BACHELOR PAPER

Term paper submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Engineering at the University of Applied Sciences Technikum Wien - Degree Program Informatik/Computer Science

Controlling a Linux SBC based smart home using Amazon Alexa

By: Felix Bauer
Student Number: 1610257002

Supervisor 1: Dipl.-Ing. (FH) Arthur Michael Zaczek

Stockerau, 15 May 2018

FH University of Applied Sciences TECHNIKUM WIEN

Declaration of Authenticity

“As author and creator of this work to hand, I confirm with my signature knowledge of the relevant copyright regulations governed by higher education acts (see Urheberrechtsgesetz/ Austrian copyright law as amended as well as the Statute on Studies Act Provisions / Examination Regulations of the UAS Technikum Wien as amended).

I hereby declare that I completed the present work independently and that any ideas, whether written by others or by myself, have been fully sourced and referenced. I am aware of any consequences I may face on the part of the degree program director if there should be evidence of missing autonomy and independence or evidence of any intent to fraudulently achieve a pass mark for this work (see Statute on Studies Act Provisions / Examination Regulations of the UAS Technikum Wien as amended).

I further declare that up to this date I have not published the work to hand nor have I presented it to another examination board in the same or similar form. I affirm that the version submitted matches the version in the upload tool.”

Place, Date	Signature

Kurzfassung

Smart Speaker sind über die letzten Jahre zunehmend populär geworden, wobei Amazons Plattform „Alexa“ den mit Abstand höchsten Marktanteil besitzt. Eine typische Verwendung für ein solches System ist die Steuerung von Smart Home Geräten. Um ein eigenes solches Gerät für die Steuerung mit Alexa einzurichten sind mehrere Schritte notwendig. Zunächst muss ein Skill im Alexa Skills Kit und eine Handler-Funktion, zumeist eine AWS Lambda, programmiert werden. Abhängig vom gewählten Workflow und Interaktionsmodell ist es auch möglich diese Funktion privat zu hosten. Einfache Endgeräte können mittels UPnP kontrolliert werden, wodurch es nicht notwendig ist einen eigenen Skill zu entwickeln. Während sie dadurch leicht aufzusetzen sind, hat auch dieser Ansatz seine Schwachpunkte.

Schlagwörter: Amazon Alexa, Amazon Echo, Smart Home, AWS Lambda, Hausautomatisierung

Abstract

Smart speakers have become very popular over the recent years with Amazon’s Alexa platform having the highest market share by far. One popular use case of those digital assistants is controlling smart home appliances. Several steps must be taken to set up a personal smart home device. A skill in the Alexa Skills Kit must be developed and a handler function, usually an AWS Lambda function must be programmed. Depending on the chosen workflow and interaction model a handler function may be hosted privately. Simple appliances may even be controlled via UPnP and therefore do not require developing an Alexa skill. While this makes them very easy to set up, this approach has weaknesses too.

Keywords: Amazon Alexa, Amazon Echo, Smart Home, AWS Lambda, Home Automation

Table of Contents

1 Introduction

Over the recent years expenditure for Internet of Things (IoT) endpoints raised exponentially and is predicted to continue skyrocketing. [1] In October 2017 Amazon's smart home speaker platform "Alexa" had a marked share of 68% with a huge margin to Google's platform having only 24%. [2] Not just expenditures but also the number of skills available for Alexa is growing exponentially [3] and reached a number of over 25,000 skills in December 2017 [4].

1.1 Scope

This bachelor thesis covers the possibilities to control smart home appliances using the Amazon Alexa virtual assistant. It also covers the process of developing, installing and deploying those appliances for personal use. It does not cover all requirements to build a commercial product such as licensing, security and cost.

The following research question derives from this scope:

> What are the possibilities to control Linux based smart home appliances using Amazon Alexa and how can they be set up for personal use?

1.2 Document structure

Firstly, this document will show the more powerful way of controlling a Linux based system by developing an Alexa skill. This makes it possible to tailor the skill to someone's very personal needs. Both methods, hosting a HTTPS endpoint and defining an AWS handler function, are discussed in this thesis.

Afterwards the simpler but less powerful way of using an existing software environment to control a device is shown. As a concrete example the WeMo environment is used.

This document ends with a conclusion summarizing findings and giving advice about which method to choose depending on the personal needs and requirements.

2 Development of an Alexa Skill

When an Alexa-enabled device detects its wake word using on-device keyword spotting, recorded audio including a fraction of a second before the wake word is streamed to the cloud to be processed. [5]
Once the audio is streamed to the Amazon cloud the Alexa Voice Service (AVS) comes into play. AVS then tries to understand what the user said, compares it with sample utterances of skills the user has enabled and opens the best matching skill. [6] Opening and interacting with a skill means triggering an AWS Lambda or sending an HTTPS POST request to a customer webserver containing all parameters that might be needed as the body. The method chosen depends on the skills configuration. [7]
When using an AWS Lambda, it can then access other services such as DynamoDB, the S3 storage bucket, manage Redshift resources or use any features the programming language the Lambda is written in allows. [8]

2.1 Configuration and development of a skill

Using the Alexa Skills Kit (ASK) Developer Console a new skill can be created. After choosing one of the three pre-built models (flash briefing, smart home or video) or a custom model, the actual skill can be developed. [7]
The flash briefing skill API is meant for skills providing news from an external source such as an RSS feed or other news APIs. [9] The video skill API on the other hand is built to provide the customer a way to control different playback devices and video sources using a unified interaction model. [10]
Since both APIs do not provide the functionality to control a smart home device, the possibilities remain to use the smart home model or build a very custom skill. Both possibilities are discussed in the following chapters.

2.1.1 AWS Lambda

AWS Lambda is a way to execute code in highly available system without thinking about the server or infrastructure running it. AWS Lambda is billed on a pay per use basis.
While Lambda may be hosted in any of the regions AWS offers, only some regions are eligible for Lambdas used in conjunction with Alexa skills: Asia Pacific (Tokyo), EU (Ireland), US East (North Virginia) and US West (Oregon). [11] When trying to use a AWS Lambda hosted in another region for a skill, an error is thrown when trying to save. [7].
Lambda functions can be written in Java, Node.js, .NET Core or Python. While their setup is simple and no virtual servers are needed to run a Lambda function, it integrates seamlessly in the AWS environment. It can be triggered by many services such as S3 bucket events, Alexa, Simple Email Service, DynamoDB and more. It can also invoke and access many other AWS services such as other AWS Lambdas, Identity and Access management and Amazon Redshift. [8]
Despite personal preferences, performance and cost are major factors when deciding for a runtime for the Lambda function. Lambda pricing is independent of the runtime environment

but only depends on the number of requests as well as the memory in combination with compute time. [12]

2.1.1.1 Performance comparison

As Table 1 shows, runtime performance of different available runtimes is similar while the time a cold start takes depends heavily on the environment used as shown in Table 2. They also correlate with the allocated memory, likely because AWS Lambda allocates CPU depending on the allocated memory. [13]

Environment	*Average (ms)*	*Maximum (ms)*
Java	1.10	18.8
C# (.NET Core 2.0)	0.37	17.3
F# (.NET Core 2.0)	0.22	16.2
Go	1.10	18.5
Node.js 4	0.64	19.0
Node.js 6	0.48	18.5
Python 2.7	0.52	20.2
Python 3.6	0.88	20.1

Table 1: Performance comparison between runtime environments [14]

Environment	Average cold start time by memory size in milliseconds				
	128 MB	256MB	512MB	1024MB	3000 MB
Java	5 436.5	2 704.3	1 364.9	667.0	346.9
C# (.NET Core 1.0)	5 042.8	2 455.0	1 261.7	611.2	366.9
C# (.NET Core 2.0)	3 293.7	1 598.1	834.7	431.0	336.7
Go	8.7	9.6	0.9	1.3	0.8
Node.js 4	7.0	5.3	6.0	5.5	2.2
Node.js 6	7.5	7.5	5.2	4.5	1.7
Python 2.7	4.8	4.0	5.1	3.2	1.4
Python 3.6	4.8	4.5	4.6	3.6	3.0

Table 2: Cold start time comparison between runtime environments [15]

Cold start time is a major factor when choosing an environment for a smart home device considering that in peak times new instances of the Lambda need to be generated and therefore end users need to wait for the instance to be started. Depending on the appliances to be controlled a waiting time of several seconds may be unacceptable. Environments with similar, low cold start times are Go, Node.js and Python.

After deciding for a runtime, a new Lambda function can be created. To do so, a name and a role must be chosen. While the name is only for informational purpose, the role defines the permissions the Lambda will have. An already existing test role has the required permissions

to create a `LogGroup`, `LogStream` and put events to those resources by default. Depending on the Lambda's tasks, further permissions may be required and a new role should be created. [8], [16]

After creating the Lambda, triggers and resources can be defined. For a Lambda to be invoked by an Alexa skill, one of the Alexa triggers is required. If the skill uses the Smart Home API, the Alexa Smart Home trigger is fired, otherwise the more generic Alexa Skills Kit trigger is set off. In either case the Application ID of the skill triggering the Lambda must be entered. [8]

In a Node.js environment, AWS Lambda handler functions always have three parameters – an event, a context and a callback.

The event parameter is the one containing the actual argument passed to the function. In case of a custom workflow this contains information about the current session, context information about the Echo (if it has a display or audio is played), and the request itself. For the smart home model this parameter represents the directive as discussed in chapter 2.1.2.2 Payload.

The context contains general information about the Lambda execution such as the available memory, the remaining time for the function and the `LogGroup` and `LogStream` name.

The callback parameters may define a callback function to be invoked when the Lambda function is finished. This parameter is unused when using Lambdas in conjunction with a custom skill. Its value is always `undefined`. When used in a smart home workflow, this parameter must be used to send a response to the ASK as seen in line number 49 in the file discovery.js in Appendix A.

2.1.1.2 Lambda API for logging and debugging

The way logging work varies between languages. In case of a JavaScript skill, any calls to `console.log` or the other console functions Node.js provides are logged. For C# and Java a logger object is provided via the context argument of the function.

Those logged contents are sent to AWS CloudWatch where they are stored according to the settings. By default, log messages got no expiration date and are stored forever. It is worth noting that a `LogGroup` is created for every Lambda. A `LogStream` within the `LogGroup` gets created whenever log entries from new sources get logged. [17] In case of AWS Lambda, this means that every time a new Lambda container is created, a new `LogStream` is created too. Additionally to the manually logged messages, some events are logged automatically. Those events include the starting and ending of an AWS Lambda and uncaught exceptions thrown within the Lambda. Due to this automatic logging, the role running the AWS Lambda needs permission to create a `LogGroup`, a `LogStream` as well as to write messages to the log to work even when no manual logging is done.

2.1.1.3 Backend server

Since Lambdas are stateless, an external service is needed to persist information about users, such as the devices a user has access to. When using account linking as discussed in 2.1.2.2 Payload the generated OAuth 2.0 token can be used as an identifier for the user. This

token should be used as a key for any stored information. In a smart home skill this information might be a set of devices associated with the user. Since the Lambda is running in an AWS environment, it might be an easy solution, to store information in one of the AWS storage solutions such as RDS or DynamoDB. Whenever a Lambda is triggered, the OAuth 2.0 token is passed to the handler function and can then be used to retrieve previously stored information.

2.1.2 Smart home model

When deciding for the smart home model, there are three example utterances provided that show its capabilities:

- Alexa, turn on the kitchen lights
- Alexa, turn off the sprinkler
- Alexa, increase thermostat by 2 degrees

Those examples just show some possibilities for the smart home model. The complete list of functionality the smart home model provides can be found in its documentation. [7], [18]
The Amazon Resource Name (ARN) of an AWS Lambda must be supplied as the endpoint. An ARN for a default region is required. Three ARNs for different regions (North America, Far East and Europe, India) may be supplied to improve the performance for customers in those regions. [7] Those three additional Lambdas, as well as the default Lambda follow the regional restrictions discussed in chapter 2.1.1.

2.1.2.1 Account linking

Account linking is a requirement for skills using the Smart Home Skill API or Video Skill API. It is the process of connecting end user identities with identities of other systems. Technically this is done using the OAuth 2.0 Authorization Framework. [19] Amazon developers may choose to use the service “Login with Amazon” as the authorization server. To do so, a new security profile for the skill must be created. The created client ID and secret need to be entered in the ASK Developer Console. Also `https://api.amazon.com/auth/o2/token` must be entered as the access token URI and `https://www.amazon.com/ap/oa/` with a `redirect_url` query parameter set as the authorization URI. The available redirect URLs can be found on the bottom of the account linking page. [7], [20]
Since Alexa-enabled devices provide no access to local infrastructure, account linking and the linking of bought devices with an account of the producer is required for the provider to link devices to users.

2.1.2.2 Payload

There are currently two payload versions available, v2 and v3. Because v2 was obsoleted with the release of v3, only the later version will be covered within this thesis.

There is a well-defined workflow and API for smart home skills and their Lambdas. When activating a smart home skill, the end user is forced to log in via an external OAuth 2.0 Authentication provider. Therefore, the skill cannot be activated via voice commands but

needs to be activated via the Alexa companion app. After authentication an OAuth 2.0 token is created that can be used to identify the user among requests and is typically used to persist user information such as devices and device settings in a database.
When the Lambda is triggered, it receives a directive object as an argument. It contains all necessary information in nested objects. The Lambda answers with an event, having the same basic structure as a directive. Directives as well as events contain three major sections: `header`, `endpoint` and `payload`. Events may contain an additional section called `context`.

`namespace`	Logically groups commands by functions	`Alexa.PowerController` `Alexa.Discovery`
`name`	Can be interpreted as the command to execute or the type of response	`TurnOn` `Discover`
`messageId`	A unique ID for the message. Should not be used for anything but logging. Recommended is to use a version 4 UUID for this purpose.	`xxxxxxxx-xxxx-4xxx-xxxx-xxxxxxxxxxxx`
`payloadVersion`	Version of the payload, either “2” or “3”	`3 for current versions`

Table 3: Header fields for the smart home workflow [18]

Typically, the Lambda will start by switching the namespace and name to find out what function to invoke as seen in index.js in Appendix A. This is done because the payload’s format varies depending on the name. Due to this and the cold start time shown in Table 2, mainly dynamically typed languages are used for ASK development. This is also shown on the Alexa GitHub page where currently 30 of their projects are written in dynamically typed languages (JavaScript and Python) and only six in statically typed ones (Java and C#) as shown in Figure 1**Fehler! Verweisquelle konnte nicht gefunden werden.**. [21]

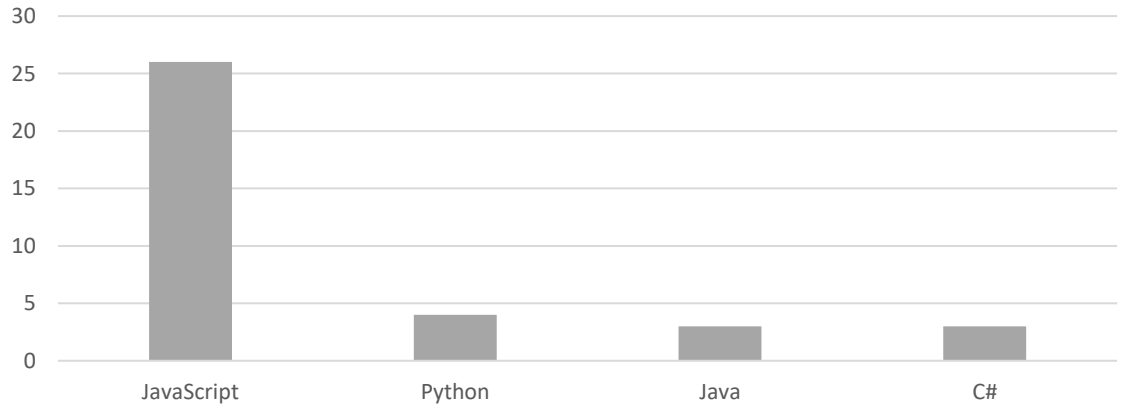

Figure 1: Usage of programming languages of the Alexa GitHub user [21]

2.1.2.3 Workflow

After authentication the user will prompt Alexa to find newly connected devices, typically using the utterances “find devices”, “look for new smart home devices” or similar. This triggers the Lambda with a namespace of `Alexa.Discovery` and a name of `Discover`. The ASK waits for up to eight seconds for responses from all smart home skills before responding newly found devices to the user.

During that time the Lambda will contact a backend environment and look for all devices a specific user has registered. To report all connected devices to the ASK, an array of endpoints is responded. Every endpoint has a list of capabilities, each containing an interface describing its type. Examples for interfaces are `Alexa.PowerController` to turn devices on and off or `Alexa.BrightnessController` to regulate the brightness of a light. The endpoint has an array of `displayCategories` defining how this endpoint is reported to this user. If an endpoint e.g. has a `displayCategory` of `LIGHT` and capabilities with interfaces `Alexa.PowerController` and `Alexa.BrightnessController`, a light that can be turned on or off and can have its brightness regulated will be reported to the user. [18], [22]

For further messages the earlier discussed interfaces are used as a namespace and each of those interfaces defines multiple names that may be used. For example, a directive with a namespace of `Alexa.BrightnessController` and a name of `SetBrightness` might be sent. The payload will then contain additional information, in this case the brightness to be set, as this example from the ASK documentation shows: [18]

```
{
 "directive": {
  "header": {
   "namespace": "Alexa.BrightnessController",
   "name": "SetBrightness",
   "payloadVersion": "3",
   "messageId": "1bd5d003-31b9-476f-ad03-71d471922820",
   "correlationToken": "dFMb0z+PgpgdDmluhJ1LddFvSqZ/jCc8ptlAKulUj90jSqg=="
  },
  "endpoint": {
   "scope": {
    "type": "BearerToken",
    "token": "access-token-from-skill"
   },
   "endpointId": "appliance-001",
   "cookie": {
   }
  },
  "payload": {
   "brightness": 42
  }
 }
}
```

An example C# .NET Core implementation of a skill handling smart home device discovery can be found on GitHub in the repository `felixhacks/AlexaSkillsKitLambda`. Since the workflow is very dynamic and response structure varies depending on the request action, a dynamically typed language such as Python or JavaScript might be better suited for the skill.

2.1.2.4 Blueprints

An example implementation of the smart home workflow can be found in the AWS Lambda blueprint `alexa-smart-home-skill-adapter`. Although this skill uses the now deprecated payload v2, it shows the workflow and provides an entry point for developing a new smart home skill. When adapting to the new directive and event format, changing namespaces and names to match with the new payload v3, this skill can serve as a skeleton for an own skill.

2.1.3 Custom model

After skill creation of a custom skill, a skill builder checklist with four steps is presented. The first step is entering the invocation name. This is how users will later call the skill when interacting with it. For a skill to be available in multiple languages, an invocation name must be supplied for every language separately. Currently there are eight languages available. Five of them are English accents and the other ones are German, Japanese and French. [7]

Step two on the checklist is defining the interaction model consisting of intents, samples and slots. An intent defines a type of action a user can make. Each action can have multiple parameters, those are called slots. [7]

Multiple slot types are provided by Amazon. Already available are for example a number or a date slot type. A more complex slot type useable for more unpredictable input is the search query type. It can be used to capture phrases without restricting to any specific input format or type. The last category of slot types available are list types. List types define a list of possible values that are considered valid. An example slot type provided by Amazon in English (US) is the `AMAZON.AT_CITY` type, a list with over 5 000 cities as possible values for a slot. The full list of available slot types can be found in the ASK documentation and varies by language. [23]

There are three built-in intents enabled by default. Those are called `AMAZON.CancelIntent`, `AMAZON.HelpIntent` and `AMAZON.StopIntent`. There are currently 144 built-in intents in nine categories [7] that may be used to provide common functionality without needing to manually enter sample utterances. [24]

Additionally, custom intents can be created. This is one of the major difference between a smart home skill and a custom skill. While the developer can not change the interaction model for a smart home skill, the developer is forced to construct an own interaction model for a custom skill.

Custom intents need at least one sample utterance. Those utterances may contain multiple slots. Slots are put in curved braces in the sample utterances and need to be assigned to a slot type. If the existing slot types do not meet the skill's requirements, a custom slot type can be created. For a custom slot type a list of possible values must be supplied. Additionally, a list of synonyms may be supplied for every value. [7]

For custom intents an intent confirmation can be enabled. This forces Alexa to prompt the user to confirm this intent before it is triggered. [7]
After finishing the interaction model and saving it, the interaction model must be built as the third step of the checklist. This process validates the model and starts a build process. After the process is complete, a success message is displayed in the developer console. This process might take several minutes. [25]
Independent of the interaction model, an endpoint must be configured to finish step four of the checklist. This can either be an AWS Lambda, which is the recommended option, or an HTTPS endpoint. [7]
When choosing an AWS Lambda as the endpoint, the same restrictions and requirements are true as for smart home skills as described in 2.1.1 AWS Lambda.
When developing a Lambda, it may make use of the npm package `ask-sdk` provided by Amazon for the Alexa Skills Kit when using a Node.js runtime. This package was released on 18-04-2018 and obsoleted the earlier package `alexa-sdk`. The source of this package is available on GitHub. [26] An official Java 8 version of this library is available via Maven or Gradle using the GroupId `com.amazon.alexa`. [27]
The `ask-sdk` package provides helper classes to define request handlers and build responses. Basic usage examples can be found in Appendix B. Starting in line 114 a `SkillBuilder` is used to build a custom skill. This is done by adding request handlers to it. Additionally, request interceptors and error handlers could be added using their respective functions in the fluent API. The `lambda` function in line 122 then creates a function to be exported as the handler function for the Lambda. The function `addRequestHandlers` receives `varargs` of type `RequestHandler`. `RequestHandler` is an interface defining two functions `canHandle` and `handle`. Exemplary implementations of the interface can be found in lines 6 ff and 86 ff. The only dependencies for this node module are a requester omitted for the sake of simplicity as well as the `ask-sdk-core` submodule. Further submodules of the `ask-sdk` can be added if needed.

2.2 Connecting a Linux system

After invoking a skill, the skill's handler, whether it is an AWS Lambda or an HTTPS endpoint, must contact the actual target device to notify it about the action to take. The way this device gets notified may vary between the AWS Lambda endpoint and HTTPS endpoint.

2.2.1 Using AWS Lambda

To connect the Lambda to the actual Linux based target device, all features of the chosen language or runtime can be used.

Node.js

In case of the common Node.js environment this is most notably the module `https` for simple HTTPS calls and REST APIs. The following code snippet shows the basic usage of this module to send HTTPS requests to a given host.

```
const request = http.request(options, responseHandler);
request.write(data);
request.end();
```

The `options` object must be either of type `URL`, a string that can be converted to a `URL` or an ordinary object. Typically, this object will at least contain information about the remote host and the used HTTP method.
Npm modules are available for more complex communication protocols such as SOAP.

.NET Core 2.0

When using C# for the Lambda, there are several possibilities to connect the Linux device. It can be done using `HttpClient` if the target device has a REST endpoint or one of many available NuGet packages if it supports SOAP. If the target device accepts a TCP connection to transmit raw data e.g. over a proprietary protocol, `TcpClient` or `Socket` may be used.
No matter what runtime is used, due to the loose coupling of the systems, it does not matter if the endpoint is based on Linux or any other system capable of the used communication protocol.
Many of the common single board computers (SBC) offer simple libraries to control general purpose input and output (GPIO) pins. For the Raspberry Pi those are e.g. `WiringPi`, `rpi-gpio` and `RPi.GPIO`. For many other SBCs clones of those libraries exist. An example is the `RPi.GPIO` library that was modified to be compatible with the Banana Pro and Banana Pi and is distributed as `RPi.GPIO_BP`.
To control the GPIO a wrapper must be provided for the library, providing an endpoint accessible from the AWS Lambda service. With security out of the mind this may be as simple as setting up a Node.js server with a single endpoint accepting two parameters `pin` and `value` which is in turn calling the underlying library to turn a pin high or low. This example uses the `gpio` package, available via npm.

```
01 var express = require('express');
02 var app = express();
03 var gpio = require('gpio');
04
05 app.use('/:pin/:value', function(req, res) {
06     var currentGpio = gpio.export(req.params.pin, {
07         direction: "out",
08         ready: function() {
09             currentGpio.set(req.params.value);
10         }
11     });
12     res.send('ok');
13 });
14 module.exports = app;
```

In line 5 the path is defined and two parameters are declared in it. In line 6 the pin parameter is used to access one of the Raspberry Pi's GPIO pins to then, in line 9, set the value accordingly. Since setting up a GPIO pin may take some time, the value is set using the callback pattern very typical for Node.js. In line 12 a response is sent to end the HTTP request successfully. In line 14 the Node.js Express app is exported so that it listens on port 3000, the default port for the Express framework.

This endpoint being contacted by the Lambda handler may serve as a central endpoint, distributing requests within the LAN so that only a single device must be accessible from outside the LAN. Doing so reduces the number of devices that have either open connections to a central server or are directly accessible over the public IP address. When only a single device is connected to the central server, the number of open connections on the central server may be cut down significantly and therefore reducing the cost. Also, only a single endpoint must be patched when choosing this method and only one endpoint can serve as a potential security leak and must be patched. Those are also the reasons why many producers of smart home devices require hubs or bridges for their devices to function.

This endpoint should then be called from the Lambda to control the GPIO pins. To establish a connection, the SBC must be accessible from the Internet and be reachable under a known IP address or hostname. To do so, one may choose to get a static IP address, usually sold with business Internet contracts only. Those contracts are usually offered with higher upload bandwidth and are more expensive than customer contracts. Cheaper options are using an external service such as ngrok or using a dynamic Domain Name Service (DDNS) solution.

ngrok

ngrok is a service providing public URLs for computers behind a network address translation (NAT) service or firewall. To do so, the computer actively opens a tunnel to https://ngrok.com where a subdomain is created for each tunnel. Since firewalls do not block outgoing traffic to port 443, this connection will be successful even behind strict firewalls. [28]

The created tunnel can forward to any port on the local machine, so even HTTP endpoints not running on port 80 may be reached using ports 80 or 443 for a secured HTTPS connection via the created URL. [28]

A free plan includes a single ngrok process with four tunnels, so up to four local ports of one single computer may be published using this method. When needing more than one computer running ngrok or needing more than four tunnels on a computer, paid plans may be used. [28]

To use ngrok, the compressed binary must be downloaded from ngrok.com and unzipped. Next, the authentication token of the used account must be entered to connect the local instance to the account. This is done by running `./ngrok authtoken <token>` from the directory containing the binary. The authentication token can be retrieved by logging in on the website using a created account or by using GitHub or Google as authentication services. After configuring the authentication token, a tunnel can be started by executing `./ngrok <protocol> <port>` where protocol can be either `http` or `tcp` and port is the local port to which the service is listening. [28]

Static IP

Static IP addresses are usually sold with business or enterprise Internet contracts and mean that the Internet service provider (ISP) assures the customer to keep his IP address over the duration of the contract. Customer contracts usually have dynamic IP addresses and fetch those whenever they need one but are not guaranteed to keep IP addresses over any duration. This way IP addresses may be saved and ISPs will need less IP addresses than they have customers since not all customers are online at the same time and therefore not all customers need an IP address at the same time.

When having a static IP address, it can be used as an HTTPS endpoint for the skill. A requirement for this is, to forward a chosen port of the public IP address to a chosen port of the webserver handling the requests from the skill. This port forwarding process is needed because the webserver is behind a NAT and therefore not directly reachable over the router's public IP address.

Dynamic DNS

The way DDNS works is by frequently updating a DNS record with the current IP address. By doing so, a static domain may be provided even for dynamic IP addresses. There may be small downtimes whenever the IP address changes, until the next update occurs and updates the DNS entry accordingly.

When choosing DDNS with a dynamic IP address as the domain for the Alexa skill downtimes of a few minutes, depending on the update interval of the DDNS service, must be expected.

The static domain must should be entered in the HTTPS endpoint field of the Alexa skill but to make the webserver accessible from the ASK, port forwarding must be configured for the same reasons as discussed in previous chapter Static IP.

2.2.2 HTTPS Endpoint

When deciding not to use AWS Lambda, which is only possible for custom skills but not for smart home skills, a defined HTTPS endpoint is called instead of the Lambda function. As a request body for the POST request it contains the same object as the Lambda function would otherwise receive as an argument.

This way the target smart home device may be used as an HTTPS endpoint and no further redirects of commands are needed. This might require a faster endpoint capable of running a webserver and may lead to increased cost. If multiple devices should be controlled with a single skill, a central sever is needed redirecting the requests accordingly.

To make the HTTPS endpoint accessible from outside the LAN, any of the methods discussed in chapter 2.2.1 Using AWS Lambda may be used. Also, the webserver itself may have a similar core but needs to parse and handle requests from the ASK correctly.

If not using the target smart home device as the HTTPS endpoint but a server outside the LAN of the target device, the target devices must be notified about actions to take. To do so, any methods from the previous chapter *2.2.1 Using AWS Lambda* can be used.

2.3 Testability

One of the most important things to build skills with high quality is testing. There are multiple aspects and methods of testing a skill. [29]

Unit testing

Unit testing is the process of testing the smallest parts of the software (functions) separated from each other by mocking dependencies and checking the result of the functions to match the expected results.

In the particular context of writing tests for an Alexa skill this means writing test cases verifying every possible intent from ASK and every major unit of code inside the software. [29]

AWS Lambda does not provide a native way to unit test the written software. There exist wrappers to test the code withing AWS Lambda but using an external service to automate unit tests such as Travis CI or Jenkins is recommended by the head of Amazon Alexa Code labs. [29]

Two of the most popular unit testing frameworks for Node.js are Mocha and Chai. [29]

Manual testing

Within the ASK Developer Console tests may be enabled for a skill. This way a skill can be manually tested using either voice over a microphone or keyboard input with the Alexa Test Simulator. This testing functionality also shows the JSON objects sent to and from the endpoint and can show the displays of an Echo Show or Echo Spot as well as the log output of the Echo device. [7]

Using manual testing not only the functionality of the endpoint can be tested but also the interaction model and user experience. This is the reason why this testing method is so important. There is almost always potential for improvement from the initial voice design. [29]

Field testing

To test a skill with actual users before publishing it, a set of users for a beta program may be entered. Metrics are tracked for any interactions of the beta users (and production users) with the skill to better understand how actual users will interact with the skill and what utterances they will use. This type of deployment is available even before the skill is submitted to Amazon for review. [7]

The skill can be tested on Alexa-enabled devices registered on the developer’s account as soon as the interaction model is built in the ASK Developer Console. This enables the developer to test a skill in a field testing like environment without involving other users. It also makes it possible to develop skills for oneself without going through any validation process and without entering any information that is needed for a skill to support the beta testing procedure.

2.4 Launch

To launch the skill and make it publicly available, several requirements must be met.
Any fields in the store preview must be filled for any language the skill will support. This includes a description, example phrases, icons in different sizes, a category, keywords as well as URLs for a Privacy Policy and the Terms of Use. [7]
Next off, legal information about the skill must be entered such as whether the skill allows users to spend money or the skill collects personal information. Also testing instructions for the Amazon team testing the skill can be provided if not obvious. This should include hardware requirements and any other information needed by the testing team. [7]
After filling all fields, the skill can be submitted to Amazon for certification review. The developer will be notified as soon as the verification process is complete. The skill must be recertified whenever it gets updated. [30]

3 Connecting to an existing software environment

A very basic approach of connecting a smart home appliance is to emulate an existing commercial smart home device and hence controlling it the same one as the original.

3.1 Emulating a Belkin WeMo device

WeMo is one of the three brands of Belkin International, the other two being Belkin and Linksys. WeMo is a brand for home automation devices. There exist different types of WeMo devices such as dimmers, switches and cameras. [31]

3.1.1 WeMo

What is special about WeMo is its open source GNU General Public License (GPL) license. [32] This gives developers a chance to integrate WeMo devices in their applications. It also introduces the possibility to emulate WeMo. There exist emulations of WeMo devices called Fauxmo (pronounced \ˈfō-mō\) for Microcontrollers, Linux and Windows. The name Fauxmo comes from the French word “Faux” meaning false and “Mo” from “WeMo”. [33]

Initially this project was started by makermusings on GitHub and was published under an MIT license. In 2015 n8henrie cloned the repository to update and extend the code. While the last commit to the initial repository was in August 2015, its clone’s latest commit was in May 2018. [33], [34]

WeMo devices are discovered using the Universal Plug and Play (UPnP) discovery protocol Simple Service Discovery Protocol (SSDP). [35] It responds with a service description as specified by the UPnP Device Architecture. [36] In case of a simple switch this service might propagate only one service descriptor. In case of WeMo this is the service of type `urn:Belkin:service:basicevent:1` which defines an action `SetBinaryState` enabling to turn the switch on and off. Those UPnP services are controlled by endpoints using a SOAP based control messages. [36]

The reason this works is, because Echo devices support this type of communication with smart home devices by default hence no skill is required. While working without a skill can be a positive argument for simplicity it can limit the capabilities as it will not contact an AWS Lambda at any time.

3.1.2 Using Fauxmo library

The initial Fauxmo library consists of a single file and requires Python 2.7 to run. By default, it populates two devices called “office lights” and “kitchen lights”. [34] A program using the Fauxmo library tinkered for the Raspberry Pi is `echo-pi` by kanesurendra which is also available under MIT license. By default, some of the Raspberry Pi’s GPIO ports are controllable as switches. [37]

This is done by defining an array of triggers (lines 43 ff) and their corresponding GPIO pins (line 22) as well as TCP ports for the listener. In line 81 the Fauxmo library is then initialized for each of the triggers with a device handler as the last argument handling requests to turn

on or off the GPIO pins. This `device_handler` class internally uses the `RPi.GPIO` library to control the pins.

```
09 import fauxmo

22 gpio_ports = {'gpio1':1, [...] ,'gpio26':26}

43 TRIGGERS = {"gpio15":50015,
...
54    "gpio26":50026}

70 if __name__ == "__main__":
71    # Startup the fauxmo server
72    fauxmo.DEBUG = True
73    p = fauxmo.poller()
74    u = fauxmo.upnp_broadcast_responder()
75    u.init_socket()
76    p.add(u)
77
78    # Register the device callback as a fauxmo handler
79    d = device_handler()
80    for trig, port in d.TRIGGERS.items():
81        fauxmo.fauxmo(trig, u, p, None, port, d)
```

When modifying the code to use triggers 16-18, only those pins will be discovered using SSDP. The pins can even be renamed when renamed both in the trigger definition and the GPIO ports definition. After renaming them e.g. to “lamp one” to “lamp three” and telling Alexa to discover devices, those three endpoints can be found in the Alexa companion app and controlled via Alexa. In the app the endpoints can be renamed and the type can be changed to either plug, which is the default, or light.

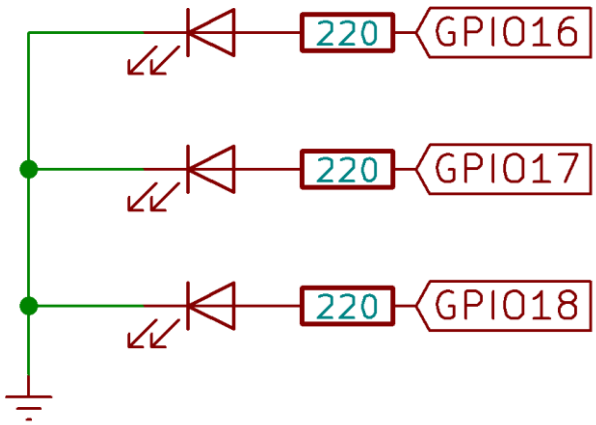

Figure 2: Example wiring to control LEDs with the Raspberry Pi

Figure 2 shows how LEDs should be wired to the GPIO pins to make them light when they are turned high. The current limiting resistors with a value of 220Ω to limit the current to 5 mA for a 2.2V LED. This prevents the LEDs from drawing too much current and destroying the LED itself or the Raspberry Pi.
When wiring the LEDs to the Raspberry Pi it is important to check which physical pin to use because physical pin number do not match with GPIO numbers as defined by Broadcom.

3.2 Application limitations

While the method of emulating other devices could be used for any kind of device supported by Amazon Echo devices, the Fauxmo library only emulates switches. New versions of the library support plugins and other device types could be added this way. [33]
Although Belkin offers a variety of smart home devices, including switches, light bulbs, heaters and coffee makers, there is no official statement which devices function with Echo devices using UPnP without an AWS Lambda invocation. [38]

4 Conclusion

Concluding it can be said, that there are three approaches available to control a single board computer using the Alexa voice service. Following paragraphs sum up the findings on differences between those approaches. Table 4 also provides a quick overview of those findings.

	Smart home skill	*Custom skill*	*Fauxmo*
Device support	Medium	High	Low
Setup effort	High	High	Low
Private hosing possible	No	Yes	N/A
Account linking	Required	Possible	Not required
Connection from AWS	Required	Depends	Not required

Table 4: Comparison of different approaches

Two similar and, compared to the other one, complex approaches are to develop an own skill using the smart home model or a custom model. Both require developing an AWS Lambda whereas this can be substituted by setting up and developing a webserver for the custom model. Whilst being the more complex variants, they offer a wide variety of use cases and support interaction with external services such as databases to persist and retrieve information.

The simpler variant is using the Fauxmo library. It is limited to switches and toggleable lights. When programming a plugin for other device types those may be used as well but still limits the possibilities to device types Amazon Echo devices support via the UPnP protocol.

When deciding between the smart home model and the custom model it's not only about developing preferences but also about usage. Smart home devices give the user the possibility to group devices and control those groups. When using the custom model this convenience function is not provided. On the other hand it is also a question about the type of device being controlled. The smart home model only works for specific device types. When inventing and developing a new type of device that cannot be represented using the available types, a custom model must be developed.

Another factor that might be considered when choosing between the skill models is the possibility to host a custom skill on a private server whilst the backend for a smart home skill must be hosted as a Lambda in the AWS cloud. When already having a webserver setup running, this might reduce cost and server management effort.

One should also keep in mind that account linking is a requirement for smart home skills. Account linking requires an OAuth 2.0 authentication provider setup, although Amazon can be used as a provider. This is especially true when developing the skill only for personal use where setting up a new authentication infrastructure would be too much effort.

When choosing to develop a skill, being it a smart home or a custom model, still the connection from the Amazon environment to the local target device is missing and must not be forgotten. When using a Lambda, either because the smart home model requires so or

because a custom model with an AWS Lambda is chosen, a connection from the AWS cloud to the device can be established using any possibilities the runtime of the Lambda provides. When using a custom model and deciding to host the handler function as a HTTPS server on the endpoint itself this connection is not required, although controlling different devices using a single HTTPS endpoint does require a connection between those endpoints.

Bibliography

[1] ‘Global IoT spending by category 2014-2020 | Statistic’, *Statista*. [Online]. Available: https://www.statista.com/statistics/485252/iot-endpoint-spending-by-category-worldwide/. [Accessed: 07-Apr-2018].

[2] T. Haselton, ‘Amazon Echo market share tops Google Home’, 12-Oct-2017. [Online]. Available: https://www.cnbc.com/2017/10/12/amazon-echo-market-share-tops-google-home.html. [Accessed: 08-Apr-2018].

[3] ‘Infographic: Amazon’s Alexa Is a Fast Learner’, *Statista Infographics*. [Online]. Available: https://www.statista.com/chart/8304/alexa-skills/. [Accessed: 07-Apr-2018].

[4] ‘Alexa skills top 25,000 in the U.S. as new launches slow’, *TechCrunch*, 15-Dec-2017. .

[5] ‘Amazon.com Help: Alexa and Alexa Device FAQs’. [Online]. Available: https://www.amazon.com/gp/help/customer/display.html?nodeId=201602230. [Accessed: 22-Apr-2018].

[6] ‘Rapidly Create Your Alexa Skill Backend with AWS CloudFormation : Alexa Blogs’. [Online]. Available: https://developer.amazon.com/de/blogs/post/Tx27NAUCY0KQ34D/Rapidly-Create-Your-Alexa-Skill-Backend-with-AWS-CloudFormation. [Accessed: 07-Apr-2018].

[7] *Alexa Skills Kit*. Amazon, 2018.

[8] ‘AWS Lambda – Serverless Compute - Amazon Web Services’, *Amazon Web Services, Inc.* [Online]. Available: https://aws.amazon.com/lambda/. [Accessed: 23-Apr-2018].

[9] ‘Understand the Flash Briefing Skill API | Flash Briefing’. [Online]. Available: https://developer.amazon.com/docs/flashbriefing/understand-the-flash-briefing-skill-api.html. [Accessed: 03-May-2018].

[10] ‘Understand the Video Skill API | Video Skills’. [Online]. Available: https://developer.amazon.com/docs/video/understand-the-video-skill-api.html. [Accessed: 03-May-2018].

[11] ‘Host a Custom Skill as an AWS Lambda Function | Custom Skills’. [Online]. Available: https://developer.amazon.com/docs/custom-skills/host-a-custom-skill-as-an-aws-lambda-function.html. [Accessed: 23-Apr-2018].

[12] ‘AWS Lambda – Pricing’, *Amazon Web Services, Inc.* [Online]. Available: https://aws.amazon.com/lambda/pricing/. [Accessed: 24-Apr-2018].

[13] ‘aws lambda - compare coldstart time with different languages, memory and code sizes’, *theburningmonk.com*, 13-Jun-2017. .

[14] Y. Z. Lin, ‘Comparing AWS Lambda performance of Node.js, Python, Java, C# and Go’, *A Cloud Guru*, 08-Mar-2018. [Online]. Available: https://read.acloud.guru/comparing-aws-lambda-performance-of-node-js-python-java-c-and-go-29c1163c2581. [Accessed: 24-Apr-2018].

[15] D. Goyal, *lambda-coldstart-runtime-vs-memory*. 2018.

[16] ‘IAM Management Console’. [Online]. Available: https://console.aws.amazon.com/iam/home#/home. [Accessed: 24-Apr-2018].

[17] ‘Working with Log Groups and Log Streams - Amazon CloudWatch Logs’. [Online]. Available: https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/Working-with-log-groups-and-streams.html. [Accessed: 01-May-2018].

[18] ‘Smart Home Skill API Message Reference | Alexa Smart Home’. [Online]. Available: https://developer.amazon.com/docs/smarthome/smart-home-skill-api-message-reference.html. [Accessed: 29-Apr-2018].

[19] ‘Link an Alexa User with a User in Your System | Custom Skills’. [Online]. Available: https://developer.amazon.com/docs/custom-skills/link-an-alexa-user-with-a-user-in-your-system.html. [Accessed: 23-Apr-2018].

[20] ‘Amazon Apps & Services Developer Portal’. [Online]. Available: https://developer.amazon.com/lwa/sp/overview.html. [Accessed: 23-Apr-2018].
[21] ‘Alexa’, *GitHub*. [Online]. Available: https://github.com/alexa. [Accessed: 01-May-2018].
[22] ‘Alexa.Discovery | Alexa Device APIs’. [Online]. Available: https://developer.amazon.com/de/docs/device-apis/alexa-discovery.html. [Accessed: 01-May-2018].
[23] ‘Slot Type Reference | Custom Skills’. [Online]. Available: https://developer.amazon.com/docs/custom-skills/slot-type-reference.html. [Accessed: 23-Apr-2018].
[24] ‘Standard Built-in Intents | Custom Skills’. [Online]. Available: https://developer.amazon.com/docs/custom-skills/standard-built-in-intents.html. [Accessed: 23-Apr-2018].
[25] ‘Interaction Model Schema (Skill Management API) | SMAPI’. [Online]. Available: https://developer.amazon.com/docs/smapi/interaction-model-schema.html. [Accessed: 23-Apr-2018].
[26] *alexa-skills-kit-sdk-for-nodejs: The Alexa Skills Kit SDK for Node.js helps you get a skill up and running quickly, letting you focus on skill logic instead of boilerplate code*. Alexa, 2018.
[27] *alexa-skills-kit-sdk-for-java: SDK and example code for building voice-enabled skills for the Amazon Echo*. Alexa, 2018.
[28] ‘ngrok - secure introspectable tunnels to localhost’. [Online]. Available: https://ngrok.com/. [Accessed: 07-Apr-2018].
[29] ‘Building Engaging Alexa Skills: Why Testing and Automation Matter : Alexa Blogs’. [Online]. Available: https://developer.amazon.com/de/blogs/alexa/post/e2f3d18c-13ca-4796-bc83-e8a196f20e57/building-engaging-alexa-skills-why-testing-and-automation-matter. [Accessed: 07-Apr-2018].
[30] ‘Launch Your Skill | Developer Console’. [Online]. Available: https://developer.amazon.com/docs/devconsole/launch-your-skill.html. [Accessed: 06-May-2018].
[31] ‘Wemo - Home Automation’, *Belkin*. [Online]. Available: http://www.belkin.com/us/Products/home-automation/c/wemo-home-automation/. [Accessed: 14-Apr-2018].
[32] ‘Open Source Code Center’, *Belkin*. [Online]. Available: http://www.belkin.com/us/support-article?articleNum=51238&_ga=2.83856022.186610628.1523106310-1973980773.1523106310. [Accessed: 07-Apr-2018].
[33] N. Henrie, *fauxmo: Emulated Belkin WeMo devices that work with the Amazon Echo*. 2018.
[34] makermusings, *fauxmo: Emulated Belkin WeMo devices that work with the Amazon Echo*. 2018.
[35] *F7C029 (Insight) source code*. Belkin International, 2015.
[36] ‘UPnP Device Architecture 2.0’. UPnP Forum, 20-Feb-2015.
[37] S. Kane, *echo-pi: Controlling devices with Amazon Echo*. 2018.
[38] ‘WEMO Support - WEMO That’. [Online]. Available: http://www.wemo.com/support/. [Accessed: 08-Apr-2018].

List of Figures

List of Tables

List of Abbreviations

ARN	Amazon Resource Name
ASK	Alexa Skills Kit
AVS	Alexa Voice Service
AWS	Amazon Web Services
DDNS	Dynamic Domain Name System
GPIO	General purpose input/output
GPL	GNU General Public License
IoT	Internet of Things
ISP	Internet service provider
NAT	Network address translation
SBC	Single board Computer
SSDP	Simple Service Discovery Protocol
UPnP	Universal Plug and Play

A: Source code of the smart home Lambda

index.js

```
001 const common = require('./common');
002 const discovery = require('./discovery');
003 const control = require('./control');
004
005 exports.handler = (request, context, callback, d, e) => {
006     common.log("DEBUG", JSON.stringify(request));
007
008     switch (request.directive.header.namespace) {
009         case 'Alexa.Discovery':
010             discovery.handleDiscovery(request, callback);
011             break;
012
013         case 'Alexa.PowerController':
014             control.handleControl(request, callback);
015             break;
016
017         default: {
018             const ns = request.directive.header.namespace;
019             const errorMessage = 'No supported namespace: ${ns}';
020             common.log('ERROR', errorMessage);
021             callback(new Error(errorMessage));
022         }
023     }
024 };
```

discovery.js

```
001 const USER_DEVICES = {
002     endpoints: [
003         {
004             endpointId: 'unique-id-for-bulb',
005             manufacturerName: 'SmartHome Product Company',
006             friendlyName: 'Smart light',
007             description: 'Smart light bulb from Product Company',
008             displayCategories: [
009                 "LIGHT"
010             ],
011             cookie: {},
012             capabilities: [
013                 {
014                     type: "AlexaInterface",
015                     interface: "Alexa.PowerController",
016                     version: "3",
017                     properties: {
018                         supported: [
019                             {
020                                 name: "powerState"
021                             }
022                         ]
023                     },
```

```
                    proactivelyReported: true,
                    retrievable: true
                }
            ]
        }
    ]
}

module.exports = {
    handleDiscovery: function(request, callback) {
        const oAuthToken =
             request.directive.payload.scope.token.trim();

        if (!oAuthToken || !common.isValidToken(oAuthToken)) {
            const errorMessage =
                   `Discovery Request failed. ` +
                   `Invalid access token: ${oAuthToken}`;
            callback(new Error(errorMessage));
        }

        const response = common.generateResponse(
            'Discover.Response',
            USER_DEVICES,
            'Alexa.Discovery');

        callback(null, response);
    }
}
```

control.js

```
const common = require('./common');

function turnOn(endpointId) {
    common.log('INFO', 'turning on ${endpointId}');
  common.turnOn(endpointId);
}

function turnOff(endpointId) {
    common.log('INFO', 'turning on ${endpointId}');
  common.turnOff(endpointId);
}

module.exports = {
    handleControl: function(req, callback) {
        const oAuthToken =
             req.directive.endpoint.scope.token.trim();

        if (!oAuthToken ||
             !common.isValidToken(oAuthToken)) {
            callback(null, common.generateResponse(
                   'InvalidAccessTokenError',
                   {})
```

```
023             );
024         return;
025     }
026
027     const endpointId = req.directive.endpoint.endpointId;
028
029     if (!endpointId) {
030         const payload = {
031                 faultingParameter: 'endpointId'
032             };
033         callback(null, common.generateResponse(
034                 'UnexpectedInformationReceivedError',
035                 payload)
036             );
037         return;
038     }
039
040     let res;
041
042     switch (req.directive.header.name) {
043         case 'TurnOn':
044             res = turnOn(endpointId);
045             break;
046
047         case 'TurnOff':
048             res = turnOff(endpointId);
049             break;
050
051         case 'SetPercentageRequest': {
052             const percentage =
053                     req.directive.payload.percentageState.value;
054             if (!percentage) {
055                 const payload = {
056                         faultingParameter: 'percentageState'
057                     };
058                 callback(null, common.generateResponse(
059                         'UnexpectedInformationReceivedError',
060                         payload)
061                     );
062                 return;
063             }
064             res = setPercentage(endpointId, percentage);
065             break;
066         }
067
068         case 'IncrementPercentageRequest': {
069             const delta =
070                     req.directive.payload.deltaPercentage.value;
071             if (!delta) {
072                 const payload = {
073                         faultingParameter: 'deltaPercentage'
074                     };
```

```
075                     callback(null, generateResponse(
076                                 'UnexpectedInformationReceivedError',
077                                 payload)
078                             );
079                     return;
080                 }
081                 res = incrementPercentage(endpointId, delta);
082                 break;
083             }
084 
085             case 'DecrementPercentageRequest': {
086                 const delta =
087                         req.directive.payload.deltaPercentage.value;
088                 if (!delta) {
089                     const payload = {
090                                 faultingParameter: 'deltaPercentage'
091                             };
092                     callback(null, common.generateResponse(
093                                 'UnexpectedInformationReceivedError',
094                                 payload)
095                             );
096                     return;
097                 }
098                 res = decrementPercentage(endpointId, delta);
099                 break;
100             }
101 
102             default: {
103                 res = common.generateResponse(
104                         'UnsupportedOperationError',
105                         {}
106                     );
107                     break;
108             }
109         }
110 
111         callback(null, res);
112     }
113 };
```

B: Source code of the custom Lambda

```
001 'use strict';
002
003 const Alexa = require('ask-sdk-core');
004 const Requester = require('./requester.js');
005
006 const LaunchRequestHandler = {
007     canHandle(handlerInput) {
008         return handlerInput.requestEnvelope.request.type
009               === 'LaunchRequest';
010     },
011     handle(handlerInput) {
012         const speechText =
013               'Welcome, say "number" and a number or "nothing"!';
014
015         return handlerInput.responseBuilder
016             .speak(speechText)
017             .reprompt(speechText)
018             .getResponse();
019     }
020 };
021
022 const SessionEndedRequestHandler = {
023     canHandle(handlerInput) {
024         return handlerInput.requestEnvelope.request.type
025               === 'SessionEndedRequest';
026     },
027     handle(handlerInput) {
028         return handlerInput.responseBuilder
029             .speak('Bye!')
030             .getResponse();
031     }
032 };
033
034 const CancelAndStopIntentHandler = {
035     canHandle(handlerInput) {
036         return handlerInput.requestEnvelope.request.type
037               === 'IntentRequest'
038             && (handlerInput.requestEnvelope.request.intent.name
039                         === 'AMAZON.CancelIntent'
040                 || handlerInput.requestEnvelope.request.intent.name
041                         === 'AMAZON.StopIntent');
042     },
043     handle(handlerInput) {
044         const speechText = 'Bye!';
045
046         return handlerInput.responseBuilder
047             .speak(speechText)
048             .getResponse();
```

```
049     }
050 };
051
052 const HelpIntentHandler = {
053     canHandle(handlerInput) {
054         return handlerInput.requestEnvelope.request.type
055               === 'IntentRequest'
056             && handlerInput.requestEnvelope.request.intent.name
057               === 'AMAZON.HelpIntent';
058     },
059     handle(handlerInput) {
060         const speechText =
061               'You can say "number" and a number or "nothing"!';
062
063         return handlerInput.responseBuilder
064             .speak(speechText)
065             .reprompt(speechText)
066             .getResponse();
067     }
068 };
069
070 const NothingIntentHandler = {
071     canHandle(handlerInput) {
072         return handlerInput.requestEnvelope.request.type
073               === 'IntentRequest'
074             && handlerInput.requestEnvelope.request.intent.name
075               === 'NothingIntent';
076     },
077     handle(handlerInput) {
078         const speechText = 'Doing nothing';
079
080         return handlerInput.responseBuilder
081             .speak(speechText)
082             .getResponse();
083     }
084 };
085
086 const NumberIntentHandler = {
087     canHandle(handlerInput) {
088         return handlerInput.requestEnvelope.request.type
089               === 'IntentRequest'
090             && handlerInput.requestEnvelope.request.intent.name
091               === 'NumberIntent';
092     },
093     handle(handlerInput) {
094         const number = handlerInput.requestEnvelope.request
095               .intent.slots.number.value;
096
097         if (number === undefined || number === '?') {
098             return handlerInput.responseBuilder
099                 .speak('Which number?')
100                 .addElicitSlotDirective('number')
```

```
                .getResponse();
        } else {
            const speechText = 'Number ' + number;

            Requester.request(number);

            return handlerInput.responseBuilder
                .speak(speechText)
                .getResponse();
        }
    }
};

exports.handler = Alexa.SkillBuilders.custom()
    .addRequestHandlers(
        LaunchRequestHandler,
        NothingIntentHandler,
        NumberIntentHandler,
        HelpIntentHandler,
        CancelAndStopIntentHandler,
        SessionEndedRequestHandler)
    .lambda();
```