**ITP-Projektarbeit**

**Alexa trifft LEGO**

**-**

**Der Guide**

Patrick Loh BIF 1A1, Betreuer: Sabrina Rubenzer, MA

Sascha Potesil BIF 1A1, Betreuer: Sabrina Rubenzer, MA

David Dittmann BIF 1A2, Betreuer: Sabrina Rubenzer, MA

Benjamin Klaassen BIF 1A2, Betreuer: Sabrina Rubenzer, MA

**Vision des Projektes:**  
Ziel des Projektes ist die Steuerung eines Lego EV3 Roboters mit Alexa durch Sprachbefehle.

**Beschreibung des Projektes**  
Die Herausforderung die Sprachassistentin Alexa und einen LEGO Mindstorms EV3 miteinander zu verbinden, um der Frage wie gut ein Roboter auf Befehle von Menschen hören kann, ist Hauptaugenmerk dieses Projektes.

Ziel ist, die notwendigen Voraussetzungen zu erheben, zu dokumentieren und anschließend benutzerdefinierte Alexa Skills zu entwickeln und erste Befehle zu implementieren.

**Kurzinformation zu dem Guide:**

Dieser Guide soll eine Schritt-für-Schritt Anleitung für zukünftige Schüler und Studenten bieten um ebenfalls an diesem Projekt arbeiten zu können bzw. um ihren eigenen Alexa-gesteuerten Roboter zu erschaffen. Hier wird ausführlich jeder notwendige Schritt erklärt der dafür notwendig ist und technische Details sowie notwendiges Wissen erklärt. Von der Erstellung eines Amazon-Kontos bis hin zum Schreiben einer C#-Desktopapplikation. Hier steht alles was Sie dafür brauchen.

1. Amazon Web Service:
   1. 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.

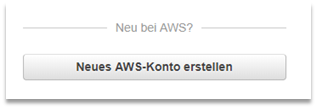
* 1. Einrichten von AWS:

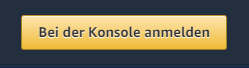
Um die Funktionen von AWS nutzen zu können sind folgende Sachen notwendig:

* Einrichtung eines Kontos auf [https://aws.amazon.com](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.

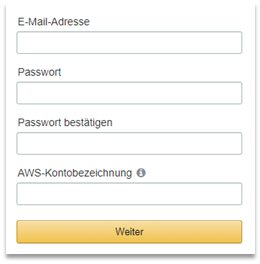
**ANLEGEN EINES KONTOS:**

Zum Anlegen eines AWS-Entwicklungskontos muss man sich zunächst unter [https://aws.amazon.com](https://aws.amazon.com/) ein Konto erstellen.

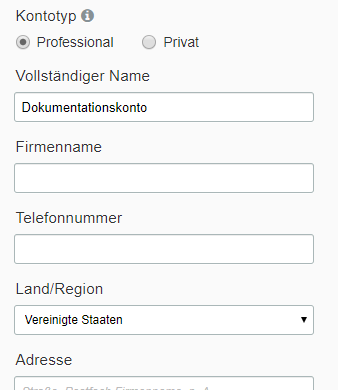


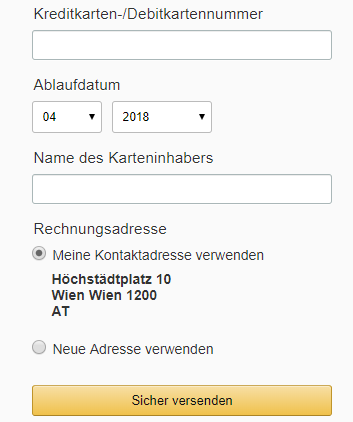
[Symbol]

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



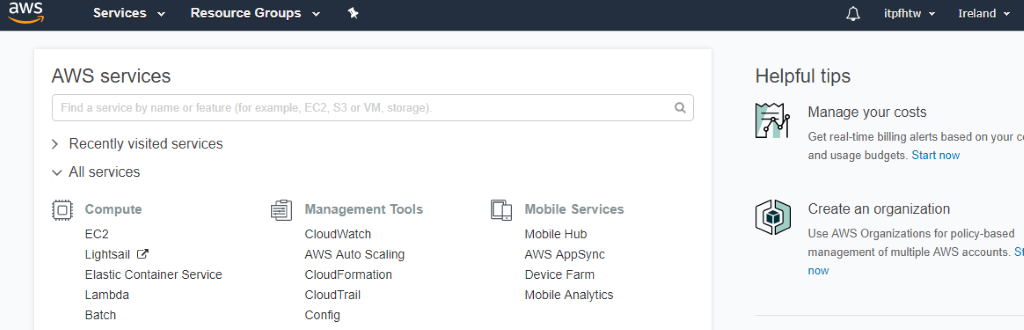
Nun werden von einem die Kontaktinformation benötigt, wobei diese alle verpflichtende Angaben sind.

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:

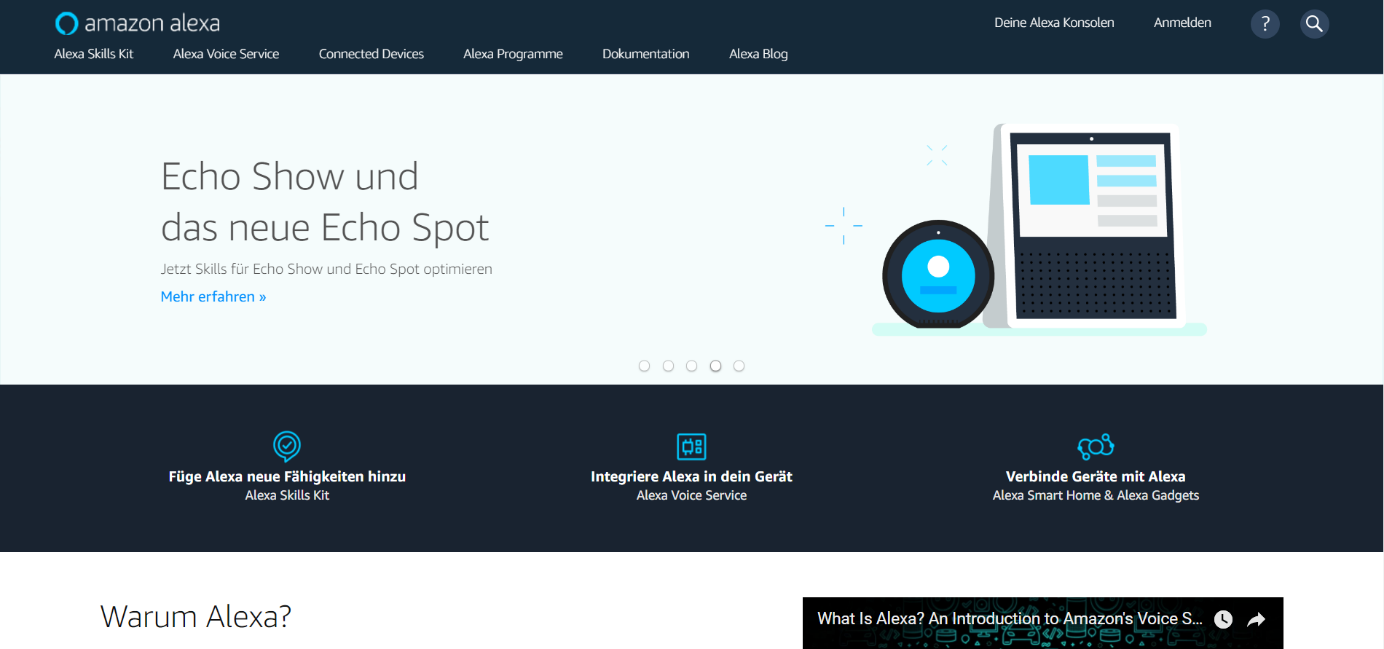


# Skills

* 1. Den ersten Skill anlegen

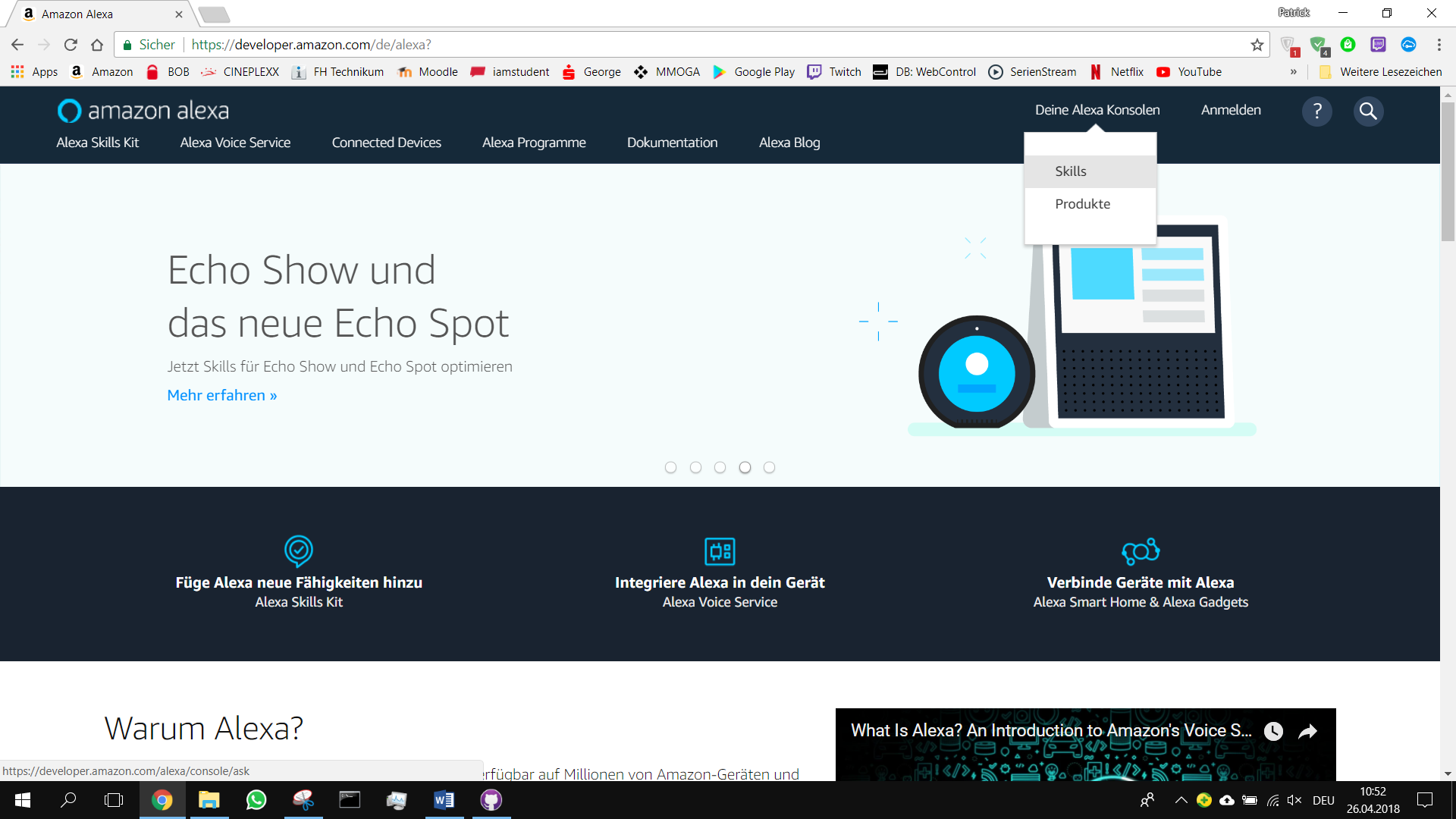
Ist man auf der Developer Seite von Amazon, muss zunächst auf „[amazon alexa](https://developer.amazon.com/de/alexa)“ geklickt werden um zu der Skill Entwicklung zu gelangen.

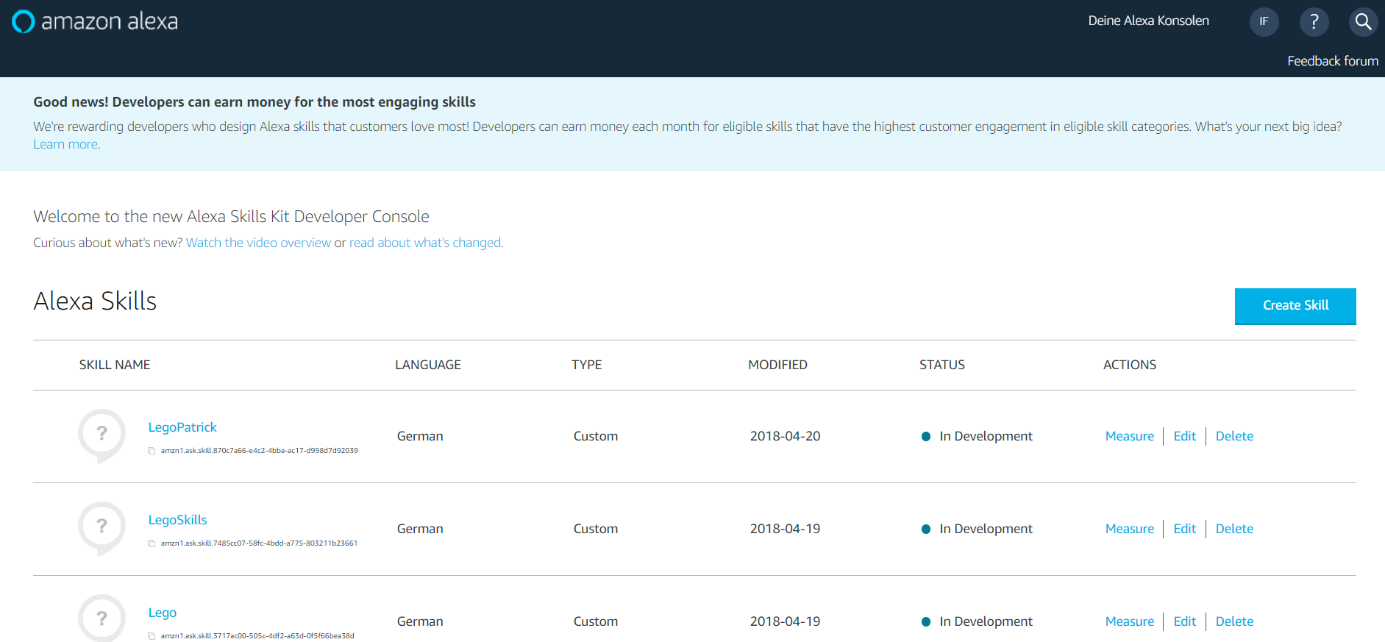


Das Design wurde von Amazon weitestgehend verändert, was Website und Amazon Skill Kit betrifft. Somit gibt es kaum Anleitungen zum aktuellen Design (Stand 26.014.2018).

Hier gelangt man zu den Skills.

Hier erhält man Informationen zur Skill Entwicklung.

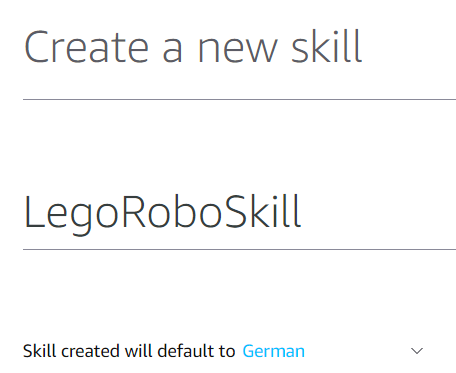




Ist man im Skill Kit so sollten hier eure Skins aufscheinen. Da ja noch kein Skill entwickelt wurde wird die Liste leer sein. In unserem Fall gibt es ja schon Skills, deshalb sieht man hier auch „LegoPatrick“, „LegoSkills“ und „Lego“.

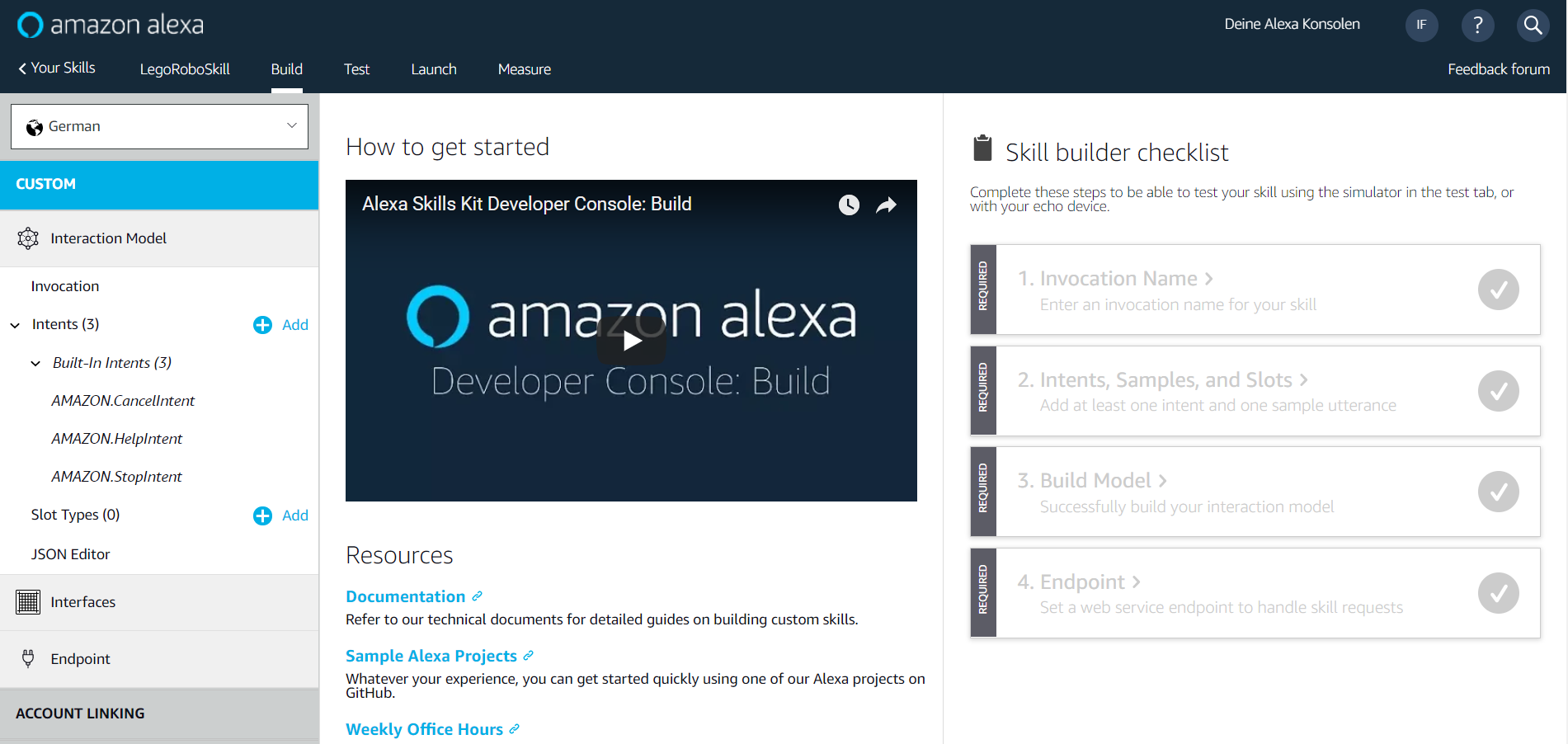
Um jetzt einen Skill zu erstellen klickt man auf „Create Skill“.

Man wird aufgefordert einen Namen einzugeben & die Sprache muss natürlich auch ausgewählt werden (in unserem Fall – German). Wir entscheiden uns für den Namen „LegoRoboSkill“.

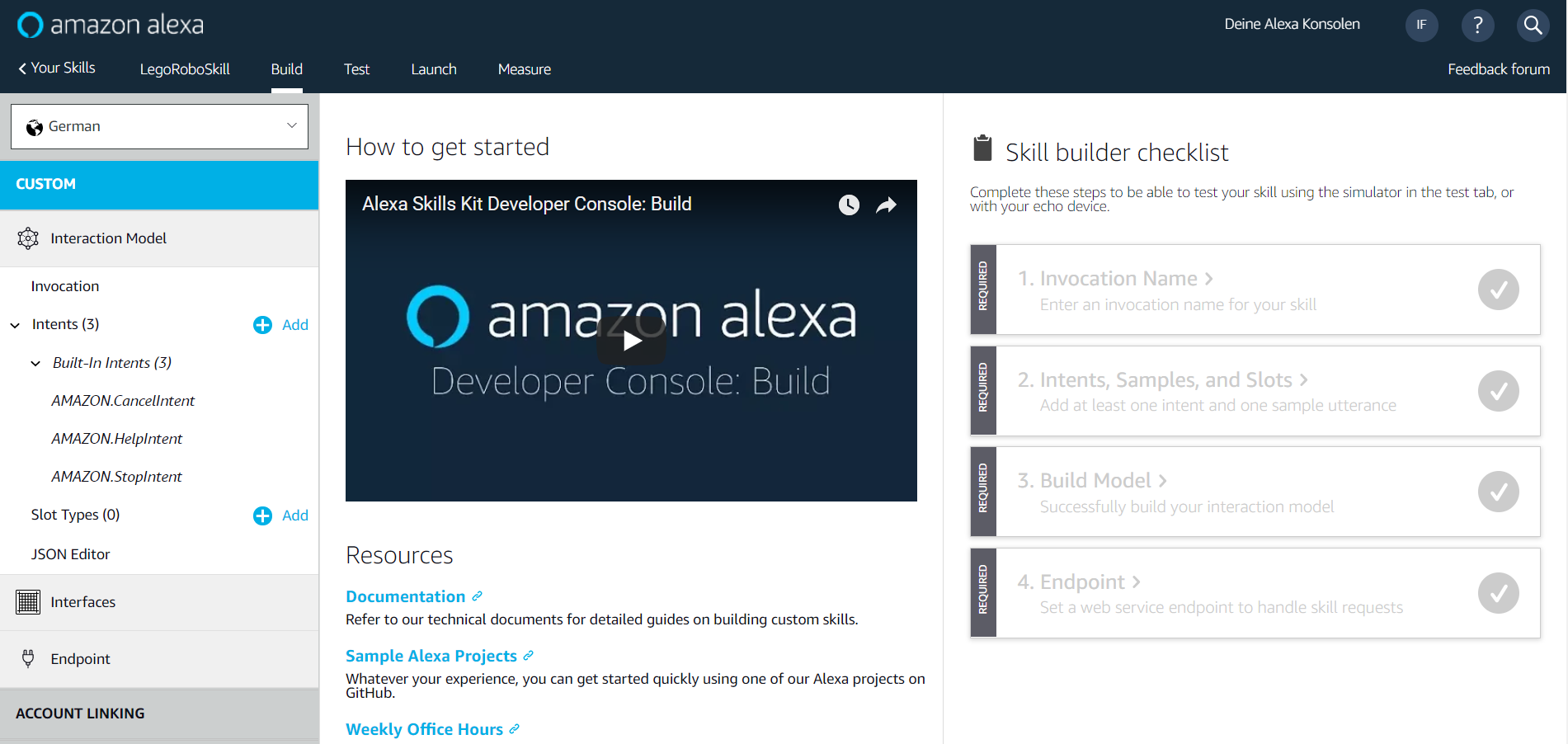


Im nächsten Schritt wählen wir aus, wofür unser Skill entwickelt werden soll. Da wir kein Smart Home oder dergleichen, sondern einen komplett eignen Skill entwickeln wollen muss „custom“ ausgewählt werden. Danach wird der Skill auch schon erstellt.

# Das Skill Kit Interface

Wie vorhin bereits erwähnt hat Amazon seine Benutzeroberfläche komplett überarbeitet und sieht nun folgendermaßen aus:

Zuerst wirkt die Seite sehr verwirrend, ist sie aber gar nicht. Sie ist eingeteilt in:  
  
*Build* (hier wird der aktuelle Skill entwickelt und ist auch der Reiter in dem wir uns am meisten bzw. fast immer aufhalten),   
  
*Test* (hier kann der Skill getestet werden, ist aber erst in ein paar Schritten wirklich nützlich, da zuerst ein Endpunkt zu AWS angelegt werden muss),   
  
*Lauch* & *Measure* sind nur wichtig falls der Skill veröffentlicht und allen zur Verfügung gestellt werden soll und da wir dies nicht vor haben ist es auch nicht wichtig.

Zurück zu dem Reiter ***Build***. Ich gebe euch einen kleinen Überblick wie die Seite aufgebaut ist, aber im Punkt 4, wird alles genau im Detail erklärt.

Wie der Name schon verrät werden hier die Endpoints hinzugefügt. (z.B.: AWS-Lambda)

Hier können Änderungen vorgenommen werden. (für uns unwichtig)

Die JSON Datei des Skills, welche auch bearbeitet werden kann.

Dadurch werden Slot Types(eine Art Variablen mit versch. Werten) hinzugefügt.

Hier können so genannte Intents(Befehle) hinzugefügt werden.

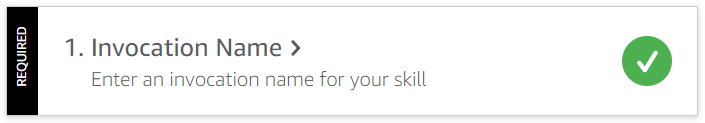
Hier wird der Invocation Name(Aufrufname) eingetragen.

# Den ersten Skill entwickeln

Auf der rechten Seite sieht man die **Skill Builder Checklist**, welche wir jetzt abarbeiten werden, um unseren Skill fertig stellen zu können.  
Als erster Punkt steht auf der Liste der Invocation Name. Das ist nichts anderes als der Name der Alexa gesagt werden muss, um das Programm bzw. den Skill zu starten. Auf der linken Seite gibt es den Punkt „Invocation“ und wenn wir darauf klicken werden wir aufgefordert diesen Namen eben einzugeben. In unserem Fall geben wir „lego skill“ ein. Der Name sollte leicht zu verstehen sein, damit Alexa es eindeutig identifizieren kann.  
Jedes Mal, wenn wir eine Änderung vornehmen sollte gespeichert werden, unter „Save Model“.

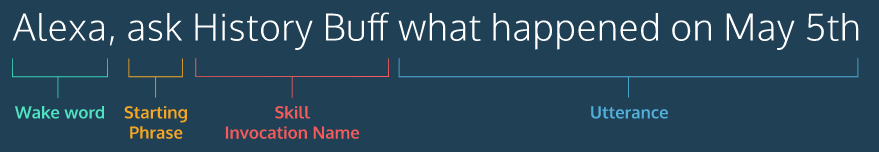


Gehen wir zurück sollte bei der Checklist ein grüner Haken sein:

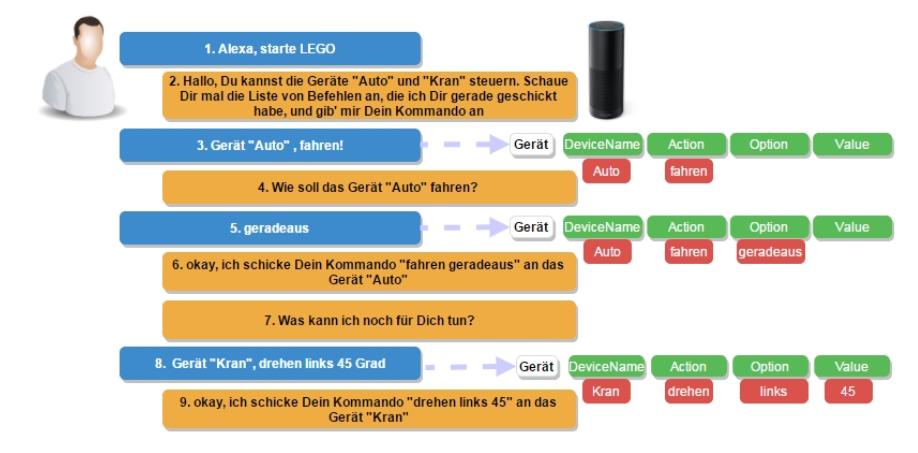


Als nächstes sollen wir Intents bzw. Utterances und Slots erstellen. Jetzt ist aber die Frage was sind diese Intents und Slots?  
Wiedermal klingt es komplizierter als es eigentlich ist.   
Intents oder Utterances sind sozusagen Befehle die Alexa von uns als Person erwartet und Slots sind Variablen die mehrere Werte haben können und diese wir ihnen auch zuweisen. Bevor man einen Skill entwickelt solle man sicher sein, was Alexa für mögliche Befehle erwarten kann, da dies wichtig ist. Am besten wir machen uns das Prinzip „learning per doing“ zu nutze.

Wir haben uns nach der Anleitung von [Vadim Kantor und Boris Kantor](https://jaxenter.de/alexa-skills-entwickeln-64751) orientiert.

Unser Alexa soll zwei Arten von Befehle verarbeiten können und zwar „Gerät Roboter fahre gerade aus“ und „Gerät Roboter drehe um 45 Grad“. Das bedeutet wir brauchen 2 Utterances und zwar einen ohne einer Grad Angabe und einen mit. Hier ein Beispiel um es zu veranschaulichen:

Hier wird der Skill mit dem Namen „History Buff“ aufgerufen und ein Utterance erwartet. Und zwar wird jetzt der Utterance „what happend on may 5th“ aufgerufen und an AWS-Lambda weitergegeben. Später dazu mehr.

Noch ein Beispiel, wie der Skill mit unseren Lego Robotern aussehen könnte:

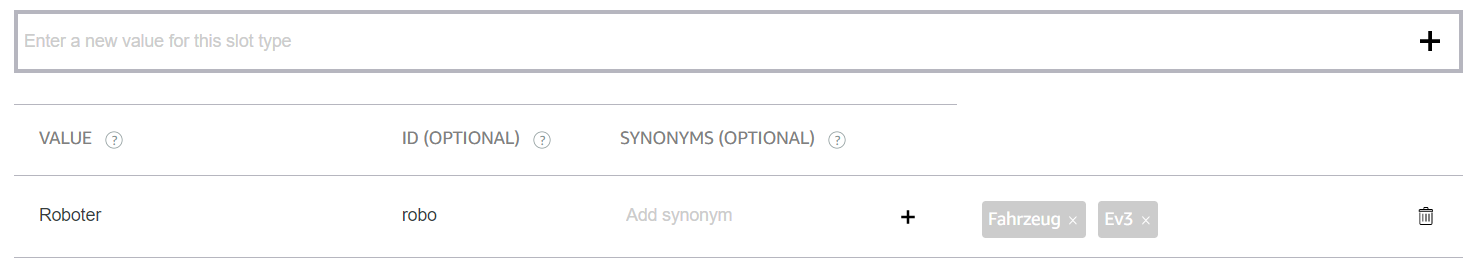
Hier sieht man auch schön wie Slot Types funktionieren. Nehmen wir unser obiges Beispiel von dem Utterance her: „Gerät Roboter fahre gerade aus“. Mit *Gerät* wird einfach unser Utterance gestartet. Danach wird der Slot Type „DeviceName“ erwartet welcher vorher definiert werden muss. Deshalb ist es wichtig vorher zu wissen wie der Alexa Skill aufgebaut werden soll. Zurück zum *DeviceName.* Der Slot Typ sollte hier auf jeden Fall *Roboter* gespeichert haben (sonst erkennt er diesen Befehl nicht) und wird ausgewählt und bei *Action,* wird *fahren* ausgewählt, usw..

## Slot Types

So da jetzt nicht mehr so unklar ist was Slot-Types sind, können wir diese nun erstellen.

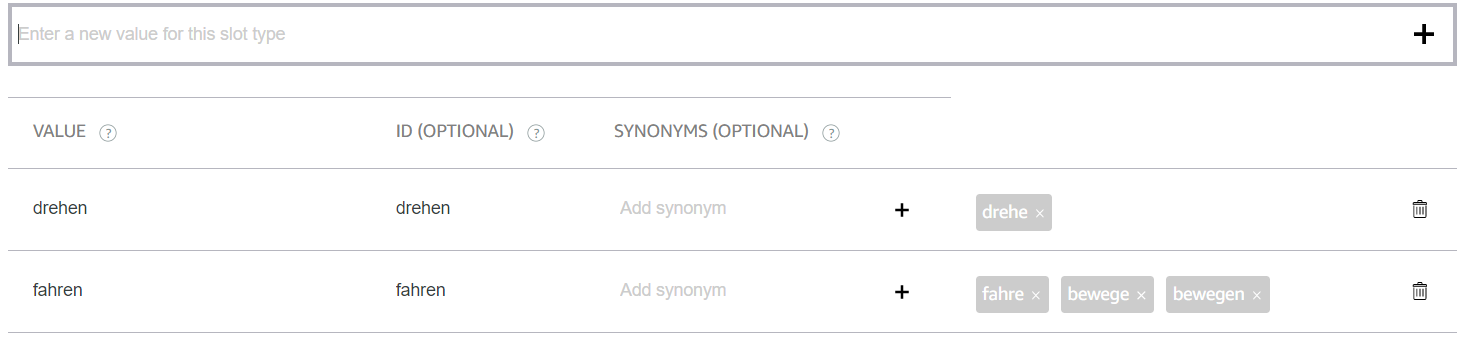
Und zwar brauchen wir 3 Slot Types. Einen für unseren Gerätenamen (*DeviceName*), einen für die Funktion was er tun soll (*Action*) und einen wie er diese Funktion ausführen soll (*Option*). Für die Grad Angabe kann der vorgefertigte Type -> *Amazon.Number* verwendet werden.

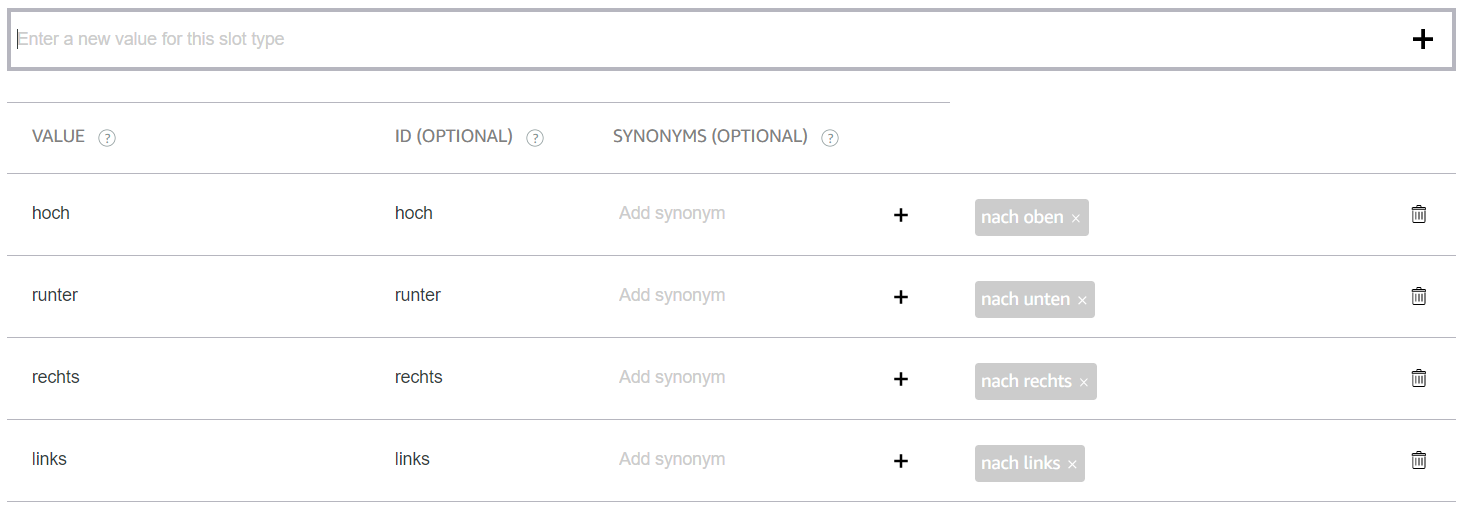
Ist *Device\_Name\_Type* erstellt fügen wir noch eine ID und Synonyme hinzu. Es sollte dann folgendermaßen aussehen:



Unser Roboter kann also über *Roboter*, *Fahrzeug* und *Ev3* aufgerufen werden.

Das gleiche machen wir nun für *Action\_Type*:

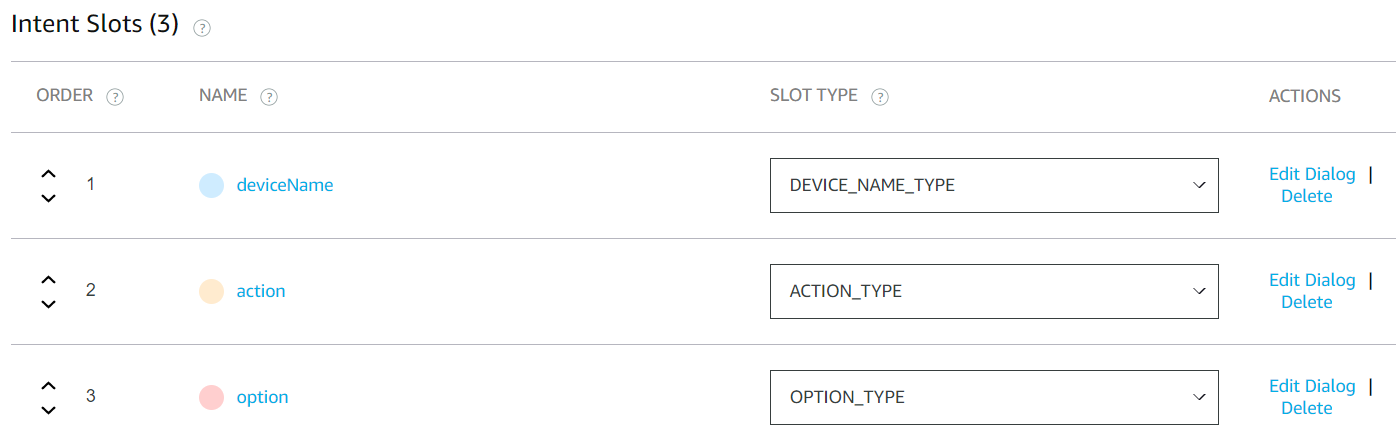
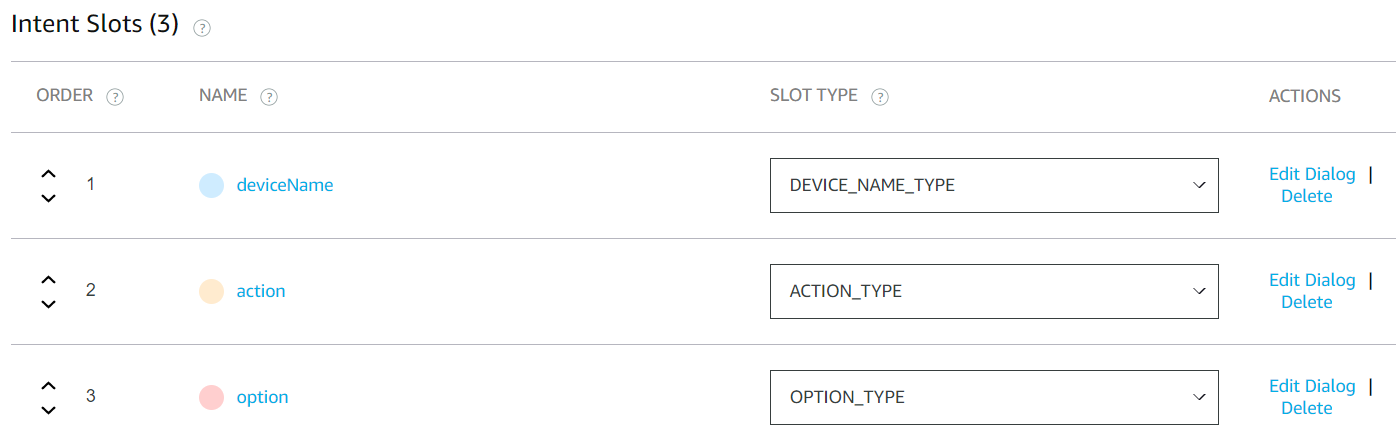


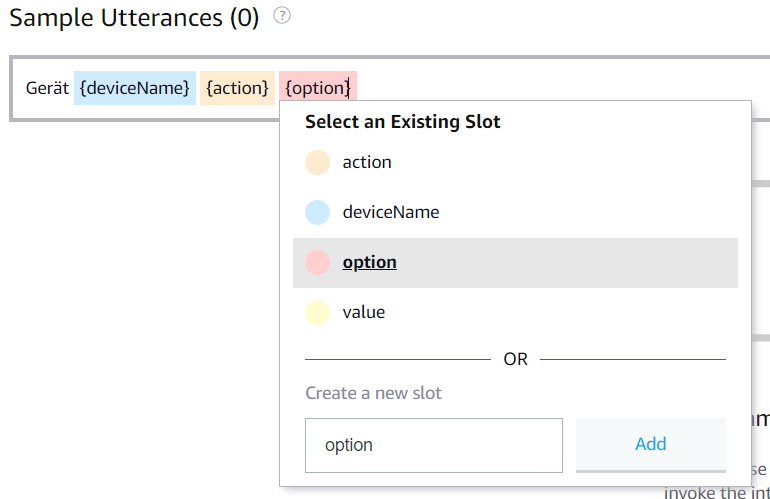
und *Option\_Type*:

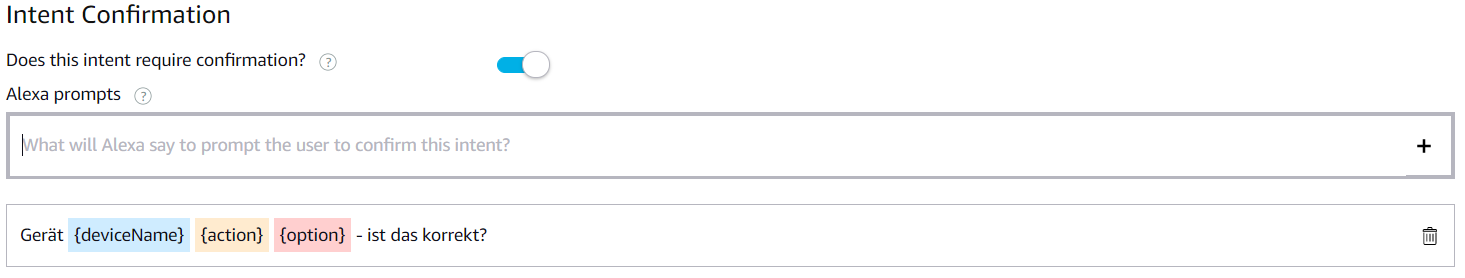
## Intents und Utterance

Wir fügen nun unseren Intent ohne Gradangabe hinzu und nennen diesen „CommandIntent“.

Als erstes müssen wir unsere Slot-Types in die Intent Slots eintragen. Wir nennen dieses Slots so, wie wir sie in unserem Utterance aufrufen wollen.

Jetzt können wir unser Utterance erstellen. Die Intent Slots werden mit einer geschwungenen Klammer aufgerufen:



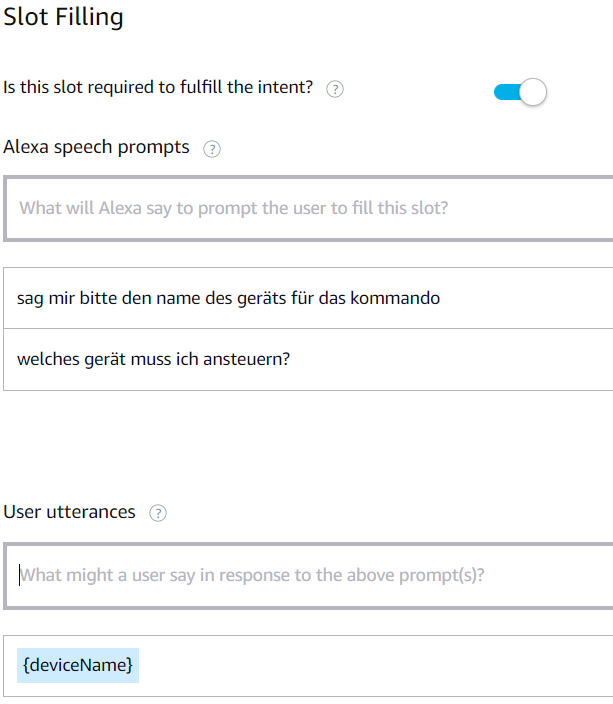
Jetzt lassen wir Alexa den verstandenen Input wiederholen und vom User bestätigen. Das Funktioniert über den Intent Confirmation.

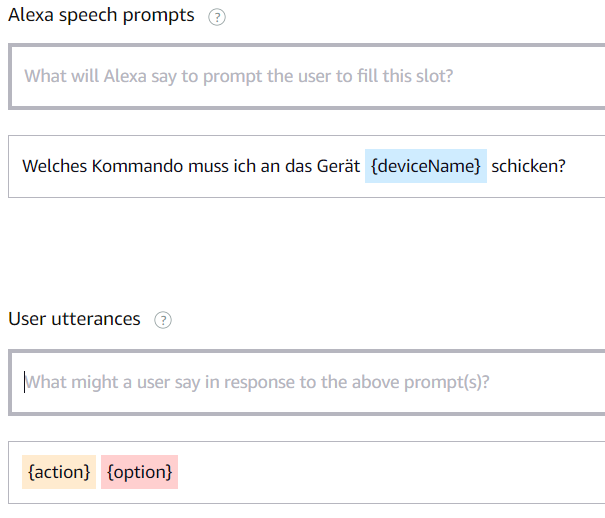
Natürlich kann es sein das ein User etwas falsch sagt oder vergisst. Somit müssen wir manche Slot Types auf required setzten und dazu weitere Einstellungen treffen.

Beginnen wir mit den *DeviceName*:

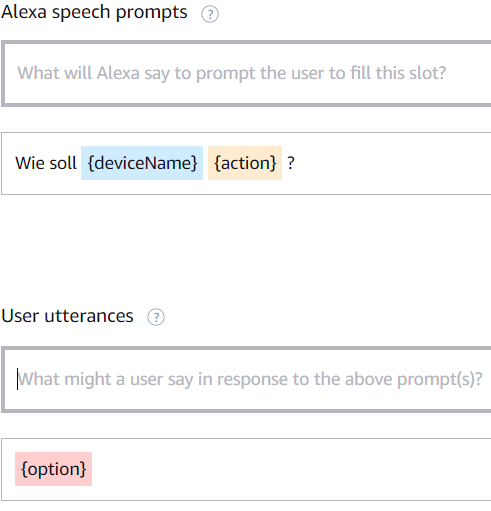
Alexa soll ja darauf regieren, wenn der User etwas falsch getan hat. Somit soll Sie ihm mitteilen an welches Gerät, sprich den Gerätenamen wir etwas übermitteln wollen.

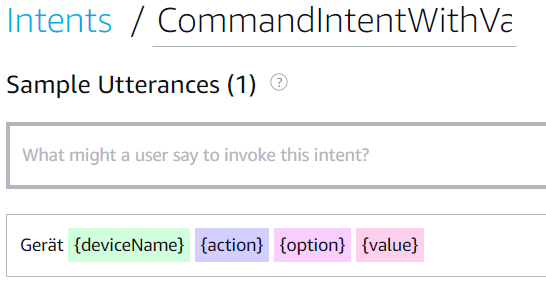
Außerdem soll die Antwort des Users wieder in einen Slot Type gespeichert werden. Dies sollte dann folgendermaßen aussehen:



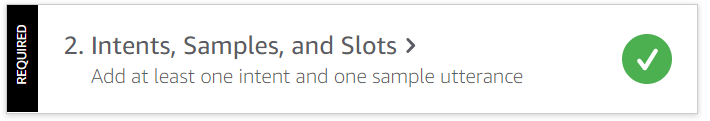
Das gleiche wiedermal für *Action*:

Und *Option*:

In diesem Utterance können nur Befehle entgegen genommen werden ohne eine präzise Drehung mit einer Grad Angabe. Somit muss der Schritt 4.2. noch einmal wiederholt werden nur mit einem Utterance der auch eine Gradangabe erwartet. Alexa erkennt dieses und weiß welchen Utterance sie verwenden soll.

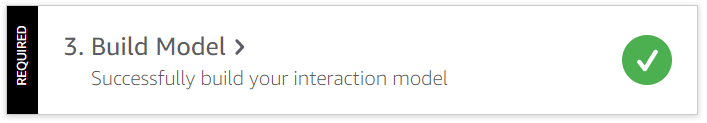


Da wir mit Intents, Utterance und Slot Types fertig sind sollte ein grünes Häkchen in der Checklist sein.



Der nächste Punkt auf der Checklist ist einfach das Projekt mal zu builden, über den Button „Build Model“.



Sind keine Fehler passiert sollte auch hier ein grüner Haken sein.

## Endpoints

Bei Endpoints wird eine Verbindung zum AWS von Amazon hergestellt. Um genau zu sein zu [Lambda](https://aws.amazon.com/de/lambda/features/).

Um zu verstehen was Lambda genau ist, zitiere ich hier dessen Beschreibung:

„AWS Lambda ist ein serverloser Datenverarbeitungsservice, der Ihren Code beim Eintreten bestimmter Ereignisse ausführt und automatisch für Sie die zugrunde liegenden Datenverarbeitungsressourcen verwaltet. Mit AWS Lambda können Sie andere AWS-Services mit benutzerdefinierter Logik erweitern oder Ihre eigenen Backend-Services erstellen und im AWS-Umfang und mit AWS-Leistung und -Sicherheit betreiben. AWS Lambda kann Code automatisch als Reaktion auf mehrere Ereignisse wie HTTP-Anforderungen über Amazon API Gateway, Änderungen von Objekten in Amazon S3-Buckets, Aktualisierungen von Tabellen in Amazon DynamoDB oder Zustandsübergänge in AWS Step Functions ausführen.

Lambda führt Ihren Code automatisch auf hochverfügbarer Datenverarbeitungsinfrastruktur aus und erledigt die gesamte Administration der Datenverarbeitungsressourcen, einschließlich Server- und Betriebssystemwartung, Kapazitätsbereitstellung und automatischer Skalierung, Code- und Sicherheitspatch-Bereitstellung sowie Code-Überwachung und -Protokollierung. Sie müssen nur den Code zur Verfügung stellen.“

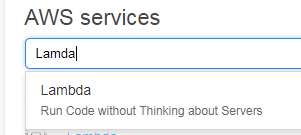
# Arbeiten mit AWS:

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

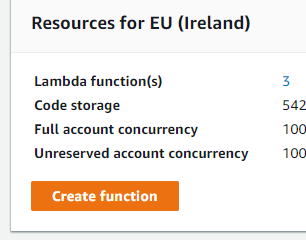
Hierfür werden die sogenannten LAMBDA-Funktionen verwendet.

**ANLEGEN EINES LAMBDA BACKEND:**

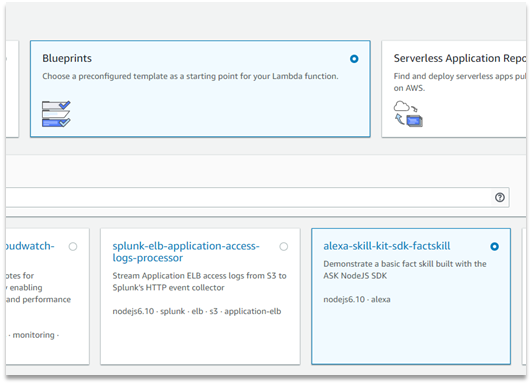
Auf der Startseite von AWS findet man alle möglichen Arten von Services. Um zu den Lambdafunktionen zu kommen sucht man diese zunächst in der Suchleiste und wählt sie aus:



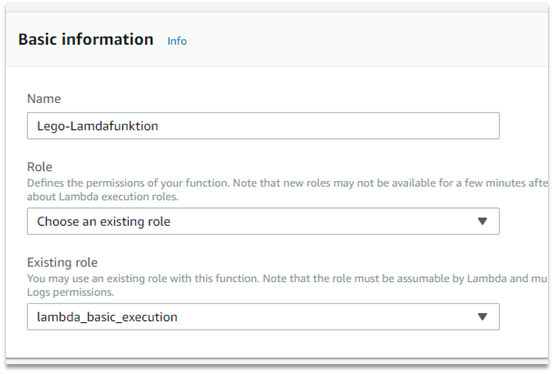
Danach muss man eine neue Lamdafunktion erstellen:



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.

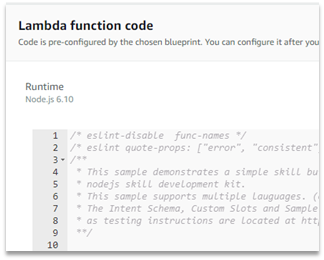


Nun muss man einige Angaben zu der Lambdafunktion machen:



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.

* 1. 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",

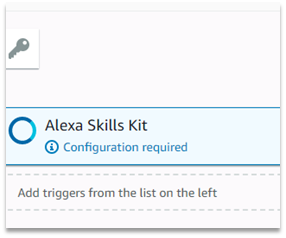
sessionAttributes: sessionAttributes,

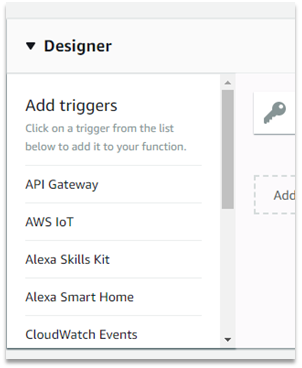
response: speechletResponse

};

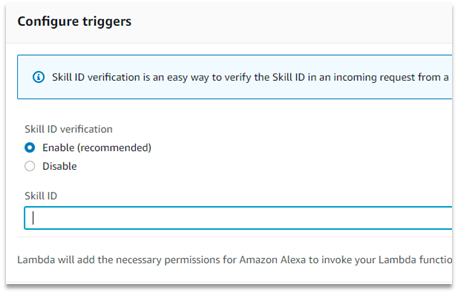
}

* 1. Verbinden der Lambdafunktion mit dem Alexa-Skill:

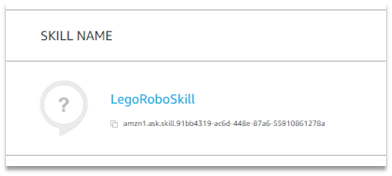
Zunächst muss die Lambdafunktion mit unserem Skillkit verbunden werden. Hierführ wählt man bei Designer das „Alexa Skills Kit“ aus.



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:

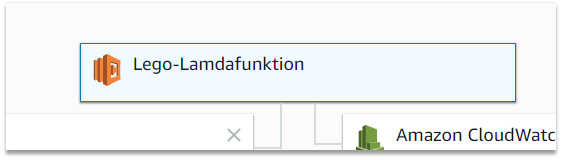
Mittels klick auf das Kopiersymbol kann die Skill-ID per einfachen Mausklick kopiert werden.

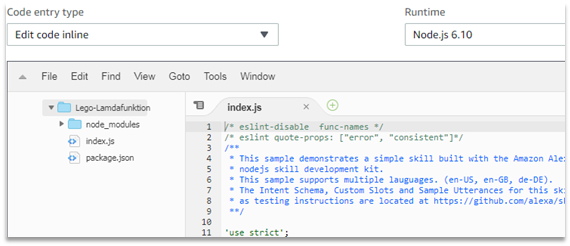
Diese muss nun bei der Lambdafunktion eingetragen werden.

**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:

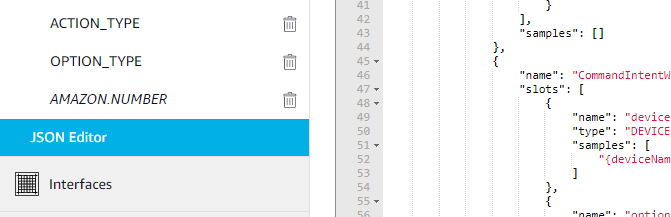


Nun können wir den Code einfügen der die Logik darstellt. 

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

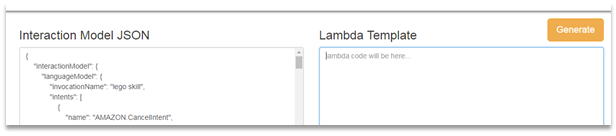
Der Code muss jetzt aus unserem Skill erst abgeleitet werden.

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



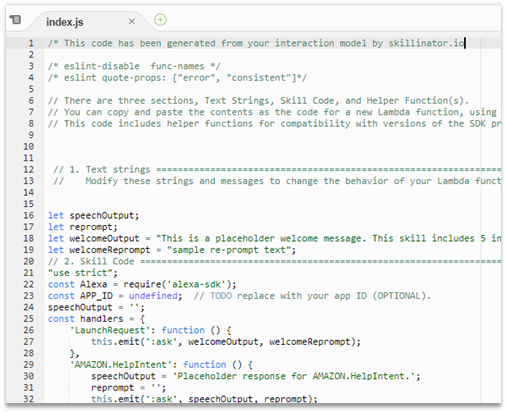
Den hier angezeigten Code kopiert man nun komplett in die Zwischenablage (z.B.: mittel STRG+A[Symbol] STRG+C). Diesen kann man nun aber nicht einfach in der Lambdafunktion einfügen, da es sich hierbei um JSON-Code handelt und nicht um ein Lambda-Template.

**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 „Save“ zu drücken, da sonst die Änderungen nicht übernommen werden.**

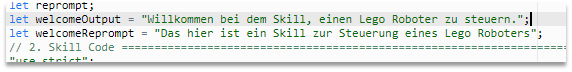
Grundsätzlich ist die Funktion jetzt operabel und kann ausgeführt werden. Jedoch stehen hier als „Reaktion“ auf die ankommenden Befehle ausschliesslich Beispiel-Rückmeldungen drinnen.

* 1. 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.

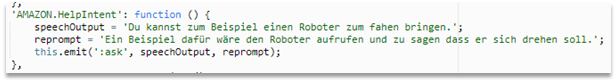
**HINWEIS: Es ist noch nicht so weit gekommen, echte Befehle an den Roboter zu übermitteln, weshalb hier daweil nur Sprachreaktionen eingefügt wurden!**

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



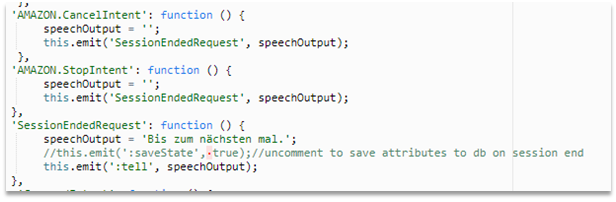
„welcomeRepromt“ steht hierbei dafür, dass wenn längere Zeit kein Befehl an die Alexa gegeben wird, sich diese noch einmal meldet, dass sie bereit ist einen Befehl entgegen zu nehmen.

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



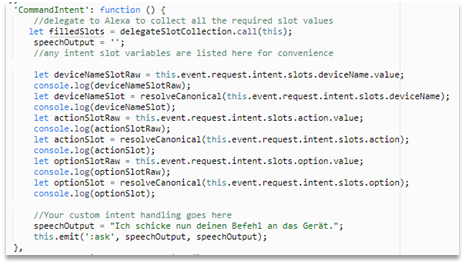
Bei der Option „:ask“ handelt es sich um eine Anweisung für die Alexa, nach der Ausgabe der Hilfe auf einen Befehl zu warten.

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:



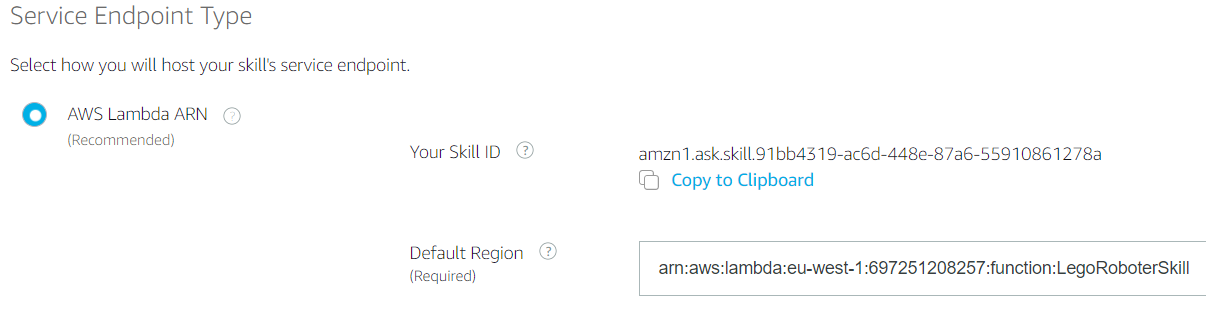
„:tell“ steht hierbei für eine einzige Sprachausgabe, die keinen weiteren Befehl erwartet. Die Unterfunktion „SessionEndedRequest“ wird hierbei auch beim Stoppen des Skills und bei Befehlsabbruch aufgerufen.

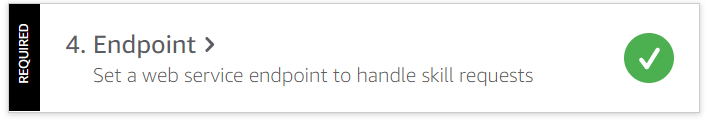
Die Reaktionen für die Befehle die im Skill definiert wurden sehen wie folgt aus:



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

Hat man die Lambda Funktion fertig muss nun dessen ARN bei Default-Region eingefügt werden.

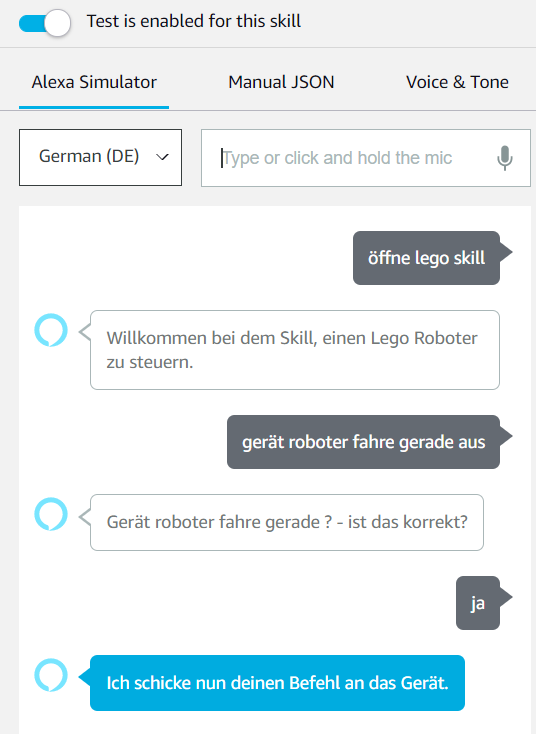




Kann der Endpoint gespeichert werden sind keine Fehler aufgetreten und es kann getestet werden.

Grundlegend gibt es 2 gute Möglichkeiten seinen Skill zu testen. Einerseits [Echosim.io](https://echosim.io/welcome) oder direkt in dem Alexa Skill Kit.

Wir Testen in Alexa Skill Kit, da es uns mehr Informationen gibt und auch über einen Text angesprochen werden kann.   
Ein Skill wird mit „öffne“ oder „starte“ + Invocation Name gestartet.



Hier sieht man wie ein möglicher Aufruf aussehen könnte.

* 1. 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.

1. SNS / SQS
   1. Was ist SNS / SQS?

SNS und SQS sind Amazon Webservices. SNS steht für „Simple Notification Service” und ermöglicht die Verteilung von Nachrichten von einem Sender an unterschiedlichste Empfänger (Email, SQS, …).

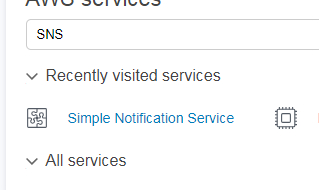
SQS steht für „Simple Queue Service“ und ist ein Message-Queue Webservice, über diesem Nachrichten Empfangen werden können.

* 1. SNS:

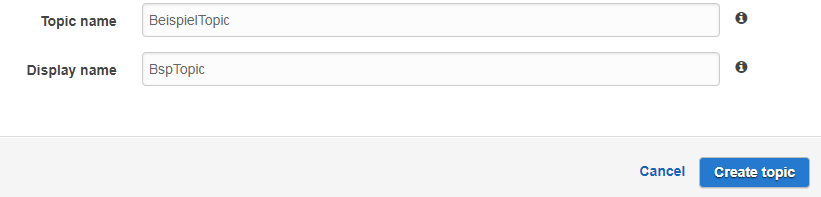
Wir benutzen SNS um die Befehle die von unserem Lambda Service ausgesendet werden an unsere SQS weiter zu leiten. Weiters bietet SNS die Möglichkeit die Befehle als Email zu verschicken, was eine gewisse Art von Debugging ermöglicht, indem man sehen kann, was der Lambda Service aussendet.

* + 1. Konfiguration eines SNS:

Um einen SNS einzurichten muss man zunächst einen neuen bei den Amazon Webservices anlegen.

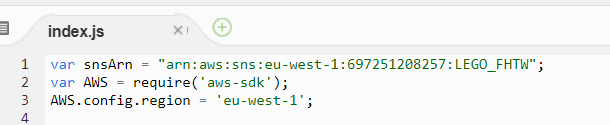


Nun muss man ein sogenanntes „Topic“ anlegen. Das entspricht einem Verteilernamen, um so den Verteilservice von SNS genau bestimmen zu können.

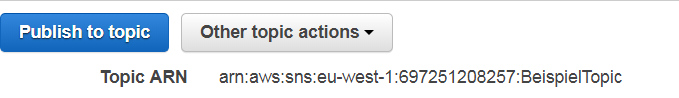


Jetzt hat man Grundsätzlich schon den SNS angelegt. Was man nur noch machen muss, ist dem SNS Nachrichten senden und Empfänger einrichten.

Um an den SNS Nachrichten senden zu können, muss man im Lambda-Code folgende Zeile editieren:

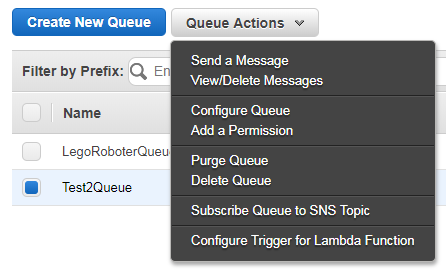


Unter snsArn muss der SNS-Service angegeben werden. Die hierfür benötigte ARN findet man direkt bei der Topic Konfiguration:

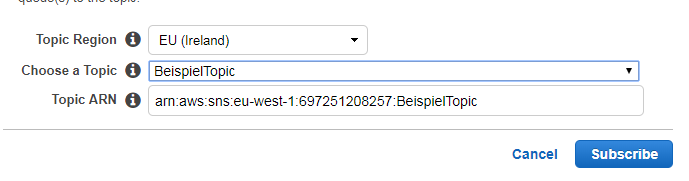


Da nun der Verteilservice Nachrichten bekommt, sollen diese auch weitergeleitet werden. Hierfür muss man sogenannte „Subscribtions“ anlegen.

Hierfür wechselt man in die Konfiguration von der SQS-Message Queue (Zu dieser kommen wir später genauer) und wählen unter den Queue-Actions die Option „Subscribe Queue to SNS Topic“.



Hier kann man nun seine vorher erstellte Queue als Empfänger bei SNS eintragen:

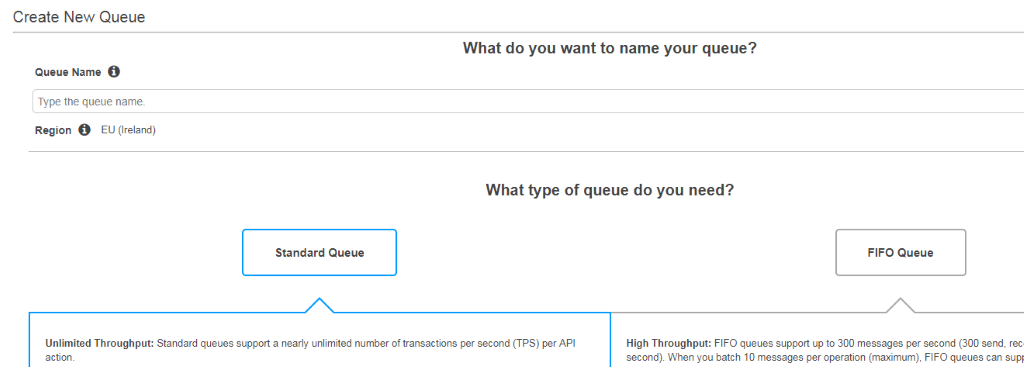


* 1. SQS:

SQS wird wie zuvor erwähnt bei uns als Message-Queue Service benutzt. Dies bietet uns die Möglichkeit die Befehle die von unserem Lambda Service kommen hier rein zu laden und sie nacheinander mittels des Desktop-Programmes von dort abzuholen. Und somit weiter an den Roboter zu senden.

* + 1. Anlegen einer neuen SQS:

Um eine neue SQS anzulegen klickt man zunächst auf den „Create New Queue“ Button. Dort bekommt man dann folgendes Fenster angezeigt:



Unter Queue Name kann man frei einen gewünschten Namen eintragen. Dieser ist nicht von weiterer Bedeutung.

Zur Auswahl stehen folgende Queues:

* 1. Standard Queue:

Diese Message Queue ist sehr performant und kann viele Nachrichten auf einmal bekommen ohne Probleme, jedoch garantiert sie nicht, das die Befehle in richtiger Reihenfolge wieder raus gehen, wie sie rein gekommen sind. Sie ist ursprünglich nur für einen hohen Datendurchsatz gedacht.

* 1. FIFO Queue:

Diese Queue ist langsamer als die standard-Version, garantiert dafür aber das FIFO (First In First Out) Prinzip bei der Nachrichtenabholung.

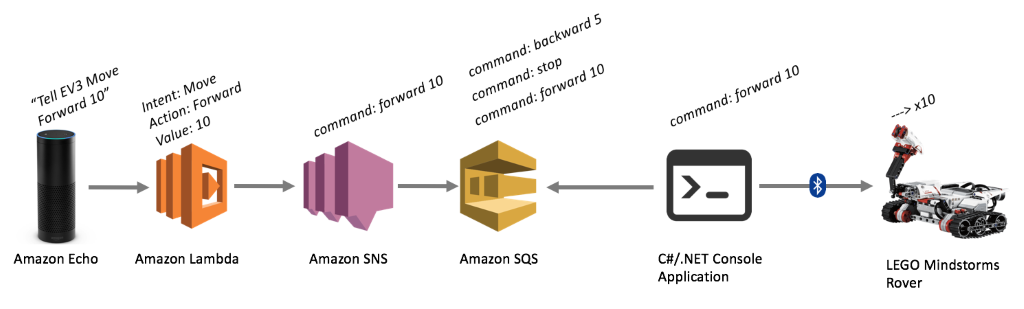
Wir haben uns hier für die Standard Queue entschieden, da die FIFO auch nach mehrmaligen Versuchen die Nachrichten nicht richtig verarbeitet. Dies ist ein bekanntes Problem bei dieser Message Queue, da sie nicht immer die Nachrichten weitergeleitet oder freigegeben hat.

Im Grunde braucht man nur wie unter SNS beschrieben die MQ an den SNS-Service einschreiben und die Nachrichten werden automatisch in die MQ geladen. Geholt werden sie nach dem Pull-Prinzip von der Desktop-Anwendung. Eine SQS kann niemals per Push-Prinzip Nachrichten weiterleiten.

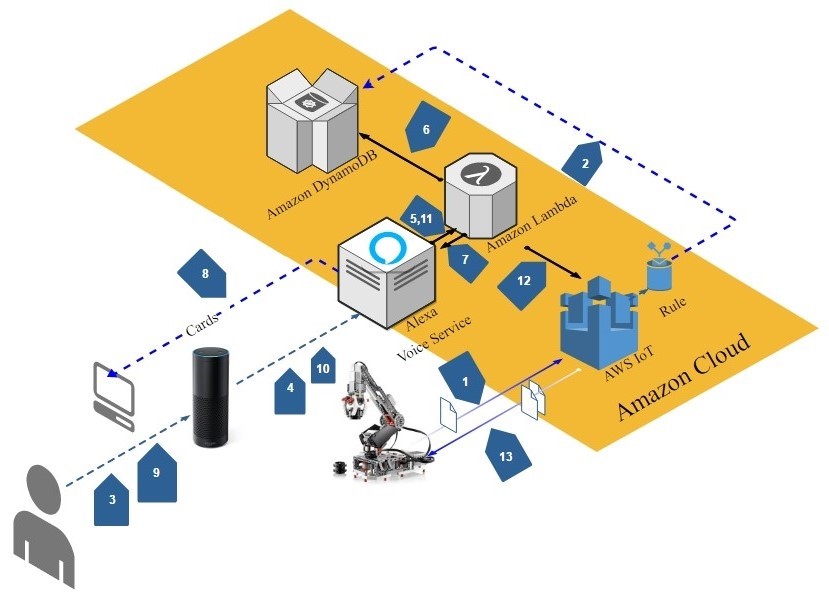
1. Wie man den Roboter mit Alexa verbindet
   1. Wahl der Verbindung

Ein kritischer Teil unseres Projekts ist die Anbindung von Alexa, die Amazon AWS Lambda als Schnittstelle verwendet mit einem Lego EV3 Roboter zu verbinden. Durch recherchieren haben wir 2 Lösungswege finden können, die uns vielversprechend erschienen.

* + 1. Anbindung via Desktop mit Bluetooth

Bei diesem Lösungsansatz haben wir uns Orientiert mittels der [Blog-Posts von Jim Drewes](http://blog.jimdrewes.com/controlling-a-mindstorms-ev3-with-amazon-echo/). Er hat die Verbindung ermöglicht mittels SQS Queue. Der Service ermöglicht es eine Queue von Befehlen abzuspeichern. Die Queue bekommt Kommandos von Alexa über Amazon Lambda und stellt sie zum Abrufen bereit. Die Kommandos werden durch eine C#/.NET-Applikation an einem Desktop abgerufen. Diese Applikation übersetzt sie und sendet sie über Bluetooth weiter an den Roboter. Für diese Möglichkeit hat sich unser Team entschlossen.

* + 1. Anbindung via Amazon IOT



Ein weiterer Weg ist die Verwendung vom Amazon IOT Service und einer speziellen Linux Distribution für Lego EV3.

Amazon stellt für die Anbindung von IOT-Geräten den [Amazon IOT Service](https://aws.amazon.com/iot/) zur Verfügung. Er besteht aus Kits für verschiedene Programmiersprachen, worunter C und C++, mit denen man mit diversen Amazon Diensten kommunizieren kann.

In unserem Fall wird eine Verbindung zu AWS Lambda hergestellt, die mit Alexa in Verbindung steht. Da wir uns nicht für diese Variante entschieden haben, wird auf diese in dem Nachfolgenden (nach 5.2.) nicht weiter erläutert.

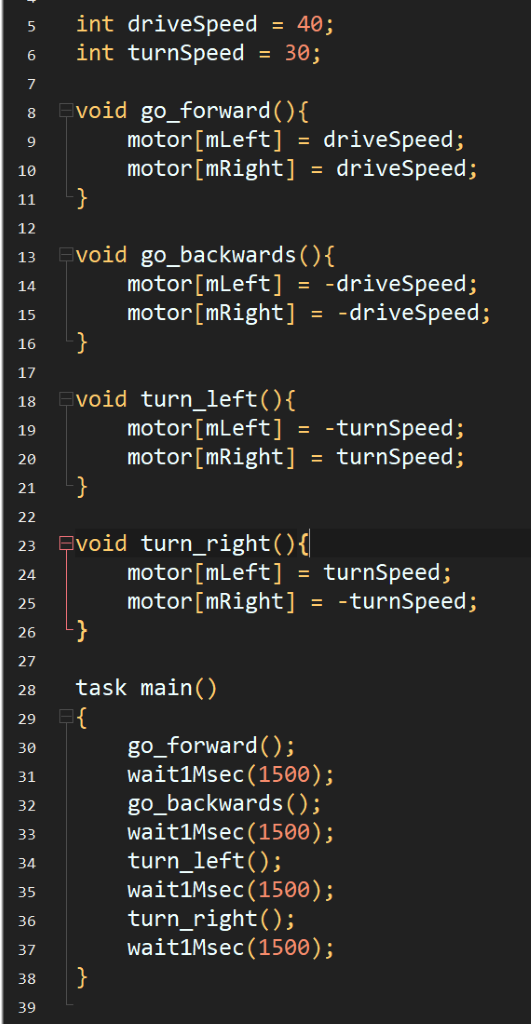
* 1. Wahl der Software am EV3

Diese Wahl ist nur für die Verbindung über Amazon IOT notwendig. Standardmäßig verwendet Lego EV3 eine eigene Programmierumgebung mit der man Code-frei Programmieren kann. Diese Umgebung unterstütz jedoch nicht die Anbindung an Amazon IOT. Es gibt jedoch Möglichkeiten um in einer anderen Programmiersprache den EV3 zu Programmieren.

Wir haben Zwei Programmiersprachen als Kandidaten gewählt, die Amazon IOT unterstützt und (indirekt) für EV3 verwendet werden können.

* + 1. Programmierung in C

Die Programmierung in C ist mit der Software „RobotC“ möglich. Es ist eine IDE die mit einer Variation von C funktioniert. Es ermöglicht die Programmierung von EV3 Robotern über USB Kabel. Dadurch können Programme schnell geschrieben, kompiliert und getestet werden.

Die Installation der IDE wurde durchgeführt und ein Beispielprogramm geschrieben (siehe Abbildung links) dass den Roboter Bewegt. Das Programm wurde getestet und funktioniert.

* + 1. Programmierung in C++

Da wir dieses Semester die Programmiersprache C++ lernen, wäre es Vorteilhaft den Roboter in dieser Sprache zu programmieren.

Die Programmierung von EV3 Robotern in C++ ist nicht mit RobotC oder ähnlichen Entwicklungsumgebungen möglich.

Die einfachste Manier ist es [EV3-dev](http://www.ev3dev.org/), ein auf Unix basierendes Betriebssystem auf einer Micro-SD Karte zu installieren, welche in den EV3 eingesteckt wird. Die Firmware des Roboters muss nicht angepasst werden.

Leider ist die Installation der Unix Version und Programmierung der C++ Programme ist nur von einem Unix-Betriebssystem möglich.

1. C#-Desktopapplikation

Die Desktopapplikation verbindet die SQS-Message-Queue mit dem Roboter. Darunter fällt das Erstellen einer Verbindung zu SQS Queue und Roboter, das Empfangen und Übersetzen von Nachrichten, das Senden von Befehlen an den Roboter entsprechend dem Kommando.

* 1. Struktur

Zur Erstellung des Programms werden 4 Dateien benötigt: Die C#-Dateien “Program.cs” und “Ev3Command.cs” und die Konfigurationsdateien “App.config” und “packages.config”.

Die “packages”-Datei enthält eine Liste der benötigten Pakete die Visual Studio benötigt, um das Projekt zu bauen. Mehr dazu im Bereich “Einrichtung von Visual Studio zur Bearbeitung”.

Die “App.config” besteht aus 3 Key-Value-Paaren:

* “Ev3Port” ist der Bluetooth Port, mit dem der Roboter verbunden ist.
* “AwsSqsAddress” ist die Adresse der SQS-Queue
* “StepsPerDegreeTurn” ist ein Faktor um die Drehung des Roboters zu kalibrieren

Die C#-Datei “Ev3Command.cs” enthält eine Klasse zum Speichern der Argumente eines Kommandos.

Die C#-Datei “Program.cs” enthält den gesamten Code außer die Ev3Command-Klasse. Ihr Inhalt und Funktionsweise wird im Bereich “Funktionsweise” erklärt.

* 1. Gebrauch

Um das Programm zu verwenden muss es mittels Visual Studio gebaut werden. Danach kann das Programm ausgeführt werden. Optional kann das Programm für den verwendeten Roboter kalibriert und erweitert werden.

* 1. Einrichtung
     1. Installation von Visual Studio Community 2017

Für die Installation und den Download von VS C17 verweisen wir hier auf die offizielle Dokumentation und Downloadbereich von Microsoft, da sich dies ständig ändern könnte:

<https://visualstudio.microsoft.com/de/downloads/>

HINWEIS: Die C#-IDE wird auf jeden Fall benötigt. Alles andere ist in diesem Fall optional bei der Installation.

* + 1. Zusätzliche Paketinstallation

Nachdem man VS erfolgreich installiert hat, werden noch zusätzliche Pakete benötigt, die nicht standardmäßig zur Auswahl stehen.

Hierfür wechselt man zunächst in die NuGet-Paketmanager Konsole. Zu dieser kommt man über die Reiter „Extras“ [Symbol] „NuGet-Paket-Manager“ [Symbol] „Paket-Manager-Konsole“

In dem aufscheinen Fenster gibt man nun folgenden Befehl ein:

„Install-Package AWSSDK.SQS -Version 3.1.0.4“

* + 1. Konfigurieren des Bluetooth Ports und der AWS-Adresse

ALLE KONFIGURATION FINDEN IN DER DATEI „App.config“ STATT!

Zunächst richten wir die Verbindung zum EV3 mittels Bluetooth ein. Hierfür muss man im Programm die COM angeben, auf der die Bluetoothverbindung hergestellt ist.

Dazu sucht man folgende Zeile und ändert das COM entsprechend ab:

<add key="Ev3Port" value="COM3" /> <!-- or for Linux/OSX, "/dev/tty.YourPortName" -->

Des weitern muss man noch einstellen von wo sich das Programm die Nachrichten/Befehle holen soll:

<add key="AwsSqsAddress" value="https://sqs.eu-west-1.amazonaws.com/697251208257/Test2Queue" />

Hier gibt man die ARN-Adresse des SQS-Services an, den man vorher erstellt hat.

WICHTIG: In der Datei „Program.cs“ gibt es unter der Function „private static void Configure()“ die Einstellung, in welcher Region sich der Service befindet! Dieser muss richtig mit dem eigenen Service zusammenpassen:

RegionEndpoint endpoint = RegionEndpoint.EUWest1;

Dies ist in unserem Fall „EUWEST1“!

* + 1. Bauen des Programms

Zu guter Letzt muss man das Programm nur noch „Bauen“ oder in Englisch „Builden“. Hierzu drückt man folgende Tastenkombination:

„STRG + SHIFT + B“

Und anschließend um das Programm auszuführen:

„F5“

* + 1. Ansteuern des Roboters

Sobald das Programm, Alexa und der EV3 bereit sind, kann angefangen werden.

Zuerst wird das Programm gestartet. Nach einigen Sekunden sollte der Roboter einen Piepston von sich geben und am Terminal am Computer die Nachricht “Connected… Waiting for Commands…” sichtbar sein. Falls nicht ist die Verbindung zum Roboter fehlgeschlagen.

Das Programm wartet nun auf Kommandos. Sobald Alexa ein Kommando übergeben wurde, erscheint der Inhalt der Nachricht am Terminal. Der Roboter sollte den Befehl nun ausführen.

Zum Beenden des Programms kann das Fenster vom Terminal geschlossen werden oder im Terminal “STRG-c” gedrückt werden.

* + 1. Kalibrieren der Drehung

Damit der verwendete Roboter sich entsprechend der Angegebenen Gradzahl, die Motoren nicht zu lange oder zu kurz dreht, kann ein Faktor geändert werden.

Der Faktor ist eine Kommazahl die sich in der “App.config” befindet.

Eine Erhöhung des Faktors entspricht einer weiteren Drehung, die Verringerung einer Kürzeren.

Standardmäßig ist dieser Wert auf 3.5 gestellt.

* + 1. Erweiterung der Funktionalität

Um Kommandos hinzuzufügen oder deren Struktur zu ändern, muss der Code angepasst werden.

Kommandos können in der für diesen Zweck markierte stelle in der Funktion “ProcessCommand” in der Datei “Program.cs” hinzugefügt werden.

Die Struktur kann in der C#-Datei “Ev3Command.cs” geändert werden. Von der Klasse abhängige Funktionen in der “Program.cs” müssen ebenfalls angepasst werden.

* 1. Funktionsweise

In diesem Bereich wird die Implementierung der Funktionalität erklärt. Es wird chronologisch, entsprechend dem Programmverlauf vorgegangen.

* + 1. Laden der Konfiguration und Verbindung zur SQS-Queue

Die „Configure“-Funktion wird aufgerufen. Sie ladet Bluetooth-Port, SQS-Adresse und den Faktor zum Kalibrieren der Drehung aus der “App.config”-Datei und stellt eine Verbindung zur SQS-Queue auf.

* + 1. Verbindung zum Roboter

Als nächstes wird die Execute-Funktion aufgerufen. Der erste Teil dieser Funktion erstellt verbindet sich mit dem Roboter über Bluetooth und gibt bei Statusinformationen aus. Die Port-Nummer wird separat über die “\_brick\_BrickChanged”-Funktion ausgegeben.

* + 1. Nachrichten abrufen

Die Funktion “PollForQueueMessage” wird aufgerufen. Sie sendet eine Anfrage für Nachrichten an die Queue. Wenn sich eine Nachricht in der Queue befindet, wird sie entgegengenommen und aus der Queue gelöscht. Danach wird die Nachricht am Terminal ausgegeben und durch die “GetEv3CommandFromJson”-Funktion bearbeitet.

* + 1. Kommando erstellen

“GetEv3CommandFromJson” deserialisiert die Nachricht und erstellt ein EV3-Kommando aus den Argumenten, die über die Nachricht übermittelt wurden.

* + 1. Kommando interpretieren und ausführen

Wieder in der “Execute”-Funktion wird das Kommando an “ProcessCommand” weitergegeben. Die Funktion wählt ob ein Kommando und wenn Ja welches an den Roboter gesendet werden. Eine der folgenden Funktionen werden Ausgeführt.

“MoveForward” zeigt eine Statusmeldung am Terminal, erstellt und sendet ein Kommando an den Roboter. Vorherigen Kommandos werden überschrieben. Das Kommando besteht aus einzelnen Einstellungen für zwei Motoren. Falls eine Distanz in Centimeter als Argument angegeben wurde, wird die Distanz berechnet und verwendet. Falls nicht, fährt der Roboter ca. 30 cm vorwärts.

“MoveBackward” funktioniert genau gleich wie “MoveForward”, mit dem Unterschied, dass die Polarität der Motoren umgekehrt ist.

“MoveLeft” und “MoveRight” haben den selben Ablauf wie “MoveForward” und “MoveBackward”, mit dem Unterschied, dass die Motoren in entgegengesetzter Richtung gedreht werden.

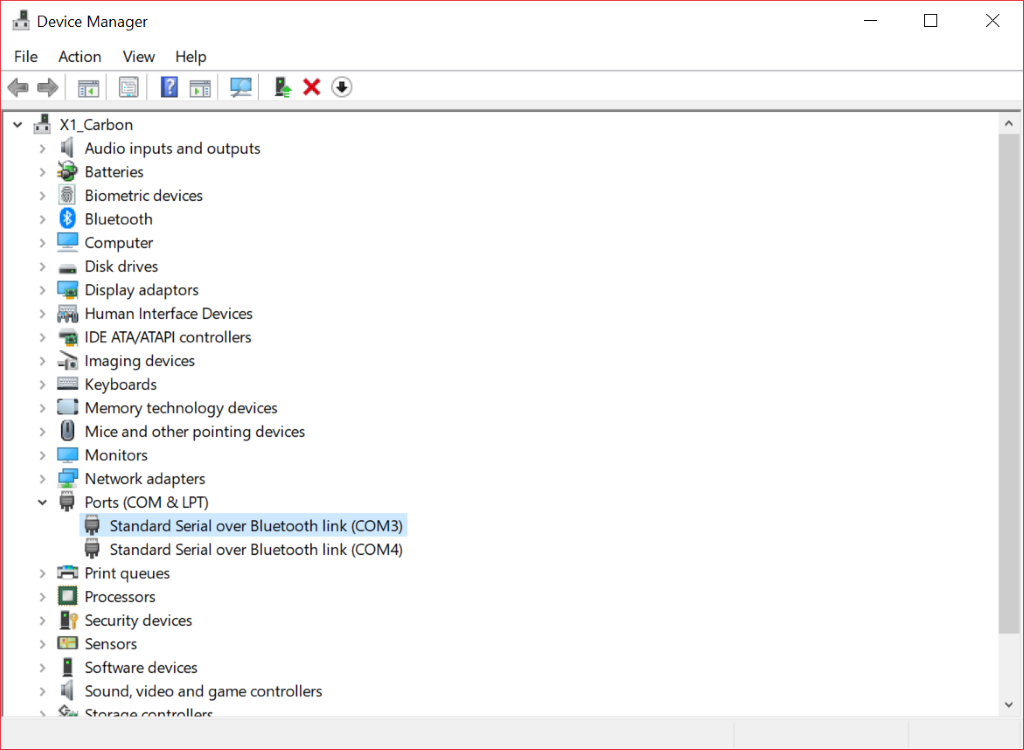
* + 1. Abschluss

Das Programm durch die Eingabe von “STRG-c” terminiert oder durch das Schließen des Terminals.

1. Verbinden des Roboters mit SQS über die C#-Application

Der Roboter musste nun nur noch mit dem Desktop verbunden werden. Das war mit dem modernen Device Manager von Windows 10, der der unter *Settings -> Devices* verfügbar ist, nicht möglich. Mit dem klassischen Gerätemanager, welcher unter *Control Panel -> Hardware and Sound -> Devices and Printers* zu finden ist, konnte die Verbindung hergestellt werden. Wir sind wie folgt vorgegangen:

1. In Windows: *Settings -> Devices*
   1. Bluetooth aktivieren
   2. *Related Settings: Related Settings -> Discovery: Allow Bluetooth devices to discover this device* ankreuzen.
2. Am Ev3: *Settings -> Bluetooth*
   1. *Visibility* ankreuzen
   2. *Bluetooth* ankreuzen
   3. *iphone/iPad/iPod* NICHT ankreuzen
3. In Windows im Klassischem Paketmanager:
   1. *Add a device*
   2. warten
   3. Mit „EV3“ verbinden
4. Im Device Manager
   1. nachsehen ob die Zwei COM-Ports zu sehen sind
   2. Ihre nummern notieren (hier COM3 und COM4)



Die Nummern werden Später benötigt um den Roboter zu verbinden. In der Config-Datei vom Programm wird der Port mit der Niedrigeren Nummer verwendet. Falls das nicht funktioniert, der andere.

1. Quellen

Das Programm ist eine modifizierte Version des Programms von Jim Drewes, welches unter diesem Link verfügbar ist:  
<https://github.com/jimdrewes/alexa-to-ev3/tree/master/alexa-to-ev3.console>

Das Programm wurde kommentiert, für unseren Roboter konfiguriert und die Funktionalität erweitert.

Artikel zum erstellen von Alexa Skills: https://jaxenter.de/alexa-skills-entwickeln-64751

1. Anmerkungen

Die beiden C#-Dateien wurden vollständig kommentiert. Zusätzlich wurde die Reihenfolge der Funktionen in “Program.cs” chronologisch entsprechend dem Aufruf sortiert.

Im Rahmen der Konfiguration wurde in der “App.config”-Datei der Bluetooth-Port und SQS-Adresse eingestellt. Außerdem wurde in “Program.cs” in den “Move…”-Funktionen der Port und die Polarität der Motoren angepasst.

Zur Kalibrierung der Drehung des Roboters wurde ein Faktor hinzugefügt. Dieser Faktor wird unter dem Eintrag “StepsPerDegreeTurn” der “App.config” eingestellt. Es ist der Faktor, um den die Räder gedreht werden müssen um den Roboter um x Grad zu drehen. Er wurde hinzugefügt da die Steuerung mit Gradzahl einfacher ist als die Eingabe der Steps, und sie so einfacher als im Code selbst geändert werden kann. Konkret wurde ein Key-Value-Paar in der Config-Datei und im Code eine Konstante mit dem Key-Namen und eine Private Variable für den Wert. Die Configure Funktion im Code wurde ebenfalls erweitert um den Wert der Privaten Variable entsprechend der Config-Datei zu setzen. Die „Moveleft“- und „Moveright“-Funktionen wurden angepasst damit sie den Faktor verwenden.

Die Struktur der EV3-Kommandos wurde geändert um die Angabe des Roboternamens und eines Betrags für die Bewegung zu unterstützen. Dafür wurde die Ev3Command-Klasse um die Einträge “DeviceName” und “Value” erweitert. Die “GetEv3CommandFromJson”-Funktion wurde angepasst um die Werte zu setzen. Der “Value” wird in den “Move…”-Funktionen verwendet um die Weite der Bewegung/Drehung zu bestimmen. Er ist Optional und ein Standardwert wird verwendet, falls er nicht angegeben wird.