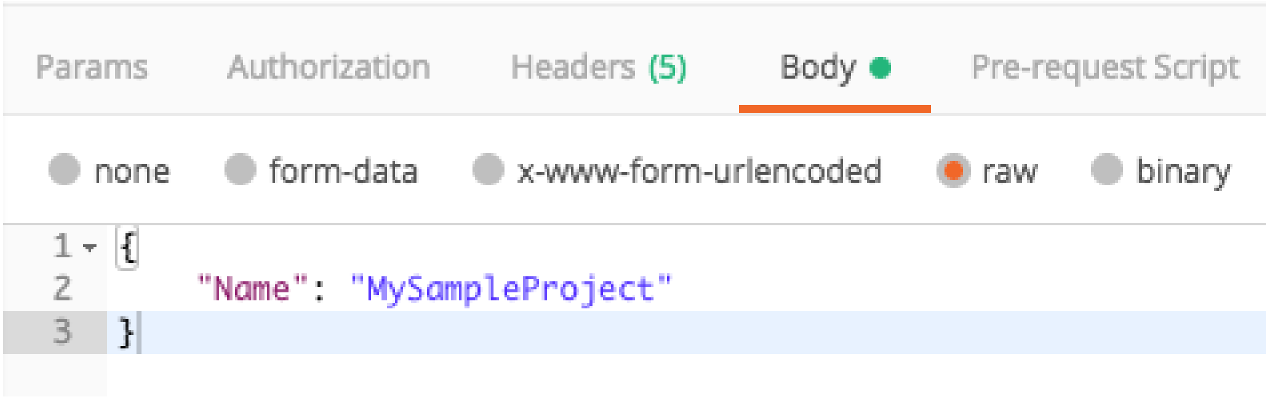


2. In the **Apps** folder, choose the **CreateApp** operation, as shown in the following image.



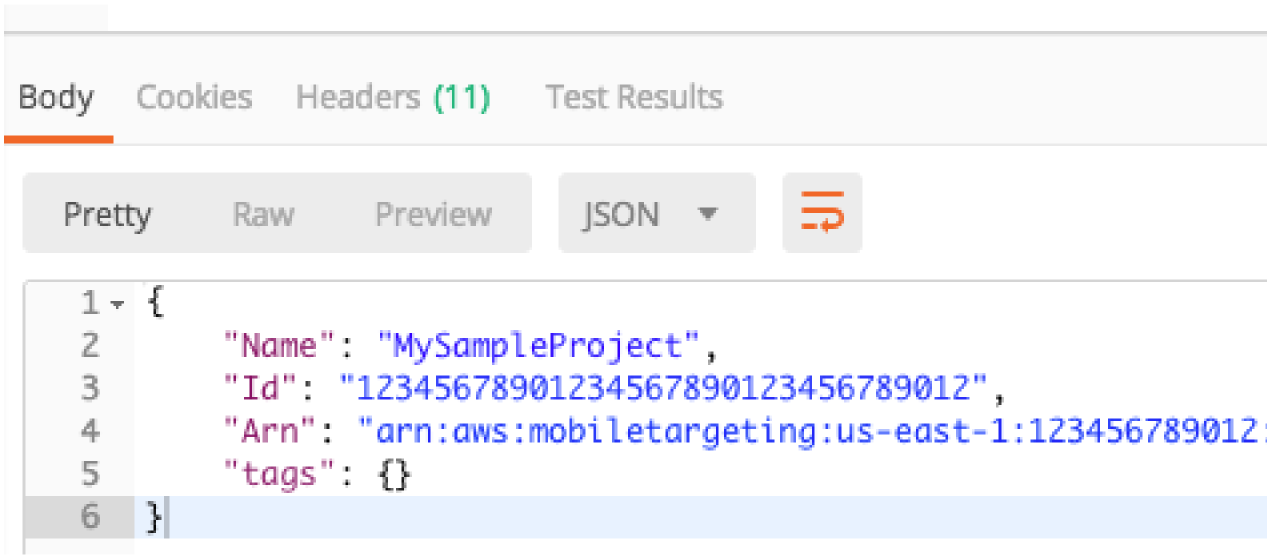
3. On the **Body** tab, next to `"Name"`, replace the placeholder value (`"string"`) with a name for the campaign, such as **`"MySampleProject"`**.

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
    "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.

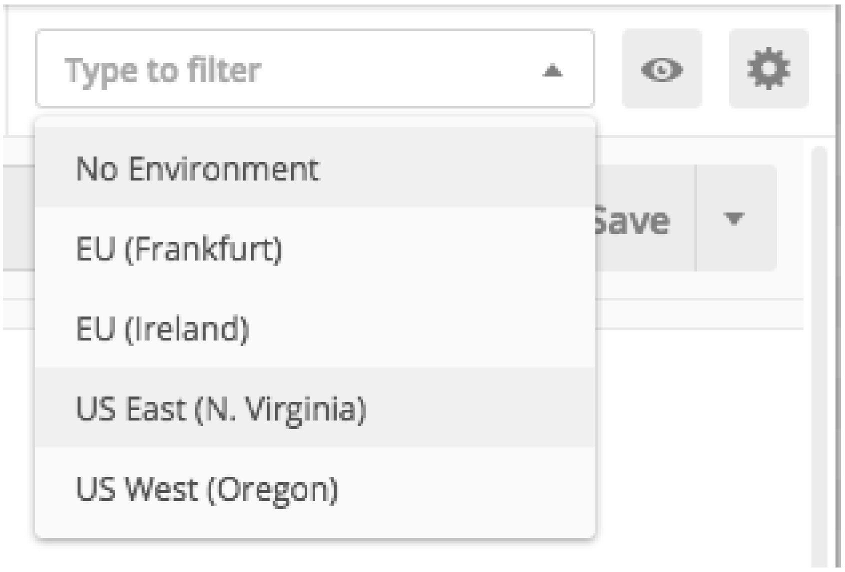
The example in this section shows you how to send a transactional email message directly to a specific email address. You can modify this request to send messages through other channels, such as SMS, mobile push, or voice.

To send an email message by submitting a SendMessages request

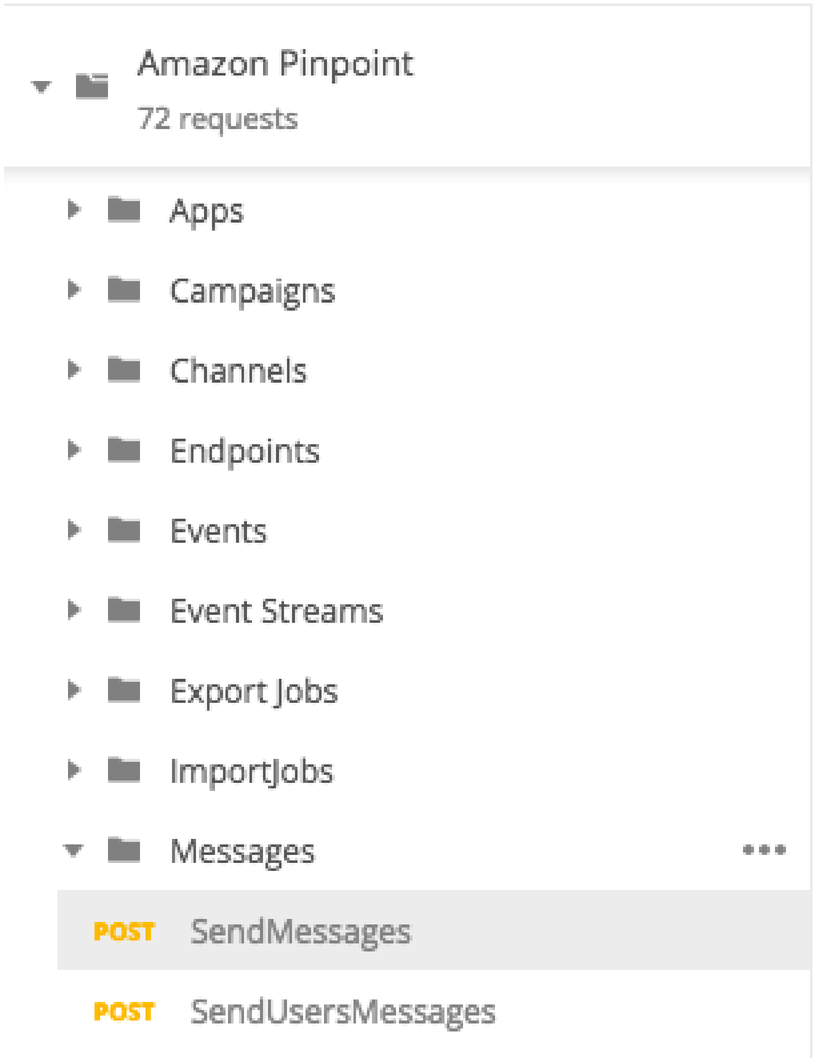
1. Verify the email address or domain that you want to use to send the message. For more information, see Verifying Email Identities in the *Amazon Pinpoint User Guide*.

   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.



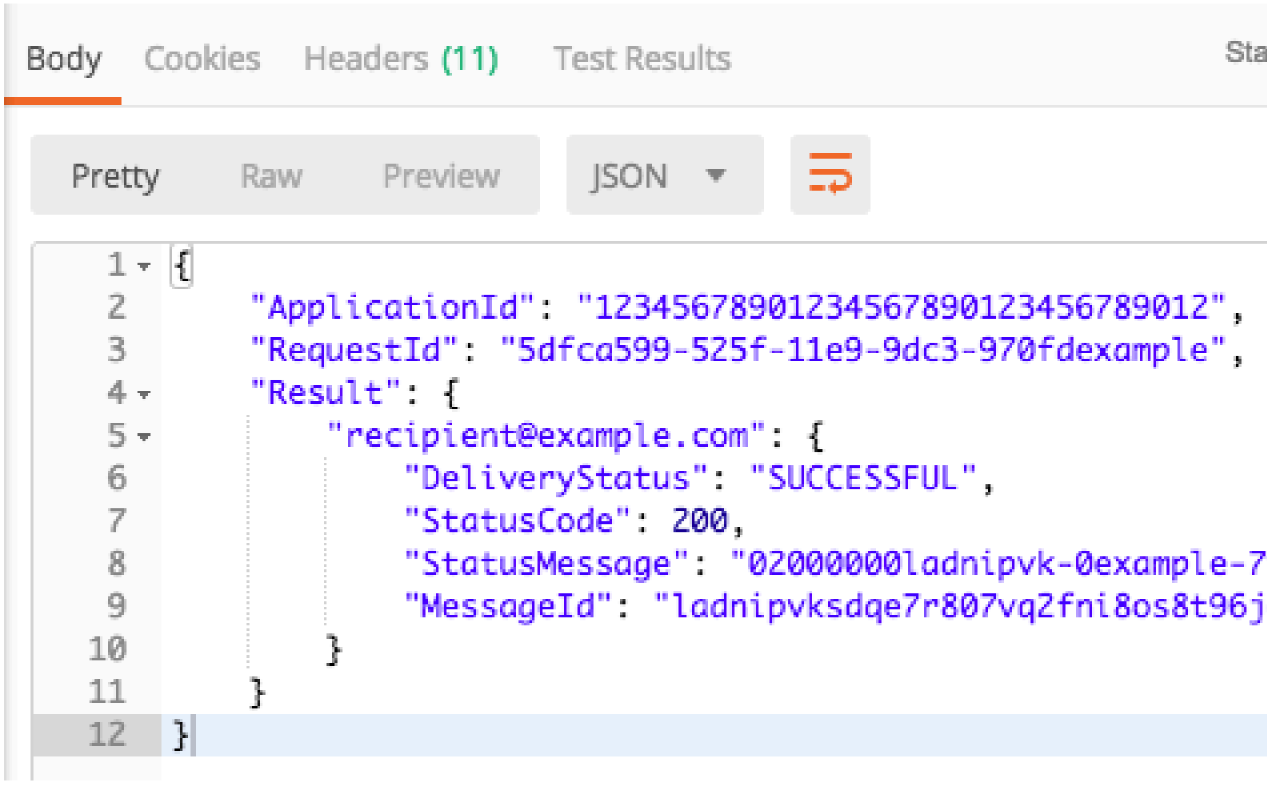
4. On the **Pre-request Script** tab, replace the value of the `projectId` variable with the ID of a project that already exists in the Region that you selected in step 2 of this section.
5. On the **Body** tab, delete the example request that's shown in the request editor. Paste the following code:

```
{
    "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": "0200000ladnipvk-0example-7
            "MessageId": "ladnipvksdqe7r807vq2fni8os8t96j
        }
    }
}
```

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 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 web site.
> - 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:

- Creates a separate record for each endpoint (email address, phone number) that it finds in the file.
- Changes the headers in the incoming spreadsheet to the record names that Amazon Pinpoint expects.
- 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 is capable of receiving SMS messages, then it creates the endpoint as an SMS endpoint. Otherwise, it creates the endpoint as a voice endpoint.
- 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.
- 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, making 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 flow of data in this solution is illustrated in the following diagram.

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 in order to download some of the packages that are necessary to use this solution. 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 in to 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 associate 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 that it takes for 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 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.

Description	Cost (US Dollars)
	Your usage of Amazon S3 in implementing this solution might be included in the 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 of 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 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 AWS Management 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 testing 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

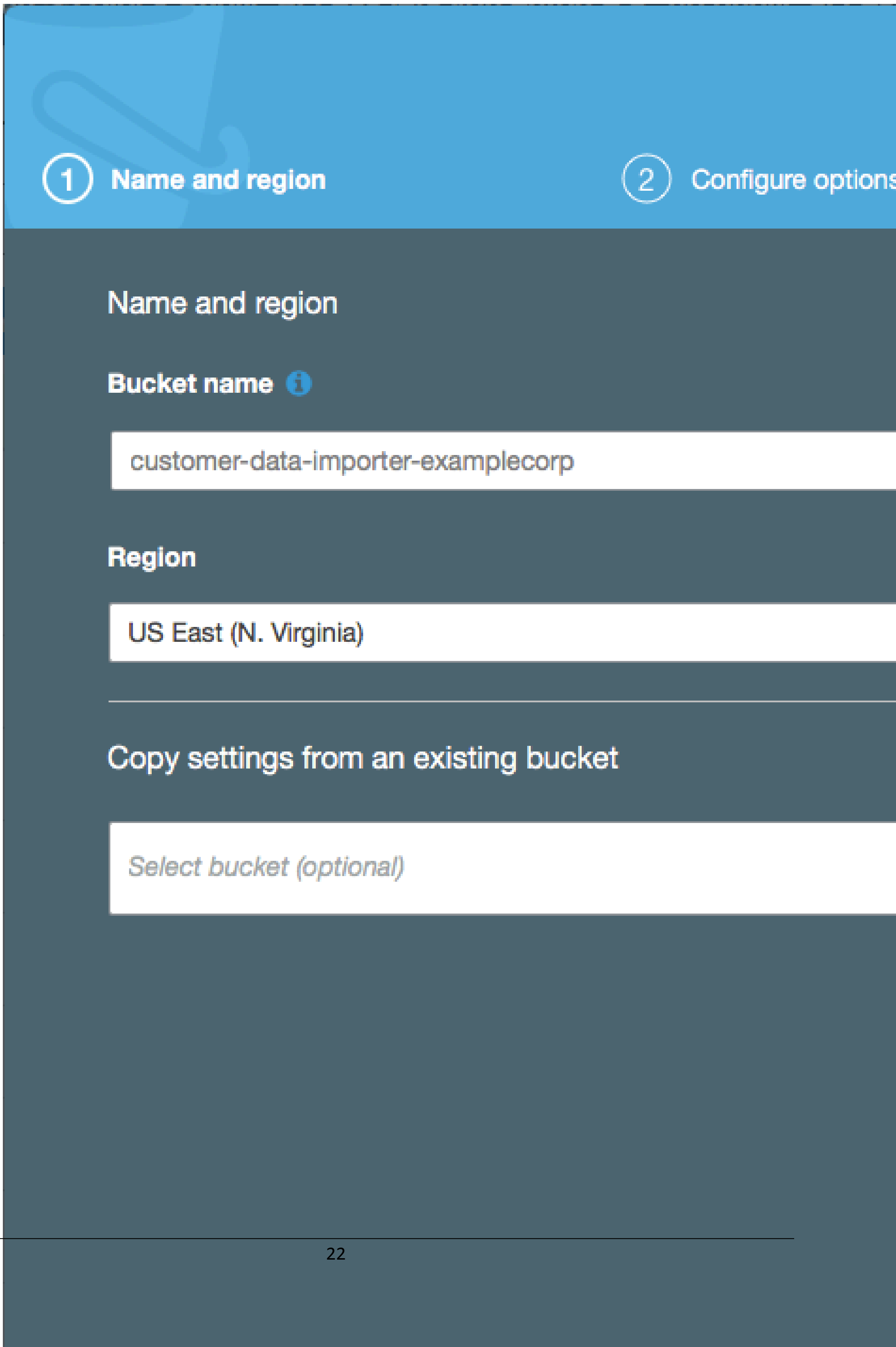
In this solution, you upload files that you want to import into an `input` folder in an Amazon S3 bucket. When you upload a file into this folder, Amazon S3 triggers a Lambda function. This function moves the input file into an `archive` folder, and also creates several smaller files in a `to_process`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

Step 1.1: Create the 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 the Amazon S3 Bucket

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



Bucket name

customer-data-importer-examplecorp

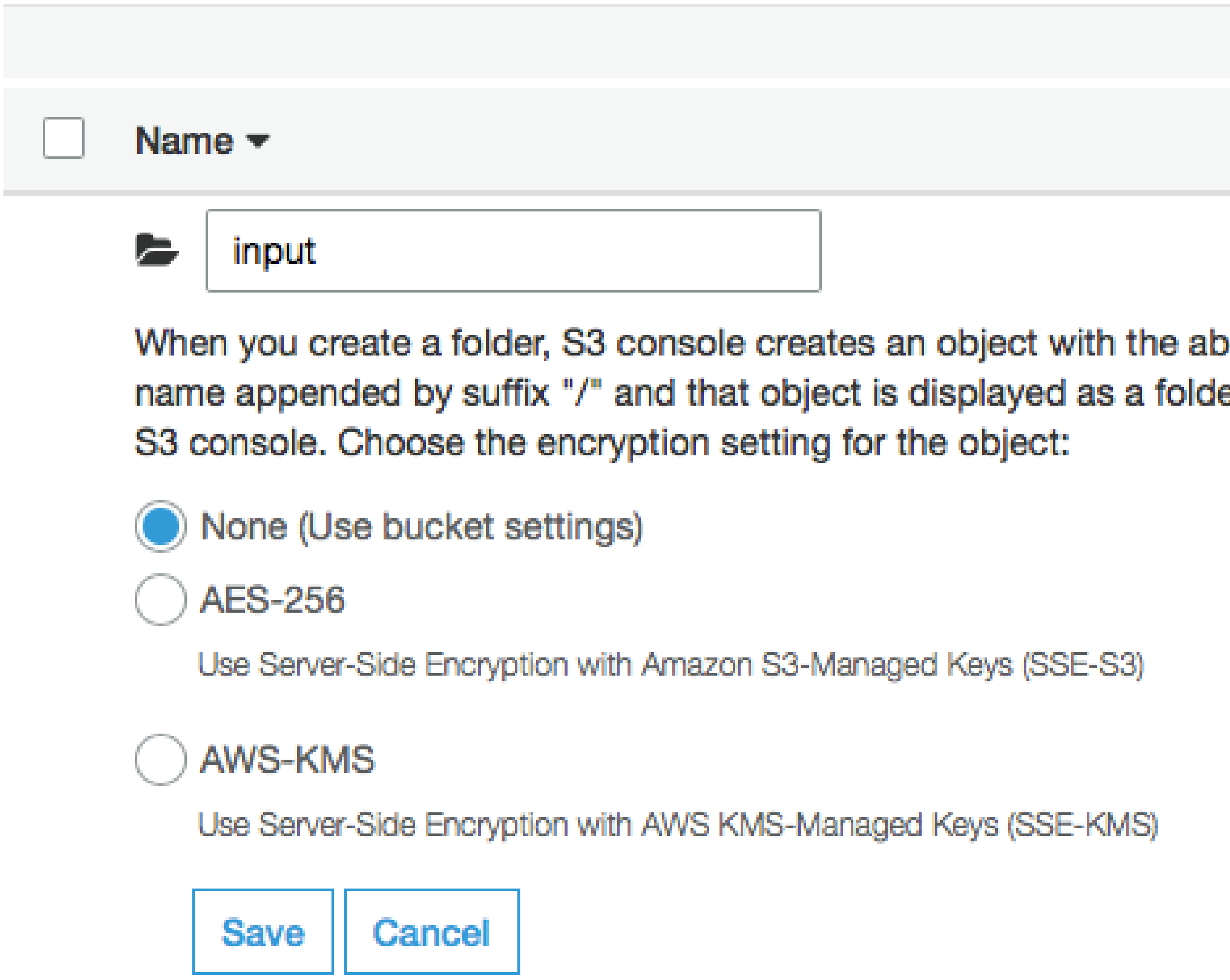
Region

US East (N. Virginia)

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**.



Name

input

When you create a folder, S3 console creates an object with the ab
name appended by suffix "/" and that object is displayed as a folde
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, allowing it 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. 21) 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. 21) 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 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 the values above. 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-import -itemtype directory
   ```

3. Enter the following command to change into 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 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. 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. 28). 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 for 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 ensure 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. 21), 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. 21). When you finish, choose **Create**.
8. Choose **Test** again. The function will execute 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. 23).
   - 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 filename 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. 21).

   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. 29). 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, because Amazon Pinpoint expects countries to 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. 29) 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. 28). 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)
```

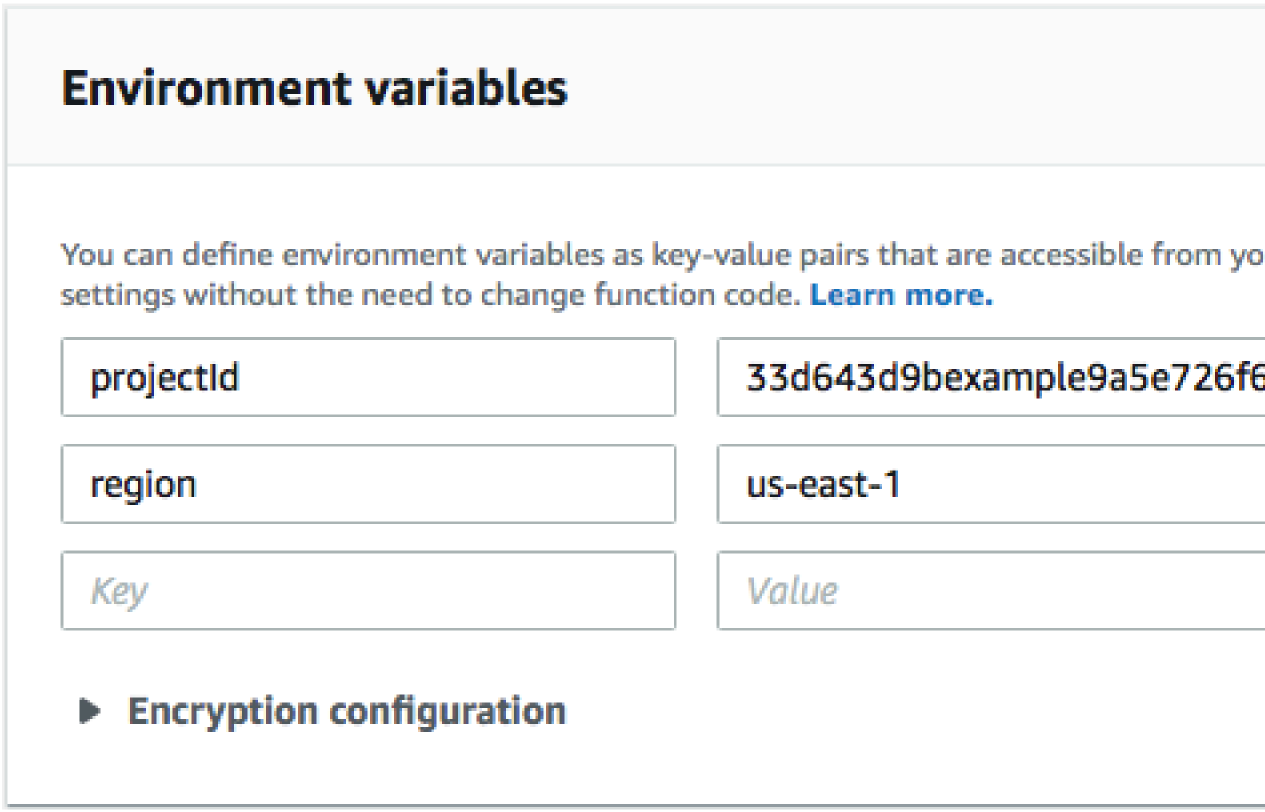
7. In the function editor, on the **File** menu, choose **Save As**. Save the file as `lambda_function.py` in the root directory for the function.
8. In the function editor, on the **File** menu, choose **New File**. The editor creates a new file named `Untitled1`.
9. Paste the following code in the 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 for 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. 26).
    - In the second row, create a variable with a key of **`region`**. Next, set the value to the Region that that you specified in the IAM policy in Step 2.2 (p. 26).

    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 yo
settings without the need to change function code. Learn more.

projectId	33d643d9bexample9a5e726f6
region	us-east-1
Key	Value

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

    Note
    Each call to the phone number validation service typically takes about .5 seconds to complete, but 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
    Higher concurrency limits 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 the limits of Amazon S3. As a result, the import process doesn't capture all of the data in your input spreadsheet, 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 ensure 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. 23). 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. 21).
   - 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. 21).

   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 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 ensure 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 for the previous two functions, 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. 26).
   - In the second row, create a variable with a key of **`region`**. Next, set the value to the Region that that you specified in the IAM policy in Step 2.2 (p. 26).
   - In the third row, create a variable with a key of **`importRoleArn`**. Next, set the value to the Amazon Resource Name of the IAM role that you created in Step 2.2 (p. 25).
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 ensure 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": "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. 23). When you finish, choose **Create**.
2. Choose **Test** again. The function will execute 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. 21).

   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_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 ensure 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.

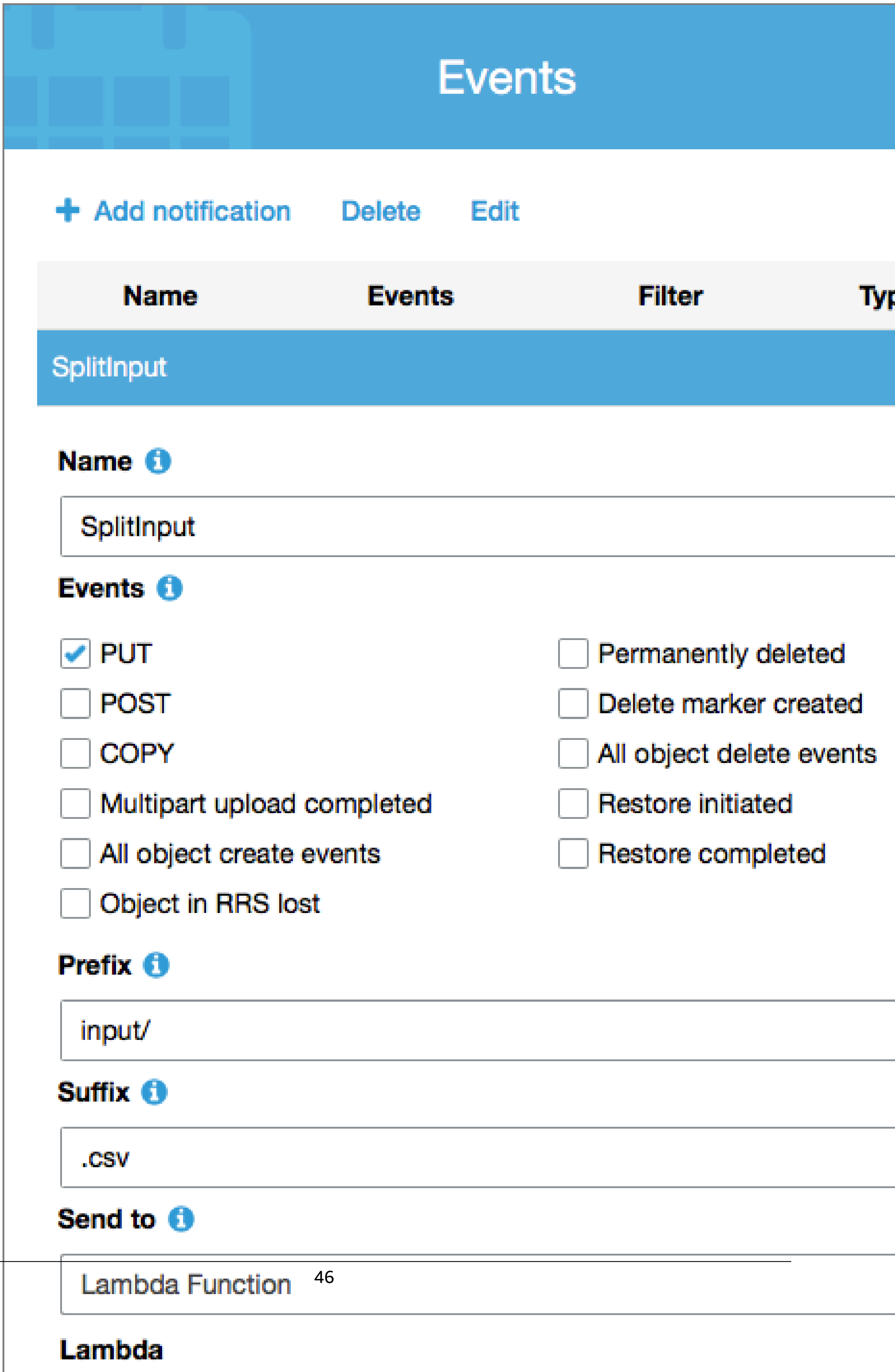
Step 7.1: Set Up the Event Notifications

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. 21).
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.


Events

Name	Events	Filter	Typ
SplitInput			

Name: SplitInput

Events

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

Prefix: input/

Suffix: .csv

Send to: Lambda Function

Lambda

- 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. When you do, it triggers the Lambda function that you created in Step 4 (p. 29). 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. 31). 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. 21), 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 function 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 that will 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 delete the input_archive, to_process, and processed folders entirely. These folders are recreated automatically the next time the functions in this tutorial are executed.

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 by 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.

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 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, and 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 will fail, because it 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.

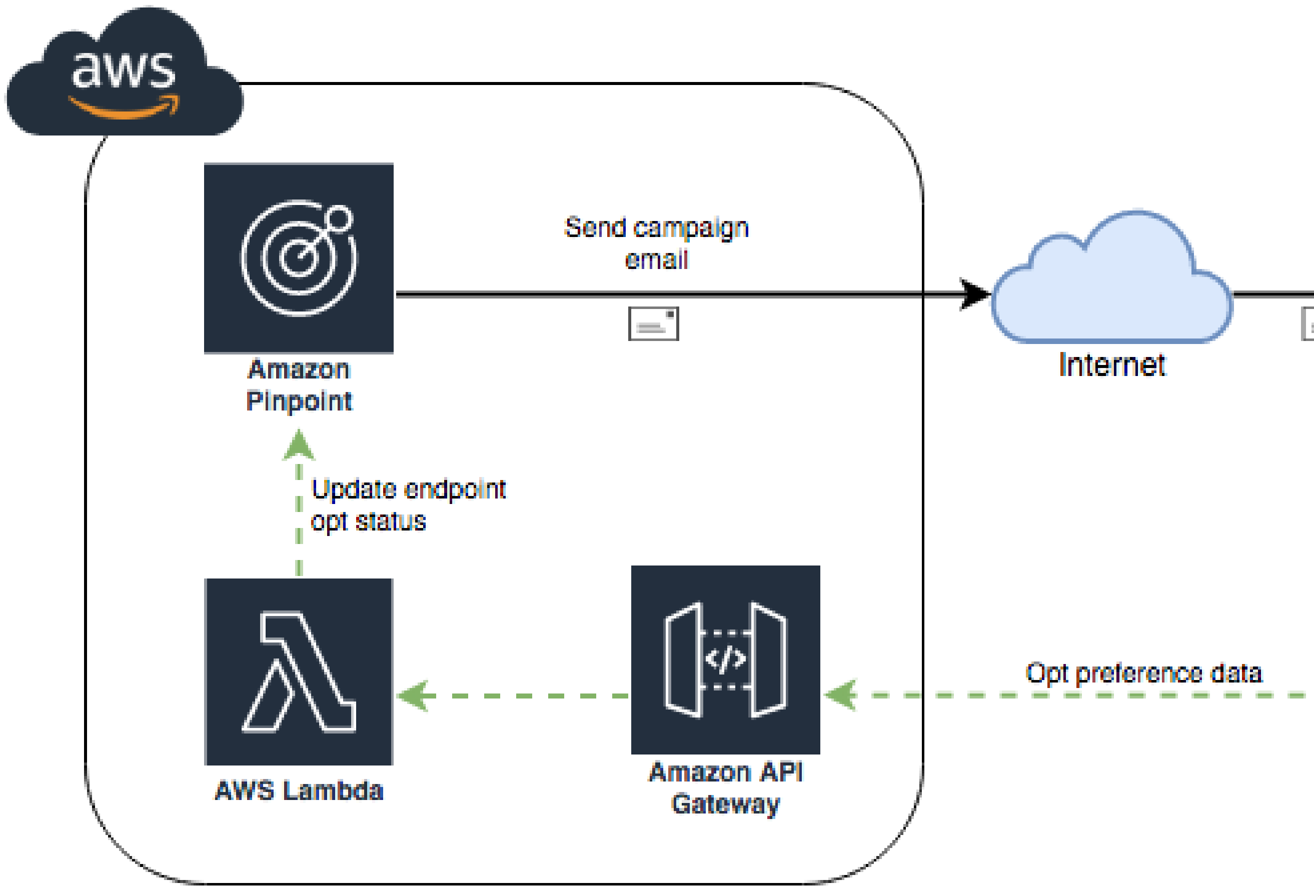
The web form also includes an option that customers can choose to opt themselves out of all email communications. When customers choose this option, they're automatically opted out of all topics. Additionally, the solution in this tutorial modifies their endpoint records so that they don't appear in 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. 51)

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 AWS Management 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. 52)

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. 51).

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.

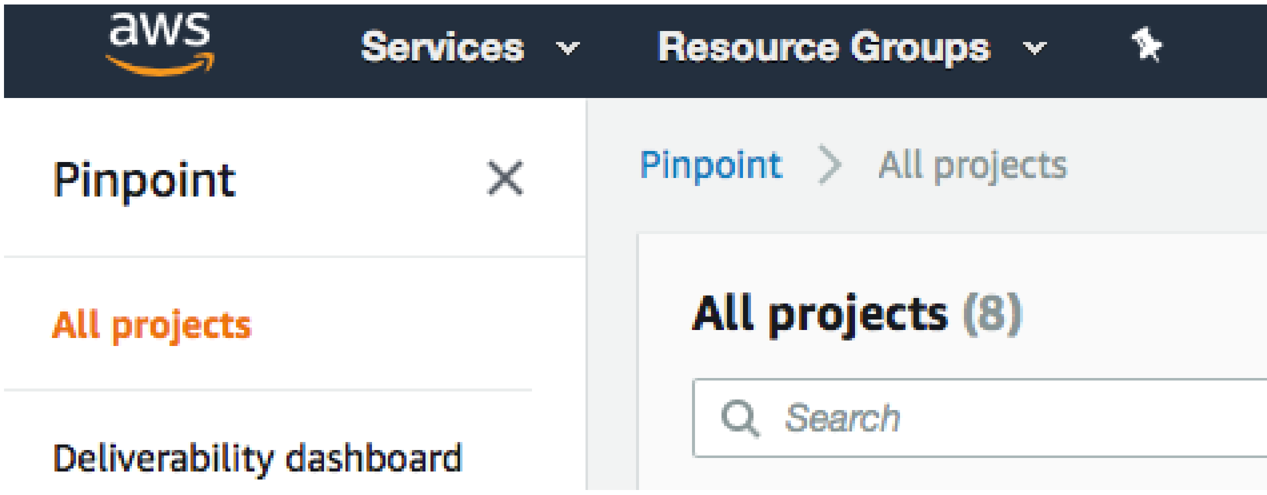
The steps involved in creating a new project differ depending on whether you've created a project in Amazon Pinpoint previously.

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 AWS Management 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.

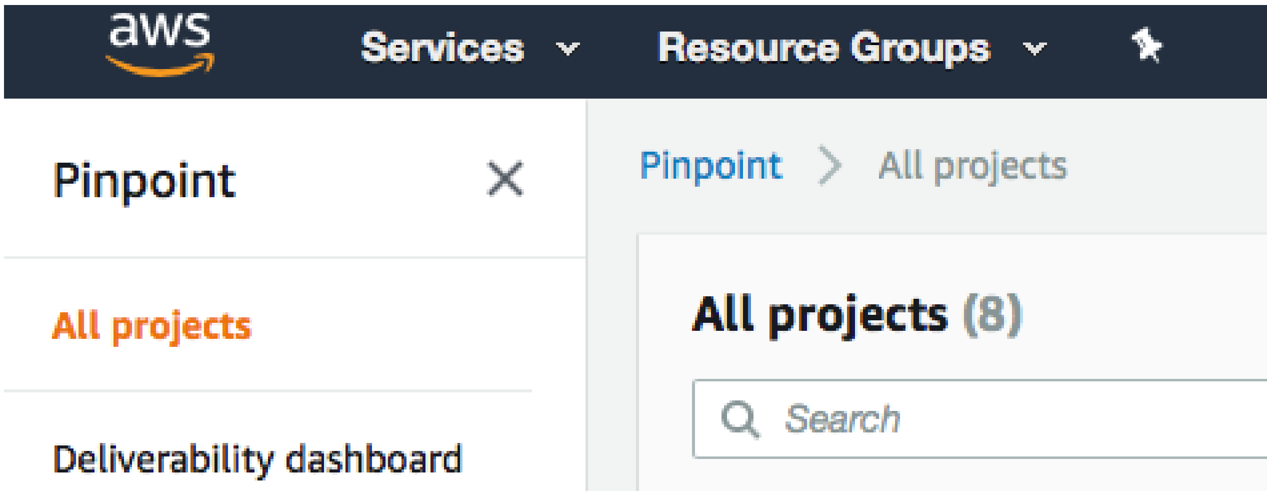
   Tip
   You need to use this ID in a few different places in this tutorial. Keep the project ID in a convenient place so that you can copy it later.

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 AWS Management 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. 55) 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. 54) 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 Route 53	Creating Records by Using the Amazon Route 53 Console
GoDaddy	Add a CNAME record (external link)
Dreamhost	How do I add custom DNS records? (external link)
Cloudflare	How do I add a CNAME record? (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 and Limitations 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. 57). 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. 52).
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 AWS Management 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. 52) 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. 59) 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

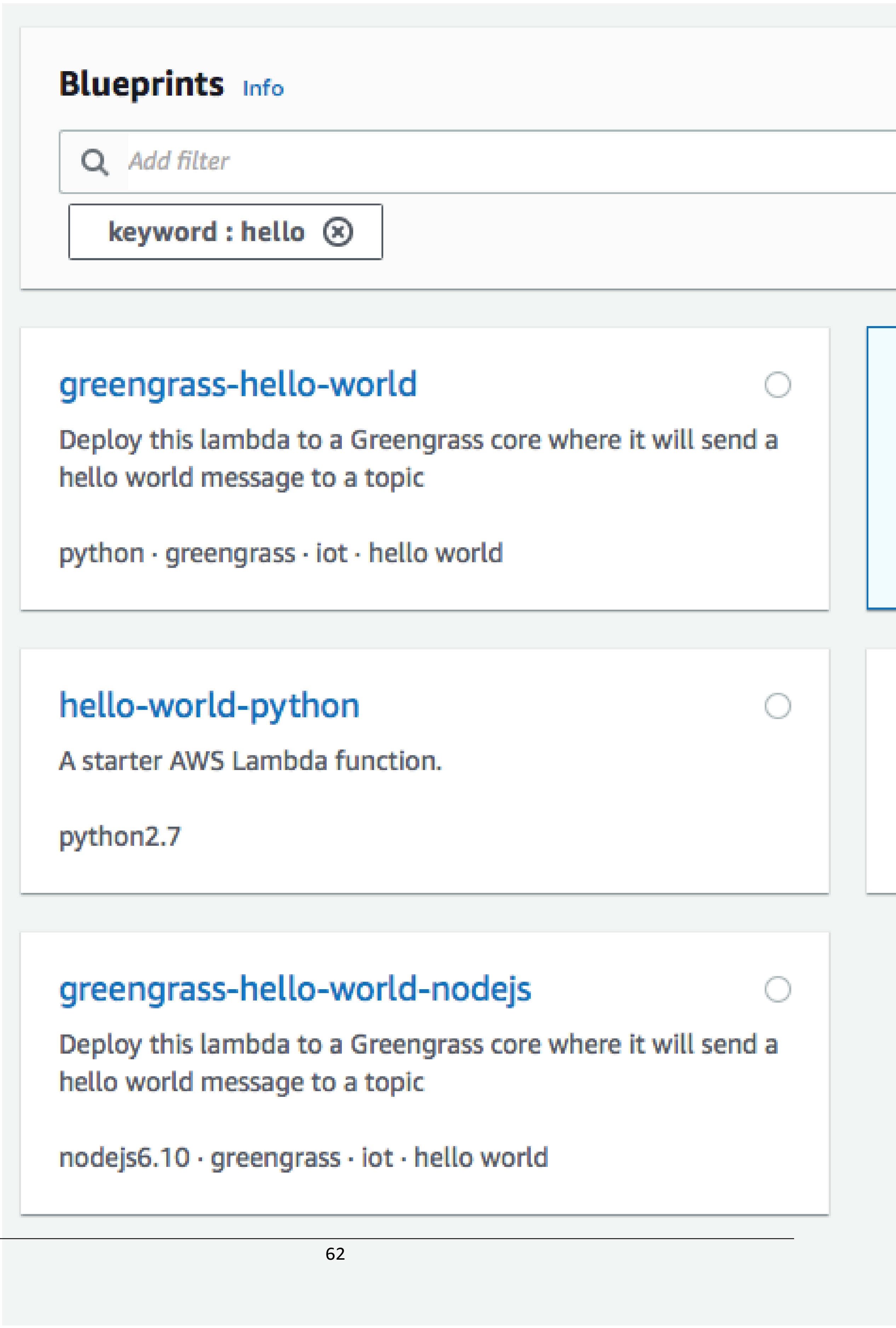
This solution uses a Lambda function to update customers' endpoint records based on the preferences that they provide on the web form. This section shows you how to create, configure, and test this function. Later, you set up API Gateway and Amazon Pinpoint to execute this function.

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**.


Blueprints Info

Add filter

keyword : hello

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-python

A starter AWS Lambda function.

python2.7

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. 60).

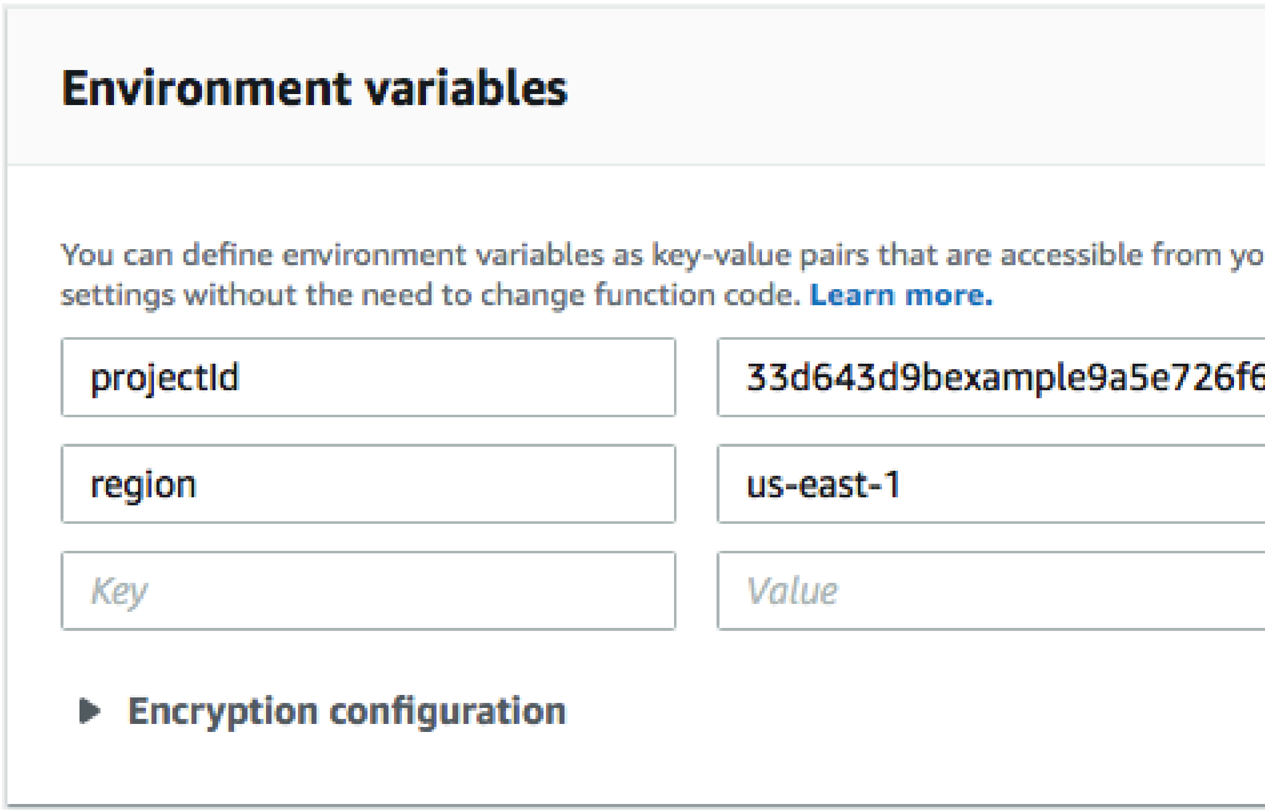
   When you finish, choose **Create function**.
6. Delete the example code in the code editor, and then paste the following code:

```
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. 52).
- 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 yo
settings without the need to change function code. Learn more.

projectId	33d643d9bexample9a5e726f6
region	us-east-1
Key	Value

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

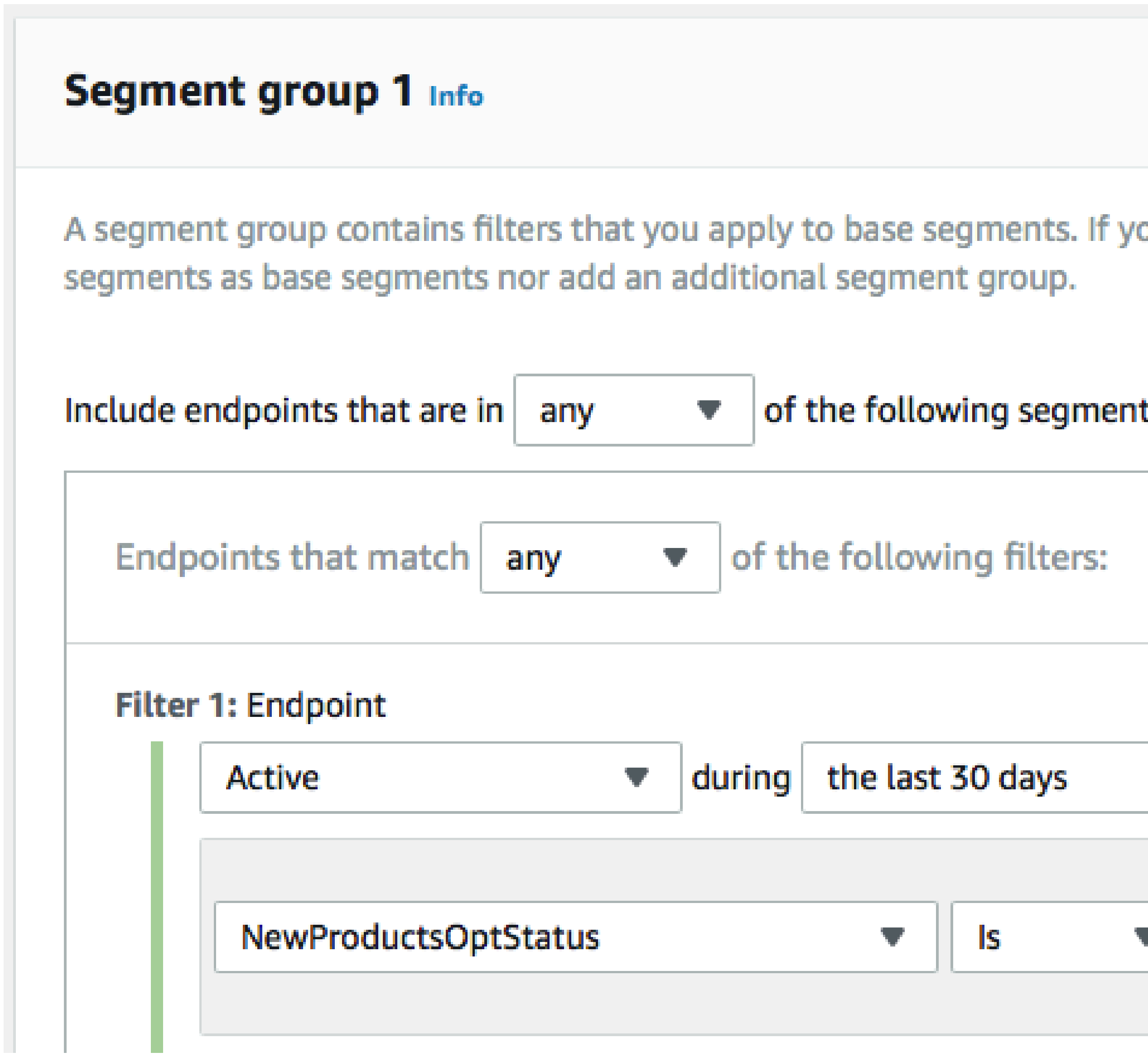
1. Choose **Test**.
2. On the **Configure test event** window, do the following:
   - Choose **Create new test event**.
   - For **Event name**, enter a name for the test event, such as **`MyTestEvent`**.
   - Erase the example code in the code editor. Paste the following code:

```
{
  "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. 57).
- 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. 52).
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 yo
segments as base segments nor add an additional segment group.

Include endpoints that are in any of the following segment

Endpoints that match any of the following filters:

Filter 1: Endpoint

Active during the last 30 days

NewProductsOptStatus Is

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. 61).

Step 5.1: Create the API

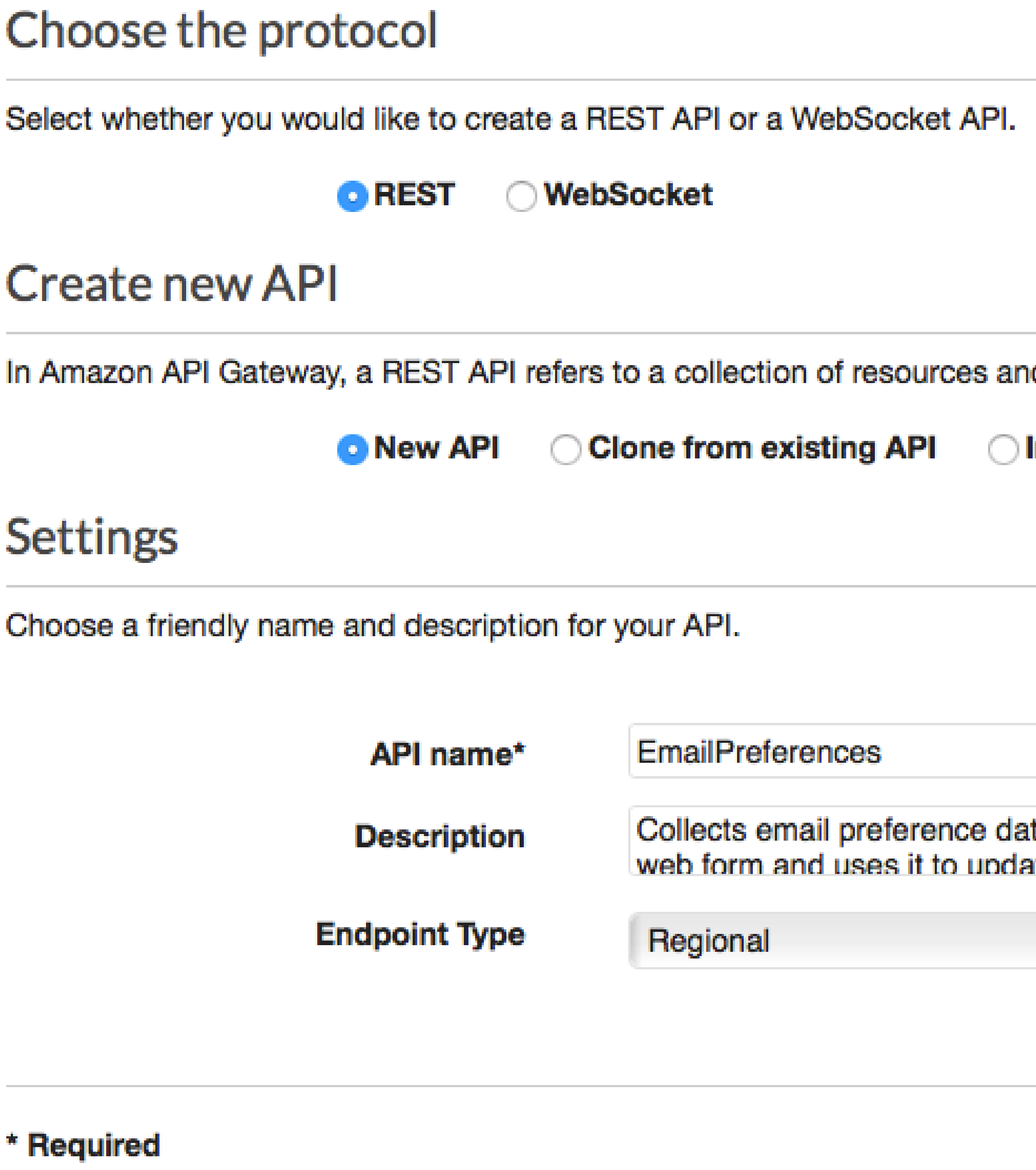
First, you have to create a new API in API Gateway. The following procedures show you how to create a new REST API.

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 an

[✓ New API] Clone from existing API I

Settings

Choose a friendly name and description for your API.

API name* EmailPreferences

Description Collects email preference dat
web form and uses it to upda

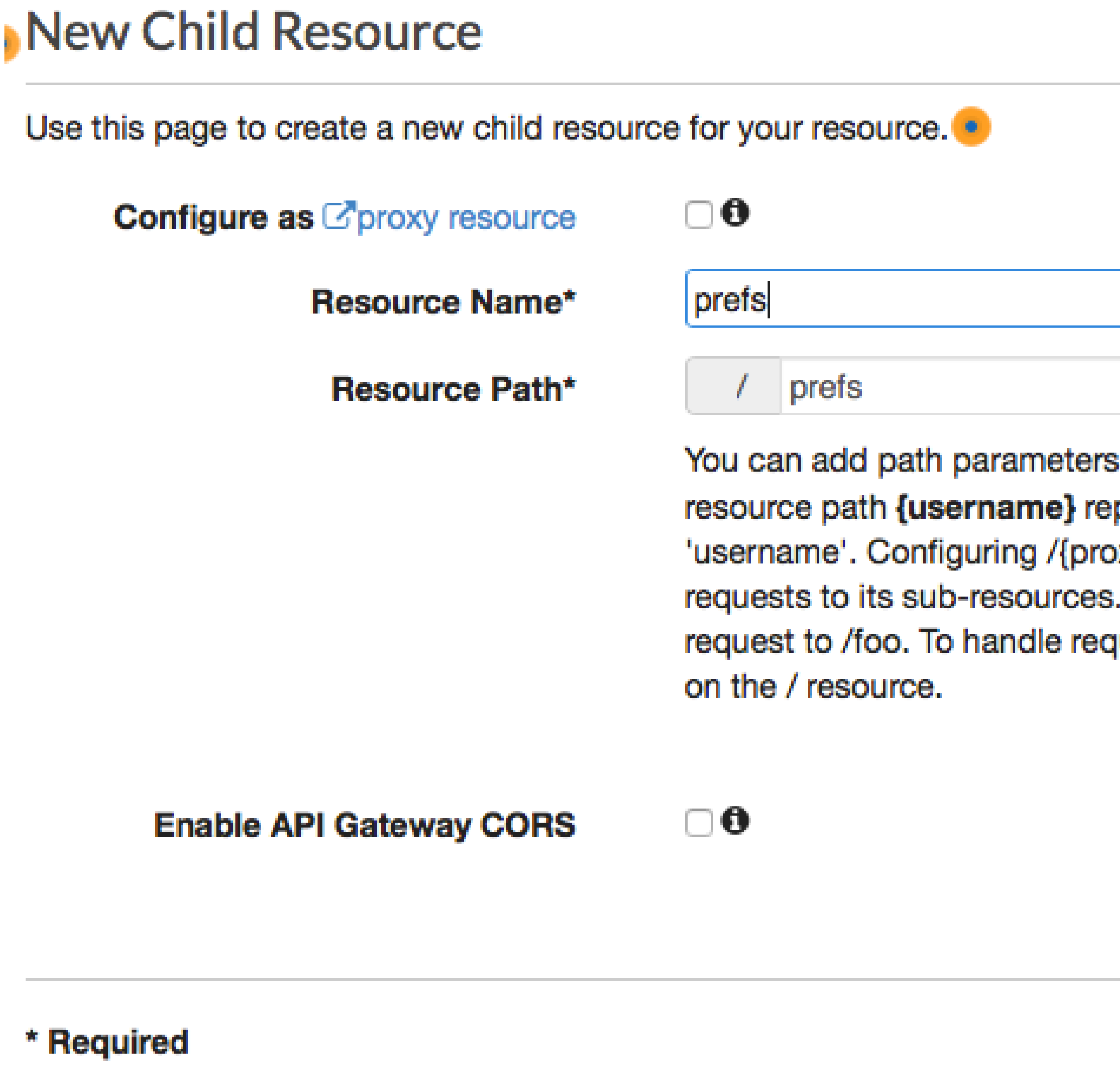
Endpoint Type Regional

* 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

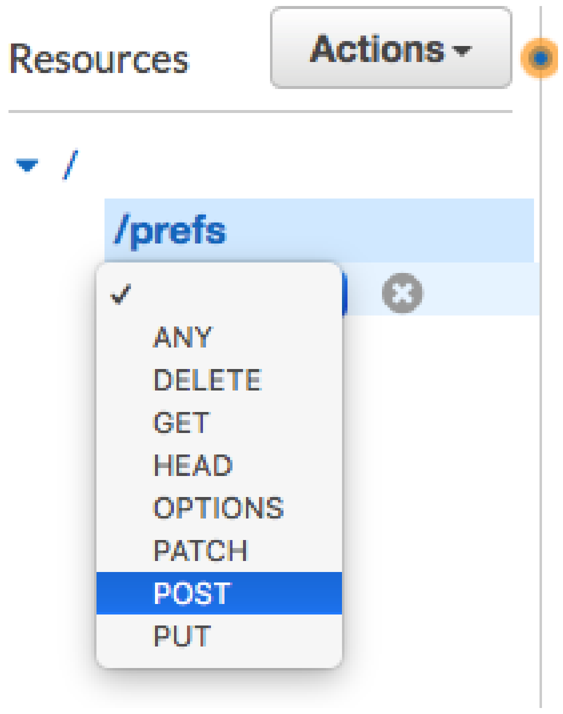
Resource Path* / prefs

You can add path parameters
resource path {username} rep
'username'. Configuring /{pro
requests to its sub-resources.
request to /foo. To handle req
on the / resource.

Enable API Gateway CORS ☐

* Required

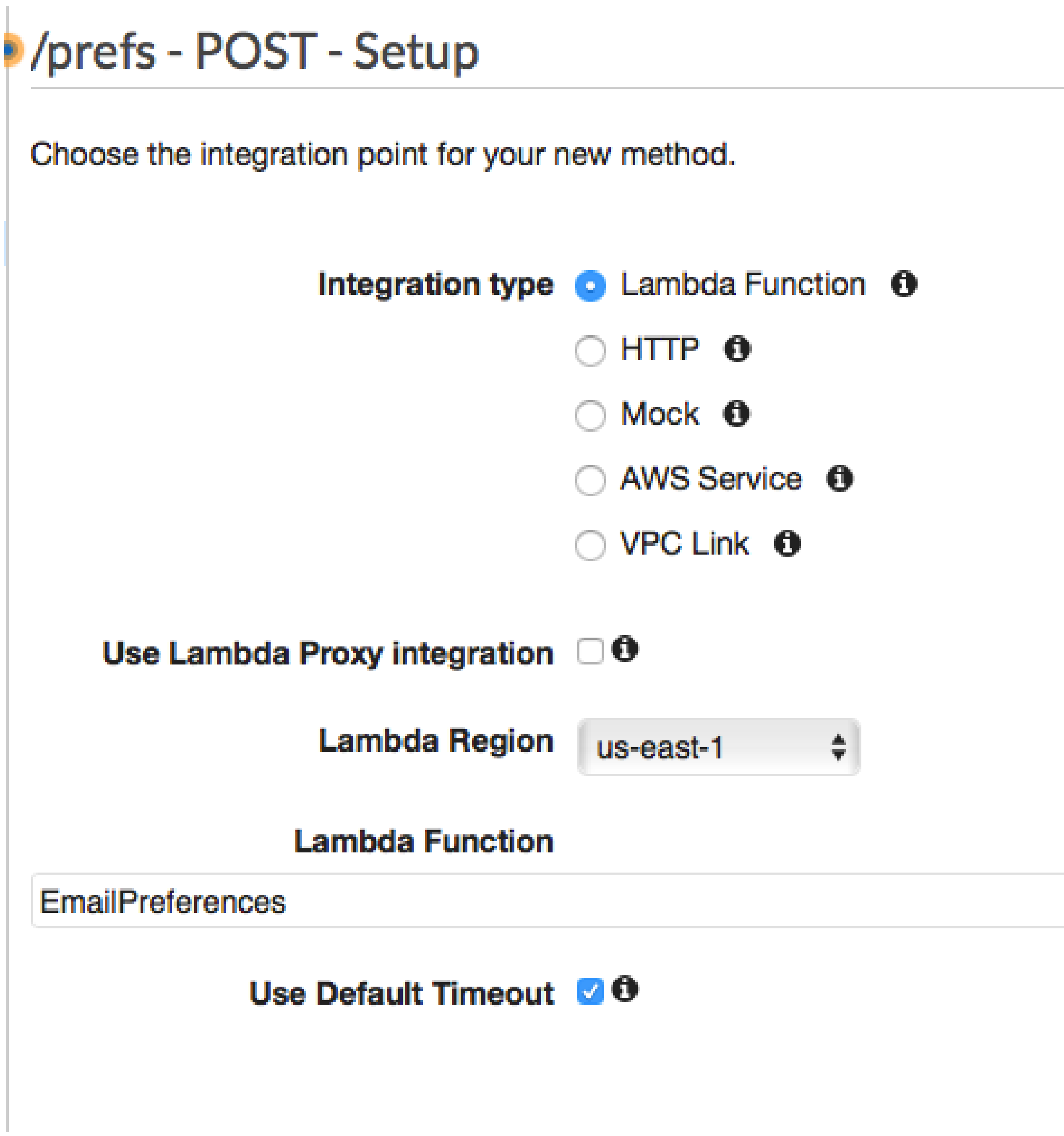
2. On the **Actions** menu, choose **Create Method**. From the menu that appears, choose **POST**, as shown in the following image. Then choose the **check mark** (✓) button.



3. In the **/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. 61).

   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

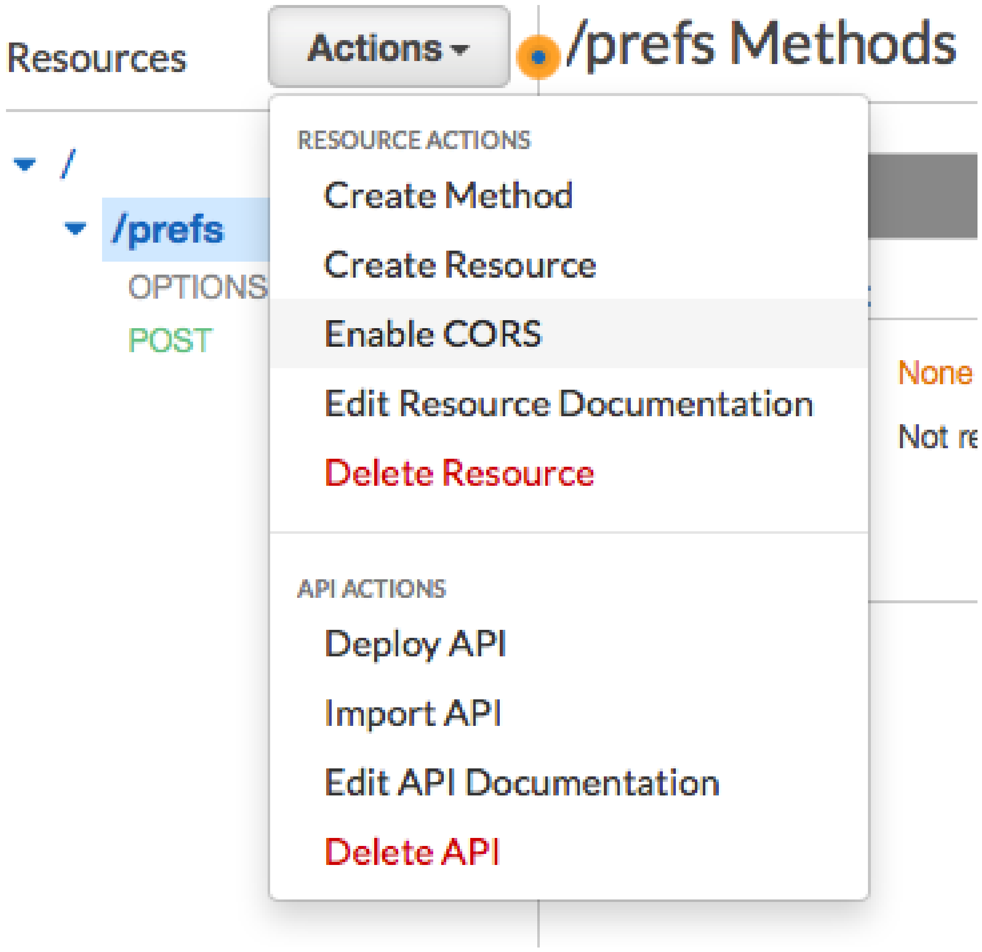
Use Default Timeout: ✓

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.

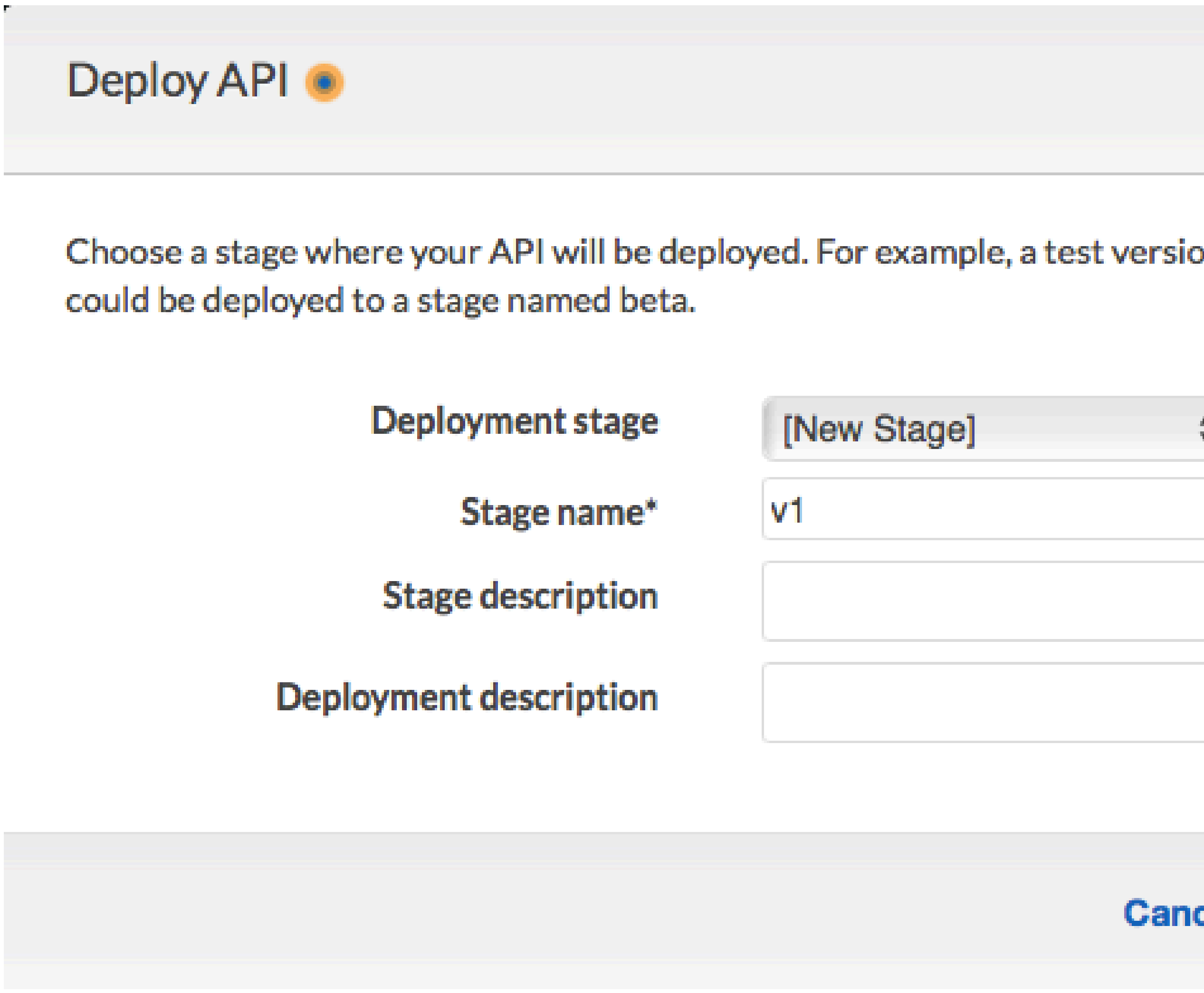
1. In the navigation pane, choose **Resources**. In the list of resources, choose the **/prefs** resource. Finally, on the **Actions** menu, choose **Enable CORS**, as shown in the following image.



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.

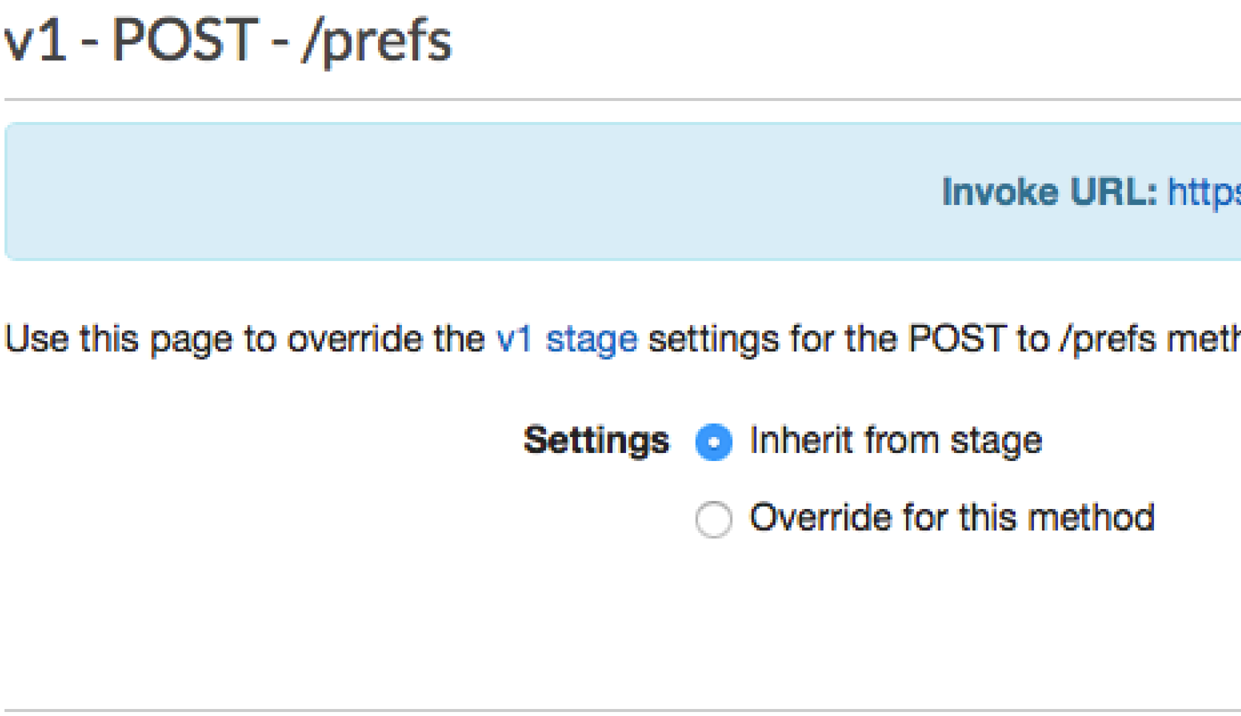


Deploy API

Choose a stage where your API will be deployed. For example, a test versio
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

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

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. 61).

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. 72).
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 Route 53, 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

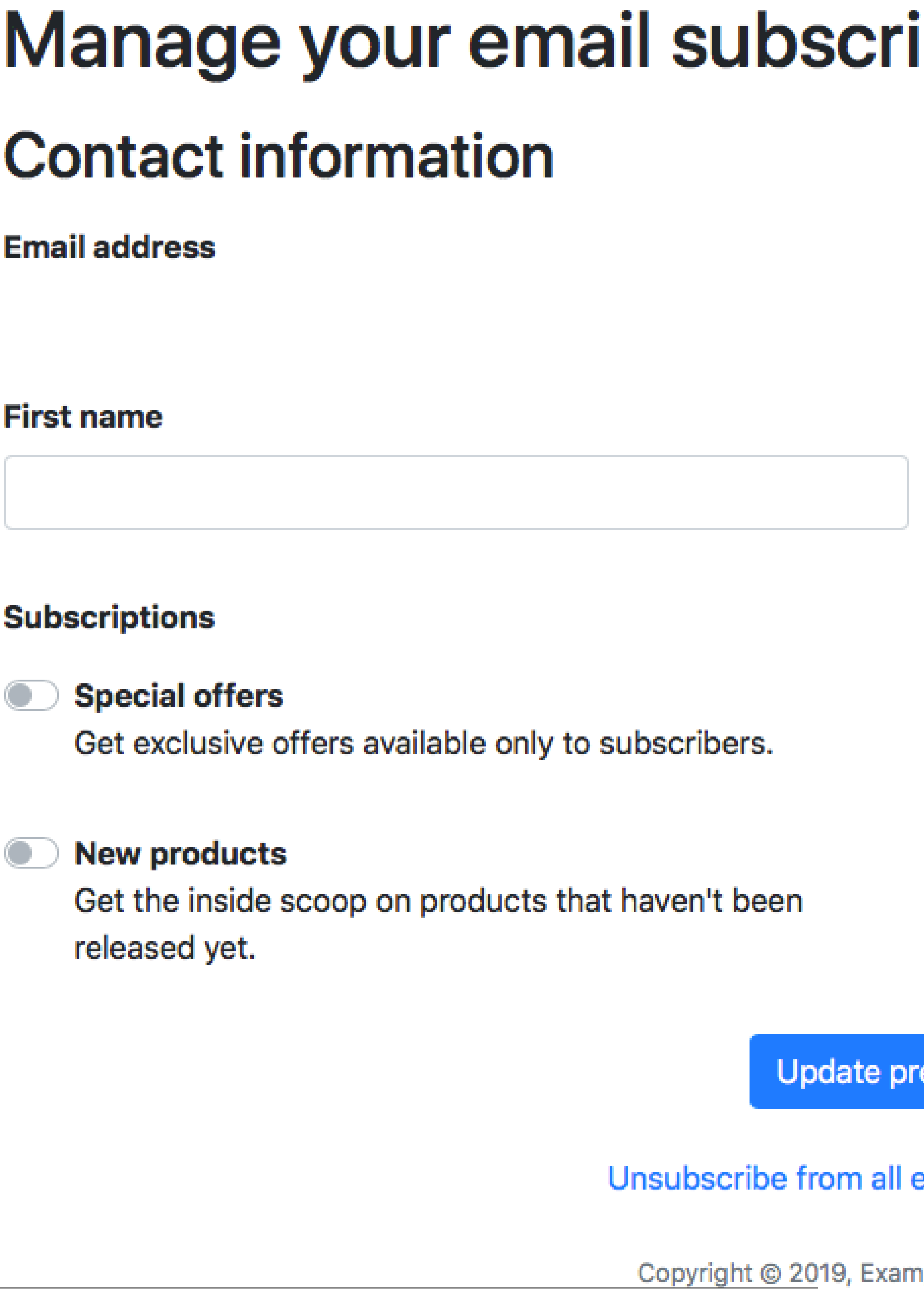
After you publish the form, you should test it to confirm that it works properly.

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. 78), you see a form that resembles the example in the following image.



Manage your email subscri

Contact information

Email address

First name

Subscriptions

Special offers

Get exclusive offers available only to subscribers.

New products

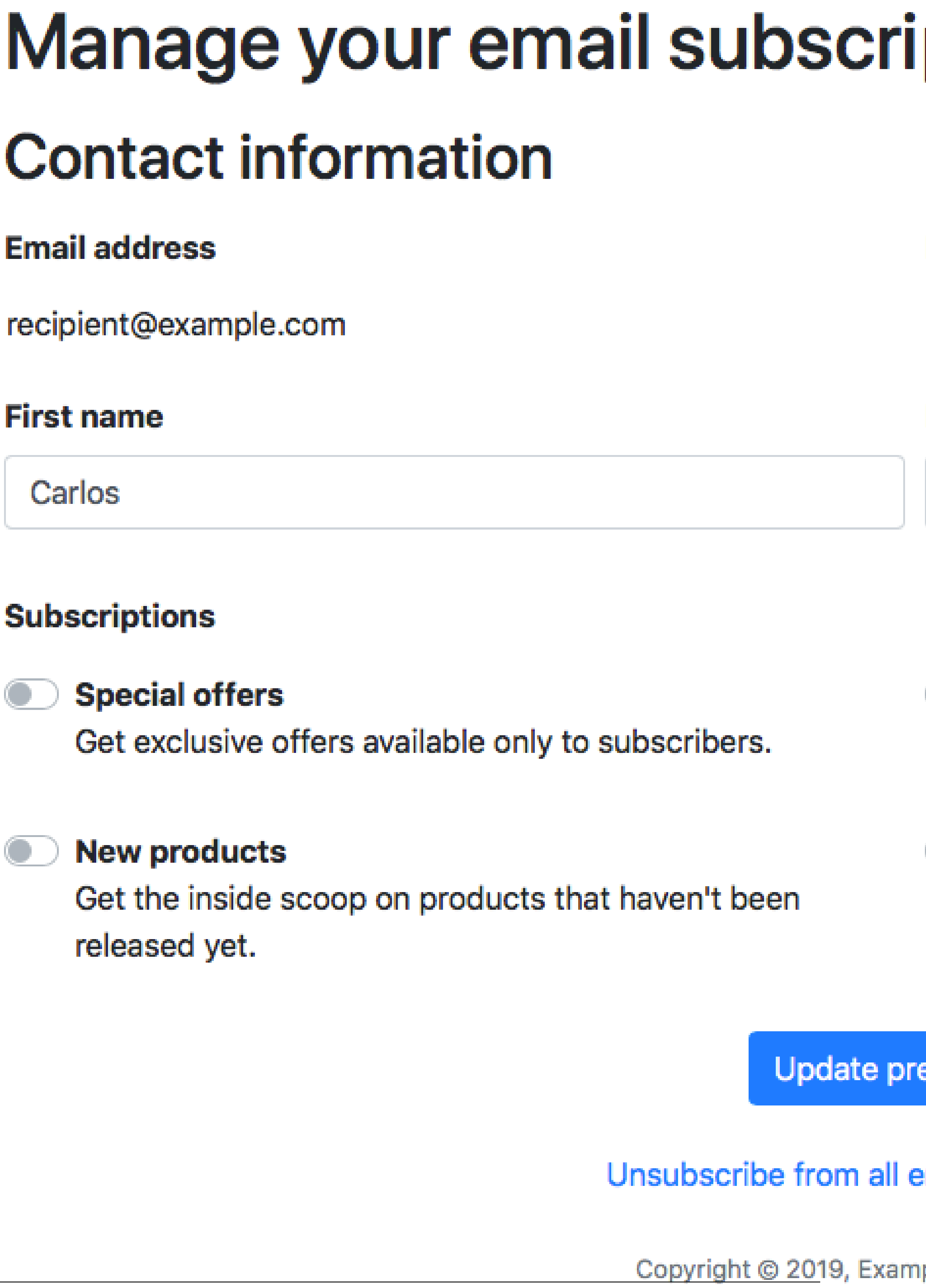
Get the inside scoop on products that haven't been released yet.

Copyright © 2019, Exam

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 subscri

Contact information

Email address

recipient@example.com

First name

Carlos

Subscriptions

Special offers
Get exclusive offers available only to subscribers.

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

Copyright © 2019, Examp

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.

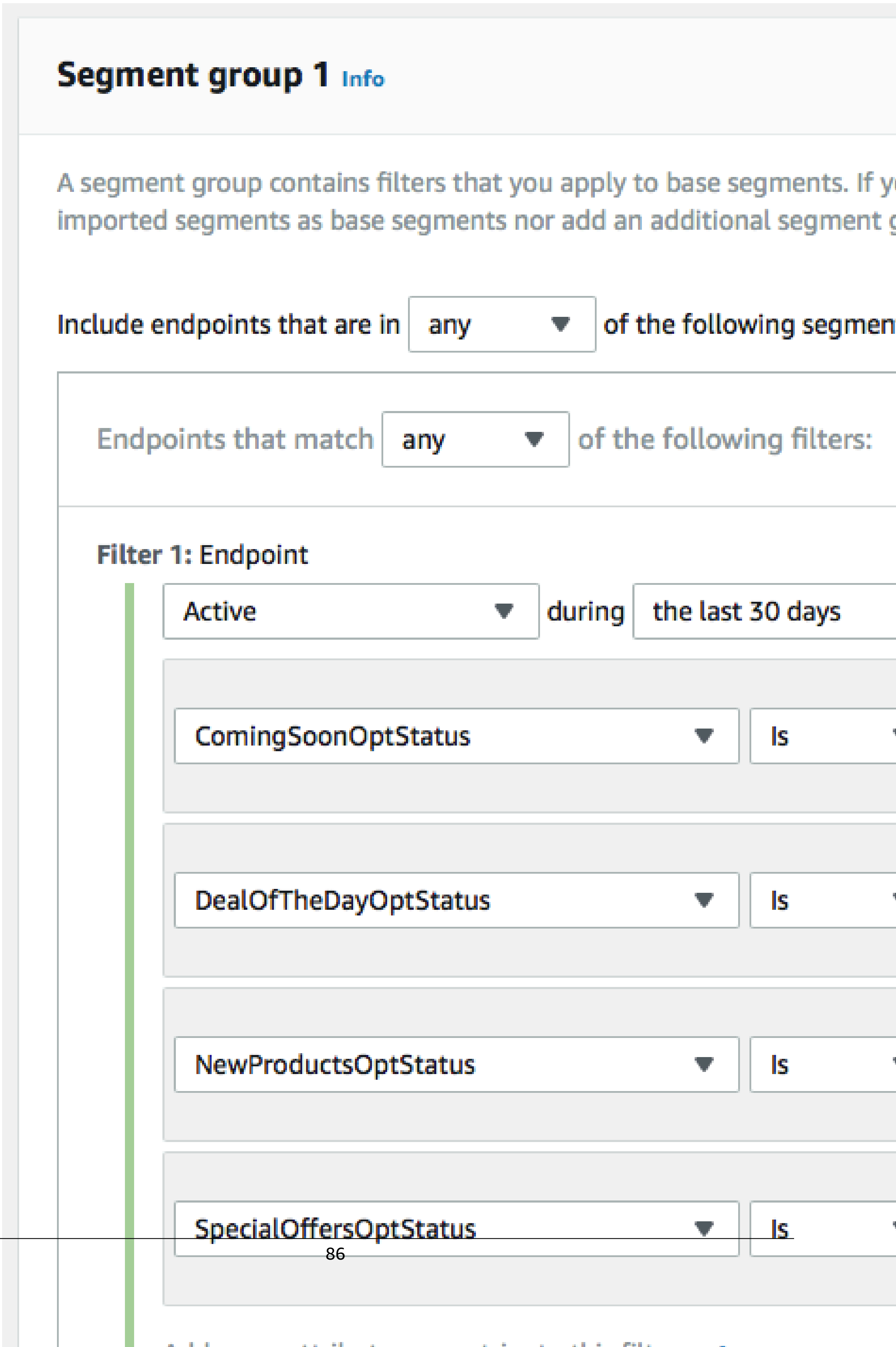
Step 6.4.2: Check the Opt Status of the Test Endpoint

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.
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 y
imported segments as base segments nor add an additional segment g

Include endpoints that are in any of the following segmen

Endpoints that match any of the following filters:

Filter 1: Endpoint

Active during the last 30 days

ComingSoonOptStatus Is

DealOfTheDayOptStatus Is

NewProductsOptStatus Is

SpecialOffersOptStatus Is

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
```

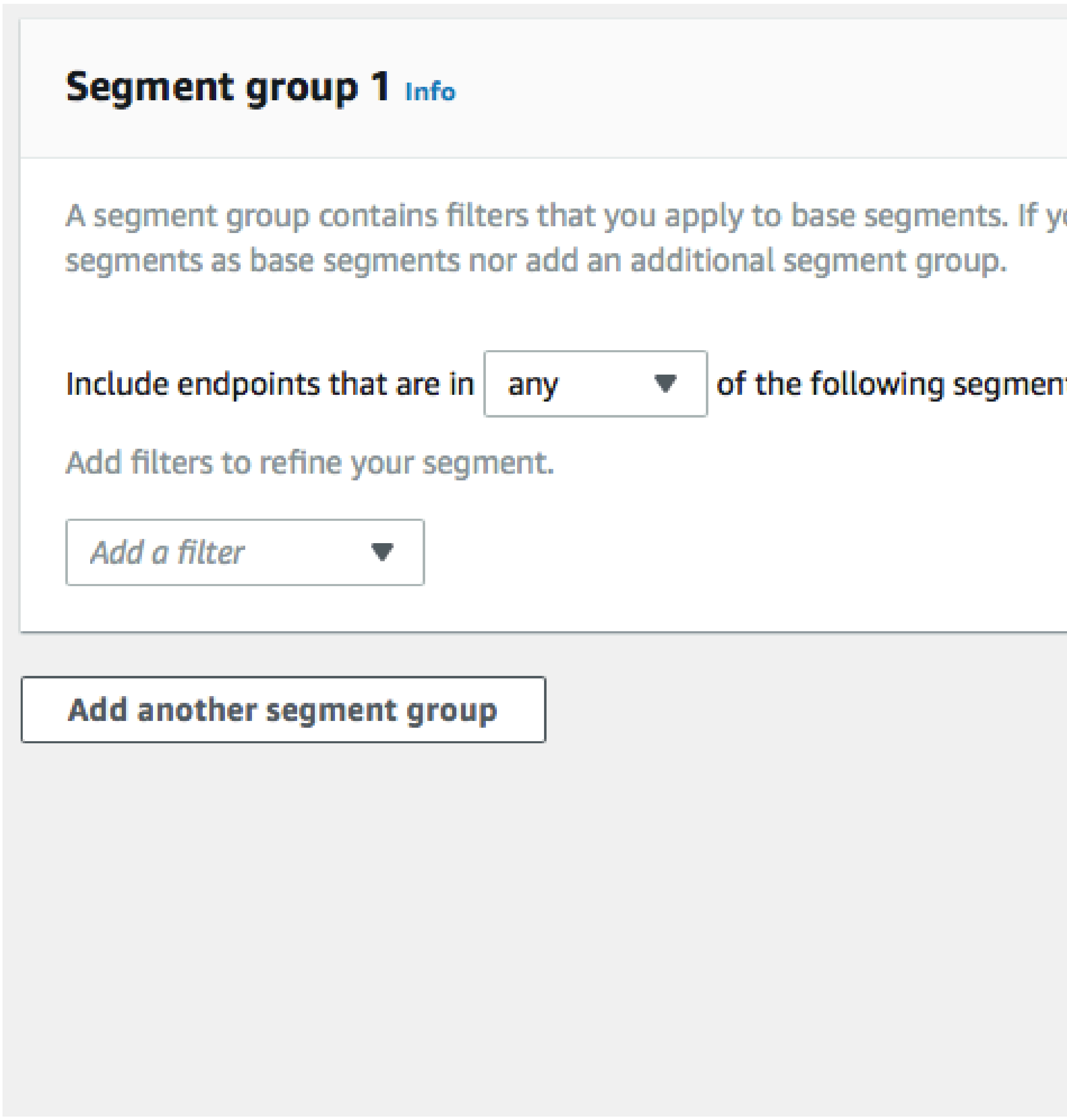
   When you add these attributes to the end of the URL and press Enter, the page updates to include the information in the attribute string.
3. On the form, choose **Unsubscribe from all email communications**.
4. Wait for approximately one minute, and then proceed to the next section.

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.
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 y
segments as base segments nor add an additional segment group.

Include endpoints that are in any of the following segmen

Add filters to refine your segment.

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. 61). 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. 61) are equal to the project ID and Region values that you specified in the IAM policy in Step 3.1 (p. 59). 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. 72).

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. 56) 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. 52).
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. 89).
    - 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. 80).

  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. 52).
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. 180).

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.

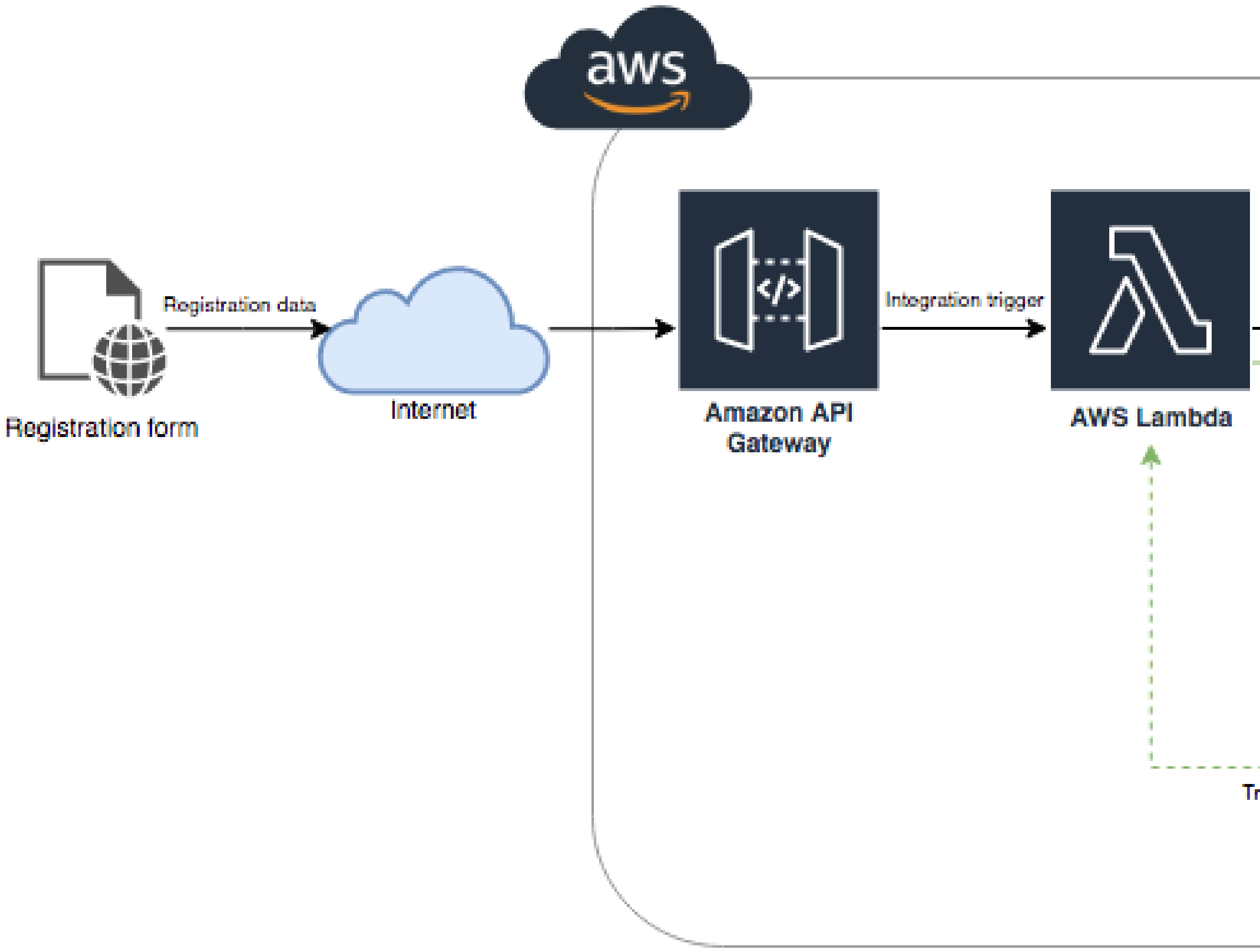
Tutorial: Setting Up an SMS Registration System

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. 95)

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 AWS Management 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. 95)

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. 95).

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.

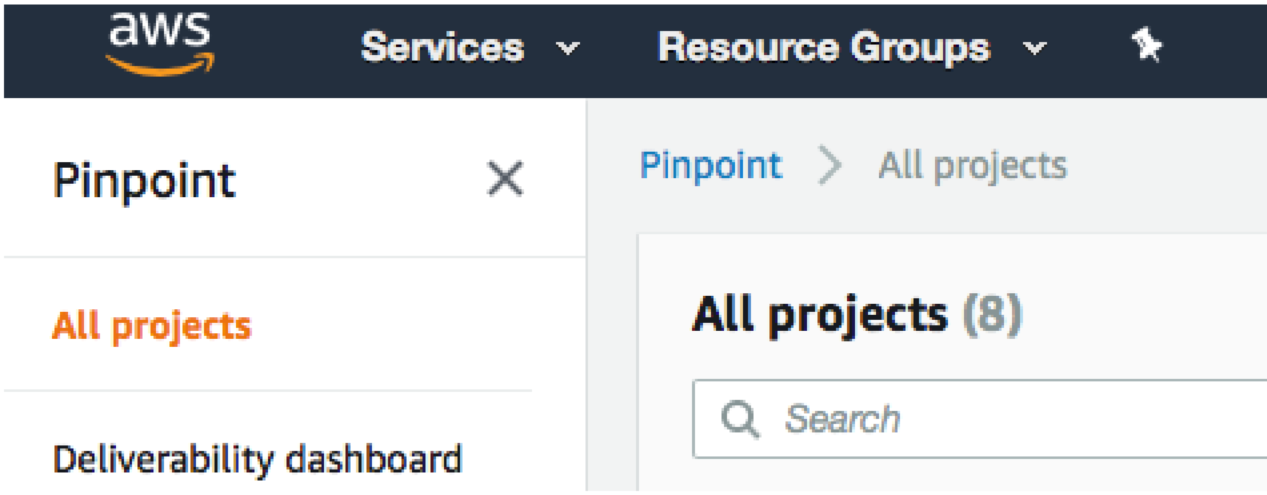
The steps involved in creating a new project differ depending on whether you've created a project in Amazon Pinpoint previously.

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 AWS Management 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.

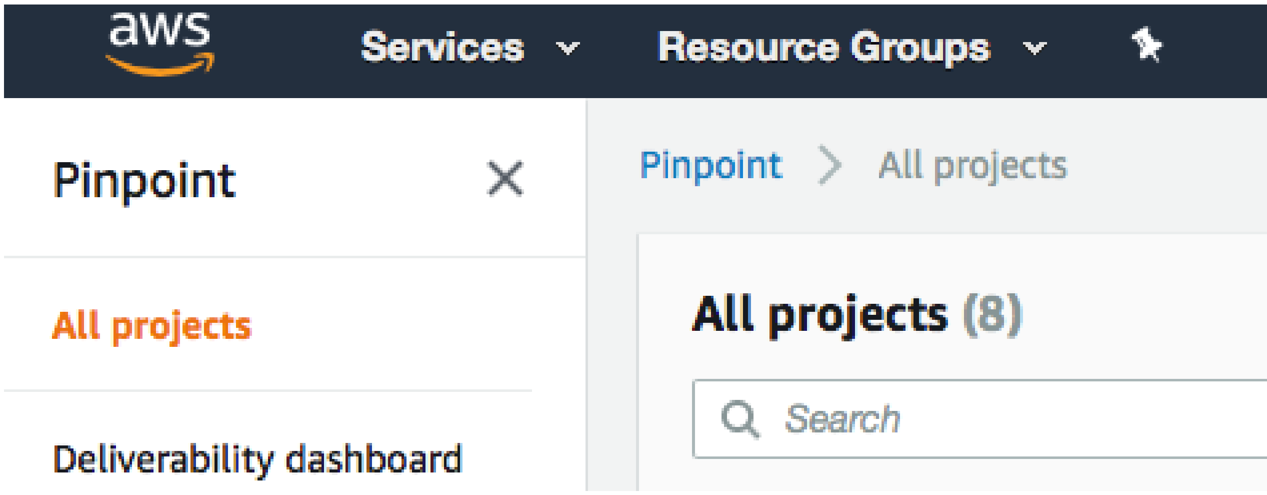
   Tip
   You need to use this ID in a few different places in this tutorial. Keep the project ID in a convenient place so that you can copy it later.

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 AWS Management 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. 98)

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 AWS Management 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"
            ],
            "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. 95) 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. 100)

Step 3: Create Lambda Functions

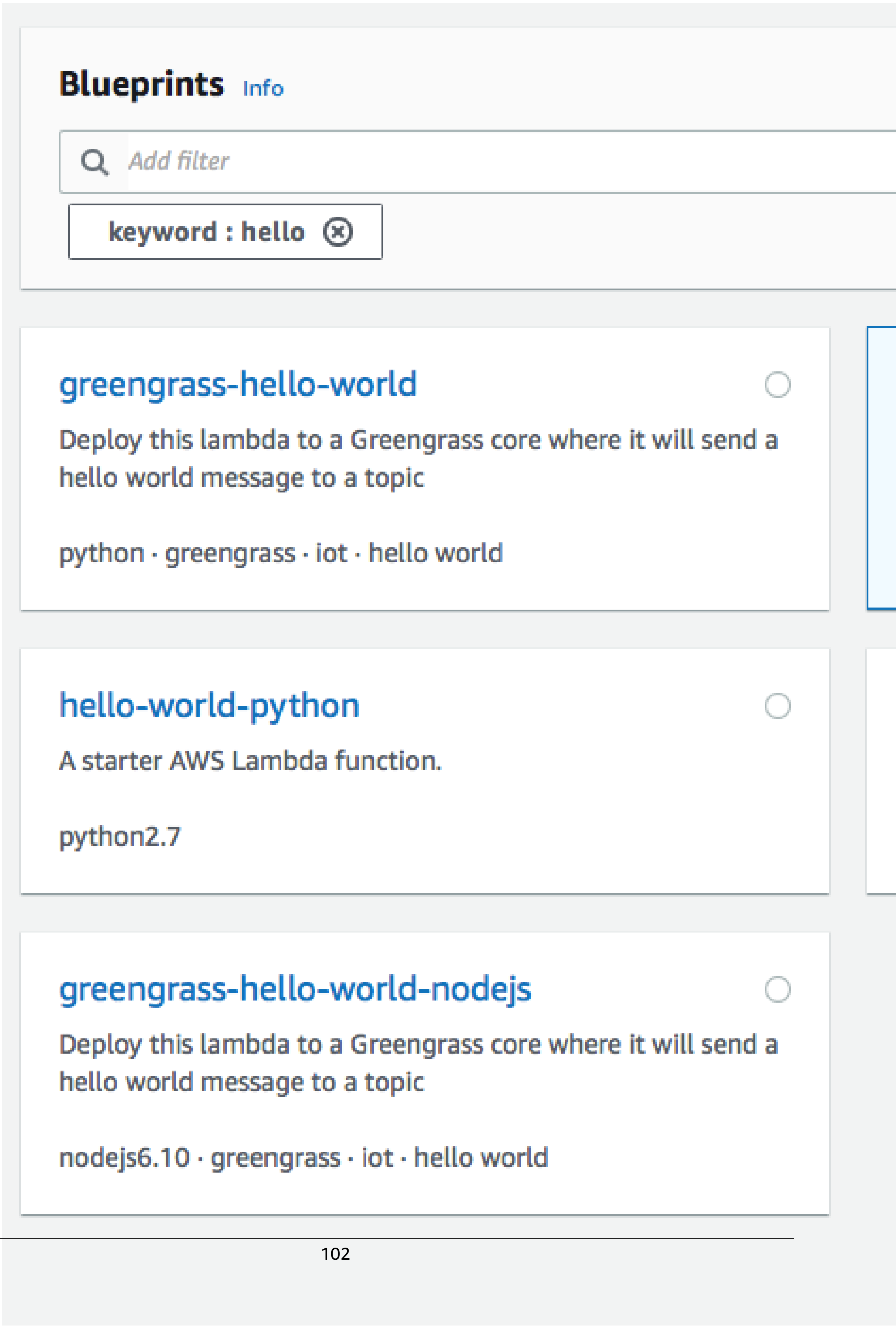
This solution uses two Lambda functions. This section shows you how to create and configure these functions. Later, you set up API Gateway and Amazon Pinpoint to execute these functions when certain events occur. Both of these functions create and update endpoints in the Amazon Pinpoint project that 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 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**.



Blueprints Info

Add filter

keyword : hello

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-python

A starter AWS Lambda function.

python2.7

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. 100).

   When you finish, choose **Create function**.
6. Delete the sample function in the code editor, and then paste the following code:

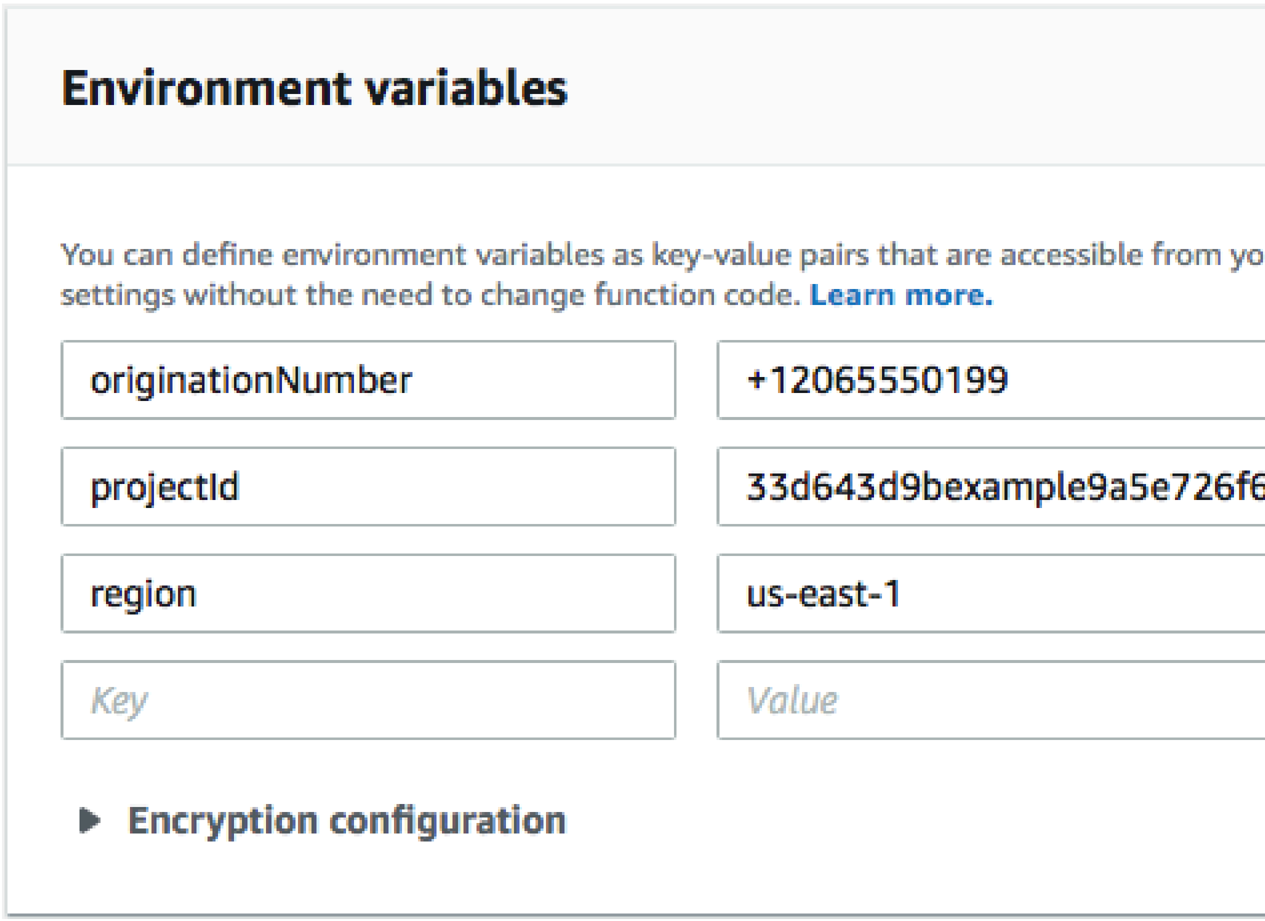
```
var AWS = require('aws-sdk');
var pinpoint = new AWS.Pinpoint({region: process.env.region});

// 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. 97).

     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. 95).
   - 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 yo
settings without the need to change function code. Learn more.

originationNumber	+12065550199
projectId	33d643d9bexample9a5e726f6
region	us-east-1
Key	Value

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"
}
```

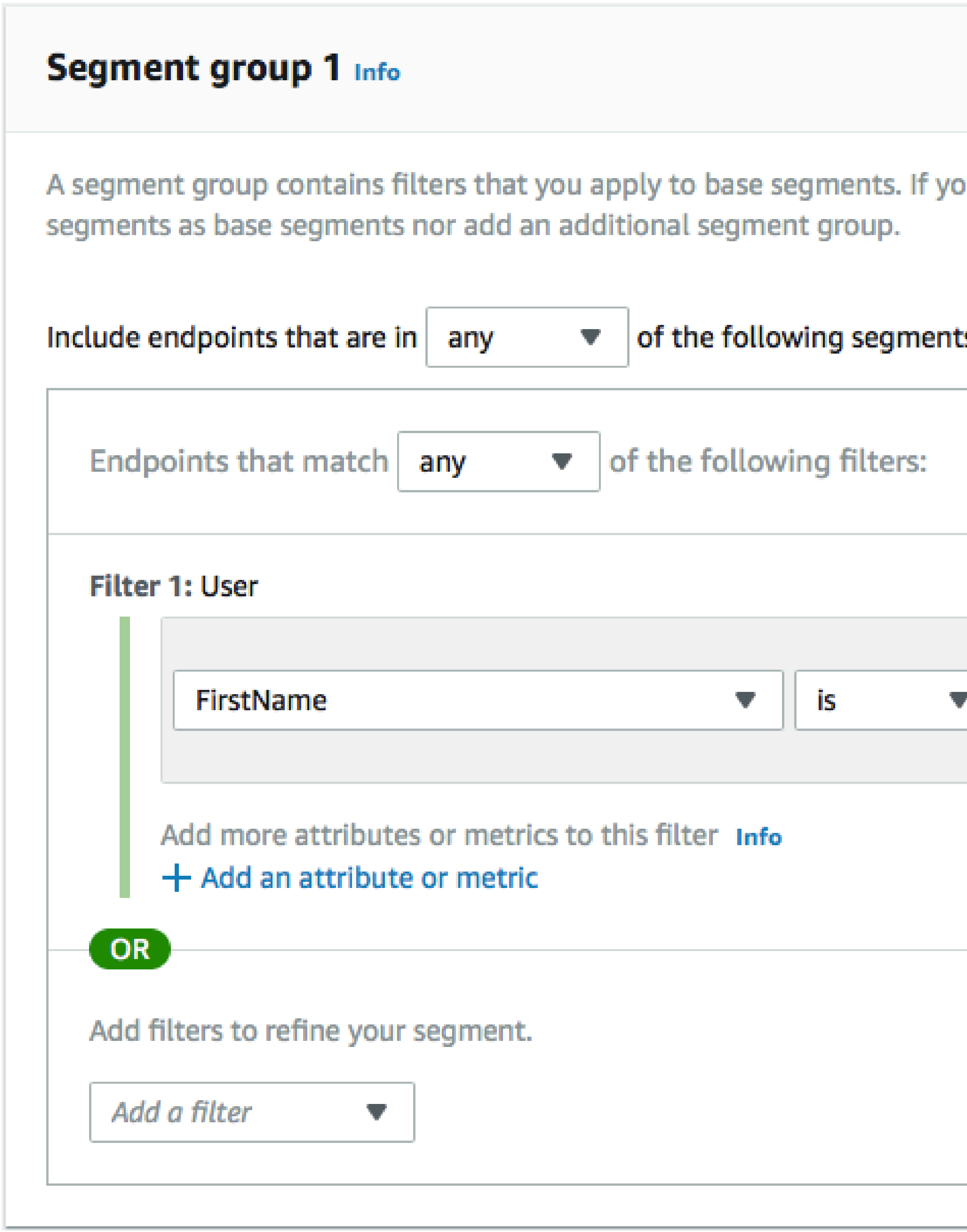
   - In the preceding code example, replace the values of the `destinationNumber`, `firstName`, and `lastName` attributes with the values that you want to use for testing, such as your personal contact details. When you test this function, it sends an SMS message to the phone number that you specify in 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. 95).
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 yo
segments as base segments nor add an additional segment group.

Include endpoints that are in any of the following segment

Endpoints that match any of the following filters:

Filter 1: User

FirstName is

Add more attributes or metrics to this filter Info
+ Add an attribute or metric

OR

Add filters to refine your segment.

Add a filter

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. 97), 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. 100).

   When you finish, choose **Create function**.
6. Delete the sample function in the code editor, and then paste the following code:

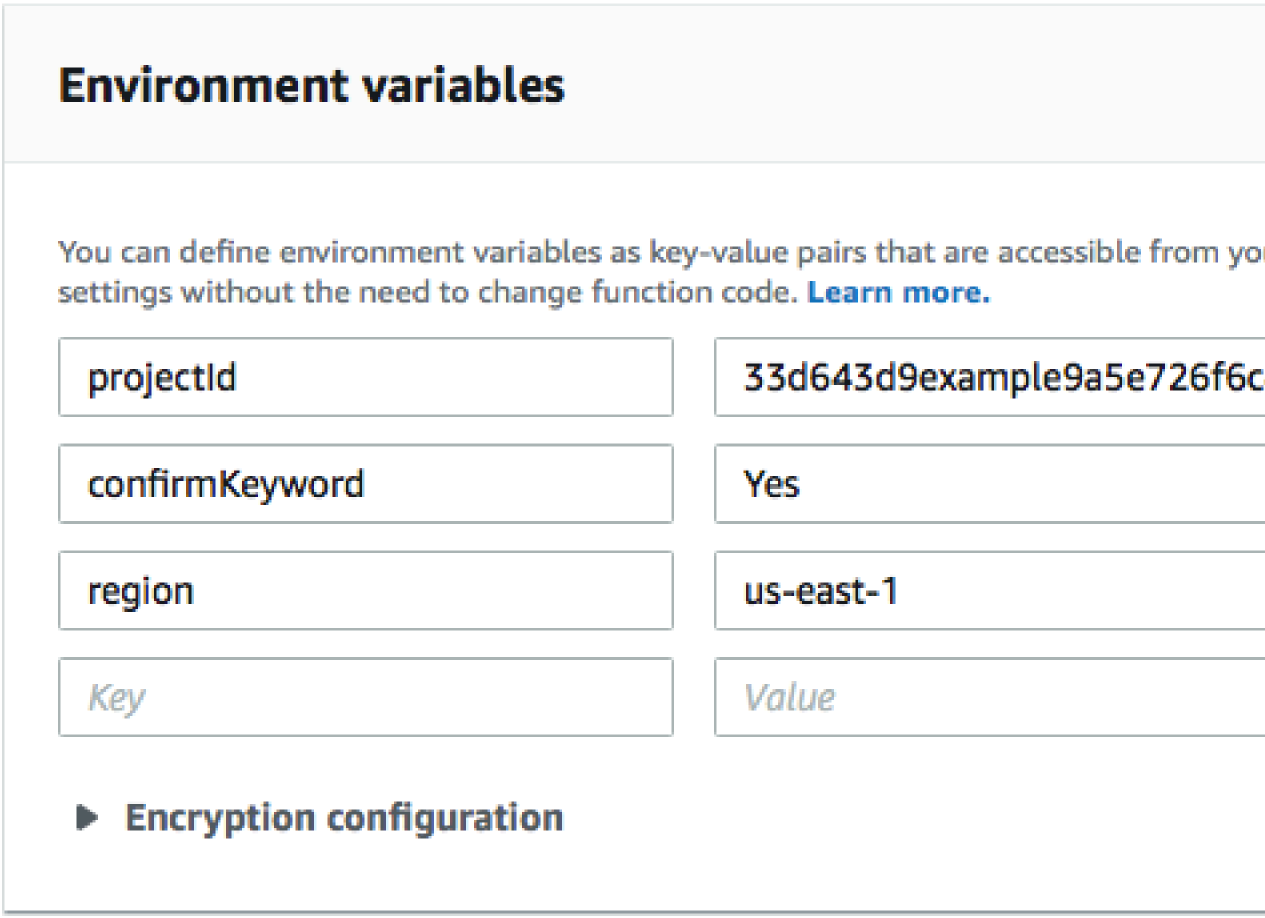
```
var AWS = require('aws-sdk');
var projectId = process.env.projectId;
var confirmKeyword = process.env.confirmKeyword.toLowerCase();
var pinpoint = new AWS.Pinpoint({region: process.env.region});

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 **`appId`**. Next, set the value to the unique ID of the project that you created in Step 1.1 (p. 95).
   - 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. 97).

     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 yo
settings without the need to change function code. Learn more.

projectId	33d643d9example9a5e726f6c
confirmKeyword	Yes
region	us-east-1
Key	Value

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. 98). 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. 95).
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. 112)

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. 100).

Step 4.1: Create the API

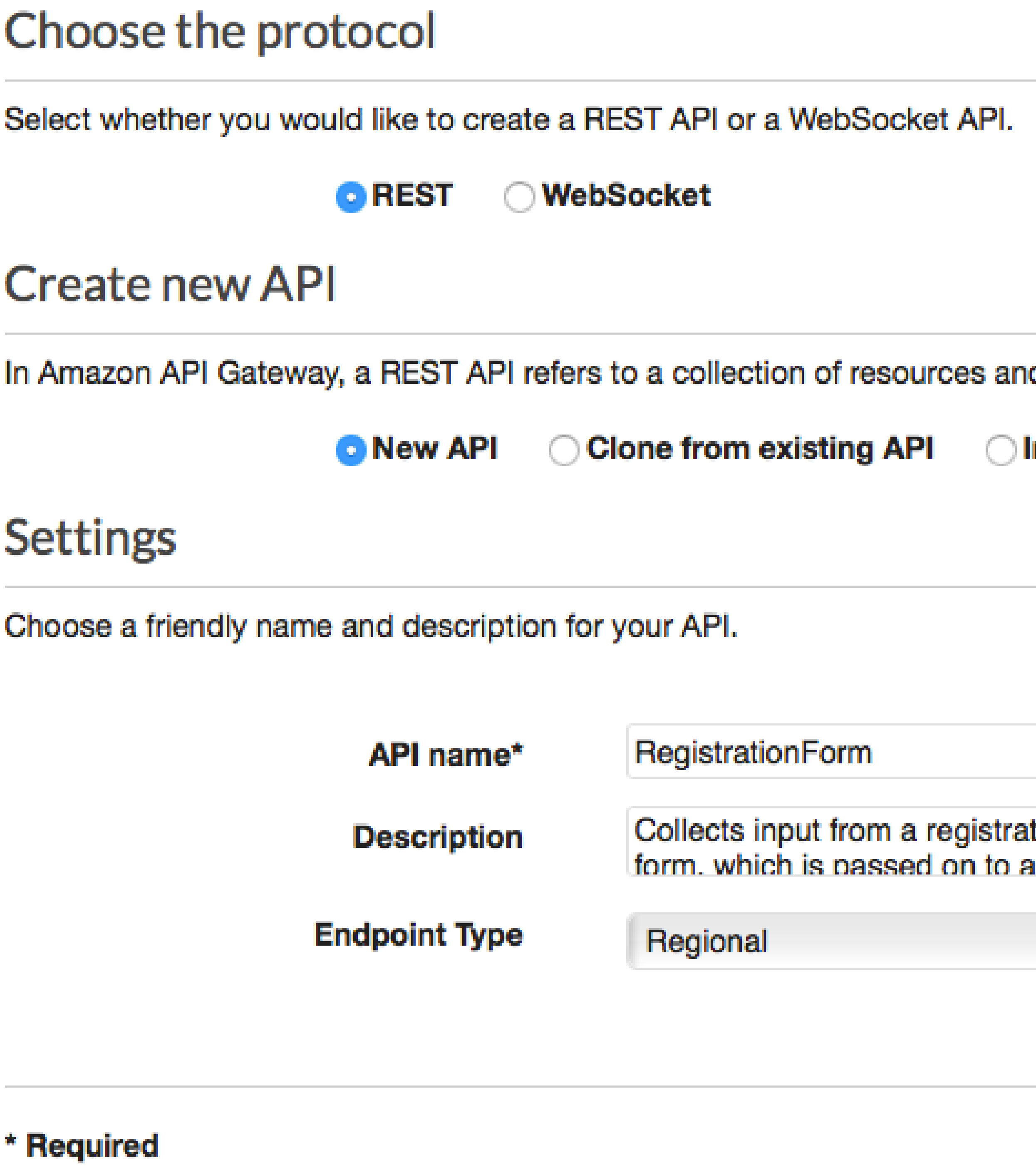
First, you have to create a new API in API Gateway. The following procedures show you how to create a new REST API.

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 anc

[✓ New API] Clone from existing API I

Settings

Choose a friendly name and description for your API.

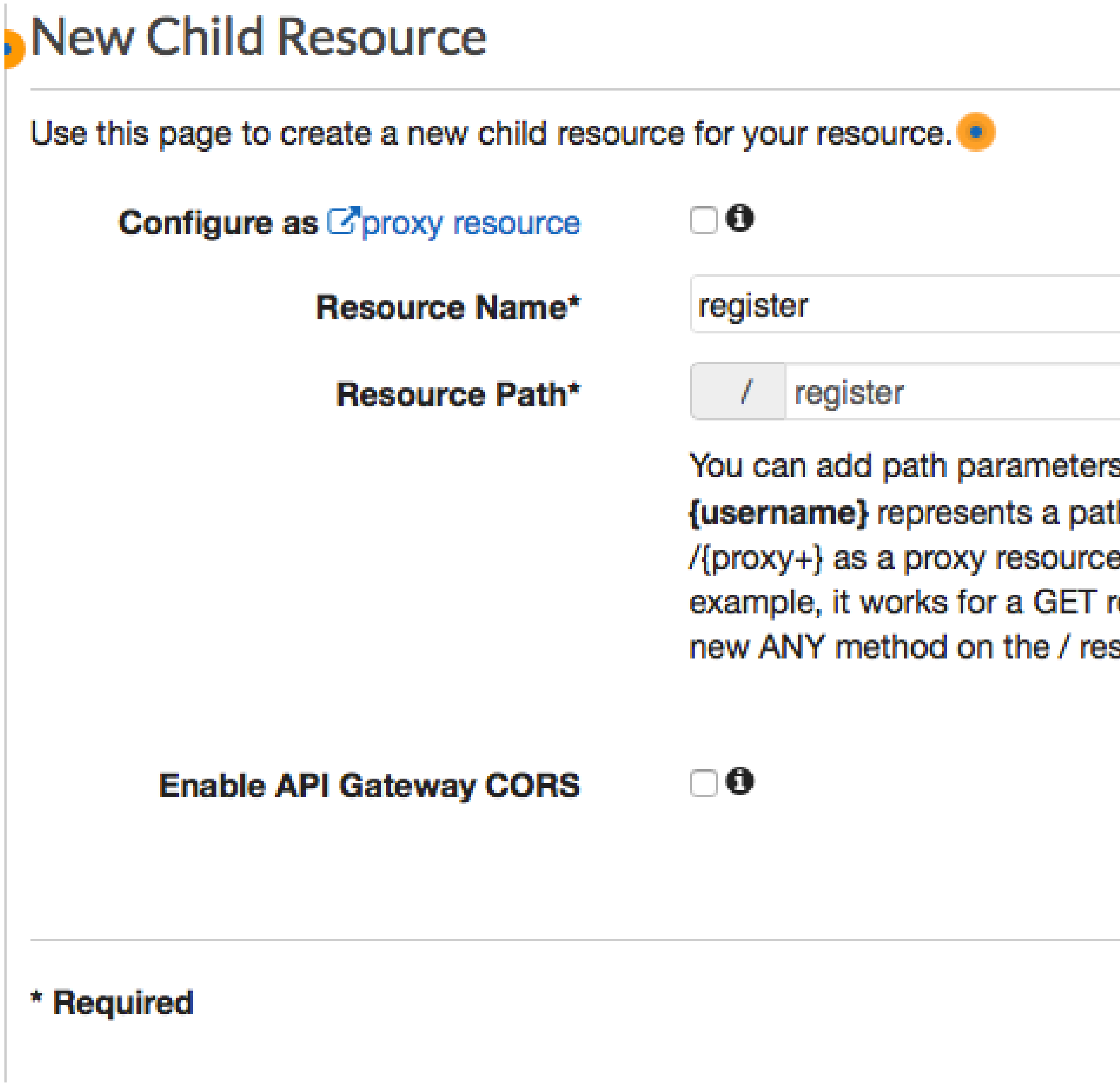
API name*	RegistrationForm
Description	Collects input from a registrat form. which is passed on to a
Endpoint Type	Regional

* 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

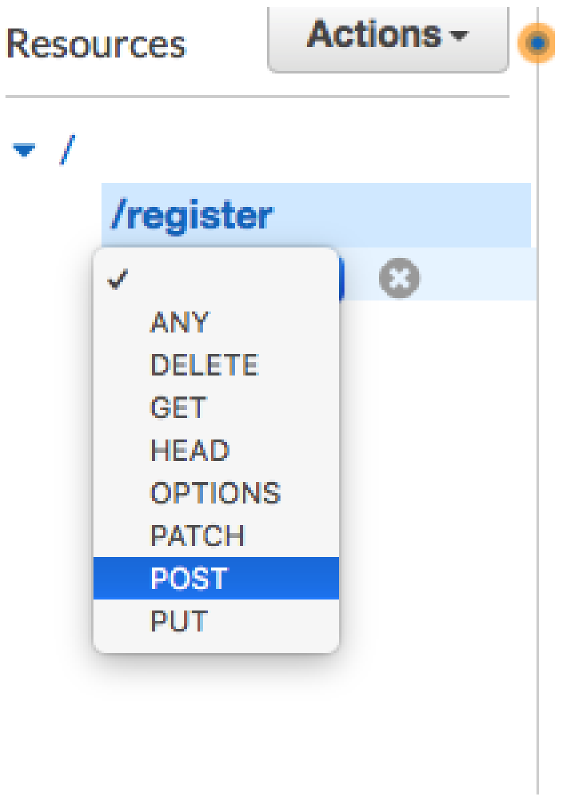
Resource Path* / register

You can add path parameters
{username} represents a pat
/{proxy+} as a proxy resource
example, it works for a GET r
new ANY method on the / res

Enable API Gateway CORS

* Required

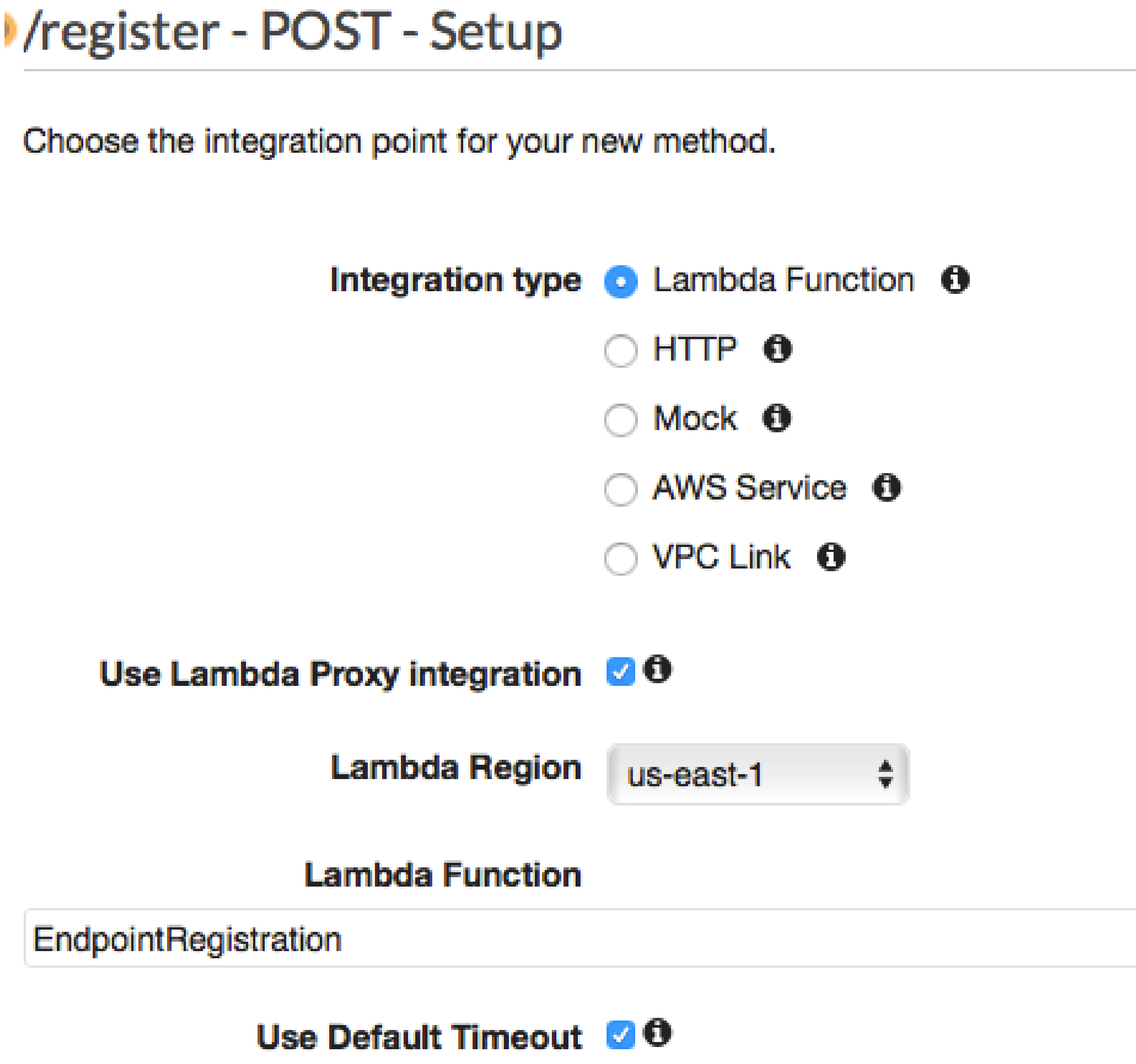
2. On the **Actions** menu, choose **Create Method**. From the menu that appears, choose **POST**, as shown in the following image. Then choose the **check mark** ( ) button.



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. 100).

   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.

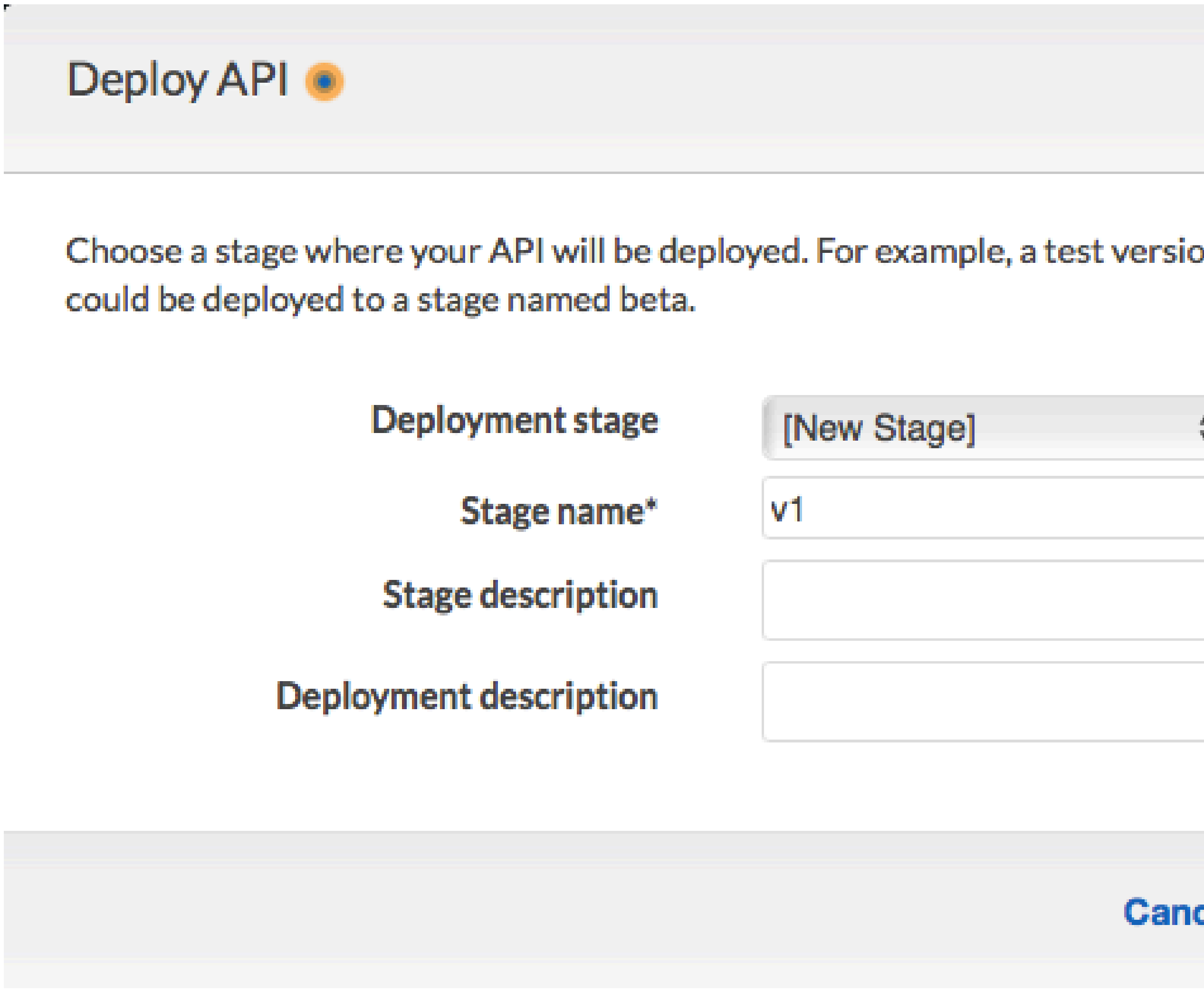
Step 4.3: Deploy the API

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.

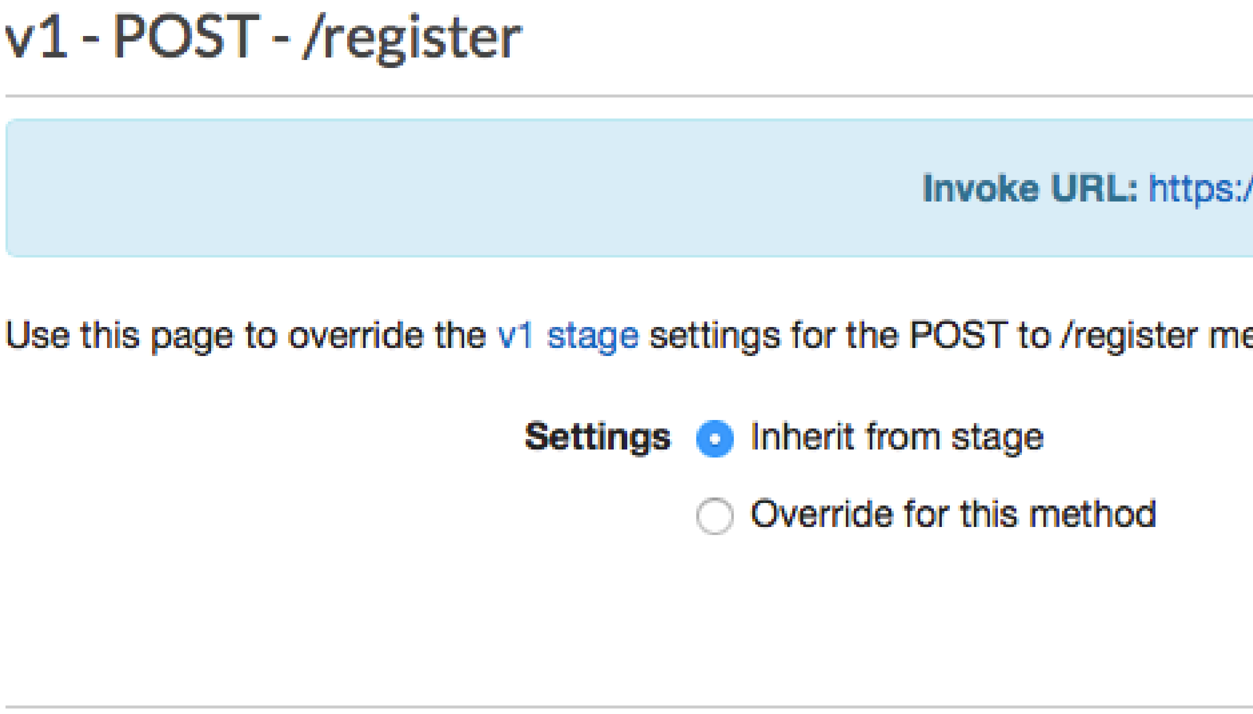
   

   Deploy API

   Choose a stage where your API will be deployed. For example, a test versio
   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.

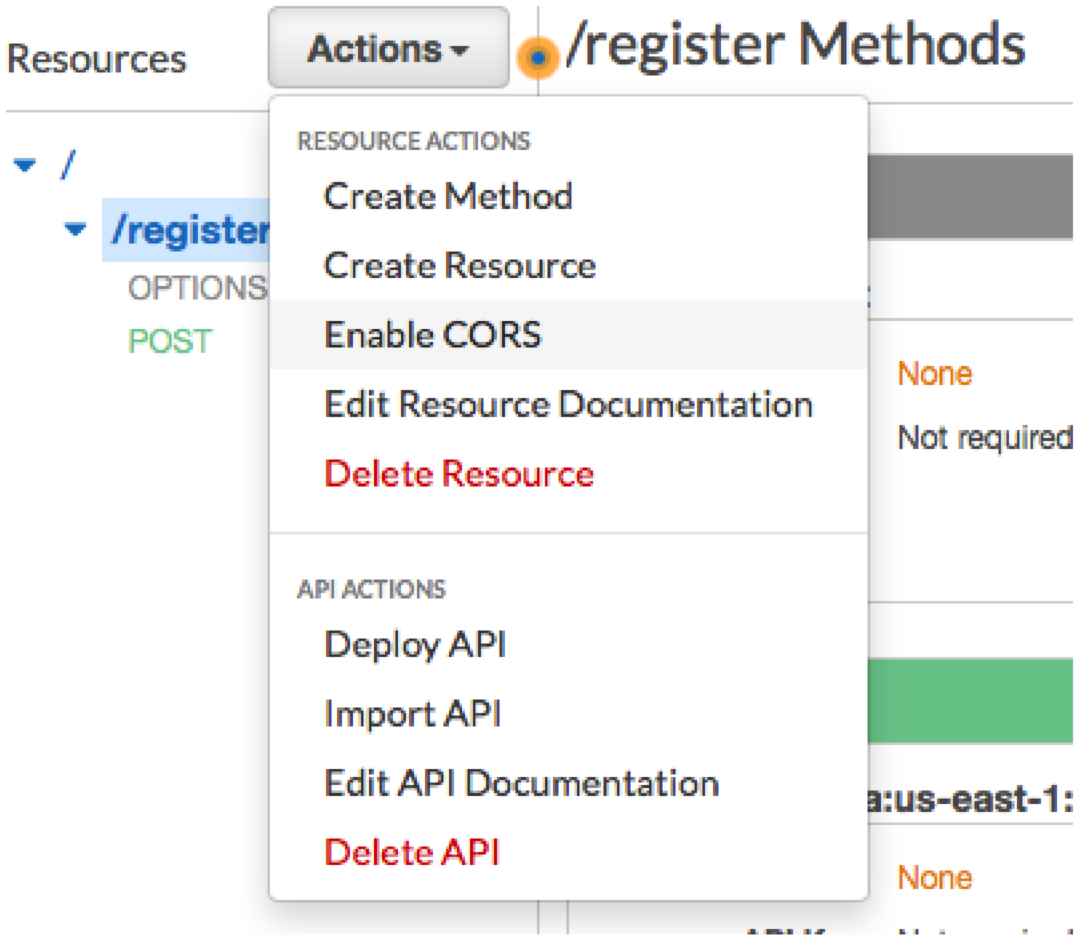

v1 - POST - /register

Invoke URL: https:/

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

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. 119)

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. 112).

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'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't send any additional messages until we receive
 your confirmation.</p><p>If you decide you don'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. 116).
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 Route 53, Amazon Simple Storage Service (Amazon S3), and Amazon CloudFront. For more information, see Host a Static Website.

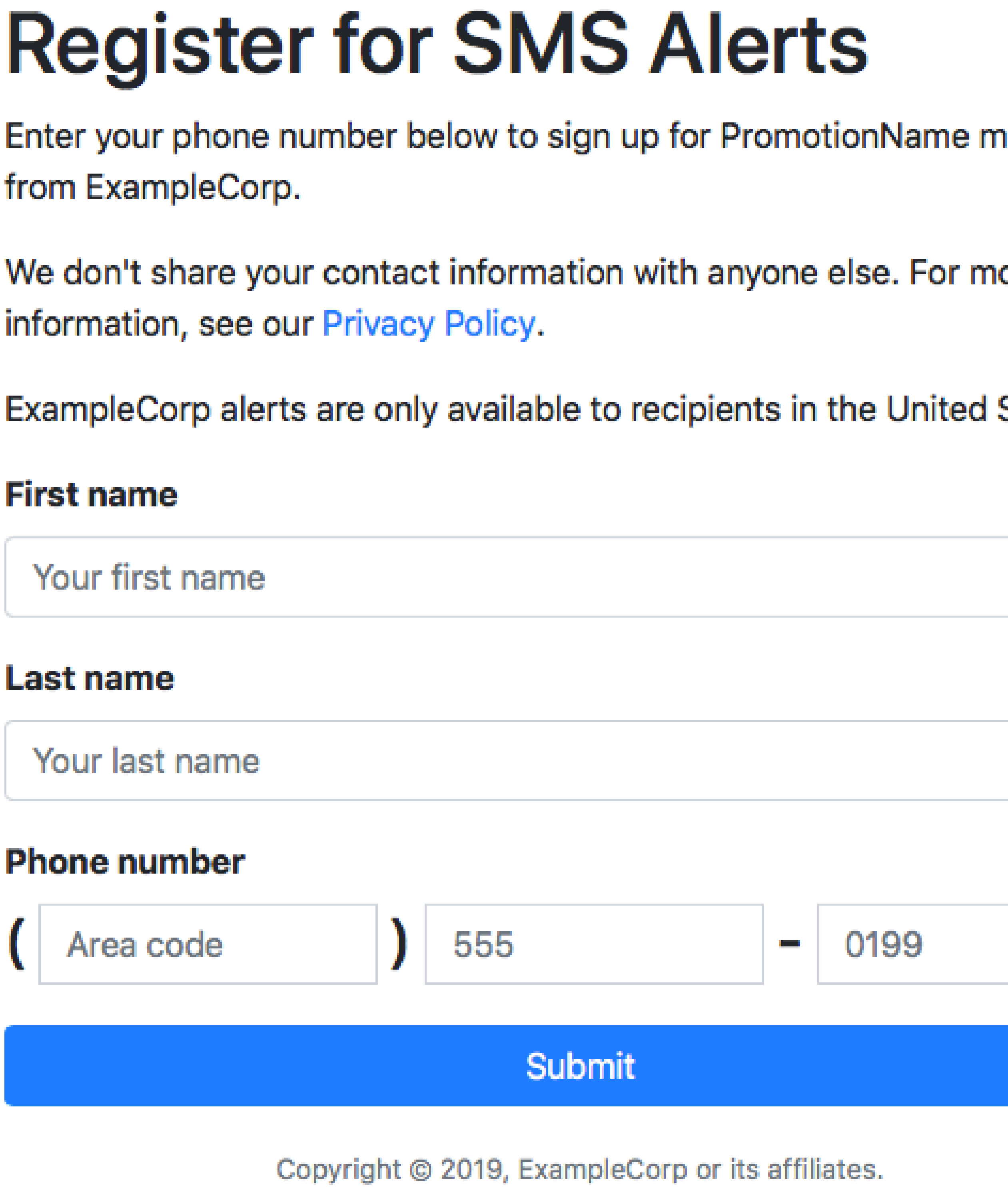
If you use another web hosting provider, consult the provider's documentation for more information about publishing webpages.

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. 121), 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 m
from ExampleCorp.

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

ExampleCorp alerts are only available to recipients in the United S

First name

Your first name

Last name

Your last name

Phone number

(Area code) 555 - 0199

Submit

Copyright © 2019, ExampleCorp or its affiliates.

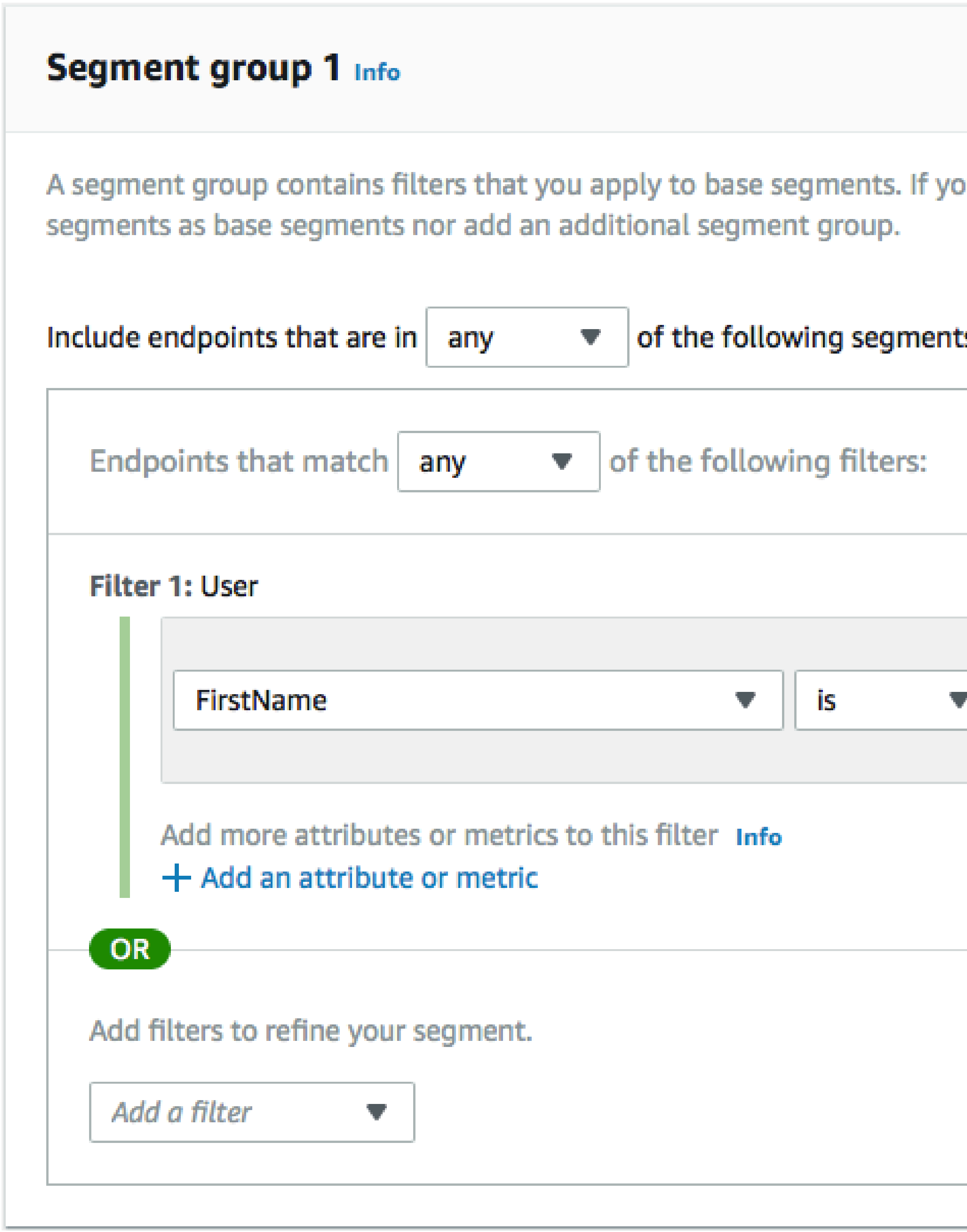
2. Enter your contact information in the **First name**, **Last name**, and **Phone number** fields.

   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. 100), 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. 95).
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 yo
segments as base segments nor add an additional segment group.

Include endpoints that are in any of the following segment

Endpoints that match any of the following filters:

Filter 1: User

FirstName is

Add more attributes or metrics to this filter Info

Add filters to refine your segment.

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. 98). 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. 129) 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. 130).

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 1.11.63 or later
- AWS SDK for .NET version 3.3.27.0 or later

- AWS SDK for PHP version 3.20.1
- AWS SDK for Python (Boto 3) 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 Get Started (JavaScript) or Get Started (React Native) in the AWS Amplify JavaScript 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. 131).

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. 130).

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. 132) 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. 225).

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. 130).

- Update your application to register endpoints. See Registering Endpoints in Your Application (p. 131).

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 appplication 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. 133).

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 Apple Push Notification service (APNs).
- For Android apps, you provide an API Key and a sender ID, 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 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 APNs for Push Notifications in the iOS SDK documentation.

Setting Up Android Push Notifications

This section describes how to obtain the credentials required to send push notifications to Android apps. The platform notification service that you can use for these push notifications is Firebase Cloud Messaging (FCM), which replaces Google Cloud Messaging (GCM). You can use your FCM 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 Setting Up FCM/GCM for Push Notifications in 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. 131) 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. 131).

Examples

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

AWS CLI

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

Example Update Endpoint Command

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

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

Where:

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

Example Endpoint Request File

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

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

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

AWS SDK for Java

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

Example Code

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

```

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

public class AddExampleEndpoint {

 public static void main(String[] args) {

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

  if (args.length < 1) {
      System.out.println(USAGE);
      System.exit(1);
  }
  String applicationId = args[0];

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

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

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

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

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

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

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

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

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

 }
}
```

HTTP

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

Example PUT Endpoint Request

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

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

Where:

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

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

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

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

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

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

Related Information

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

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

For the limits that apply to endpoints, such as the number of attributes you can assign, see the section called “Endpoint Limits” (p. 278).

Associating Users with Amazon Pinpoint Endpoints

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

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

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

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

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

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

Examples

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

AWS CLI

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

Example Update Endpoint Command

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

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

Where:

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

Example Endpoint Request File

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

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

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

AWS SDK for Java

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

Example Code

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

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

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

public class AddExampleUser {

    public static void main(String[] args) {

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

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

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

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

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

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

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

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

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

    }
}
```

HTTP

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

Example Put Endpoint Request with User Definition

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

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

Where:

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

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

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

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

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

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

Related Information

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

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

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

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

For the limits that apply to endpoints, including the number of user attributes you can assign, see the section called “Endpoint Limits” (p. 278).

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. 150) instead.

Examples

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

AWS CLI

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

Example Update Endpoints Batch Command

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

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

Where:

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

Example Endpoint Batch Request File

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

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

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

AWS SDK for Java

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

Example Code

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

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

import java.util.Arrays;

public class AddExampleEndpoints {

    public static void main(String[] args) {

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

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


        String applicationId = args[0];

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

    }

}
```

HTTP

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

Example Put Endpoints Request

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

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

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

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

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

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

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

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

Related Information

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

Importing Endpoints into Amazon Pinpoint

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

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

Each import job can transfer up to 1 GB of data. In a typical job, where each endpoint is 4 KB or less, you could import around 250,000 endpoints. You can run up to two concurrent import jobs per AWS account. If you need more bandwidth for your import jobs, you can submit a service limit increase request with AWS Support. For more information, see Requesting a Limit Increase (p. 285).

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. 267).

Examples

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

Files with Endpoint Definitions

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

CSV

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

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

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

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

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

JSON

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

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

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

Import Job Requests

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

AWS CLI

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

Example S3 CP Command

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

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

Where:

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

Example Create Import Job Command

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

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

Where:

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

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

The response includes details about the import job:

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

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

Example Get Import Job Command

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

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

Where:

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

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

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

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

AWS SDK for Java

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

Example Code

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

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

```

package com.amazonaws.examples.pinpoint;

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

public class ImportEndpoints {

    public static void main(String[] args) {

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

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

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

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

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

    }

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

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

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

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

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

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

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

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

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

        try {

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

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

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

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

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

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

    }

}
```

HTTP

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

Example S3 PUT Object Request

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

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

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

Where:

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

Example POST Import Job Request

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

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

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

Where:

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

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

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

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

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

Example GET Import Job Request

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

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

Where:

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

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

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

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

Related Information

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

Deleting Endpoints from Amazon Pinpoint

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

Examples

The following examples show you how to delete an endpoint.

AWS CLI

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

Example Delete Endpoint Command

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

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

Where:

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

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

AWS SDK for Java

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

Example Code

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

```

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

import java.util.Arrays;

public class DeleteEndpoints {

    public static void main(String[] args) {

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

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

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

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

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

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

HTTP

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

Example DELETE Endpoint Request

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

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

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

Where:

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

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

Accessing Audience Data in Amazon Pinpoint

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

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

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

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

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

Before you can look up endpoints, you must add them to your Amazon Pinpoint project. To add endpoints, see Defining Your Audience to Amazon Pinpoint (p. 136).

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. 165).

Examples

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

AWS CLI

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

Example Get Endpoint Command

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

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

Where:

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

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

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

AWS SDK for Java

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

Example Code

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

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

public class LookUpEndpoint {

    public static void main(String[] args) {

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

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

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

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

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

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

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

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

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

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

HTTP

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

Example GET Endpoint Request

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

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

Where:

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

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

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

Related Information

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

Exporting Endpoints from Amazon Pinpoint

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

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

Before You Begin

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

- An Amazon S3 bucket. To create a bucket, see Create a Bucket in the *Amazon Simple Storage Service Getting Started Guide*.
- An AWS Identity and Access Management (IAM) role that grants Amazon Pinpoint write permissions for your Amazon S3 bucket. To create the role, see IAM Role for Exporting Endpoints or Segments (p. 268).

Examples

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

AWS CLI

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

Example Create Export Job Command

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

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

Where:

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

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

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

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

Example Get Export Job Command

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

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

Where:

- *application-id* is the ID the Amazon Pinpoint project that you exported the endpoints from.

- *job-id* is the ID of the job that you're checking.

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

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

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

Example S3 CP Command

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

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

Where:

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

AWS SDK for Java

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

Example Code

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

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

```

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

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

public class ExportEndpoints {

    public static void main(String[] args) {

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

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

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

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

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

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

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

        // Downloads the exported endpoints files to the specified directory.
        downloadFromS3(s3BucketName, endpointFileKeys, downloadDirectory);
    }
    public static List<String> exportEndpointsToS3(String s3BucketName, String
 iamExportRoleArn,
                                                   String applicationId) {

        // The S3 path that Amazon Pinpoint exports the endpoints to.
        SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd-
HH_mm:ss.SSS_z");
        String endpointsKeyPrefix = "exports/" + applicationId + "_" +
 dateFormat.format(new Date
                ());
        String s3UrlPrefix = "s3://" + s3BucketName + "/" + endpointsKeyPrefix + "/";

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

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

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

        List<String> objectKeys = null;

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

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

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

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

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

        return objectKeys;
    }

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

        GetExportJobResult getExportJobResult;
        String jobStatus;

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

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

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

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

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

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

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

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

    }

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

        // Writes the contents of the S3 object to a file.
        File endpointsFile = new File(filePath.toAbsolutePath().toString());
        try (FileOutputStream fos = new FileOutputStream(endpointsFile);
             S3ObjectInputStream s3is = object.getObjectContent()) {
            byte[] read_buf = new byte[1024];
            int read_len = 0;
            while ((read_len = s3is.read(read_buf)) > 0) {
                fos.write(read_buf, 0, read_len);
            }
        } catch (IOException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

HTTP

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

Example POST Export Job Request

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

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

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

Where:

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

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

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

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

Example GET Export Job Request

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

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

Where:

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

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

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

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

Related Information

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

Listing Endpoint IDs with Amazon Pinpoint

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

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

1. Calls the example `exportEndpointsToS3` method from the example code in Exporting Endpoints from Amazon Pinpoint (p. 165). This method exports the endpoint definitions from an Amazon Pinpoint project. The endpoint definitions are added as gzip files to an Amazon S3 bucket.
2. Downloads the exported gzip files.
3. Reads the gzip files and obtains the endpoint ID from each endpoint's JSON definition.
4. Prints the endpoint IDs to the console.
5. Cleans up by deleting the files that Amazon Pinpoint added to Amazon S3.

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

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

public class ListEndpointIds {

    public static void main(String[] args) {

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

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

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

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

        if (args.length < 1) {
            System.out.println(USAGE);
            System.exit(1);
        }
        String applicationId = args[0];
        String s3BucketName = args[1];
        String iamExportRoleArn = args[2];

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

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

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

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

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

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

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

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

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

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

        return endpointIds;
    }

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

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

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

        // Reads each endpoint entry in the file and adds the ID to the list.
        try (GZIPInputStream gzipInputStream =
                     new GZIPInputStream(endpointsFile.getObjectContent());
             BufferedReader reader =
                     new BufferedReader(new InputStreamReader(
                                 gzipInputStream, "UTF-8"))) {
            String endpointString;
            while ((endpointString = reader.readLine()) != null) {
                JsonObject endpointJson = gson.fromJson(endpointString, JsonObject.class);
                endpointIdsFromFile.add(endpointJson
                        .getAsJsonPrimitive("Id")
                        .getAsString());
            }
        } catch (IOException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }

        return endpointIdsFromFile;
    }

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

        AmazonS3 s3Client = AmazonS3ClientBuilder.defaultClient();

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

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

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

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

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

Creating Segments

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

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

For more information, see Segments.

Topics

Building Segments

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

Building Segments With the AWS SDK for Java

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

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

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

public class PinpointSegmentSample {

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

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

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


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

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

        CreateSegmentResult createSegmentResult =
 client.createSegment(createSegmentRequest);

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

        return createSegmentResult.getSegmentResponse();
    }

}
```

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

```
Segment ID: 09cb2967a82b4a2fbab38fead8d1f4c4
```

Importing Segments

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

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

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

To create the IAM role, see IAM Role for Importing Endpoints or Segments (p. 267). For information about importing a segment by using the Amazon Pinpoint console, see Importing Segments in the *Amazon Pinpoint User Guide*.

Importing a Segment

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

```

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

public class ImportSegment {

    public static void main(String[] args) {

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

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

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

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

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

    }

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

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

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

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

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

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

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

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

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

        try {

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

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

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

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

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

    }

}
```

Customizing Segments with AWS Lambda

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

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

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

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

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

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

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

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

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

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

Note
You can also use the `CampaignHook` settings to assign a Lambda function that handles the message delivery. This type of function is useful for delivering messages through custom channels that Amazon Pinpoint doesn't support, such as social media platforms. For more information, see Creating Custom Channels with AWS Lambda (p. 286).

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` in the `Messages` resource in the Amazon Pinpoint API.
- `ApplicationId` – The ID of the Amazon Pinpoint project that the campaign belongs to.
- `CampaignId` – The ID of the Amazon Pinpoint project that the function is invoked for.
- `TreatmentId` – The ID of a campaign variation that's used for A/B testing.
- `ActivityId` – The ID of the activity that's being performed by the campaign.
- `ScheduledTime` – The date and time, 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 create a Lambda function, see Building Lambda Functions in the *AWS Lambda Developer Guide*.

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

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

Amazon Pinpoint only accepts endpoint definitions in the function's return value. The function 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 only honors the `TitleOverride` and `BodyOverride` attributes for endpoints where the value of the `ChannelType` attribute is one of the following: `SMS`, `GCM`, `APNS`, `APNS_SANDBOX`, `APNS_VOIP`, `APNS_VOIP_SANDBOX`, `ADM`, or `BAIDU`.
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 (Lambda Function Policies) in the *AWS Lambda Developer Guide*.

Example Function Policy

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

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

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

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

Granting Amazon Pinpoint Invocation Permission

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

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

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

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

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

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

Assigning a Lambda Function to a Campaign

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

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

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

- `LambdaFunctionName` – The name or ARN of the Lambda function that Amazon Pinpoint invokes before sending messages for the campaign.
- `Mode` – Set to `FILTER`. With this mode, Amazon Pinpoint invokes the function and waits for it to return the modified endpoints. After receiving them, Amazon Pinpoint sends the message. Amazon Pinpoint waits for up to 15 seconds before failing the message delivery.

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

Creating Campaigns

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

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

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

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

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

For more information, see Campaigns.

Creating Standard Campaigns

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

Creating Campaigns With the AWS SDK for Java

The following example demonstrates how to create a campaign with the AWS SDK for Java.

```
import com.amazonaws.services.pinpoint.AmazonPinpointClient;
import com.amazonaws.services.pinpoint.model.Action;
import com.amazonaws.services.pinpoint.model.CampaignResponse;
import com.amazonaws.services.pinpoint.model.CreateCampaignRequest;
import com.amazonaws.services.pinpoint.model.CreateCampaignResult;
import com.amazonaws.services.pinpoint.model.Message;
import com.amazonaws.services.pinpoint.model.MessageConfiguration;
import com.amazonaws.services.pinpoint.model.Schedule;
import com.amazonaws.services.pinpoint.model.WriteCampaignRequest;

import java.util.ArrayList;
import java.util.List;

public class PinpointCampaignSample {

    public CampaignResponse createCampaign(AmazonPinpointClient client, String appId,
 String segmentId) {
        Schedule schedule = new Schedule()
                .withStartTime("IMMEDIATE");

        Message defaultMessage = new Message()
                .withAction(Action.OPEN_APP)
                .withBody("My message body.")
                .withTitle("My message title.");

        MessageConfiguration messageConfiguration = new MessageConfiguration()
                .withDefaultMessage(defaultMessage);

        WriteCampaignRequest request = new WriteCampaignRequest()
                .withDescription("My description.")
                .withSchedule(schedule)
                .withSegmentId(segmentId)
                .withName("MyCampaign")
                .withMessageConfiguration(messageConfiguration);

        CreateCampaignRequest createCampaignRequest = new CreateCampaignRequest()
                .withApplicationId(appId).withWriteCampaignRequest(request);

        CreateCampaignResult result = client.createCampaign(createCampaignRequest);

        System.out.println("Campaign ID: " + result.getCampaignResponse().getId());

        return result.getCampaignResponse();
    }

}
```

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

```
Campaign ID: b1c3de717aea4408a75bb3287a906b46
```

Creating A/B Test Campaigns

An A/B test behaves like a standard campaign, but enables you to define different treatments for the campaign message or schedule.

Creating A/B Test Campaigns With the AWS SDK for Java

The following example demonstrates how to create an A/B test campaign with the AWS SDK for Java.

```
import com.amazonaws.services.pinpoint.AmazonPinpointClient;
import com.amazonaws.services.pinpoint.model.Action;
import com.amazonaws.services.pinpoint.model.CampaignResponse;
import com.amazonaws.services.pinpoint.model.CreateCampaignRequest;
import com.amazonaws.services.pinpoint.model.CreateCampaignResult;
import com.amazonaws.services.pinpoint.model.Message;
import com.amazonaws.services.pinpoint.model.MessageConfiguration;
import com.amazonaws.services.pinpoint.model.Schedule;
import com.amazonaws.services.pinpoint.model.WriteCampaignRequest;
import com.amazonaws.services.pinpoint.model.WriteTreatmentResource;

import java.util.ArrayList;
import java.util.List;

public class PinpointCampaignSample {

    public CampaignResponse createAbCampaign(AmazonPinpointClient client, String appId,
 String segmentId) {
        Schedule schedule = new Schedule()
                .withStartTime("IMMEDIATE");

        // Default treatment.
        Message defaultMessage = new Message()
                .withAction(Action.OPEN_APP)
                .withBody("My message body.")
                .withTitle("My message title.");

        MessageConfiguration messageConfiguration = new MessageConfiguration()
                .withDefaultMessage(defaultMessage);

        // Additional treatments
        WriteTreatmentResource treatmentResource = new WriteTreatmentResource()
                .withMessageConfiguration(messageConfiguration)
                .withSchedule(schedule)
                .withSizePercent(40)
                .withTreatmentDescription("My treatment description.")
                .withTreatmentName("MyTreatment");

        List<WriteTreatmentResource> additionalTreatments = new
 ArrayList<WriteTreatmentResource>();
        additionalTreatments.add(treatmentResource);

        WriteCampaignRequest request = new WriteCampaignRequest()
                .withDescription("My description.")
                .withSchedule(schedule)
                .withSegmentId(segmentId)
                .withName("MyCampaign")
                .withMessageConfiguration(messageConfiguration)
                .withAdditionalTreatments(additionalTreatments)
                .withHoldoutPercent(10); // Hold out of A/B test

        CreateCampaignRequest createCampaignRequest = new CreateCampaignRequest()
                .withApplicationId(appId).withWriteCampaignRequest(request);

        CreateCampaignResult result = client.createCampaign(createCampaignRequest);

        System.out.println("Campaign ID: " + result.getCampaignResponse().getId());

        return result.getCampaignResponse();
    }

}
```

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

```
Campaign ID: b1c3de717aea4408a75bb3287a906b46
```

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 three methods that you can use to send transactional email messages:

- By using the SendMessages operation in the Amazon Pinpoint API (p. 190): 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 SendEmail operation in the Amazon Pinpoint Email API (p. 197): You can use the `SendEmail` operation in the Amazon Pinpoint Email API to send email messages only.

  The advantage of using this operation is that the Amazon Pinpoint Email API includes additional options that are specific to the email channel. For example, when you send an email by using the Amazon Pinpoint Email API, you can specify CC and BCC recipients. You can also specify a configuration set. You can use configuration sets to specify which of your dedicated IP addresses your emails should be sent from, or to specify how Amazon Pinpoint handles open and click events.
- By using the Amazon Pinpoint SMTP interface (p. 206): 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. If you need to be able to send messages to CC and BCC recipients, or use dedicated IP pools, or specify message tags in your emails, you should use the `SendEmail` operation in the Amazon Pinpoint Email API.

The following decision tree shows some of the factors that you should consider when you decide which method you use to send transactional email in Amazon Pinpoint.

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 up AWS Credentials and Region for Development 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.

```
package com.amazonaws.samples;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

import com.amazonaws.services.pinpoint.AmazonPinpoint;
import com.amazonaws.services.pinpoint.AmazonPinpointClientBuilder;
import com.amazonaws.services.pinpoint.model.AddressConfiguration;
import com.amazonaws.services.pinpoint.model.ChannelType;
import com.amazonaws.services.pinpoint.model.DirectMessageConfiguration;
import com.amazonaws.services.pinpoint.model.EmailMessage;
import com.amazonaws.services.pinpoint.model.MessageRequest;
import com.amazonaws.services.pinpoint.model.SendMessagesRequest;
import com.amazonaws.services.pinpoint.model.SimpleEmail;
import com.amazonaws.services.pinpoint.model.SimpleEmailPart;

public class SendMessages {

    // 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/
    public 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.
    public 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.
    public 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.
    public static String appId = "ce796be37f32f178af652b26eexample";

    // The subject line of the email.
    public static String subject = "Amazon Pinpoint test";

    // The email body for recipients with non-HTML email clients.
    static final String textBody = "Amazon Pinpoint Test (SDK for Java 2.x)\r\n"
            + "---------------------------------\r\n"
            + "This email was sent using the Amazon Pinpoint "
            + "API using the AWS SDK for Java 2.x.";

    // The body of the email for recipients whose email clients support
    // HTML content.
    static final String htmlBody = "<h1>Amazon Pinpoint test (AWS SDK for Java 2.x)</
h1>"
            + "<p>This email was sent through the <a href='https://aws.amazon.com/
pinpoint/'>"
            + "Amazon Pinpoint</a> Email API using the "
            + "<a href='https://aws.amazon.com/sdk-for-java/'>AWS SDK for Java 2.x</
a>";

    // The character encoding the you want to use for the subject line and
    // message body of the email.
    public static String charset = "UTF-8";

    public static void main(String[] args) throws IOException {

        try {
            Map<String,AddressConfiguration> addressMap =
                new HashMap<String,AddressConfiguration>();

            addressMap.put(toAddress, new AddressConfiguration()
                .withChannelType(ChannelType.EMAIL));

            AmazonPinpoint client = AmazonPinpointClientBuilder.standard()
                .withRegion(region).build();

            SendMessagesRequest request = (new SendMessagesRequest()
                .withApplicationId(appId)
                .withMessageRequest(new MessageRequest()
                    .withAddresses(addressMap)
                    .withMessageConfiguration(new DirectMessageConfiguration()
                        .withEmailMessage(new EmailMessage()
                            .withSimpleEmail(new SimpleEmail()
                                .withHtmlPart(new SimpleEmailPart()
                                    .withCharset(charset)
                                    .withData(htmlBody)
                                )
                                .withTextPart(new SimpleEmailPart()
                                    .withCharset(charset)
                                    .withData(textBody)
                                )
                                .withSubject(new SimpleEmailPart()
                                    .withCharset(charset)
                                    .withData(subject)
                                )
                            )
                        )
                    )
                )
            );
            System.out.println("Sending message...");
            client.sendMessages(request);
            System.out.println("Message sent!");
    } catch (Exception ex) {
        System.out.println("The message wasn't sent. Error message: "
                + ex.getMessage());
        }
    }
}
```

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 (Boto 3). This example assumes that you've already installed and configured the SDK for Python (Boto 3). For more information, see Quickstart in the *AWS SDK for Python (Boto 3) 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 (Boto 3) API Reference*.

This code example was tested using the SDK for Python (Boto 3) 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 Email API

This section contains complete code examples that you can use to send email through the Amazon Pinpoint Email 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.PinpointEmail;
using Amazon.PinpointEmail.Model;

namespace PinpointEmailSDK
{
    class MainClass
    {
        // The AWS Region that you want to use to send the email. For a list of
        // AWS Regions where the Amazon Pinpoint Email API is available, see
        // https://docs.aws.amazon.com/pinpoint-email/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";

        // 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";

        // The configuration set that you want to use to send the email.
        static string configSet = "ConfigSet";

        // The subject line of the email.
        static string subject = "Amazon Pinpoint Email API test";

        // The body of the email for recipients whose email clients don't
        // support HTML content.
        static string textBody = "Amazon Pinpoint Email Test (.NET)\r\n"
                               + "This email was sent using the Amazon Pinpoint Email"
                               + "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 API Test (AWS SDK for .NET)</h1>
  <p>This email was sent using the
    <a href='https://aws.amazon.com/pinpoint/'>Amazon Pinpoint</a> Email API
    using the <a href='https://aws.amazon.com/sdk-for-net/'>
      AWS SDK for .NET</a>.</p>
</body>
</html>";

        // The message tags that you want to apply to the email.
        static MessageTag tag0 = new MessageTag
        {
            Name = "key0",
            Value = "value0"
        };
        static MessageTag tag1 = new MessageTag
        {
            Name = "key1",
            Value = "value1"
        };

        // 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
 AmazonPinpointEmailClient(RegionEndpoint.GetBySystemName(region)))
            {
                // Create a request to send the message. The request contains
                // all of the message attributes and content that were defined
                // earlier.
                var sendRequest = new SendEmailRequest
                {
                    FromEmailAddress = senderAddress,

                    // An object that contains all of the destinations that you
                    // want to send the message to. You can send a message to up
                    // to 50 recipients in a single call to the API.
                    Destination = new Destination
                    {
                        // You can include multiple To, CC, and BCC addresses.
                        ToAddresses = new List<string>
                        {
                            toAddress,
                            //additional recipients here
                        },
                        CcAddresses = new List<string>
                        {
                            ccAddress,
                            //additional recipients here
                        },
                        BccAddresses = new List<string>
                        {
                            bccAddress,
                            //additional recipients here
                        }
                    },

                    // The body of the email message.
                    Content = new EmailContent
                    {
                        // Create a new Simple email message. If you need to
                        // include attachments, you should send a RawMessage
                        // instead.
                        Simple = new Message
                        {
                            Subject = new Content
                            {
                                Charset = charset,
                                Data = subject
                            },
                            Body = new Body
                            {
                                // The text-only body of the message.
                                Text = new Content
                                {
                                    Charset = charset,
                                    Data = textBody
                                },
                                // The HTML body of the message.
                                Html = new Content
                                {
                                    Charset = charset,
                                    Data = htmlBody
                                }
                            }
                        }
                    },

                    // The configuration set that you want to use when you send
                    // this message.
                    ConfigurationSetName = configSet,

                    // A list of tags that you want to apply to this message.
                    EmailTags = new List<MessageTag>
                    {
                        tag0,
                        tag1
                    }
                };

                // Send the email, and provide a confirmation message if it's
                // sent successfully.
                try
                {
                    Console.WriteLine("Sending email using the Amazon Pinpoint Email
 API...");
                    var response = client.SendEmail(sendRequest);
                    Console.WriteLine("Email sent!");
                }

                // If the message can't be sent, return the error message that's
                // provided by the Amazon Pinpoint Email API.
                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 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 up AWS Credentials and Region for Development in the *AWS SDK for Java Developer Guide*.

```
package com.amazonaws.samples;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;
import com.amazonaws.services.pinpointemail.AmazonPinpointEmail;
import com.amazonaws.services.pinpointemail.AmazonPinpointEmailClientBuilder;
import com.amazonaws.services.pinpointemail.model.Body;
import com.amazonaws.services.pinpointemail.model.Content;
import com.amazonaws.services.pinpointemail.model.Destination;
import com.amazonaws.services.pinpointemail.model.EmailContent;
import com.amazonaws.services.pinpointemail.model.Message;
import com.amazonaws.services.pinpointemail.model.MessageTag;
import com.amazonaws.services.pinpointemail.model.SendEmailRequest;

public class SendMessage {

    // The AWS Region that you want to use to send the email. For a list of
    // AWS Regions where the Amazon Pinpoint Email API is available, see
    // https://docs.aws.amazon.com/pinpoint-email/latest/APIReference
    static final 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 final 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 final String toAddress = "recipient@example.com";

    // CC and BCC addresses. If your account is in the sandbox, these
    // addresses have to be verified.
    // In this example, there are several CC recipients. Each address is separated
    // with a comma. We convert this string to an ArrayList later in the example.
    static final String ccAddress = "cc_recipient0@example.com,
 cc_recipient1@example.com";
    static final String bccAddress = "bcc_recipient@example.com";

    // The configuration set that you want to use to send the email.
    static final String configurationSet = "ConfigSet";

    // The subject line of the email.
    static final String subject = "Amazon Pinpoint Email API test";

    // The email body for recipients with non-HTML email clients.
    static final String textBody = "Amazon Pinpoint Email Test (Java)\r\n"
            + "This email was sent using the Amazon Pinpoint "
            + "Email API using the AWS SDK for Java.";

    // The body of the email for recipients whose email clients support
    // HTML content.
    static final String htmlBody = "<h1>Amazon Pinpoint test (AWS SDK for Java)</h1>"
            + "<p>This email was sent through the <a href='https://aws.amazon.com/
pinpoint/'>"
            + "Amazon Pinpoint</a> Email API using the "
            + "<a href='https://aws.amazon.com/sdk-for-java/'>AWS SDK for Java</a>";

    // The message tags that you want to apply to the email.
    static final String tagKey0 = "key0";
    static final String tagValue0 = "value0";
    static final String tagKey1 = "key1";
    static final String tagValue1 = "value1";

    // The character encoding the you want to use for the subject line and
    // message body of the email.
    static final String charset = "UTF-8";

    public static void main(String[] args) throws IOException {

        // Convert comma-separated lists of To, CC, and BCC Addresses into collections.
        // This works even if the string only contains a single email address.
        Collection<String> toAddresses =
                new ArrayList<String>(Arrays.asList(toAddress.split("\\s*,\\s*")));
        Collection<String> ccAddresses =
                new ArrayList<String>(Arrays.asList(ccAddress.split("\\s*,\\s*")));
        Collection<String> bccAddresses =
                new ArrayList<String>(Arrays.asList(bccAddress.split("\\s*,\\s*")));

        try {
            // Create a new email client
            AmazonPinpointEmail client = AmazonPinpointEmailClientBuilder.standard()
                    .withRegion(region).build();

            // Combine all of the components of the email to create a request.
            SendEmailRequest request = new SendEmailRequest()
                    .withFromEmailAddress(senderAddress)
                    .withConfigurationSetName(configurationSet)
                    .withDestination(new Destination()
                        .withToAddresses(toAddresses)
                        .withCcAddresses(ccAddresses)
                        .withBccAddresses(bccAddresses)
                    )
                    .withContent(new EmailContent()
                        .withSimple(new Message()
                            .withSubject(new Content()
                                .withCharset(charset)
                                .withData(subject)
                            )
                            .withBody(new Body()
                                .withHtml(new Content()
                                    .withCharset(charset)
                                    .withData(htmlBody)
                                )
                                .withText(new Content()
                                    .withCharset(charset)
                                    .withData(textBody)
                                )
                            )
                        )
                    )
                    .withEmailTags(new MessageTag()
                        .withName(tagKey0)
                        .withValue(tagValue0)
                    )
                    .withEmailTags(new MessageTag()
                        .withName(tagKey1)
                        .withValue(tagValue1)
                    );
            client.sendEmail(request);
            System.out.println("Email sent!");
            System.out.println(request);
        } catch (Exception ex) {
            System.out.println("The email wasn't sent. Error message: "
                    + ex.getMessage());
        }
    }
}
```

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';

var 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 Email API is available, see
// https://docs.aws.amazon.com//pinpoint-email/latest/APIReference
var 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.
var senderAddress = "Mary Major <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.
// Note: All recipient addresses in this example are in arrays, which makes it
// easier to specify multiple recipients. Alternatively, you can make these
// variables strings, and then modify the To/Cc/BccAddresses attributes in the
// params variable so that it passes an array for each recipient type.
var toAddresses = [ "recipient@example.com" ];

// CC and BCC addresses. If your account is in the sandbox, these
// addresses have to be verified.
var ccAddresses = [ "cc_recipient1@example.com","cc_recipient2@example.com" ];
var bccAddresses = [ "bcc_recipient@example.com" ];

// The configuration set that you want to use to send the email.
var configuration_set = "ConfigSet";

// The subject line of the email.
var subject = "Amazon Pinpoint Test (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 Email 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 message tags that you want to apply to the email.
var tag0 = { 'Name':'key0', 'Value':'value0' };
var tag1 = { 'Name':'key1', 'Value':'value1' };

// 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, and specify which
// profile to use in the 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 PinpointEmail object.
var pinpointEmail = new AWS.PinpointEmail();

// Specify the parameters to pass to the API.
var params = {
  FromEmailAddress: senderAddress,
  Destination: {
    ToAddresses: toAddresses,
    CcAddresses: ccAddresses,
    BccAddresses: bccAddresses
  },
  Content: {
    Simple: {
      Body: {
        Html: {
          Data: body_html,
          Charset: charset
        },
        Text: {
          Data: body_text,
          Charset: charset
        }
      },
      Subject: {
        Data: subject,
        Charset: charset
      }
    }
  },
  ConfigurationSetName: configuration_set,
  EmailTags: [
    tag0,
    tag1
  ]
};

//Try to send the email.
pinpointEmail.sendEmail(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.MessageId);
  }
});
```

Python

Use this example to send email by using the AWS SDK for Python (Boto 3). This example assumes that you've already installed and configured the SDK for Python (Boto 3). For more information, see Quickstart in the *AWS SDK for Python (Boto 3) 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 (Boto 3) API Reference*.

This code example was tested using the SDK for Python (Boto 3) 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 Email API is available, see
# https://docs.aws.amazon.com/pinpoint-email/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.
TOADDRESSES = ["recipient@example.com"]

# CC and BCC addresses. If your account is in the sandbox, these
# addresses have to be verified.
CCADDRESSES = ["cc_recipient1@example.com", "cc_recipient2@example.com"]
BCCADDRESSES = ["bcc_recipient@example.com"]

# The configuration set that you want to use to send the email.
CONFIGURATION_SET = "ConfigSet"

# The subject line of the email.
SUBJECT = "Amazon Pinpoint Test (SDK for Python)"

# 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.
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</a>.</p>
</body>
</html>
            """

# The message tags that you want to apply to the email.
TAG0 = {'Name': 'key0', 'Value': 'value0'}
TAG1 = {'Name': 'key1', 'Value': 'value1'}

# The character encoding that you want to use for the subject line and message
# body of the email.
CHARSET = "UTF-8"
```

```
# Create a new Pinpoint resource and specify a region.
client = boto3.client('pinpoint-email', region_name=AWS_REGION)

# Send the email.
try:
    # Create a request to send the email. The request contains all of the
    # message attributes and content that were defined earlier.
    response = client.send_email(

        FromEmailAddress=SENDER,

        # An object that contains all of the email addresses that you want to
        # send the message to. You can send a message to up to 50 recipients in
        # a single call to the API.
        Destination={
            'ToAddresses': TOADDRESSES,
            'CcAddresses': CCADDRESSES,
            'BccAddresses': BCCADDRESSES
        },
        # The body of the email message.
        Content={
            # Create a new Simple message. If you need to include attachments,
            # you should send a RawMessage instead.
            'Simple': {
                'Subject': {
                    'Charset': CHARSET,
                    'Data': SUBJECT,
                },
                'Body': {
                    'Html': {
                        'Charset': CHARSET,
                        'Data': BODY_HTML
                    },
                    'Text': {
                        'Charset': CHARSET,
                        'Data': BODY_TEXT,
                    }
                }
            }
        },
        # The configuration set that you want to use when you send this message.
        ConfigurationSetName=CONFIGURATION_SET,
        EmailTags=[
            TAG0,
            TAG1
        ]
    )
# Display an error if something goes wrong.
except ClientError as e:
    print("The message wasn't sent. Error message: \"" + e.response['Error']['Message']
 + "\"")
else:
    print("Email sent!")
    print("Message ID: " + response['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 user name.
        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.

```
"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 up AWS Credentials and Region for Development in the *AWS SDK for Java Developer Guide*.

```
package com.amazonaws.samples;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

import com.amazonaws.services.pinpoint.AmazonPinpoint;
import com.amazonaws.services.pinpoint.AmazonPinpointClientBuilder;
import com.amazonaws.services.pinpoint.model.AddressConfiguration;
import com.amazonaws.services.pinpoint.model.ChannelType;
import com.amazonaws.services.pinpoint.model.DirectMessageConfiguration;
import com.amazonaws.services.pinpoint.model.MessageRequest;
import com.amazonaws.services.pinpoint.model.SMSMessage;
import com.amazonaws.services.pinpoint.model.SendMessagesRequest;

public class SendMessage {

    // 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/
    public static 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.
    public static String originationNumber = "+12065550199";

    // The recipient's phone number.  For best results, you should specify the
    // phone number in E.164 format.
    public static String destinationNumber = "+14255550142";

    // The content of the SMS message.
    public static String message = "This message was sent through Amazon Pinpoint "
            + "using the AWS SDK for Java. 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.
    public static 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.
    public static String messageType = "TRANSACTIONAL";

    // The registered keyword associated with the originating short code.
    public static 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
    public static String senderId = "MySenderID";

    public static void main(String[] args) throws IOException {

        try {
            Map<String,AddressConfiguration> addressMap =
                    new HashMap<String,AddressConfiguration>();

            addressMap.put(destinationNumber, new AddressConfiguration()
                    .withChannelType(ChannelType.SMS));

            AmazonPinpoint client = AmazonPinpointClientBuilder.standard()
```

```
                    .withRegion(region).build();

            SendMessagesRequest request = new SendMessagesRequest()
                    .withApplicationId(appId)
                    .withMessageRequest(new MessageRequest()
                    .withAddresses(addressMap)
                    .withMessageConfiguration(new DirectMessageConfiguration()
                            .withSMSMessage(new SMSMessage()
                                    .withBody(message)
                                    .withMessageType(messageType)
                                    .withOriginationNumber(originationNumber)
                                    .withSenderId(senderId)
                                    .withKeyword(registeredKeyword)
                            )
                    )
            );
            System.out.println("Sending message...");
            client.sendMessages(request);
            System.out.println("Message sent!");
    } catch (Exception ex) {
        System.out.println("The message wasn't sent. Error message: "
                + ex.getMessage());
        }
    }
}
```

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 (Boto 3). This example assumes that you've already installed and configured the SDK for Python. For more information, see Quickstart in the AWS SDK for Python (Boto 3) 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 (Boto 3) 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*.

```

package com.amazonaws.samples;

import java.io.IOException;

import com.amazonaws.services.pinpointsmsvoice.AmazonPinpointSMSVoice;
import com.amazonaws.services.pinpointsmsvoice.AmazonPinpointSMSVoiceClientBuilder;
import com.amazonaws.services.pinpointsmsvoice.model.SSMLMessageType;
import com.amazonaws.services.pinpointsmsvoice.model.SendVoiceMessageRequest;
import com.amazonaws.services.pinpointsmsvoice.model.VoiceMessageContent;

public class SendMessage {

    // 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/
    static final String 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.
    static final String originationNumber = "+12065550110";

    // The recipient's phone number.  For best results, you should specify the
    // phone number in E.164 format.
    static final String destinationNumber = "+12065550142";

    // 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
    static final String voiceName = "Matthew";

    // 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
    static final String languageCode = "en-US";
    // The content of the message. This example uses SSML to customize and control
    // certain aspects of the message, such as by adding pauses and changing
    // phonation. The message can't contain any line breaks.
    static final String ssmlMessage = "<speak>This is a test message sent from "
                                    + "<emphasis>Amazon Pinpoint</emphasis> "
                                    + "using the <break strength='weak'/>AWS "
                                    + "SDK for Java. "
                                    + "<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.
    static final String callerId = "+12065550199";

    // The configuration set that you want to use to send the message.
    static final String configurationSet = "ConfigSet";

    public static void main(String[] args) throws IOException {
        try {
            AmazonPinpointSMSVoice client =
 AmazonPinpointSMSVoiceClientBuilder.standard()
             .withRegion(region).build();
            SendVoiceMessageRequest request = new SendVoiceMessageRequest()
             .withCallerId(callerId)
             .withDestinationPhoneNumber(destinationNumber)
             .withOriginationPhoneNumber(originationNumber)
             .withConfigurationSetName(configurationSet)
             .withContent(new VoiceMessageContent()
              .withSSMLMessage(new SSMLMessageType()
               .withLanguageCode(languageCode)
               .withVoiceId(voiceName)
               .withText(ssmlMessage)
              )
             );
            client.sendVoiceMessage(request);
            System.out.println("The message was sent successfully.");
        } catch (Exception ex) {
            System.out.println("The message wasn't sent. Error message: " +
 ex.getMessage
        }
    }
}
```

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 (Boto 3). This example assumes that you've already installed and configured the SDK for Python (Boto 3).

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 (Boto 3) 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'])
```

Streaming Amazon Pinpoint Events to Kinesis

The Kinesis platform offers services that you can use to load and analyze streaming data on AWS. You can configure Amazon Pinpoint to send events to Amazon Kinesis Data Firehose or Amazon Kinesis Data Streams. By streaming your events, you enable more flexible options for analysis and storage. For more information, and for instructions on how to set up event streaming in the Amazon Pinpoint console, see Streaming App and Campaign Events with Amazon Pinpoint in the *Amazon Pinpoint User Guide*.

Setting up Event Streaming

The following examples demonstrate how to configure Amazon Pinpoint to automatically send the event data from an app to an Kinesis stream or Kinesis Data Firehose delivery stream.

Prerequisites
These examples require the following input:

- The app ID of an app that is integrated with Amazon Pinpoint and reporting events. For information about how to integrate, see Integrating Amazon Pinpoint with Your Application (p. 129).
- The ARN of an Kinesis stream or Kinesis Data Firehose delivery stream in your AWS account. For information about creating these resources, see Amazon Kinesis Data Streams in the *Amazon Kinesis Data Streams Developer Guide* or Creating an Amazon Kinesis Data Firehose Delivery Stream in the *Amazon Kinesis Data Firehose Developer Guide*.
- The ARN of an AWS Identity and Access Management (IAM) role that authorizes Amazon Pinpoint to send data to the stream. For information about creating a role, see IAM Role for Streaming Events to Kinesis (p. 271).

AWS CLI

The following AWS CLI example uses the `put-event-stream` command. This command configures Amazon Pinpoint to send app and campaign events to an Kinesis stream:

```
aws pinpoint put-event-stream --application-id application-id --write-event-stream
 DestinationStreamArn=stream-arn,RoleArn=role-arn
```

AWS SDK for Java

The following Java example configures Amazon Pinpoint to send app and campaign events to an Kinesis stream:

```
public PutEventStreamResult createEventStream(AmazonPinpoint pinClient, String appId,
                                                          String streamArn, String roleArn)
 {
    WriteEventStream stream = new WriteEventStream()
            .withDestinationStreamArn(streamArn)
            .withRoleArn(roleArn);

    PutEventStreamRequest request = new PutEventStreamRequest()
            .withApplicationId(appId)
            .withWriteEventStream(stream);

    return pinClient.putEventStream(request);
}
```

This example constructs a `WriteEventStream` object that stores the ARNs of the Kinesis stream and the IAM role. The `WriteEventStream` object is passed to a `PutEventStreamRequest` object to configure Amazon Pinpoint to stream events for a specific app. The `PutEventStreamRequest` object is passed to the `putEventStream` method of the Amazon Pinpoint client.

You can assign an Kinesis stream to multiple apps. Amazon Pinpoint will send event data from each app to the stream, enabling you to analyze the data as a collection. The following example method accepts a list of app IDs, and it uses the previous example method, `createEventStream`, to assign a stream to each app:

```
public List<PutEventStreamResult> createEventStreamFromAppList(
        AmazonPinpoint pinClient, List<String> appIDs, String streamArn, String roleArn) {
    return appIDs.stream()
            .map(appId -> createEventStream(pinClient, appId, streamArn, roleArn))
            .collect(Collectors.toList());
}
```

With Amazon Pinpoint, you can assign one stream to multiple apps, but you cannot assign multiple streams to one app.

Disabling Event Streaming

If you assigned an Kinesis stream to an app, you can disable event streaming for that app. Amazon Pinpoint stops streaming the events, but you can view analytics based on the events in the Amazon Pinpoint console.

AWS CLI

Use the `delete-event-stream` command:

```
aws pinpoint delete-event-stream --application-id application-id
```

AWS SDK for Java

Use the `deleteEventStream` method of the Amazon Pinpoint client:

```
pinClient.deleteEventStream(new DeleteEventStreamRequest().withApplicationId(appId));
```

Event Data

After you set up event streaming, Amazon Pinpoint sends each event reported by your application, and each campaign event created by Amazon Pinpoint, as a JSON data object to your Kinesis stream.

The event type is indicated by the `event_type` attribute in the event JSON object.

App Events

After you integrate your app with Amazon Pinpoint and you run one or more campaigns, Amazon Pinpoint streams events about user activity and campaign message deliveries.

Example

The JSON object for an app event contains the data shown in the following example.

```
{
  "event_type": "_session.stop",
  "event_timestamp": 1487973802507,
  "arrival_timestamp": 1487973803515,
  "event_version": "3.0",
  "application": {
    "app_id": "a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6",
    "cognito_identity_pool_id": "us-east-1:a1b2c3d4-e5f6-g7h8-i9j0-k1l2m3n4o5p6",
    "package_name": "main.page",
    "sdk": {
      "name": "aws-sdk-mobile-analytics-js",
      "version": "0.9.1:2.4.8"
    },
    "title": "title",
    "version_name": "1.0",
    "version_code": "1"
  },
  "client": {
    "client_id": "m3n4o5p6-a1b2-c3d4-e5f6-g7h8i9j0k1l2",
    "cognito_id": "us-east-1:i9j0k1l2-m3n4-o5p6-a1b2-c3d4e5f6g7h8"
  },
  "device": {
    "locale": {
      "code": "en_US",
      "country": "US",
      "language": "en"
    },
    "make": "generic web browser",
    "model": "Unknown",
    "platform": {
      "name": "android",
      "version": "10.10"
    }
  },
  "session": {
    "session_id": "f549dea9-1090-945d-c3d1-e496780baac5",
    "start_timestamp": 1487973202531,
    "stop_timestamp": 1487973802507
  },
  "attributes": {},
  "metrics": {}
}
```

Standard App Event Types

Amazon Pinpoint streams the following standard types for app events:

- _campaign.send
- _monetization.purchase

- _session.start
- _session.stop
- _session.pause
- _session.resume
- _userauth.sign_in
- _userauth.sign_up
- _userauth.auth_fail

Email Events

If the email channel is enabled, Amazon Pinpoint streams events about email deliveries, complaints, opens, and more.

Example

The JSON object for an email event contains the data shown in the following example.

```
{
  "event_type": "_email.delivered",
  "event_timestamp": 1487973802507,
  "arrival_timestamp": 1487973803515,
  "event_version": "3.0",
  "application": {
    "app_id": "a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6",
    "cognito_identity_pool_id": "us-east-1:a1b2c3d4-e5f6-g7h8-i9j0-k1l2m3n4o5p6",
    "package_name": "main.page",
    "sdk": {
      "name": "aws-sdk-mobile-analytics-js",
      "version": "0.9.1:2.4.8"
    },
    "title": "title",
    "version_name": "1.0",
    "version_code": "1"
  },
  "client": {
    "client_id": "m3n4o5p6-a1b2-c3d4-e5f6-g7h8i9j0k1l2",
    "cognito_id": "us-east-1:i9j0k1l2-m3n4-o5p6-a1b2-c3d4e5f6g7h8"
  },
  "device": {
    "locale": {
      "code": "en_US",
      "country": "US",
      "language": "en"
    },
    "make": "generic web browser",
    "model": "Unknown",
    "platform": {
      "name": "android",
      "version": "10.10"
    }
  },
  "session": {
    "session_id": "f549dea9-1090-945d-c3d1-e496780baac5",
    "start_timestamp": 1487973202531,
    "stop_timestamp": 1487973802507
  },
  "attributes": {},
  "metrics": {}
}
```

Standard Email Event Types

Amazon Pinpoint streams the following standard types for the email channel:

- _email.send
- _email.delivered
- _email.rejected
- _email.hardbounce
- _email.softbounce
- _email.complaint
- _email.open
- _email.click
- _email.unsubscribe

SMS Events

If the SMS channel is enabled, Amazon Pinpoint streams events about SMS deliveries.

Example

The JSON object for an SMS event contains the data shown in the following example.

```
{
  "account_id": "123412341234",
  "event_type": "_SMS.SUCCESS",
  "arrival_timestamp": 2345678,
  "timestamp": 13425345,
  "timestamp_created": "1495756908285",
  "application_key": "AppKey-688037015201",
  "unique_id": "uniqueId-688037015201140879759 69",
  "attributes": {
    "message_id": "12234sdv",
    "sender_request_id": "abdfg",
    "number_of_message_parts": 1,
    "record_status": "SUCCESS",
    "message_type": "Transactional",
    "keyword": "test",
    "mcc_mnc": "456123",
    "iso_country_code": "US"
  },
  "metrics": {
    "price_in_millicents_usd": 0.0
  },
  "facets": {},
  "additional_properties": {}
}
```

Standard SMS Event Types

Amazon Pinpoint streams the following standard types for the SMS channel:

- _sms.send
- _sms.success
- _sms.fail
- _sms.optout

Event Attributes

The JSON object for an event contains the following attributes.

Event

Attribute	Description
`event_type`	The type of event reported by your app.
`event_timestamp`	The time at which the event is reported as Unix time in milliseconds. If your app reports events using the AWS Mobile SDK for Android or the AWS Mobile SDK for iOS, the time stamp is automatically generated.
`arrival_timestamp`	The time at which the event is received by Amazon Pinpoint as Unix time in milliseconds. Does not apply to campaign events.
`event_version`	The version of the event JSON schema. Check this version in your event-processing application so that you know when to update the application in response to a schema update.
`application`	Your Amazon Pinpoint app. See the *Application* table.
`client`	The app client installed on the device that reports the event. See the *Client* table.
`device`	The device that reports the event. See the *Device* table.
`session`	The app session on the device. Typically a session begins when a user opens your app.
`attributes`	Custom attributes that your app reports to Amazon Pinpoint as part of the event.
`metrics`	Custom metrics that your app reports to Amazon Pinpoint as part of the event.

Application

Attribute	Description
`app_id`	The unique ID of the Amazon Pinpoint app that reports the event.
`cognito_identity_pool_id`	The unique ID of the Amazon Cognito identity pool used by your app.
`package_name`	The name of your app package. For example, `com.example.my_app`.
`sdk`	The SDK used to report the event.
`title`	The title of your app.
`version_name`	The customer-facing app version, such as `V2.0`.

Attribute	Description
`version_code`	The internal code that represents your app version.

Client

Attribute	Description
`client_id`	The unique ID for the app client installed on the device. This ID is automatically generated by the AWS Mobile SDK for iOS and the AWS Mobile SDK for Android.
`cognito_id`	The unique ID assigned to the app client in the Amazon Cognito identity pool used by your app.

Device

Attribute	Description
`locale`	The device locale.
`make`	The device make, such as `Apple` or `Samsung`.
`model`	The device model, such as `iPhone`.
`platform`	The device platform, such as `ios` or `android`.

Session

Attribute	Description
`session_id`	The unique ID for the app session.
`start_timestamp`	The time at which the session starts as Unix time in milliseconds.
`stop_timestamp`	The time at which the session stops as Unix time in milliseconds.

Logging Amazon Pinpoint API Calls with AWS CloudTrail

Amazon Pinpoint is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or 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 in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to Amazon Pinpoint, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, including how to configure and enable it, see the AWS CloudTrail User Guide.

Amazon Pinpoint Information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When supported event activity occurs in Amazon Pinpoint, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing Events with CloudTrail Event History.

For an ongoing record of events in your AWS account, including events for Amazon Pinpoint, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- Overview for Creating a Trail
- CloudTrail Supported Services and Integrations
- Configuring Amazon SNS Notifications for CloudTrail
- Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see CloudTrail userIdentity Element.

You can create a trail and store your log files in your Amazon S3 bucket for as long as you want, and define Amazon S3 lifecycle rules to archive or delete log files automatically. By default, your log files are encrypted with Amazon S3 server-side encryption (SSE).

To be notified of log file delivery, configure CloudTrail to publish Amazon SNS notifications when new log files are delivered. For more information, see Configuring Amazon SNS Notifications for CloudTrail.

You can also aggregate Amazon Pinpoint log files from multiple AWS Regions and multiple AWS accounts into a single Amazon S3 bucket.

For more information, see Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts.

You can use CloudTrail to log actions for the following Amazon Pinpoint APIs:

- Amazon Pinpoint API (p. 233)
- Amazon Pinpoint Email API (p. 234)
- Amazon Pinpoint SMS and Voice API (p. 235)

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
- CreateImportJob
- CreateSegment
- DeleteAdmChannel
- DeleteApnsChannel
- DeleteApnsSandboxChannel
- DeleteApnsVoipChannel
- DeleteApnsVoipSandboxChannel
- DeleteApp
- DeleteBaiduChannel
- DeleteCampaign
- DeleteEmailChannel
- DeleteEventStream
- DeleteGcmChannel
- DeleteSegment
- DeleteSmsChannel
- GetAdmChannel
- GetApnsChannel
- GetApnsSandboxChannel
- GetApnsVoipChannel
- GetApnsVoipSandboxChannel
- GetApp
- GetApplicationSettings
- GetApps
- GetBaiduChannel
- GetCampaign
- GetCampaignActivities

- GetCampaignVersion
- GetCampaignVersions
- GetCampaigns
- GetEmailChannel
- GetEventStream
- GetGcmChannel
- GetImportJob
- GetImportJobs
- GetSegment
- GetSegmentImportJobs
- GetSegmentVersion
- GetSegmentVersions
- GetSegments
- GetSmsChannel
- ListTagsForResource
- PutEventStream
- TagResource
- UntagResource
- UpdateAdmChannel
- UpdateApnsChannel
- UpdateApnsSandboxChannel
- UpdateApnsVoipChannel
- UpdateApnsVoipSandboxChannel
- UpdateApplicationSettings
- UpdateBaiduChannel
- UpdateCampaign
- UpdateEmailChannel
- UpdateGcmChannel
- UpdateSegment
- UpdateSmsChannel

The following Amazon Pinpoint API actions **aren't** logged in CloudTrail:

- GetEndpoint
- SendMessages
- SendUsersMessages
- UpdateEndpoint
- UpdateEndpointsBatch

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
- DeleteConfigurationSet
- GetConfigurationSetEventDestinations
- CreateConfigurationSetEventDestination
- UpdateConfigurationSetEventDestination
- DeleteConfigurationSetEventDestination

The following Amazon Pinpoint SMS and Voice API action **isn't** logged in CloudTrail:

- SendVoiceMessage

Example: Amazon Pinpoint Log File Entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the `GetCampaigns` and `CreateCampaign` actions of the Amazon Pinpoint API.

```
{
  "Records": [
    {
      "awsRegion": "us-east-1",
      "eventID": "example0-09a3-47d6-a810-c5f9fd2534fe",
      "eventName": "GetCampaigns",
      "eventSource": "pinpoint.amazonaws.com",
      "eventTime": "2018-02-03T00:56:48Z",
      "eventType": "AwsApiCall",
      "eventVersion": "1.05",
      "readOnly": true,
      "recipientAccountId": "123456789012",
      "requestID": "example1-b9bb-50fa-abdb-80f274981d60",
      "requestParameters": {
        "application-id": "example71dfa4c1aab66332a5839798f",
        "page-size": "1000"
      },
      "responseElements": null,
      "sourceIPAddress": "192.0.2.0",
      "userAgent": "Jersey/${project.version} (HttpUrlConnection 1.8.0_144)",
      "userIdentity": {
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "accountId": "123456789012",
        "arn": "arn:aws:iam::123456789012:root",
        "principalId": "123456789012",
        "sessionContext": {
          "attributes": {
            "creationDate": "2018-02-02T16:55:29Z",
            "mfaAuthenticated": "false"
          }
        },
        "type": "Root"
      }
    },
    {
      "awsRegion": "us-east-1",
      "eventID": "example0-09a3-47d6-a810-c5f9fd2534fe",
      "eventName": "CreateCampaign",
      "eventSource": "pinpoint.amazonaws.com",
      "eventTime": "2018-02-03T01:05:16Z",
      "eventType": "AwsApiCall",
      "eventVersion": "1.05",
      "readOnly": false,
      "recipientAccountId": "123456789012",
      "requestID": "example1-b9bb-50fa-abdb-80f274981d60",
      "requestParameters": {
        "Description": "***",
        "HoldoutPercent": 0,
        "IsPaused": false,
        "MessageConfiguration": "***",
        "Name": "***",
        "Schedule": {
          "Frequency": "ONCE",
          "IsLocalTime": true,
          "StartTime": "2018-02-03T00:00:00-08:00",
          "Timezone": "utc-08"
        },
        "SegmentId": "exampleda204adf991a80281aa0e591",
        "SegmentVersion": 1,
        "application-id": "example71dfa4c1aab66332a5839798f"
      },
      "responseElements": {
        "ApplicationId": "example71dfa4c1aab66332a5839798f",
        "CreationDate": "2018-02-03T01:05:16.425Z",
        "Description": "***",
        "HoldoutPercent": 0,
        "Id": "example54a654f80948680cbba240ede",
        "IsPaused": false,
        "LastModifiedDate": "2018-02-03T01:05:16.425Z",
        "MessageConfiguration": "***",
        "Name": "***",
        "Schedule": {
          "Frequency": "ONCE",
          "IsLocalTime": true,
          "StartTime": "2018-02-03T00:00:00-08:00",
          "Timezone": "utc-08"
        },
        "SegmentId": "example4da204adf991a80281example",
        "SegmentVersion": 1,
        "State": {
          "CampaignStatus": "SCHEDULED"
        },
        "Version": 1
      },
      "sourceIPAddress": "192.0.2.0",
      "userAgent": "aws-cli/1.14.9 Python/3.4.3 Linux/3.4.0+ botocore/1.8.34",
      "userIdentity": {
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "accountId": "123456789012",
        "arn": "arn:aws:iam::123456789012:user/userName",
        "principalId": "AIDAIHTHRCDA62EXAMPLE",
        "type": "IAMUser",
        "userName": "userName"
      }
    }
  ]
}
```

The following example shows a CloudTrail log entry that demonstrates the `CreateConfigurationSet` and `CreateConfigurationSetEventDestination` actions in the Amazon Pinpoint SMS and Voice API.

```
{
  "Records": [
    {
      "eventVersion":"1.05",
      "userIdentity":{
        "type":"IAMUser",
        "principalId":"AIDAIHTHRCDA62EXAMPLE",
        "arn":"arn:aws:iam::111122223333:user/SampleUser",
        "accountId":"111122223333",
        "accessKeyId":"AKIAIOSFODNN7EXAMPLE",
        "userName":"SampleUser"
      },
      "eventTime":"2018-11-06T21:45:55Z",
      "eventSource":"sms-voice.amazonaws.com",
      "eventName":"CreateConfigurationSet",
      "awsRegion":"us-east-1",
      "sourceIPAddress":"192.0.0.1",
      "userAgent":"PostmanRuntime/7.3.0",
      "requestParameters":{
        "ConfigurationSetName":"MyConfigurationSet"
      },
      "responseElements":null,
      "requestID":"56dcc091-e20d-11e8-87d2-9994aexample",
      "eventID":"725843fc-8846-41f4-871a-7c52dexample",
      "readOnly":false,
      "eventType":"AwsApiCall",
      "recipientAccountId":"123456789012"
    },
    {
      "eventVersion":"1.05",
      "userIdentity":{
        "type":"IAMUser",
        "principalId":"AIDAIHTHRCDA62EXAMPLE",
        "arn":"arn:aws:iam::111122223333:user/SampleUser",
        "accountId":"111122223333",
        "accessKeyId":"AKIAIOSFODNN7EXAMPLE",
        "userName":"SampleUser"
      },
      "eventTime":"2018-11-06T21:47:08Z",
      "eventSource":"sms-voice.amazonaws.com",
      "eventName":"CreateConfigurationSetEventDestination",
      "awsRegion":"us-east-1",
      "sourceIPAddress":"192.0.0.1",
      "userAgent":"PostmanRuntime/7.3.0",
      "requestParameters":{
        "EventDestinationName":"CloudWatchEventDestination",
        "ConfigurationSetName":"MyConfigurationSet",
        "EventDestination":{
          "Enabled":true,
          "MatchingEventTypes":[
            "INITIATED_CALL",
            "INITIATED_CALL"
          ],
          "CloudWatchLogsDestination":{
            "IamRoleArn":"arn:aws:iam::111122223333:role/iamrole-01",
            "LogGroupArn":"arn:aws:logs:us-east-1:111122223333:log-group:clientloggroup-01"
          }
        }
      },
      "responseElements":null,
      "requestID":"81de1e73-e20d-11e8-b158-d5536example",
      "eventID":"fcafc21f-7c93-4a3f-9e72-fca2dexample",
      "readOnly":false,
      "eventType":"AwsApiCall",
      "recipientAccountId":"111122223333"
    }
  ]
}
```

Tagging Amazon Pinpoint Resources

A *tag* is a label that you optionally define and associate with projects (apps), campaigns, and segments in Amazon Pinpoint. Tags can help you categorize and manage these types of 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. A project, campaign, or segment can have as many as 50 tags.

Topics

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 project and `Production` for the other project.

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, digits, 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 limit 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, but 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 projects, campaigns, and segments, 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 the specified AWS Region for your AWS account, including Amazon Pinpoint resources, you can use the AWS Resource Groups Tagging API.

After you start implementing tags, you can apply tag-based, resource-level permissions to AWS Identity and Access Management (IAM) policies and API operations, including operations that support the addition of tags to resources when resources are created. This can help you 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 might create a policy that allows users to have full access to the Amazon Pinpoint resources that they've tagged:

```
{
   "Action": "mobiletargeting:*",
   "Effect": "Allow",
   "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 are 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* and the section called “IAM Policies for Users” (p. 249) in this guide.

Adding Tags to Resources

The following examples show you how to add a tag to an Amazon Pinpoint project (app), campaign, or segment by using the AWS CLI and the Amazon Pinpoint API.

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

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

To add a tag to an existing resource, use the `tag-resource` command:

```
$ 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. To get a list of ARNs for Amazon Pinpoint resources, you can display a list of all Amazon Pinpoint resources that have tags (p. 242).
- *`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 will set the value to an empty string.

To create a new resource and add a tag to it, use the appropriate `create` command for the resource and include the `tags` parameter. For example, the following command creates a new project named `MyApp` and adds a `Stack` tag key with a `Test` tag value to the project:

```
$ aws pinpoint create-app \
> --create-application-request '[
       {"Name": "MyApp"},
       {"tags": {"Stack": "Test"}}
]'
```

For information about the commands that you can use to create an Amazon Pinpoint resource, see the AWS CLI Command Reference.

HTTP

You can use Amazon Pinpoint by sending HTTP(S) requests directly to the Amazon Pinpoint API.

To create a new resource and add a tag to it, send a POST request to the appropriate resource URI and include the `tags` parameter and attributes in the body of the request. For example, the following request creates a new project named `MyApp` 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": "MyApp",
   "tags": {
      "Stack": "Test"
   }
}
```

To add a tag to an existing resource, send a POST request to the Tags URI, including the Amazon Resource Name (ARN) of the resource. 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 add a tag to an existing campaign or segment by sending a PUT request to the Campaign URI or the Segment URI, respectively, and including the `tags` parameter and attributes 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.

Displaying Tags for Resources

The following examples show you how to use the AWS CLI and the Amazon Pinpoint API to display a list of the tags that are associated with an Amazon Pinpoint project (app), campaign, or segment.

AWS CLI

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

To display a list of all the tags that are associated with a resource, use the `list-tags-for-resource` command:

```
$ aws pinpoint list-tags-for-resource \
> --resource-arn resource-arn \
```

Where *resource-arn* is the Amazon Resource Name (ARN) of the resource.

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 and set the `resource-type-filters` parameter to `mobiletargeting`, for example:

```
$ 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.

HTTP

You can use Amazon Pinpoint by sending HTTP(S) requests directly to the Amazon Pinpoint API.

To display all the tags that are associated with a specific resource, send a GET request to the Tags URI, including the Amazon Resource Name (ARN) of the resource. For example, the following request gets all tag information for the 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
```

In the JSON response to the preceding request, the `tags` attribute 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 (project, campaign, or segment), send a GET request to the appropriate resource URI. For example, the following request gets 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 preceding request lists all the campaigns in the project. The `tags` attribute 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:

- Whether you want to update a tag key, a tag value, or both.
- The type of resource that you want to update tags for.
- Whether you want to update a tag for one resource or multiple resources at the same time.

To update a tag key for a resource, you can remove the current tag (p. 244) and add a new tag (p. 240) by using the AWS CLI or the Amazon Pinpoint API.

To update only the tag value of a tag key for a resource, you can use the AWS CLI or the Amazon Pinpoint API. The examples in this section show you how.

To update a tag key or value for multiple resources at the same time, you can use the resource groups tagging operations of the AWS CLI or the AWS Resource Groups Tagging API.

AWS CLI

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

To update a tag value for a resource by using a resource ID, use the appropriate `update` and `write request` commands for the resource type and 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:

```
$ 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 tags you want to update.

To update a tag value for a resource by using an Amazon Resource Name (ARN), use the `tag-resource` command and include the `tags-model` parameter and arguments:

```
$ aws pinpoint tag-resource \
> --resource-arn resource-arn
 \
> --tags-model '{"tags": {"key": "value"}}'
```

Where:

- *`resource-arn`* is the ARN of the resource whose tags you want to update. To get a list of ARNs for Amazon Pinpoint resources, you can display a list of all Amazon Pinpoint resources that have tags (p. 242).
- *`key`* is the tag key. The `key` argument is required.
- *`value`* is the tag value that you want to use for the specified tag key (`key`). The `value` argument is required. To remove the tag value, don't specify a value for the `value` argument. Amazon Pinpoint will set the value to an empty string.

For more information about the commands that you can use to update an Amazon Pinpoint resource, see the AWS CLI Command Reference.

HTTP

You can use Amazon Pinpoint by sending HTTP(S) requests directly to the Amazon Pinpoint API.

To update a tag value for a project, use the `TagResources` action of the AWS Resource Groups Tagging API. The Amazon Pinpoint API currently doesn't support updates to tags for projects.

To update a tag value for a campaign or segment, send a PUT request to the appropriate resource URI of the Amazon Pinpoint API, including the ID of the campaign or segment to update. In the body of the request, include the `tags` parameter and attributes. In the `tags` parameter, specify the corresponding tag key for the `tag` attribute and, for the `tagvalue` attribute, do one of the following:

- To use a new value, specify the value.
- To remove the value, don't specify a value. Amazon Pinpoint will set the tag value to an empty string.

For example, the following request changes the tag value of the `Stack` tag key from `Test` to `Production` for 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 tags you want to update.

Removing Tags from Resources

The following examples show you how to remove a tag from an Amazon Pinpoint project (app), campaign, or segment by using the AWS CLI and the Amazon Pinpoint API.

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.

AWS CLI

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

To remove a tag, both the tag key and its associated value, from a resource, use the `untag-resource` command and include the `tag-keys` parameter and argument:

```
$ 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. 242).
- *`key`* is the tag that you want to remove from the resource. The `key` argument is required.

To remove multiple tags, both tag keys and their associated values, from a resource, use the `untag-resource` command and include the `tag-keys` parameter and arguments:

```
$ 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.

To remove only a specific tag value, not a tag key, from a resource, you can update the tags for the resource (p. 243).

HTTP

You can use Amazon Pinpoint by sending HTTP(S) requests directly to the Amazon Pinpoint API.

To remove a tag, both the tag key and its associated value, from a resource, send a DELETE request to the Tags URI. For the URI, append a query string that includes 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.

To remove multiple tags, both 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
```

To remove only a specific tag value, not a tag key, from a tag for a resource, you can update the tags for the resource (p. 243).

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.

Deleting Data from Amazon Pinpoint

Depending on how you use it, Amazon Pinpoint might store certain data that could be considered personal. For example, each endpoint in Amazon Pinpoint contains contact information for an end user, such as that person's email address or mobile phone number.

You can use the Amazon Pinpoint API to permanently delete personal data. This section includes procedures for deleting various types of data that could be considered personal.

Deleting Endpoints

An endpoint represents a single method of contacting one of your customers. Each endpoint can refer to a customer's email address, mobile device identifier, or email address. In many jurisdictions, this type of information might be considered personal.

The procedure in this section uses the AWS CLI to interact with the Amazon Pinpoint API. This procedure assumes that you've already installed and configured the AWS CLI. For more information, see Installing the AWS Command Line Interface in the *AWS Command Line Interface User Guide*.

To delete an endpoint by using the AWS CLI

- At the command line, type the following command:

```
aws pinpoint delete-endpoint --application-id example1884c7d659a2feaa0c5 --endpoint-
id ad015a3bf4f1b2b0b82example
```

  In the preceding command, replace *example1884c7d659a2feaa0c5* with the ID of the application or project that the endpoint is located in. Also, replace *ad015a3bf4f1b2b0b82example* with the unique ID of the endpoint itself.

Deleting Segment and Endpoint Data Stored in Amazon S3

You can import segments from a file that's stored in an Amazon S3 bucket by using the Amazon Pinpoint console or the API. You can also export application, segment, or endpoint data from Amazon Pinpoint to an Amazon S3 bucket. Both the imported and exported files can contain personal data, including email addresses, mobile phone numbers, and information about the physical location of the endpoint.

Content delivered to Amazon S3 buckets might contain customer content. For more information about removing sensitive data, see How Do I Empty an S3 Bucket? or How Do I Delete an S3 Bucket?.

Delete All AWS Data by Closing Your AWS Account

It's also possible to delete all data that you've stored in Amazon Pinpoint by closing your AWS account. However, this action also deletes all other data—personal or non-personal—that you've stored in every other AWS service.

When you close your AWS account, we retain the data in your AWS account for 90 days. At the end of this retention period, we delete this data permanently and irreversibly.

> **Warning**
> The following procedure completely removes all data that's stored in your AWS account across all AWS services and regions.

You can close your AWS account by using the AWS Management Console.

To close your AWS account

1. Open the AWS Management Console at https://console.aws.amazon.com.
2. Go to the **Account Settings** page at https://console.aws.amazon.com/billing/home?#/account.

   > **Warning**
   > The following two steps permanently delete all of the data you've stored in all AWS services across all AWS Regions.
3. Under **Close Account**, read the disclaimer that describes the consequences of closing your AWS account. If you agree to the terms, select the check box, and then choose **Close Account**.
4. On the confirmation dialog box, choose **Close Account**.

Permissions

To use Amazon Pinpoint, users in your AWS account require permissions that allow them to view analytics data, define user segments, create and deploy campaigns, and more. Users of your app also require access to Amazon Pinpoint so that your app can register endpoints and report usage data. To grant access to Amazon Pinpoint features, create AWS Identity and Access Management (IAM) policies that allow Amazon Pinpoint actions (p. 249).

IAM is a service that helps you securely control access to AWS resources. IAM policies include statements that allow or deny specific actions that users can perform on specific resources. Amazon Pinpoint provides a set of actions for IAM policies (p. 253) that you can use to specify granular permissions for Amazon Pinpoint users. You can grant the appropriate level of access to Amazon Pinpoint without creating overly permissive policies that might expose important data or compromise your campaigns. For example, you can grant unrestricted access to an Amazon Pinpoint administrator, and grant read-only access to individuals in your organization who need access to only analytics.

For more information about IAM policies, see Policies and Permissions in the *IAM User Guide*.

To import endpoint definitions, you must grant Amazon Pinpoint read-only access to an Amazon S3 bucket (p. 267).

Topics

IAM Policies for Amazon Pinpoint Users

You can add Amazon Pinpoint API actions to AWS Identity and Access Management (IAM) policies to allow or deny specific actions for Amazon Pinpoint users in your account. The Amazon Pinpoint API actions in your policies control what users can do in the Amazon Pinpoint console. These actions also control which programmatic requests users can make with the AWS SDKs, the AWS Command Line Interface (AWS CLI), or the Amazon Pinpoint APIs.

In a policy, you specify each action with the `mobiletargeting` namespace followed by a colon and the name of the action, such as `GetSegments`. Most actions correspond to a request to the Amazon Pinpoint API using a specific URI and HTTP method. For example, if you allow the `mobiletargeting:GetSegments` action in a user's policy, the user is allowed to make an HTTP GET request against the `/apps/`*`projectId`*`/segments` URI. This policy also allows the user to view the segments for a project in the console, and to retrieve the segments by using an AWS SDK or the AWS CLI.

Each action is performed on a specific Amazon Pinpoint resource, which you identify in a policy statement by its Amazon Resource Name (ARN). For example, the `mobiletargeting:GetSegments` action is performed on a specific app, which you identify with the ARN, `arn:aws:mobiletargeting:`*`region`*`:`*`accountId`*`:apps/`*`projectId`*.

Topics

Example Policies

The following examples demonstrate how you can manage Amazon Pinpoint access with IAM policies.

Amazon Pinpoint API Actions

Amazon Pinpoint Administrator

The following policy allows full access to all Amazon Pinpoint actions and resources:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "mobiletargeting:*"
            ],
            "Resource": "arn:aws:mobiletargeting:*:accountId:*"
        }
    ]
}
```

Note
As a best practice, you should create policies that follow the principle of *least privilege*. In other words, when you create IAM policies, they should only include the minimum number of permissions required to perform the task that you need to perform. For more information, see the IAM User Guide.

Read-Only Access

The following policy allows read-only access to all of the projects in your Amazon Pinpoint account in a specific AWS Region. This policy only applies to the Amazon Pinpoint API. For a policy that you can use to create read-only console users, see the next section (p. 251).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "mobiletargeting:Get*"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:mobiletargeting:region:accountId:*"
        }
    ]
}
```

In the preceding policy example, replace *`region`* with the name of an AWS Region, and replace *`accountId`* with your AWS account ID.

Console Read-Only Access

The following policy provides users with read-only access to the Amazon Pinpoint console. It includes read-only access to other services that the Amazon Pinpoint console depends on, such as Amazon SES, IAM, and Kinesis.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "mobiletargeting:Get*",
            "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 can also create read-only policies that provide access only to specific projects. The following policy lets users sign in to the console and view a list of applications. However, it only lets users view additional information about the project that's specified in the policy. You can modify this policy to allow access to additional projects or Regions.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "mobiletargeting:GetApps",
            "Resource": "arn:aws:mobiletargeting:region:accountId:*"
        },
        {
            "Effect": "Allow",
            "Action": "mobiletargeting:Get*",
            "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 policy 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.

Amazon Pinpoint SMS and Voice API Actions

Administrator Access

The following policy grants full access to the Amazon Pinpoint SMS and Voice API:

```
{
    "Version": "2018-09-05",
    "Statement": [
        {
            "Action": [
                "sms-voice:*"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

Read-Only Access

The following policy allows read-only access to the Amazon Pinpoint SMS and Voice API:

```
{
    "Version": "2018-09-05",
    "Statement": [
        {
            "Action": [
                "sms-voice:Get*",
                "sms-voice:List*",
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

Amazon Pinpoint Email API Actions

Administrator Access

The following policy grants full access to the Amazon Pinpoint Email API:

```
{
    "Version": "2018-09-05",
    "Statement": [
        {
            "Action": [
                "ses:*"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

Note
This policy also grants full access to the Amazon SES API.

Read-Only Access

The following policy allows read-only access to the Amazon Pinpoint Email API:

```
{
    "Version": "2018-09-05",
    "Statement": [
        {
            "Action": [
                "ses:Describe*",
                "ses:Get*",
                "ses:List*"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

Note
This policy also grants read-only access to the Amazon SES API.

Amazon Pinpoint API Actions

This section contains API actions that you can add to the IAM policies in your AWS account. By adding these policies to an IAM user account, you can specify which Amazon Pinpoint features that user is allowed to use.

To learn more about the Amazon Pinpoint API, see the Amazon Pinpoint API Reference.

Categories:

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`

Delete 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`

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`

Delete 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`

Update the certificate and private key for the APNs channel for a project. This allows Amazon Pinpoint to send push notifications to your iOS app.

- URI – `/apps/`*`projectId`*`/channels/apns`
- Method – PUT
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`accountId`*`:apps/`*`projectId`*`/channels/apns`

`mobiletargeting:DeleteApnsSandboxChannel`

Delete 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`

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`

Delete 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`

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:DeleteApnsVoipChannel`

Delete 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:GetApnsVoipChannel`

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:UpdateApnsVoipChannel`

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`

Delete 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`

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`

Delete 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`

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`

Delete the Firebase Cloud Messaging (FCM), formerly Google Cloud Messaging (GCM), channel for a project.

- 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, formerly GCM, channel for a project.

- URI – `/apps/`*`projectId`*`/channels/gcm`
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`accountId`*`:apps/`*`projectId`*`/channels/gcm`

`mobiletargeting:UpdateGcmChannel`

Update the API key for the FCM, formerly GCM, channel for a project. This allows Amazon Pinpoint to send push notifications to your Android app.

- URI – `/apps/`*`projectId`*`/channels/gcm`
- Method – PUT
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`accountId`*`:apps/`*`projectId`*`/channels/gcm`

`mobiletargeting:DeleteSmsChannel`

Delete 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`

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`

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: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`

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 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`

Messages

The following permissions are related to sending SMS messages and push notifications from your Amazon Pinpoint account. You can use the `SendMessages` and `SendUsersMessages` operations to send messages to specific endpoints without creating segments and campaigns first.

`mobiletargeting:SendMessages`

Send an SMS message or push notification to specific endpoints.

- URI – `/apps/projectId/messages`
- Method – POST
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:apps/projectId/messages`

`mobiletargeting:SendUsersMessages`

Send an SMS 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`*`/users-messages`

Phone Number Validate

The following permissions are related to using the Phone Number Validate feature 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 a project.

- URI – `/apps`
- Method – POST
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`accountId`*`:apps`

`mobiletargeting:DeleteApp`

Delete a project.

- URI – `/apps/`*`projectId`*
- Method – DELETE
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`accountId`*`:apps/`*`projectId`*

`mobiletargeting:GetApp`

Retrieve information about a specific project in your Amazon Pinpoint account.

- URI – `/apps/`*`projectId`*
- Method – GET
- Resource ARN – `arn:aws:mobiletargeting:`*`region`*`:`*`accountId`*`:apps/`*`projectId`*

`mobiletargeting:GetApps`

Retrieve a list of projects in 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`

Reports

The following permission is related to retrieving reports and metrics for your Amazon Pinpoint account.

`mobiletargeting:GetReports`

View analytics in the Amazon Pinpoint console.

- URI – Not applicable
- Method – Not applicable
- Resource ARN – `arn:aws:mobiletargeting:region:accountId:reports`

Segments

The following permissions are related to managing segments in your Amazon Pinpoint account. In Amazon Pinpoint, *segments* are groups of recipients for your campaigns that share certain attributes that you define.

`mobiletargeting:CreateSegment`

Create a segment. To allow a user to create a segment by importing endpoint data from outside 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 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 managing tags for resources in your Amazon Pinpoint account.

`mobiletargeting:ListTagsforResource`

Retrieve information about the tags that are associated with a project, campaign, 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, 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, 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 of 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 of 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 contains API actions that you can add to the IAM policies in your AWS account. By adding these policies to an IAM user account, you can specify which features of the Amazon Pinpoint SMS and Voice API a user is allowed to use.

To learn more about the Amazon Pinpoint SMS and Voice API, see the Amazon Pinpoint SMS and Voice API Reference.

`sms-voice:CreateConfigurationSet`

Create a configuration set for sending voice messages.

- URI – `/sms-voice/configuration-sets`
- Method – POST
- Resource ARN – Not available; use `*`

`sms-voice:DeleteConfigurationSet`

Delete a voice message configuration set.

- URI – /sms-voice/configuration-sets/*`ConfigurationSetName`*
- Method – DELETE
- Resource ARN – Not available; use `*`

`sms-voice:GetConfigurationSetEventDestinations`

Get information about a configuration set and the event destinations that it contains.

- URI – /sms-voice/configuration-sets/*`ConfigurationSetName`*/event-destinations
- Method – GET
- Resource ARN – Not available; use `*`

`sms-voice:CreateConfigurationSetEventDestination`

Create an event destination for voice events.

- URI – /sms-voice/configuration-sets/*`ConfigurationSetName`*/event-destinations
- Method – POST
- Resource ARN – Not available; use `*`

`sms-voice:UpdateConfigurationSetEventDestination`

Update an event destination for voice events.

- URI – /sms-voice/configuration-sets/*`ConfigurationSetName`*/event-destinations/*`EventDestinationName`*
- Method – PUT
- Resource ARN – Not available; use `*`

`sms-voice:DeleteConfigurationSetEventDestination`

Delete an event destination for voice events.

- URI – /sms-voice/configuration-sets/*`ConfigurationSetName`*/event-destinations/*`EventDestinationName`*
- Method – DELETE
- Resource ARN – Not available; use `*`

`sms-voice:SendVoiceMessage`

Create and send voice messages.

- URI – /sms-voice/voice/message
- Method – POST
- Resource ARN – Not available; use *

IAM Role for Importing Endpoints or Segments

With Amazon Pinpoint, you 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 by creating an AWS Identity and Access Management (IAM) role and attaching the following policies to the role:

- The `AmazonS3ReadOnlyAccess` AWS managed policy. This policy is created and managed by AWS, and it grants read-only access to your Amazon S3 bucket.
- A *trust policy* that allows Amazon Pinpoint to assume the role.

For more information about IAM roles, see IAM Roles in the *IAM User Guide*.

After you create the role, you can use Amazon Pinpoint to import segments. For an example of how to import a segment by using the AWS SDK for Java, see Importing Segments (p. 177). For information about creating the Amazon S3 bucket, creating endpoint files, and importing a segment by using the console, see Importing Segments in the *Amazon Pinpoint User Guide*.

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": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "pinpoint.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Creating the IAM Role (AWS CLI)

Complete the following steps to create the IAM role by using the AWS Command Line Interface (AWS CLI).

If you have not installed the AWS CLI, see 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.

   When you run this command, the AWS CLI prints the following output in your terminal:

   ```
   {
       "Role": {
           "AssumeRolePolicyDocument": {
               "Version": "2012-10-17",
               "Statement": [
                   {
                       "Action": "sts:AssumeRole",
                       "Effect": "Allow",
                       "Principal": {
                           "Service": "pinpoint.amazonaws.com"
                       }
                   }
               ]
           },
           "RoleId": "AIDACKCEVSQ6C2EXAMPLE",
           "CreateDate": "2016-12-20T00:44:37.406Z",
           "RoleName": "PinpointSegmentImport",
           "Path": "/",
           "Arn": "arn:aws:iam::111122223333:role/PinpointSegmentImport"
       }
   }
   ```

3. Use the `attach-role-policy` command to attach the `AmazonS3ReadOnlyAccess` AWS managed policy to the role:

   ```
   aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess
    --role-name PinpointSegmentImport
   ```

IAM Role for Exporting Endpoints or Segments

You can obtain a list of endpoints by creating an export job. When you create an export job, you have to specify a project ID, and you can optionally specify a segment ID. Amazon Pinpoint then exports a list of the endpoints associated with the project or segment to an Amazon Simple Storage Service (Amazon S3) bucket. The resulting file contains a JSON-formatted list of endpoints and their attributes, such as channel, address, opt-in/opt-out status, creation date, and endpoint ID.

To create an export job, you have to configure an IAM role that allows Amazon Pinpoint to write to an Amazon S3 bucket. The process of configuring the role consists of two steps:

1. Create an IAM policy that allows an entity (in this case, Amazon Pinpoint) to write to a specific Amazon S3 bucket.
2. Create an IAM role and attach the policy to it.

This section contains procedures for completing both of these steps. These procedures assume that you've already created an Amazon S3 bucket, and a folder in that bucket, for storing exported segments. For more information about creating buckets, see Create a Bucket in the *Amazon Simple Storage Service Getting Started Guide*.

These procedures also assume that you've already installed and configured the AWS CLI. For more information about setting up the AWS CLI, see Installing the AWS Command Line Interface in the *AWS Command Line Interface User Guide*.

Step 1: Create the IAM Policy

An IAM policy defines the permissions for an entity, such as an identity or resource. To create a role for exporting Amazon Pinpoint endpoints, you have to create a policy that grants permission to write to a specific folder in a specific Amazon S3 bucket. The following policy example follows the security practice of granting least privilege—that is, it 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. At the command line, navigate to the directory where `s3policy.json` is located. Then type the following command to create the policy:

```
aws iam create-policy --policy-name s3ExportPolicy --policy-document
 file://s3policy.json
```

If the policy was created successfully, you see output similar to the following:

```
{
    "Policy": {
        "CreateDate": "2018-04-11T18:44:34.805Z",
        "IsAttachable": true,
        "DefaultVersionId": "v1",
        "AttachmentCount": 0,
        "PolicyId": "ANPAJ2YJQRJCG3EXAMPLE",
        "UpdateDate": "2018-04-11T18:44:34.805Z",
        "Arn": "arn:aws:iam::123456789012:policy/s3ExportPolicy",
        "PolicyName": "s3ExportPolicy",
        "Path": "/"
    }
}
```

Copy the Amazon Resource Name (ARN) of the policy (`arn:aws:iam::123456789012:policy/s3ExportPolicy` in the preceding example). In the next section, you must supply this ARN when you create the role.

Note
If you see a message stating that your account isn't authorized to perform the `CreatePolicy` operation, then you need to attach a policy to your user account that lets you create new IAM policies and roles. For more information, see 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. At the command line, navigate to the directory where `trustpolicy.json` is located. Then type the following command to create a new role:

```
aws iam create-role --role-name s3ExportRole --assume-role-policy-document
 file://trustpolicy.json
```

If the command runs successfully, you see output similar to the following:

```
{
    "Role": {
        "RoleName": "s3ExportRole",
        "AssumeRolePolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Principal": {
                        "Service": "pinpoint.amazonaws.com"
                    },
                    "Action": "sts:AssumeRole"
                }
            ]
        },
        "RoleId": "AROAICPO353GIPEXAMPLE",
        "Arn": "arn:aws:iam::123456789012:role/s3ExportRole",
        "CreateDate": "2018-04-11T18:52:36.712Z",
        "Path": "/"
    }
}
```

3. At the command line, type the following command to attach the policy that you created in the previous section to the role 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 Streaming Events to Kinesis

Amazon Pinpoint can automatically send app usage data, or *event data*, from your app to a Kinesis stream or Amazon Kinesis Data Firehose delivery stream in your AWS account. Before Amazon Pinpoint can begin streaming the event data, you must delegate the required permissions to Amazon Pinpoint.

If you use the console to set up event streaming, Amazon Pinpoint automatically creates an AWS Identity and Access Management (IAM) role with the required permissions. For more information, see Streaming Amazon Pinpoint Events to Amazon Kinesis in the *Amazon Pinpoint User Guide*.

If you want to create the role manually, attach the following policies to the role:

- A permissions policy that allows Amazon Pinpoint to send records to your stream.
- A trust policy that allows Amazon Pinpoint to assume the role.

For more information about IAM roles, see IAM Roles in the *IAM User Guide*.

After you create the role, you can configure Amazon Pinpoint to automatically send events to your stream. For more information, see Streaming Amazon Pinpoint Events to Kinesis (p. 225).

Permissions Policies

To allow Amazon Pinpoint to send event data to your stream, attach one of the following policies to the role.

Amazon Kinesis Data Streams

The following policy allows Amazon Pinpoint to send event data to a Kinesis stream.

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Action": [
            "kinesis:PutRecords",
            "kinesis:DescribeStream"
        ],
        "Effect": "Allow",
        "Resource": [
            "arn:aws:kinesis:region:account-id:stream/stream-name"
        ]
    }
}
```

Amazon Kinesis Data Firehose

The following policy allows Amazon Pinpoint to send event data to a Kinesis Data Firehose delivery stream.

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": [
         "firehose:PutRecordBatch",
         "firehose:DescribeDeliveryStream"
        ],
        "Resource": [
         "arn:aws:firehose:region:account-id:deliverystream/delivery-stream-name"
     ]
    }
}
```

Trust Policy

To allow Amazon Pinpoint to assume the IAM role and perform the actions allowed by the permissions policy, attach the following trust policy to the role:

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

Creating the IAM Role (AWS CLI)

Complete the following steps to create the IAM role by using the AWS Command Line Interface (AWS CLI).

If you have not installed the AWS CLI, see Getting Set Up with the AWS Command Line Interface in the *AWS Command Line Interface User Guide*.

To create the role by using the IAM console, see Setting up Event Streaming in the *Amazon Pinpoint User Guide*.

To create the IAM role by using the AWS CLI

1. Create a JSON file that contains the trust policy for your role, and save the file locally. You can copy the trust policy provided in this topic.
2. Use the `create-role` command to create the role and attach the trust policy:

   ```
   aws iam create-role --role-name PinpointEventStreamRole --assume-role-policy-document
    file://PinpointEventStreamTrustPolicy.json
   ```

   Following the `file://` prefix, specify the path to the JSON file that contains the trust policy.

   When you run this command, the AWS CLI prints the following output in your terminal:

   ```
   {
       "Role": {
           "AssumeRolePolicyDocument": {
               "Version": "2012-10-17",
               "Statement": [
                   {
                       "Action": "sts:AssumeRole",
                       "Effect": "Allow",
                       "Principal": {
                           "Service": "pinpoint.amazonaws.com"
                       }
                   }
               ]
           },
           "RoleId": "AIDACKCEVSQ6C2EXAMPLE",
           "CreateDate": "2017-02-28T18:02:48.220Z",
   ```

```
        "RoleName": "PinpointEventStreamRole",
        "Path": "/",
        "Arn": "arn:aws:iam::111122223333:role/PinpointEventStreamRole"
    }
}
```

3. Create a JSON file that contains the permissions policy for your role, and save the file locally. You can copy one of the policies provided in the Permissions Policies (p. 272) section.
4. Use the `put-role-policy` command to attach the permissions policy to the role:

   ```
   aws iam put-role-policy --role-name PinpointEventStreamRole --
   policy-name PinpointEventStreamPermissionsPolicy --policy-document
    file://PinpointEventStreamPermissionsPolicy.json
   ```

   Following the `file://` prefix, specify the path to the JSON file that contains the permissions policy.

IAM Role for Streaming Email Events to Kinesis Data Firehose

In the Amazon Pinpoint 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 Amazon Pinpoint 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 Amazon Pinpoint Email API can deliver information about the following 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 Amazon Pinpoint 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 Amazon Pinpoint Email API to send email delivery and response records to your stream.

For more information about IAM roles, see IAM Roles in the *IAM User Guide*.

After you create the role, you can configure Amazon Pinpoint to automatically send events to your stream. For more information, see Streaming Amazon Pinpoint Events to Kinesis (p. 225).

Trust Policy

To allow the Amazon Pinpoint 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 Amazon Pinpoint 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 more information about installing and configuring the AWS CLI, see Getting Set Up with the AWS Command Line Interface in the *AWS Command Line Interface User Guide*.

To create the IAM role by using the AWS CLI

1. Create a JSON file that contains the trust policy for your role, and then save the file locally. You can copy the trust policy (p. 275) 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.

   When 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. 275) 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.

Limits in Amazon Pinpoint

The following sections describe limits within Amazon Pinpoint.

Topics

General Limits

The following limits affect general use of Amazon Pinpoint.

Resource	Default Limit	Eligible for Increase
API request payload size	7 MB per request	No
Apps	100 per account	No

Endpoint Limits

The following limits apply to the Endpoints resource in the Amazon Pinpoint API.

Resource	Default Limit	Eligible for Increase
Attributes assigned to the `Attributes`, `Metrics`, and `UserAttributes` parameters collectively	40 per app	No
Attributes assigned to the `Attributes` parameter	40 per app	No
Attributes assigned to the `Metrics` parameter	40 per app	No
Attributes assigned to the `UserAttributes` parameter	40 per app	No
Attribute name length	50 characters	No
Attribute value length	100 characters	No

Resource	Default Limit	Eligible for Increase
`EndpointBatchItem` objects in an `EndpointBatchRequest` payload	100 per payload. The payload size can't exceed 7 MB.	No
Endpoints with the same user ID	10 unique endpoints per user ID	No
Values assigned to `Attributes` parameter attributes	50 per attribute	No
Values assigned to `UserAttributes` parameter attributes	50 per attribute	No

Endpoint Import Limits

The following limits apply when you import endpoints into Amazon Pinpoint.

Resource	Default Limit	Eligible for Increase
Concurrent import jobs	2 per account	Yes (p. 285)
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. 285)

Segment Limits

The following limits apply to the Segments resource in the Amazon Pinpoint API.

Resource	Default Limit	Eligible for Increase
Maximum number of dimensions that can be used to create a segment	100 per segment	No

Campaign Limits

The following limits apply to the Campaigns resource in the Amazon Pinpoint API.

Resource	Default Limit	Eligible for Increase
Active campaigns	200 per account **Note** An *active campaign* is a campaign that hasn't	Yes (p. 285)

Resource	Default Limit	Eligible for Increase
	completed or failed. Active campaigns have a status of `SCHEDULED`, `EXECUTING`, or `PENDING_NEXT_RUN`.	
Message sends	100 million per campaign activity	Yes (p. 285)
Event-based campaigns	Each project can include up to 10 campaigns that are sent when events occur. Campaigns that use event-based triggers have to use dynamic segments (that is, they can't use imported segments). Event-based campaigns are only sent to customers who use apps that run version 2.7.2 or later of the AWS Mobile SDK for Android or version 2.6.30 or later of the AWS Mobile SDK for iOS. 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 re-deliver it.	No

Mobile Push Limits

The following limits apply to messages that Amazon Pinpoint delivers through mobile push channels.

Resource	Default Limit	Eligible for Increase
Maximum number of mobile push notifications that can be sent per second	25,000 notifications per second	No
Amazon Device Messaging (ADM) message payload size	6 KB per message	No
Apple Push Notification service (APNs) message payload size	4 KB per message	No
APNs sandbox message payload size	4 KB per message	No
Baidu Cloud Push message payload size	4 KB per message	No
Firebase Cloud Messaging (FCM) message payload size	4 KB per message	No

Email Limits

The limits in the following sections apply to the Email channel.

Email Sending Limits

Resource	Default Limit	Eligible for Increase
Number of emails that can be sent per 24-hour period (*sending quota*)	If your account is in the sandbox: 200 emails per 24-hour period. If your account is out of the sandbox, the quota varies based on your specific use case. **Note** This quota is based on the number of recipients, as opposed to the number of unique messages sent. A *recipient* is any email address on the To: line.	Yes (p. 285)
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. 285)

Email Message Limits

Resource	Default Limit	Eligible for Increase
Maximum message size (including attachments)	10 MB per message.	No
Number of verified identities	10,000 identities. **Note** *Identities* refers to email addresses or domains, or any combination of the two. Every email	No

Resource	Default Limit	Eligible for Increase
	you send using Amazon Pinpoint must be sent from a verified identity.	

Email Sender and Recipient Limits

Resource	Default Limit	Eligible for Increase
Sender address	All sending addresses or domains must be verified.	No
Recipient address	If your account is still in the sandbox, all recipient email addresses or domains must be verified. If your account is out of the sandbox, you can send to any valid address.	Yes (p. 285)
Number of recipients per message	50 recipients per message.	No
Number of identities that you can verify	10,000 identities per AWS Region. **Note** *Identities* refers to email addresses or domains, or any combination of the two. Every email you send using Amazon Pinpoint must be sent from a verified identity.	No

SMS Limits

The following limits apply to the SMS channel.

Resource	Default Limit	Eligible for Increase
Account spend threshold	USD$1.00 per account.	Yes (p. 285)
Number of SMS messages that can be sent each second (*sending rate#*)	20 messages per second.	No
Number of Amazon SNS topics for two-way SMS	100,000 per account.	Yes (p. 285)

Voice Limits

The following limits apply to the Voice channel.

Resource	Default Limit	
Number of voice messages that can be sent in a 24-hour period	If your account is in the sandbox: 20 messages. If your account is out of the sandbox: unlimited.	
Number of voice messages that can be sent to a single recipient in a 24-hour period	5 messages.	
Number of voice messages that can be sent per minute	If your account is in the sandbox: 5 calls per minute. If your account is out of the sandbox: 20 calls per minute.	
Number of voice messages that can be sent from a single originating phone number per second	1 message per second.	
Voice message length	If your account is in the sandbox: 30 seconds. If your account is out of the sandbox: 5 minutes.	
Ability to send voice messages to international phone numbers	If your account is in the sandbox, you can only send messages to recipients in the following countries: • Australia • Canada • China • Germany • Hong Kong • Israel • Japan • Mexico • Singapore • Sweden • United States • United Kingdom If your account is out of the sandbox, you can send messages to recipients in any country.	

Resource	Default Limit	
	Note International calls are subject to additional fees, which vary by destination country or region.	
Number of characters in a voice message	3,000 billable characters (characters in words that are spoken) 6,000 characters total (including billable characters and SSML tags)	
Number of configuration sets	10,000 voice configuration sets per AWS Region.	

Event Ingestion Limits

The following limits apply to the ingestion of events using the AWS Mobile SDKs and the Amazon Pinpoint Events API.

Resource	Default Limit	Eligible for Increase
Maximum number of custom event types	1,500 per app	No
Maximum number of custom attribute keys	500 per app	No
Maximum number of custom attribute values per attribute key	100,000	No
Maximum number of characters per attribute key	50	No
Maximum number of characters per attribute value	200	No
Maximum number of custom metric keys	500 per app	No
Maximum number events in a request	100 per request	No
Maximum size of a request	4 MB	No
Maximum size of an individual event	1,000 KB	No
Maximum number of attribute keys and metric keys for each event	40 per request	No

Requesting a Limit Increase

If the value in the **Eligible for Increase** column in any of the tables above is **Yes**, you can request a change to that limit.

To request a limit increase

1. Sign in to the AWS Management Console at https://console.aws.amazon.com/.
2. Create a new Support case at https://console.aws.amazon.com/support/home#/case/create.
3. On the **Create Case** page, make the following selections:
   - For **Regarding**, choose **Service Limit Increase**.
   - For **Limit Type**, choose one of the following options:
     - Choose **Amazon Pinpoint** for limit increases related to Amazon Pinpoint campaigns and imports.
     - Choose **Amazon Pinpoint Email** for limit increases related to the email channel.
     - Choose **Amazon Pinpoint SMS** for limit increases related to the SMS channel.
4. For **Use Case Description**, explain why you are requesting the limit increase.
5. For **Support Language**, choose the language you prefer to use when communicating with the AWS Support team.
6. For **Contact Method**, choose your preferred method of communicating with the AWS Support team.
7. Choose **Submit**.

Creating Custom Channels with AWS Lambda

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

Amazon Pinpoint supports messaging channels for mobile push, email, and SMS. However, some messaging use cases might require unsupported channels. For example, you might want to send a message to an instant messaging service, such as Facebook Messenger, or you might want to display a notification within your web application. In such cases, you can use AWS Lambda to create a custom channel that performs the message delivery outside of Amazon Pinpoint.

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

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

To create a custom channel, you define a Lambda function that handles the message delivery for an Amazon Pinpoint campaign. Then, you assign the function to a campaign by defining the campaign's `CampaignHook` settings. These settings include the Lambda function name and the `CampaignHook` mode. By setting the mode to `DELIVERY`, you specify that the Lambda function handles the message delivery instead of Amazon Pinpoint.

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

With the `CampaignHook` settings defined, Amazon Pinpoint automatically invokes the Lambda function when it runs the campaign, without sending the campaign's message to a standard channel. Instead, Amazon Pinpoint sends *event data* about the message delivery to the function, and it allows the function to handle the delivery. The event data includes the message body and the list of endpoints to which the message should be delivered.

After Amazon Pinpoint successfully invokes the function, it generates a successful send event for the campaign.

> **Note**
> You can also use the `CampaignHook` settings to assign a Lambda function that modifies and returns a campaign's segment before Amazon Pinpoint delivers the campaign's message. For more information, see Customizing Segments with AWS Lambda (p. 180).

To create a custom channel with AWS Lambda, first create a function that accepts the event data sent by Amazon Pinpoint and handles the message delivery. Then, authorize Amazon Pinpoint to invoke the function by assigning a Lambda function policy. Finally, assign the function to one or more campaigns by defining `CampaignHook` settings.

Event Data

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

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

The event data provides the following attributes:

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

Creating a Lambda Function

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

Example Lambda Function

The following example Lambda function receives event data when Amazon Pinpoint runs a campaign, and it sends the campaign's message to Facebook Messenger:

```
"use strict";

var https = require("https");
var q = require("q");

var VERIFY_TOKEN = "my_token";
var PAGE_ACCESS_TOKEN = "EAF...DZD";
/* this constant can be put in a constants file and shared between this function and your
 Facebook Messenger webhook code */
var FACEBOOK_MESSENGER_PSID_ATTRIBUTE_KEY = "facebookMessengerPsid";

exports.handler = function(event, context, callback) {

    var deliverViaMessengerPromises = [];

    if (event.Message && event.Endpoints) {
        for (var endpoint in event.Endpoints) {
            if (isFbookMessengerActive(event.Endpoints[endpoint])) {
                deliverViaMessengerPromises.push(deliverViaMessenger(event.Message,
 event.Endpoints[endpoint].User));
```

```
            }
        }
    }

    /* default OK response */
    var response = {
        body: "ok",
        statusCode: 200
    };

    if (deliverViaMessengerPromises.length > 0) {
        q.all(deliverViaMessengerPromises).done(function() {
            callback(null, response);
        });
    } else {
        callback(null, response);
    }

}

/**
Example Pinpoint Endpoint User object where we've added custom attribute
 facebookMessengerPsid to store the PSID needed by Facebook's API
{
    "UserId": "7a9870b7-493c-4521-b0ca-08bbbc36e595",
    "UserAttributes": {
        "facebookMessengerPsid": [ "1667566386619741" ]
    }
}
**/
function isFbookMessengerActive(endpoint) {
    return endpoint.User && endpoint.User.UserAttributes &&
 endpoint.User.UserAttributes[FACEBOOK_MESSENGER_PSID_ATTRIBUTE_KEY];
}

/**
Sample message object from Pinpoint. This sample was an SMS so it has "smsmessage"
 attribute but this will vary for each messaging channel
{
    "smsmessage": {
        "body": "This message should be intercepted by a campaign hook."
    }
}
**/
function deliverViaMessenger(message, user) {
    var deferred = q.defer();

    var messageText = message["smsmessage"]["body"];
    var pinpointUserId = user.UserId;
    var facebookPsid = user.UserAttributes[FACEBOOK_MESSENGER_PSID_ATTRIBUTE_KEY][0];
    console.log("Sending message for user %s and page %s with message:", pinpointUserId,
 facebookPsid, messageText);

    var messageData = {
        recipient: {
            id: facebookPsid
        },
        message: {
            text: messageText
        }
    };

    var body = JSON.stringify(messageData);
    var path = "/v2.6/me/messages?access_token=" + PAGE_ACCESS_TOKEN;
    var options = {
        host: "graph.facebook.com",
        path: path,
        method: "POST",
        headers: {
            "Content-Type": "application/json"
        }
    };

    var req = https.request(options, httpsCallback);

    req.on("error", function(e) {
        console.log("Error posting to Facebook Messenger: " + e);
        deferred.reject(e);
    });

    req.write(body);
    req.end();

    return deferred.promise;

    function httpsCallback(response) {
        var str = "";
        response.on("data", function(chunk) {
            str += chunk;
        });
        response.on("end", function() {
            console.log(str);
            deferred.resolve(response);
        });
    }
}
```

Assigning a Lambda Function Policy

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

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

Example Function Policy

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

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

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

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

Granting Amazon Pinpoint Invocation Permission

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

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

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

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

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

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

Assigning a Lambda Function to a Campaign

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

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

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

- `LambdaFunctionName` – The name or ARN of the Lambda function that Amazon Pinpoint invokes to send messages for the campaign.
- `Mode` – Set to `DELIVERY`. With this mode, Amazon Pinpoint uses the function to deliver the messages for a campaign, and it doesn't attempt to send the messages through the standard channels.

Document History for Amazon Pinpoint

The following table describes the documentation for this release of Amazon Pinpoint.

- **Latest documentation update:** May 14, 2019

Change	Description	Date
Importing segments from external systems	Added a tutorial (p. 18) that describes a solution for bringing customer data into Amazon Pinpoint from external systems, such as Salesforce or Marketo.	May 14, 2019
Regional availability	Amazon Pinpoint is now available in the AWS Asia Pacific (Mumbai) and Asia Pacific (Sydney) Regions.	April 25, 2019
Using Postman with Amazon Pinpoint	Added a tutorial (p. 3) that describes how to use Postman to interact with the Amazon Pinpoint API.	April 8, 2019
Setting Up an SMS Registration System	Added a Tutorials section (p. 3), and added a tutorial that describes how to create a solution that handles SMS user registration (p. 92).	February 27, 2019
Tagging support	Added information about tagging Amazon Pinpoint resources (p. 239).	February 27, 2019
Code examples	Added code examples (p. 189) in several programming languages that show you how to send email (p. 189), SMS (p. 214), and voice (p. 220) messages programmatically.	February 6, 2019
Regional availability	Amazon Pinpoint is now available in the AWS US West (Oregon) and EU (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	November 15, 2018

Change	Description	Date
	the Amazon Pinpoint SMS and Voice API.	
EU (Ireland) Availability	Amazon Pinpoint is now available in the AWS EU (Ireland) Region.	October 25, 2018
Events API	Use the Amazon Pinpoint API to record events (p. 133) and associate them with endpoints.	August 7, 2018
Code examples for defining and looking up endpoints	Code examples are added that show you how to define, update, delete, and look up endpoints. Examples are provided for the AWS CLI, AWS SDK for Java, and the Amazon Pinpoint API. For more information, see Defining Your Audience to Amazon Pinpoint (p. 136) and Accessing Audience Data in Amazon Pinpoint (p. 161).	August 7, 2018
Endpoint export permissions	Configure an IAM policy (p. 268) that allows you to export Amazon Pinpoint endpoints to an Amazon S3 bucket.	May 1, 2018
Updated topics for Amazon Pinpoint integration	Integrate Amazon Pinpoint (p. 129) 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. 232).	February 6, 2018
Updated service limits	Updated *Limits* (p. 278) with additional information about email limits.	January 19, 2018
Public beta for Amazon Pinpoint extensions	Use AWS Lambda functions to customize segments (p. 180) or create custom messaging channels (p. 286).	November 28, 2017
External ID removed from IAM trust policies	The external ID key is removed from the example trust policy (p. 267) and example Java code (p. 177) for importing segments.	October 26, 2017
Push notification payload limits	The limits include payload sizes for mobile push messages (p. 280).	October 25, 2017

Change	Description	Date
Updated service limits	Added SMS and email channel information to *Limits* (p. 278).	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
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. 131) and SDK for Java (p. 141).	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. 132).	August 31, 2017
Updated sample events	The example events (p. 226) 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. 225).	March 24, 2017
Permissions	See Permissions (p. 249) 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