MINI GLO

JUAN CARLOS LARA ANAYA

ERICK SALAS ROMERO

INDEX:

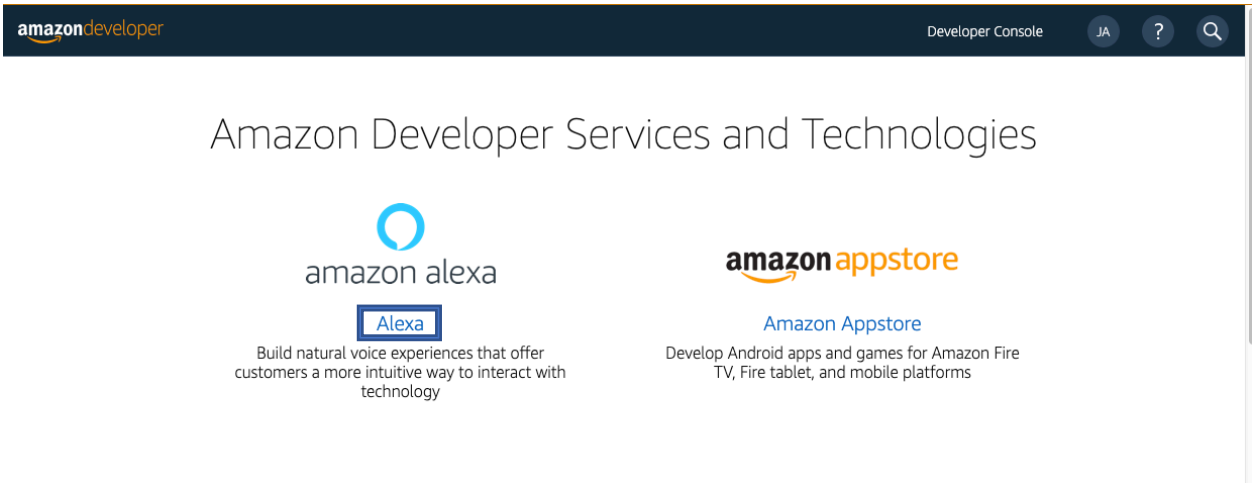
1. Create an Alexa skill with lambda function based on java 8
2. Amazon Web Services
3. Install Maven
4. Create Project
    - 4.1. Pom.xml
    - 4.2. Source code
        - 4.2.1. Handlers
        - 4.2.2. Attributes
        - 4.2.3. Slots
        - 4.2.4. HTTP Requests
    - 4.3. Compile Project
    - 4.4. Upload jar to Lambda
    - 4.5. Account Linking
5. CloudWatch
6. Mini Glo Manual
    - 6.1. Alexa Skill

ALEXA SKILL DOCUMENTATION

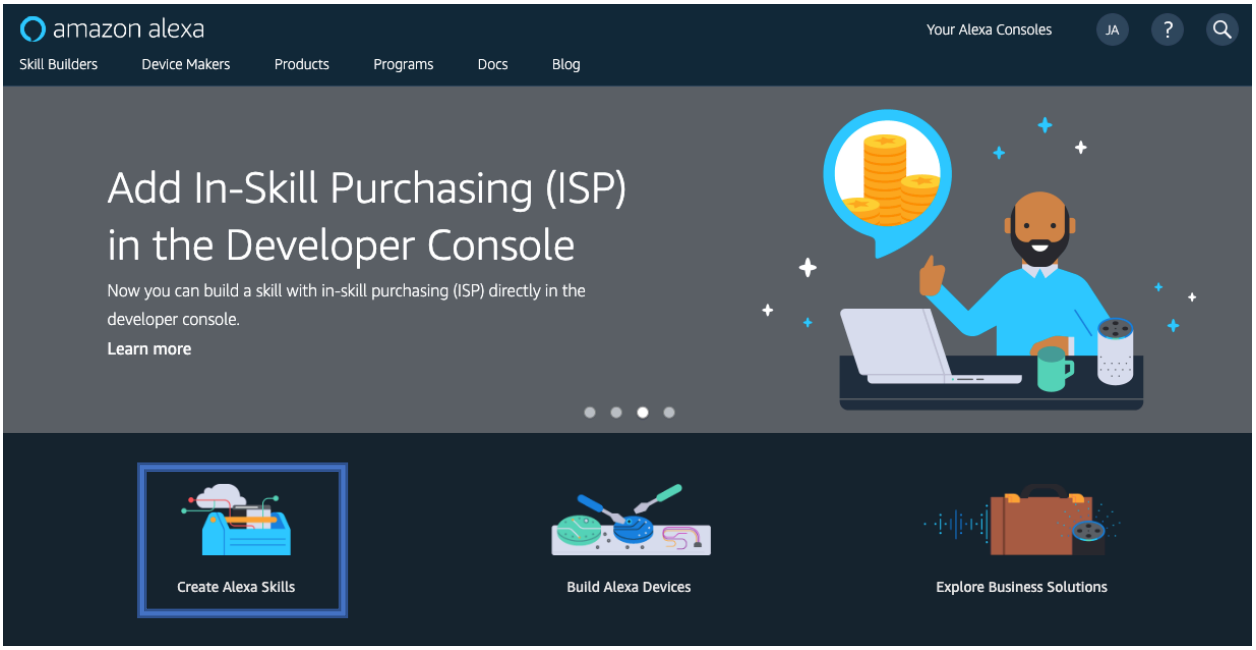
Amazon Documentation:
https://developer.amazon.com/alexa-skills-kit
https://github.com/alexa/alexa-skills-kit-sdk-for-java
https://alexa-skills-kit-sdk-for-java.readthedocs.io/en/latest/
https://github.com/alexa/skill-samples-java

Create an Alexa Skill with lambda Function based on Java 8

In order to create an Alexa skill, an Amazon Developers account is needed, once signed in or registered, hit click on Amazon Alexa link:



After that hit on create Alexa Skill:



And then in the button ‘Start Skill’.

To know more about how to create, please refer to link: https://developer.amazon.com/docs/devconsole/create-a-skill-and-choose-the-interaction-model.html

When the skill is already created, it's time to create an intent; an intent is the way of interacting between you and Alexa A.I, here it can be added the way of invoking of each intent, to do it, write the text the user will have to say to activate that intent.

Inside intent you can have slots, that are variables that Amazon provide us to obtain information that the user said when the invocation was made. Slots are pretty helpful when we want to create something with a name, when we want to get an answer or we want to get a type of something; to create a slot, inside your intent add opening and closing braces '{}' and inside of them add the name of the slot.

Slots can be of multiple types that Amazon provide us, refer to this link to get more information about slot types: https://developer.amazon.com/docs/custom-skills/slot-type-reference.html and to create your own slot types refer to https://developer.amazon.com/docs/custom-skills/create-and-edit-custom-slot-types.html

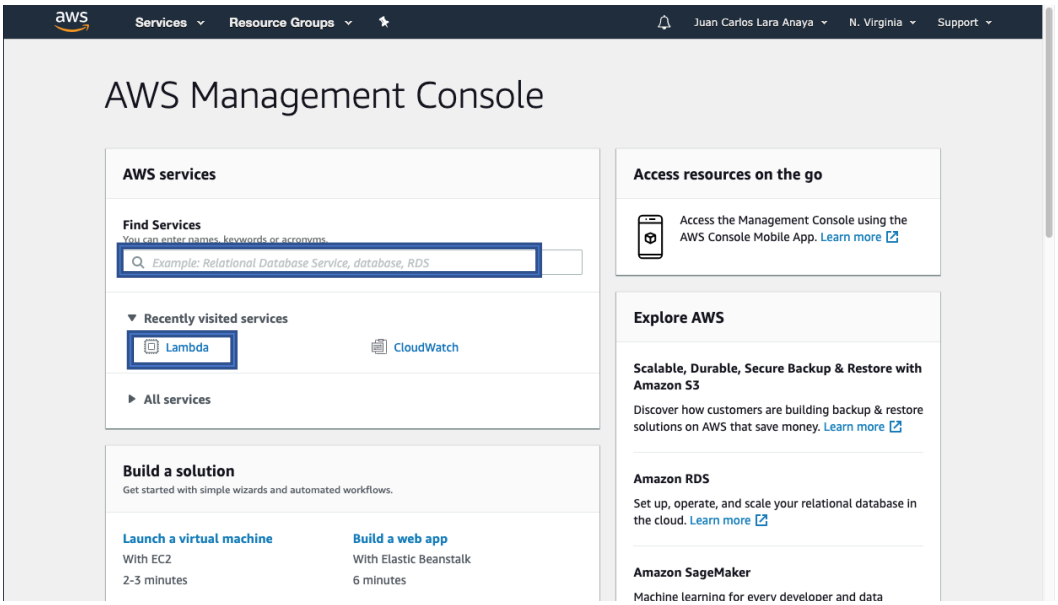
After created the intents, it's time to add your code, to achieve this we will need a powerful tool of Amazon called

Amazon Web Services.

AWS Documentation: https://docs.aws.amazon.com/index.html#lang/en_us

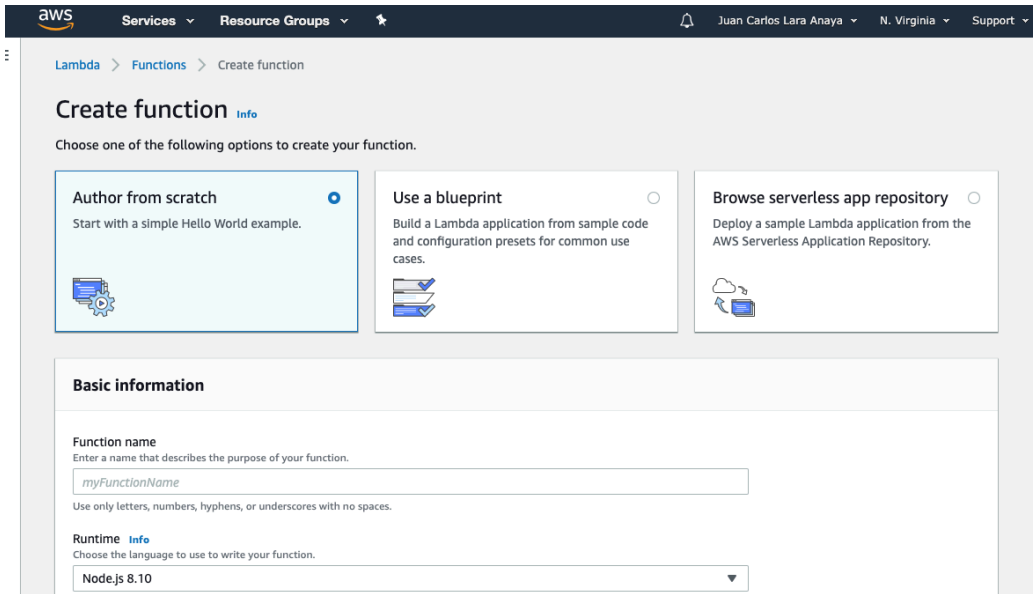
Inside here we can find multiple helpful items, settings and tools to our Alexa skill or other projects, but here we will center on create a Lambda Function. To know more of Lambda please see documentation: https://docs.aws.amazon.com/lambda/index.html?id=docs_gateway#lang/en_us

To create our lambda function first we need to create an account, it is not needed to select a pay plan, there's a free option where we can test al function for a year. After we signed, we will go to AWS console:

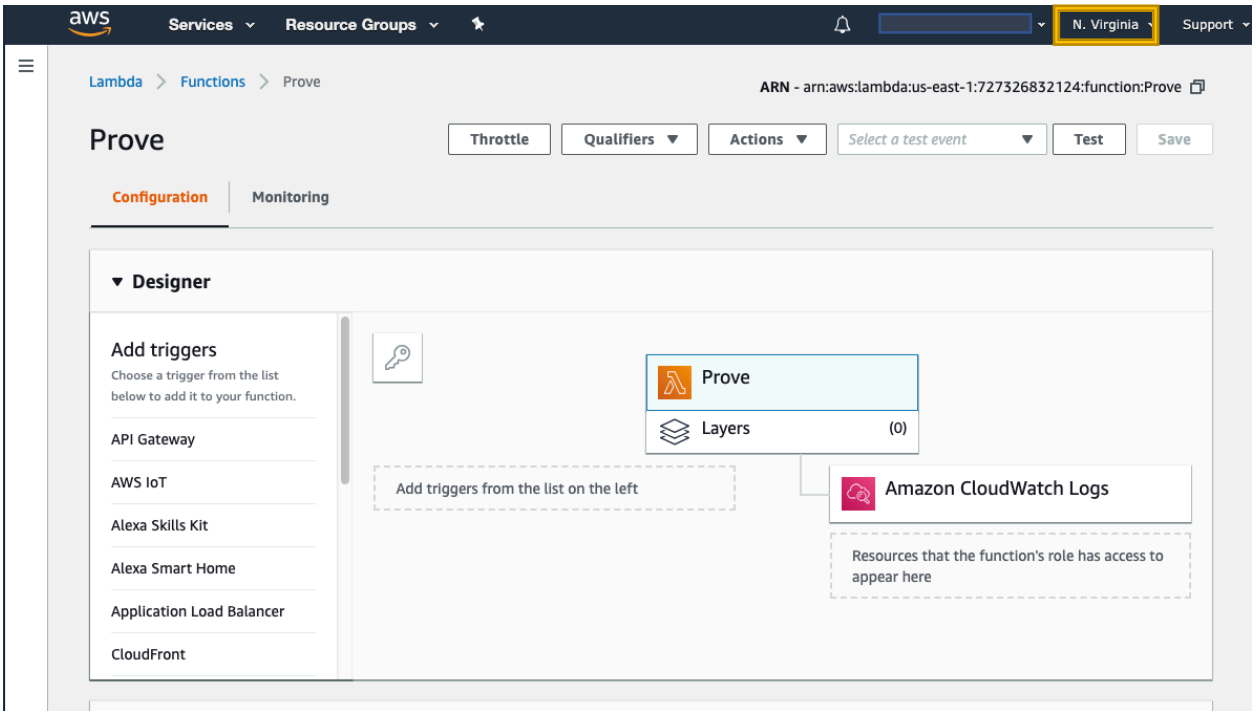


Here are all the tools to use. We will have to look in the Find Services Textbox Lambda.

Click there then we create a function clicking on orange button that's says create Function:

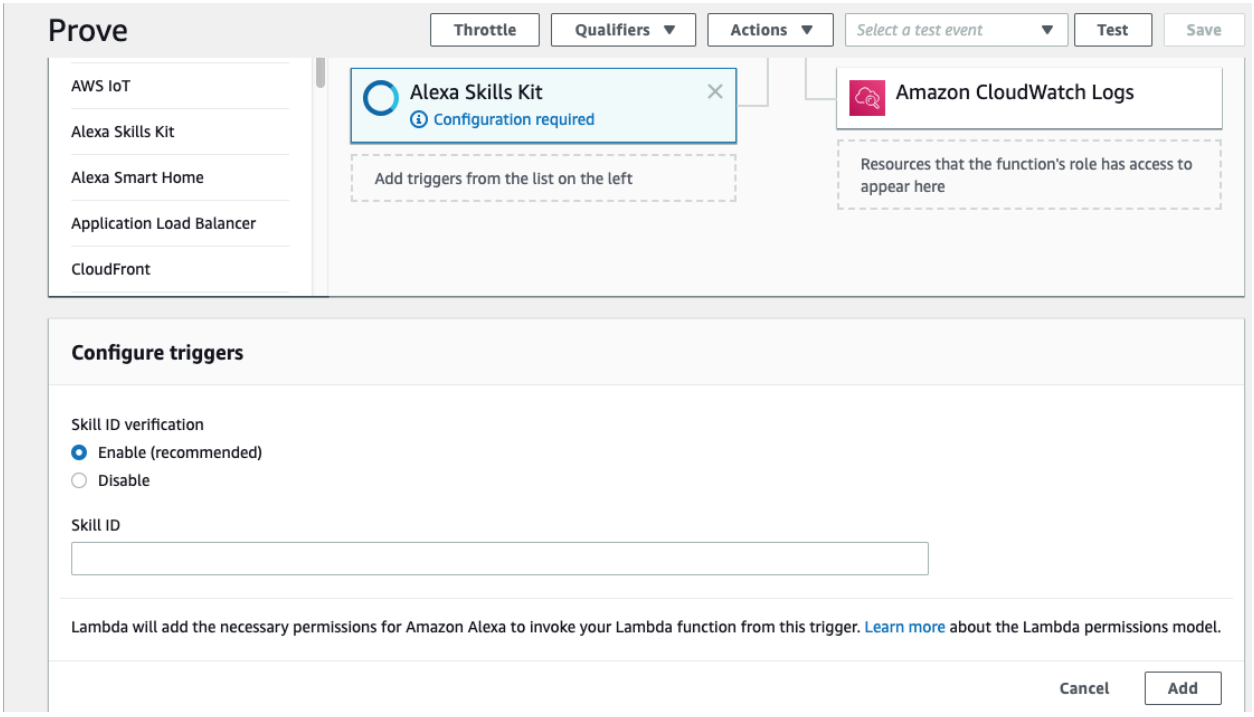


Here we will click on Author from Scratch option, then We assign a Function Name, in runtime we select Java 8, and in permission we select choose or create execution role and select Create a new role with basic Lambda permissions and click on create.

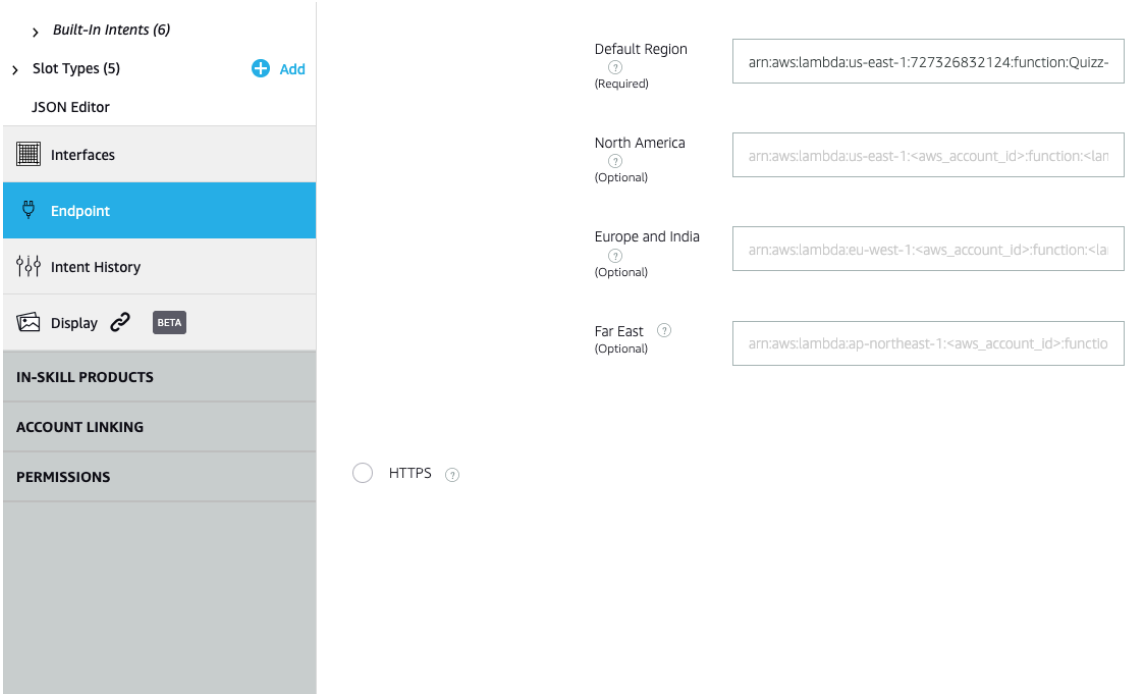


Please, take a look on yellow rectangle in the top right side, that our zone is N. Virginia, this because here we can use the Alexa Skill Kit that's needed to configure our Alexa Skill.

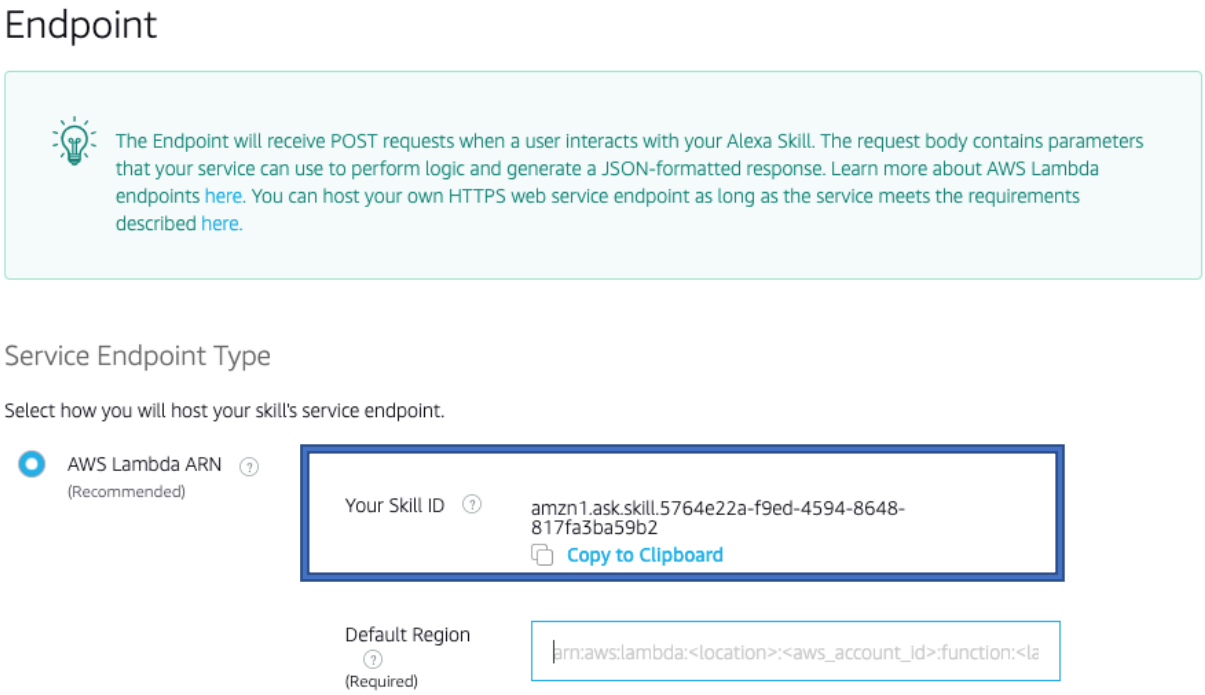
The next Step is to click the Alexa Skills Kit to add it in the Lambda Function.



The last step should have added this new trigger and the configuration chart for it. In this one they ask us for or Skill ID, so, go to the Alexa Develop Management, enter to our skill, click on build, and go down to the endpoint label.



On the top of this label you will find our Amazon Alexa Skill ID.



You can see down the Blue rectangle that there's a text box asking us for Default Region, we can find this on the top of our lambda function, after we add that we will click on the button in the Alexa skill to save endpoints.

After all this, we can start creating our code in Java 8, and Maven!

To see more documentation of how to create an Alexa Skill with Java 8, click on the next link: https://docs.aws.amazon.com/lambda/latest/dg/java-programming-model.html

Install Maven

Before starting coding we need to be sure we have all needed, first of all, we need JDK 8, get it from: https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

After we installed java JDK 8, we need a second thing called apache Maven, to get more information about Maven please look at: https://maven.apache.org/index.html.

To install on Linux, we will use the command on terminal:

```
sudo apt-get install mvn
```

If we are using Homebrew, we use

```
brew install maven
```

If you are using Windows 10, please go to the link: https://maven.apache.org/download.cgi and down load the source zip archive. Unzip it, then open a CMD with Ctrl + R, write CMD and enter, this will open a new terminal, in there write
set PATH="{The route where you unzip the maven source}\apache-maven-3.x.y\bin";$PATH

CREATE PROJECT

After this, we are ready to start. Create an empty project in your favorite IDE, I recommend seeing Intelli J Idea IDE.

When it created, we will create a pom.xml in the next route:

```
project-dir/pom.xml
```

This pom will help us to add dependencies and creating the route to compile our project using Maven.
Then we have to create a route needed for Lambda Function and Maven that is:

```
project-dir/src/main/java/  (your code goes here)
```

POM.XML

The first thing you have to create is the pom.xml with the next text on it:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
 <modelVersion>4.0.0</modelVersion>
 <groupId>{group-NAME}</groupId>
 <artifactId>{ARTIFACT-NAME}</artifactId>
 <packaging>jar</packaging>
 <version>1.0</version>
 <name>{NAME}</name>
 <url>http://developer.amazon.com/ask</url>
 <licenses>
  <license>
   <name>The Apache License, Version 2.0</name>
   <url>http://www.apache.org/licenses/LICENSE-2.0.txt</url>
  </license>
 </licenses>
 <developers>
  <developer>
   <name>Alexa Skills Kit</name>
   <email>ask-sdk-java@amazon.com</email>
   <organization>Alexa</organization>
   <organizationUrl>http://developer.amazon.com/ask</organizationUrl>
  </developer>
 </developers>
```

```
<scm>
  <connection>scm:git:https://github.com/amzn/alexa-skills-kit-java.git</connection>
  <developerConnection>scm:git:https://github.com/amzn/alexa-skills-kit-java.git</developerConnection>
  <url>https://github.com/amzn/alexa-skills-kit-java.git</url>
 </scm>

 <dependencies>
  <dependency>
    <groupId>com.amazon.alexa</groupId>
    <artifactId>ask-sdk</artifactId>
    <version>2.5.5</version>
  </dependency>
 </dependencies>

 <build>
  <sourceDirectory>src</sourceDirectory>
  <pluginManagement>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.7.0</version>
        <configuration>
          <source>1.8</source>
          <target>1.8</target>
          <forceJavacCompilerUse>true</forceJavacCompilerUse>
        </configuration>
      </plugin>
    </plugins>
  </pluginManagement>
 </build>

</project>
```

Feel free to change names between {}.

After that created, we will be ready to create our src code.

SOURCE CODE

Inside our project create a directory src/main/java/

Here create a package starting with com. 'whatever you want here'.

And create your first Class called MainSkillStreamHandler, this will be the “main” controller of your skill, and here you will refer all your Intent Handlers. This class will extend from SkillStreamHandler, its necessary to import com.amazon.ask.*;

In this part we need to manage all created handler intents so, create a constructor and add super (Skills.standard().addRequestHandlers(“In this part it will go all handlers.”).build();

The next thing to do is create a directory called handlers

Handlers

All request has to implements RequestHandler and override all methods from this interface.

This interface has two methods: the Boolean canHandle and the Optional<Response> handle, both receives the HandlerInput input, that is the request that the user do and the Alexa skill parses.

The first Handler we have to create is the LaunchRequestHandler, this trigger the hole skill.

In the canHandle method it always has to happen one thing the return of an input.matches(Predicates.***), where the *** is the type of predication we want. For LaunchRequestHandler we must put *** = requestType(LaunchRequest.class). For other handlers we must put *** = intentName(“intent-Name”).

In the handle we need to return a response, so that Alexa can respond with a text, with a card, continue or not session, etc.

To create the response we need to return the next thing:

Input.getResonseBuilder()
.//between these two lines we can add multiple options, please see Documentation for more information:

https://alexa-skills-kit-sdk-for-java.readthedocs.io/en/latest/Response-Building.html
.build();

Session Attributes :

https://alexa-skills-kit-sdk-for-java.readthedocs.io/en/latest/Skill-Attributes.html

The Attributes in an Alexa Skill are those variables that are saved for a lapse of time, like during the session of the skill, when the echo dot is on, etc., and they are really helpful, cause the let us save variables and work with them in all places of our handlers.

See Documentation to know how to declare.

Slots

Here things start to get interesting, how about obtain information directly from the request of the user that made to the Echo Dot.

This are the variables of the request, and there are many type of them:

https://developer.amazon.com/docs/custom-skills/slot-type-reference.html

Also we can create ours:

https://developer.amazon.com/docs/custom-skills/create-and-edit-custom-slot-types.html

To obtain them, we will need the IntentRequest, to get it we need to put the next code:

IntentRequest intentRequest = (IntentRequest) input.getRequestEnvelope();

With this we can obtain slots, slots are returned in a HashMap with a String like key and a Slot like a value

Map<String,Slot> slots = intentRequest.getIntent().getSlots;

And with this we have obtained all slots used in our intents.

Make HTTP requests with Java

To make request to an external API with Java and parse the JSON response we can use a dependency called: com.google.gson (If you copy the pom given you will have already included):

```
<dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
    <version>2.8.5</version>
</dependency>
```

To make a HTTP request we have to import :

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.net.HttpURLConnection;
import java.net.URL;
import java.net.URLEncoder;
import java.util.Iterator;
```

We have to create a HTTPURLConnection private variable initialized with null, also we can have constants to avoid overwriting, that contains our USER_AGENT that must be : "Mozilla/5.0" and our URL that we will be using (sometimes we do testes with a different URL so this one will help us change all the URL of our requests easily.

Here we will include examples of the most used methods to do requests : GET, POST, and DELETE.

```
//GET
public  BufferedReader sendGet(String url, Map<String,String>
params) throws IOException{
    if(params!=null && params.size()>0){
        url += "?";
        Iterator<String>paramsItertor =
params.keySet().iterator();
        while(paramsItertor.hasNext()){
            String key = paramsItertor.next();
            String value = params.get(key);
            url += key + "=" + value;
            if(paramsItertor.hasNext()) url += "&";
            System.out.println(url);
        }
    }
    URL obj = new URL(url);
    connection = (HttpURLConnection)obj.openConnection();
    connection.setRequestMethod("GET");
    connection.setRequestProperty("User-Agent",USER_AGENT);
    connection.setRequestProperty("Content-Type","text/plain");
    connection.setRequestProperty("charset","utf-8");
```

```
    int responseCode = connection.getResponseCode();

    System.out.println("\nSending 'Get' request to URL: " +
url);
    System.out.println("Response Code:" + responseCode );

    BufferedReader in =  new BufferedReader(
            new InputStreamReader(connection.getInputStream())
    );

    disconnect();

    return in;

}
//DELETE
public BufferedReader sendDelete(String url, Map<String,String>
params) throws IOException{
    if(params!=null && params.size()>0){
        url += "?";
        Iterator<String>paramsItertor =
params.keySet().iterator();
        while(paramsItertor.hasNext()){
            String key = paramsItertor.next();
            String value = params.get(key);
            url += key + "=" + value;
            if(paramsItertor.hasNext()) url += "&";
            System.out.println(url);
        }
    }
    URL obj = new URL(url);
    connection = (HttpURLConnection) obj.openConnection();
    connection.setUseCaches(false);
    connection.setDoInput(true);
    connection.setRequestMethod("DELETE");
    connection.setRequestProperty("User-Agent",USER_AGENT);
    connection.setRequestProperty("Content-Type","text/plain");
    connection.setRequestProperty("charset","utf-8");
    int responseCode = connection.getResponseCode();
```

```
    System.out.println("Sending 'Delete' to url: " + url);
    System.out.println("Response Code:" + responseCode);
    BufferedReader in = new BufferedReader(
            new InputStreamReader(connection.getInputStream()));
    disconnect();
    return in;
}

//DISCONNECT
public  void disconnect(){
        if(connection!=null) connection.disconnect();
    }
//POST
public  BufferedReader sendPost(String url, Map<String,String>
params)throws IOException{
        URL obj = new URL(url);
        connection = (HttpURLConnection) obj.openConnection();
        connection.setUseCaches(false);
        connection.setDoInput(true); //true indicates the server
returns response
        connection.setRequestMethod("POST");
        StringBuffer requestParams = new StringBuffer();
        System.out.println("Sending 'POST' to " + url);
        if(params!=null && params.size()>0) {
            connection.setDoOutput(true); // true indicates POST
request
            Iterator<String> paramIterator =
params.keySet().iterator();
            while(paramIterator.hasNext()) {
                String key = paramIterator.next();
                String value = params.get(key);
                requestParams.append(URLEncoder.encode(key,
"UTF-8"));
                requestParams.append("=");
                requestParams.append(URLEncoder.encode(value,
"UTF-8"));
                if(paramIterator.hasNext())
requestParams.append("&");
                System.out.println(requestParams);
            }
```

```
        }
        //SEND Post Data
        OutputStreamWriter writer = new OutputStreamWriter(
                connection.getOutputStream());
        writer.write(requestParams.toString());
        writer.flush();
        System.out.println(connection.getResponseCode());
        BufferedReader in =new BufferedReader(
                new
InputStreamReader(connection.getInputStream()));
        disconnect();
        return in;
    }
```

This is the correctly way to create the methods, If you see, they receive to things, the URL where we will make the request and a Hash map where we put the parameters, this parameters will be sent in different ways, in the delete and get methods we send them as query parameters, and in Post we send them in body. This is important to know because the Get and Delete doesn't receive any parameters by body.

After making the request we have to disconnect, in other way, some pages will take this as an attack and they will block our request.

To parse the result of the request we use gson: https://github.com/google/gson

To get information about this see: https://www.javadoc.io/doc/com.google.code.gson/gson/2.8.5

We see that our methods returns us a Buffered Reader, that is the result, to get it as a JsonObject (A simple JSON) we have to do:

JsonObject prove = new JsonParser().parse(in).GetAsJsonObject();

With this action we will obtain a prove JSON that can get our elements like this:

JsonElement element = prove.get("the think we want").GetAsJsonElement();

If we obtain a JsonArray we have to parse it as:

JsonArray prove = new JsonParser().parse(in).GetAsJsonArray();

And this will give us the JSON array.

Compile Project

To compile the java project, we need to do it from terminal, to achieve this we have to move our terminal to the custom folder of our project, right there we have to put this code:

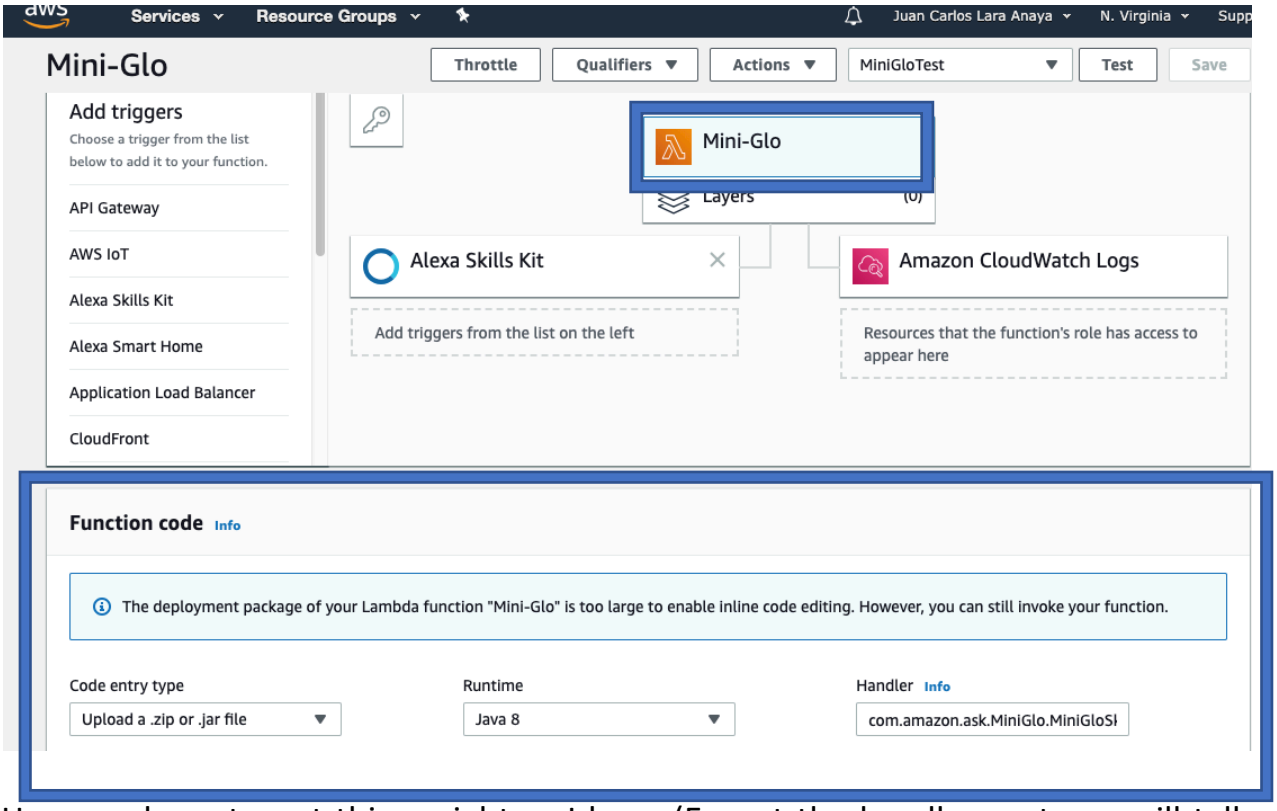
mvn assembly:assembly -DdescriptorId=jar-with-dependencies package

This will create a target folder and here will be saved two jars: {name} – {version}.jar and {name}-{version}with-dependencies.jar.

The one we have to upload to lambda is the one with the dependencies.

Upload jar to lambda

To upload we just have to go to our lambda function, click on the lambda in the between and this will open a label down:



Here we have to put things right as I have (Except the handler part, we will talk about that later).

In the Function code label there's a button that says Upload, click in that and that will open a browser dialog, look for the jar with dependencies and upload it.

Then in the Handler zone we have to reference to the SkillStreamHandler of our java code.

To do this we have to specify in what zone of our code is it, for example in my case, the Skill stream handler is in package com.amazon.ask.MiniGlo and the name of my skill stream handler is MiniGloSkillStreamHandler, so the complete reference is:

com.amazon.ask.MiniGlo.MiniGloSkillStreamHandler

After doing all this, click on the orange button on the right top side that says Save.

This will completely upload the jar to the lambda Function.

Account Linking

This one is the less specific thing in the documentation of Amazon, but I will help you to do correctly.

Inside the alexa developer page there's a zone in build that says account linking, don't use it, for some reasons is not correctly working.

Instead, we will install the CLI ASK (Command Line Interface for Alexa Skills) : https://developer.amazon.com/docs/smapi/ask-cli-command-reference.html.

To do this we must have installed npm. Please see documentation of npm here: https://www.npmjs.com/get-npm

After we installed npm, we have to put this command:

```
sudo npm install ask-cli
```

If this doesn't work use `sudo npm install -g ask-cli`

After installed, to create an account linking we need to have ready the next things:

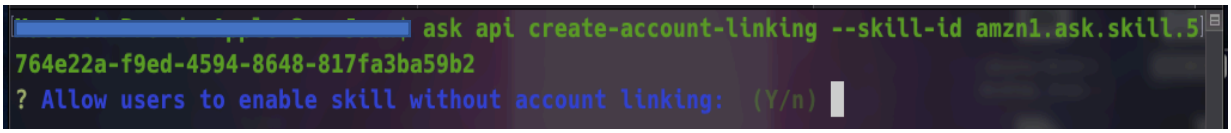
Our Alexa Skill ID, our Client Id and Client Secret of our OAuth app, the authorize and token links and the scopes that our Oauth app require.

Then we put the next command in our terminal:

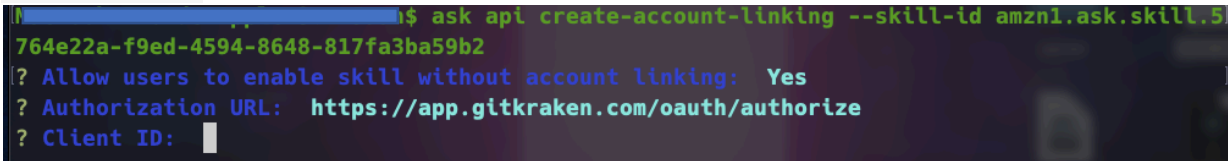
```
ask api create-account-linking [-s|--skill-id <skillId>]
```

```
Example:
```

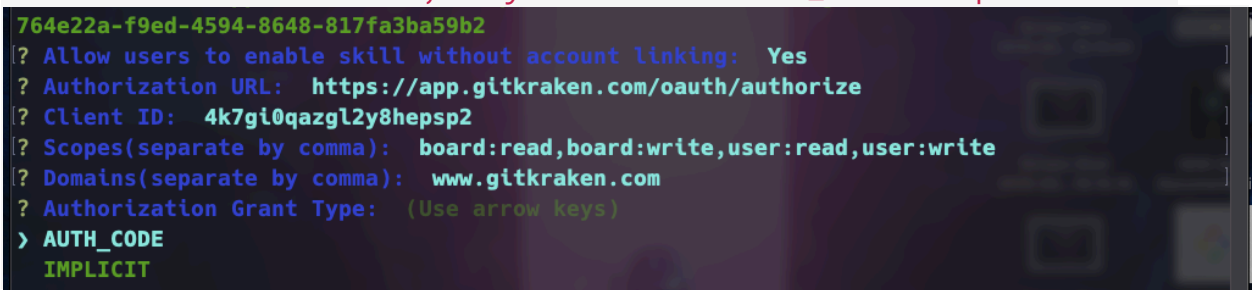
```
First we put the command:
```



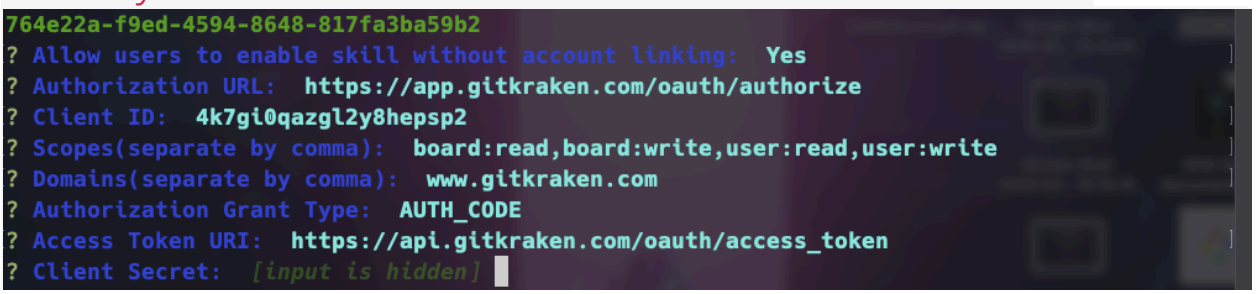
Then we put ‘y’ and enter
After we have to put our authorization URL, search that on your oauth app information:



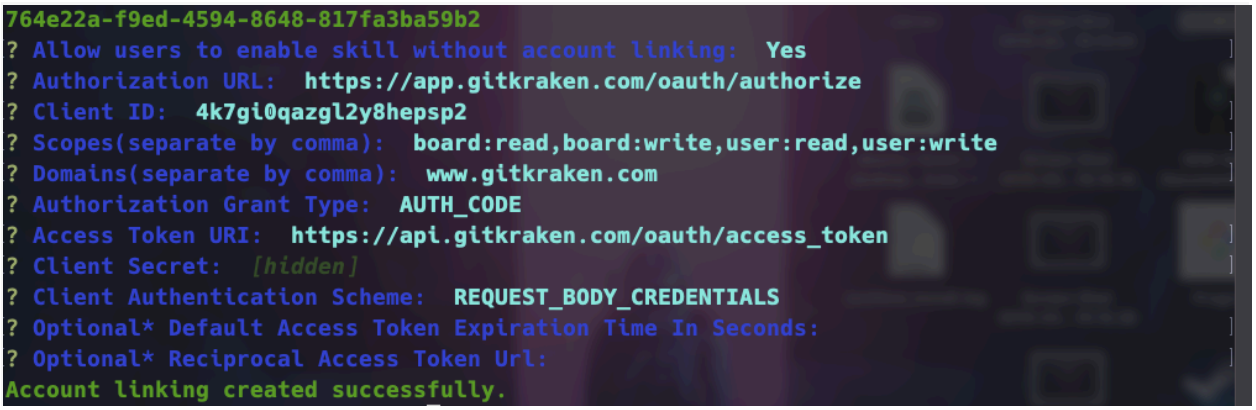
Then we put our Client ID, enter, scopes separated by commas, enter, the domain of our Oauth App, enter, then it will appear an option to select, if we need a token or not, if yes we select Auth_code and press enter:



Then they will ask us our access token URI and our Client Secret:



Here comes the tricky part that the Alexa developer browser page doesn’t let us configure and its really important when we have body credentials, (look in your documentation if your app need body credentials), then it comes to *Optional*, see in your documentations if you need them, if not just press enter until the process ends:



And with this you have ended correctly. This is the only solution I find to do successfully this action.

CloudWatch

CloudWatch is another tool of AWS that let us see the Back Trace of the skill, and is useful to see where the Exceptions occurred or why did the famous error: “There was a problem with the requested Skill response” was triggered. Some errors doesn’t appear here, like the ones that are involved with the text response.

Manual

Alexa Skill:

For Mini-Glo the Alexa Skill can do basic movements: like

Create, Edit or Delete Boards, Columns and Cards

Voice Commands:

```
case ("BOARDS"):
case "OPEN":
"open board (and the name), or search board (and the name), or
show board(and the name)";
case "CREATE":
"create board called (and the name) or create board with name
(and the name)";
case "EDIT":
"change name of board (and the name) to (new name) or
"change board (and the name) name to (new name)";
case "DELETE":
"delete current board. or. delete board (and the board Name)"
case "COLUMNS":
case "CREATE":
"add column with name (and the name). or Create column called
(and the name) "add column called  (and the name).";
case "EDIT":
"edit column (and the name) changing name to (new name) or
change column (and the name),current name to (new name) ";
case "DELETE":
"delete columns (and the column Name)";
case "CARDS":
case "CREATE":
"add card (and the name) in current column or add card (and the
name) to column (and the column name) or create card (and the
name) in column (and the column name) with description (and the
description)
```

```
case "EDIT":
"Add description (and the description) to card (and the name).
or Add label (and the label name) to card (and the name) or
Add due date (and the date) to card (and the name) or
Change due date of card (and the name) for (and the date). or. "
"Remove label (and the label name) of card (and the name). or. "
"Change description of card (and the name) for (and the
description)";
case "DELETE":
"delete (and the name)  card. or. delete card called(and the
name)";
```