# Amazon Web Service:

## Wofür wird AWS gebraucht?

„*Amazon Web Services (AWS) ist eine sichere Plattform für Cloud-Services, die Rechenleistung, Datenbankspeicherung, Bereitstellen von Inhalten und weitere Funktionen bietet, und das Wachstum Ihres Unternehmens zu unterstützen.*“ - https://aws.amazon.com/de/what-is-aws/

AWS ist in unserem Fall zuständig für die Bearbeitung von Sprachbefehlen. Die Sprachbefehle, welche im Skillset definiert werden, werden wenn erkannt von Alexa an AWS geschickt und dort die damit verbunden Aktionen ausgelöst. AWS liefert an Alexa eine Rückmeldung zurück (z.B.: in Form einer Sprachausgabe) und führt den im Javascript definierten, mit dem Befehl verbunden Code aus. Im Prinzip ist das AWS die Entscheidungslogik, was bei einem Befehl passiert.

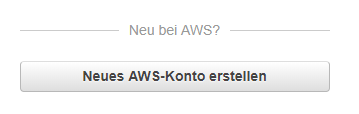
## Einrichten von AWS:

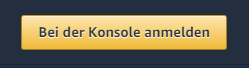
Um die Funktionen von AWS nutzen zu können sind folgende Sachen notwendig:

* Einrichtung eines Kontos auf <https://aws.amazon.com>
* Eine Kreditkarte

HINWEIS: Die Kreditkarte wird benötigt um die Echtheit des AWS-Kontos zu verifizieren. Auch bei einem kostenfreien Entwicklungspaket! Von der Kreditkarte wird 1 Dollar abgebucht und nach der Verifizierung wieder auf der Kreditkarte gutgeschrieben.

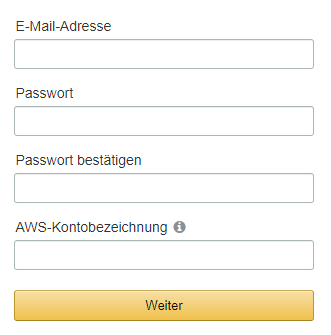
**ANLEGEN EINES KONTOS:**

Zum Anlegen eines AWS-Entwicklungskontos muss man sich zunächst unter <https://aws.amazon.com> ein Konto erstellen.

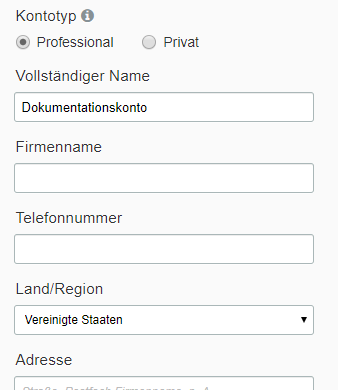


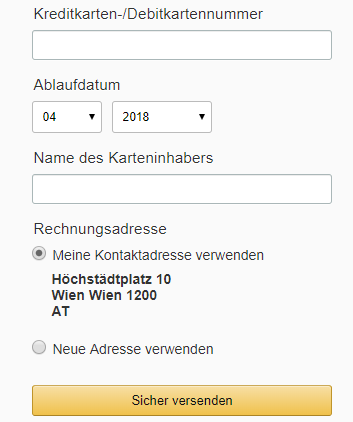
🡪

Bei der Registrierung muss man nun die eigene Email-Adresse, ein Passwort und einen Benutzernamen (AWS-Kontebezeichnung) angeben.



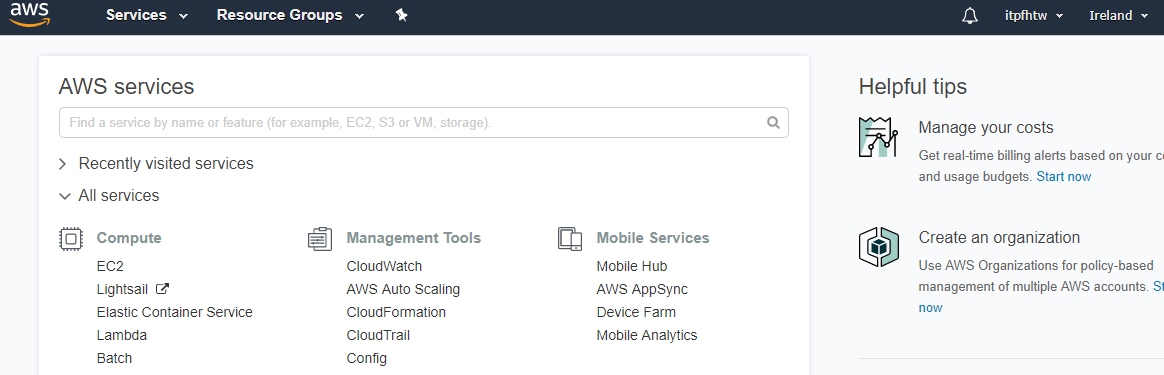
Nun werden von einem die Kontaktinformation benötigt, wobei diese alle verpflichtende Angaben sind.

Als Firmenname und Adresse haben wir die FHTW und ihre Adresse angegeben.

Zum Abschluss muss man nun die Kreditkarteninformationen angeben, über welche das Konto verifiziert wird.

**Die Verifizierung des Kontos dauert nun 24 Stunden**, erst dann kann man darauf zugreifen.

Wenn dies geschehen ist, muss man sich nur noch einloggen. („In der Konsole einloggen“) und nun sollte man diese Anzeige sehen:



## Arbeiten mit AWS:

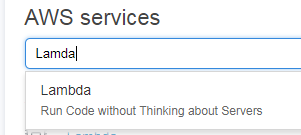
Um mit AWS arbeiten zu können muss zuerst der Skill fertig gestellt werden, dazu aber später mehr.

Zunächst brauchen wir eine Backend-Logik, die aus unseren Sprachbefehlen die aus dem angelegten Skill kommen, eine Aktion machen die ausgeführt wird.

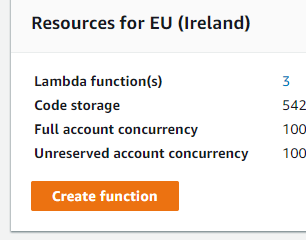
Hierfür werden die sogenannten LAMBDA-Funktionen verwendet.

**ANLEGEN EINES LAMBDA BACKEND:**

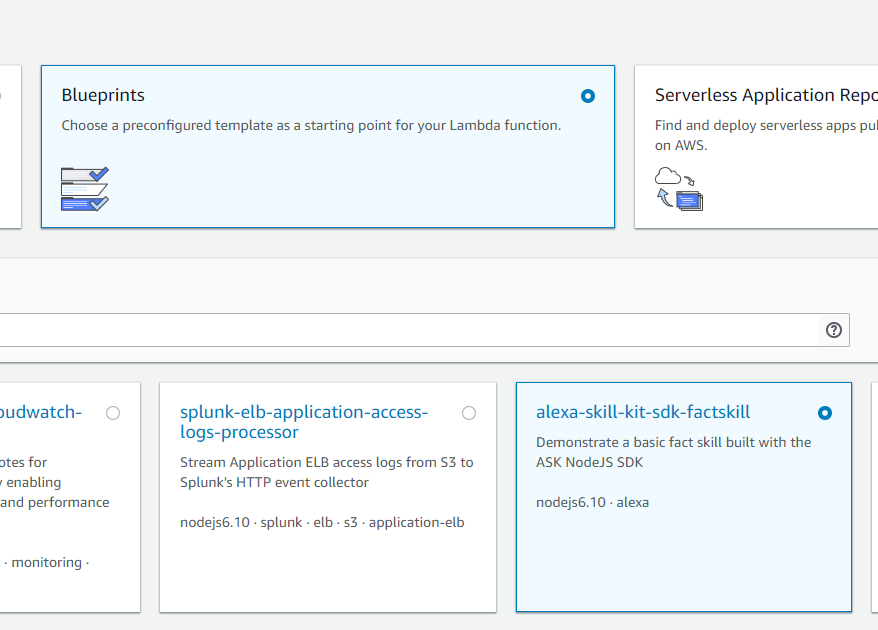
Auf der Startseite von AWS findet man alle möglichen Arten von Services. Um zu den Lambdafunktionen zu kommen sucht man diese zunächst in der Suchleiste und wählt sie aus:



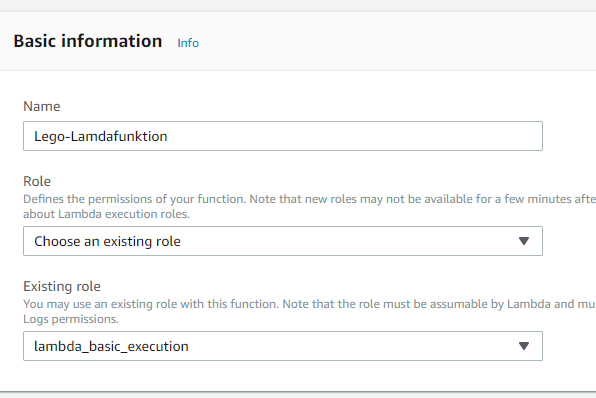
Danach muss man eine neue Lamdafunktion erstellen:



Hier kann man sich nun entscheiden, ob man den ganzen Code selber schreiben will (Author from scratch), oder ob man vorgefertigte „Blueprints“ verwendet, die schon die wichtigsten Libraries mit eingebunden haben. In unserem Fall haben wir uns für die Blueprints entschieden (genauer gesagt den „alexa-skill-kit-sdk-factskill“) da wir uns hiermit einiges an Arbeit ersparen.

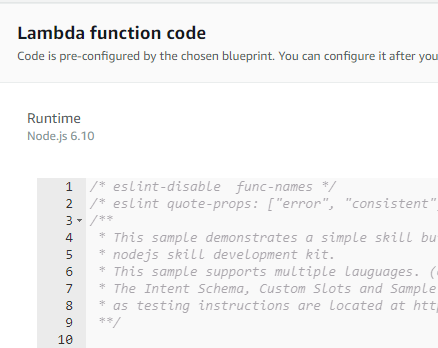


Nun muss man einige Angaben zu der Lambdafunktion machen:



Im Namen tragen wir eine Bezeichnung ein, wie unsere Funktion heißen soll. Dies hat keine Auswirkung auf die Funktion selbst, sondern dient nur der Übersicht, wenn man mehrere davon schreiben sollte.

Bei der Rolle wählen wir aus „Choose an existing role“. Darunter wählt man nun „lambda\_basic\_execution“ aus. Für diese Rolle sind schon durch den Blueprint alle Rechte vergeben, die die Funktion braucht um den Code ausführen zu können.

Zuletzt sieht man Feld in das Code rein gespielt werden kann, dieses Feld wird aber zunächst mal so gelassen wie es ist. Der Code kann später noch geändert werden.

Nun geht man auf „create function“ und kommt zur Einstellungsseite der Lambdafunktion.

## Die Lambda-Funktion

Da die Lambda Funktion auch programmiert werden muss ist hier der auskommentierte Programmcode um zu verstehen wie Lambda mit SNS und Amazon arbeitet.

*//*Verbindung zu unserem SNS aufbauen

var snsArn = "arn:aws:sns:eu-west-1:697251208257:LEGO\_FHTW";

var AWS = require('aws-sdk');

AWS.config.region = 'eu-west-1';

*//*Starte Funktion für das Programm

exports.handler = function (*event*, *context*) {

var contextMessage;

var sessionMessage;

*//*Versuche Session zu starten oder Requests abzurufen

try {

console.log("event.session.application.applicationId=" + event.session.application.applicationId);

*//*Wenn eine neue Session gestartat wird

if (event.session.new) {

onSessionStarted({requestId: event.request.requestId}, event.session);

}

*//*Nach dem Start findet der LaunchRequest statt

if (event.request.type === "LaunchRequest") {

onLaunch(event.request,

event.session,

function callback(*sessionAttributes*, *speechletResponse*) {

contextMessage = buildResponse(sessionAttributes, speechletResponse);

sessionMessage = sessionAttributes;

});

*//*Wenn nach einem Intent gefragt wird

} else if (event.request.type === "IntentRequest") {

onIntent(event.request,

event.session,

function callback(*sessionAttributes*, *speechletResponse*) {

contextMessage = buildResponse(sessionAttributes, speechletResponse);

sessionMessage = sessionAttributes;

});

*//*Falls die Session beendet werden soll

} else if (event.request.type === "SessionEndedRequest") {

onSessionEnded(event.request, event.session);

}

} catch (e) {

*//*Bei Fehler diesen ausgeben

context.fail("Exception: " + e);

}

*//*Speichere den Befehl in eventText als string

var eventText = JSON.stringify(sessionMessage, null, 2);

console.log("Received event:", eventText);

*//*Erzeuge eine neue SNS-Klasse

var sns = new AWS.SNS();

*//*Speichere die Werte in eine Liste params

var params = {

Message: eventText,

Subject: "EV3 Alexa Command",

TopicArn: snsArn

};

console.log('context message: ');

console.log(contextMessage);

*//*Sende die Liste sammt Informationen an SNS

sns.publish(params, function (*err*, *data*) {

console.log('published data: ');

console.log(data);

context.succeed(contextMessage);

});

*//*Funktion die überprüft ob kein Timeout passiert ist beim Senden der Nachricht an SNS

var timeout = new Date().getTime();

while ((new Date().getTime()) - timeout > 500) {}

};

*//*Die Funktion wird zum Start einer Session aufgerufen

function onSessionStarted(*sessionStartedRequest*, *session*) {

console.log("onSessionStarted requestId=" + sessionStartedRequest.requestId +

", sessionId=" + session.sessionId);

}

*//*Wenn der User einen Skill aufruft ohne genau diesem mitzuteilen was dieser tun soll

function onLaunch(*launchRequest*, *session*, *callback*) {

console.log("onLaunch requestId=" + launchRequest.requestId +

", sessionId=" + session.sessionId);

getWelcomeResponse(callback);

}

*//*Wird aufgerufen wenn der User einen Intent von einem Skill abfragt

function onIntent(*intentRequest*, *session*, *callback*) {

console.log("onIntent requestId=" + intentRequest.requestId +

", sessionId=" + session.sessionId);

*//*Speichere Intent, sowie den Intentnamen

var intent = intentRequest.intent,

intentName = intentRequest.intent.name;

*//*Nach dem Intent suchen, welcher verwendet werden soll

if ("CommandIntent" === intentName) {

setCommandSession(intent, session, callback);

} else if ("CommandIntentWithValue" === intentName) {

setCommandWithValueSession(intent, session, callback);

} else if ("StopIntent" === intentName) {

endCommandSession(intent, session, callback);

endCommandWithValueSession(intent, session, callback);

} else if ("AMAZON.HelpIntent" === intentName) {

getWelcomeResponse(callback);

} else {

*//*Error bei Fehler

throw "Kein gültiger Intent.";

}

}

*//*Wird aufgerufen wenn der User die Session beendet

*//*Wird nicht bei shouldEndSession=true aufgerufen

function onSessionEnded(*sessionEndedRequest*, *session*) {

console.log("onSessionEnded requestId=" + sessionEndedRequest.requestId +

", sessionId=" + session.sessionId);

}

*//* --------------- Funktionen zum Verhalten des Skills -----------------------

*//*Funktion mit Willkommensnachricht

function getWelcomeResponse(*callback*) {

var sessionAttributes = {};

var cardTitle = "Willkommen bei der Lego Kontrollsteuerung.";

var speechOutput = "Willkommen bei der Lego Kontrollsteuerung. " +

"Sage mir wie ich deinen Roboter steuern soll. " +

        "Zum Beispiel fahre gerade aus.";

var repromptText = "Sage mir wie ich deinen Roboter steuern soll. " +

        "Zum Beispiel fahre gerade aus.";

var shouldEndSession = false;

callback(sessionAttributes,

buildSpeechletResponse(cardTitle, speechOutput, repromptText, shouldEndSession));

}

*//*Funktion die die Variblen speichert, überpüft und weiter verarbeitet

function setCommandSession(*intent*, *session*, *callback*) {

var cardTitle = "Command";

*//*Speichere die Variablen aus den Slots

var deviceNameSlot = intent.slots.deviceName;

var actionSlot = intent.slots.action;

var optionSlot = intent.slots.option;

var repromptText = "";

var sessionAttributes = {};

var shouldEndSession = false;

var speechOutput = "";

*//*Es müssen 4 Variablen existieren dass heißt not null sein damit ein Command with

*//*Value erfüllt ist. Das wird hier überprüft und bei einem Fehler eine Rückmeldung

*//*ausgegeben

if(deviceNameSlot) {

var device = deviceNameSlot.value;

var action = null;

var option = null;

if (actionSlot && actionSlot.value != null) {

action = actionSlot.value;

if (optionSlot && optionSlot.value != null) {

option = optionSlot.value;

*//*Erzeugt eine Liste und speichert den Befehl als string ab

sessionAttributes = createCommandAttributes(device, action, option);

speechOutput = device+" "+action+" "+option;

repromptText = "";

} else {

speechOutput = "Ich bin mir nicht sicher welche option ich durchführen soll.";

repromptText = "Ich bin mir nicht sicher welche option ich durchführen soll." +

"Probiere es mit zum beispiel links oder rechts.";

}

} else {

speechOutput = "Ich bin mir nicht sicher welche aktion ich durchführen soll.";

repromptText = "Ich bin mir nicht sicher welche aktion ich durchführen soll." +

"Probiere es mit zum beispiel gerade aus fahren.";

}

} else {

speechOutput = "Ich habe den Gerätenamen nicht gefunden";

repromptText = "Ich habe den Gerätenamen nicht gefunden" +

"Probiere EV3, Roboter, Robo und versuche es erneut.";

}

callback(sessionAttributes,

buildSpeechletResponse(cardTitle, speechOutput, repromptText, shouldEndSession));

}

*//*Erzeugt eine Liste mit den Werten des Befehls und gibt diese zurück

function createCommandAttributes(*device*, *action*, *option*) {

return {

device: device,

action: action,

        option: option

};

}

*//*Funktion zum Beenden der Session CommandSession

function endCommandSession(*intent*, *session*, *callback*) {

*//*Setze repromptText auf null da wir dem User nicht mitteilen wollen das die Session beendet wurde

var repromptText = null;

var sessionAttributes = {};

var shouldEndSession = true;

var speechOutput = "Fertig mit dem steuern des Roboters.";

*//*Wenn der User etwas sagt das nicht verstanden wird oder garnichts sagt wird die Session beendet

*//*sonst passiert eine Rückmeldung

callback(sessionAttributes,

buildSpeechletResponse(intent.name, speechOutput, repromptText, shouldEndSession));

}

*//*Funktion die die Variblen speichert, überpüft und weiter verarbeitet

function setCommandWithValueSession(*intent*, *session*, *callback*) {

var cardTitle = "CommandWithValue";

*//*Speichere die Variablen aus den Slots

var deviceNameSlot = intent.slots.deviceName;

var actionSlot = intent.slots.action;

var optionSlot = intent.slots.option;

var valueSlot = intent.slots.value;

var repromptText = "";

var sessionAttributes = {};

var shouldEndSession = false;

var speechOutput = "";

*//*Es müssen 4 Variablen existieren dass heißt not null sein damit ein Command with

*//*Value erfüllt ist. Das wird hier überprüft und bei einem Fehler eine Rückmeldung

*//*ausgegeben

if(deviceNameSlot) {

var device = deviceNameSlot.value;

var action = null;

var option = null;

var value = null;

if (actionSlot && actionSlot.value != null) {

action = actionSlot.value;

if (optionSlot && optionSlot.value != null) {

option = optionSlot.value;

if(valueSlot && valueSlot.value != null){

value = valueSlot.value;

*//*Es kommt bei JSON ein ? statt einem int an ...

*//*Erzeugt eine Liste und speichert den Befehl als string ab

if(value == "?")

{

sessionAttributes = createCommandAttributes(device, action, option);

speechOutput = device+" "+action+" "+option;

repromptText = "";

}

else

{

sessionAttributes = createCommandWithValueAttributes(device, action, option, value);

speechOutput = device+" "+action+" "+option+" "+value;

repromptText = "";

}

} else {

speechOutput = "Ich bin mir nicht sicher welche Zahl ich verstanden habe.";

repromptText = "Ich bin mir nicht sicher welche Zahl ich verstanden habe." +

"Probiere es mit zum beispiel 40 oder 10.";

}

} else {

speechOutput = "Ich bin mir nicht sicher welche option ich durchführen soll.";

repromptText = "Ich bin mir nicht sicher welche option ich durchführen soll." +

"Probiere es mit zum beispiel links oder rechts.";

}

} else {

speechOutput = "Ich bin mir nicht sicher welche aktion ich durchführen soll.";

repromptText = "Ich bin mir nicht sicher welche aktion ich durchführen soll." +

"Probiere es mit zum beispiel gerade aus fahren.";

}

} else {

speechOutput = "Ich habe den Gerätenamen nicht gefunden";

repromptText = "Ich habe den Gerätenamen nicht gefunden" +

"Probiere EV3, Roboter, Robo und versuche es erneut.";

}

callback(sessionAttributes,

buildSpeechletResponse(cardTitle, speechOutput, repromptText, shouldEndSession));

}

*//*Erzeugt eine Liste mit den Werten des Befehls und gibt diese zurück

function createCommandWithValueAttributes(*device*, *action*, *option*, *value*) {

return {

device: device,

action: action,

option: option,

value: value

};

}

*//*Funktion zum Beenden der Session CommandWithValueSession

function endCommandWithValueSession(*intent*, *session*, *callback*) {

*//*Setze repromptText auf null da wir dem User nicht mitteilen wollen das die Session beendet wurde

var repromptText = null;

var sessionAttributes = {};

var shouldEndSession = true;

var speechOutput = "Fertig mit dem Steuern des Roboters.";

*//*Wenn der User etwas sagt das nicht verstanden wird oder garnichts sagt wird die Session beendet

*//*sonst passiert eine Rückmeldung

callback(sessionAttributes,

buildSpeechletResponse(intent.name, speechOutput, repromptText, shouldEndSession));

}

*//* --------------- Funktionen die für die Rückmeldung zuständig sind -----------------------

*//*Diese Funktion wird nach jedem Befehl aufgerufen der ausgeführt wird

*//*Damit kann der User eine Rückmeldung erhalten, dass zB sein Befehl nicht verstanden wurde

function buildSpeechletResponse(*title*, *output*, *repromptText*, *shouldEndSession*) {

return {

outputSpeech: {

type: "PlainText",

text: output

},

card: {

type: "Simple",

title: "EV3 - " + title,

content: output

},

reprompt: {

outputSpeech: {

type: "PlainText",

text: repromptText

}

},

shouldEndSession: shouldEndSession

};

}

*//*Funktion bei Start des Skills, welche allgemeine Infos ausgibt

function buildResponse(*sessionAttributes*, *speechletResponse*) {

return {

version: "1.0",

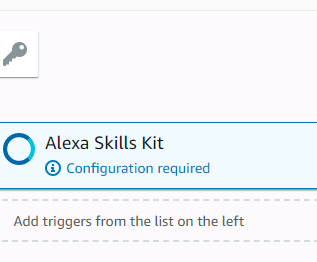
sessionAttributes: sessionAttributes,

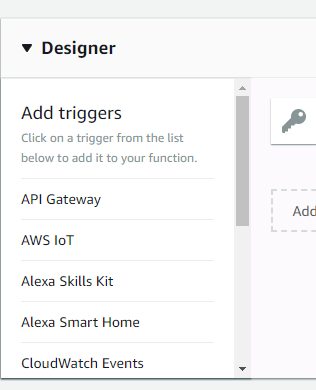
response: speechletResponse

};

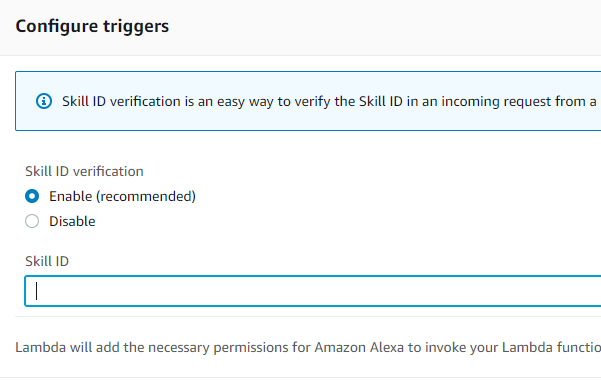
}

## Verbinden der Lambdafunktion mit dem Alexa-Skill:

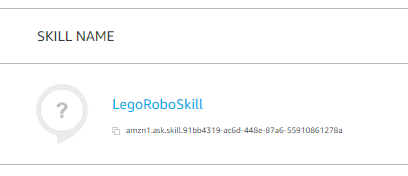
Zunächst muss die Lambdafunktion mit unserem Skillkit verbunden werden. Hierführt wählt man bei Designer das „Alexa Skills Kit“ aus.



Damit die Lambdafunktion nun weiß, welche Skills darauf zugreifen dürfen, muss man die Skill-ID angeben. Alternativ kann man auch per „Disable“ jedem Skill auf der Welt den Zugriff darauf erlauben.

Der Skill ist hier als sogenannter Trigger definiert. Dies bedeutet, dass diese Amazonfunktion eine Aktion in unserem Backend auslösen kann.

Die Skill-ID bekommt man von der Skill-Developement Seite:

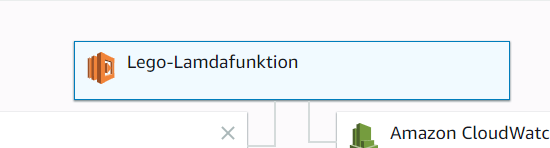
Mittels klick auf das Kopiersymbol kann die Skill-ID per einfachen Mausklick kopiert werden.

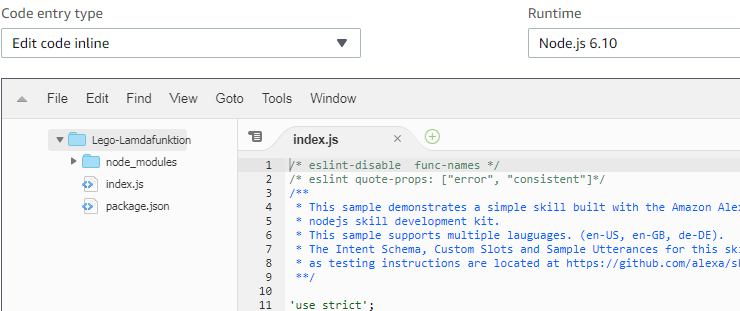
Diese muss nun bei der Lambdafunktion eingetragen werden.

**WICHTIG: Dannach klickt man auf „ADD“ am unteren rechten Rand und anschliessend auf „Safe“ rechts oben.**

Jetzt ist unser erstellter Skill mit der Funktion verbunden. Unsere Funktion hat aber noch keine echte Logik. Diese wird jetzt eingefügt.

Dazu klickt man bei dem Reiter „Designer“ auf den Button mit dem Namen unserer Funktion:

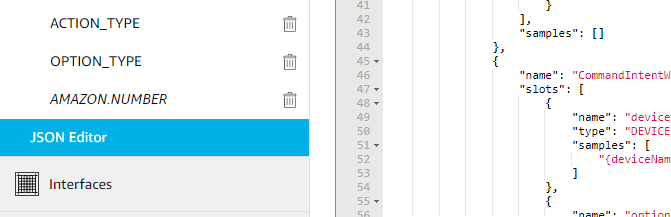


Nun können wir den Code einfügen der die Logik darstellt. 

Nun fragt man sich sicher: Schön dass ich alles eingerichtet habe, aber Code hab ich bis jetzt noch nicht gesehen. Wo kommt der her?

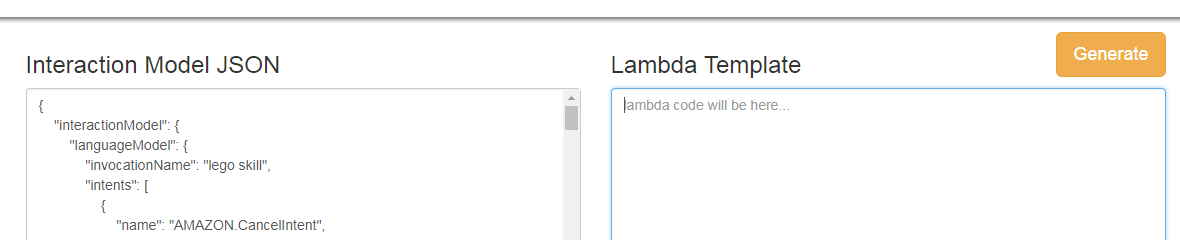
Der Code muss jetzt aus unserem Skill erst abgeleitet werden.

Hierzu wechselt man in die Alexa-Developer Seite zu dem erstellten Skill und wählt den Punkt „JSON Editor“ auf:



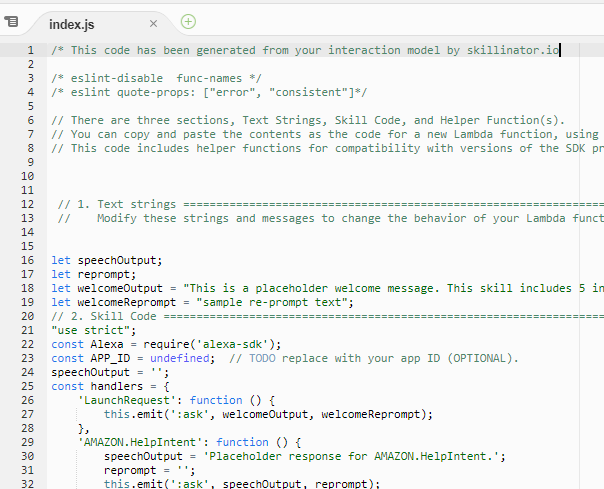
Den hier angezeigten Code kopiert man nun komplett in die Zwischenablage (z.B.: mittel STRG+A 🡪 STRG+C). Diesen kann man nun aber nicht einfach in der Lambdafunktion einfügen, da es sich hierbei um JSON-Code handelt und nicht um ein Lambda-Template.

**Um zum Lambda-Template zu kommen muss man zunächst diesen Code auf** [**www.skillinator.io**](http://www.skillinator.io) **auf der Seite „Interaction Model JSON“ einfügen.**



Anschliessend drückt man den „Generate“ Button und erhält den fertigen Lambda-Code. Diesen kopiert man nun abermals komplett und geht wieder auf die AWS-Lambdafunktions Seite.

Hier muss man nur noch den vorhandenen Beispielcode mit den nun generierten und zu unserem Skill passenden Code austauschen. Dies sollte dann ca. so aussehen:



**WICHTIG: Hier ist es nun wichtig rechts oben auf „Safe“ zu drücken, da sonst die Änderungen nicht übernommen werden.**

Grundsätzlich ist die Funktion jetzt operabel und kann ausgeführt werden. Jedoch stehen hier als „Reaktion“ auf die ankommenden Befehle ausschliesslich Beispiel-Rückmeldungen drinnen.

## Reaktionen der Lambdafunktion bearbeiten:

Da es relativ sinnfrei wäre nur Beispiel-Sprachreaktionen zu bekommen auf Befehle die man der Alexa gibt, werden diese auch noch geändert.

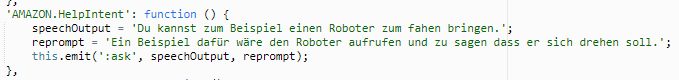
**HINWEIS: Es ist noch nicht so weit gekommen, echte Befehle an den Roboter zu übermitteln, weshalb hier daweil nur Sprachreaktionen eingefügt wurden!**

Zunächst kann man mal die Sprachausgabe bearbeiten, die kommt wenn man den Skill startet:



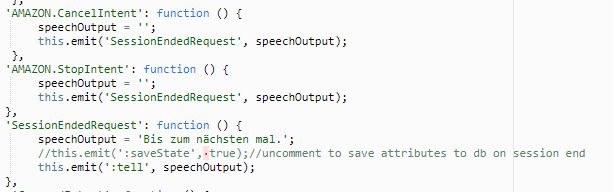
„welcomeRepromt“ steht hierbei dafür, dass wenn längere Zeit kein Befehl an die Alexa gegeben wird, sich diese noch einmal meldet, dass sie bereit ist einen Befehl entgegen zu nehmen.

Mit dem Befehl „Hilfe“ wird einem ein möglicher Befehl oder Aktion ausgegeben, die mit diesem Skill durchgeführt werden kann.



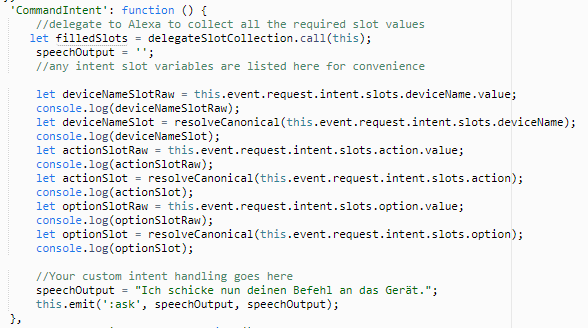
Bei der Option „:ask“ handelt es sich um eine Anweisung für die Alexa, nach der Ausgabe der Hilfe auf einen Befehl zu warten.

Wenn man nun den Skill beenden, unterbrechen oder einen Befehl stoppen will, haben wir uns zwischenzeitlich zum Test dafür entschieden, bei all diesen Befehlen den Skill zu beenden:



„:tell“ steht hierbei für eine einzige Sprachausgabe, die keinen weiteren Befehl erwartet. Die Unterfunktion „SessionEndedRequest“ wird hierbei auch beim Stoppen des Skills und bei Befehlsabbruch aufgerufen.

Die Reaktionen für die Befehle die im Skill definiert wurden sehen wie folgt aus:



Auch hier wird daweil nur eine Sprachausgabe ausgeführt und durch „:ask“ auf weitere Befehle gewartet.

## Weitere Ziele mit AWS:

Nachdem die grundsätzliche Befehlsstruktur vorhanden ist, haben wir nach Möglichkeiten gesucht die Befehle an den Roboter zu schicken.

Hierbei sind wir auf die Amazon Services SQS und SNS gestoßen. Diese stellen eine Art „Message-Queue“ zur Verfügung, auf die der Roboter zugreifen kann. Da wir allerdings uns hier noch nicht so eingearbeitet haben, stehen diese Punkte noch aus.