

Table of Contents

1. Account Linking	1
Skill Configuration	1
Understanding the OAuth Flow	3
Understanding OAuth Versions	4
Registering a New Twitter Application	5
Setting the Twitter App Permissions	7
Adding the Authorization URL	8
Testing the Account Linking Flow	8
Implementing a TweetAirportStatusIntent Handler	11
Implementing a TwitterHelper class	12
Using the TwitterHelper	12
Updating the Skill Service	13
Updating the Skill Interface	14
Testing the TweetAirportStatus Intent	15

Account Linking

In this chapter you will extend the Airport Info skill you previously built, granting users the ability to tweet the status of the airport they have requested to their personal Twitter timeline.

Figure 1.1 Requesting Airport Status is Posted to Twitter

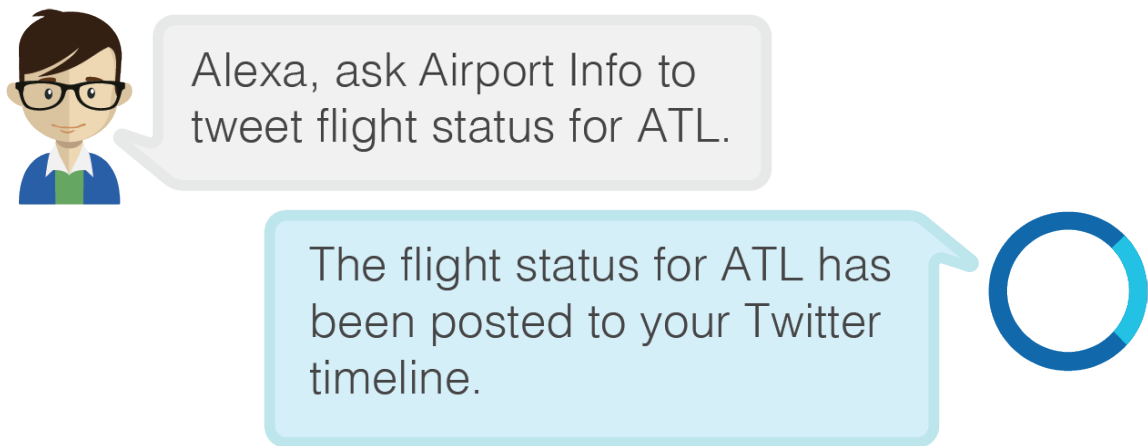
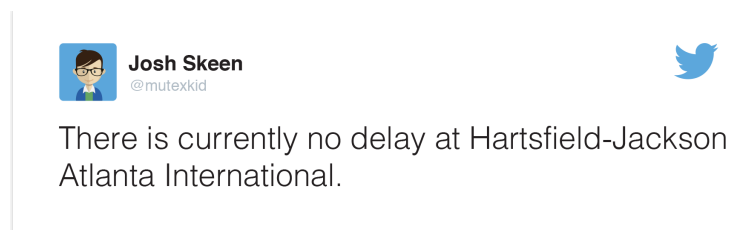


Figure 1.2



You will leverage the account linking feature of the Alexa Skills Kit platform to enable users to sign in to their personal Twitter account in a web browser and securely link their account with your skill. This action will share an access token from Twitter with the skill so that using the Twitter API from the skill is possible once authentication is complete. Upon initially enabling your skill in the in the skills section of the Alexa App, users will be prompted to authenticate with their Twitter credentials so that an access token may be generated and shared with the skill. In this chapter you will also take a look at OAuth web services, which enable services to share credentials with one another.

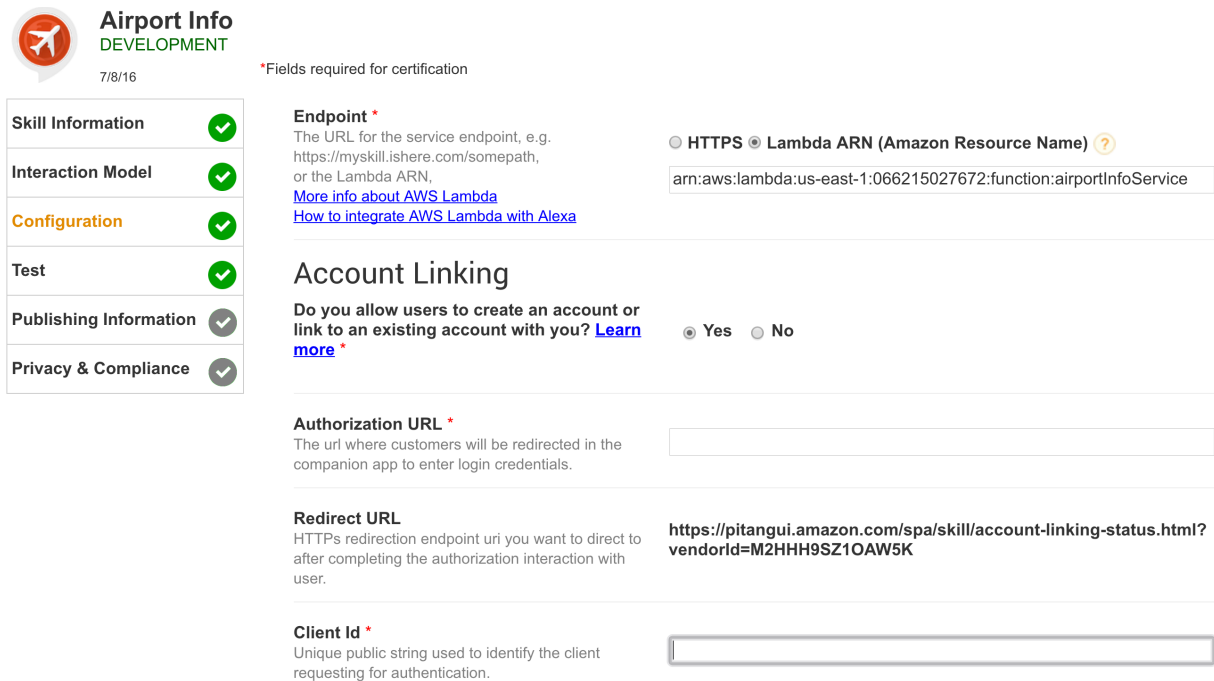
Skill Configuration

Begin by returning to the skill interface page for the Airport Info skill you configured earlier in the Amazon Developer Console, accessible from

<https://developer.amazon.com/edw/home.html#/skills/list>

Advance to the Configuration step in the skill interface for Airport Info and select "Yes" for Account Linking.

Figure 1.3 Enabling the Account Linking Option



Airport Info
DEVELOPMENT
7/8/16

*Fields required for certification

Skill Information	✓
Interaction Model	✓
Configuration	✓
Test	✓
Publishing Information	⊗
Privacy & Compliance	⊗

Endpoint *
The URL for the service endpoint, e.g. <https://myskill.ishere.com/somepath>, or the Lambda ARN.
[More info about AWS Lambda](#)
[How to integrate AWS Lambda with Alexa](#)

☐ HTTPS ☒ Lambda ARN (Amazon Resource Name) ?

Account Linking
Do you allow users to create an account or link to an existing account with you? [Learn more](#) *

☒ Yes ☐ No

Authorization URL *
The url where customers will be redirected in the companion app to enter login credentials.

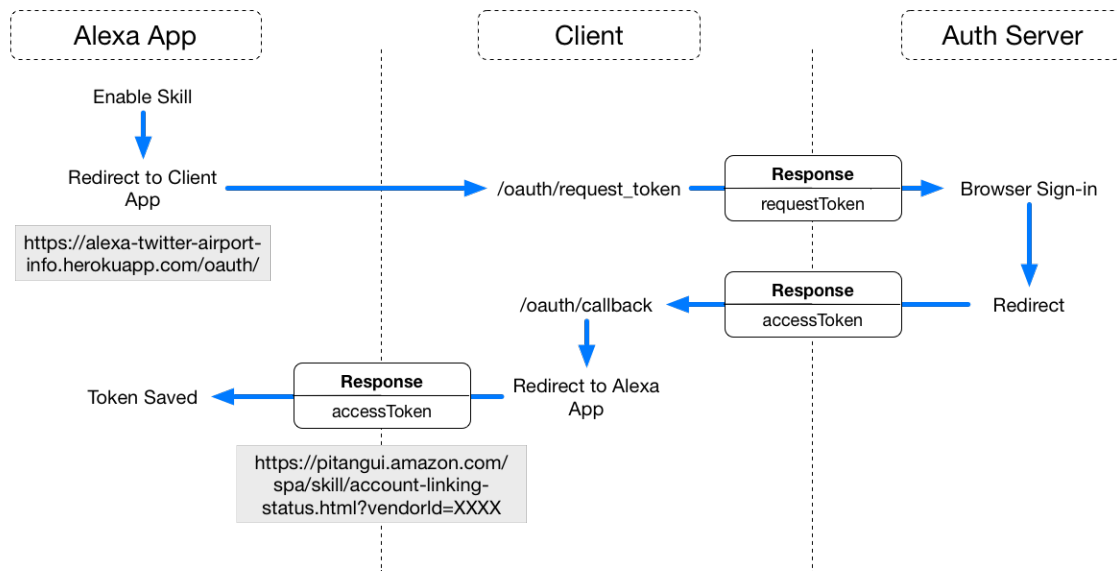
Redirect URL
HTTPs redirection endpoint uri you want to direct to after completing the authorization interaction with user.

Client Id *
Unique public string used to identify the client requesting for authentication.

Once you have made this selection, you will see additional fields for Authorization URL and Privacy Policy URL presented. You will return to complete these fields soon, but first you must configure the OAuth flow to work with the Twitter API you will be integrating with.

Understanding the OAuth Flow

Figure 1.4 OAuth flow diagram



The account linking feature will leverage an OAuth web flow within the Alexa App's card interface to allow a user to sign in using their Twitter credentials. This token grants the skill access to make posts to the Twitter timeline.

OAuth, an industry standard protocol for authentication, is widely used for authentication and securely sharing permissions between different user accounts on different servers. If you are new to OAuth, you can learn more about the protocol here:

<http://oauth.net/>

Prior to the workshop, a basic client OAuth integration for the Airport Info skill was written and deployed to the cloud. Because building the server code for the OAuth client leg of the authentication flow is beyond the scope of the course, you will use the already live implementation of the OAuth client needed to support the account linking behavior.

You may explore the source code for the client portion of the OAuth flow you will integrate with by downloading it here:

<https://github.com/bignerdranch/alex-twitter-airport-info>

The deployed OAuth client lives at `https://alex-twitter-airport-info.herokuapp.com`, and handles the redirect to the login page on Twitter and fetching a requestToken. Upon logging in, users will be redirected to the OAuth client integration with an accessToken. Once the accessToken is received, the OAuth client integration redirects users to the skills section in the Alexa App page, where the token is saved for use in the skill.

The OAuth client is a small web application that hosts two endpoints which are necessary to complete the OAuth flow with the skill interface.

Table 1.1 OAuth Client Endpoints

Endpoint	Purpose
/oauth/request_token	generates a request token via the Twitter API and redirects to the Twitter account sign in page.
/oauth/callback	redirected to with access token, once visitors sign in via the Twitter account sign in page. Then, redirects to the Alexa App page, saving the token to the skill.

Understanding OAuth Versions

There are two primary versions of OAuth in use with OAuth-based authentication services today: 2.0, and 1.0a. For Account Linking, Amazon officially supports 2.0, the latest version of protocol. Some services, like Twitter, still make use the 1.0a version of the protocol. Integrating with either a 2.0 or 1.0a endpoint will be a similar process, with 1.0a requiring some additional work.

To integrate with an OAuth 2.0 endpoint, you provide the Authorization URL for the service in the Alexa Skill Developer Portal. Then, you specify whether the OAuth service you're integrating with uses Implicit Grants or Auth Code grants for returning the access token. If the OAuth 2.0 service you're integrating with requires the Auth Code grant type, you'll provide additional values, client secret and authorization code values, in the Alexa Skill Developer portal. Refer to the documentation for the service you are integrating with for these values to determine if the Auth Code grant type is required.

In the Twitter integration example, you will integrate with an OAuth 1.0a based endpoint (as Twitter requires it), so we need an OAuth client implementation of our own to make the integration work correctly. The OAuth client in the Twitter example is an additional web service. When you configure the OAuth 1.0a client for your skill, you provide the redirect URL, the twitter session key, and session secret values. This is an unneeded step for OAuth 2.0 endpoints, but required for the older OAuth 1.0a endpoints. An OAuth 2.0 integration will not require configuring the session key and secret values in the redirect URL, or including the ruby client to make the integration work. The following diagram shows the simplified authentication flow when working with an OAuth 2.0 based endpoint. Notice, it does not require a OAuth client to function correctly.

Figure 1.5 An example OAuth 2.0 Flow

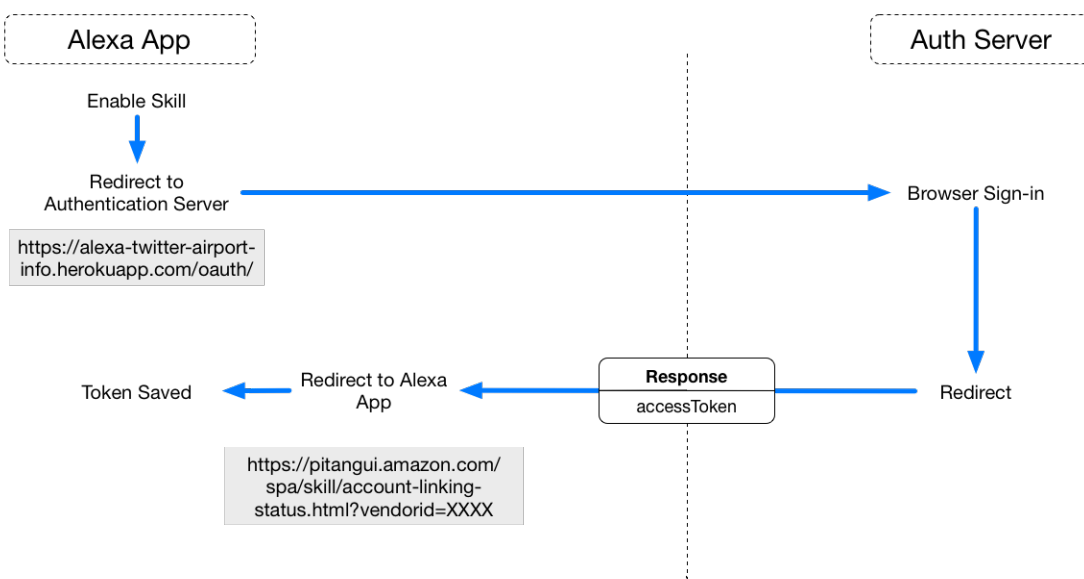
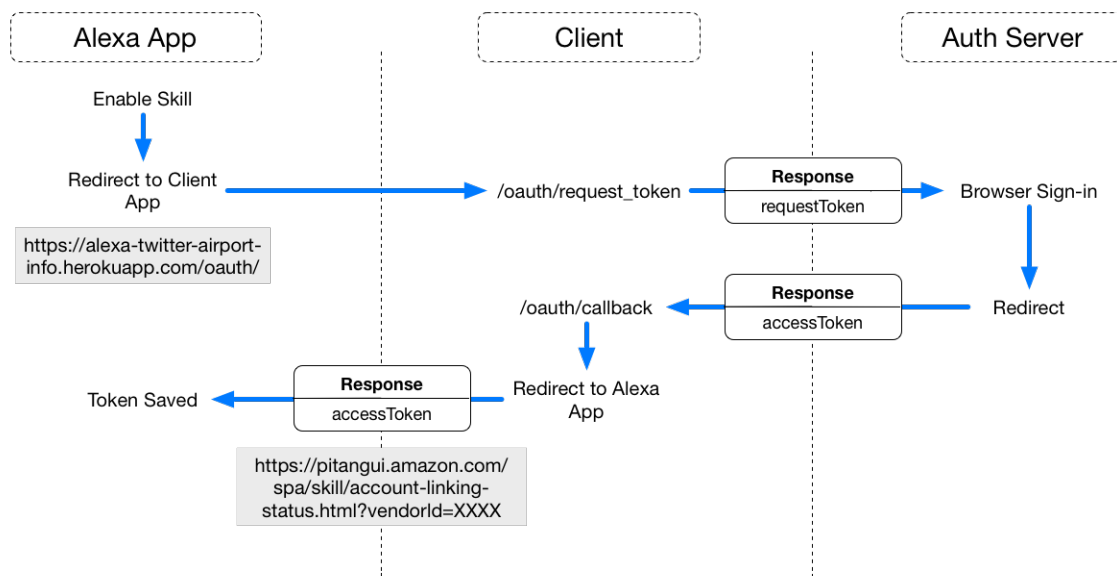


Figure 1.6 OAuth 1.0a flow diagram



If you compare the above diagrams you will notice that the OAuth 1.0a endpoint requires an additional "client" step to allow the generation of an authorization url and handle the redirect with token.

Registering a New Twitter Application

In order for the OAuth flow to work correctly against Twitter's server, you must register a new Twitter App to retrieve a consumer key and secret. If you do not have a Twitter account yet, you will require one to proceed — visit <https://twitter.com/signup?lang=en> and follow the signup process. To create a Twitter App, you also must associate a phone number with your Twitter account if you have not already done so. This can be found on <https://twitter.com/settings/devices>, once you have created an account and signed in.

Visit <https://apps.twitter.com/> and sign in with your Twitter account.

Figure 1.7 Twitter Apps





Click Create New App. On the subsequent page, enter "Airport Info Twitter App" for Name, "The Alexa skills Twitter integration for Airport Information" for Description, "<https://alexa-twitter-airport-info.herokuapp.com/app>" for Website, and enter

<https://alexa-twitter-airport-info.herokuapp.com/oauth/callback>

for the Callback URL. Click "Yes, I agree" to the "Developer Agreement", and click Create your Twitter Application.

Figure 1.8 Registering a new Twitter App

 Application Management 

Create an application

Application Details

Name *

Your application name. This is used to attribute the source of a tweet and in user-facing authorization screens. 32 characters max.

Description *

Your application description, which will be shown in user-facing authorization screens. Between 10 and 200 characters max.

Website *

Your application's publicly accessible home page, where users can go to download, make use of, or find out more information about your application. This fully-qualified URL is used in the source attribution for tweets created by your application and will be shown in user-facing authorization screens.
(If you don't have a URL yet, just put a placeholder here but remember to change it later.)

Callback URL

Where should we return after successfully authenticating? [OAuth 1.0a](#) applications should explicitly specify their `oauth_callback` URL on the request token step, regardless of the value given here.
To restrict your application from using callbacks, leave this field blank.

You will be redirected to the new Twitter App management page. Click on the Keys and Access Tokens tab, and copy the Consumer Key (API Key) and Consumer Secret (API Secret) values down, as they will be needed in the next step.

Figure 1.9 Copying the Application Consumer Key and Consumer Secret

Airport Info Twitter App

Test OAuth

Details Settings Keys and Access Tokens Permissions

Application Settings

Keep the "Consumer Secret" a secret. This key should never be human-readable in your application.

Consumer Key (API Key) [REDACTED]

Consumer Secret (API Secret) [REDACTED]

Access Level	Read and write (modify app permissions)
Owner	mutexkid
Owner ID	15166940

Application Actions

[Regenerate Consumer Key and Secret](#) [Change App Permissions](#)

Setting the Twitter App Permissions

For the Twitter app to successfully post to the timeline, the custom Twitter app permissions must be set to "Read, Write and Access direct messages". Click on the Permissions tab, and select Read, Write, and Access direct messages..

Figure 1.10 Setting the Twitter App Permissions

Details Settings Keys and Access Tokens Permissions

Access

What type of access does your application need?

[Read more about our Application Permission Model.](#)

☐ Read only

☐ Read and Write

☒ Read, Write and Access direct messages

Note:

Changes to the application permission model will only reflect in access tokens obtained after the permission model change is saved. You will need to re-negotiate existing access tokens to alter the permission level associated with each of your application's users.

[Update Settings](#)

After you have done this, click Update Settings.

Adding the Authorization URL

Note the Redirect URL field listed on the skill interface configuration page - this address is where the OAuth flow will return the token that is needed to authenticate with Twitter. This is a unique URL that allows Amazon to associate a resulting accessToken with your user's skill.

Figure 1.11 The Skill Redirect URL



Note the vendorId attribute provided in the Redirect URL - this value will be passed to the OAuth client in order to successfully redirect upon completing the auth flow.

In a production OAuth client implementation, the consumer_key, consumer_secret, and vendorId values would be defined as environment variables on the server. In this learning example you will pass the values to the server as URL parameters instead, so that you are not required to deploy your own OAuth server. The OAuth client is hosted at

`https://alexa-twitter-airport-info.herokuapp.com .`

On the Configuration page, under Authorization URL, enter

`https://alexa-twitter-airport-info.herokuapp.com/oauth/request_token?vendor_id=XXXXXX&consumer_key=YYYYYY
&consumer_secret=ZZZZZZ`

Replace "XXXXXXXXXX" with the value for "vendorId" found for the redirectURL. Replace "YYYYYYYYYY" and "ZZZZZZZZ" with the Consumer key and Consumer secret values you noted down on the Twitter App Management page.

For the Privacy Policy URL, enter `https://alexa-twitter-airport-info.herokuapp.com/policy`.

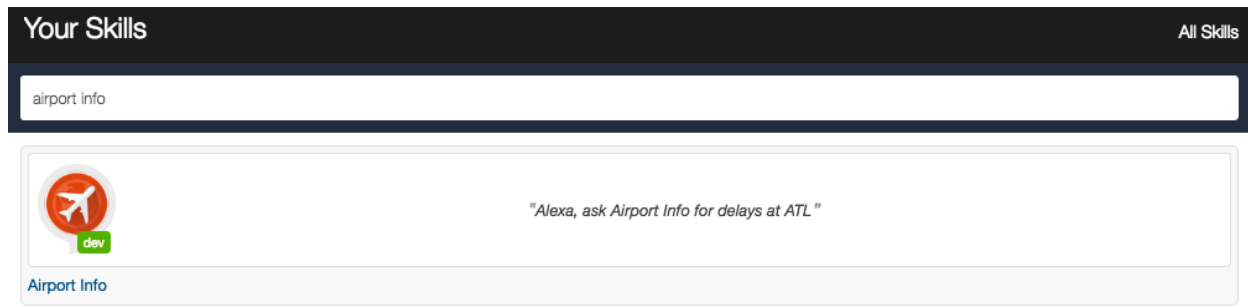
Testing the Account Linking Flow

You can now test that the account linking flow for the Airport Info skill works as expected. Visit

`http://alexa.amazon.com/#skills`

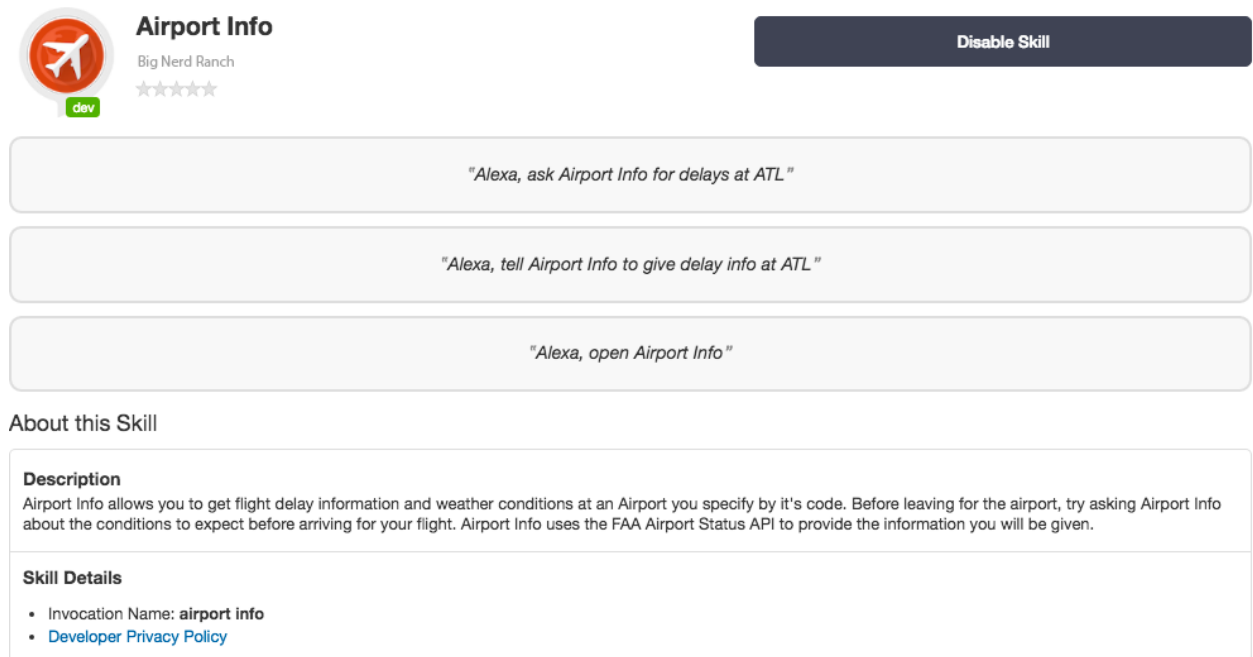
and search for Airport Info. Ensure you have clicked Your Skills at the top right of the interface.

Figure 1.12 Searching for the Airport Info Skill



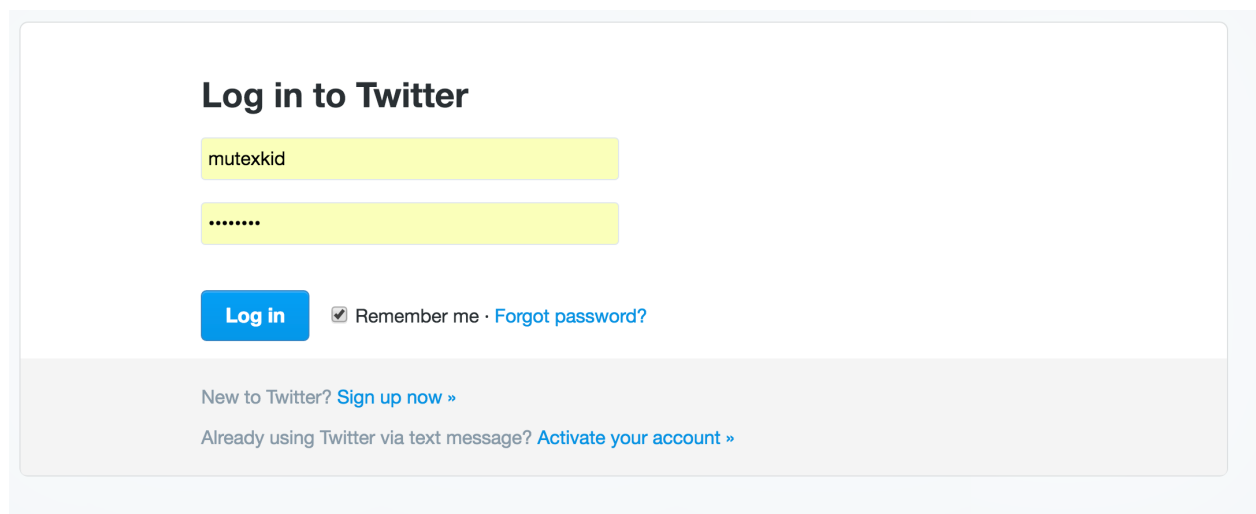
Once you have found the skill, click on it. The skill will currently be enabled, but you would like to test that account linking occurs upon an initial installation. Click Disable.

Figure 1.13 Disabling the Skill



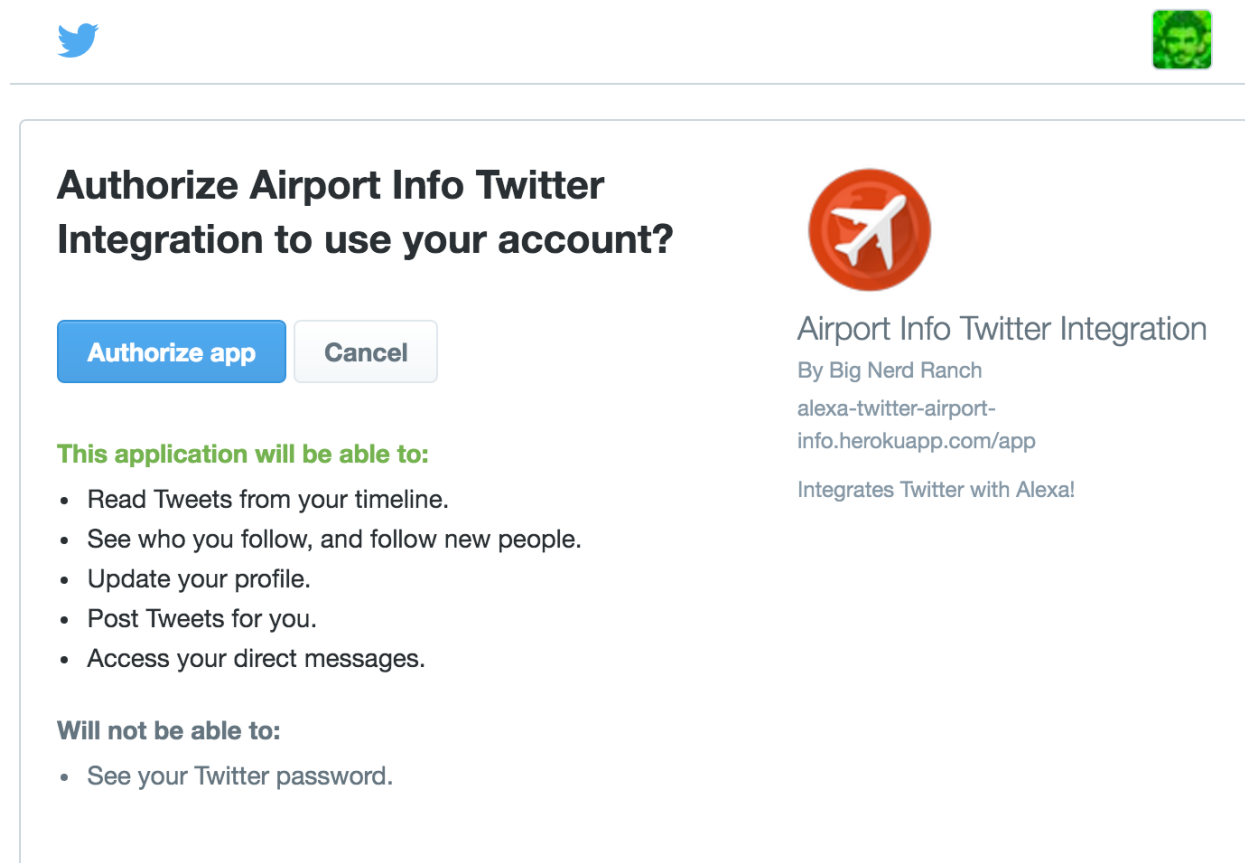
Now, Click Enable. Upon enabling the skill, the skill should redirect to the Twitter Sign In page.

Figure 1.14 Twitter Sign In



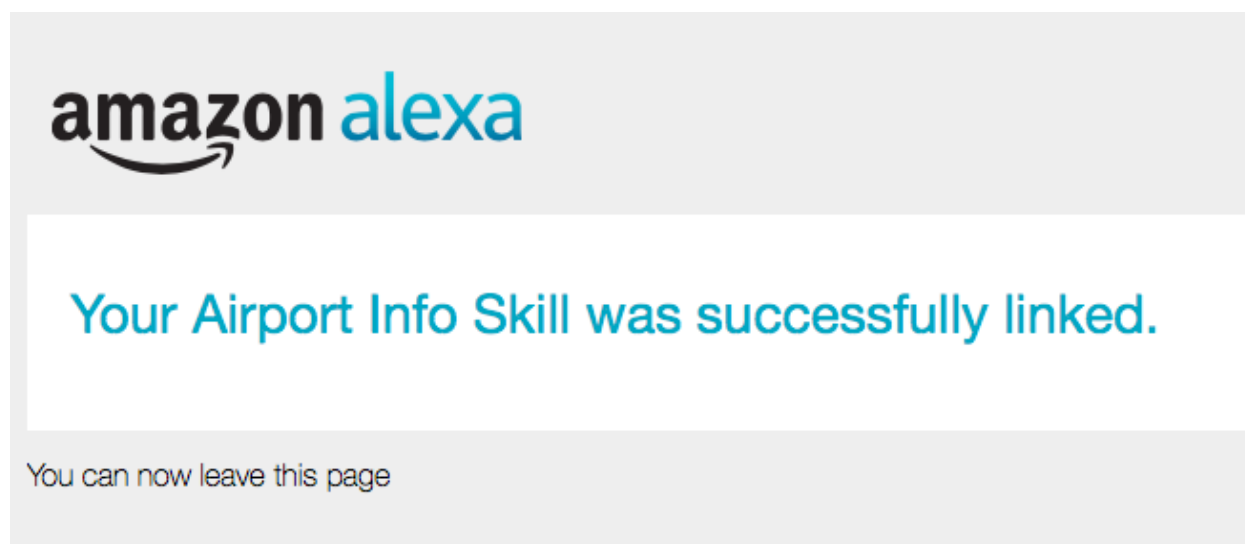
After logging in, you click "Authorize App" to accept the Airport Info Twitter Integration can use your account.

Figure 1.15 Authorizing Twitter



Click Authorize App. The browser should redirect to the skill page in the Alexa App, and display a success message.

Figure 1.16 Account Linking Success



Implementing a TweetAirportStatusIntent Handler

Now you can use the `accessToken` which has been associated with the skill since the account has been linked. You will use the `accessToken` to post a message to the user's Twitter account via the Twitter API. The twitter functionality will be an extension to the Airport Info skill you wrote earlier. Open the existing `index.js` file in the `airportinfo` directory you created during the Slots, Slot Types, and Utterances chapter. Add a new intent handler, called `TweetAirportStatusIntent` to the file. Add the new handler right below the launch intent handler. This handler will respond to the utterance "Alexa, ask airport info to tweet status for {AIRPORTCODE}".

Listing 1.1 Defining the tweetAirportStatusIntent Handler

```
skill.launch(function(request, response) {
  var prompt = 'For delay information, tell me an Airport code.';
  response.say(prompt).reprompt(prompt).shouldEndSession(false);
});
skill.intent('tweetAirportStatusIntent', {
  'slots': {
    'AIRPORTCODE': 'FAACODES'
  },
  'utterances': ['tweet {|delay|status} {|info} {|for} {-|AIRPORTCODE|}']
},
function(request, response) {
  var accessToken = request.sessionDetails.accessToken;
  if (accessToken === null) {
    //no token! display card and let user know they need to sign in
  } else {
    //has a token, post the tweet!
  }
});
```

Upon a user triggering the `tweetAirportStatusIntent` with their voice, the skill pulls the `request.sessionDetails.accessToken` the user has linked with the skill, if present. This is the OAuth value that was returned from the authentication flow.

If present, the skill should post a tweet of the status for the airport they requested. If not present, the skill should display a card instructing the user to log in with Twitter. For handling this situation, the Alexa Skills Kit offers a special card called a `LinkAccount` card. Next, update the case where no token is found with the following code:

Listing 1.2 Handling a Missing Token

```
skill.launch(function(request, response) {
  var prompt = 'For delay information, tell me an Airport code.';
  response.say(prompt).reprompt(prompt).shouldEndSession(false);
});

skill.intent('tweetAirportStatusIntent', {
  'slots': {
    'AIRPORTCODE': 'FAACODES'
  },
  'utterances': ['tweet {|delay|status} {|info} {|for} {-|AIRPORTCODE|}']
},
function(request, response) {
  var accessToken = request.sessionDetails.accessToken;
  if (accessToken === null) {
    response.linkAccount().shouldEndSession(true).say('Your Twitter account is not linked. Please use the Alexa app to link the account.');
```

```
    return true;
  } else {
    //has a token, post the tweet!
  }
});
```

The above code above handles a case where the skill currently finds no token for the user. The `response.linkAccount()` method displays a special card in the user's web browser within the Alexa App that guides them to login to link their Twitter account.

Implementing a TwitterHelper class

Before we can implement the logic to post the tweet, we will first define a new class for managing the Twitter-posting functionality. The new class will make use of an open source library, `twit`. The `twit` library integrates Twitter's REST API in a convenient Node.js library. To install `twit`, navigate to the `airportinfo` directory in the terminal, and run the following command:

Listing 1.3 Installing Twit

```
$ npm install twit --save
```

Now that `twit` is installed, create a new file called `twitter_helper.js` within the `airportinfo` directory. Add the following code to the file:

Listing 1.4 Building the TwitterHelper class

```
'use strict';
module.change_code = 1;
var _ = require('lodash');
var Twitter = require('twit');
var CONSUMER_KEY = 'XXXXX';
var CONSUMER_SECRET = 'XXXXX';
function TwitterHelper(accessToken) {
  this.accessToken = accessToken.split(',');
  this.client = new Twitter({
    consumer_key: CONSUMER_KEY,
    consumer_secret: CONSUMER_SECRET,
    access_token: this.accessToken[0],
    access_token_secret: this.accessToken[1]
  });
}

TwitterHelper.prototype.postTweet = function(message) {
  return this.client.post('statuses/update', {
    status: message
  }).catch(function(err) {
    console.log('caught error', err.stack);
  });
};
module.exports = TwitterHelper;
```

Replace `CONSUMER_KEY` and `CONSUMER_SECRET` values with the matching values you copied during the Twitter App creation step.

The **TwitterHelper** class accepts an access token, and implements the **postTweet** method, which will update the timeline associated with the token. The `twit` library also helpfully returns the results in the form of a promise object, because it is an asynchronous call.

Using the TwitterHelper

Open your `index.js` file. At the top of the file, and import the newly created **TwitterHelper**:

Listing 1.5 Posting the Tweet

```
'use strict';
module.change_code = 1;
var _ = require('lodash');
var Alexa = require('alexa-app');
var skill = new Alexa.app('airportinfo');
var FAADDataHelper = require('./faa_data_helper');
var TwitterHelper = require('./twitter_helper');
...
```

Update the case where the token is found to retrieve the airport status and call the **postTweet(status)** method on a **TwitterHelper** instance.

Listing 1.6 Posting the Tweet

```
skill.launch(function(request, response) {
  var prompt = 'For delay information, tell me an Airport code.';
  response.say(prompt).reprompt(prompt).shouldEndSession(false);
});

skill.intent('tweetAirportStatusIntent', {
  'slots': {
    'AIRPORTCODE': 'FAACODES'
  },
  'utterances': ['tweet {|delay|status} {|info} {|for} {-|AIRPORTCODE|}'],
},
function(request, response) {
  var accessToken = request.sessionDetails.accessToken;
  if (accessToken === null) {
    response.linkAccount().shouldEndSession(true).say('Your Twitter account is not linked.
    Please use the companion app to link the account.');
```

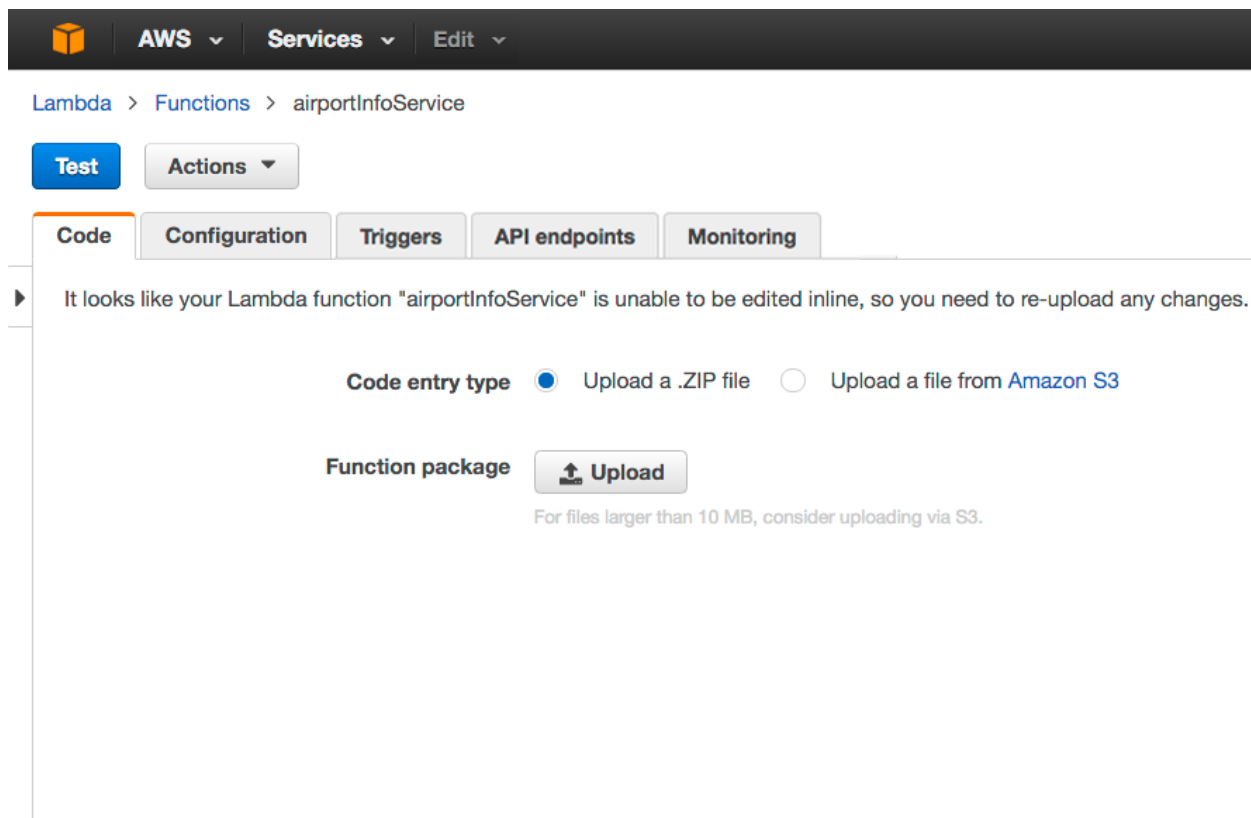
Please use the companion app to link the account.');

```
    return true;
  } else {
    var airportCode = request.slot('AIRPORTCODE');
    if (_.isEmpty(airportCode)) {
      //i've got a token! make the tweet
      var twitterHelper = new TwitterHelper(request.sessionDetails.accessToken);
      var faaHelper = new FAADataHelper();
      var prompt = 'i didn\'t have data for an airport code of ' + airportCode;
      response.say(prompt).send();
    } else {
      faaHelper.getAirportStatus(airportCode).then(function(airportStatus) {
        return faaHelper.formatAirportStatus(airportStatus);
      }).then(function(status) {
        return twitterHelper.postTweet(status);
      }).then(
        function(result) {
          response.say('I\'ve posted the status to your timeline').send();
        }
      );
      return false;
    }
  }
});
```

Updating the Skill Service

Now you will update the AWS Lambda function with your changes to the skill service. Create a new archive including the contents of the `airportinfo` directory. Visit the existing Airport Info Service Lambda function and click Upload, selecting the archive you created of the updated skill service source code.

Figure 1.17 Updating the Service



Updating the Skill Interface

You next update the intent schema and utterances for the AirportInfo Skill. Visit the skill interface in the Amazon Developer Console for the existing AirportInfo skill. Go to the Interaction Model page within the console. Update the Intent Schema section to include the tweetAirportStatusIntent intent:

Listing 1.7 Updating the Intent Schema

```
{
  "intents": [
    {
      "intent": "tweetAirportStatusIntent",
      "slots": [
        {
          "name": "AIRPORTCODE",
          "type": "FAACODES"
        }
      ]
    },
    {
      "intent": "airportInfoIntent",
      "slots": [
        {
          "name": "AIRPORTCODE",
          "type": "FAACODES"
        }
      ]
    }
  ]
}
```


Update the Sample Utterances to include the tweetAirportStatusIntent:

Listing 1.8 Updating the Sample Utterances

```
tweetAirportStatusIntent tweet {AIRPORTCODE}
tweetAirportStatusIntent tweet delay {AIRPORTCODE}
tweetAirportStatusIntent tweet status {AIRPORTCODE}
tweetAirportStatusIntent tweet info {AIRPORTCODE}
tweetAirportStatusIntent tweet delay info {AIRPORTCODE}
tweetAirportStatusIntent tweet status info {AIRPORTCODE}
tweetAirportStatusIntent tweet for {AIRPORTCODE}
tweetAirportStatusIntent tweet delay for {AIRPORTCODE}
tweetAirportStatusIntent tweet status for {AIRPORTCODE}
tweetAirportStatusIntent tweet info for {AIRPORTCODE}
tweetAirportStatusIntent tweet delay info for {AIRPORTCODE}
tweetAirportStatusIntent tweet status info for {AIRPORTCODE}
airportInfoIntent {AIRPORTCODE}
airportInfoIntent flight {AIRPORTCODE}
airportInfoIntent airport {AIRPORTCODE}
airportInfoIntent delay {AIRPORTCODE}
airportInfoIntent flight delay {AIRPORTCODE}
airportInfoIntent airport delay {AIRPORTCODE}
airportInfoIntent status {AIRPORTCODE}
airportInfoIntent flight status {AIRPORTCODE}
airportInfoIntent airport status {AIRPORTCODE}
airportInfoIntent info {AIRPORTCODE}
airportInfoIntent flight info {AIRPORTCODE}
airportInfoIntent airport info {AIRPORTCODE}
airportInfoIntent delay info {AIRPORTCODE}
airportInfoIntent flight delay info {AIRPORTCODE}
airportInfoIntent airport delay info {AIRPORTCODE}
airportInfoIntent status info {AIRPORTCODE}
airportInfoIntent flight status info {AIRPORTCODE}
airportInfoIntent airport status info {AIRPORTCODE}
airportInfoIntent for {AIRPORTCODE}
airportInfoIntent flight for {AIRPORTCODE}
airportInfoIntent airport for {AIRPORTCODE}
airportInfoIntent delay for {AIRPORTCODE}
airportInfoIntent flight delay for {AIRPORTCODE}
airportInfoIntent airport delay for {AIRPORTCODE}
airportInfoIntent status for {AIRPORTCODE}
airportInfoIntent flight status for {AIRPORTCODE}
airportInfoIntent airport status for {AIRPORTCODE}
airportInfoIntent info for {AIRPORTCODE}
airportInfoIntent flight info for {AIRPORTCODE}
airportInfoIntent airport info for {AIRPORTCODE}
airportInfoIntent delay info for {AIRPORTCODE}
airportInfoIntent flight delay info for {AIRPORTCODE}
airportInfoIntent airport delay info for {AIRPORTCODE}
airportInfoIntent status info for {AIRPORTCODE}
airportInfoIntent flight status info for {AIRPORTCODE}
airportInfoIntent airport status info for {AIRPORTCODE}
```

Once you have updated the Intent Schema and Sample Utterances fields, click Save.

Testing the TweetAirportStatus Intent

To test the new TweetAirportStatus intent handler, visit the skill interface in the Amazon Developer Console and go to the Test page for the AirportInfo skill. Under the Service Simulator section, enter "tweet status for ATL" and click Ask Airport Info. Within the Lambda Response area, you should see the following response:

Listing 1.9 Posting the Tweet

```

{
  "version": "1.0",
  "response": {
    "outputSpeech": {
      "type": "SSML",
      "ssml": "<speack>I've posted the status to your timeline</speak>"
    },
    "shouldEndSession": true
  },
  "sessionAttributes": {}
}

```

Figure 1.18 Testing the Twitter Integration via Service Simulator

Service Simulator

Use Service Simulator to test your lambda function.

Text
Json

Enter Utterance *

tweet status for ATL

Ask Airport Info
Reset

Lambda Request

```

1 {
2   "session": {
3     "sessionId": "SessionId.d77b6dae-0295-4b30-98
4     "application": {
5       "applicationId": "amzn1.echo-sdk-ams.app.d6
6     },
7     "user": {
8       "userId": "amzn1.ask.account.AFP3ZWPOS2BGJR
9       "accessToken": "15166940-fqA2cYQGDJkjqZEKAc
10    },
11    "new": true
12  },
13  "request": {
14    "type": "IntentRequest",
15    "requestId": "EdwRequestId.d6ef7354-8aae-417b
16    "timestamp": "2016-04-20T20:01:33Z",
17    "intent": {
18      "name": "tweetAirportStatusIntent",
19      "slots": {
20        "AIRPORTCODE": {
21          "name": "AIRPORTCODE",
22          "value": "ATL"
23        }
24      }
25    }
26  }
27 }

```

Lambda Response

```

1 {
2   "version": "1.0",
3   "response": {
4     "outputSpeech": {
5       "type": "SSML",
6       "ssml": "<speack>I've posted the status to y
7     },
8     "shouldEndSession": true
9   },
10   "sessionAttributes": {}
11 }

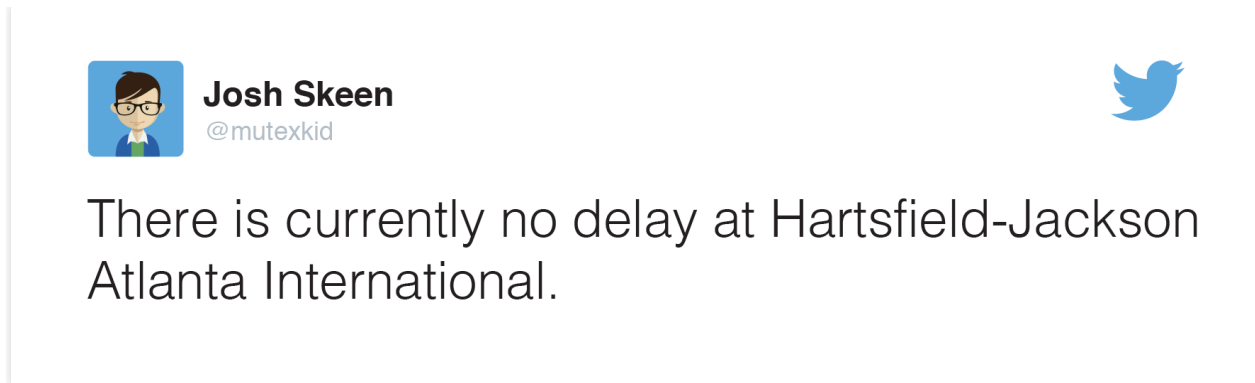
```

Listen

Submit for Certification
Next

Now, visit your Twitter timeline. You should see a new tweet indicating the status of the airport you specified.

Figure 1.19 Airport Status Tweet



Congratulations! You have successfully implemented account linking to integrate an external user account with your skill.

For more information and use cases with the account linking feature, check out:

<https://goo.gl/7x1Gwa>

