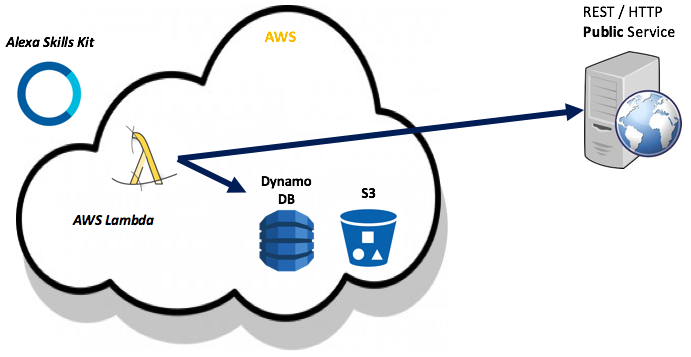
## Publishing Both Code and Data to AWS Lambda

Robert McCauley  
Tags: Alexa, How To, APIs

Welcome to Alexa! We’ve been talking about how to get started with Alexa using the [Alexa Skills Kit page](https://developer.amazon.com/ask), and sample skills, such as the [Color Expert](https://developer.amazon.com/public/solutions/alexa/alexa-skills-kit/docs/developing-an-alexa-skill-as-a-lambda-function), using [AWS Lambda](https://console.aws.amazon.com/lambda/) functions. In a previous post, we learned a way to develop your skill code locally, and publish to AWS Lambda via a command line script. Be sure you have completed that exercise before continuing here.

### The Scenario

AWS Lambda code can easily make calls to public and cloud resources, such as Dynamo DB, S3, and other public web service endpoints. As globally visible services, and will respond and serve network requests from AWS Lambda.

Many companies would like to have their skill Lambda code be able to call into their internal network, for example to access a CRM or ERP database, or read from an Intranet filesystem. However, this is usually not feasible, as these private systems sit behind a corporate firewall which blocks all externally initiated requests. It may be a long and difficult process to have the IT security group create a firewall exception, or “hole”, to allow outside requests to pull corporate data.

### A Solution

Let us re-define the problem. The goal is for the skill to have corporate data accessible to the Alexa skill. We can achieve this by inverting the connection model. We can create a process to publish corporate data directly into a Lambda function on a frequent basis. Once the data is in Lambda, it can be accessed by Skill users without needing any further network requests. By enhancing our locally-developed project in the previous blog post, and automating the publish task on a schedule, private data can be made available to Skill users.

### Case Study

### We own and operate a local grocery store. Our customers telephone us to ask how many apples, oranges, and other fruits are currently available. Our clerk logs into to the store’s inventory database and performs a lookup of the requested item, and tells the customer the current on-hand quantity, such as “we have 12 oranges in stock”.

#### The Skill

### We want to develop an Alexa skill to answer these types of questions with voice. We create a new skill with invocation name “inventory dashboard”. The skill is configured like so:

|  |
| --- |
| Intent Schema |
| {  "intents": [  {  "intent": "QuantityOnHandIntent",  "slots":[  {  "name":"Item",  "type":"LIST\_OF\_ITEMS"  }  ]  },  {  "intent": "EndIntent"  }  ]  } |

|  |
| --- |
| LIST\_OF\_ITEMS custom slot |
| apples  bananas  carrots  oranges  watermelons  peaches  pears  plums  lemons  limes  cherries  melons  blueberries  raspberries  strawberries  nectarines  tangerines  clementines |

|  |
| --- |
| Sample Utterances |
| QuantityOnHandIntent tell me about {Item}  QuantityOnHandIntent how many {Item}  QuantityOnHandIntent how many {Item} are there  QuantityOnHandIntent how many {Item} do we have  QuantityOnHandIntent how many {Item} are in stock  QuantityOnHandIntent how many {Item} exist  QuantityOnHandIntent how many {Item} are on hand  QuantityOnHandIntent do we have {Item}  QuantityOnHandIntent are there {Item}  QuantityOnHandIntent {Item}  EndIntent thanks  EndIntent thank you  EndIntent great  EndIntent goodbye |

#### The Code

Create a sub-folder called “src”

Create a new javascript file called “index.js”

Paste in the following code.

|  |
| --- |
| index.js |
| // Inventory Dashboard Alexa Skill - publish your data with your code!  **var** dataset = require('./datafiles/dataset.js');  exports.handler = **function**( event, context ) {  **var** say = "";  **var** endsession = **false**;  **var** sessionAttributes = {};   **var** myItem = "apples";  **var** Qty = 0;    **if** (event.session.attributes) {  sessionAttributes = event.session.attributes;  }   **if** (event.request.type === "LaunchRequest") {  say = "Welcome to Supermarket Inventory! You can ask me about apples, oranges, and other items.";   } **else** {  **var** IntentName = event.request.intent.name;   **if** (IntentName === "QuantityOnHandIntent") {   **if**(event.request.intent.slots.Item.value) {   myItem = event.request.intent.slots.Item.value;   **if** (!sessionAttributes.ItemList) {sessionAttributes.ItemList = []; }  sessionAttributes.ItemList.push(myItem );   **for** (**var** i = 0; i < dataset.length; i++) {  **if**(myItem == dataset[i].item) {  Qty = dataset[i].qty;  }  }   say = "there are " + Qty + " " + myItem ;   } **else** {  say = "you can say things like, tell me about oranges";  }   } **else if** (IntentName === "EndIntent") {  say = "You asked for " + sayItems(sessionAttributes.ItemList.toString()) + " Thank you for visiting!";  endsession = **true**;   }  }   **var** response = {  outputSpeech: {  type: "SSML",  ssml: "<speak>" + say + "</speak>"  },  reprompt: {  outputSpeech: {  type: "SSML",  ssml: "<speak>Please try again. " + say + "</speak>"  }  },  card: {  type: "Simple",  title: "My Card Title",  content: "My Card Content, displayed on the Alexa Companion mobile App or alexa.amazon.com"  },   shouldEndSession: endsession  };     Respond( // Respond with normal speech only  **function**() {context.succeed( {sessionAttributes: sessionAttributes, response: response } ); }  );  };  // -----------------------------------------------------------------------------  **function** Respond(callback) {  callback(); // execute the caller's context.succeed function to complete }   **function** sayItems(items) { // format a comma separated array or list of items, to include the word "and" before the final item  **var** startingList = "";  **if** (**typeof** items == "string") {  startingList = items;  } **else** {  startingList = items.toString();  }   **var** formattedString = "";  **var** lastComma = startingList.lastIndexOf(",");   **if** (lastComma == -1) { // single element  formattedString = startingList;  } **else** {  formattedString = startingList.substr(0,lastComma) + ", and " + startingList.substr(lastComma +1, 1000);  formattedString = formattedString.replace(",", ", ");  formattedString = formattedString.replace(" ", " ");  }  **return** formattedString;  } |

#### The Data

Create a new folder, underneath the “src” folder, called “datafiles”.

Within the datafiles folder, create a new file called “dataset.js”

|  |
| --- |
| dataset.js |
| var dataset = [  {item:"apples", qty:"10"},  {item:"bananas", qty:"7"},  {item:"carrots", qty:"0"},  {item:"oranges", qty:"14"},  {item:"watermelons", qty:"5"}  ];  module.exports=dataset; |

This file represents the corporate data we wish to embed in our Lambda function. The file is static for now, but can be used to test that the lookup function is working correctly.

The code in index.js will be called when a user asks for the status of a particular item. The code will receive this as a slot value, iterate through this array, and attempt to match the Item slot value with one of the item elements. If a match is made, the “qty” value will be accessed and returned to the user via a spoken utterance.

#### The Setup

Zip both index.js and datafiles/dataset.js into a new zip archive.

Within the AWS Lambda console, create a new function called InventoryDashboard, specifying “Alexa ASK” as an event source. Use the default IAM role, “lambda\_basic\_execution”.

Upload the zip archive and click Create Function.

Test your skill by asking “Alexa, ask Inventory Dashboard how many oranges are in stock?”

#### The magic

We can now write a separate program or script to generate this data file automatically. For example, a script could connect to a MySQL database, query a table of items, and return “item” and “qty” columns in a JSON format. With a little work, the script can write the first, and last lines shown here, wrapped around the JSON block of raw data.

|  |
| --- |
| getdata.sh |
| * Connect to local SQL Database * Issue a SQL command: “select item, qty from inventory” * Write results to file src/dataset.js in JSON format (as above) |

This script can be executed at the beginning of our ./publish.sh script

**./getdata.sh**  
  
rm index.zip  
cd src  
zip –X –r ../index.zip \*  
cd ..  
aws lambda update-function-code --function-name **InventoryDashboard** --zip-file fileb://index.zip

#### The Schedule

Batch scripts can be configured by your local OS to run on a regular schedule. You can choose how often to refresh your Lambda function. If you refresh every five minutes, the skill data will never be more than a few minutes old.

|  |
| --- |
| Automation commands |
| * Mac/Linux : Crontab * Windows : Task Scheduler, execute **publish** batch every 5 minutes |

### Impact

You now have enabled your Alexa skill to make use of protected corporate data. The data is kept wholly within AWS Lambda, which simplifies the security risks, as opposed to setting up a new SQL database or data store which would expand the footprint of the solution and introduce more moving parts as data is access and transferred between services. Access to the Lambda code can be locked down so that developers may only publish updates, and the ASK service is the only entity which can execute the code.

Currently there are no documented charges for uploading new code to AWS Lambda, so publishing on a 5 minute or even 1 minute schedule should not add costs.

### Summary and Next Steps

This article has shown you how to setup a deploy script using the AWS CLI, so you can quickly and easily publish changes from your local IDE development environment into AWS Lambda.

Learn more about Alexa at <http://developer.amazon.com/ask> and more about Lambda at <https://aws.amazon.com/lambda>

-Rob @RobMcCauley