AMAZON ALEXA: ANATOMIA DI UNA SKILL

Amazon Alexa | Amazon Echo | AWS Lambda

beSharp | 31 Maggio 2019

Quanti di voi possiedono un **dispositivo Alexa enabled** o comunque un assistente vocale?

Questi dispositivi stanno creando un modo di interagire con servizi ed applicazioni del tutto diverso da quanto siamo abituati a conoscere.

Amazon, con **Alexa**, è uno dei principali attori di quella che sta diventando una rivoluzione dell'esperienza utente.

In questo articolo spiegheremo ad alto livello **il funzionamento di una Alexa Skill** e vi forniremo un **modello di interazione**, oltre allo **snippet di codice** necessario per creare un back-end minimale.

Prima di entrare nei dettagli è importante definire un vocabolario; nei paragrafi seguenti andremo ad introdurre i termini e i concetti che utilizzeremo in tutto il resto dell'articolo.

Cos'è Alexa?

Alexa è l'**assistente vocale di Amazon** basato su cloud e che consente il funzionamento di **Amazon Echo.** Oltre ad Amazon Echo è possibile trovare una vasta gamma di dispositivi che supportano Alexa. Inoltre è anche possibile costruirne di nuovi utilizzando AVS (Alexa voice service: il kit e le API messe a disposizione da Amazon).

Alexa Skill

Le skill sono le "abilità", ovvero le **applicazioni che aggiungono capacità ad Alexa.**Ogni cosa che Alexa può fare viene implementata in una Skill.Alexa dispone di alcune skill predefinite e disponibili in tutte le lingue supportate, come ad esempio l'orologio, le liste, i timer e i promemoria. Tutte le altre, invece, sono sviluppate da terze parti e raccolte in un catalogo del tutto simile ad un application store. Per attivare una skill di terze parti occorre cercarla sul catalogo dall'app Alexa ed abilitarla per i propri dispositivi, in alternativa è possibile attivare una skill usando solo la voce, ad esempio chiedendo esplicitamente ad un Amazon Echo di avviarla.

L’anatomia di una skill

Entrando nello specifico, una skill si compone essenzialmente di due parti:

1. **Un modello di interazione**
2. **Una API di back-end**

Il modello di interazione consiste in un **file json che modellizza tutti gli intenti,** cioè il tipo di azioni rese possibili agli utenti e che la skill può gestire. Per ogni intento è necessario definire anche quali siano i parametri necessari, il loro tipo e le frasi che Alexa dovrà utilizzare per richiederli. Il modello contiene anche molte frasi di esempio per ogni intento **(utterances),** frasi che verranno utilizzate dalla IA per riconoscere le richieste degli utenti verso la propria skill.

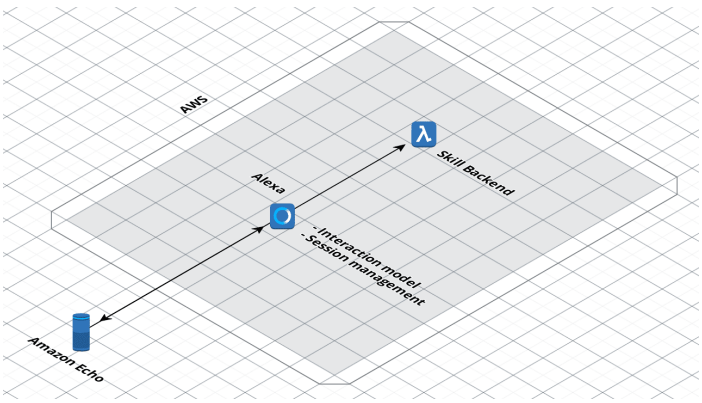
Il back-end della skill è un servizio web.

Può essere ospitato su **AWS Lambda** che permette una facile autenticazione di Alexa e ha un modello di pricing piuttosto allettante.

In alternativa può essere ospitato su qualsiasi servizio permetta di esporre un endpoint pubblico accessibile attraverso https.

Indipendentemente dalla scelta dell’hosting, il back-end dovrà essere raggiungibile dalle chiamate di Alexa. Entro pochi secondi dovrà rispondere con un messaggio contenente istruzioni per Alexa, ad esempio il testo che dovrà essere sintetizzato all’utente e/o un altro tipo di istruzione (ad esempio comandi IoT, dati da mostrare su eventuali display, ...).

Approfondiremo meglio il funzionamento del back-end tra qualche paragrafo.



Ecco come funziona l’interazione con una skill ad alto livello:

Quando l’utente parla ad un dispositivo Echo, questo **registra l’audio e lo invia ad Alexa usando AVS.** A questo punto Alexa sarà in grado di individuare la skill da interpellare in uno dei seguenti modi:

1. La richiesta contiene esplicitamente il nome della skill desiderata, ad esempio “Alexa chiedi a [nome skill] di … “ o forme simili.
2. L’utente ha già abilitato una skill in grado di soddisfare l’intento espresso; in questo caso Alexa selezionerà tale skill.
3. Se l’utente non ha abilitato nessuna skill in grado di soddisfare la richiesta, verrà scelta la skill di default per l’intento (selezionata da Amazon).

Una volta identificata la skill che l’utente intende utilizzare, entra in gioco il modello di interazione sviluppato dal programmatore.

Alexa utilizza le frasi di esempio e la definizione degli intenti e dei parametri per capire cosa l’utente desidera. In caso la richiesta non contenga tutti i parametri necessari, l’utente sarà interrogato fino a quando non avrà fornito tutte le informazioni.

Una volta completata la prima fase dell’interazione, Alexa contatta il back-end dell’applicazione tramite una chiamata POST, fornendogli tutti i dettagli ed attendendo l’output da comunicare all’utente.

Per skill complicate, è possibile conservare nel back-end le regole e la logica di reperimento dei parametri, permettendo di modificarli in funzione di quanto acquisito fino a quel momento.

Tra le funzioni di Alexa vi è anche quella di **mantenere i dati di sessione:** sarà quindi possibile mantenere lo stato di una conversazione inserendo e modificando variabili in un’apposita sezione del messaggio restituito dal back-end. Queste informazioni saranno inviate da Alexa alle successive chiamate al back-end fino a quando l’utente non concluderà l’interazione con la skill.

Sfruttando la sessione è pertanto possibile realizzare skill complesse che “ricordano” le risposte o le richieste precedenti dell’utente, realizzando di fatto **una conversazione realistica.**

Una skill triviale

Finora abbiamo parlato del modello di interazione e del back-end senza entrare troppo nel dettaglio.

Ecco pertanto un modello di esempio

```
{
    "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": []
        }
    }
}
```

Questo modello definisce un solo intento con identificativo “Salutare”, oltre agli intenti standard di Amazon “cancel”, “stop” ed “help”. Non sono necessari parametri, e sono fornite poche frasi di esempio.

Nella sua semplicità, si tratta di un **modello funzionante** ed in grado di assolvere allo scopo di farsi salutare da Alexa.

Manca tuttavia il back-end, ovvero il servizio che è in grado di rispondere alle richieste che Alexa esegue una volta individuato l’intento.

Supponendo di utilizzare AWS Lambda, ecco uno snippet per rispondere correttamente alle chiamate relative all’intento definito sopra.

Si tratta di una **funzione Nodejs,** il cui codice utilizza l’sdk di Alexa per node. Abbiamo evidenziato la funzione che viene invocata per l’intento “Salutare” in rosso.

Il resto del codice è sostanzialmente solo boilerplate e rimane quasi completamente invariato anche in caso di skill più complesse.

```
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];
}
```

In questo articolo abbiamo trattato i concetti fondamentali che stanno alla base del funzionamento di Alexa. Abbiamo poi analizzato l'anatomia di **una skill semplice e funzionante,** il primo passo per la realizzazione di skill più complesse e potenzialmente pubblicabili.

Per la pubblicazione vi sono però vincoli di compliance e regole di cui tenere conto durante la progettazione di un'interfaccia vocale.

Di questo e di molti altri aspetti importanti sulla fase di pubblicazione e realizzazione ci occuperemo nei prossimi articoli. Ma prima pubblicheremo un tutorial dettagliato per la realizzazione di una skill... non "banale".

Non vi anticipiamo nulla 🙂

Stay tuned!

Vuoi approfondire l'uso di AWS Lambda o saperne di più sullo sviluppo Serverless? Leggi la nostra serie di articoli dedicati!

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