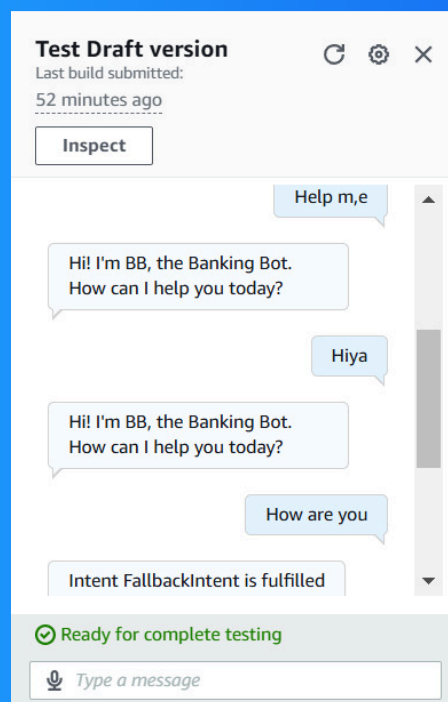


Build a Chatbot with Amazon Lex



Introducing Today's Project!

What is Amazon Lex?

Amazon Lex is a tool from AWS for building chatbots that understand and respond to user messages. It's useful since it enables businesses to easily create chatbots for customer support. It also uses AI/ML capabilities in classifying user's intent.

How I used Amazon Lex in this project

In this project, I used Amazon Lex to create a BankerBot, a chatbot that greets users & provides assistance. It also returns error messages when it doesn't understand a user's intent, guiding them to clarify their requests for smoother interactions.

One thing I didn't expect in this project was...

One thing I didn't expect in this project was how easy and quick it is to build a chatbot, along with the impressive capabilities of Amazon Lex.

This project took me...

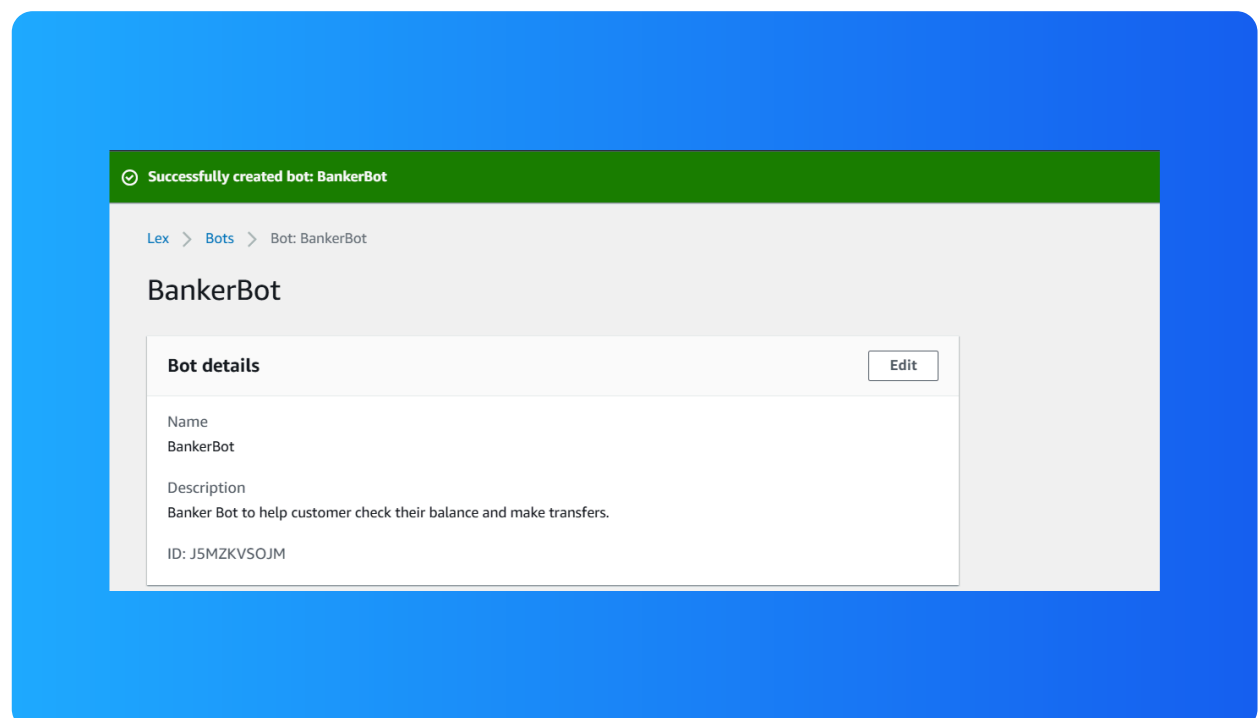
The whole process took me just about 40 minutes, and I was impressed by how powerful and user-friendly the platform is.

Setting up a Lex chatbot

I created my chatbot from scratch with Amazon Lex. Setting it up took me about 4 minutes.

While creating my chatbot, I also created a role with basic permissions because Amazon Lex needs the permission to call other AWS services like AWS Lambda and Polly which are very useful in this project.

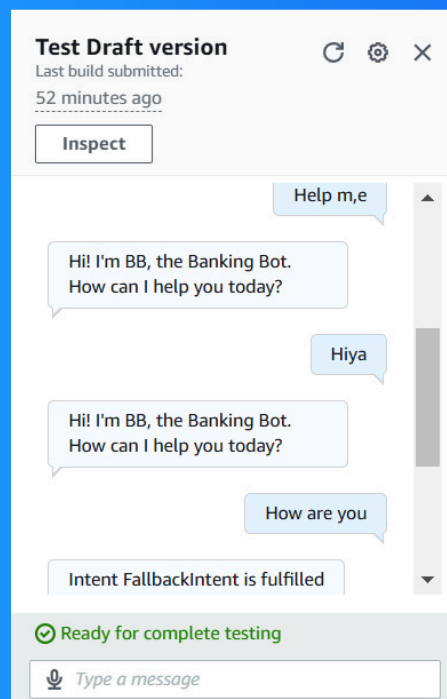
In terms of the intent classification confidence score, I kept the default value of 0.40. This means my chatbot must be at least 40% confident in understanding the user's query to respond; otherwise, error message will display if below this level.



Intents

Intents are the goals users aim to accomplish with the chatbot, like checking a balance or booking a flight. In Amazon Lex, you define and categorize intents so one chatbot can manage multiple related requests.

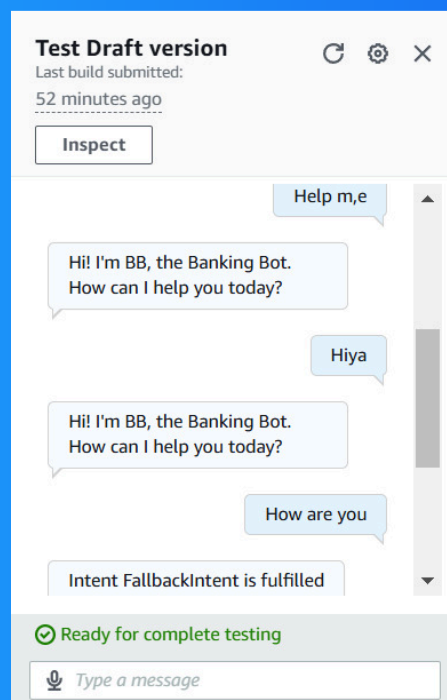
I created my first intent, Welcome Intent, to greet the user when they say phrases like "Hello," "Hi," or "I need help," which I included in the sample utterances. I also set up a closing response to define how the chatbot replies.



FallbackIntent

I launched and tested my chatbot, which could respond successfully to the sample utterances and similar ones like "Hiya" or even typos like "help m,e."

My chatbot returned the error message Intent FallbackIntent is fulfilled when I entered phrases like "Good morning." This error occurred because my chatbot couldn't understand the intent of the phrase.



Configuring FallbackIntent

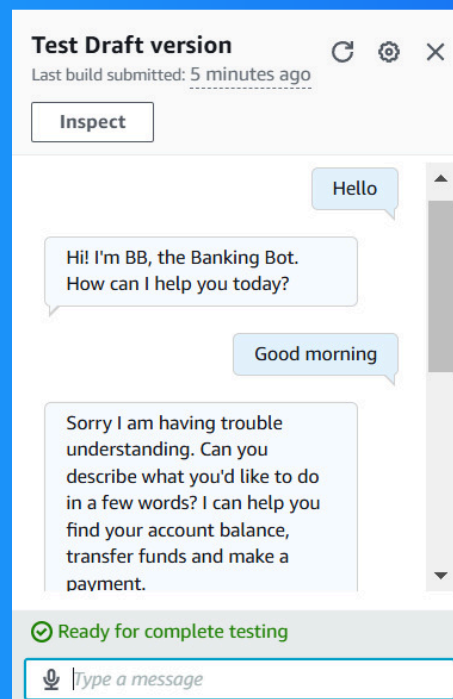
FallbackIntent is a default intent in every chatbot that gets triggered when the chatbot doesn't recognize the user's intent/goal.

I wanted to configure FallbackIntent because the default closing response wasn't very clear for the user. This way, I can provide a more helpful message when the chatbot doesn't understand something.

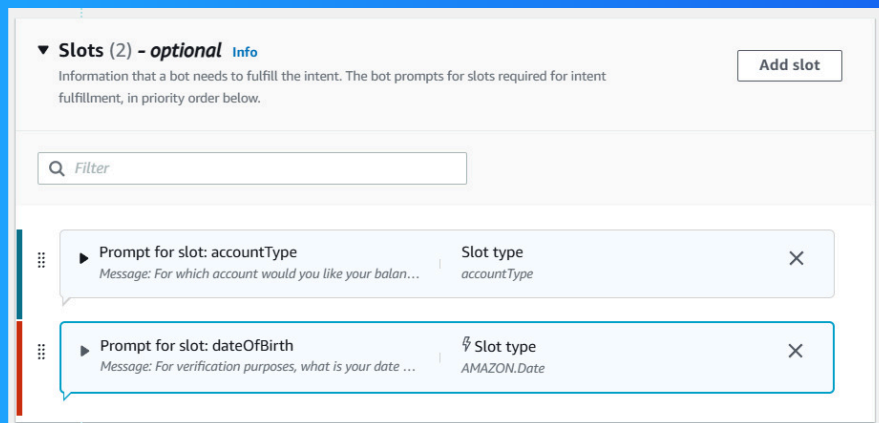
Variations

To configure FallbackIntent, I had to customize my own closing response in the intent's set up page.

I also added variations! What this means for an end user is they'll receive different response styles from my chatbot, making interactions feel more natural and dynamic. It helps keep conversations engaging by avoiding repetitive replies.



Build a Chatbot with Custom Slots



Introducing Today's Project!

What is Amazon Lex?

Amazon Lex is a tool from AWS for building chatbots that understand and respond to user messages. It's useful since it enables businesses to easily create chatbots for customer support. It also uses AI/ML capabilities in classifying user's intent.

How I used Amazon Lex in this project

In today's project, I used Amazon Lex to build a chatbot named BankerBot. It helps users check their account balances by recognizing requests through custom slots and intents, making interactions efficient and user-friendly.

One thing I didn't expect in this project was...

One thing I didn't expect in this project was how easy it was to set up custom slots in Amazon Lex. In this proj, it streamlined the process of capturing different bank account types, making the chatbot more efficient in understanding user requests.

This project took me...

This project took me about 2 hours, including the documentation, and it was well worth the time invested in building it.

Slots

Slots are details a chatbot needs to fulfill a request, like blanks in a form. For example, if the intent is to book a table, the chatbot needs specifics such as the restaurant name, date, time, and number of people.

In this project, I created a custom slot type to capture specific details unique to the user's request. This allowed the chatbot to better understand and respond accurately.

This slot type has restricted slot values, which means the chatbot will only accept specific, preset answers. For example, I used the slot type "accountType" to ensure that only the options I defined are considered valid account types.

Slot type values

Modify the list of values used to train the machine learning model to recognize values for a slot.

Checking	Tab or ; or enter return for new value	×
Savings	Tab or ; or enter return for new value	×
Credit	Tab or ; or enter return for new value	×
	credit card × visa × mastercard ×	
	amex × american express ×	

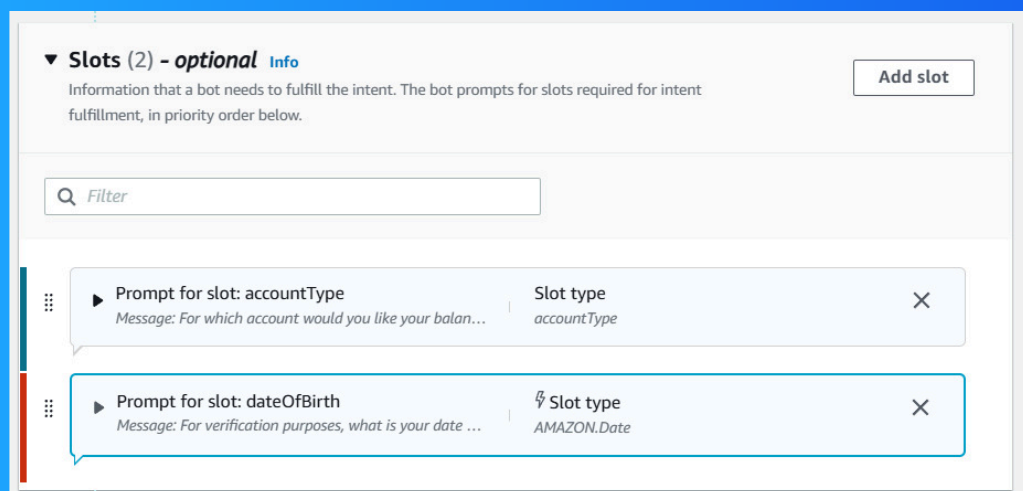
Tab or ; or enter return for new value

Maximum 140 characters. Valid characters: A-Z, a-z, 0-9, @, #, \$

☐ Use slot values as custom vocabulary [Info](#)

Connecting slots with intents

I associated my custom slot with CheckBalance, enabling Amazon Lex to automatically recognize and fill in the {accountType} slot from user input. This way, if a user specifies an account type, Lex won't need to ask again, streamlining the process.



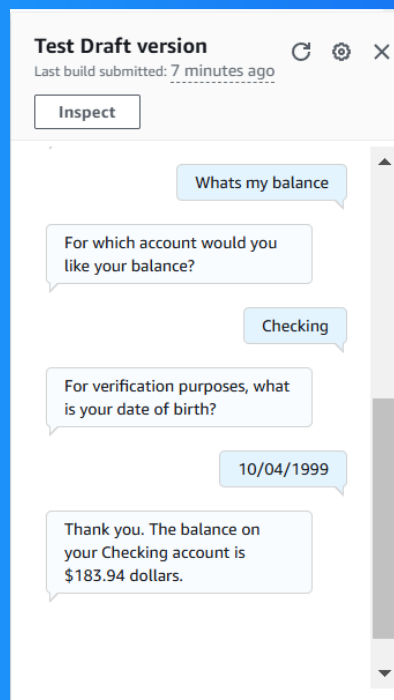
Slot values in utterances

I included slot values in some of the utterances by adding the slot name directly. For example, "What's the balance in my {accountType} account?" lets the bot capture the account type right away, making the conversation quicker and easier.

By adding custom slots in utterances, the user experience is improved as the bot can automatically recognize the account type the user wants to check without needing to ask repeatedly. This streamlines the conversation and saves the user time.

The screenshot displays the Microsoft Bot Framework DevOps console interface. On the left, the 'Inspect' panel shows the 'Summary' tab selected, displaying the intent 'CheckBalance' and the slots 'accountType' (Savings) and 'dateOfBirth' (1999-10-04). The 'Test Draft version' panel on the right shows a conversation flow: 'What's the balance in my savings account?' followed by 'For verification purposes, what is your date of birth?' and the user input '10/04/1999'. The final message is 'Intent CheckBalance is fulfilled'. A green checkmark indicates 'Ready for complete testing'.

Connect a Chatbot with Lambda



Introducing Today's Project!

What is Amazon Lex?

Amazon Lex is a tool from AWS for building chatbots that understand and respond to user messages. It's useful since it enables businesses to easily create chatbots for customer support. It also uses AI/ML capabilities in classifying user's intent.

How I used Amazon Lex in this project

In today's project, I used Amazon Lex to create a BankerBot chatbot that helps users check their bank balance. I set up intents, defined custom slots for account types, and connected it to Lambda functions to retrieve balance information dynamically.

One thing I didn't expect in this project was...

I was surprised at how fast I could build a working chatbot with Amazon Lex and connect it to AWS Lambda for real-time data. I learned that Lambda makes it easy to access database information, improving the chatbot's functionality.

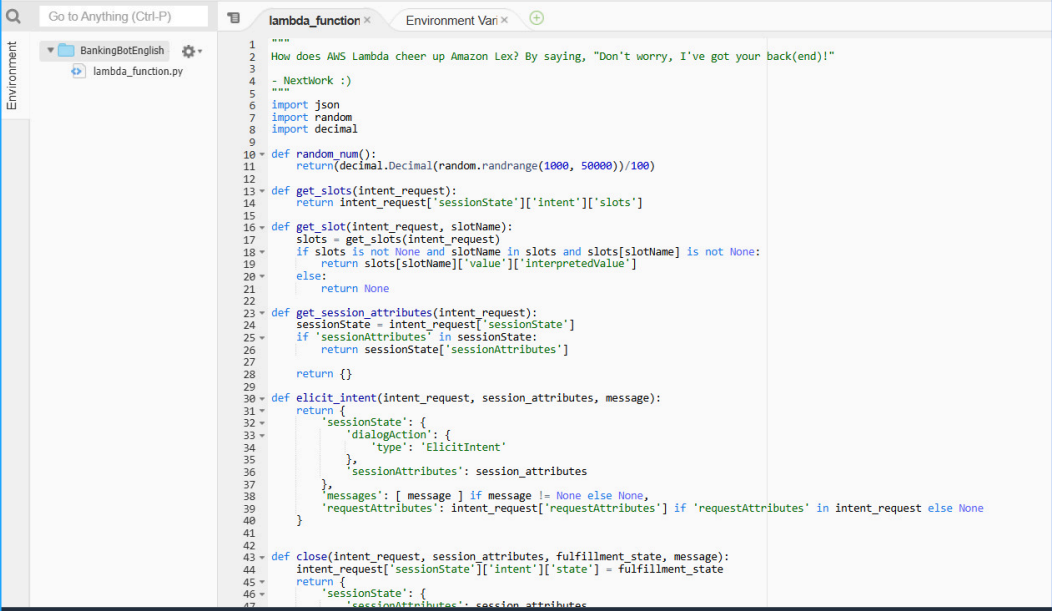
This project took me...

This project took me 2.5 hours to complete, including all the documentation.

AWS Lambda Functions

AWS Lambda is a serverless compute service that runs your code in the cloud without server management. It executes code only when needed and scales automatically. Just provide your code in any supported language.

In this project, I created a Lambda function to generate a user's bank balance. Here, it's a random number, but in real use, Lambda could retrieve the balance from a database. Amazon Lex alone can't access this data, so this connection is essential.



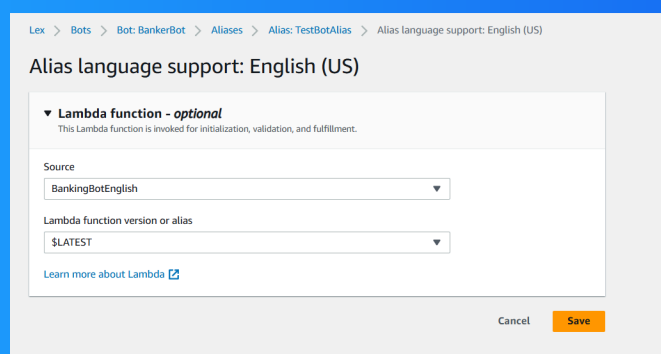
```
1 """
2 How does AWS Lambda cheer up Amazon Lex? By saying, "Don't worry, I've got your back(end)!"
3
4 - NextWork :)
5 """
6 import json
7 import random
8 import decimal
9
10 def random_num():
11     return(decimal.Decimal(random.randrange(1000, 50000))/100)
12
13 def get_slots(intent_request):
14     return intent_request['sessionState']['intent']['slots']
15
16 def get_slot(intent_request, slotName):
17     slots = get_slots(intent_request)
18     if slots is not None and slotName in slots and slots[slotName] is not None:
19         return slots[slotName]['value']['interpretedValue']
20     else:
21         return None
22
23 def get_session_attributes(intent_request):
24     sessionState = intent_request['sessionState']
25     if 'sessionAttributes' in sessionState:
26         return sessionState['sessionAttributes']
27     return {}
28
29 def elicit_intent(intent_request, session_attributes, message):
30     return {
31         'sessionState': {
32             'dialogAction': {
33                 'type': 'ElicitIntent'
34             },
35             'sessionAttributes': session_attributes
36         },
37         'messages': [ message ] if message != None else None,
38         'requestAttributes': intent_request['requestAttributes'] if 'requestAttributes' in intent_request else None
39     }
40
41
42
43 def close(intent_request, session_attributes, fulfillment_state, message):
44     intent_request['sessionState']['intent']['state'] = fulfillment_state
45     return {
46         'sessionState': {
47             'sessionAttributes': session_attributes
```

Chatbot Alias

An alias in Amazon Lex acts as a pointer to a specific bot version, so when connecting to other AWS services or apps, they link to the alias, which directs to the desired bot version.

TestBotAlias is the default version of your bot for testing and development. It is a playground to ensure everything works smoothly before launching updates.

To connect Lambda with my BankerBot, I visited my bot's TestBotAlias and linked it to the latest version of my AWS Lambda function. This setup allows my bot to access Lambda during testing and development.



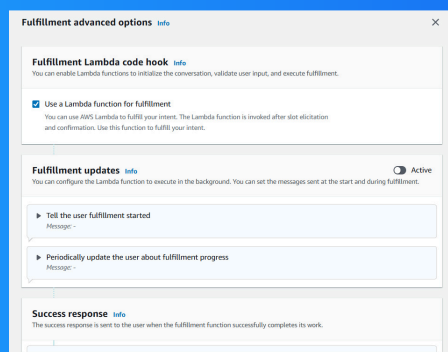
The screenshot shows the Amazon Lex console interface for configuring a bot alias. The breadcrumb navigation at the top reads: Lex > Bots > Bot: BankerBot > Aliases > Alias: TestBotAlias > Alias language support: English (US). The main heading is "Alias language support: English (US)". Below this, there is a section titled "▼ Lambda function - optional" with a sub-note: "This Lambda function is invoked for initialization, validation, and fulfillment." Underneath, there are two dropdown menus: "Source" with the value "BankingBotEnglish" and "Lambda function version or alias" with the value "\$LATEST". A link "Learn more about Lambda" is present below the second dropdown. At the bottom right of the configuration box are "Cancel" and "Save" buttons.

Code Hooks

A code hook connects my chatbot to custom Lambda functions to perform specific tasks. They handle complex actions, such as checking database data or making decisions based on previous interactions, that a basic chatbot setup can't manage alone.

Even though I connected my Lambda function with my chatbot's alias, I needed to use code hooks because the chatbot can't calculate or return a bank balance by itself. Code hooks are essential for managing complex tasks like this effectively.

I could find code hooks in the Fulfillment panel by expanding the "On successful fulfillment" bubble and accessing the advanced option. I then configured the code hook by checking the box next to 'Use a Lambda function for fulfillment.'



Fulfillment advanced options info ✕

Fulfillment Lambda code hook info
You can enable Lambda functions to initialize the conversation, validate user input, and execute fulfillment.

☒ **Use a Lambda function for fulfillment**
You can use AWS Lambda to fulfill your intent. The Lambda function is invoked after slot elicitation and confirmation. Use this function to fulfill your intent.

Fulfillment updates info 🔴 Active
You can configure the Lambda function to execute in the background. You can set the messages sent at the start and during fulfillment.

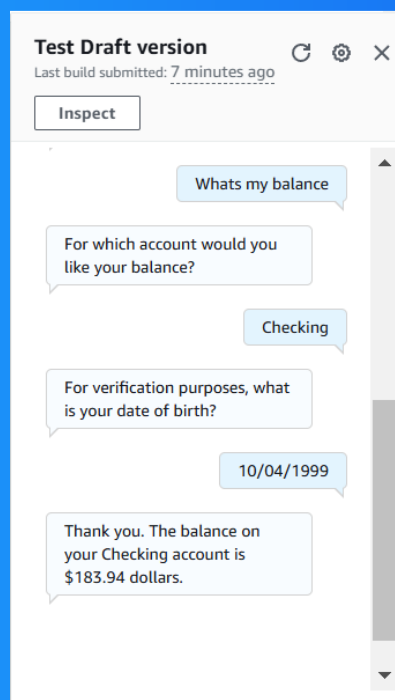
▶ **Tell the user fulfillment started**
Message: -

▶ **Periodically update the user about fulfillment progress**
Message: -

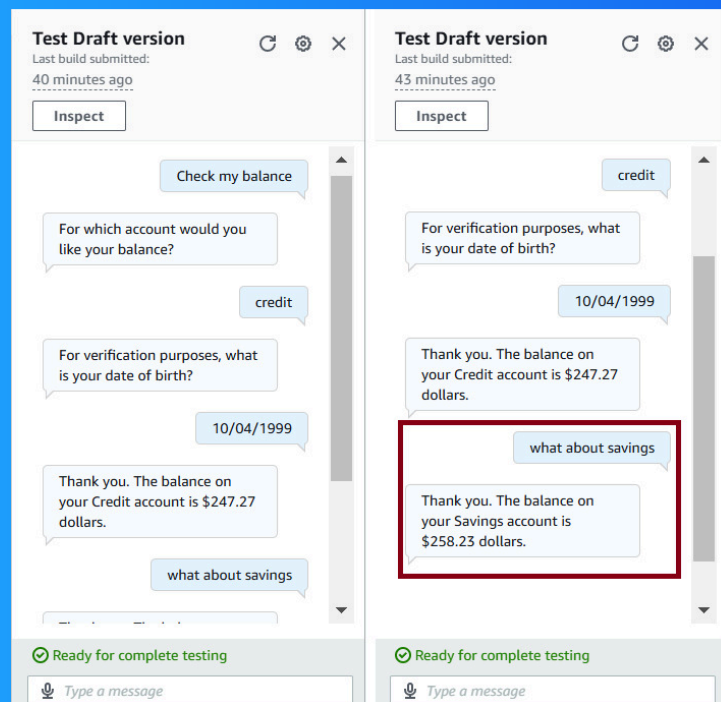
Success response info
The success response is sent to the user when the fulfillment function successfully completes its work.

The final result!

I've set up my chatbot to trigger Lambda and return a random dollar figure when a user requests their bank balance by entering phrases like 'Check my balance' or any other sample utterances from the CheckBalance intent and verifying the request.



Save User Info with your Chatbot



Introducing Today's Project!

What is Amazon Lex?

Amazon Lex is a tool from AWS for building chatbots that understand and respond to user messages. It's useful since it enables businesses to easily create chatbots for customer support. It also uses AI/ML capabilities in classifying user's intent.

How I used Amazon Lex in this project

In today's project, I used Amazon Lex to enable context carryover, allowing my chatbot to remember user info, (birthdate), from the CheckBalance intent. This lets FollowupCheckBalance access that info for follow-up balance checks w/o re-verification.

One thing I didn't expect in this project was...

I didn't expect in this project is how smoothly context carryover worked to retain user info, making follow-up interactions simpler and more seamless.

This project took me...

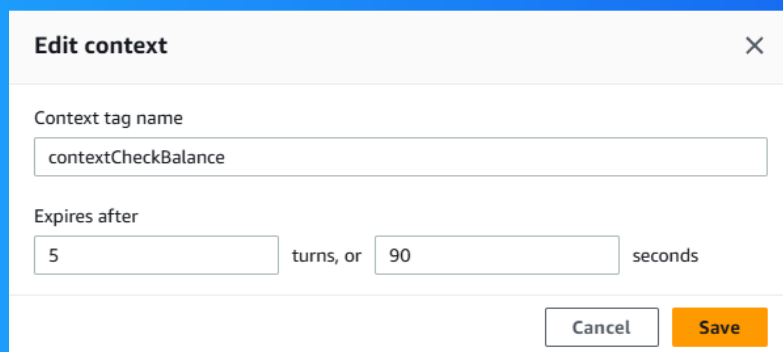
I completed this project in about 3 hours, including building the chatbot from scratch (starting from project one in this series) and the project documentation.

Context Tags

Context tags are used to store and check for specific information across different parts of a conversation. They help save the user from having to repeat certain information.

There are two types of context tags: Output context tags, which store details after an intent finishes to share with other parts of the conversation, & Input context tags, which check if specific details are already present before intent activation.

I created a context tag called 'contextCheckBalance' in the CheckBalance intent. It stores information about the user's account type, so the bot can access this detail later in the conversation, & for smoother responses without needing to re-ask.



The screenshot shows a dialog box titled "Edit context" with a close button (X) in the top right corner. Inside the dialog, there is a section labeled "Context tag name" with a text input field containing the value "contextCheckBalance". Below this, there is a section labeled "Expires after" with two input fields: the first contains the number "5" and the second contains the number "90". Between these two input fields is the text "turns, or", and to the right of the second input field is the text "seconds". At the bottom right of the dialog, there are two buttons: "Cancel" and "Save".

FollowUpCheckBalance

I created a new intent called FollowupCheckBalance. The purpose of this intent is to enable a quick follow-up balance check without requiring re-authentication, so users don't need to re-enter their date of birth for verification.

This intent is connected to the previous intent, CheckBalance, because FollowupCheckBalance is triggered only when a user makes a second balance check request, allowing the conversation to flow smoothly without re-verification.

How about my {accountType} account?

What about {accountType}?

And in {accountType}?

Input Context Tag

I created an input context, contextCheckBalance, that matches the output context tag from the CheckBalance intent. This setup allows FollowupCheckBalance to access data provided during CheckBalance, enabling smooth follow-up without re-entering info.

▼ Default values - *optional*

#contextCheckBalance.dateOfBirth

×

Provide a default value, #value for a context value, or [variable] for session variable.

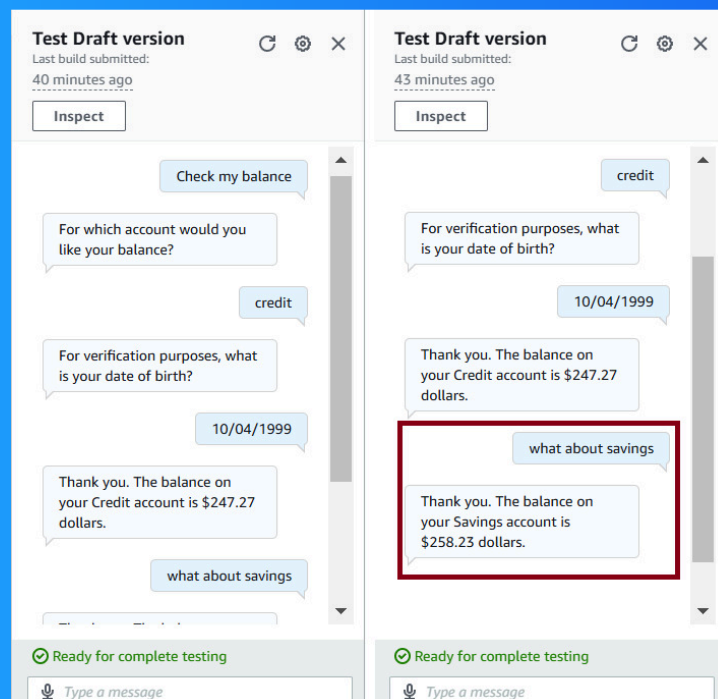
San Diego, #ContextTag.SlotName, [SessionAttributeName]

Add default value

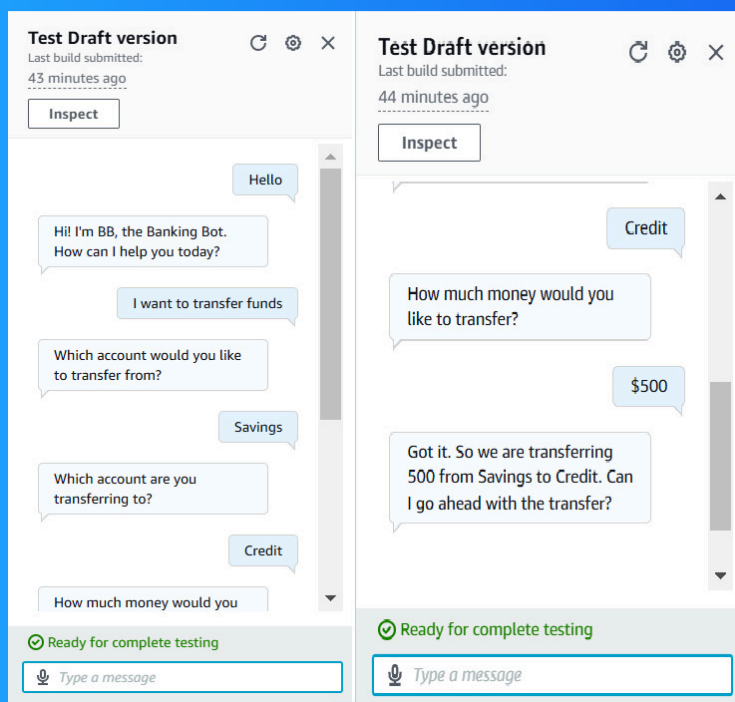
The final result!

To see the context tags and followup intent in action, I triggered CheckBalance by saying "check my balance" and entered my birthdate for verification, then followed up with "what about savings?" to activate FollowupCheckBalance.

If I had gone straight to trying to trigger FollowupCheckBalance without setting up any context, my chabot wouldn't have the necessary information from the initial CheckBalance intent and will prompt for the 'FallbackIntent.'



Build a Chatbot with Multiple Slots



Introducing Today's Project!

What is Amazon Lex?

Amazon Lex is a tool from AWS for building chatbots that understand and respond to user messages. It's useful since it enables businesses to easily create chatbots for customer support. It also uses AI/ML capabilities in classifying user's intent.

How I used Amazon Lex in this project

In today's project, I used Amazon Lex to complete BankerBot by creating the "TransferFunds" intent, enabling users to transfer funds between accs with shared slot types for source & target accounts, plus a confirmation prompt for added verification.

One thing I didn't expect in this project was...

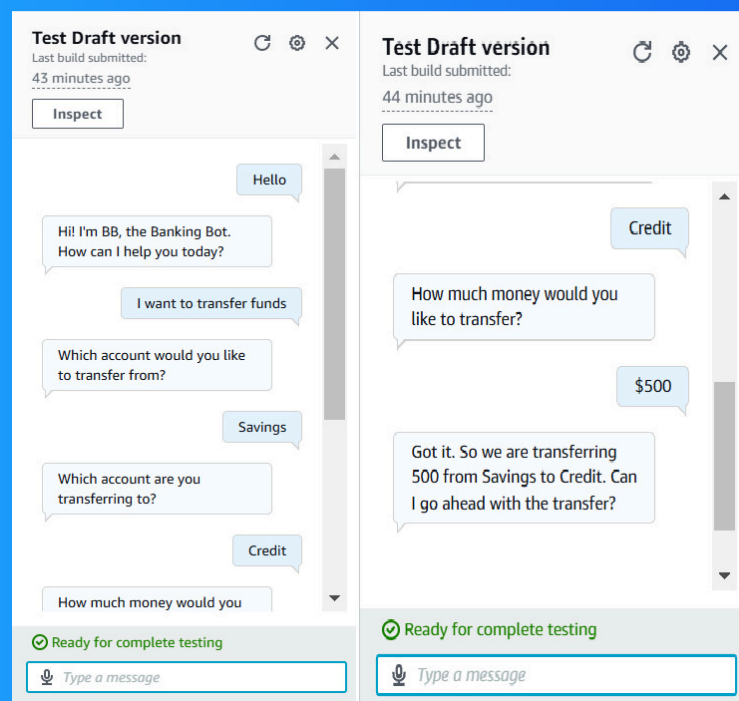
One thing I didn't expect in this project was how seamlessly I completed BankerBot with Amazon Lex and utilized Amazon CloudFormation to swiftly deploy resources from various AWS services, including the entire BankerBot defined in the YAML file.

This project took me...

I spent about 1 hour completing my BankerBot and an additional 1.5 hours using Amazon CloudFormation for the first time, troubleshooting errors during the bot's deployment and working on the documentation process.

TransferFunds

An intent I created for my chatbot was TransferFunds, which help user transfer funds between different bank accounts.



Using multiple slots

For this intent, I used the same slot type twice because the TransferFunds intent needs both a source account, where funds are transferred from, and a target account, where funds are transferred to.

I also learned to create confirmation prompts, which let the chatbot confirm a user's intent. In this project, a prompt was set to confirm transfers between accounts, and I added a decline response in case the user decides not to proceed.

Confirmation [Info](#) Active

Prompts help to clarify whether the user wants to fulfill the intent or cancel it.

▼ Prompts to confirm the intent	Responses sent when the user declines the intent
<i>Message: Got it. So we are transferring {transferAmount}...</i>	<i>Message: The transfer has been cancelled.</i>

Confirmation prompt
What will the bot say to prompt the user to confirm this intent.

Got it. So we are transferring {transferAmount} from {sourceAccountType} to {targetAccountType}. Can I go ahead

Decline response
What will the bot say if the user says NO to the confirmation prompt.

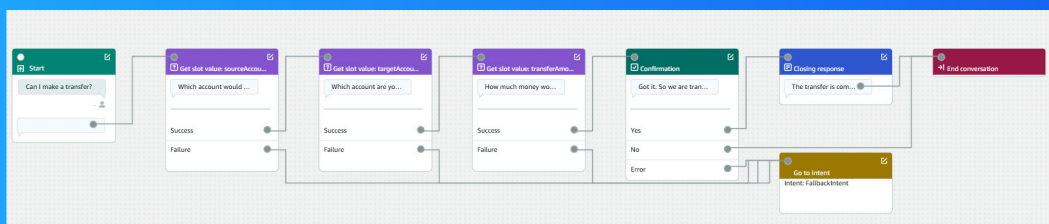
The transfer has been cancelled.

Advanced options
Configure confirmation prompts and decline responses.

Exploring Lex features

Lex also has a special conversation flow feature that guides the interaction between the user and the chatbot. This editable structure helps the chatbot follow a clear path, making conversations smoother and more intuitive.

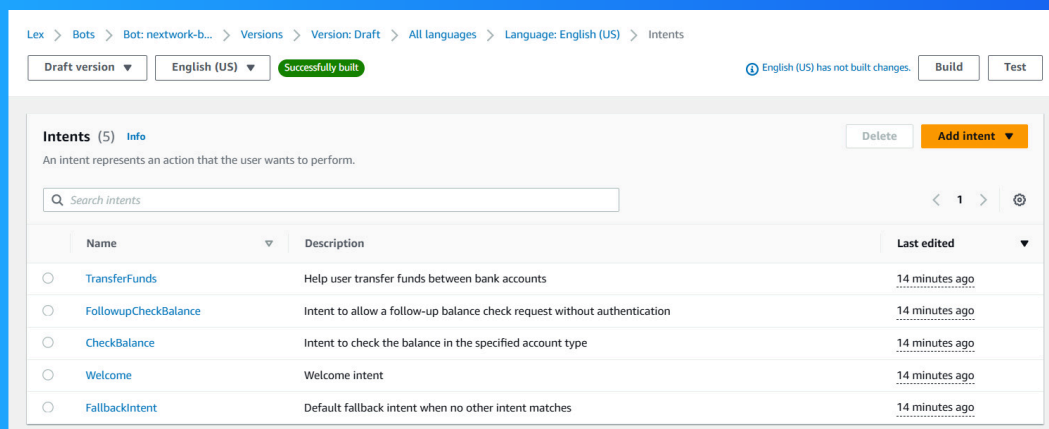
You could also set up your intent using a visual builder! A visual builder transforms the setup page into a flowchart, making it easy to visualize and organize the conversation flow. You can also adjust intents as needed.



AWS CloudFormation

AWS CloudFormation is service that gives you an easy way to create and set up AWS resources. It's an infrastructure as code service - meaning you will use a file that describes all the resources you want to create and their dependencies as code.

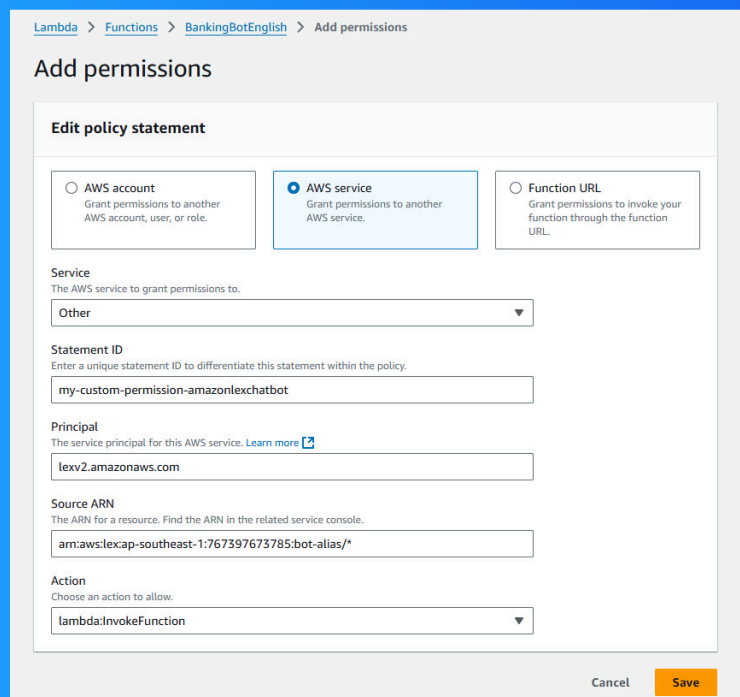
I used CloudFormation to quickly deploy BankerBot with a YAML file, setting it up in one go or in a single stack file.



The final result!

Re-building my bot with CloudFormation was quick. It only took me a minute to set up CloudFormation stack and 5-6 minutes for deployment to finish.

There was an error after deploying my bot! It showed Access Denied when invoking the Lambda function. I resolved it by updating the function's policy to grant the necessary permissions, allowing the bot to access the Lambda as needed.



The screenshot shows the 'Add permissions' page in the AWS Lambda console. The breadcrumb trail at the top reads: [Lambda](#) > [Functions](#) > [BankingBotEnglish](#) > [Add permissions](#). The main heading is 'Add permissions'. Below this is a section titled 'Edit policy statement' which contains three radio button options: 'AWS account' (unselected), 'AWS service' (selected), and 'Function URL' (unselected). Each option has a brief description of what it grants permissions to. Below these options, there are several input fields: 'Service' (a dropdown menu currently showing 'Other'), 'Statement ID' (a text box containing 'my-custom-permission-amazonlexchatbot'), 'Principal' (a text box containing 'lexv2.amazonaws.com'), 'Source ARN' (a text box containing 'arn:aws:lex:ap-southeast-1:767397673785:bot-alias/*'), and 'Action' (a dropdown menu showing 'lambda:InvokeFunction'). At the bottom right of the form are 'Cancel' and 'Save' buttons.