Table of Contents

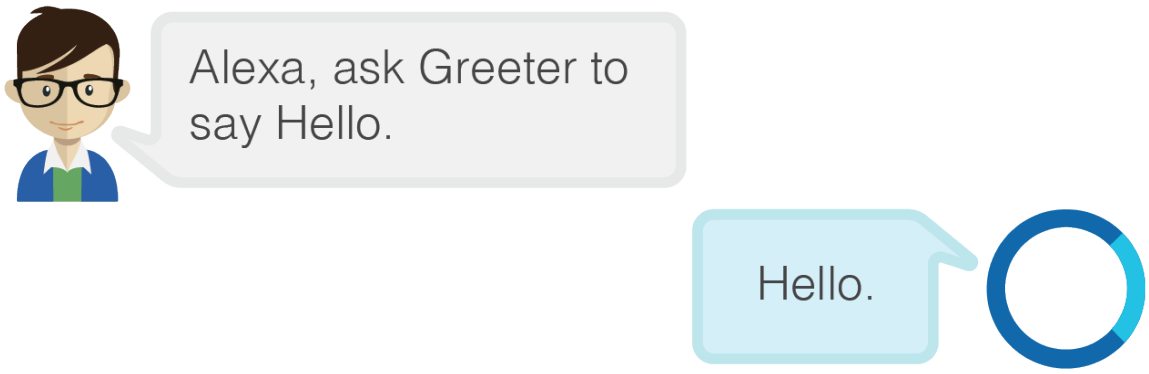
1. Hello Alexa 1
    Creating an Account 2
    Creating the Skill Service 2
    Defining index.js 3
    Defining the onLaunch handler 3
    Defining an Intent Handler 4
    Obtaining an Application ID 4
    Deploying the Service to AWS Lambda 6
    Configuring the Alexa Skill Interface 9
    Defining an Intent Schema and Sample Utterances 10
    Specifying the ARN 11
    Testing the Interaction with the Service Simulator 12
    Understanding the Greeter Skill 14
    Silver Challenge: Bonjour, Alexa! 14
    Gold Challenge: Good Morning, Good Afternoon, Good Night! 14
    Platinum Challenge: ES6 support 15

1

Hello Alexa

In this chapter, you will build and deploy a basic Alexa skill called "Greeter". This exercise will leverage the basic requirements to get a skill up and running, and show you the basic components of a skill. The illustration shows an interaction with the Greeter skill. The user invokes Greeter by saying "Alexa, ask Greeter to say Hello" and Alexa responds with "Hello".

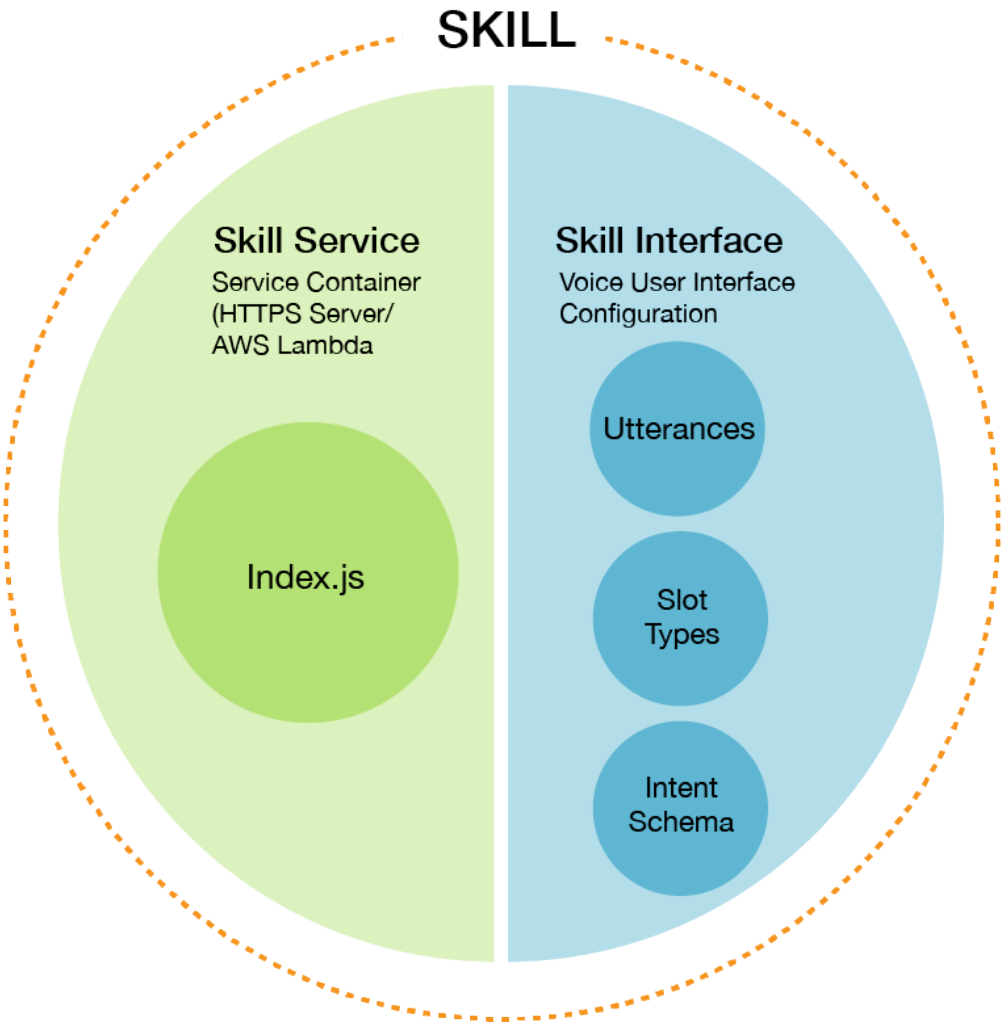
Figure 1.1



The Greeter skill will consist of two portions - a skill *service* you will deploy to a remote service container, and a skill *interface* you will register on Amazon's Alexa skill console. For the service portion, you will be working with AWS Lambda, a cloud-based service hosting platform and server environment.

While there are many options for hosting your skill's service code over HTTPS, you will be using AWS Lambda because of it's streamlined interface within the Alexa skill development pipeline. It is possible to use your own HTTPS server, but to do so it requires additional configuration to enable SSL and a signed certificate. No additional configuration is required with AWS Lambda.

Figure 1.2 A skill consists of a skill service and skill interface



Creating an Account

Before beginning development, you should ensure that you have registered an AWS account with Amazon first. Visit

```
https://aws.amazon.com/lambda/
```

and log in. Check that you have access to the AWS Lambda console located there and that your Lambda account is active. If not, register an account and follow the steps for enabling AWS Lambda on your AWS account as prompted in the signup process.

Creating the Skill Service

You will begin by building the service portion of the skill. Now that you have verified an active AWS Lambda account, you will begin developing the skill by creating a new directory called `greeter`. Download the base `AlexaSkill` module, located at

```
https://raw.githubusercontent.com/amzn/alexa-skills-kit-js/master/samples/scoreKeeper/src/AlexaSkill.js
```

`AlexaSkill.js` will provide a base set of functionality for your skill that you will extend to flesh out a skill service with. Save this file to the `greeter` directory you created.

An Alexa skill service can be written using any platform that may be hosted on an HTTPS server endpoint. For this course, you will be working with JavaScript and the Node.js runtime. Node.js is a supported language on AWS Lambda and is supported by a vibrant open source community. Node.js is also convenient to develop in and debug, requiring a minimal toolchain for development with. The Node.js runtime enables JavaScript to run on the server. You will be deploying to AWS Lambda, a cloud based server environment that supports the Node.js runtime.

Defining index.js

Within the `greeter` directory add a new file called `index.js`, where you will now begin implement the skill service for Greeter using Node.js/JavaScript.

Listing 1.1 Creating greeter/index.js

```
'use strict';
var APP_ID = undefined;
var AlexaSkill = require('./AlexaSkill');

var GreeterService = function() {
  AlexaSkill.call(this, APP_ID);
};
GreeterService.prototype = Object.create(AlexaSkill.prototype);
```

You have now defined a **`GreeterService`** function which inherits from the `AlexaSkill.js` class. In the next sections you will build upon this base, defining how the skill service will handle requests from the skill interface.

Notice you have also enabled strict mode in your JavaScript file. This will help catch common JavaScript programming blunders. For example when strict mode has been enabled an error will be thrown if you assign a string to an undefined variable. It is advised to always use strict mode in JavaScript programs. For a detailed (somewhat lengthy) write-up of how strict mode works, check out

```
http://ejohn.org/blog/ecmascript-5-strict-mode-json-and-more/.
```

Defining the onLaunch handler

You will next add an **`onLaunch`** event handler to the GreeterService function. The **`onLaunch`** event handler will be invoked when the user first launches or opens the skill with it's invocation name, which you will specify later. To add the **`onLaunch`** event handler, add the following to `index.js`:

Listing 1.2 Adding an onLaunch Event Handler

```
'use strict';
var APP_ID = undefined;
var AlexaSkill = require('./AlexaSkill');
var SPEECH_OUTPUT = 'Hello';

var GreeterService = function() {
  AlexaSkill.call(this, APP_ID);
};
GreeterService.prototype = Object.create(AlexaSkill.prototype);

var helloResponseFunction = function(intent, session, response) {
  response.tell(SPEECH_OUTPUT);
};

GreeterService.prototype.eventHandlers.onLaunch = helloResponseFunction;
```

You also defined a **`helloResponseFunction`** function. This function will build a response to the Alexa skill interface that tells Alexa how to respond to the user's request. The **`onLaunch`** event will be fired by the skill interface and sent to the service when the Alexa skill is started with nothing else said. For example, the **`onLaunch`** event is triggered if

the skill is invoked with the phrase "Alexa, open Greeter" or "Alexa, start Greeter". You will learn more about the specifics of how the skill interface is configured soon.

Defining an Intent Handler

An *intent* is a description of what a user would like to accomplish that is sent to the skill service from the skill interface. To define how the service will handle the intent, you will add what is called an intent handler.

A user's request is resolved to the handler by providing the skill interface a list of "utterances" you will soon configure. In this simple example, the **`onLaunch`** handler will do the same thing your intent handler will do - respond with "Hello". You will assign the same **`helloResponseFunction`** to the new intent handler definition. Modify `index.js` to include a 'HelloWorldIntent' intent handler.

```
'use strict';
var APP_ID = undefined;
var AlexaSkill = require('./AlexaSkill');
var SPEECH_OUTPUT = 'Hello';

var GreeterService = function() {
  AlexaSkill.call(this, APP_ID);
};
GreeterService.prototype = Object.create(AlexaSkill.prototype);

var helloResponseFunction = function(intent, session, response) {
  response.tell(SPEECH_OUTPUT);
};
GreeterService.prototype.eventHandlers.onLaunch = helloResponseFunction;

GreeterService.prototype.intentHandlers = {
  'HelloWorldIntent': helloResponseFunction
};
```

Next, you will add a Lambda handler method definition. This definition will allow the skill service you have written to run on AWS Lambda correctly.

Listing 1.3 Adding the AWS Lambda Handler

```
...
var helloResponseFunction = function(intent, session, response) {
  response.tell(SPEECH_OUTPUT);
};
GreeterService.prototype.eventHandlers.onLaunch = helloResponseFunction;

GreeterService.prototype.intentHandlers = {
  'HelloWorldIntent': helloResponseFunction
};

exports.handler = function(event, context) {
  var GreeterService = new GreeterService();
  GreeterService.execute(event, context);
};
```

Now that you have added the handler, AWS Lambda will be able to route the event and context information sent from the skill interface to your skill service as JSON data. This JSON payload includes session, environment, and information about the request from the Alexa account from which the skill was invoked. You will use more of these attributes as you work through the course.

Obtaining an Application ID

Before adding the skill service code you have written to AWS Lambda, you will need an Application ID from the Amazon skill interface. This will ensure the requests made from the skill interface to the skill service are from the correct source.

Go to the page

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

and click add a New Skill. Enter "Greeter" for Name, and "Greeter" for *invocation name*. The *invocation name* you specify here is how customers will address your skill when speaking with Alexa. In the diagram below, the example interaction shows the invocation name in use with the Greeter skill.

Figure 1.3 Defining the Invocation Name

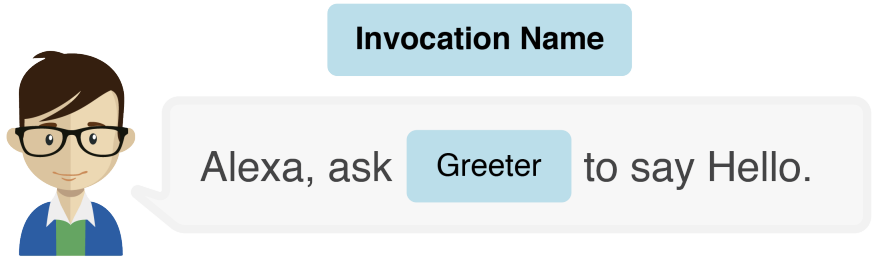
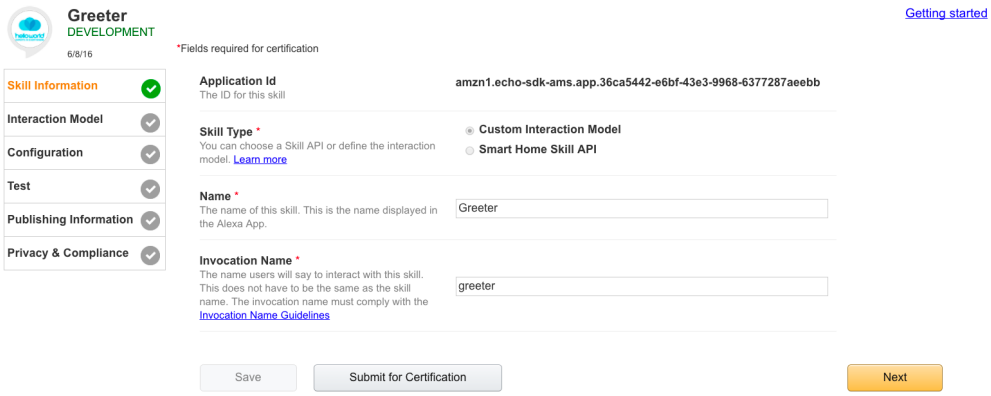


Figure 1.4 Copying the Application ID



Copy the Application ID value from the Skill Information step, and update the `APP_ID` variable in `index.js`.

Listing 1.4 Adding the Application ID

```
'use strict';
~~var APP_ID = undefined;~~
var APP_ID = 'amzn1.echo-sdk-ams.app.21133313-882b-4dcf-a90a-123123dd1ad';
var AlexaSkill = require('./AlexaSkill');
var SPEECH_OUTPUT = 'Hello World!';

var GreeterService = function() {
  AlexaSkill.call(this, APP_ID);
};
...
```

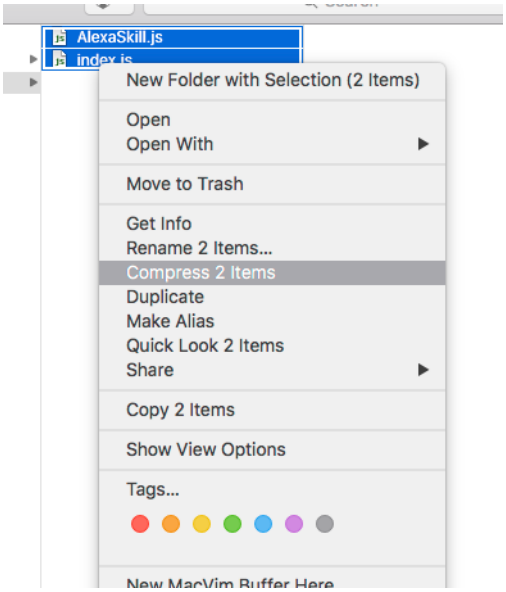
Deploying the Service to AWS Lambda

Next, you deploy the skill service to AWS Lambda. To begin deployment, visit:

```
https://console.aws.amazon.com/console/
```

and click on Lambda. To upload the skill service, create an archive of the files within `greeter` by selecting all of the files in the directory, control clicking and selecting Compress.

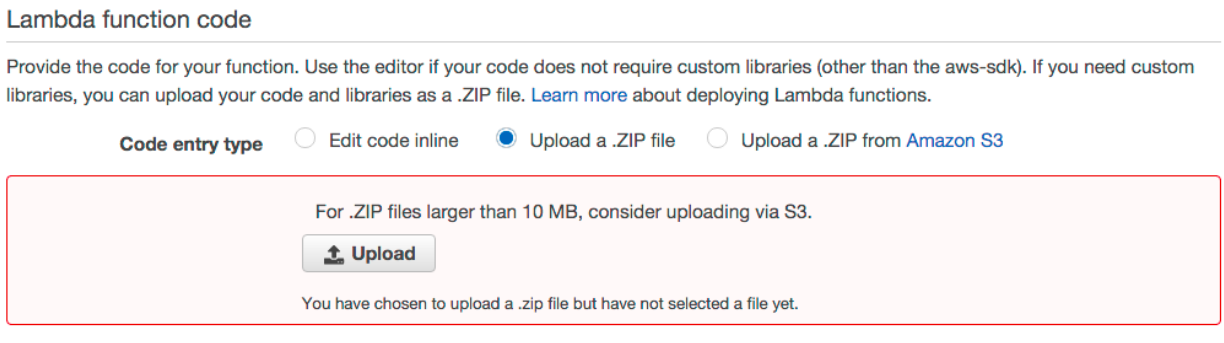
Figure 1.5 Compressing the Skill Service Files



Next, click on Get Started Now. On the Select blueprint page, click Skip. For the Name field, enter "GreeterService". For the Runtime field, select "Node.js 4.3". For the Role dropdown, select Basic Execution Role, "lambda_basic_execution". Upon making the selection the browser will redirect to a security page where you click Allow to enable the new Role selection.

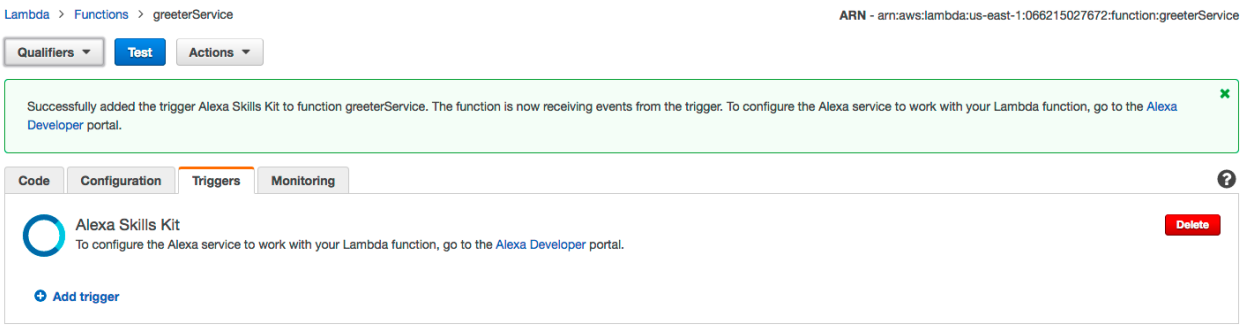
Select Upload a .ZIP file under code Code entry type and upload the archive you created previously by selecting it after clicking the Upload button.

Figure 1.6 Uploading the Skill Service Archive



Now, click Next and Create Function. On the resulting screen, click Triggers and Add trigger. For Triggers, select Alexa Skills Kit and click Submit.

Figure 1.7 Alexa Skills Kit Trigger Enabled



Notice the value displayed in the top right of the AWS Lambda Management Console. This is the ARN, or *Amazon Resource Name*, which you will use to configure the skill interface. The ARN serves as an address that points to your skill service and indicates where requests should be routed. You will require the ARN value in several steps - copy the ARN down to a text file so that you have it ready when needed.

The skill service should now be ready to receive requests from the skill interface. Next, you verify the skill service works correctly. Click the Actions dropdown and select Configure test event.

Figure 1.8 Configuring the Test Event 1/2

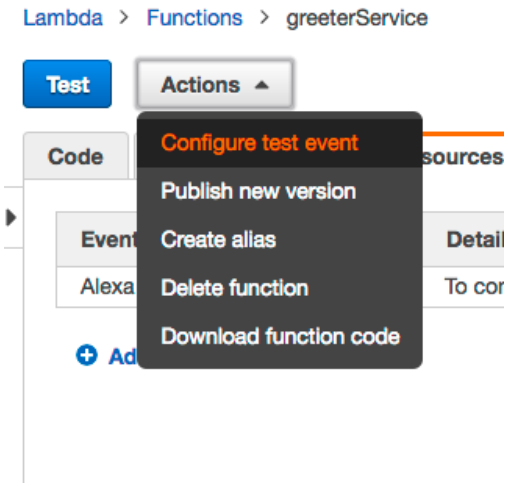
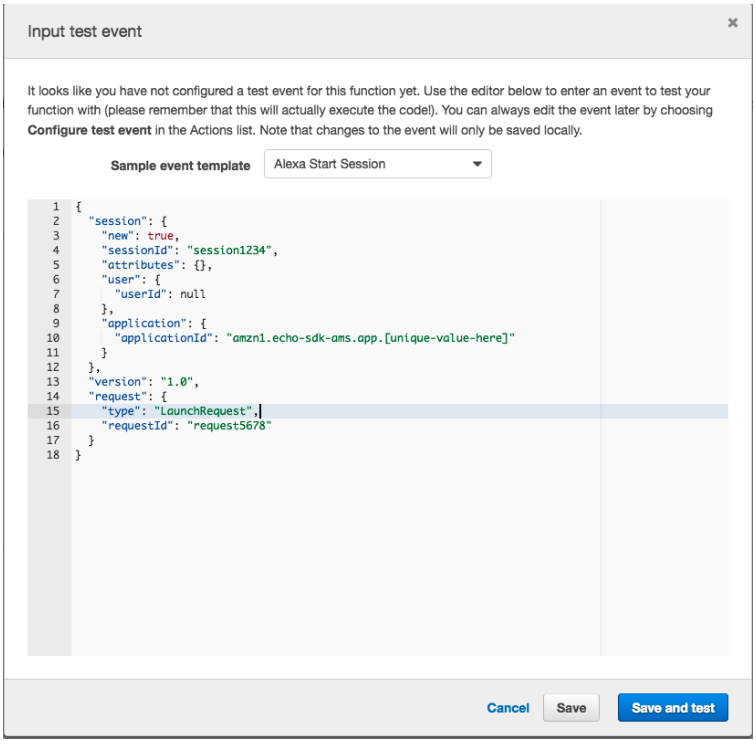


Figure 1.9 Configuring the Test Event 2/2



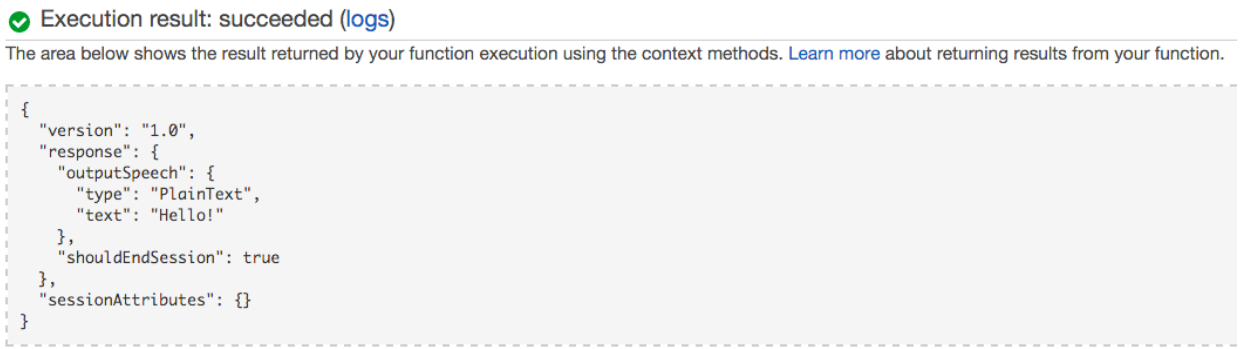
Select Alexa Start Session under the Sample event template. Before clicking Save and test you will need to change the `applicationId` value in the request template. Use the Application ID you obtained from the skill interface previously.

Listing 1.5 Adding Application ID to the Test Event

```
{
  "session": {
    "new": true,
    "sessionId": "session1234",
    "attributes": {},
    "user": {
      "userId": null
    },
    "application": {
      "applicationId": "amzn1.echo-sdk-ams.app.[unique-value-here]"
      "applicationId": "amzn1.echo-sdk-ams.app.36ca5442-e6bf-43e3-9968-6377287aeebb"
    }
  },
  "version": "1.0",
  "request": {
    "type": "LaunchRequest",
    "requestId": "request5678"
  }
}
```

Press the Save and test button. You should see something similar to the following appear in the Execution Result pane:

Figure 1.10 Inspecting the Execution Results



Configuring the Alexa Skill Interface

Notice the data that is displayed in the Execution Results pane. This is the JSON response from the skill service with a payload instructing the Alexa-enabled device to say "hello".

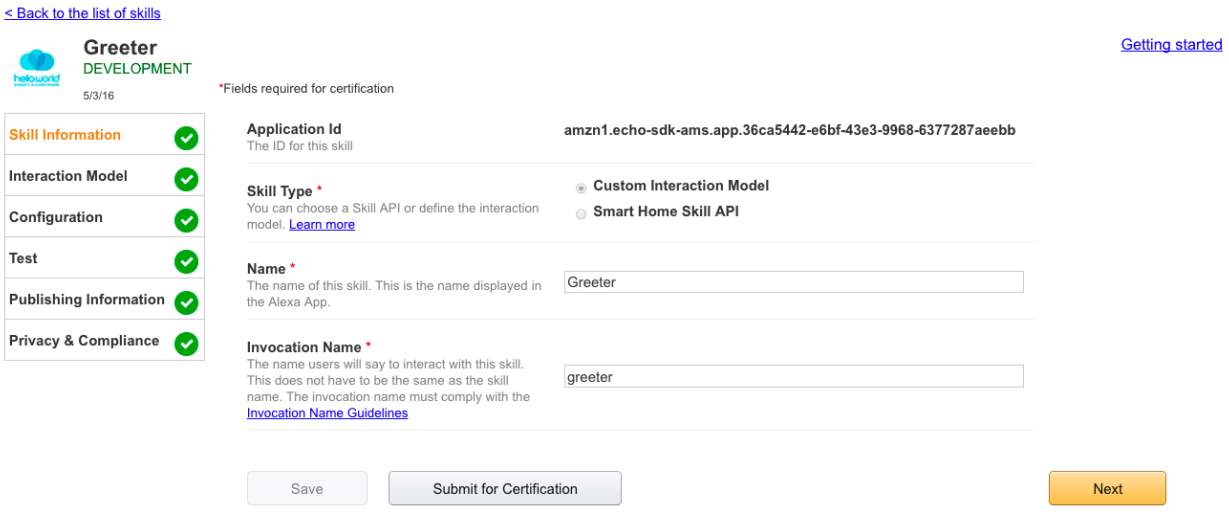
Now that the skill service is set up, you configure the skill interface in the Alexa Developer console. Visit

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

and click on the "Greeter" skill you began configuring earlier.

Verify that within "Skill Information" you have "Name" and "Invocation name" set to "Greeter".

Figure 1.11 Configuring the Skill Information



Defining an Intent Schema and Sample Utterances

Click on Interaction Model in the side area. You will provide an *intent schema* and *sample utterances* list the skill interface configuration. Under the Intent Schema field enter the following:

Listing 1.6 Adding application ID to the Test Event

```
{
  "intents": [
    {
      "intent": "HelloWorldIntent"
    }
  ]
}
```

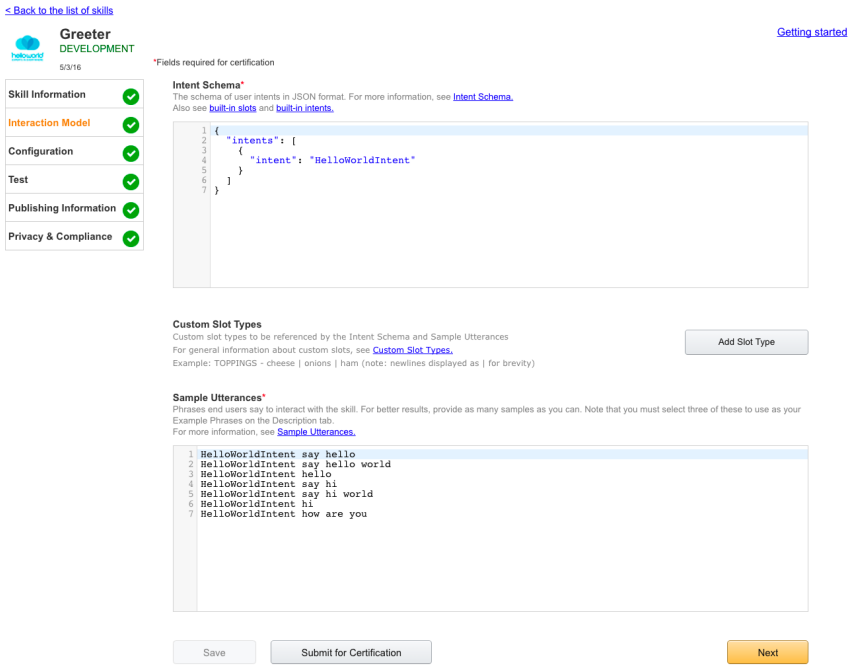
This schema will let the skill interface know that your skill can handle an `Intent` called `HelloWorldIntent` that can be resolved to by spoken utterance.

Next, you provide *sample utterances* for the `HelloWorldIntent`. In the Sample Utterances field, enter the following:

Listing 1.7 Sample Utterances for Greeter

```
HelloWorldIntent say hello
HelloWorldIntent say hello world
HelloWorldIntent hello
HelloWorldIntent say hi
HelloWorldIntent say hi world
HelloWorldIntent say hey there world
HelloWorldIntent hi
HelloWorldIntent how are you
```

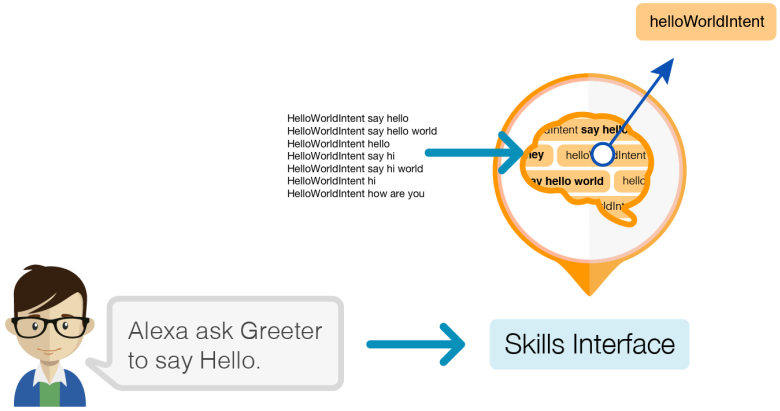
Figure 1.12 configuring utterance and interaction model



This list of sample utterances allows the Alexa skill interface to resolve spoken words to an intent. It is not required that all of the words that are specific to matching the intent are contained in the sample utterances list, though providing a more comprehensive set of samples to build the model from improves the chances of a match. Click the Next button to save this configuration.

The list of sample utterances that you defined allows the skill interface to resolve spoken words to intent events (see diagram below).

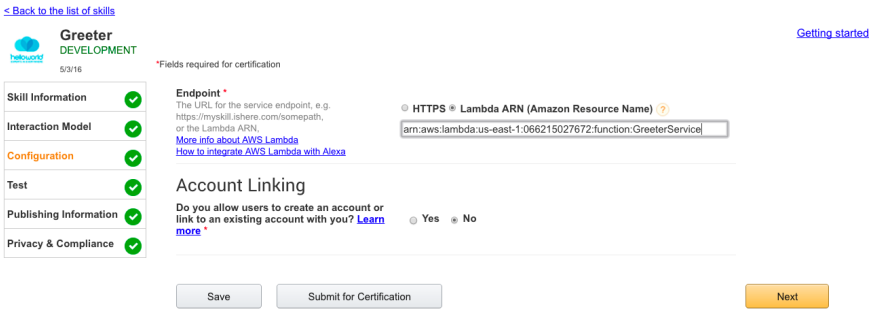
Figure 1.13 Sample Utterances Resolve Spoken Words to Intents



Specifying the ARN

Now you specify the endpoint for your skill service you uploaded by providing the ARN value you copied down earlier. Select Lambda ARN for Endpoint. Paste the ARN value into the Endpoint field. For the Account Linking option, select No and click Next. You will work with Account Linking at a later time.

Figure 1.14 Providing the Service ARN



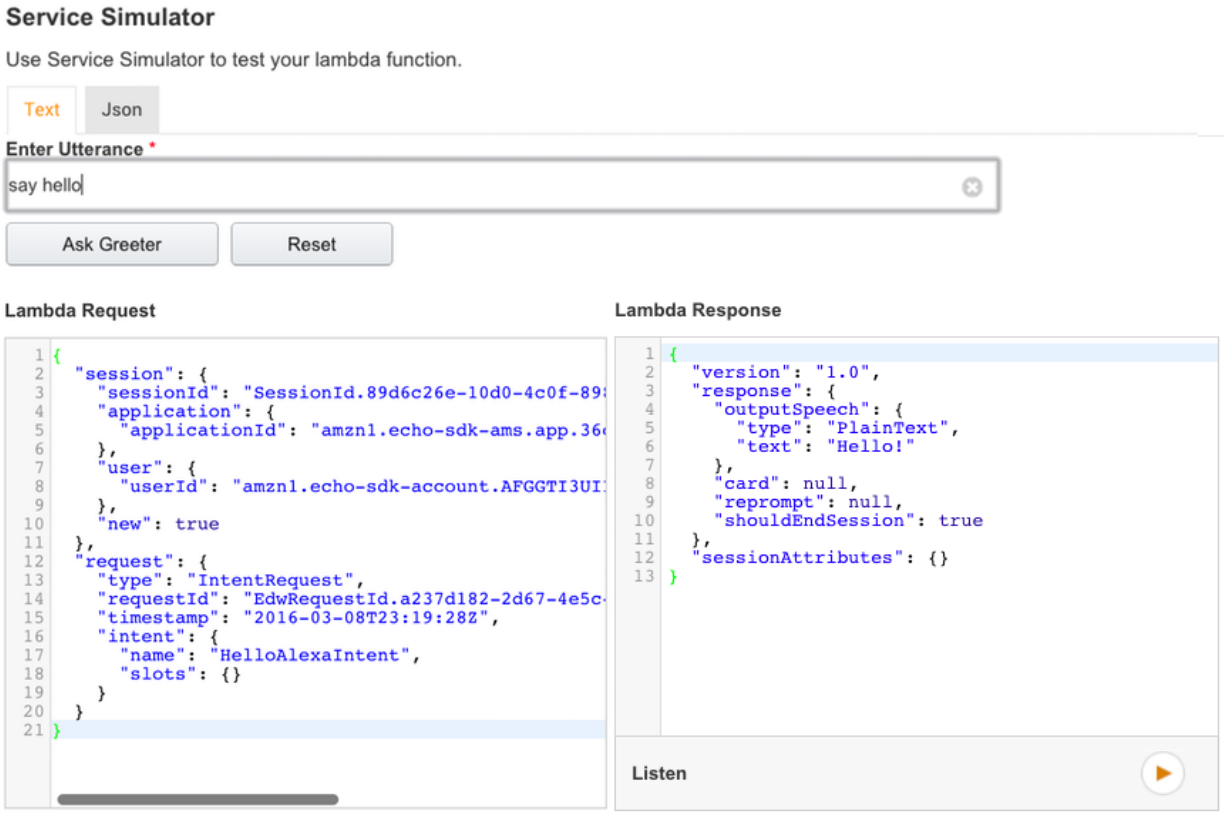
Testing the Interaction with the Service Simulator

Now that the skill service and skill interface are both configured, you may test an interaction with the skill. Make sure you are on the Test section within the Greeter skill interface configuration. You will see the Service Simulator on this page. Under the field "Enter Utterance", enter "say hello" and click "ask Greeter". This would be equivalent to speaking to an Echo and saying "Alexa, ask Greeter to say hello". Verify that the Lambda response pane contains text similar to the following:

Listing 1.8

```
{
  "version": "1.0",
  "response": {
    "outputSpeech": {
      "type": "PlainText",
      "text": "Hello"
    },
    "card": null,
    "reprompt": null,
    "shouldEndSession": true
  },
  "sessionAttributes": {}
}
```

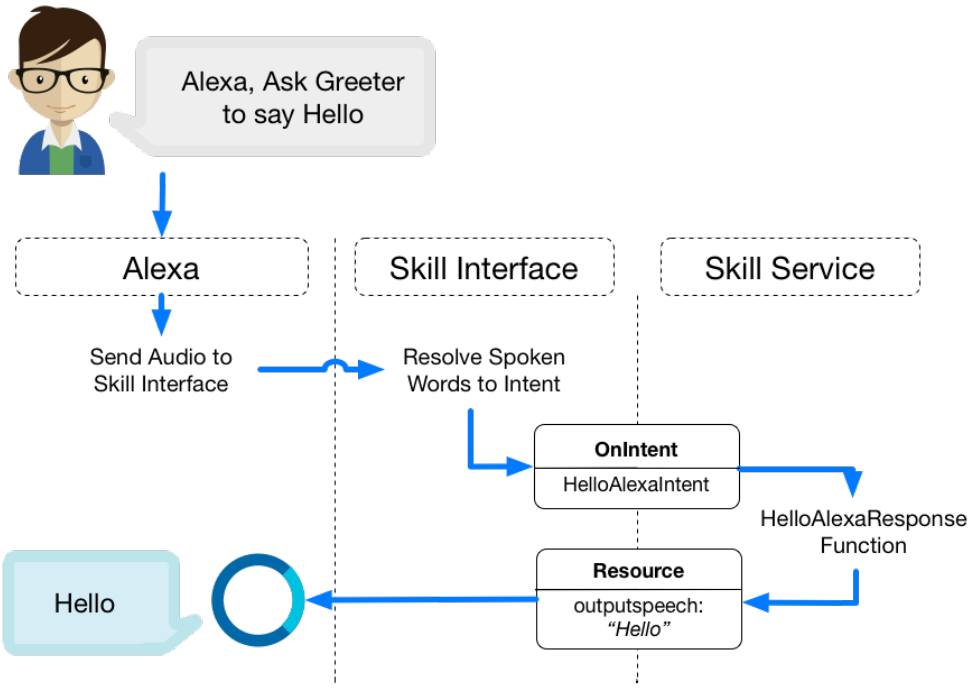
Figure 1.15 Service Simulator Results



Congratulations! You have successfully completed your first Alexa skill. If you have an Echo available, sign in with the developer account that you created this skill under to test it on an Alexa-enabled device.

Understanding the Greeter Skill

Figure 1.16 For the More Curious: A Skill's Typical Lifecycle



`Greeter` shows the typical situation with a skill interaction between the user, Echo device, skill interface, and skill service. Echo routes spoken words from the user to the skill interface, where a determination can be made about the intent from the sample utterances. At this point, a request containing a JSON payload describing what the skill interface resolved is sent to the skill service. The skill service's resulting JSON response is then returned to the skill interface where it is forwarded on to the Echo and finally played back.

The technology provided by the skill interface that resolves spoken words into intents is called the *Natural Language Processing* classifier. Natural Language Processing is an actively advancing field of research within the Machine Learning space. Simply stated, Natural Language Processing applies statistical techniques and the latest AI research to the problem of resolving a user's spoken words to intents the skill service can act upon. For a more in-depth summary of NLP, check out

```
https://en.wikipedia.org/wiki/Natural_language_processing
```

Silver Challenge: Bonjour, Alexa!

A skill can feature many intent handlers, each responding to different sample utterance mappings. For this extra challenge, register a new intent within the schema and provide sample utterances for handling a (slightly) French speaker's request to your skill: "Alexa, ask greeter to say Bonjour!". The new version of the skill should handle both old requests to say "hello" and the new request that returns "bonjour!" instead. Solving this challenge will require changes to both the skill service and the skill interface.

Gold Challenge: Good Morning, Good Afternoon, Good Night!

Saying Hello is good, but a skill that appropriately responds with "Good Morning", "Good Afternoon" and "Good Night" is even better! For this challenge, extend the HelloWorldIntent handler to check the current time on the

server. Use the following rules to determine the response: If the time is between 12 AM and 12 PM, the response should be "Good Morning!". If the time is between 12 PM and 5 PM, the response should be "Good Afternoon!". If the time is between 5 PM and 12 AM, the response should be "Good Night!". Do not worry about Timezone support in this exercise, the server's current time without accounting for a user's location will work for the challenge! As a starting place, JavaScript has the helpful `Date` class for doing things with time.

```
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date
```

Platinum Challenge: ES6 support

If you are already familiar with Node.js, you may also be familiar with the different features available in the most recent version of JavaScript, ES6. ES6 support was recently added to the AWS Lambda runtime, and is an option for further simplifying the required code to implement a skill in Node.js. Rewrite the skill service to make use of the new lambda shorthand syntax and const keyword where appropriate. For example, an ES6 rewrite of the `helloResponseFunction` would look like this:

Listing 1.9 Rewriting helloResponseFunction with ES6 Lambdas

```
var helloResponseFunction = function(intent, session, response) {
  response.tell(SPEECH_OUTPUT);
};
var helloResponseFunction = (intent, session, response) => { response.tell(SPEECH_OUTPUT); };
```