

Kimberly- A Banking Bot (Using Amazon Lex)

▼ Amazon Lex

- It is a AWS service for building conversational interfaces using voice and text basically it makes it easy to build chatbots
- The “FallbackIntent” is triggered when the chatbot’s confidence score is below the set threshold for all defined intents. It is triggered when the chatbot doesn’t recognize user’s input.
- “intents” in Amazon Lex represents what the user is trying to achieve in their conversation with the chatbot
- ‘intent classification confidence score threshold’ represents the minimum score for the chatbot to confidently understand the user. for this project i went with 0.40 (default value). This means that my chatbot should be 40% confident to respond to a user’s input
- The purpose of defining ‘variations’ in a chatbot response is to provide a dynamic range of responses, making the chatbot sound more conversational.
- Amazon Lex provides many ready-to-use slot types for common information, like dates and times, but you can also create your own custom slot types to fit your specific needs.
- ‘utterances’ are phrases that users might say to trigger a specific intent

Creating a bot

- Log in to the **AWS console**.
- Navigate to Amazon Lex (type Lex into the search bar of your Console).
- Check your URL in your web browser - does it say ...console.aws.amazon.com/lexv2/...?

- If you're not seeing "lexv2" in your URL, click on **Switch to the new Lex V2 console** link in your left-hand menu.
- Select **Create bot**.
- Select **Create a blank bot**.

Configure bot settings [Info](#)

Creation method

- ☒ **Create a blank bot**
Create a basic bot with no preconfigured languages, intents, and slot types.
- ☐ **Start with an example**
An example bot has preconfigured languages, intents, and slot types. You can change these settings.
- ☐ **Start with transcripts**
Automatically generate intents from conversation transcripts that you upload. Only English (US) language is available when starting with a transcript.

- For Bot name, enter BankerBot
- For Description, enter:

Banker Bot to help customer check their balance and make transfers.

Bot configuration

Bot name

BankerBot

Maximum 100 characters. Valid characters: A-Z, a-z, 0-9, -, _

Description - *optional*

This description appears on bot list page. It can help you identify the purpose of your bot.

Banker Bot to help customer check their balance and make transfers.

Maximum 200 characters.

- Under IAM permissions, select **Create a role with basic Amazon Lex permissions.**

💡 Why do we need Amazon Lex permissions?

Amazon Lex needs the permission to call other AWS services on your behalf, later in this project series you'll be integrating Lex with another service called Lambda!

- I used it to call another service called Lambda (shown later in the doc)
- Under **Bot error logging**, select **Disabled**.

💡 What is bot error logging?

Bot error logging is a way for your chatbot to keep track of problems that happen during conversations. When something goes wrong, like the bot doesn't understand a user or fails to respond, these issues are recorded in a log file. These logs are stored in services like AWS CloudWatch or S3, and can be encrypted for security. Reviewing error logs helps you find and fix problems, making your bot smarter and more reliable.

IAM permissions [Info](#)

IAM roles are used to access other services on your behalf.

Runtime role

Choose a role that defines permissions for your bot. To create a customised role, use the IAM console.

☒ Create a role with basic Amazon Lex permissions.

☐ Use an existing role.

i Creating a role takes a few minutes. Don't delete the role or edit the trust or permissions policies in this role until we've finished creating it.

New role

Amazon Lex creates a runtime role with permission to upload to Amazon CloudWatch Logs.

`AWSServiceRoleForLexV2Bots_7XESZFVVCH8`

Bot error logging [Info](#)

Debug unexpected issues on Lex bots.

Error logs

☐ Enabled

☒ Disabled

[Learn more about error logs](#)

[Go to error logs](#)

- Under **Children's Online Privacy Protection Act (COPPA)**, select **No**.
- Under **Idle session timeout**, keep the default of **5 minutes**.

The screenshot shows two configuration panels in the Amazon Lex console. The first panel is titled 'Children's Online Privacy Protection Act (COPPA)' with an 'Info' link. It contains a question: 'Is use of your bot subject to the Children's Online Privacy Protection Act (COPPA)?' with two radio button options: 'Yes' and 'No'. The 'No' option is selected. The second panel is titled 'Idle session timeout'. It includes a sub-header 'Idle session timeout' and a descriptive paragraph: 'You can configure how long a session is maintained when the user does not provide any input and the session is idle. Amazon Lex retains context information until a session ends.' Below this, there is a 'Session timeout' section with a text input field containing the number '5' and a dropdown menu set to 'minute(s)'. A note at the bottom of this section states: 'By default, session duration is 5 minutes, but you can specify any duration between 1 and 1,440 minutes (24 hours).'

💡 What does Idle session timeout mean?

Amazon Lex will only maintain a session for a set length of time. If the user is idle and doesn't add any input for 5 minutes, their session will end.

- Select **Next**.
- Keep the language as **English** so you can explore Lex's full set of features in this project.
- Under **Voice interaction**, click on the dropdown that says **Kimberly**.

💡 These voices are borrowed from Amazon Polly, which is another AWS service all about turning your text into speech!

- For **Intent classification confidence score threshold**, keep the default value of **0.40**.

💡 What is intent classification confidence score threshold?

When you're using Amazon Lex to build a chatbot, this threshold is like a minimum score for your chatbot to confidently understand what the user is trying to say.

Setting this to 0.4 means that your chatbot needs to be at least 40% confident that it understands what the user is asking to be able to give a response.

So if a user's input is ambiguous and your chatbot's confidence score is below 0.4, it'll throw an error message.

- Select **Done**.

Creating my first Intent WelcomeIntent

- Once I created my bot, I trained it so that it can give a friendly user experience. I trained it for welcome intent
- When the bot is created, one will automatically see a page called **Intent: NewIntent**.

💡 What are intents?

An intent is what the user is trying to achieve in their conversation with the chatbot. For example, checking a bank account balance; booking a flight; ordering food.

In Amazon Lex, you build your chatbot by defining and categorising different intents. If you set up different intents, one single chatbot can manage a bunch of requests that are usually related to each other.

- I created my first intent known as WelcomeIntent to greet users when they say hello or ask for help. I also launched and tested my chatbot and it responded successfully
- Under **Intent details**, enter WelcomeIntent for the **Intent name**.
- Add the description Welcoming a user when they say hello.

▼ **Intent details** [Info](#)

Intent name

WelcomeIntent

Maximum 100 characters. Valid characters: A-Z, a-z, 0-9, -, _

Description - optional

Welcome a user when they say hello

Maximum 2000 characters.

- Scroll down to the **Sample utterances** panel.
- Click the **Plain Text** button.
- Copy the text below, which represent the user inputs (called utterances) that will trigger this intent, and paste it into the text window:

Hi
Hello
I need help
Can you help me?

- Click back to the **Preview** button to see these utterances in chat form.

Sample utterances (4) [Info](#)

Representative phrases that you expect a user to speak or type to invoke this intent. Amazon Lex extrapolates based on the sample utterances to interpret any user input that may vary from the samples. The priority order of the sample utterances is not used to determine intent classification output.

Sort by added (ascending) ▼

Preview ▼

Hi

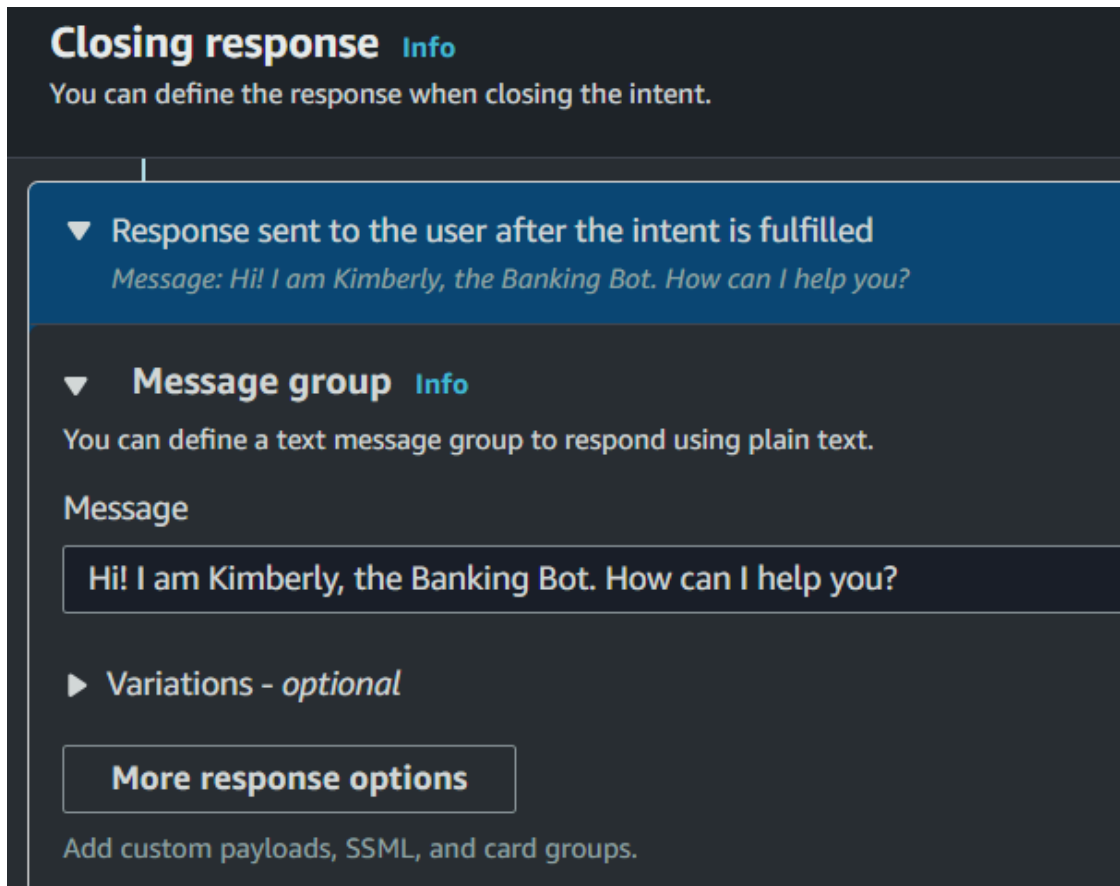
Hello

I need help

Can you help me?

- Scroll down to **Closing response**, and expand the arrow for **Response sent to the user after the intent is fulfilled**.
- In the **Message** field, enter the following message:

Hi! I'm Kimberly, the Banking Bot. How can I help you today?



- Choose **Save intent**.
- Choose **Build**, which is close to the top of the screen.
- This can take 30 seconds - while we wait...
- Choose **Test**.

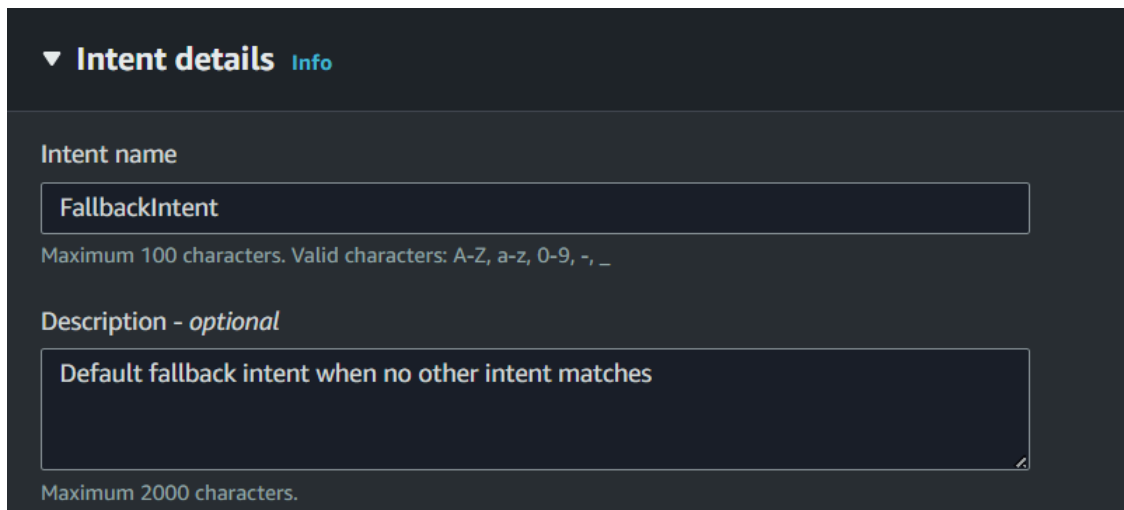
▼ How does my chatbot respond to these user inputs?

The first three are successfully recognized - Amazon Lex is able to use its ML techniques to match what you have said against your utterances.

But the last two fail, resulting in an **Intent FallbackIntent is fulfilled** response - meaning Amazon Lex doesn't quite recognize your utterance. We'll learn what FallbackIntent means in the next step.

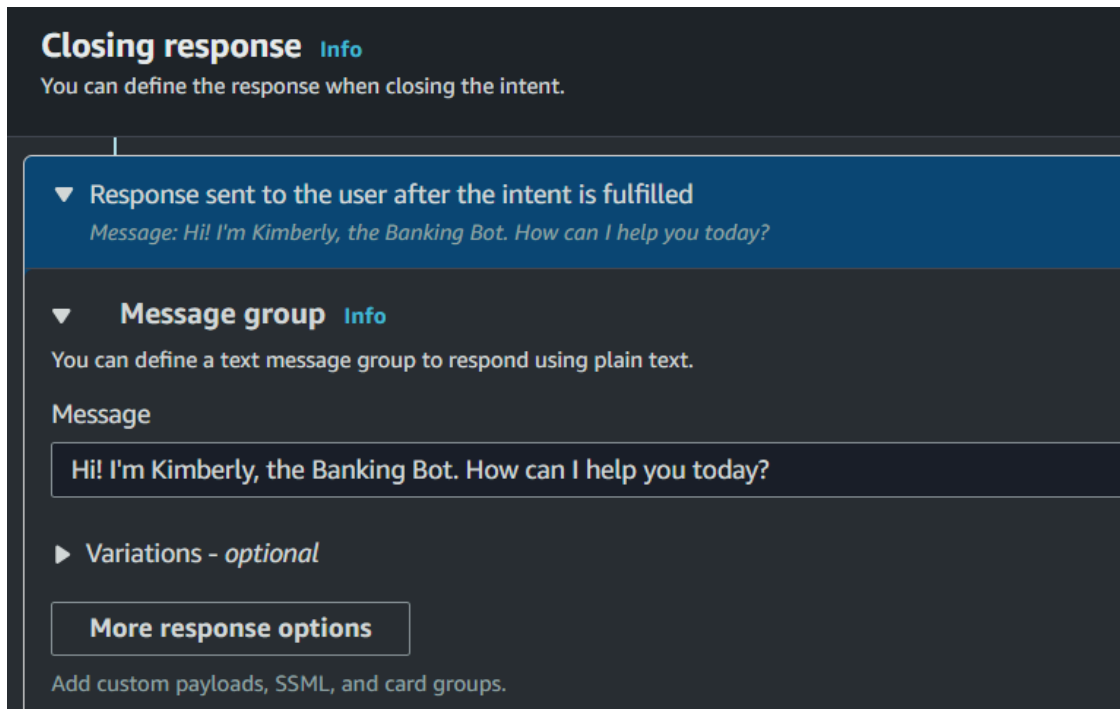
Creating FallbackIntent

- My chatbot returned the error message 'intent FallbackIntent is fulfilled' when I entered 'who are you' and 'I am Aprajita'. This error message occurred because my bot couldn't match it to the initially defined intents.
- My chatbot has been built using the **English (US)** dialect, so some English accents may not be perfectly transcribed - this is why Amazon Lex actually supports multiple English dialects.
- I wanted to configure FallbackIntent because I want my BankerBot to respond to the users in a friendly manner even when there is an error



The screenshot shows the 'Intent details' section in the Amazon Lex console. It features a dark-themed interface with a header 'Intent details' and a sub-header 'Info'. Below this, there are two main input fields. The first is labeled 'Intent name' and contains the text 'FallbackIntent'. Below this field is a note: 'Maximum 100 characters. Valid characters: A-Z, a-z, 0-9, -, _'. The second field is labeled 'Description - optional' and contains the text 'Default fallback intent when no other intent matches'. Below this field is a note: 'Maximum 2000 characters.'.

- To configure FallbackIntent, I customized the closing response messages and added variations so that it sounds more natural



- I added variations so that the end users can see a slightly different response each time the bot doesn't understand. I have also set up some initial responses (eg: hmm this is interesting, one moment etc) for a little humanized experience when the chatbot recognizes the initial intent
- the key concepts that i learnt from this project is intents, utterances, FallbackIntent and confidence score threshold
- In your left hand navigation panel, choose **FallbackIntent**.
- The default FallbackIntent message you saw just now ("Intent FallbackIntent is fulfilled") can be a little confusing.
- Scroll down to **Closing responses**.
- Expand the arrow for **Response sent to the user after the intent is fulfilled**.
- In the **Message** field, add the following text:

Sorry I am having trouble understanding. Can you describe what you'd like to do in a few words? I can help you find your account balance, transfer funds and make a payment.

- You'll notice another arrow next to the label **Variations - optional**.
- Expand the arrow.
- Enter the following text:

Hmm could you try rephrasing that? I can help you find your account balance, transfer funds and make a payment.

💡 What are variations?

Variations are literally variations of the same Message in the main Message box. When Amazon Lex needs to return a Fallback response, it will randomly choose a message from the group and return that.

Variations will give your users a dynamic range of responses, making them sound more conversational!

- Choose **Save intent**.
- Choose **Build**
- What's the most surprising part of this project is that no coding is required to create this project
- It was most rewarding to see my bot in action and respond the way i wanted
- I chose to do this project because i wanted to learn Amazon Lex and build a chatbot

Creating customized Slots

- I created a custom slot type for account types because we need to collect the information from user to respond to their banking request
- Slots are pieces of information that a chatbot needs, in order to complete a user's request. Think of them as blanks that needs to be filled in a form
- Amazon Lex provides many ready-to-use slot types for common information, like dates and times, but you can also create your own custom slot types to fit your specific needs. That's what we're about to do.

- While creating my own custom slot for account Type, I restricted the slot values because I wanted my chatbot to only consider 4 account types, savings, credit, Fixed deposit and recurring deposit. I don't want the user to put in more than what my bank can offer. Otherwise, Amazon Lex might use machine learning to accept other values that it frequently encounters from users. I **didn't** want Lex to use machine learning to accept other values.

Steps

- In the **Amazon Lex** console, choose **Slot types** in your left hand navigation panel.
- Choose **Add slot type**.
- From the dropdown, choose **Add blank slot type**.
- Enter accountType for the **Slot type name**.
- Choose **Add**.
- This will bring up a large **Slot types** editor panel
- In the **Slot value resolution** panel, choose **Restrict to slot values**.

What does this selection mean?

Selecting **Restrict to slot values** makes sure that only the values that you specify will count as a valid accountType!

Otherwise, Amazon Lex might use machine learning to accept other values that it frequently encounters from users.

In our case, we **don't** want Lex to use machine learning to accept other values. We have a set list of bank account types that we offer - Checking, Savings, and Credit accounts - and we don't offer any other bank account types.

If Lex starts accepting other values outside of these three, users could end up having conversations about bank account types we don't actually offer! To prevent that from happening, we'll restrict Lex to only acknowledge the bank account types we set up here.

- Now let's add the three account types!
- In the **Values** field, enter Checking

- Select **Add value**, or just press Enter on your keyboard.
- Do the same for Savings
- Enter Credit, and add a few synonyms in the second field. Press ; on your keyboard after every time you add in a new one:
 - credit card
 - visa
 - mastercard
 - amex
 - american express
- Choose **Add value** to finish up your work for Credit.

Why are we creating slot values?

In this project, we're setting up a new intent that lets users check their bank account balance. Our BankerBot will need to know what **bank account type** the user is wanting to check, so BankerBot will have to ask for the user's bank account type in the conversation.

By entering specific slot type values, we make sure that our BankerBot only recognizes and accepts the bank account types we offer - **Checking, Savings, and Credit**. This prevents any confusion or misunderstandings with the user, as the chatbot won't suggest or accept account types that aren't available.

Creating the CheckBalance Intent

- I set up a new intent that lets users check their bank account balance. Our BankerBot will need to know what bank account type the user is wanting to check, so BankerBot will have to ask for the user's bank account type in the conversation. By entering specific slot type values, we make sure that our BankerBot only recognizes and accepts the bank account types we offer - **Checking, Savings, and Credit**. This prevents any confusion or misunderstandings with the user, as the chatbot won't suggest or accept account types that aren't available.
- In your left hand navigation panel, head back to **Intents**.

- Choose **Add intent**, then **Add empty intent**.
- Enter CheckBalance as your intent name.
- Choose **Add**.

Enter the following description in the Intent details panel: Intent to check the balance in the specified account type.

- Scroll down to **Sample utterances**.
- Switch to **Plain Text** and paste in the following utterances:

What's the balance in my account?
 Check my account balance
 What's the balance in my {accountType} account?
 How much do I have in {accountType} ?
 I want to check the balance
 Can you help me with account balance?
 Balance in {accountType}

▼ Why do some of the utterances have the text {accountType}?

- This means that Amazon Lex is now prepared to look for slot values from the user's input.
- If a word fits what's expected for the **accountType** slot, Lex will automatically fill in that information and won't need to prompt the user for their accountType anymore (saving time for the user)!
- Scroll down until you can see the **Slots** pane.
- Choose **Add slot** button.
- For slot's **Name**, enter accountType.
- For the **Slot type**, choose your custom slot value accountType
- Enter the following for **Prompts**: For which account would you like your balance?
- Choose **Add**.

- You might remember that we mentioned the CheckBalance intent should also check for the user's birthday... let's create a Slot for the birth date!
- Choose **Add slot**.
- Use these values for your next slot:
 - Name: dateOfBirth
 - Slot type: AMAZON.Date
 - Prompts: For verification purposes, what is your date of birth?
- Always save your intent and select 'build'
- I associated my custom slot with CheckBalance, which collects user data and helps them check their bank balance

▼ Why does my bot say 'Intent CheckBalance fulfilled'?

To successfully check a user's bank balance, BankerBot needs to:

1. Get the **user's details** i.e. birthday and account type.
2. Find the user's account **balance** figure using those details.
3. **Return** the account balance figure to the user.

So far, your BankerBot can do the first item in the list - we created slots for accountType and dateOfBirth to get the user's details!

The second item? Not yet, BankerBot will actually find the user's bank balance in the next project of this series.

If your bot returns Intent CheckBalance fulfilled, that means your bot has successfully taken in the user's details, but it doesn't actually know how to calculate the bank balance yet.

Because of this, it just returns the generic closing response to tell you it's finished running the intent.

- By adding custom slots in utterances my chatbot's users can provide inputs like savings, recurring, fixed or credit responses which were customized by me

Why does my bot say 'Intent CheckBalance fulfilled'?

To successfully check a user's bank balance, BankerBot needs to:

1. Get the **user's details** i.e. birthday and account type.
2. Find the user's account **balance** figure using those details.
3. **Return** the account balance figure to the user.

So far, your BankerBot can do the first item in the list - we created slots for accountType and dateOfBirth to get the user's details!

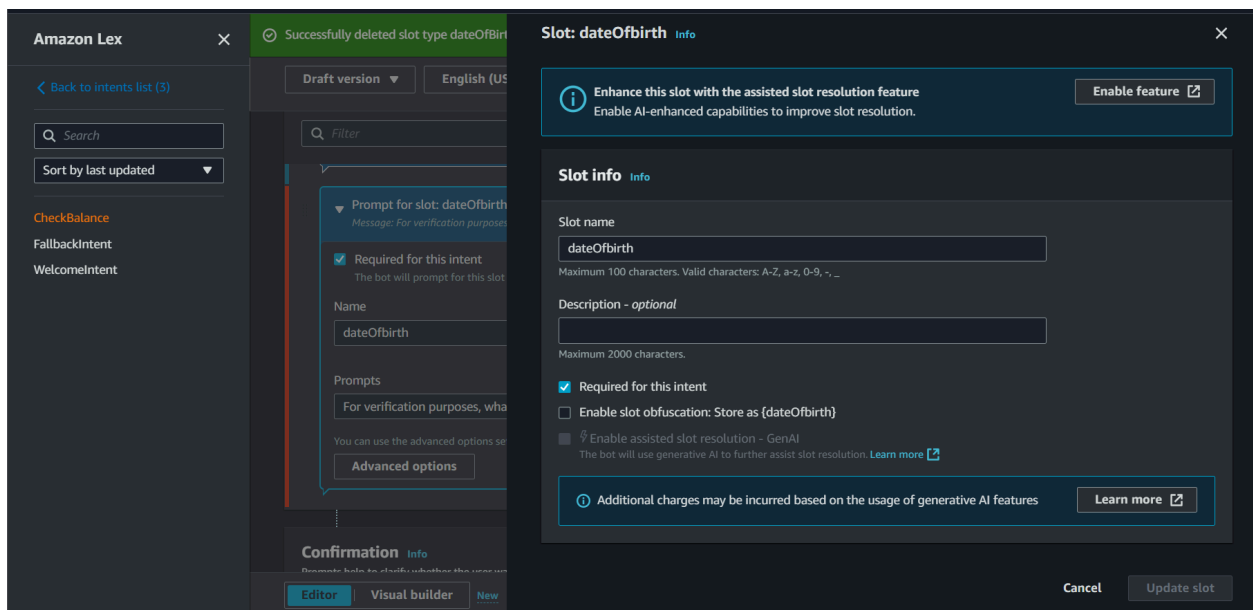
The second item? Not yet, BankerBot will actually find the user's bank balance in the next project of this series.

If your bot returns Intent CheckBalance fulfilled, that means your bot has successfully taken in the user's details, but it doesn't actually know how to calculate the bank balance yet.

Because of this, it just returns the generic closing response to tell you it's finished running the intent.

Handling Errors

- To handle error handling, i will add failure responses.
- intent → CheckBalance → scroll down and inside dateOfBirth → Advanced options



- Scroll down to Slot prompts. This is where one can ask the user their DOB in different ways

Slot prompts [Info](#)
 Prompts to elicit the slot.

▼ Bot elicits information
Message: For verification purposes, what is your date of birth?

☒ Play the messages in order [Info](#)
 Messages will be used in the predefined order as slot prompts by your bot.

▼ Message group [Info](#)
 You can define a text message group to respond using plain text.

Message 1

▼ Variations - optional

Message 2

Message 3

More prompt options

 Add custom payloads, SSML, and card groups.

Cancel Update slot

▼ What does that mean for the users?

Playing the messages in order means that my slots prompts won't be selected randomly. My chatbot would only send message 2 if the user's response to the first prompt is invalid

- Scroll down to slot capture: failure response
- Slot capture: failure response deals with scenarios where the user is not giving a valid slot value. In this case, it's when the user isn't providing a date where the chatbot asks for one. The chatbot will send a response to clarify the user that they are not providing the right information and may even clarify in which order the chatbot wants the response

- After adding, update slot → save intent → build → test

Slot: dateOfBirth [Info](#) ✕

Slot capture: failure response [Info](#)

You can provide responses, set values, and next steps. You can also branch based on conditions.

▼ **Response when slot value isn't understood** [Info](#)

Message: *It looks like the date provided isn't right. Re-enter your birthdate in the following format MM/DD/YYYY or 5th June, 1995*

▼ **Message group** [Info](#)

You can define a text message group to respond using plain text.

Message

It looks like the date provided isn't right. Re-enter your birthdate in the following format MM/DD/YYYY or 5th June

▼ **Variations - optional**

I don't think you have entered a date. I need your birthdate for verification purposes

Hmm...let's try again. Please enter your date of birth

More response options

Add custom payloads, SSML, and card groups.

▶ **Set values**

-

Next step in conversation

Switch to intent: *FallbackIntent*

Cancel

Update slot

- I tried it out

Kimberly- A Banking Bot (Using Amazon Lex)

19

Test Draft version

Last build submitted: 1 minute ago



Inspect

Sorry that wasn't clear to me,
what's your birthdate?

egfdgrg

Hmm...let's try again. Please
enter your date of birth

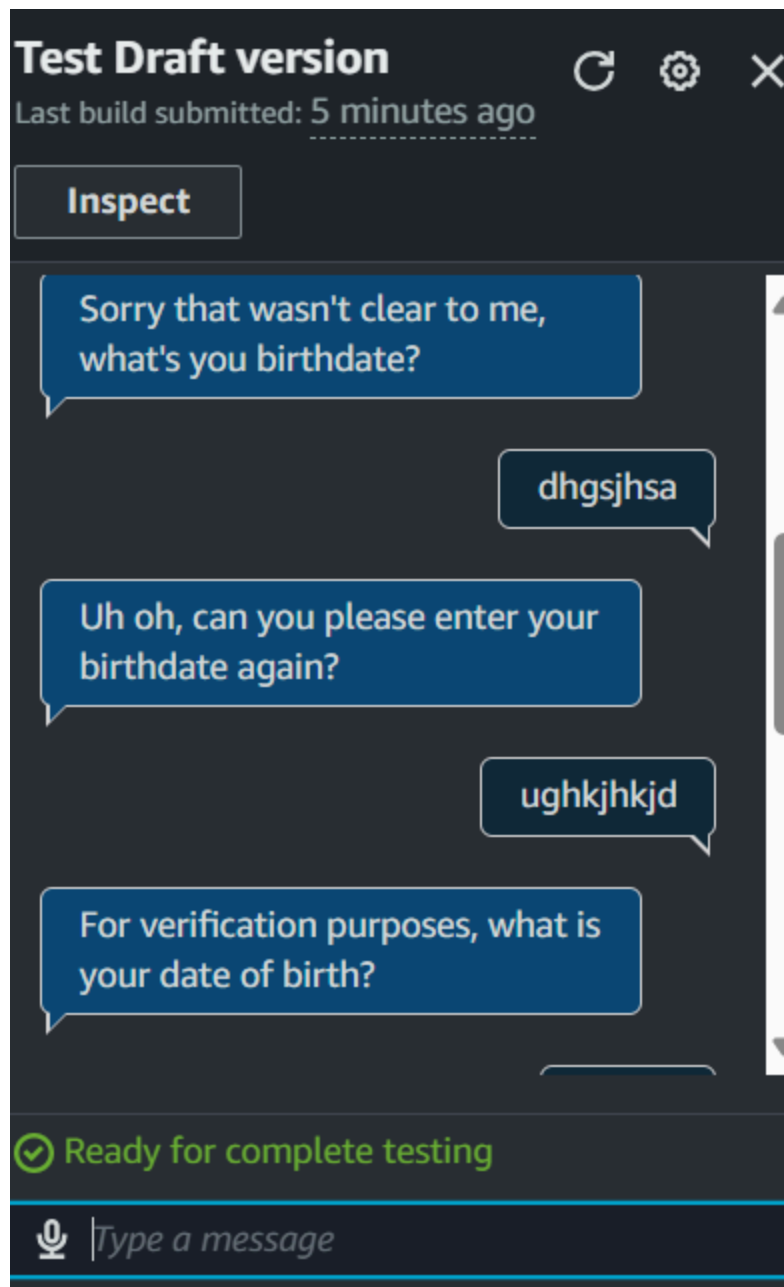
One moment...

Yikes!

✓ Ready for complete testing



Type a message



- ▼ Why did my bot go through all the 3 variations first?
 - Lex is programmed to try all different ways it can get the slot value before resorting to a failure response
- ▼ Why does FallbackIntent shows up?

the failure response activates the FallbackIntent by default. This happens when the bot can't get the date of birth and Lex figures out that maybe the

user is looking for something else like what if the user isn't looking for bank balance

- Since I don't want the FallbackIntent to get triggered. I will go to slot capture: failure response again and under set values make the below changes

The screenshot shows the 'Slot capture: failure response' configuration page in the Amazon Lex console. At the top, there is a blue header with the title 'Slot capture: failure response' and an 'Info' icon. Below the header, a message states: 'You can provide responses, set values, and next steps. You can also branch based on conditions.'

The main configuration area is divided into two columns. The left column is titled 'Set values' and contains a section for 'Response when slot value isn't understood' with a message: 'Message: It looks like the date provided isn't right. Re-enter your birthdate in the following format MM/DD/YYYY or 5th June, 1995'. Below this, there is a 'Slot values - optional' section with a text area containing three lines: '{slot} = "value"', '{slot} = \$.transcriptions[N]...', and '{slot} = [session attribute]'. A note below the text area says 'Separate values with a new line.' The right column is titled 'Next step in conversation' and contains a 'Session attributes - optional' section with a text area containing three lines: '[session attribute] = "value"', '[session attribute] = \$.transcriptions[N]...', and '[session attribute] = {slot}'. A note below the text area says 'Separate values with a new line.'

Below these two columns, there is a 'Next step in conversation' dropdown menu with 'Elicit a slot' selected. Below that is a 'Slot' dropdown menu with 'dateOfBirth' selected. At the bottom, there is a toggle switch for 'Skip elicitation prompt' which is currently turned on.

- update slot → build → test

Test Draft version

Last build submitted: Now



Inspect

Balance in credit

For verification purposes, what is your date of birth?

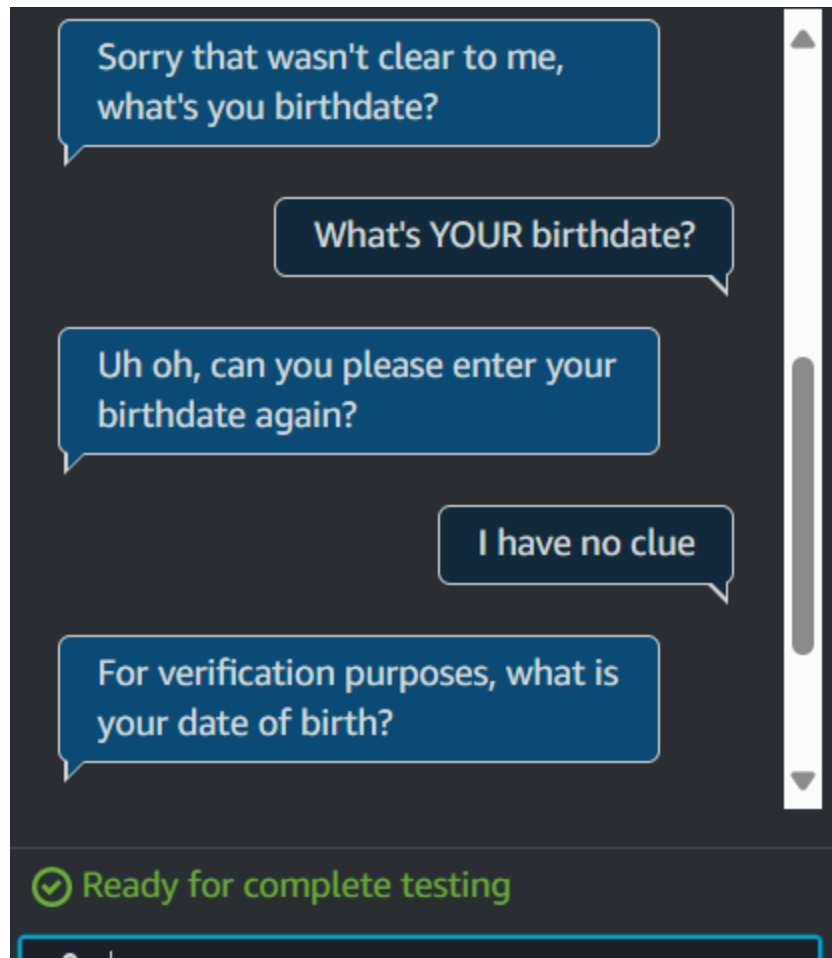
cheese

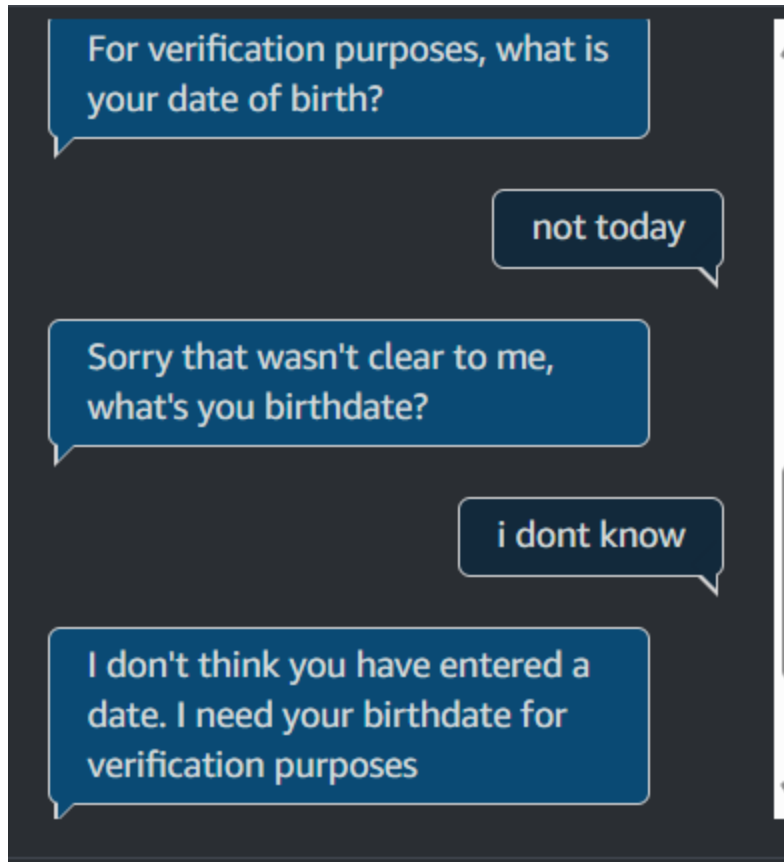
Sorry that wasn't clear to me, what's your birthdate?

✓ Ready for complete testing



Type a message





- Lex doesn't come with the smarts to calculate bank balances on its own, but luckily it doesn't have to. We'll be using another AWS service, **AWS Lambda**, to generate a random number on the fly (whenever a user asks for their balance). You can think of Lex as the interface that the user sees and chats with, while Lambda is the calculator that's out of sight but works in the background.

Creating Lambda Function on AWS Lambda

▼ What is AWS Lambda?

AWS Lambda is a service that lets you run code in the cloud without needing to manage any computers/servers - Lambda will manage them for you.

Lambda runs your code only when needed and scales automatically, from a few requests per day to thousands per second - all you need to do is supply your code in one of the languages that Lambda supports.

- On AWS Lambda we create a function

The screenshot shows the AWS Lambda 'Create function' page. At the top, there are two options: 'Author from scratch' (selected) and 'Use a blueprint'. Below this, the 'Basic information' section contains the following fields:

- Function name:** A text input field containing 'BankingBotEnglish'.
- Runtime:** A dropdown menu set to 'Python 3.12'.
- Architecture:** Radio buttons for 'arm64' and 'x86_64', with 'x86_64' selected.
- Permissions:** A section with a link to 'Change default execution role'.

- I scroll down and add the lambda_function.py (given in my GitHub repo)

```
import json
import random
import decimal

def random_num():
    return(decimal.Decimal(random.randrange(1000, 50000))/100)

def get_slots(intent_request):
    return intent_request['sessionState']['intent']['slots']

def get_slot(intent_request, slotName):
    slots = get_slots(intent_request)
    if slots is not None and slotName in slots and slots[slotName] is not None:
        return slots[slotName]['value']['interpretedValue']
    else:
        return None

def get_session_attributes(intent_request):
```

```

sessionState = intent_request['sessionState']
if 'sessionAttributes' in sessionState:
    return sessionState['sessionAttributes']

return {}

def elicit_intent(intent_request, session_attributes, message):
    return {
        'sessionState': {
            'dialogAction': {
                'type': 'ElicitIntent'
            },
            'sessionAttributes': session_attributes
        },
        'messages': [ message ] if message != None else None,
        'requestAttributes': intent_request['requestAttributes'] if 'requestAttribute
s' in intent_request else None
    }

def close(intent_request, session_attributes, fulfillment_state, message):
    intent_request['sessionState']['intent']['state'] = fulfillment_state
    return {
        'sessionState': {
            'sessionAttributes': session_attributes,
            'dialogAction': {
                'type': 'Close'
            },
            'intent': intent_request['sessionState']['intent']
        },
        'messages': [message],
        'sessionId': intent_request['sessionId'],
        'requestAttributes': intent_request['requestAttributes'] if 'requestAttribute
s' in intent_request else None
    }

```

```

def CheckBalance(intent_request):
    session_attributes = get_session_attributes(intent_request)
    slots = get_slots(intent_request)
    account = get_slot(intent_request, 'accountType')
    #The account balance in this case is a random number
    #Here is where you could query a system to get this information
    balance = str(random_num())
    text = "Thank you. The balance on your "+account+" account is $" +balance+" dollars."
    message = {
        'contentType': 'PlainText',
        'content': text
    }
    fulfillment_state = "Fulfilled"
    return close(intent_request, session_attributes, fulfillment_state, message)

```

```

def FollowupCheckBalance(intent_request):
    session_attributes = get_session_attributes(intent_request)
    slots = get_slots(intent_request)
    account = get_slot(intent_request, 'accountType')
    #The account balance in this case is a random number
    #Here is where you could query a system to get this information
    balance = str(random_num())
    text = "Thank you. The balance on your "+account+" account is $" +balance+" dollars."
    message = {
        'contentType': 'PlainText',
        'content': text
    }
    fulfillment_state = "Fulfilled"
    return close(intent_request, session_attributes, fulfillment_state, message)

```

```

def dispatch(intent_request):
    intent_name = intent_request['sessionState']['intent']['name']
    response = None

```

```
# Dispatch to your bot's intent handlers
if intent_name == 'CheckBalance':
    return CheckBalance(intent_request)
elif intent_name == 'FollowupCheckBalance':
    return FollowupCheckBalance(intent_request)

raise Exception('Intent with name ' + intent_name + ' not supported')

def lambda_handler(event, context):
    response = dispatch(event)
    return response
```

▼ The above python script contains

This Python script helps your chatbot give users quick answers about their account balances.

When someone asks about their account balance to your chatbot, Lex will ask your Lambda function to run this code, which will pick a random number to pretend it's the balance.

Lambda will pass this random number to Lex, who will then push the bank balance figure to the user through your chatbot.

- Now Head back to your **Amazon Lex** console.
- Select **BankerBot**.
- On the left-hand menu, choose **Aliases**.

💡 What are aliases, why are we using them?

Think of an alias in Amazon Lex as a pointer for a specific version of your bot.

So when you're connecting Lex with other AWS services or your custom applications, those external resources will connect to an alias, which will point to the specific version of your bot that you want to use.

Now, instead of always updating your apps to connect to the newest version of the bot, you can just update the alias to point to that new version. All your apps

will automatically start using the updated bot without needing any changes on their end - this saves developers a TON of time and reduces the risk of errors!

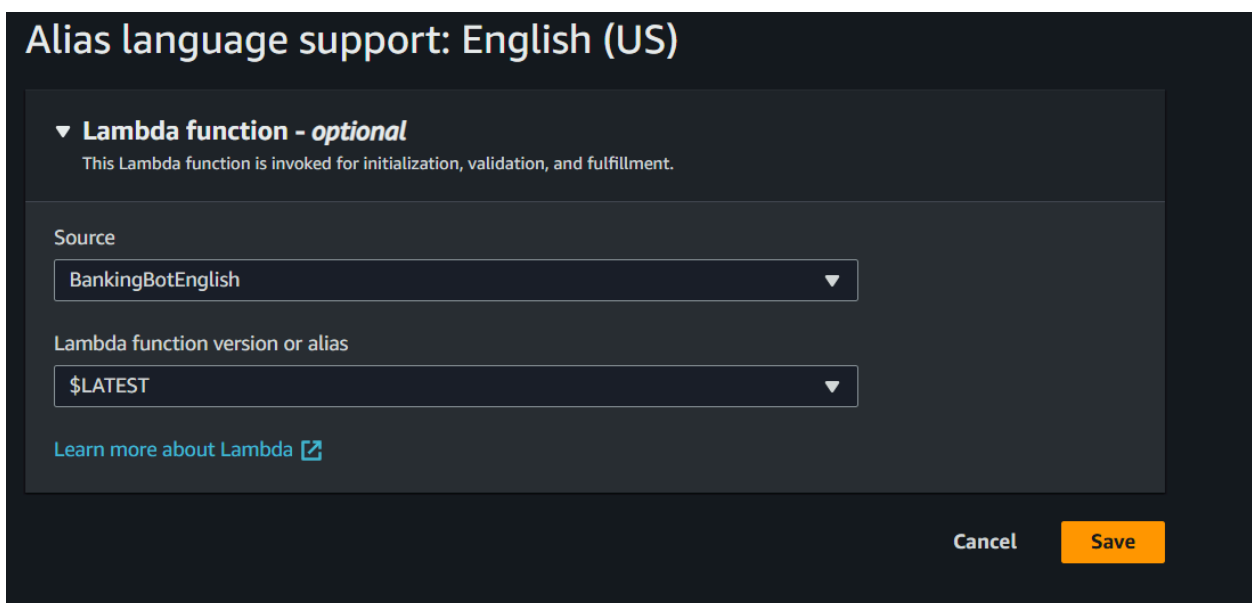
- Click on the default **TestBotAlias**.

What is TestBotAlias?

TestBotAlias is a default version of your bot that's made for testing or development.

This is the playground version of your bot that you'll use to make sure everything works smoothly before rolling out changes!

- On the **Languages** panel, click **English (US)**.
- For **Source**, choose your Lambda function **BankingBotEnglish**.
- Leave the Lambda function version or alias field at the default **\$LATEST**.



Alias language support: English (US)

▼ **Lambda function - optional**
This Lambda function is invoked for initialization, validation, and fulfillment.

Source
BankingBotEnglish ▼

Lambda function version or alias
\$LATEST ▼

[Learn more about Lambda](#)

Cancel Save

- Navigate to your **CheckBalance** intent.
- Scroll down to **Fulfilment** panel.

💡 What is fulfilment?

In Amazon Lex, **fulfilment** means completing the intent.

With your BankingBot, after a user tells your bot:

1. The account they want to check, and
2. Their birthday for verification

The bot has all the information it needs and moves to fulfillment. This is where it will use the Lambda function to get the account balance and pass it back to the user.

- Expand the **On successful fulfillment** bubble.
- Choose **Advanced options**.
- Under the **Fulfillment Lambda code hook** panel, check the checkbox 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.

Cancel Update options

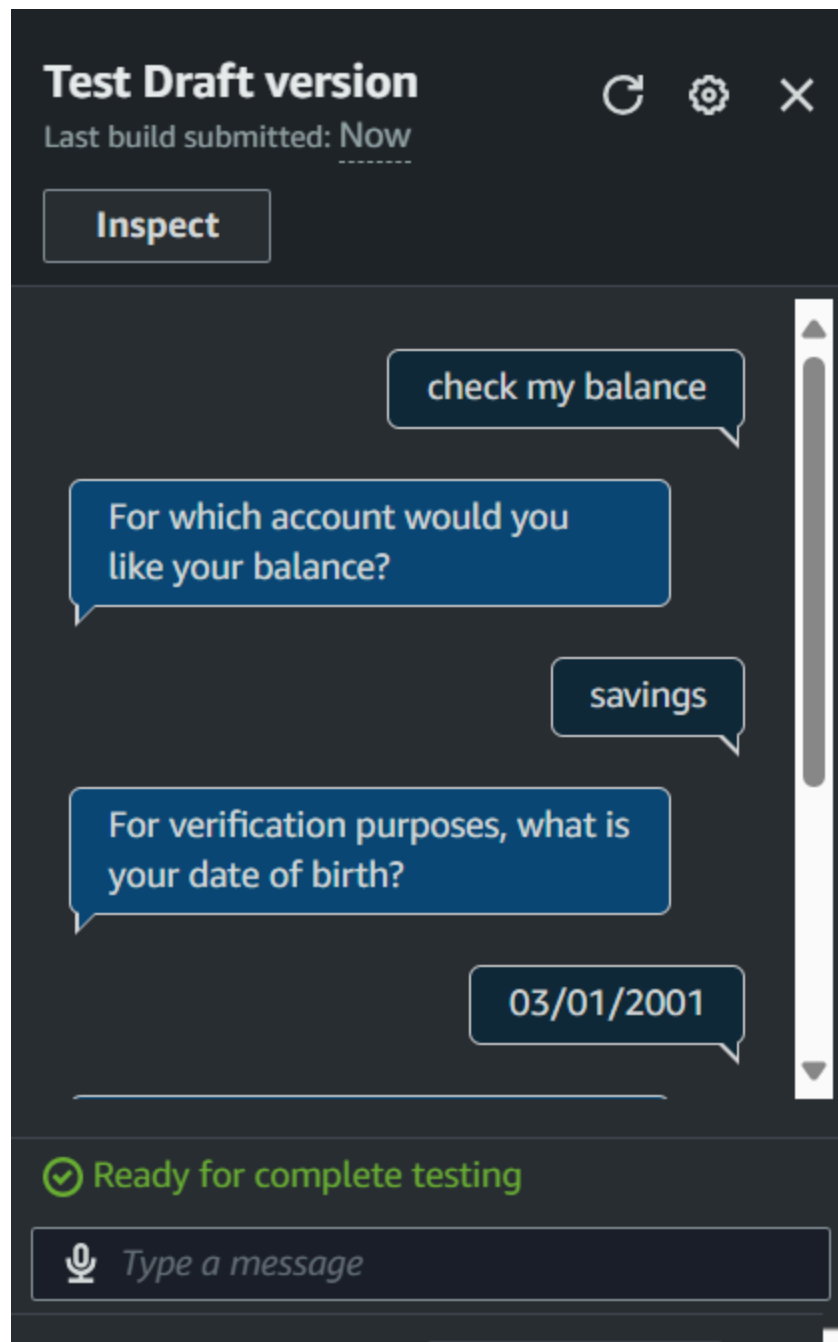
💡 What are code hooks?

Code hooks help you connect your chatbot to custom Lambda functions for doing specific tasks during a conversation.

They're used to handle more complex actions that the basic chatbot setup can't do on its own, like checking data from a database or making decisions based on past conversations.

Essentially, code hooks make your chatbot smarter and more useful by allowing it to perform these extra steps seamlessly during chats.

- Choose **Update options**.
- Choose **Save intent**.
- Choose **Build**
- Choose **Test**.
- Ask for the balance of any of your accounts - your bot should now be able to return (random) bank balance figures!



- Now I will customize my lambda function
- Head back to your AWS Lambda console.
- In the Functions page, click into BankingBotEnglish
Scroll down to the Code source panel.

▼ How does Lex respond with a bank balance figure when i check my account balance

- Recap: When you ask the chatbot about your account balance, it triggers the CheckBalance intent. CheckBalance will fulfill the user's request (i.e. get their bank balance) using a Lambda function called Banking BotEnglish. This function is linked to your bot through an Alias you set up.
- When fulfillment starts, Lex sends a piece of data structured in JSON to the Lambda function. The JSON data will tell Lambda things like the intent name (i.e. CheckBalance) and any relevant slot values the accountType.
- `def dispatch(intent_request)` is the first function that Lambda runs in your Python code. This function will read the JSON data, spot that CheckBalance is this request's intent name, and then run the matching CheckBalance function inside your Lambda code to generate the bank balance figure.

▼ How does the CheckBalance function work?

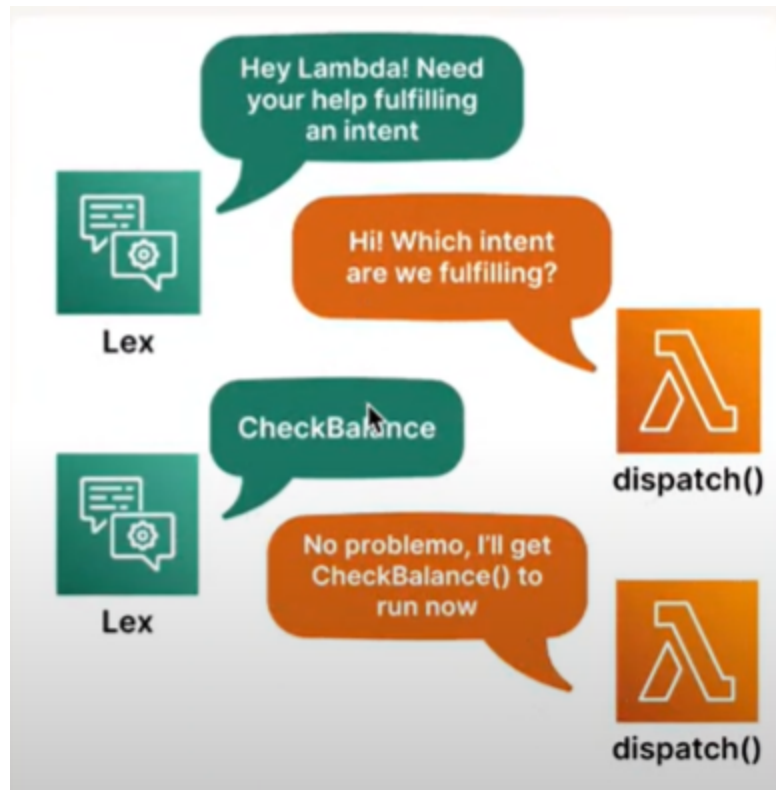
The CheckBalance function will:

1. Get the Lex request's accountType using another function called `get_slots()` in the same code snippet
2. Generate a random number using another function called `random_num()` in the same code snippet
3. Combine the random number and the accountType in a text message that says Thank you. The balance on your "ACCOUNT-TYPE" account is \$"RANDOM-NUMBER" dollars.
4. Send that text message to Lex using another function called `close()` that packages up the response

▼ Why do we need dispatch function?

- Right now in our BankerBot, CheckBalance is the only intent using a Lambda function.
- Butttt here's a small spoiler on the next project-the next intent you set up, FollowUpCheckBalance, will also use this Lambda function!

- Because the same piece of code can actually handle two intents, your code has split up the work for each intent into their own mini functions. That's why there's a `CheckBalance()` function and a `FollowUpCheckBalance()` function in your code. Lambda needs to run `dispatch()` to figure out the intent of each request from Lex, which lets Lambda decide which of the two intent's functions it should run next.



▼ I replaced the following code snippet with

```
text = "Thank you. The balance on your "+account+" account is $" + balance + " dollars."
```

- With the below code snippet

```
descriptions = ["Aprajita's top secret digital vault","your personal treasure box", "Bank of Aprajita"]
description = random.choice(descriptions)
```

```
text = f"Hello! Your {account} account balance, tucked away in {description}, is currently ₹{balance}"
```

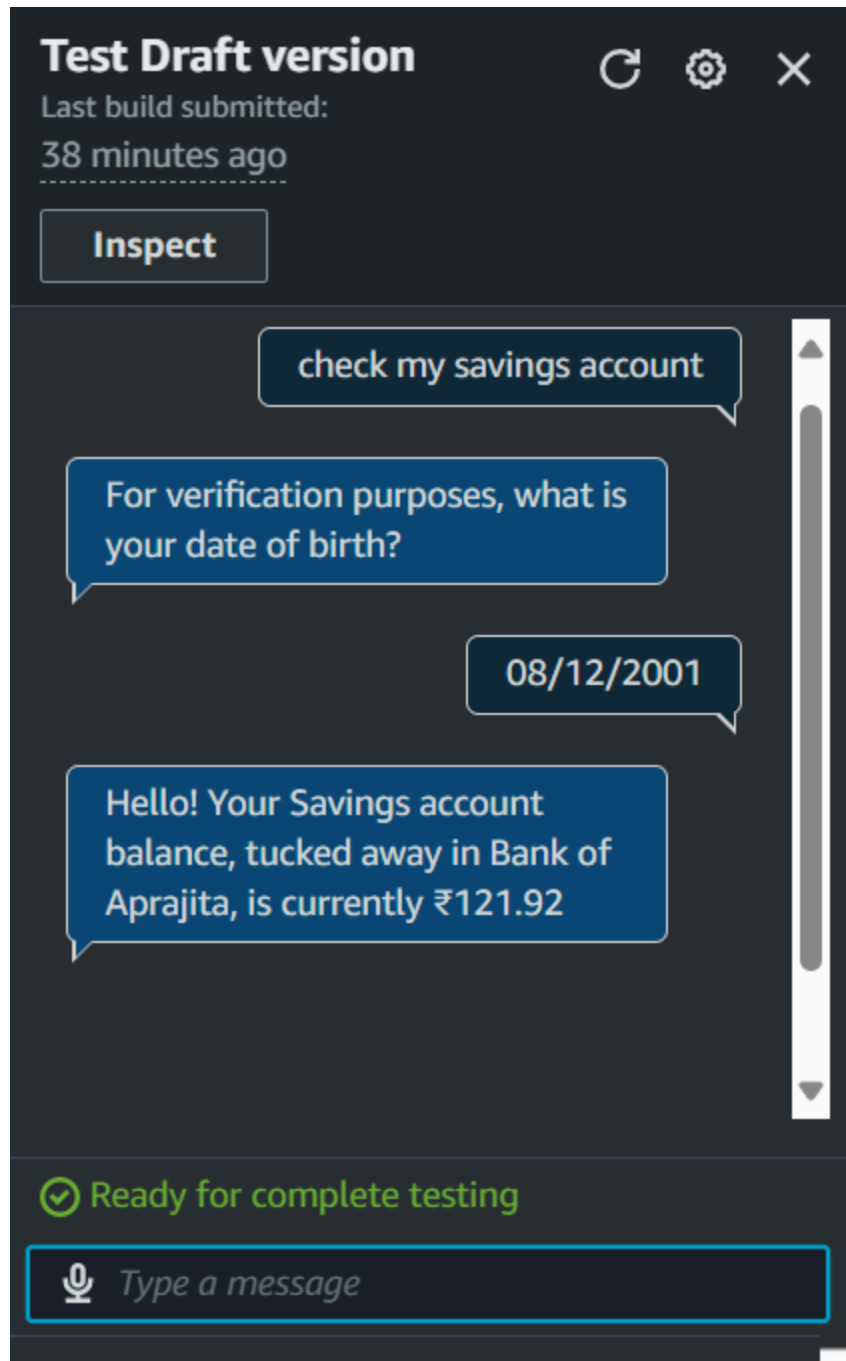
▼ What changes am I making to my code?

I am updating the message that my chatbot would send back to users when they're checking their account balance.

1. The first line, descriptions= writes a list of different funky locations that your user's money could be stored in.
2. The second line, description=random.choice() will randomly pick one of the locations in that list every time the user checks their balance.
3. Then, text pieces together the randomly picked location with the bank balance generated earlier in this function! Lambda will now send back this updated text message.

```
def CheckBalance(intent_request):  
    session_attributes = get_session_attributes(intent_request)  
    slots = get_slots(intent_request)  
    account = get_slot(intent_request, 'accountType')  
    #The account balance in this case is a random number  
    #Here is where you could query a system to get this information  
    balance = str(random_num())  
    descriptions = ["Aprajita's top secret digital vault", "your personal treasure box", "Bank of Aprajita"]  
    description = random.choice(descriptions)  
    text = f"Hello! Your {account} account balance, tucked away in {description}, is currently ₹{balance}"  
    message = {  
        'contentType': 'PlainText',  
        'content': text  
    }
```

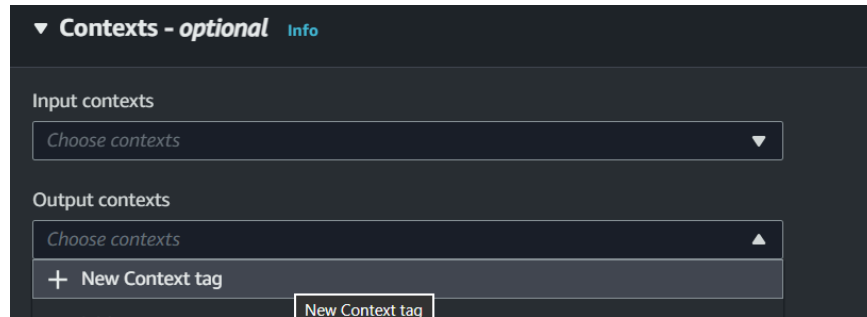
- I am making those changes to the code in order to level up the connection between lambda and lex. I added random descriptions so that the message is fun and unique
- Select Deploy next to your code snippet window. Head back to your Lex console.
- Still in your CheckBalance intent, select Test and test your bot again! If you already have a test window open, make sure to select the refresh icon.



- In your **CheckBalance** intent page, scroll down to the **Contexts** panel.

Creating Context Tags

- Under the **Output contexts** drop-down, choose **New context tag**.

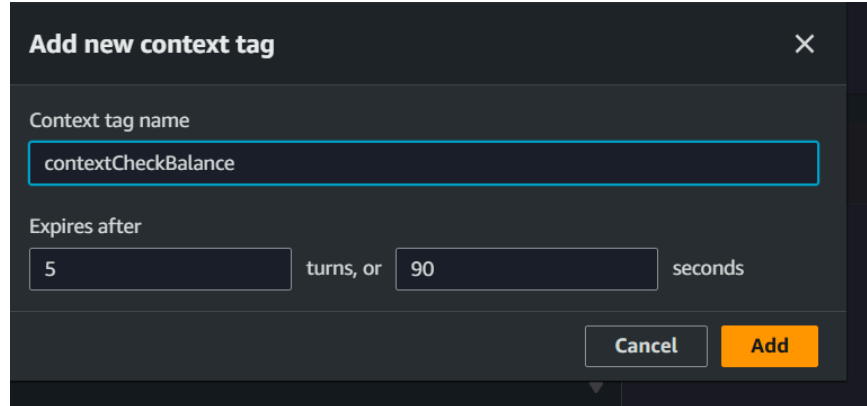


What are context tags?

Context tags in Amazon Lex 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 in Amazon Lex:

1. **Output context tag:** This tells the chatbot to remember certain details after an intent is finished, so other parts of the conversation can use this stored information later. For example, the account type from BalanceCheck could be saved and reused
 2. **Input context tag:** This checks if specific details are already available before an intent activates. For example, FollowupCheckBalance will check if this conversation already has the user's date of birth saved somewhere, so it won't need to ask for that information again.
- Name your new context contextCheckBalance
 - Set up the timeout for **5 turns**, or **90 seconds**. We will keep this short so your chatbot doesn't remember a user's birthday for too long (which might become a security risk).



Add new context tag [X]

Context tag name
contextCheckBalance

Expires after
5 turns, or 90 seconds

Cancel Add

- Choose **Add**.
- Choose **Save intent**.
- Choose **Build** - time for a few quick questions.
- Choose **Test**.
- Check that the bot still operates the same as usual - i.e., no errors have popped from creating the context tag.

Creating a FollowupCheckBalance

- From your left hand navigation panel, head back to the **Intents** page.
- Choose **Add intent**.
- Choose **Add empty intent**.
- Under the Intent details pane, use the following properties to set up your next intent:
 - Name: FollowupCheckBalance
 - Description: Intent to allow a follow-up balance check request without authentication.

▼ **Intent details** [Info](#)

Intent name

Maximum 100 characters. Valid characters: A-Z, a-z, 0-9, -, _

Description - *optional*

Maximum 2000 characters.

- Under the Contexts pane, select **contextCheckBalance** from the Input context:
- Under the Sample utterances, enter the following:

How about my {accountType} account?
 What about {accountType} ?
 And in {accountType} ?

Sample utterances (5) [Info](#)

Representative phrases that you expect a user to speak or type to invoke this intent. The intent classification output.

1	How about my {accountType} account?
2	What about {accountType} ?
3	And in {accountType} ?
4	

- Add a new slot:
 - Name: accountType
 - Prompt: For which account would you like your balance?

- Slot type: **accountType**

- Add another new slot:
 - Name: dateOfBirth
 - Prompt: For verification purposes, what is your date of birth?
 - Slot type: **AMAZON.Date**

- Choose **Save intent**.
- Still in your **FollowupCheckBalance** intent page, expand the **dateOfBirth** slot.
- Choose **Advanced options**.

- Scroll all the way down to the **Default values** panel.
- This panel lets us create default values for the intent's slots.
- Enter the below:

```
#contextCheckBalance.dateOfBirth
```

💡 What does #contextCheckBalance.dateOfBirth mean?

This tells Amazon Lex that the input context contextCheckBalance should have the value of dateOfBirth in CheckBalance.

- Choose **Add default value**.
- Choose **Update slot**.

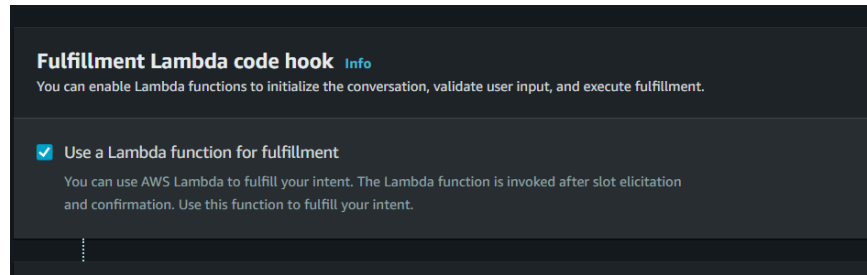
▼ Default values - *optional*

No default values
You haven't added any default values yet.

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

#contextCheckBalance.dateOfBirth Add default value

- Head to the **Fulfillment** pane to make sure the Lambda function is also connected to this intent (so random balances are still being returned to the user).
- Expand **On successful fulfillment**.
- Choose **Advanced options**.
- Head to the **Fulfillment Lambda code hook** panel.
- Select the checkbox to enable a fulfillment Lambda for this intent.

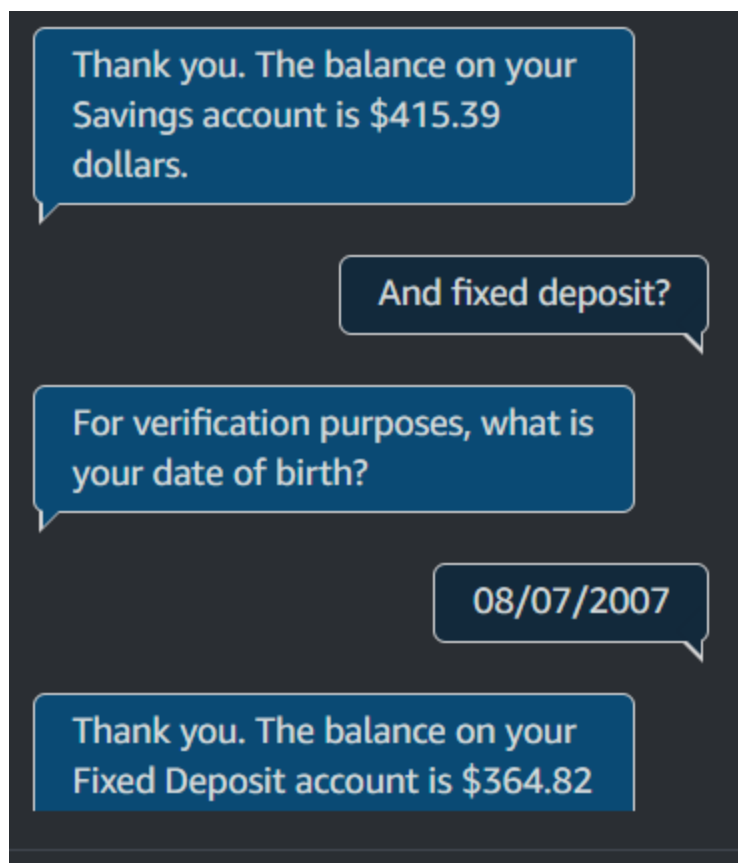
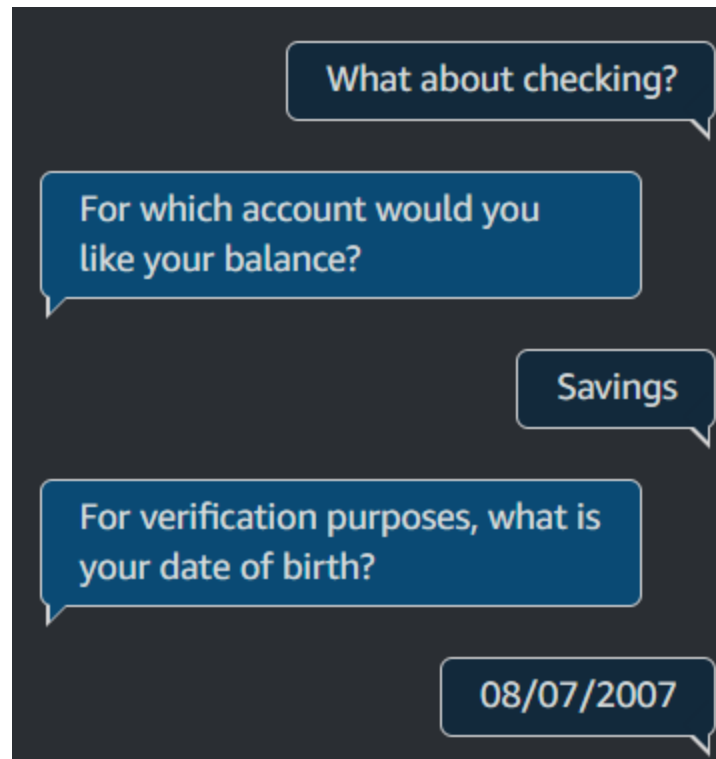


- Choose **Update options**.
- Choose **Save intent**.
- Choose **Build** - you know the drill!
- Choose **Test**.
- In first test, try to trigger the new FollowupCheckBalance intent you've just created **without** triggering CheckBalance first.
 - e.g. ask your chatbot What about checking?

Why isn't the intent working?

No matter which utterance you use, you will just get an error response. This is because the FollowupCheckBalance intent's input context isn't available yet. The intent doesn't know your birthday!

- For second test, ask for a balance in your account - activate the **CheckBalance** intent first.
- Then, the context for date of birth will be carried over to the **FollowupCheckBalance** intent.
- The bot now uses the context from your first check balance request, and doesn't ask for your birthday again for your second request.



- Head back to the CheckBalance intent.
- Scroll down to the Contexts panel.
- Notice the subtext under contextCheckBalance - it says the context expires in 5 turns or 90 seconds.

Context Expiry

▼ What does context expiry mean?

- Think of context expiry as how long your bot will remember information for. In this case, our BankerBot will only remember information from the CheckBalance intent for:
 - 5 turns i.e. once your chatbot sends another 5 messages after gathering slot values from the user (accountType and dateOfBirth), your bot's memory is wiped clean
 - 90 seconds i.e. once 90 seconds has passed since your bot gathered slot values from the user (accountType and dateOfBirth), your bot's memory is wiped clean.
- Select Test and let's test your bot again.
- Trigger the CheckBalance intent again:
 - Check my Savings account balance
- Pass a date of birth:
 - 19th January 1993
- Open the Inspect window.
- Notice that in the last row, contextCheckBalance is active and says there are 5 turns or 90 seconds left!
- Now let's activate FollowUpCheckBalance:
 - What about Checking?
- Notice how contextCheckBalance updated itself to just 4 turns and bit less time left.

- Make sure at least 90 seconds has passed since you've passed a date of birth to your chatbot.
- In your test window, try to trigger the FollowUpCheckBalance intent again:
 - What about Credit?
- ▼ My bot doesn't give me an account balance anymore!
 - That's right, your bot is going to give you the Fallbackintent response instead.
 - The saved context on your birthday has expired now, so your bot doesn't remember that you've asked for an account balance before. Your bot can't accept your FollowUpCheckBalance request!
 - You might also notice that your chat's Inspect window is cleared, which tells us that there are no active contexts running anymore.
 - Let's customise your context's expiry and make it even shorter
- Back in your CheckBalance intent page, stay in the Contexts panel.
- Select the gear icon in the contextCheckBalance option.
- Select Save.
- Select Save intent.
- Select Build.
- Select Test.
- Trigger the CheckBalance intent:
 - Check my Savings account balance
- Pass a date of birth:
 - 19th January 1993
- Open the Inspect window.
- In the last row, contextCheckBalance is active and says there are 1 turns or 5 seconds left!
- Now let's activate FollowUpCheckBalance:

- What about Checking?
- ▼ Why am I seeing Fallbackintent so quickly?

In our last test, our bot could still return another bank balance when we asked about the Chocking account.

Your bot's memory is much, much shorter now, so it'll forget context from your first balance check in just 1 message, or 5 seconds.

▼ How is context expiry useful?

- As you might imagine, having a short memory becomes very useful in situations where information is sensitive and you'd want to protect the user from attackers getting access to chat windows with lots of active contexts.
- On the other hand, having a longer memory would be helpful if you expect the user to have long conversations and run through multiple intents in the same session. Ideally they wouldn't have to share the same information again and again!
- a long context expiry window is helpful when we want our bot to remember the user's information for a longer conversation
- a shorter context expiry window is preferred when we want a more secured conversation especially when sharing sensitive data
- now I will create new transfer funds so that our users can transfer funds between accounts

Creating the intent TransferFunds

- Create a new empty intent with the following properties:
 - Name: TransferFunds
 - Description: Help user transfer funds between bank accounts

Add empty intent ✕

Create a custom intent for your bot.

Intent name

TransferFunds

Maximum 100 characters. Valid characters: A-Z, a-z, 0-9, -, _

Cancel Add

▼ Intent details [Info](#)

Intent name

TransferFunds

Maximum 100 characters. Valid characters: A-Z, a-z, 0-9, -, _

Description - *optional*

Help user transfer funds between bank accounts

Maximum 2000 characters.

- Sample utterances:

Can I make a transfer?

I want to transfer funds

I'd like to transfer {transferAmount} from {sourceAccountType} to {targetAccountType}

Can I transfer {transferAmount} to my {targetAccountType}

Would you be able to help me with a transfer?

Need to make a transfer

- Slots: add a new slot called sourceAccountType, with the prompt Which account would you like to transfer from? and the slot type **accountType**.

Add slot

A slot is used to capture information from the user to fulfill the intent.

☒ Required for this intent
The bot will prompt for this slot during the conversation if a value is not provided by the user.

Name

Slot type

sourceAccountType

accountType

Prompts

Which account would you like to transfer from?

Cancel

Add

- Slot: add another new slot called targetAccountType, with the prompt Which account are you transferring to? and the slot type **accountType**.

Add slot

A slot is used to capture information from the user to fulfill the intent.

☒ Required for this intent
The bot will prompt for this slot during the conversation if a value is not provided by the user.

Name

Slot type

targetAccountType

accountType

Prompts

Which account are you transferring to?

Cancel

Add

- Slot: add another new slot called transferAmount, with the prompt How much money would you like to transfer? and the slot type **AMAZON.Number**.

☒ Required for this intent
The bot will prompt for this slot during the conversation if a value is not provided by the user.

Name

Slot type

transferAmount

AMAZON.Number

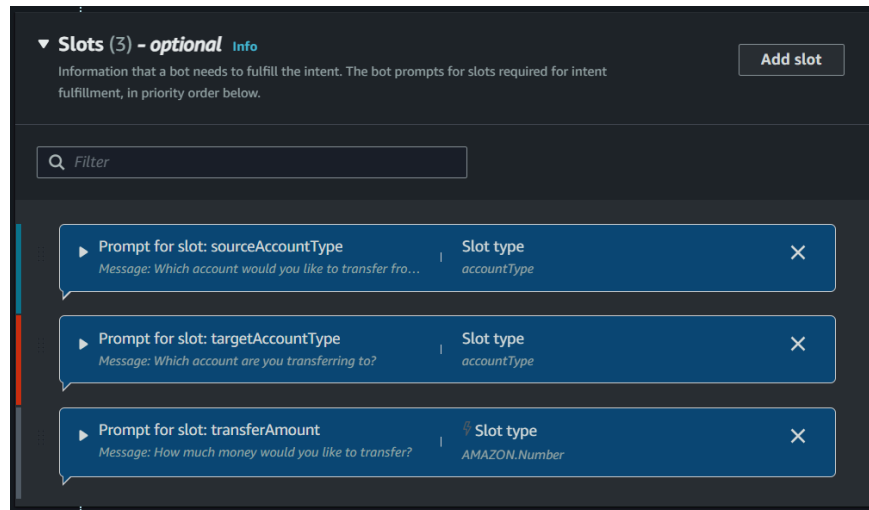
Prompts

How much money would you like to transfer?

💡 Two of my slots have the same slot type!

It's totally possible for multiple slots to have the same slot type.

When two slots have the same slot type, it becomes important that you're using clear slot names, like `sourceAccountType` and `targetAccountType`, to make it easy to identify their differences.



- We now want to add ✨ confirmation prompts ✨

💡 What are confirmation prompts?

Confirmation prompts typically repeat back information for the user to confirm. e.g. "Are you sure you want to do x?" If the user confirms the intent, the bot fulfills the intent

If the user declines, then the bot responds with a decline response that you set up.

- Scroll to the **Confirmation** panel.
- In the **Confirmation prompt** panel, enter the following:
 - Got it. So we are transferring {transferAmount} from {sourceAccountType} to {targetAccountType}. Can I go ahead with the transfer?
- In **Decline** response, enter:
 - The transfer has been cancelled.

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
Message: Got it. So we are transferring {transferAmount}...	Message: The transfer has been cancelled.

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.

- Scroll to the **Closing response** pane, and add this to the Message field:
 - The transfer is complete. {transferAmount} should now be available in your {targetAccountType} account.

Closing response Info Active

You can define the response when closing the intent.

▼ **Response sent to the user after the intent is fulfilled**
Message: The transfer is complete. {transferAmount} should now be available in your {targetAccountType} account.

▼ **Message group** Info
You can define a text message group to respond using plain text.

Message

The transfer is complete. {transferAmount} should now be available in your {targetAccountType} account

► Variations - optional

More response options
Add custom payloads, SSML, and card groups.

- Choose **Save intent**.
- Choose **Build**
- Choose **Test**.
- Now let's test out your chatbot! Ask your chatbot: I'd like to transfer money.
- Complete the conversation with your bot!

Exploring Amazon Lex

▼ Exploring Lex

Conversation flow

- Scroll to the very top of this intent's page.
- Head to the **Conversation flow** panel.
- Expand the arrow to see an example conversation flow 🙄👄🙄

💡 Oooo so what's the conversation flow feature for?

1. This flow will update as you continue editing this intent.
2. It shows every step in a conversation in a logical, chronological order.
3. You'll also see some blank 'ghost like' responses. These are recommendations for what you could add to your Intent set up and they're clickable!
4. If you click on the chat bubble, you'll get taken to an edit screen.

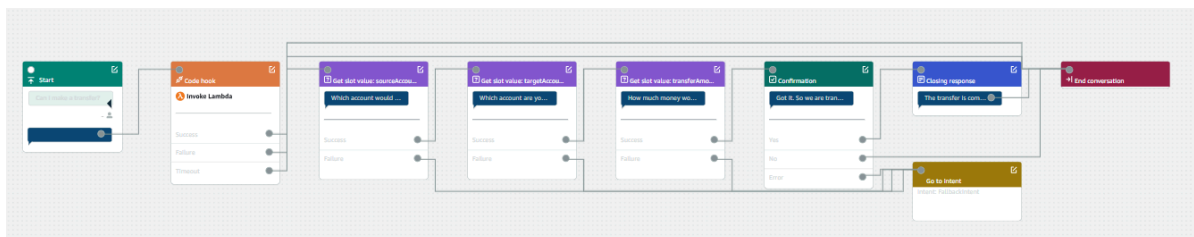
We won't be adding any of these to keep our chatbot simple for today, but it's a pretty handy tool if you're ever creating your own chatbot 😊

Visual builder

- Now look at the bottom bar of your screen.
- Select **Visual builder**.
- Now look at that! This is a visual representation of the intent you have just built.

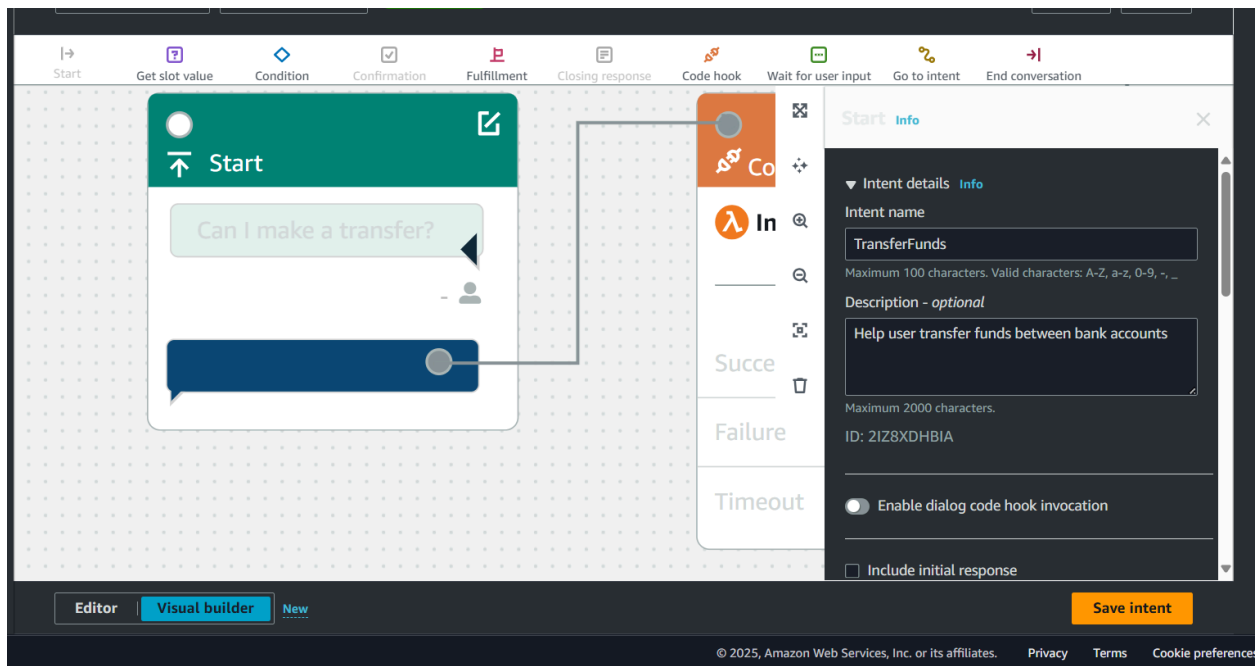
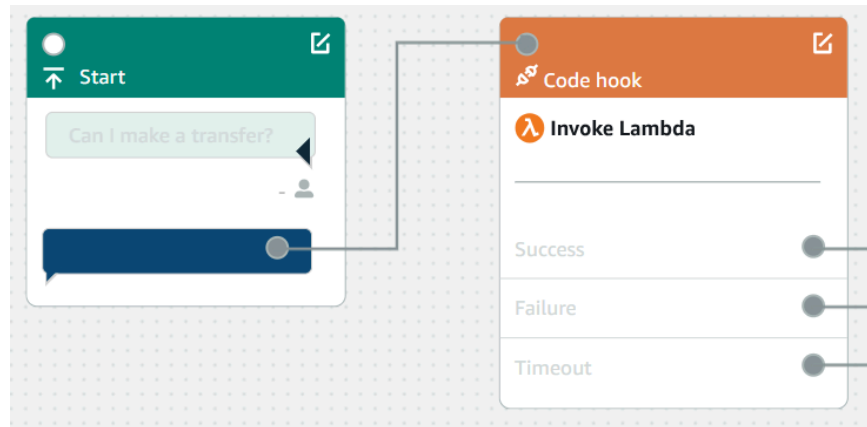
In the future, you could use the Visual builder to build your intent from scratch (not just view the flow itself)!

We won't be doing this today (unless you're unlocking today's secret mission 🧐), but it's definitely a fun way to create chatbots.



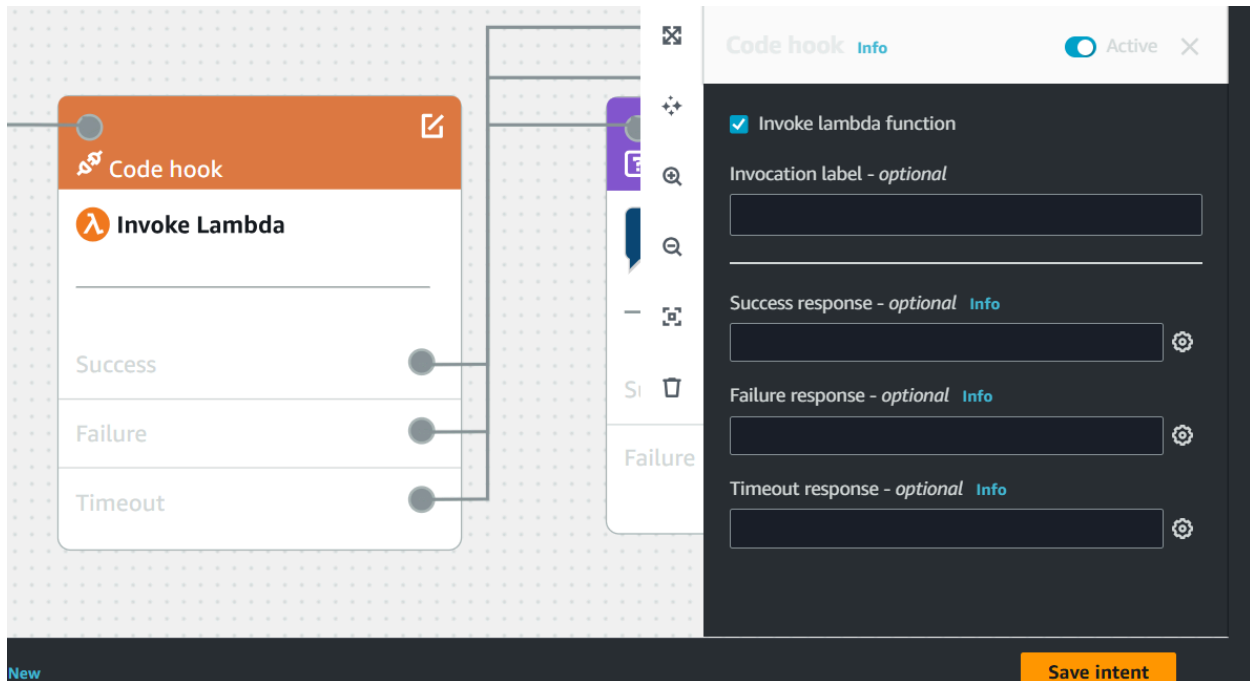
- I will now use the visual builder to add a new step to the TransferFunds intent

- We'll start with getting to know the Start card that triggers TransferFunds. Scroll to the left hand side of your screen.
- Click on the pencil icon on the top right hand corner to reveal an editor mode that lets you add new utterances.

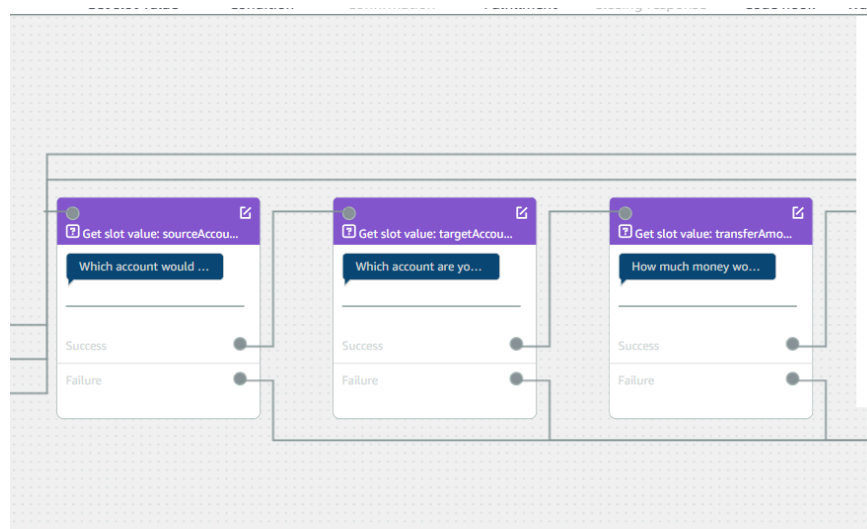


- So what does the Start card do?
 - The Start card is where you set up BankerBot's trigger to begin the TransferFunds process.
- Next is the Lambda code hook. The visual builder sees connecting Lex with Lambda as its own event.

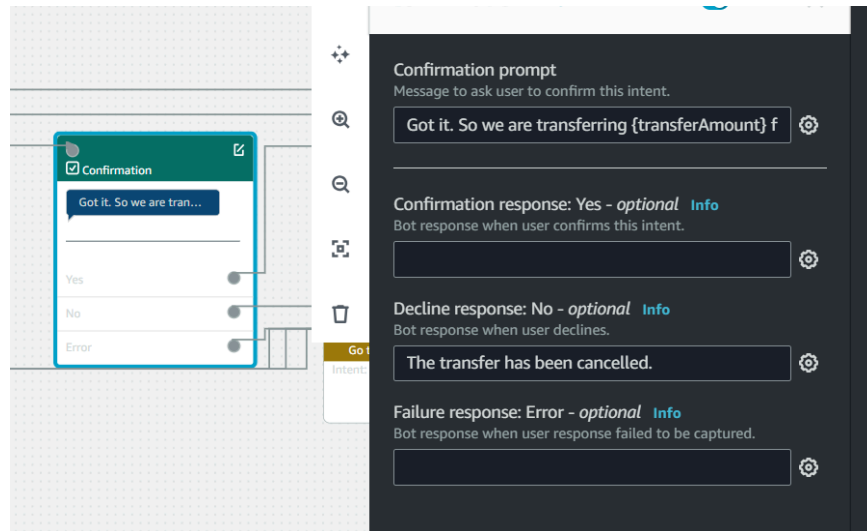
- Wait a second... but Transfer Funds doesn't have any connection with Lambda. That's true! This code hook is at the start of the flow, to prepare for events that can happen later on in your event. But, because you're not actually using a Lambda function in this intent, this card is virtually skipped in your visual builder until you need it.



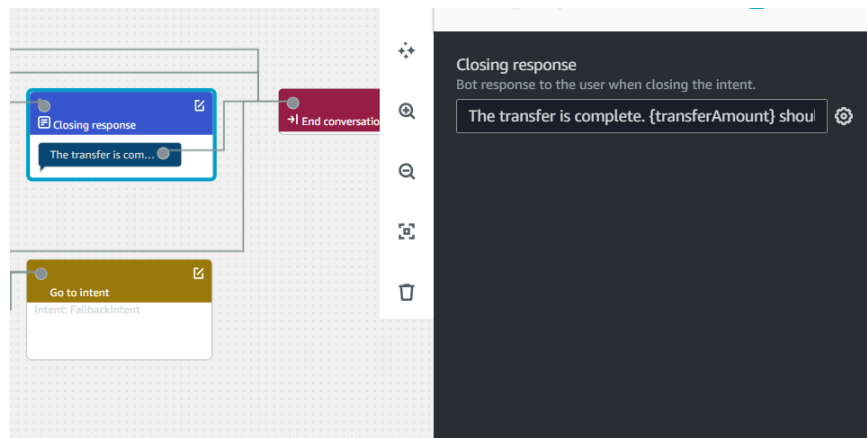
- Next we have a row of Get slot value cards. This is where the chatbot will ask a user for the accounts and the amount of transfer



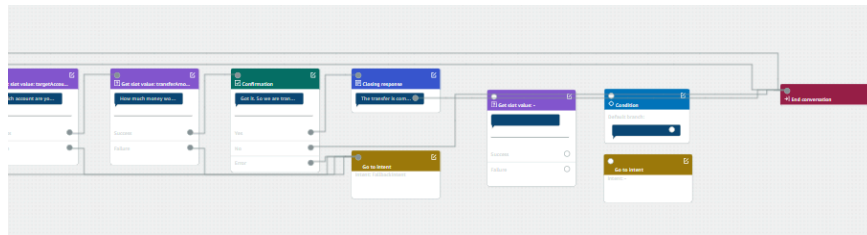
- Notice how there are Success arrows that points the user to the next question if they've given valid responses, while Failure arrows would take the user to the Fallbackintent instead.
- Once your user gives your chatbot all the information it needs for the transfer, can you remember what your chatbot does next?
- If you've noticed that dark green Confirmation card pop up next -bingo.



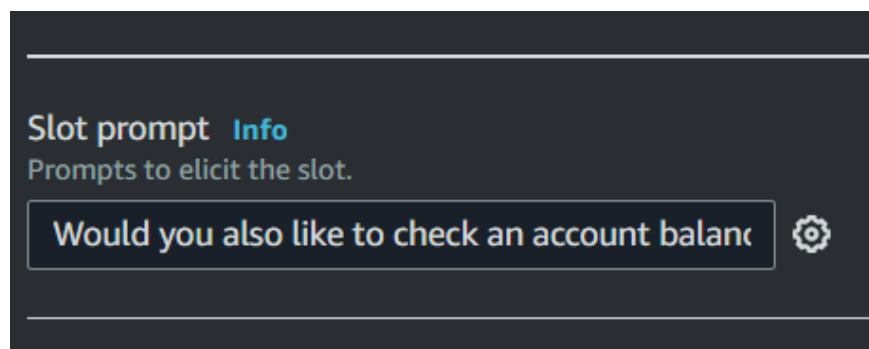
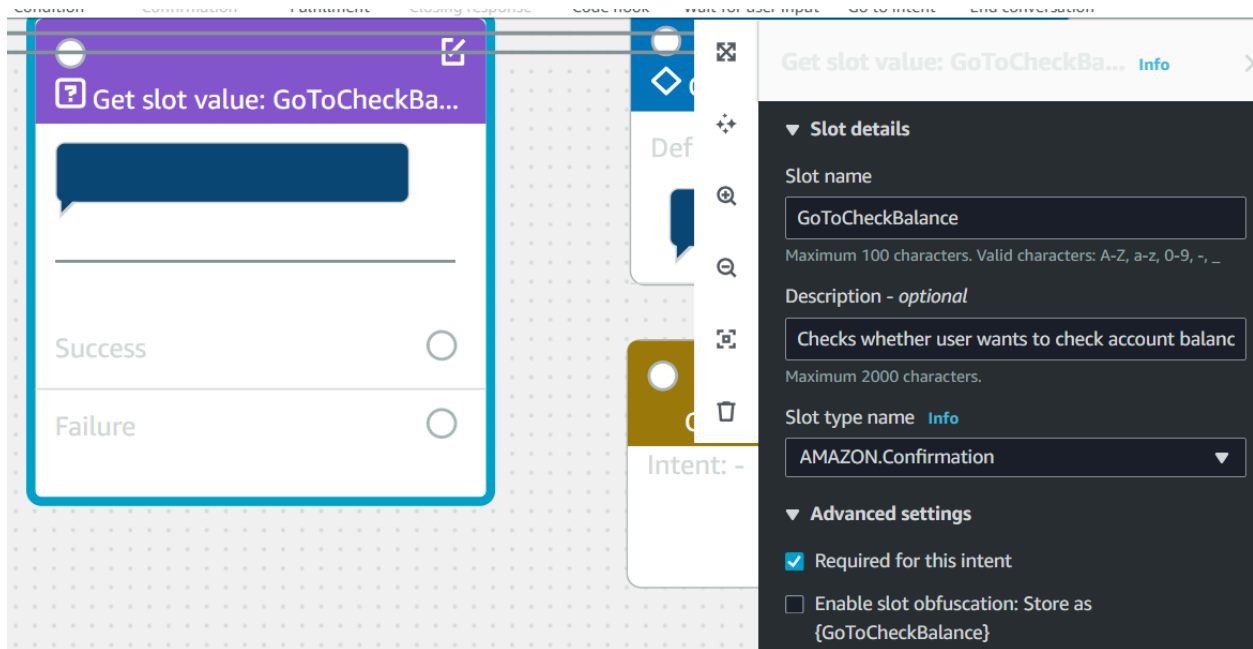
- Click on the pencil icon on the top right hand corner of the Confirmation card. You can even customise the confirmation prompt and responses for every possible scenario!
- Finally the intent ends at providing a closing response to the user and then ending the conversation



- Now that you've learnt the flow of your TransferFunds intent in the visual builder, let's wire up three new cards!
Here's your challenge: How would you set up your bot to automatically ask the user whether they'd like to check an account balance after a transfer?
- Click on red End conversation card and drag it to the far right of the screen. Let's add some space between our last two cards for some new things.
- From the visual builder's top menu bar, drag the Get slot value option in your workspace. Plop it right next to your Closing response card!
- Next, let's drag the Condition option from the top menu bar to sit right next to Get slot value.
- Finally, we'll drag the Go to intent option from the top bar and place it underneath your Condition card.



- Now let's dive into the details for each card.
- Select the pencil icon on the top right corner of your Get slot value card.
- Under Slot name, enter GoToCheckBalance
- Under Description optional, enter "Checks whether the user wants to check account balance."
- Under Slot type name, select Amazon.CONFIRMATION



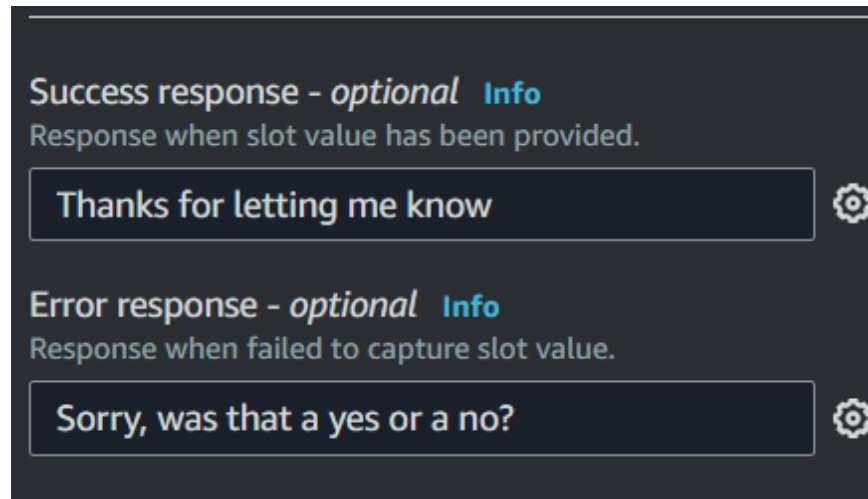
▼ What does Amazon.CONFIRMATION slot type mean?

The Amazon.CONFIRMATION slot type is for understanding phrases that match one of four categories: Yes, No, Maybe and Don't know.

Example phrases that AMAZON.Confirmation is programmed to recognise:

1. Yes: Yeah, Yep, Ok, Sure, I have it, I can agree...
 2. No: Nope, Negative, Naw, Forget it, I'll decline, No way...
 3. Maybe: It's possible, Perhaps, Sometimes, I might, That could be right...
 4. Don't know: Dunno, Unknown, No idea, Not sure about it, Who knows....
- Make sure the Required for this intent checkbox is ticked.

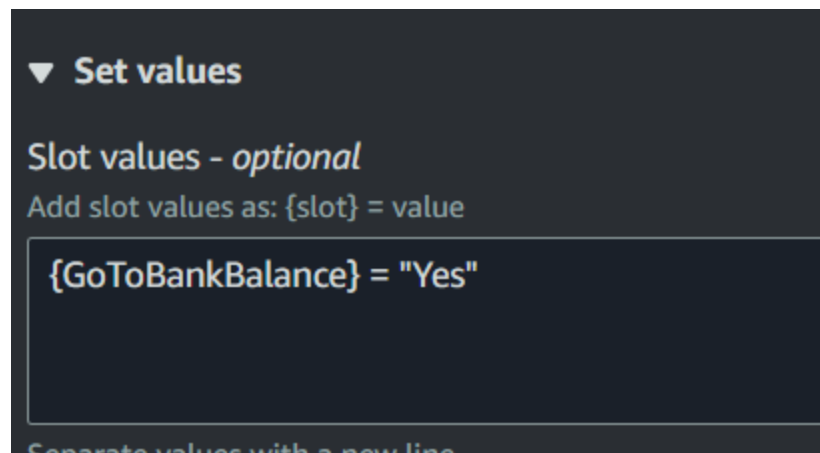
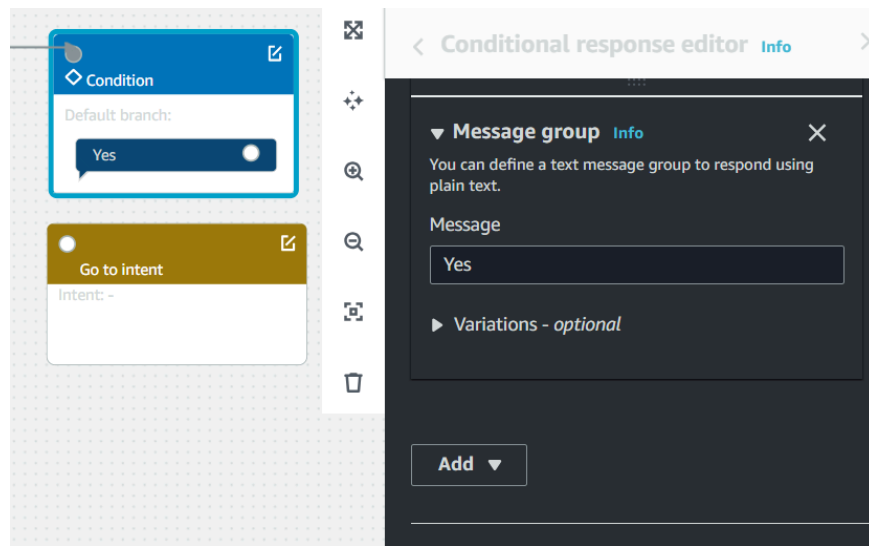
- Under Slot prompt, enter Would you also like to check an account balance?
- Let's add two more customizations to this card. Scroll all the way down to Success response-optional. Add the response "Thanks for letting me know"
- Under Error response, add the response "Sorry, was that a yes or a no?"



- Now let's wire things up in your blue Closing response card, select the arrow pointing out from the The transfer is complete message.
- Press backspace/delete on your keyboard.
- ▼ Why are we deleting this arrow?

We don't want our bot to close the conversation after confirming the transfer anymore. Now we want the bot to ask the user if they want to check an account balance! Let's delete the existing arrow that ends the conversation and rewire our Closing response.
- Click and hold the empty circle next to the The transfer is complete message. Drag your mouse over to the start of your Get slot value card.
- Nice, now let's drag an arrow from the Success outcome in your Get slot value card. The conversation should move towards the Condition card if it was a success.
- Next, if the slot capture was a Failure (i.e. your user can't confirm whether they want to check an account balance), move the conversation to the Fallbackintent.

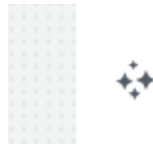
- Next up, select the pencil icon on the top right of your Confirmation card.
- Under Branch name, enter 'Yes'. This sets up what we want to do if the user wants to check their account balance.
- Under Condition, tell Lex that we'll use the Yes branch if the user's response matches a 'Yes' in Amazon.CONFIRMATION: '{GoToCheckBalance} = "Yes"'



- ▼ So what does a Condition card do?
 - All a Condition card does is help you split up your user's journey into two (or multiple) pathways.
 - Now that we've set up a condition for the user responding "yes", that means the default branch will catch all scenarios where the user responds

with other confirmation values e.g. "no", "not sure", "maybe not".

- We'll wire things to complete the Condition:
 - Drag an arrow from the Default branch to End conversation.
 - Drag an arrow from the Yes branch to your new Go to intent card.
- Finally, let's select the pencil icon on the top right corner of your Go to intent card.
- Can you tell which intent we'll take the user in this card?
- Under the Intent name dropdown, select CheckBalance.
- to tidy up your work, use handy **Auto Arrange** button



- Save → build → Test

Deployment

What is AWS CloudFormation?

AWS CloudFormation is a 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. Then, you can use that template to create, update, and delete the entire stack of resources you described, instead of managing your resources individually

- Head to **CloudFormation** in your AWS Console.
- Choose **Create stack**.
- Choose **Choose an existing template**.
- Choose **Upload a template file**. (Given in GitHub Repo)
- Upload your CloudFormation template!

- Choose **Next**.
- For **Stack name**, you can enter banker-bot
- Choose **Next**.
- Scroll to the bottom and select the checkboxes under **Capabilities and transforms**.
- Choose **Next** again! We can skip Stack configuration in this simple implementation.
- Choose **Submit**.
- Your stack might take around 5 minutes to create. Wait until the status of your **Stacks** page says **CREATE_COMPLETE**.
- Head to your Amazon Lex console, and you'll notice that your bot is all created and ready for testing already.
- Click into **banker-bot-BankerBot**.
- Select **Intents** from your left-hand navigation menu.
- Connect your **TestBot Alias** with a Lambda function. Select the one with **V2** in the name.

Lex > ... > Bot: network-banker-bot-... > Aliases > Alias: TestBotAlias > Alias language support: English (US)

Alias language support: English (US)

▼ **Lambda function - optional**
This Lambda function is invoked for initialization, validation, and fulfillment.

Source

nextwork-banker-bot-LexBotFunction-CPo15kBDYCI5

nextwork-banker-bot-LexV2CfnCr--LexV2CfnCrFunction-5KRP8bZnKcmV

[Learn more about Lambda](#)

Cancel Save

- Select **Save**.
- Under the **Resource-based policy statements**, select **Add permissions**. Let's create a new Resource-based policy statement that gives your chatbot aliases access to your new function.
- Under the **edit policy statement** panel, select AWS service and enter the following details:
 - Service: Other
 - Statement ID: my-custom-permission-amazonlexchatbot
 - Principal: lexv2.amazonaws.com
 - Source name: find the ARN from the error message (eg. arn:aws:lex:ap-southeast-2:640168412593:bot-alias/*).
 - Action: lambda:InvokeFunction

Invalid Bot Configuration: Access denied while invoking lambda function arn:aws:lambda:ap-southeast-2:640168412593:function:BankingBotEnglish from arn:aws:lex:ap-southeast-2:640168412593:bot-alias/I7HNFHA7Q0/TSTALIASID. Please check the policy on this function.

[Lambda](#) > [Functions](#) > [testfunctionDELETEThis](#) > Add permissions

Add permissions

Edit policy statement

☐ AWS account

Grant permissions to another AWS account, user, or role.

☒ AWS service

Grant permissions to another AWS service.

☐ Function URL

Grant permissions to invoke your function through the function URL.

Service

The AWS service to grant permissions to.

Other ▼

Statement ID

Enter a unique statement ID to differentiate this statement within the policy.

my-custom-permission-amazonlexchatbot

Principal

The service principal for this AWS service. [Learn more](#) 

lexv2.amazonaws.com

Source ARN

The ARN for a resource. Find the ARN in the related service console.

arn:aws:lex:us-west-2:471112976395:bot-alias/*

Action

Choose an action to allow.

lambda:InvokeFunction ▼

Cancel

Save

- Select Save.
- Test whether your **CheckBalance** intent works