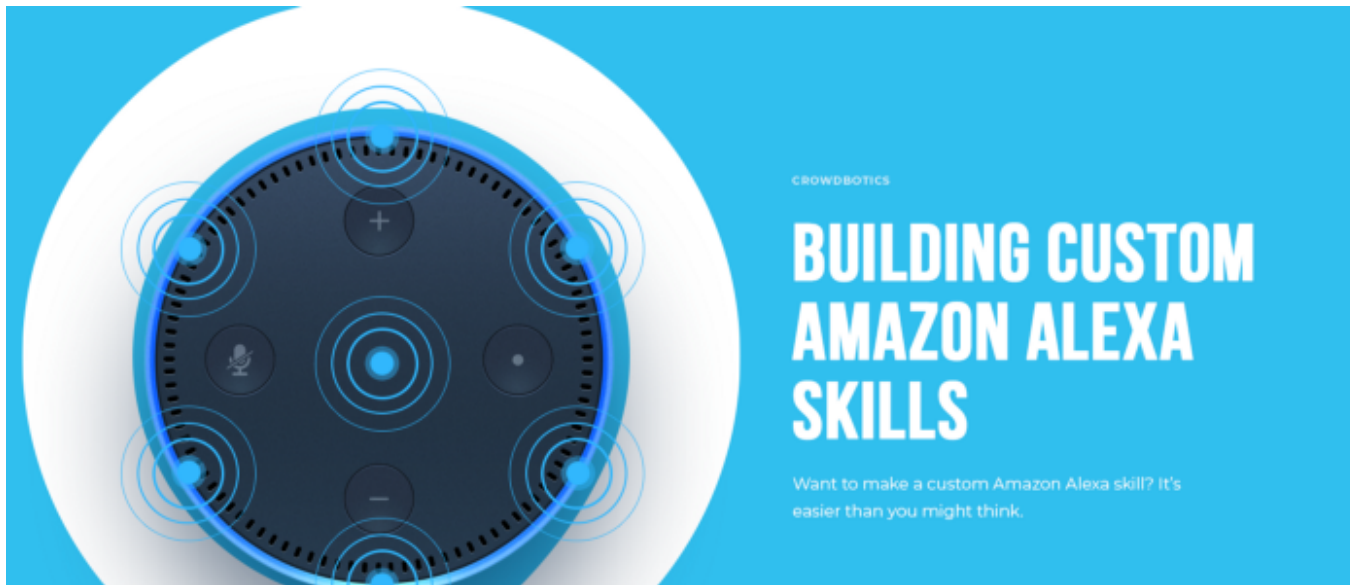


# How To Build A Custom Amazon Alexa Skill, Step-By-Step: My Favorite Chess Player



Uros Ralevic [Follow](#)

Jul 25, 2018 · 17 min read





Want to make a custom Amazon Alexa skill? It's easier than you might think. Here's everything you need to know to build your first custom Amazon Alexa skill.

## Introduction To Alexa Skills

Alexa is Amazon's cloud-based voice service which powers the Echo family of devices as well as the companion app on the Android and iOS smartphones.

Out of the box, a user can give Alexa a number of voice commands such as creating a to-do-list, set the alarm, play a song, or provide the news. The tasks Alexa performs upon user request are called "Alexa Skills". Essentially, an Alexa Skill is a voice-driven Alexa app.

Alexa has a number of built-in skills, but developers can build new custom skills, by using [Alexa Skill Kit \(ASK\)](#). The ASK, a collection of APIs and tools, handles the hard work related to the voice interfaces including speech

recognition, text-to-speech encoding and natural language processing. ASK helps developers build a skills quickly and easily.

In this post, we'll introduce the basics of Alexa skill development by building a simple custom skill called "My Favorite Chess Player".

You can then apply the same process to turn your own idea for a custom voice command into a working Amazon Alexa skill.

## **The User — Custom Alexa Skill Communication**

The communication between the user and the custom skill is achieved via an Alexa powered device such as the Echo.

Activation of a particular skill is done by saying the skill's invocation name along with the trigger word ("Alexa") which activates the Echo device. For example, the command "Alexa, open my favorite chess player" will start the "My Favorite Chess Player" skill.

Upon activating the skill, the user is able to send other voice commands, or speech requests, to the skill.

## **Communication within the Custom Alexa Skill**

The Alexa skill consists of two main components: the skill interface and the **skill service**.

The **skill interface** processes the user's speech requests and then maps them to intents within the interaction model. The intents are actions that fulfill the spoken requests from the user. Every intent has at least one utterance, a predefined word, phrase, or sentence which the user might say to invoke the intent. If a specific intent is detected, the skill interface creates a json encoded event, which is passed to the skill service.

The **skill service** determines what actions to take in response to the JSON encoded event received from the skill interface. Upon reaching a decision the skill service returns a JSON encoded response to the skill interface for further processing. After processing, the speech response is sent back to the user through the Echo.





User-Skill Communication

## Key Ingredients for Building a Custom Skill

Now that we know, in principle, how to communicate with a custom skill, it is time to prepare the tools for building one.

The skill interface is implemented within the Amazon Alexa developers platform. This means that you'll need an Amazon Developers Services (ADS) account in order to build the skill and its interaction model.

---

*Note: If you don't have an ADS account, now is the right time to create one. It is easy and free.*

---

We are going to use the Amazon Web Service (AWS) as the skill service, so you will also need an AWS services account. In AWS, we'll write a function which will make decisions and create responses based on the received event.

---

*Note: Creating an AWS account may take some time, due to the required verification procedures. A debit/credit card info is necessary and the cost for setting up the account is 1\$. However, you'll get a bunch of free tiers which you can use forever.*

---

## Building A Custom Skill

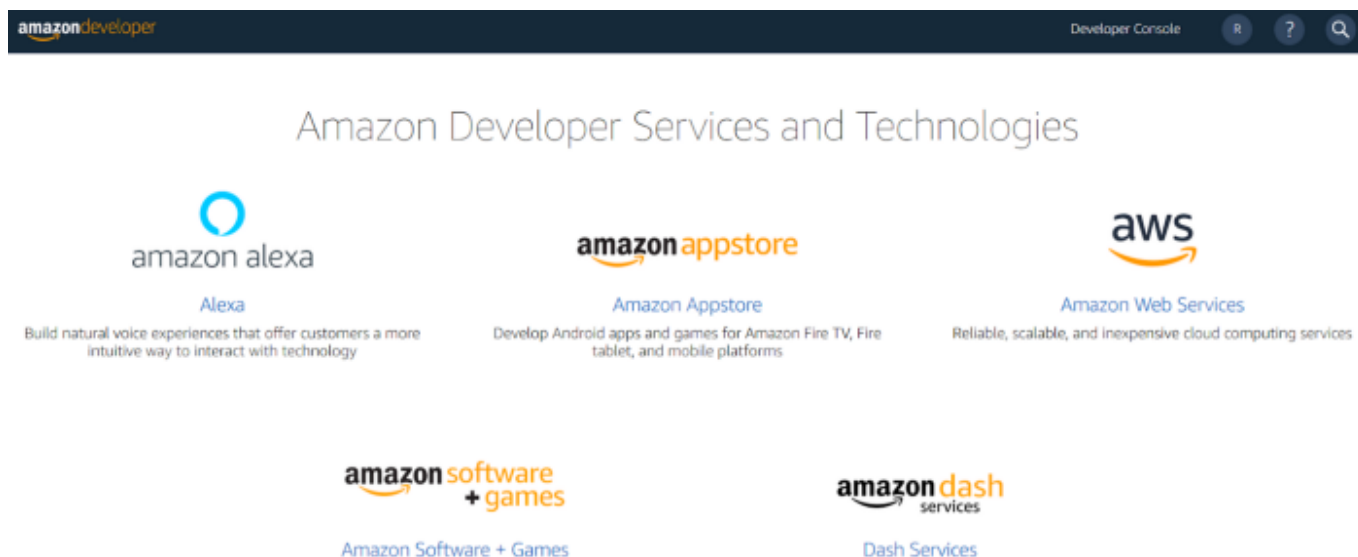
With your account set up, we are now going to build our custom Alexa skill.

Our custom amazon Alexa skill is going to do the following: when activated, it will provide a list of chess players and ask the user if he wants to hear more about any of the listed players.

If the user agrees and says something like “Tell me about Bobby Fisher” the skill will read a short biography corresponding to the chosen player if he is in the list. If the user decides that the topic is too boring, he can reply with “No” and the skill will close. The name of our skill will be “My Favorite Chess Player.”

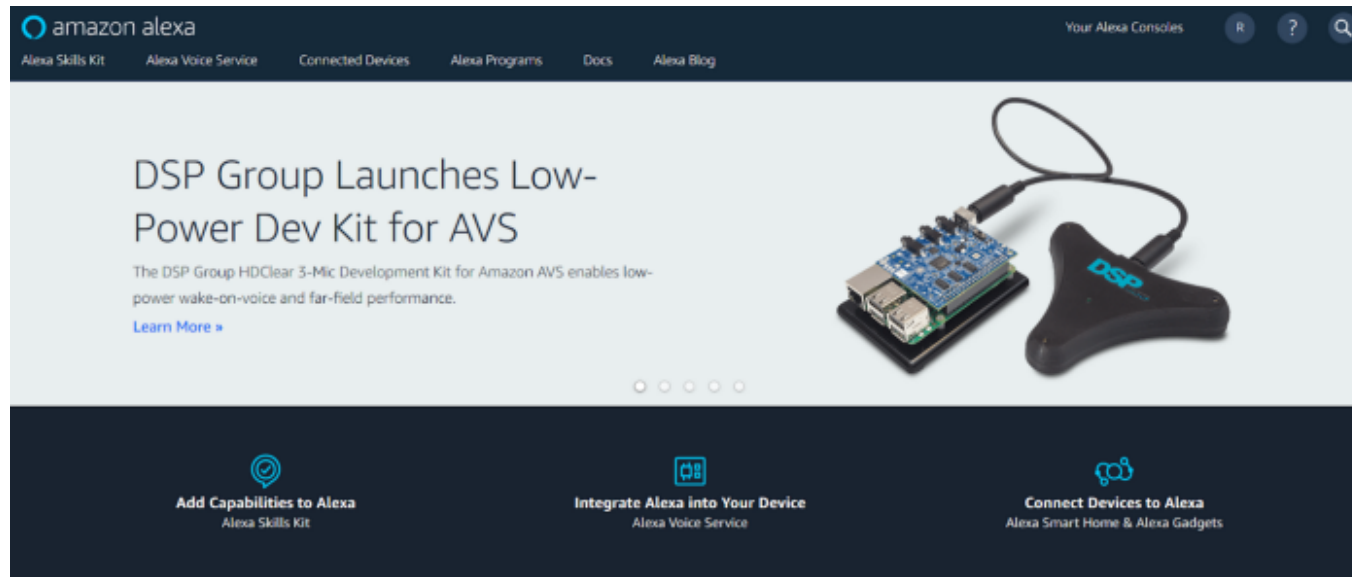
To build the My Favorite Chess Player skill:

- **Sign in to your ADS account and choose Alexa.**



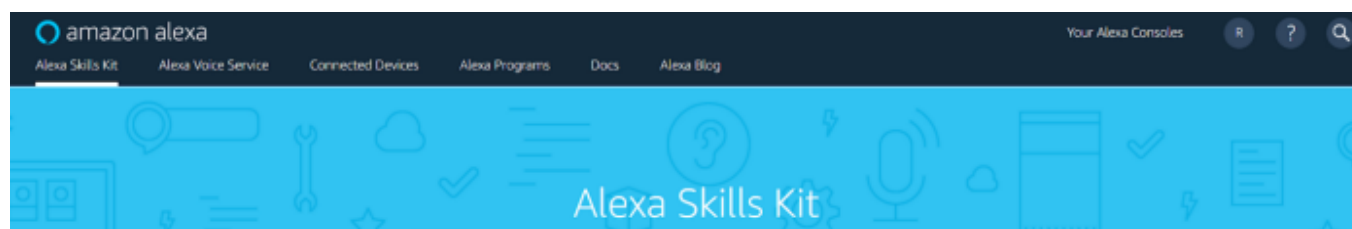
## Amazon Developers Services page

- Click the “Alexa Skills Kit”

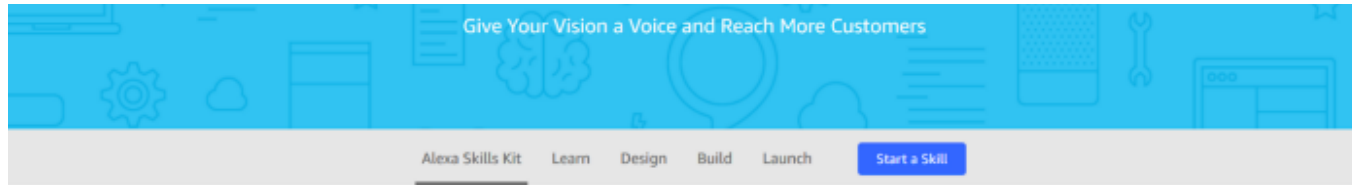


Amazon Alexa page

- Click the “Start a Skill” button.

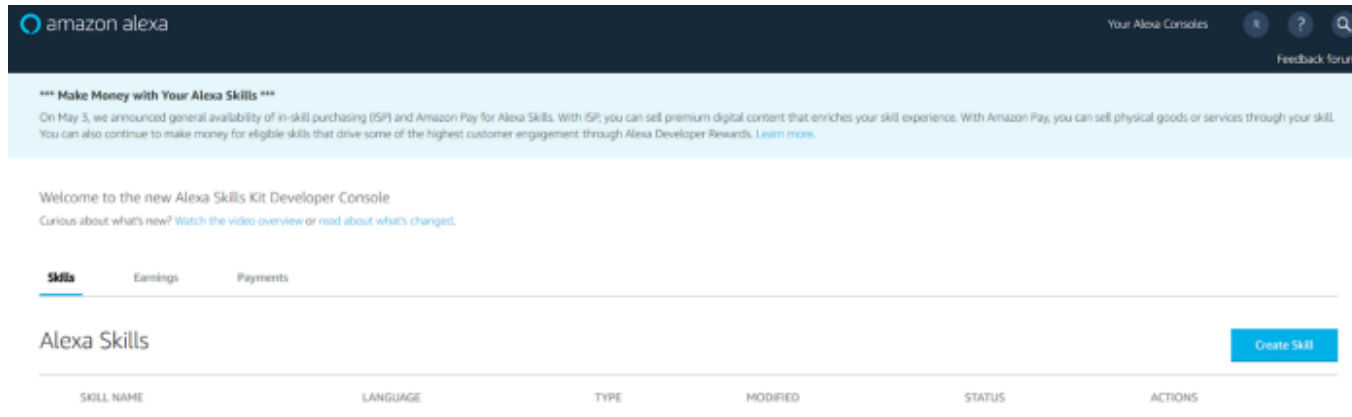






Alexa Skills Kit page

- Click the “Create Skill” button (this will open the “Create a new skill” form).



Alexa Skills Kit Developer Console page

- Enter “My Favorite Chess Player” in the “Skill name field”, and select “Custom”.
- Click the “Create a skill” button.

Create a new skill

Skill name  
My Favourite Chess Player 25/50 characters

Default language  
English (US)

More languages can be added to your skill after creation

Choose a model to add to your skill  
There are many ways to start building a skill. You can design your own custom model or start with a pre-built model. Pre-built models are interaction models that contain a package of intents and utterances that you can add to your skill.

Custom	Flash Briefing	Smart Home	Video
Design a unique experience for your users. A custom model enables you to create all of your skill's interactions.	Give users control of their news feed. This pre-built model lets users control what updates they listen to. "Alexa, what's in the news?"	Give users control of their smart home devices. This pre-built model lets users turn off the lights and other devices without getting up. "Alexa, turn on the kitchen lights"	Let users find and consume video content. This pre-built model supports content searches and content suggestions. "Alexa, play Interstellar"

Create a new skill form

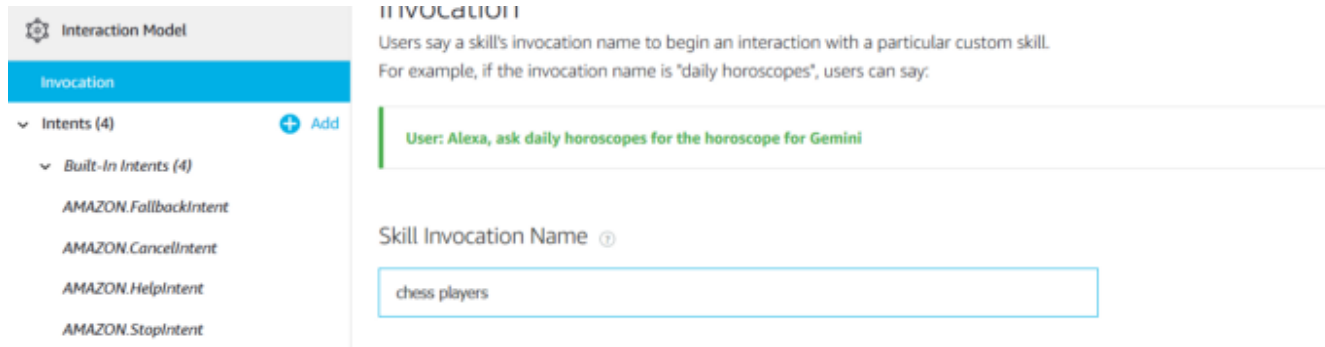
## Part 1: Building the Interaction Model

The Interaction model is a very important part of our skill. It contains the skill's **invocation name**, **intents** and **utterances**, etc., which are crucial for the interaction with the skill services.

First we are going to set up the invocation name for our skill:

- Select the “Invocation” tab.
- Enter “chess players” in the “Skill Invocation Name” field.

The invocation name is what we say to start the interaction with our skill.



Invocation name setup

Now we are going to set up the intents. For our skill we need:

1. an intent that handles the user speech request to hear the biography of a chess player,
2. an intent that handles the user speech request with which the user declines to interact with the skill or wants to stop the interaction,
3. an intent that handles the user speech request which doesn't match any of the utterances,
4. an intent that handles the user speech request for help.

Luckily, ASK has a large library of built-in intents so we will have to create only one custom intent.

*Note: Built-in intents have built-in utterances. It is possible to extend the number of existing utterances by adding new utterances. When a skill is created, four built-in intents are added to the Interaction model by default. These are: AMAZON.HelpIntent, AMAZON.CancelIntent, AMAZON.FallbackIntent and AMAZON.StopIntent.*

To create a custom intent we need to:

- Click the “Add” sign near the “Intents” tab.
- Select “Custom Intent”.
- Enter “playerBio” in the field below.
- Click the “Create custom intent” button.

**CUSTOM**

Interaction Model

Invocation

Intents (4) [+ Add](#)

▼ Built-In Intents (4)

- AMAZON.FallbackIntent
- AMAZON.CancelIntent
- AMAZON.HelpIntent
- AMAZON.StopIntent

Slot Types (0) [+ Add](#)

### Add Intent

An intent represents an action that fulfills a user's spoken request. [Learn more](#) about intents.

☒ Create custom intent [?](#)

[Create custom intent](#)

☐ Use an existing intent from Alexa's built-in library [?](#)

[Learn more](#) about using built-in intents.

[🔍](#)

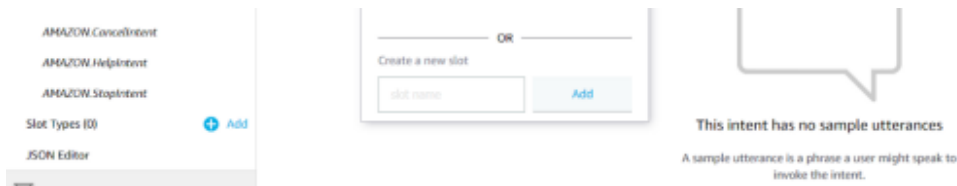
## Custom intent

The “playerBio” intent handles the user spoken request to hear the biography of a chess player. Now that we have created an intent, we must create the corresponding utterances. Since we are going to use a list of chess players, we’ll need to employ a custom slot for our utterances.

To create an utterance:

- Select the “playerBio” tab.
- Enter the utterance text in the “Sample Utterances” filed in the form of “tell me about {}”.
- Create a new slot in the window, which pops up when “{}” is entered, by entering “player” in the “Create a new slot” field and by clicking the “Add” button.
- Click the “plus” sign at the right end of the “Sample Utterances” field.





Utterances with a slot

We can create a few more utterances with the same “player” slot. In the “Sample Utterances” field we’ll enter the following utterances:

- “{player}”,
- “Who is {player}”,

and add them by clicking the “plus” sign.

*Note: Slots are a very powerful accessory for building a custom Alexa skill. For example, the statement “Tell me about {player}” means that the user can ask our skill about any chess player from the slot player. Furthermore, multiple slots provide the possibility of using dialogs in which the Alexa skill prompts the user to fill all slot values in order to fulfill the intent.*

The “player” slot is empty and we can’t populate it directly with the names of the chess players. However, we can associate the “player” slot to a slot type which contains the chess player names. ASK includes a large library of

slot types, but for our purposes we need to create a custom slot type. To create a custom slot type:

- Click the “Add” sign within the “Slot Types” tab.
- Select “Create custom slot type”.
- Enter “playerNames” in the field below.
- Click the “Create custom slot type” button.

**CUSTOM**

Interaction Model

Invocation

Intents (5) [Add](#)

- playerBio [Add](#)
- player [Add](#)

Built-In Intents (4)

- AMAZON.FallbackIntent
- AMAZON.CancelIntent
- AMAZON.HelpIntent
- AMAZON.StopIntent

Slot Types (0) [Add](#)

JSON Editor

## Add Slot Type

Slot types define how data in an intent slot is recognized and handled. All intent slots must be as

☒ Create custom slot type [?](#)

playerNames [Create custom slot type](#)

☐ Use an existing slot type from Alexa's built-in library [?](#)

[Learn more](#) about using built-in slot types.

Search built-ins [Search](#)

Name

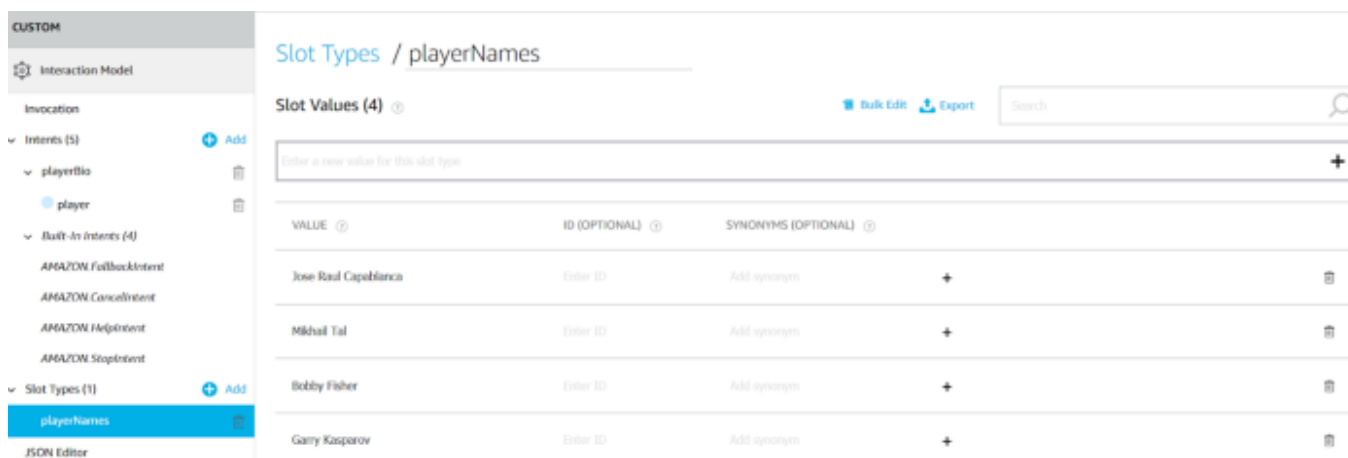
> [List Types](#)  
88 built-ins

These slot types each represent

Custom slot type

Let's populate the “playerNames” slot type with chess player names.

- Select “Slot Types” tab.
- Enter the names in the “Slot Values” field.
- Add names to the slot type by clicking the “plus” sign at the right end of the “Slot Values” field. Four names should be inserted: Bobby Fisher, Mikhail Tal, Jose Raul Capablanca and Garry Kasparov.

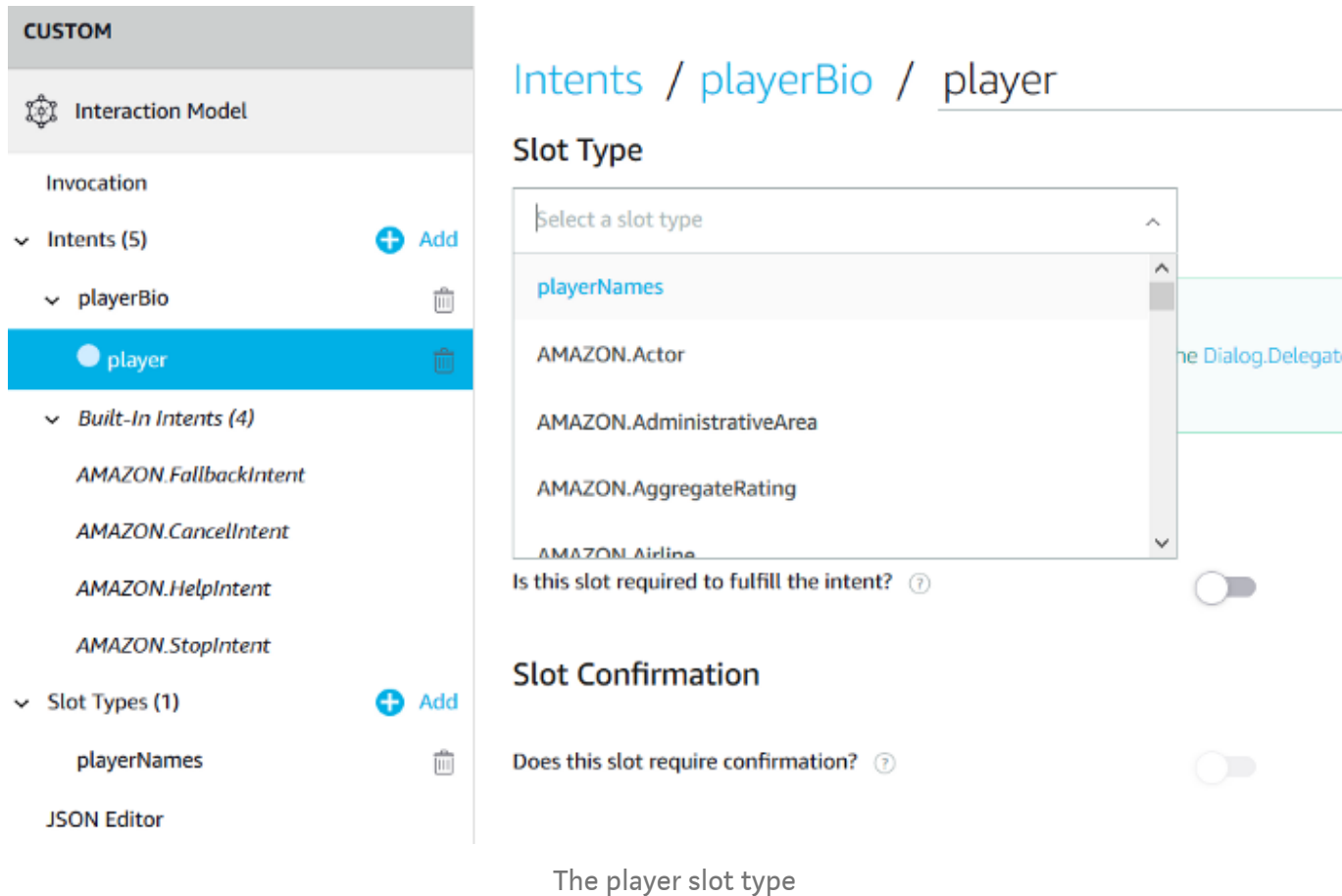


The playerNames slot type

Finally, the last step needed for setting up the “playerBio” intent is to assign the slot type to the player slot.



- Select the “player” tab.
- Choose the “playerNames” type from the “Slot Type” list.



The remaining intents are built-in intents.

The “AMAZON.NoIntent”, “AMAZON.CancelIntent” and “AMAZON.StopIntent” will handle the user speech request when the user declines to interact with the skill, or wants to stop the interaction.

The “AMAZON.FallbackIntent” will handle the user speech request which doesn’t match any of the utterances.

The “AMAZON.HelpIntent” will handle the user speech request for help.

The “AMAZON.NoIntent” is not included by default, like the other built-in intents, so we’ll have to add it.

To add the “AMAZON.NoIntent”:

- Click the “Add” sign near the “Intents” tab.
- Select “Use an existing intent from Alexa’s built-in library”.
- Enter “NoIntent” in the search field.
- Click the “+ Add intent” button.

#### Add Intent

An intent represents an action that fulfills a user's spoken request. [Learn more](#) about intents.

☐ Create custom intent ⓘ

Enter name for intent Create custom intent

☒ Use an existing intent from Alexa's built-in library ⓘ  
\*Learn more about using built-in intents.

1/145 built-ins

Name	Description
AMAZON.NoIntent	

+ Add Intent

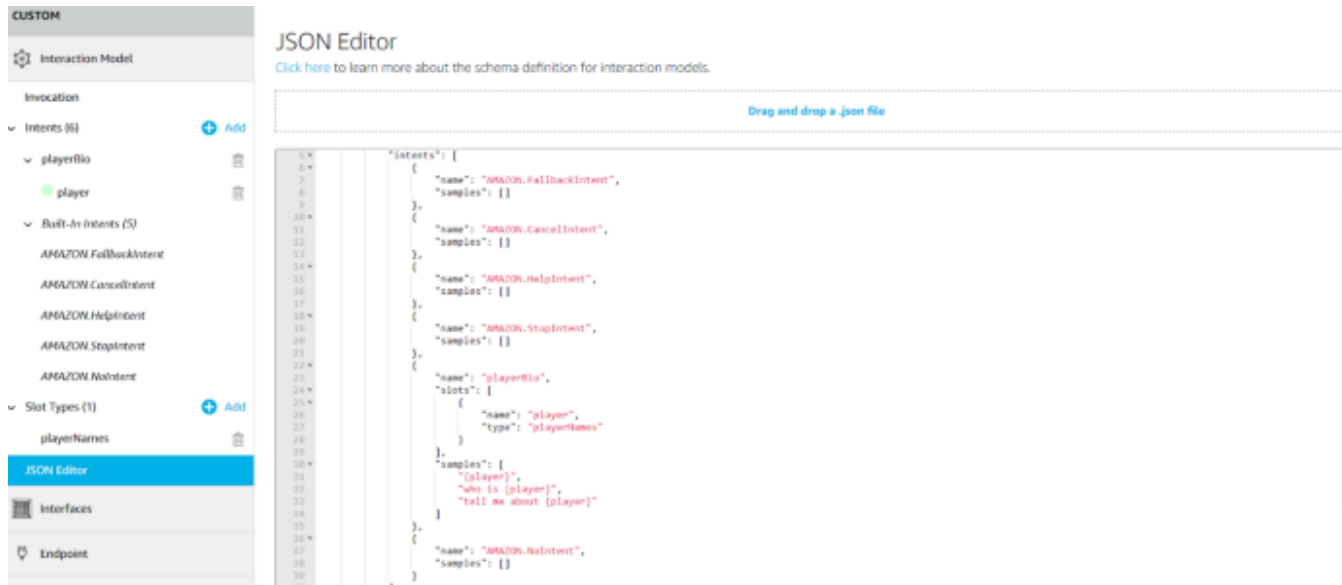
The built-in AMAZON.NoIntent

We should now save our interaction model by clicking the “Save Model” button, and build it by clicking the “Build Model” button.

That is it. But before we proceed further with building our skill, we should talk about the interaction model schema.

Every interaction model has its interaction model schema, which can be accessed by selecting the “JSON Editor” tab. The interaction model schema, written in a json file, contains intents, utterances, slots, i.e. everything that is implemented within the interaction model.

Consequently, if the interaction model changes the interaction model schema will change accordingly, and vice versa. This means that the entire interaction model can be built directly by writing a bit of json code in the “JSON Editor” or by uploading a json file.



JSON Editor with the Interaction model schema

In the text below you will find the interaction model schema which you can simply paste in the JSON Editor and create the My Favorite Chess Player interaction model.

```
{
  "interactionModel": {
    "languageModel": {
      "invocationName": "chess players",
      "intents": [
        {
          "name": "AMAZON.FallbackIntent",
          "samples": []
        },
        {
          "name": "AMAZON.CancelIntent",
          "samples": []
        }
      ]
    }
  }
}
```

```

    },
    {
      "name": "AMAZON.HelpIntent",
      "samples": []
    },
    {
      "name": "AMAZON.StopIntent",
      "samples": []
    },
    {
      "name": "playerBio",
      "slots": [
        {
          "name": "player",
          "type": "playerNames"
        }
      ],
      "samples": [
        "{player}",
        "who is {player}",
        "tell me about {player}"
      ]
    },
    {
      "name": "AMAZON.NoIntent",
      "samples": []
    }
  ],
  "types": [
    {
      "name": "playerNames",
      "values": [
        {
          "name": {
            "value": "Jose Raul Capablanca"
          }
        },
        {
          "name": {
            "value": "Mikhail Tal"
          }
        }
      ]
    }
  ]
}

```

```
    },  
    {  
      "name": {  
        "value": "Bobby Fisher"  
      }  
    },  
    {  
      "name": {  
        "value": "Garry Kasparov"  
      }  
    }  
  ]  
}  
]  
}
```

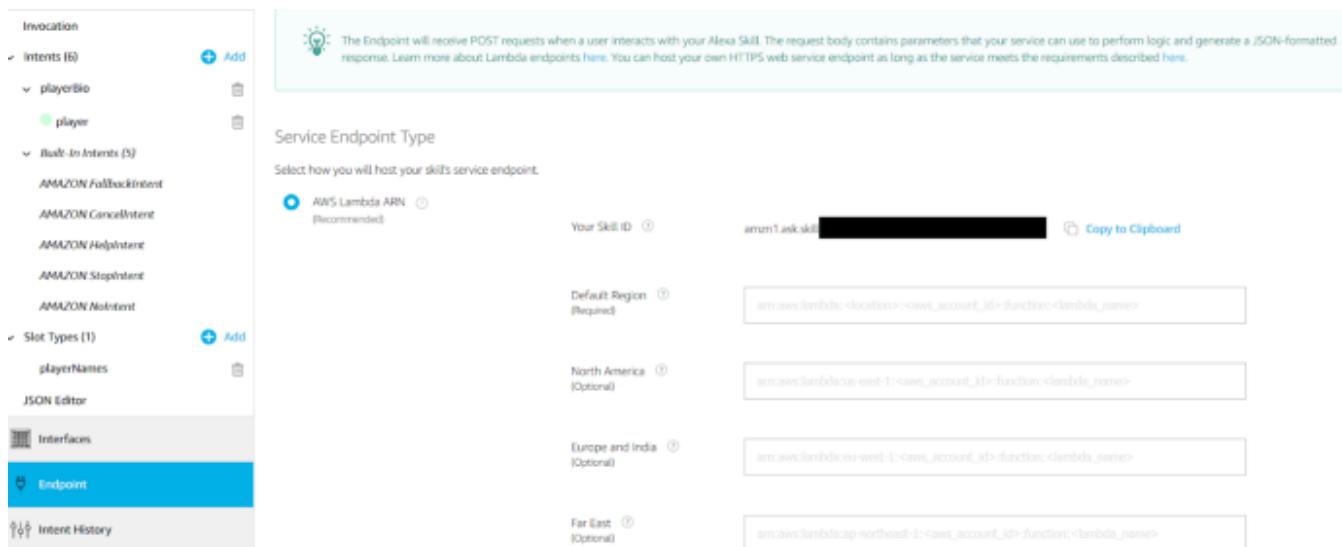
## Part 2: Choosing the Endpoints

The skill service can be either a user implemented web service or the AWS service. This is configured in the “Endpoints”. For our skill, we are going to use the AWS service, which communicates with the skill interface via the AWS Lambda function.

To choose the AWS service:

- Click the “Endpoints” tab.
- Select the “AWS Lambda function”.

In order to connect our skill interface with the AWS Lambda function we need to pass the skill the skill ID to the Lambda function. The skill ID is located in the “Endpoints”, and we will copy it by clicking “Copy to Clipboard”.



The Endpoints

## Part 3: Creating the AWS Lambda Function

To complete our skill we need to create and configure the AWS lambda function. This function receives events from and returns responses to the skill interface. Events contain the information about the way in which the user is interacting with the skill including the type of request user has triggered.

There are three main request types:

- ***LaunchRequest*** is sent within the event to the Lambda function when the user invokes the skill by saying its invocation name.
- ***IntentRequest*** is sent within the event to the Lambda function when the user interacts with the skill, i.e. when his speech request is mapped to an intent. The intent is also inscribed in the event.
- ***SessionEndedRequest*** is sent within the event to the Lambda function when the session ends, due to an error, when the user says “exit”, when the user doesn’t respond while the device is listening, or when the user says something that doesn’t match an intent defined in your skill interface while the device is listening.

The events are encoded in json format by the skill interface in accordance with the interaction model. The responses sent by the AWS service to the skill interface are also encoded in json format.

For our simple custom skill we’ll configure the Lambda function using Python, which is supported on the AWS services.

The response of our Lambda function is going to be in the following format:



```
{
  "body": {
    "version": "1.0",
    "response": {
      "outputSpeech": {
        "type": "PlainText",
        "text": ""
      },
      "card": {
        "type": "Simple",
        "title": "",
        "content": ""
      },
      "reprompt": {
        "outputSpeech": {
          "type": "PlainText",
          "text": ""
        }
      },
      "shouldEndSession": value
    }
  }
}
```

For this type of the response format, the output provided to the user will be:

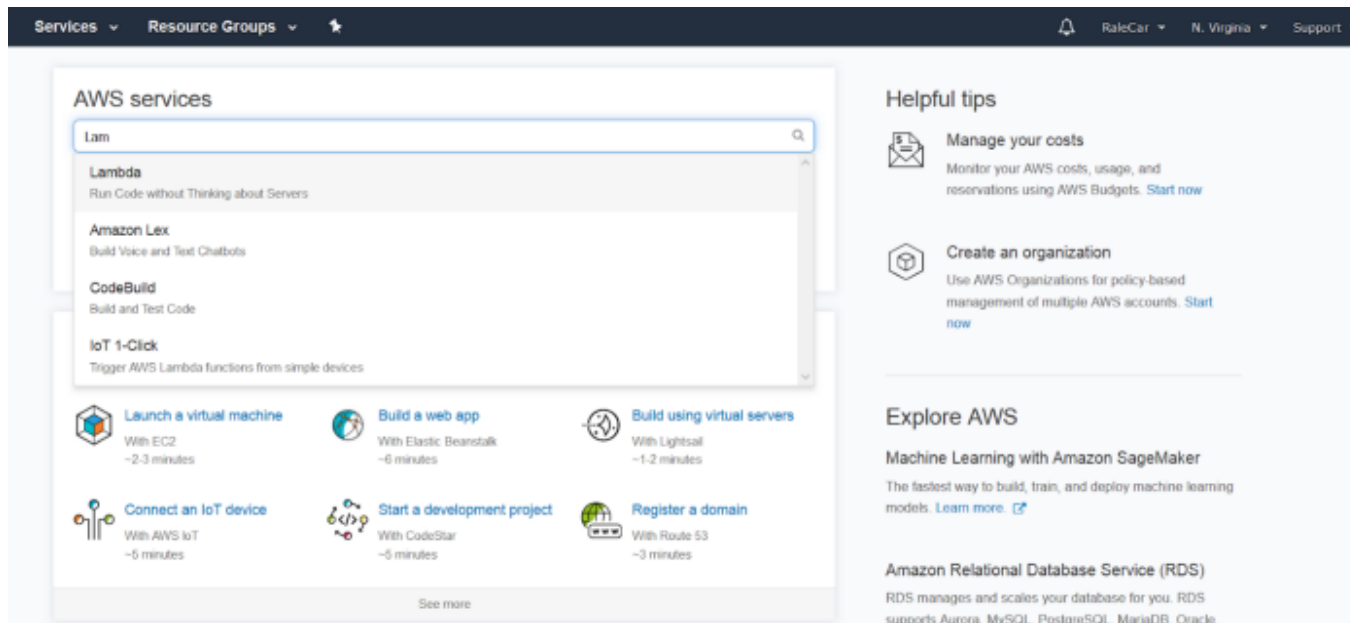
- A speech response, triggered by the “outputSpeech” object. The message written in the “text” field of this object is conveyed to the user via the Echo.
- A text on the device having the Alexa app such as a smartphone. This response is triggered by the “card” object and the message that

will be displayed is located in the “title” and “content” fields of this object.

- A re-prompt speech response from the skill, sent to the user through the Echo. This response is triggered by the “reprompt” object which contains the “outputSpeech” object.

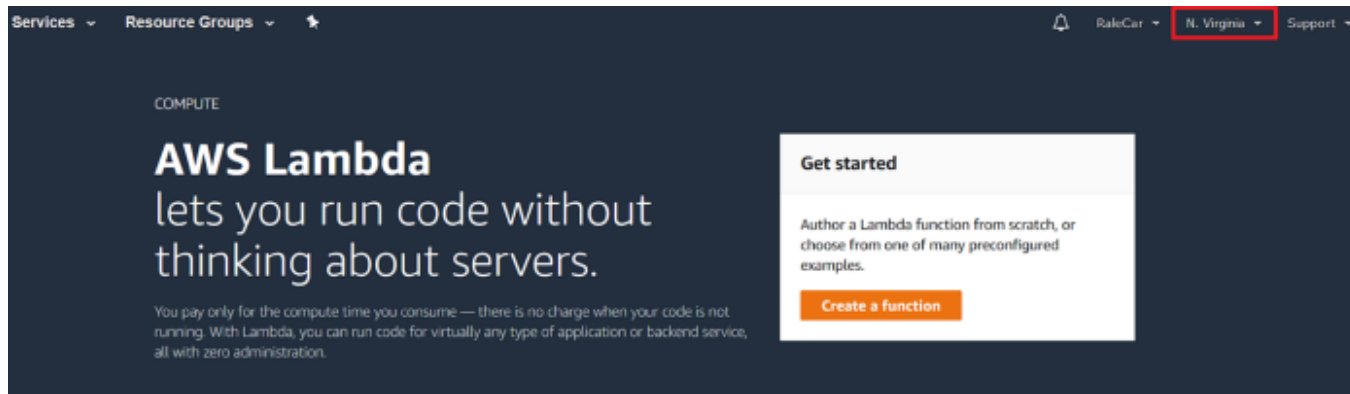
We are ready to create the AWS Lambda function using the AWS services.

- Log in to your account and choose the Lambda services.



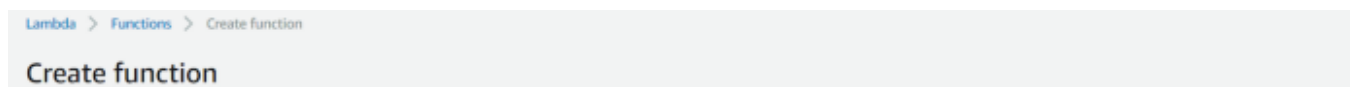
AWS home page

- In the upper right corner select “N. Virginia”, to make the Alexa Skill Kit trigger available to the AWS Lambda function.
- Click the “Create a function” button.



AWS Lambda service

- In the “Create function” form select “Author from scratch”
- In the “Name” field enter the the function name “chess\_lambda\_player”.
- From the “Runtime” list select Python 3.6.
- From the “Role” list select a “Custom Role”, and wait for the IAM manager to open.



The screenshot shows the 'Create function' form in the AWS Lambda console, specifically the 'Author from scratch' tab. At the top, there are three tabs: 'Author from scratch' (selected), 'Blueprints', and 'Serverless Application Repository'. The 'Author from scratch' tab contains the following fields:

- Name:** A text input field containing 'chess\_lambda\_player'.
- Runtime:** A dropdown menu showing 'Python 3.6'.
- Role:** A dropdown menu showing 'Create a custom role'.

Below the 'Role' dropdown, there is a note: 'The custom role creation experience will open in a new tab. Ensure that popups are enabled to create a custom role.'

AWS Lambda service — the Create function form

- In the IAM manager click on the “Allow” button and you will be redirected back to the “Create function” form.

The screenshot shows the 'Role Summary' page in the AWS IAM console. The page has a dark header with the AWS logo and navigation links. Below the header, there is a section titled 'Role Summary' with a 'Hide Details' link. The 'Role Summary' section contains the following information:

- Role Description:** Lambda execution role permissions
- IAM Role:** A dropdown menu showing 'lambda\_basic\_execution'.
- Policy Name:** A dropdown menu showing 'Create a new Role Policy'.

At the bottom of the 'Role Summary' section, there is a link: 'View Policy Document'.

[Cancel](#) [Allow](#)

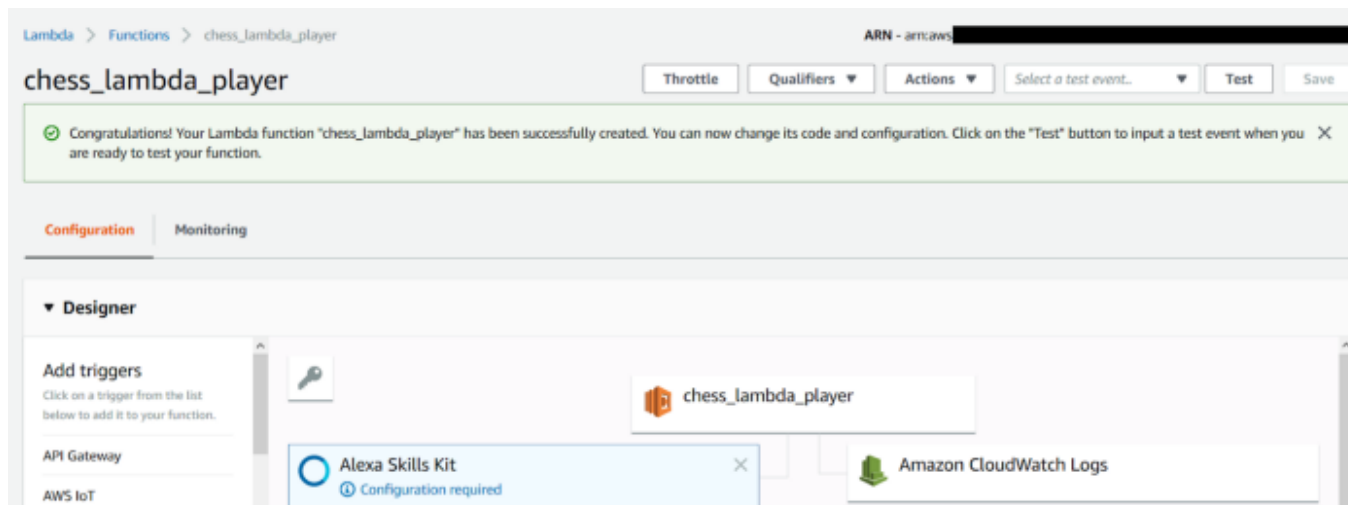
## IAM manager

- Click the “Create function” button, located in the lower-right corner of the “Create function” form.

Now that the AWS Lambda function has been created, we need to set the function triggers and write the code. If everything was done correctly, we should have the “chess\_lambda\_player” form in our browser.

To set the function trigger:

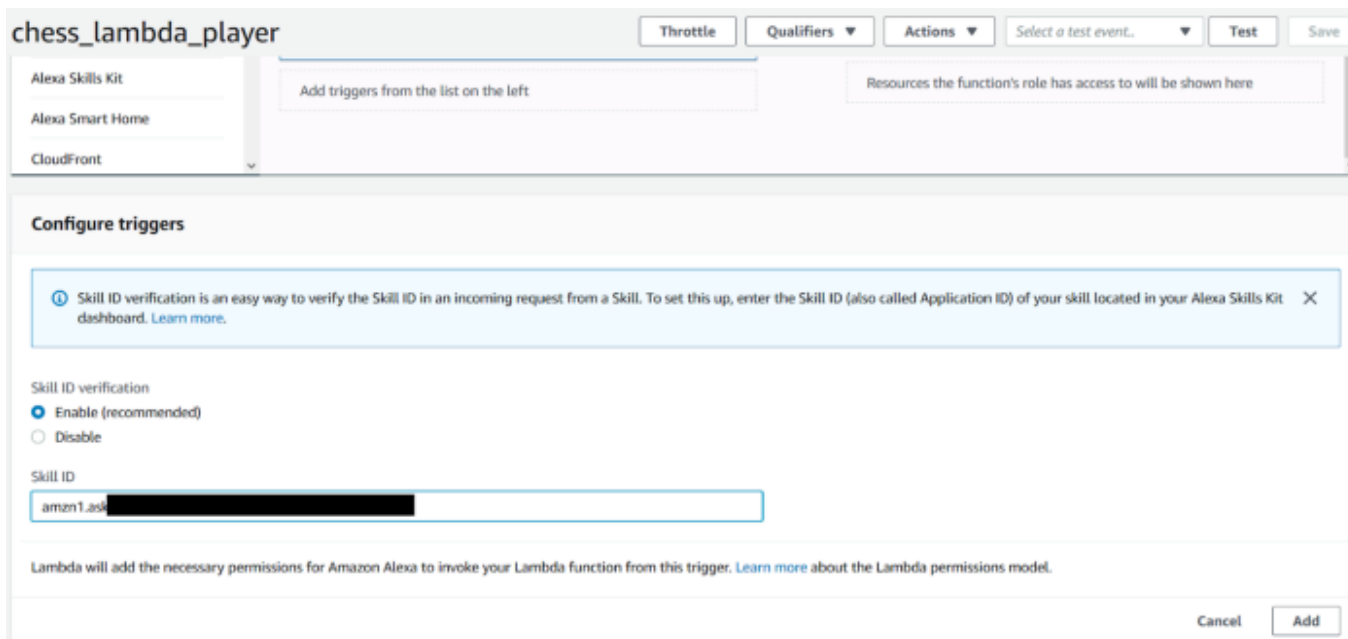
- Select the “Alexa Skills Kit” in the “Add triggers” list.





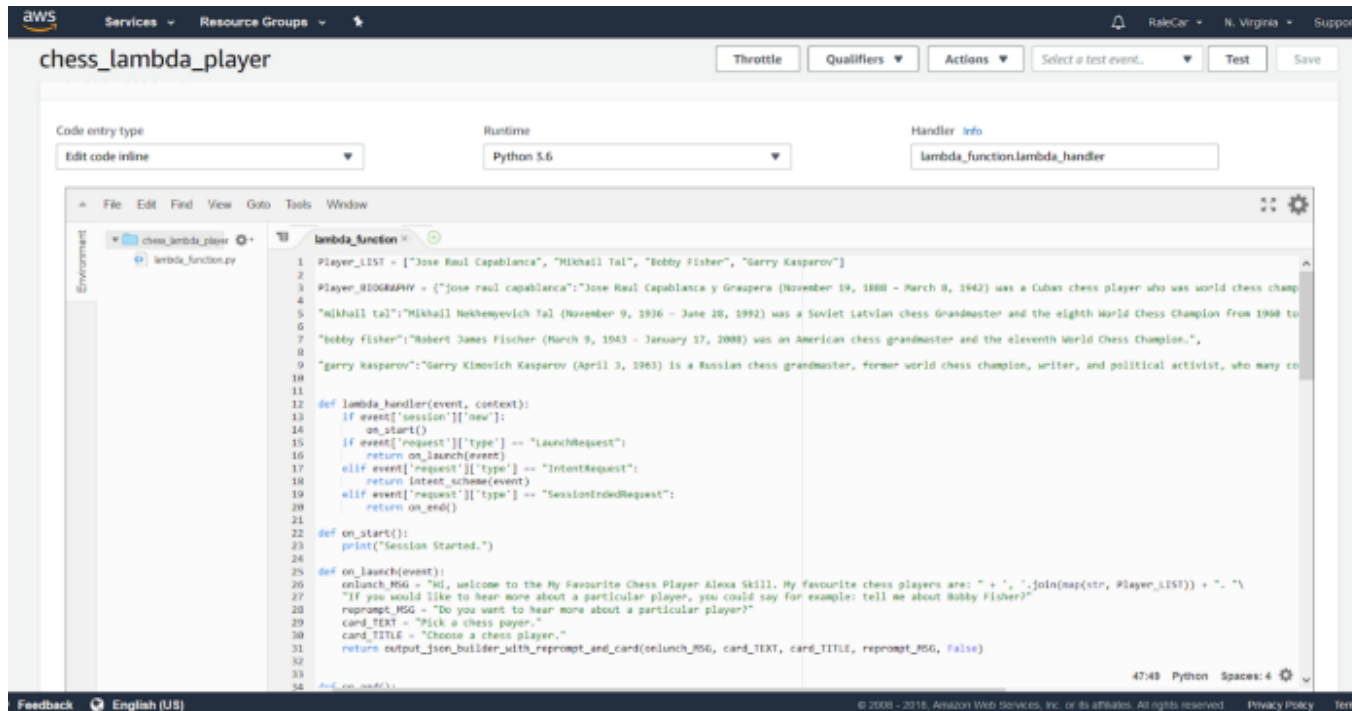
AWS Lambda service — chess\_lambda\_player form showing the active "Alexa Skills Kit" panel with notification "Configuration required"

- Scroll down the page and in the "Skill ID" field paste the skill ID that you have copied earlier, from the "Endpoints".
- Click the "Add" button located at the bottom of the page.



"chess\_lambda\_player" form — the "Skill ID" field

- The function code is accessed by clicking on the “chess\_lambda\_player” panel. Scroll down the page to find the editor where the code is located.



“chess\_lambda\_player” form — Lambda function code

- Delete the existing code and paste the following code in the editor:

```

#-----Part1-----
# In this part we define a list that contains the player names, and
# a dictionary with player biographies

```

```

Player_LIST = ["Jose Raul Capablanca", "Mikhail Tal", "Bobby Fisher",
"Garry Kasparov"]

Player_BIOGRAPHY = {"jose raul capablanca":"Jose Raul Capablanca y
Graupera (November 19, 1888 - March 8, 1942) was a Cuban chess player
who was world chess champion from 1921 to 1927.",

"mikhail tal":"Mikhail Nekhemyevich Tal (November 9, 1936 - June 28,
1992) was a Soviet Latvian chess Grandmaster and the eighth World
Chess Champion from 1960 to 1961.",

"bobby fisher":"Robert James Fischer (March 9, 1943 - January 17,
2008) was an American chess grandmaster and the eleventh World Chess
Champion.",

"garry kasparov":"Garry Kimovich Kasparov (April 3, 1963) is a
Russian chess grandmaster, former world chess champion, writer, and
political activist, who many consider to be the greatest chess player
of all time"}

#-----Part2-----
# Here we define our Lambda function and configure what it does when
# an event with a Launch, Intent and Session End Requests are sent. #
The Lambda function responses to an event carrying a particular
# Request are handled by functions such as on_launch(event) and
# intent_scheme(event).

def lambda_handler(event, context):
    if event['session']['new']:
        on_start()
    if event['request']['type'] == "LaunchRequest":
        return on_launch(event)
    elif event['request']['type'] == "IntentRequest":
        return intent_scheme(event)
    elif event['request']['type'] == "SessionEndedRequest":
        return on_end()

#-----Part3-----
# Here we define the Request handler functions

```



```
def on_start():
    print("Session Started.")

def on_launch(event):
    onlunch_MSG = "Hi, welcome to the My Favourite Chess Player Alexa Skill. My favourite chess players are: " + ', '.join(map(str, Player_LIST)) + ". \"\n    \"If you would like to hear more about a particular player, you could say for example: tell me about Bobby Fisher?\"
    reprompt_MSG = "Do you want to hear more about a particular player?"
    card_TEXT = "Pick a chess payer."
    card_TITLE = "Choose a chess player."
    return output_json_builder_with_reprompt_and_card(onlunch_MSG, card_TEXT, card_TITLE, reprompt_MSG, False)

def on_end():
    print("Session Ended.")

#-----Part3.1-----
# The intent_scheme(event) function handles the Intent Request.
# Since we have a few different intents in our skill, we need to
# configure what this function will do upon receiving a particular
# intent. This can be done by introducing the functions which handle
# each of the intents.

def intent_scheme(event):

    intent_name = event['request']['intent']['name']

    if intent_name == "playerBio":
        return player_bio(event)
    elif intent_name in ["AMAZON.NoIntent", "AMAZON.StopIntent", "AMAZON.CancelIntent"]:
        return stop_the_skill(event)
    elif intent_name == "AMAZON.HelpIntent":
        return assistance(event)
    elif intent_name == "AMAZON.FallbackIntent":
        return fallback_call(event)
```

```

#-----Part3.1.1-----
# Here we define the intent handler functions

def player_bio(event):
    name=event['request']['intent']['slots']['player']['value']
    player_list_lower=[w.lower() for w in Player_LIST]
    if name.lower() in player_list_lower:
        reprompt_MSG = "Do you want to hear more about a particular
player?"
        card_TEXT = "You've picked " + name.lower()
        card_TITLE = "You've picked " + name.lower()
        return
    output_json_builder_with_reprompt_and_card(Player_BIOGRAPHY[name.lower()
r()], card_TEXT, card_TITLE, reprompt_MSG, False)
    else:
        wrongname_MSG = "You haven't used the full name of a player.
If you have forgotten which players you can pick say Help."
        reprompt_MSG = "Do you want to hear more about a particular
player?"
        card_TEXT = "Use the full name."
        card_TITLE = "Wrong name."
        return
    output_json_builder_with_reprompt_and_card(wrongname_MSG, card_TEXT,
card_TITLE, reprompt_MSG, False)

def stop_the_skill(event):
    stop_MSG = "Thank you. Bye!"
    reprompt_MSG = ""
    card_TEXT = "Bye."
    card_TITLE = "Bye Bye."
    return output_json_builder_with_reprompt_and_card(stop_MSG,
card_TEXT, card_TITLE, reprompt_MSG, True)

def assistance(event):
    assistance_MSG = "You can choose among these players: " + ',
'.join(map(str, Player_LIST)) + ". Be sure to use the full name when
asking about the player."
    reprompt_MSG = "Do you want to hear more about a particular
player?"
    card_TEXT = "You've asked for help."

```

```

        card_TITLE = "Help"
        return output_json_builder_with_reprompt_and_card(assistance_MSG,
card_TEXT, card_TITLE, reprompt_MSG, False)

def fallback_call(event):
    fallback_MSG = "I can't help you with that, try rephrasing the
question or ask for help by saying HELP."
    reprompt_MSG = "Do you want to hear more about a particular
player?"
    card_TEXT = "You've asked a wrong question."
    card_TITLE = "Wrong question."
    return output_json_builder_with_reprompt_and_card(fallback_MSG,
card_TEXT, card_TITLE, reprompt_MSG, False)

#-----Part4-----
# The response of our Lambda function should be in a json format.
# That is why in this part of the code we define the functions which
# will build the response in the requested format. These functions
# are used by both the intent handlers and the request handlers to
# build the output.

def plain_text_builder(text_body):
    text_dict = {}
    text_dict['type'] = 'PlainText'
    text_dict['text'] = text_body
    return text_dict

def reprompt_builder(repr_text):
    reprompt_dict = {}
    reprompt_dict['outputSpeech'] = plain_text_builder(repr_text)
    return reprompt_dict

def card_builder(c_text, c_title):
    card_dict = {}
    card_dict['type'] = "Simple"
    card_dict['title'] = c_title
    card_dict['content'] = c_text
    return card_dict

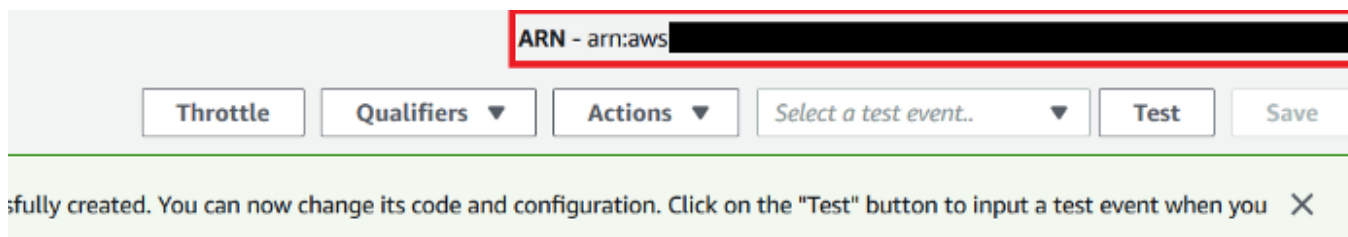
```

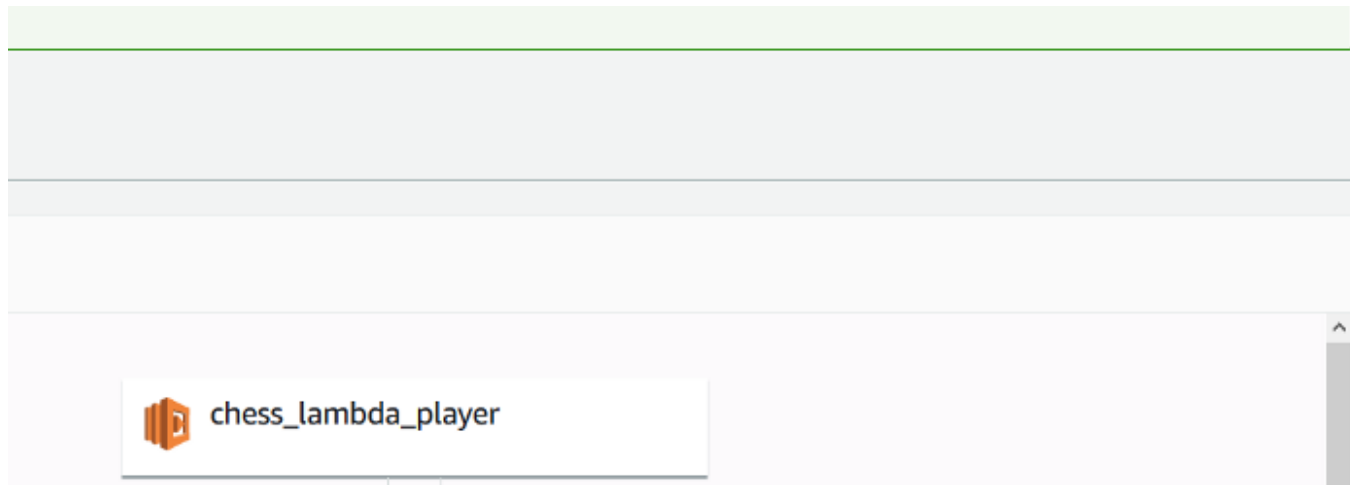
```
def response_field_builder_with_reprompt_and_card(outputSpeech_text,
card_text, card_title, reprompt_text, value):
    speech_dict = {}
    speech_dict['outputSpeech'] =
plain_text_builder(outputSpeech_text)
    speech_dict['card'] = card_builder(card_text, card_title)
    speech_dict['reprompt'] = reprompt_builder(reprompt_text)
    speech_dict['shouldEndSession'] = value
    return speech_dict

def output_json_builder_with_reprompt_and_card(outputSpeech_text,
card_text, card_title, reprompt_text, value):
    response_dict = {}
    response_dict['version'] = '1.0'
    response_dict['response'] =
response_field_builder_with_reprompt_and_card(outputSpeech_text,
card_text, card_title, reprompt_text, value)
    return response_dict
```

- Click the “Save” button to save the Lambda function, and copy the Amazon Resource Name (ARN) displayed in the upper-right corner.

The ARN is used to complete the connection between the skill interface and the Lambda function, which is why our next step would be to pass it to the skills “Endpoints”.





Amazon Resource Name (ARN)

- Go back to the Alexa developers portal and select “Endpoints”. Paste the ARN in the “Default Region” field and click the “Save Endpoints” button.

A screenshot of the Alexa Developer Console 'Endpoints' page. At the top is a 'Save Endpoints' button. Below it is a section titled 'Endpoint' with a light blue background and a lightbulb icon. The text explains that the endpoint receives POST requests and provides links for more information. Below this is the 'Service Endpoint Type' section, which asks how to host the skill's service endpoint. The 'AWS Lambda ARN (Recommended)' option is selected. Below this, there are two fields: 'Your Skill ID' with the value 'amzn1.ask...' and a 'Copy to Clipboard' button, and 'Default Region (Required)' with the value 'arn:aws:lambda:...'.

North America  
(Optional)

arn:aws:lambda:us-east-1:<aws\_account\_id>:function:<lambda\_name>

Connecting the AWS Lambda and the skill

The My Favorite Chess Player custom Alexa skill is completed and ready for testing!

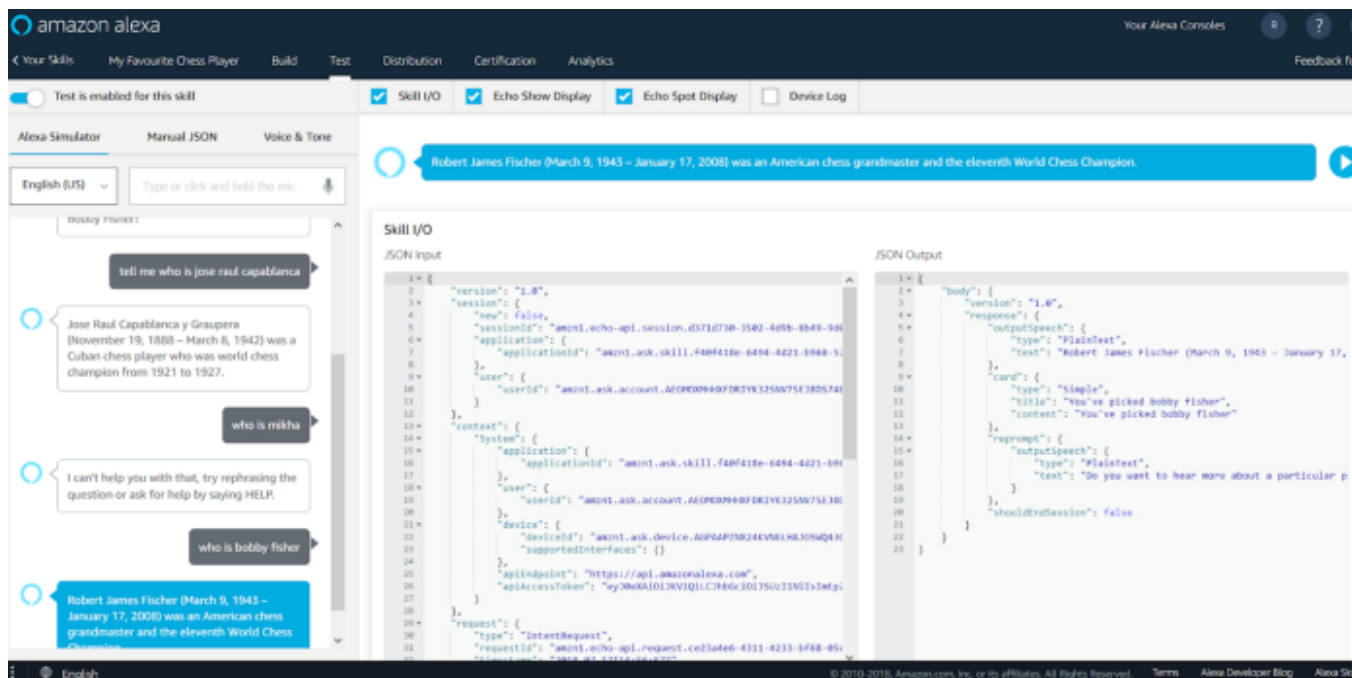
## Testing the Skill

Finally, we can test our custom Alexa skill using:

1. any type of Alexa device such as the Echo or Echo Dot,
2. the Echo Simulator Community Edition at Echoism.io,
3. the service simulator under the Test tab.

To test the skill in the service simulator:

- Click the “Test” tab.
- Toggle the Test button on.
- Write what you want to say in the empty field or hold the microphone and speak.



Service simulator test of the My Favorite Chess Players skill

Besides the speech which you will hear and the text which will be displayed on your screen, the service simulator also displays the JSON input (json event sent by the skill interface to the Lambda function) and the JSON output (json response sent by the Lambda function to the skill interface).

If you are satisfied with your skill and are sure that it works properly, you can publish it to the AWS Alexa.

**That is it! You now have a working custom Amazon Alexa Skill.**

Let's do a quick recap.

First, we have learned that an Alexa skill is voice-driven app for Alexa which preforms a specific task for the user at his request.

We then learned that the user request (to the skill) is handled first by the skill interface, using the interaction model, and then by the skill service that forms the response for the user, using the Lambda function.

Finally, by building our own custom skill we have learned the very basics of practical skill building and scratched the surface of the Amazon Alexa developers console and AWS Lambda services.

Have fun while making your own custom skills!

—

*Special thanks to those who reviewed and commented on early drafts of this post: Aleksa Miladinovic, William Wickey.*



# crowdbotics



## Building A Web Or Mobile App?

*Crowdbotics is the fastest way to build, launch and scale an application.*

*Developer? Try out the **Crowdbotics App Builder** to quickly scaffold and deploy apps with a variety of popular frameworks.*

*Busy or non-technical? Join hundreds of happy teams building software with Crowdbotics PMs and expert developers. Scope timeline and cost with Crowdbotics **Managed App Development** for free.*

[Tutorial](#)[Alexa](#)[Alexa Skills](#)[Voice Assistant](#)[Development](#)

### Discover Medium

Welcome to a place where words matter. On Medium, smart voices and original ideas take center stage - with no ads in sight. [Watch](#)

### Make Medium yours

Follow all the topics you care about, and we'll deliver the best stories for you to your homepage and inbox. [Explore](#)

### Become a member

Get unlimited access to the best stories on Medium — and support writers while you're at it. Just \$5/month. [Upgrade](#)

---

**Medium**[About](#)[Help](#)[Legal](#)

