AMAZON ALEXA: ANATOMY OF A SKILL

Amazon Alexa | Amazon Echo | AWS Lambda

beSharp | 31 May 2019

How many of you have an **Alexa-enabled device** or a voice assistant?

These devices create a way of interacting with services and applications that are completely different from what we are used to.

Through **Alexa**, Amazon has become one of the main players in a real revolution of user experience.

In this article, we will explain in detail **how an Alexa Skills works**. We will also provide you with an **interaction model** plus the **code snippet** for creating a minimal back-end.

Definitions are important. Before going further, we will introduce the terms and concepts that we are going to use throughout the following paragraphs of this article.

What is Alexa?

Alexa is **Amazon's cloud-based voice assistant** that enables the functioning of the **Amazon Echo.** In addition to Amazon Echo, it is possible to find a wide range of devices that support Alexa. You can also build new ones using AVS (Alexa Voice Service: the kit and the APIs offered by Amazon).

Alexa Skills

Alexa Skills are the "skills" or the **applications that enhance its capacity.** Everything that Alexa does is implemented in the Skills. Alexa has specific predefined abilities available in all supported languages, such as the clock, lists, timers, and reminders. All the others, however, are developed by third parties and collected in a catalog that is similar to an app store. To activate a third-party skill, search the Alexa app catalog and enable it for your own devices. Alternatively, activate a skill only through voice. For example, this can be done by asking Amazon Echo to start it.

The anatomy of a skill

Essentially, a skill is composed of two parts:

1. **An interaction model**
2. **A back-end API**

The interaction model consists of a **json file that models all intents,** i.e., the type of actions made possible for users in which the skill can manage. You will need to define the parameters for each inquiry, including their type and the phrases that Alexa needs for requesting them. The model also contains many example sentences for each intent **(utterances)** and phrases that will be used by the AI in recognizing user requests through its own skill.

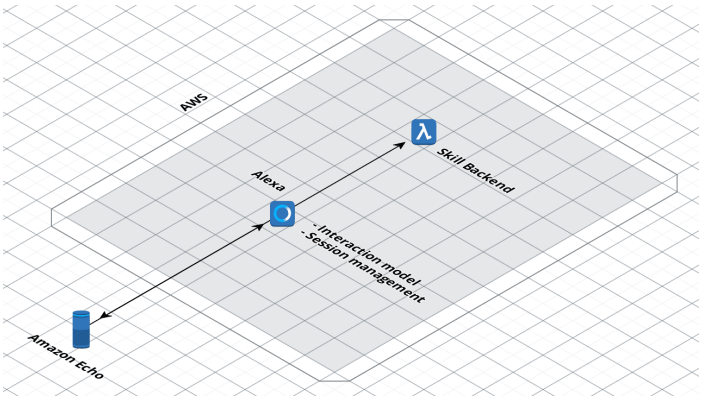
The skill's back-end is a web service.

It can be hosted on **AWS Lambda.** This enables easy authentication of Alexa and has a rather attractive pricing model.

Alternatively, it can be hosted on any service. This means it can expose a public endpoint accessible through https.

Regardless of the hosting choice, the back-end must be accessible for Alexa calls. It needs to reply within a few seconds with a message containing instructions for Alexa, e.g. the synthesized user text and/or another instruction (IoT commands, data to be shown on any display, etc.).

We will go further into how the back-end functions in a few paragraphs.



Here is how a high-level skill interaction works:

When the user speaks to an Echo device, it **records the audio and sends it to Alexa via AVS.** At this point, Alexa will be able to identify and address the skill in one of the following ways:

1. The request explicitly contains the name of the desired skill, e.g. "Alexa ask [name of skill] about ..." or similar forms.
2. The user has already enabled a skill that can satisfy the expressed intent. In this case, Alexa will select this skill.
3. If the user has not enabled any skills capable of satisfying the request, then the default intent skill will be chosen (selected by Amazon).

Once the user's intended skill has been identified, then the programmer-developed interaction model comes into play.

Alexa uses the sample sentences and intent definition parameters to understand what the user wants. If the request fails to contain all the necessary parameters, then the user will be interrogated until all information is provided.

Once the first phase of the interaction is complete, Alexa contacts the application back-end via a POST. It provides all the details and waits for the output to be communicated to the user.

For complicated skills, it is possible to keep the rules and the logic for retrieving the parameters in the back-end. This enables them to be modified according to what was acquired up to that point.

Alexa's functions include **maintaining session data.** This makes maintaining the status of a conversation possible by inserting and modifying variables in a specific section of the message returned by the back-end. This information will be sent by Alexa to subsequent calls in the back-end until the user concludes the interaction with the skill.

You can then create complex skills that "remember" the user's previous answers or requests. This makes for **a realistic conversation.**

A trivial skill

So far, we have not gone into too much detail when talking about the interaction model and the back-end.

So here is an example:

```
{
    "interactionModel": {
        "languageModel": {
            "invocationName": "saluti",
            "intents": [
                {
                    "name": "AMAZON.CancelIntent",
                    "samples": [
                        "no",
                        "niente",
                        "non fa niente",
                        "lascia perdere",
                        "annulla"
                    ]
                },
                {
                    "name": "AMAZON.HelpIntent",
                    "samples": []
                },
                {
                    "name": "AMAZON.StopIntent",
                    "samples": [
                        "abbandona",
                        "esci",
                        "fine",
                        "stop"
                    ]
                },
```

```
                {
                    "name": "Salutare",
                    "slots": [],
                    "samples": [
                        "salutarmi",
                        "salutami",
                        "saluti",
                        "ciao"
                    ]
                }
            ],
            "types": []
        }
    }
}
```

This model defines a single intent with the “Greeting” identification, plus the standard Amazon intent of “cancel”, “stop”, and “help”. No parameters are required. Rather, a few sample phrases are provided.

It is a **working model** in its simplicity. Additionally, it fulfills the purpose of being greeted by Alexa.

The back-end, meaning the service that is able to respond to the requests that Alexa performs once the intent has been identified is, however, missing.

Here is a snippet to correctly answer calls related to the intent defined above, assuming the use of AWS Lambda.

It involves a **Nodejs function,** whose code uses the Alexa SDK for each node. In red, we have highlighted in red the function that is invoked for the “Greeting” intent.

The rest of the code is essentially a boilerplate. It remains almost completely unaltered, even for more complex skills.

```
let speechOutput;
let reprompt;
let welcomeOutput = "Benvenuto in 'Ciao' in italiano. Puoi salutarmi per essere saluta
to a tua volta";
let welcomeReprompt = "puoi solo dire 'ciao' per essere salutato";

"use strict";
const Alexa = require('alexa-sdk');
const APP_ID = undefined;
speechOutput = '';
const handlers = {
'LaunchRequest': function() {
this.emit(':ask', welcomeOutput, welcomeReprompt);
},
'AMAZON.HelpIntent': function() {
speechOutput = 'Placeholder response for AMAZON.HelpIntent.';
reprompt = '';
this.emit(':ask', speechOutput, reprompt);
},
```

```
'AMAZON.CancelIntent': function() {
speechOutput = 'Ok, annullato’;
this.emit(':tell', speechOutput);
},
'AMAZON.StopIntent': function() {
speechOutput = 'Arrivederci';
this.emit(':tell', speechOutput);
},
'SessionEndedRequest': function() {
speechOutput = '';
this.emit(':tell', speechOutput);
},
'Salutare': function() {
speechOutput = '';
speechOutput = "Ciao! Questo è tutto quello che puoi fare per ora”;
this.emit(":tell", speechOutput, speechOutput);
},
'Unhandled': function() {
speechOutput = "Non ho capito. Riprova";
this.emit(':ask', speechOutput, speechOutput);
}
};

exports.handler = (event, context) => {
const alexa = Alexa.handler(event, context);
alexa.appId = APP_ID;
// To enable string internationalization (i18n) features, set a resources object.
//alexa.resources = languageStrings;
alexa.registerHandlers(handlers);
//alexa.dynamoDBTableName = 'DYNAMODB_TABLE_NAME'; //uncomment this line to save attri
butes to DB
alexa.execute();
};

// END of Intent Handlers {} ========================================================
==============================
// 3. Helper Function ===============================================================
==================================

function resolveCanonical(slot) {
//this function looks at the entity resolution part of request and returns the slot va
lue if a synonyms is provided
let canonical;
try {
canonical = slot.resolutions.resolutionsPerAuthority[0].values[0].value.name;
}
catch (err) {
console.log(err.message);
canonical = slot.value;
};
return canonical;
};

function delegateSlotCollection() {
console.log("in delegateSlotCollection");
console.log("current dialogState: " + this.event.request.dialogState);
if (this.event.request.dialogState === "STARTED") {
console.log("in Beginning");
let updatedIntent = null;
// updatedIntent=this.event.request.intent;
```

```
//optionally pre-fill slots: update the intent object with slot values for which
//you have defaults, then return Dialog.Delegate with this updated intent
// in the updatedIntent property
//this.emit(":delegate", updatedIntent); //uncomment this is using ASK SDK 1.0.9 or ne
wer

//this code is necessary if using ASK SDK versions prior to 1.0.9
if (this.isOverridden()) {
return;
}
this.handler.response = buildSpeechletResponse({
sessionAttributes: this.attributes,
directives: getDialogDirectives('Dialog.Delegate', updatedIntent, null),
shouldEndSession: false
});
this.emit(':responseReady', updatedIntent);

}
else if (this.event.request.dialogState !== "COMPLETED") {
console.log("in not completed");
// return a Dialog.Delegate directive with no updatedIntent property.
//this.emit(":delegate"); //uncomment this is using ASK SDK 1.0.9 or newer

//this code necessary is using ASK SDK versions prior to 1.0.9
if (this.isOverridden()) {
return;
}
this.handler.response = buildSpeechletResponse({
sessionAttributes: this.attributes,
directives: getDialogDirectives('Dialog.Delegate', null, null),
shouldEndSession: false
});
this.emit(':responseReady');

}
else {
console.log("in completed");
console.log("returning: " + JSON.stringify(this.event.request.intent));
// Dialog is now complete and all required slots should be filled,
// so call your normal intent handler.
return this.event.request.intent;
}
}


function randomPhrase(array) {
// the argument is an array [] of words or phrases
let i = 0;
i = Math.floor(Math.random() * array.length);
return (array[i]);
}

function isSlotValid(request, slotName) {
let slot = request.intent.slots[slotName];
//console.log("request = "+JSON.stringify(request)); //uncomment if you want to see th
e request
let slotValue;

//if we have a slot, get the text and store it into speechOutput
if (slot && slot.value) {
```

```
//we have a value in the slot
slotValue = slot.value.toLowerCase();
return slotValue;
}
else {
//we didn't get a value in the slot.
return false;
}
}

//These functions are here to allow dialog directives to work with SDK versions prior
 to 1.0.9
//will be removed once Lambda templates are updated with the latest SDK

function createSpeechObject(optionsParam) {
if (optionsParam && optionsParam.type === 'SSML') {
return {
type: optionsParam.type,
ssml: optionsParam['speech']
};
}
else {
return {
type: optionsParam.type || 'PlainText',
text: optionsParam['speech'] || optionsParam
};
}
}

function buildSpeechletResponse(options) {
let alexaResponse = {
shouldEndSession: options.shouldEndSession
};

if (options.output) {
alexaResponse.outputSpeech = createSpeechObject(options.output);
}

if (options.reprompt) {
alexaResponse.reprompt = {
outputSpeech: createSpeechObject(options.reprompt)
};
}

if (options.directives) {
alexaResponse.directives = options.directives;
}

if (options.cardTitle && options.cardContent) {
alexaResponse.card = {
type: 'Simple',
title: options.cardTitle,
content: options.cardContent
};

if (options.cardImage && (options.cardImage.smallImageUrl || options.cardImage.largeI
mageUrl)) {
alexaResponse.card.type = 'Standard';
alexaResponse.card['image'] = {};
```

```
delete alexaResponse.card.content;
alexaResponse.card.text = options.cardContent;

if (options.cardImage.smallImageUrl) {
alexaResponse.card.image['smallImageUrl'] = options.cardImage.smallImageUrl;
}

if (options.cardImage.largeImageUrl) {
alexaResponse.card.image['largeImageUrl'] = options.cardImage.largeImageUrl;
}
}
}
else if (options.cardType === 'LinkAccount') {
alexaResponse.card = {
type: 'LinkAccount'
};
}
else if (options.cardType === 'AskForPermissionsConsent') {
alexaResponse.card = {
type: 'AskForPermissionsConsent',
permissions: options.permissions
};
}

let returnResult = {
version: '1.0',
response: alexaResponse
};

if (options.sessionAttributes) {
returnResult.sessionAttributes = options.sessionAttributes;
}
return returnResult;
}

function getDialogDirectives(dialogType, updatedIntent, slotName) {
let directive = {
type: dialogType
};

if (dialogType === 'Dialog.ElicitSlot') {
directive.slotToElicit = slotName;
}
else if (dialogType === 'Dialog.ConfirmSlot') {
directive.slotToConfirm = slotName;
}

if (updatedIntent) {
directive.updatedIntent = updatedIntent;
}
return [directive];
}
```

Here, we have covered the fundamental concepts underlying the Alexa functioning. We then analyzed the anatomy of **a simple and functional skill,** which is the first step in realizing more complex and potentially publishable skills.

However, for publication, there are compliance limitations and rules to be considered when designing a voice interface.

We will address this and many other important aspects of the publication and implementation phase throughout the following articles. But first, we will publish a detailed tutorial for the creation of a non-“trivial” skill.

We will not give you any hints 🙂

Stay tuned!

Would you like to learn more about **AWS Lambda or Serverless development?**

Then read our series of articles on these topics!

beSharp

Dal 2011 beSharp guida le aziende italiane sul Cloud. Dalla piccola impresa alla grande multinazionale, dal manifatturiero al terziario avanzato, aiutiamo le realtà più all'avanguardia a realizzare progetti innovativi in campo IT.

Get in touch

beSharp.it
proud2becloud@besharp.it