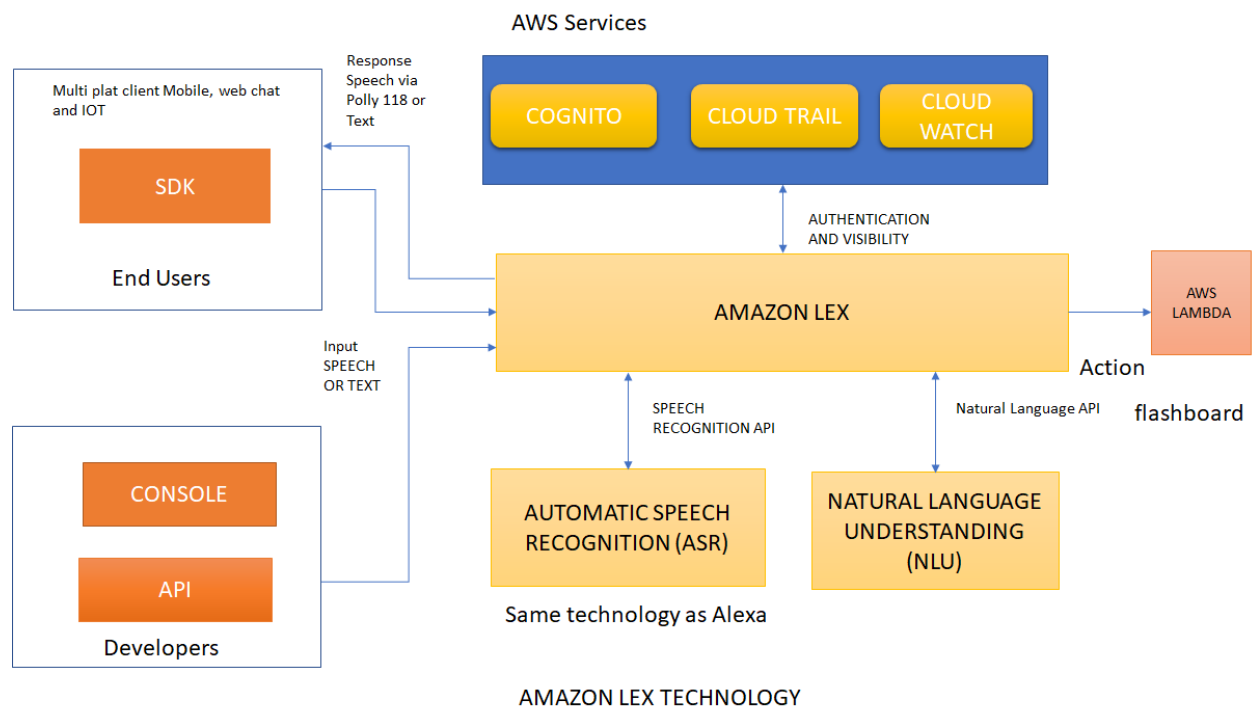


Amazon Lex

Amazon Lex helps us to build conversational chatbot interfaces through voice and text. It provides functionalities of Natural Language Understanding (NLU) and Automatic Speech Recognition (ASR). This service can be embedded into new and existing applications.

Working of Lex

Amazon Lex uses the Alexa technology of natural language understanding and speech recognition. We can build a chatbot using a console with the user intents. Lambda functions can be triggered from Lex as a fulfillment or dialog codehook. Lex responds back to the user via speech through AWS Polly or text. Console helps in bot creation, building and publishing/deploying and testing of bot.



source: <https://medium.com/xclipse/amazon-lex-a-detailed-analysis-on-what-it-is-and-its-features-45e513ac12d6>

Terminologies:

Bot: Bots are the future engagement tools for everyday life. A bot performs automated tasks like ordering a pizza, booking a taxi, ordering flowers, etc. Each bot must have a unique name within a single account.

A custom bot can be created by giving conversation flow in the Amazon lex console. For reference, we shall use a bot named “*Test* “, which navigates a robot using voice commands.

Intent: An intent is the action that the user wants to perform. We can create a bot with multiple intents.

For example, this bot has multiple intents with the name Movebot, Stopbot and Previous.

Every Intent is defined and described through:

Intent name– A descriptive call for the intent. Intent names must be unique within a single account.

Sample utterances – The number of ways in which a user might convey the action (intent). For example, Movebot has the following sample utterance: "I would like to move the robot “or “move the robot to Hall".

Slot: An intent can have zero or more slots. Slots are like function parameters and in this context used for intent configuration. At runtime, Amazon Lex prompts the user for these slot values if appropriate values are not provided in the utterance. Lex can fulfill the intent only when the user provides valid values for all the slots.

Slots are added while configuring the intent. For every slot, the type of slot and prompts are provided so that if slot value is not matched it prompts the user to elicit the slot values.

We can create our own custom slot types or use built-in slot types.

Custom slot creation:

Add slot type

Slot type name

Roomtype

Description

Type of rooms available

Slot Resolution

☒ Expand Values ⓘ

☐ Restrict to Slot values and Synonyms ⓘ

Value ⓘ

e.g. Small

+

Hall

✕

For example, AMAZON.NUMBER is a built-in slot type which identifies numbers in the form of words, AMAZON.Country is another built-in slot type that represents a Country name.

Go through the following link to understand more about built-in slots:

<https://docs.aws.amazon.com/lex/latest/dg/howitworks-builtins.html>

Ways to fulfill an intent:

We can fulfill the intent by creating:

i) **Lambda function** - This function can perform validations and fulfillment tasks. This option is ideal for scenarios where an external content portal/API needs to be invoked to fetch data relevant to the given utterance.

ii) **Intent is configured** such that Amazon Lex simply returns the information (relating to specific intent matched by an utterance and slot values) back to the client application to do the necessary fulfillment.

Lambda functions are serverless services which are part of AWS. They can be used for validations and fulfillments of an intent.

As part of initialization the validation code hook (the Lambda function) checks basic validation. For example, if the user wants to move the robot to a place which is not present in the slot type, Lambda function directs Lex to prompt the user for a valid destination that matches one of the pre-defined slot values.

To know more on fulfillment, please go through https://docs.aws.amazon.com/lex/latest/dg/API_FulfillmentActivity.html

Get started with some hands-on with Lex:

The following example shows constructing a chatbot using AWS Console.

Steps for Creating a Custom Bot

STEP 1

By selecting a Custom Bot in Amazon Lex console we can build our own chatbot from scratch. Bot can be configured as shown below. Here is the link [https://us-west-2.console.aws.amazon.com/lex/home?region=us-west-2#bot-create:](https://us-west-2.console.aws.amazon.com/lex/home?region=us-west-2#bot-create;).

Amazon Lex enables any developer to build conversational chatbots quickly and easily. With Amazon Lex, no deep learning expertise is necessary—you just specify the basic conversational flow directly from the console, and then Amazon Lex manages the dialogue and dynamically adjusts the response. To get started, you can choose one of the sample bots provided below or build a new custom bot from scratch.

CREATE YOUR OWN

TRY A SAMPLE

Custom bot

BookTrip

OrderFlowers

ScheduleAppointment

Bot name

NavigationRobot

Language

English (US)

Output voice

Salli

STEP 2

In the Editor, Create intents and add Sample Utterances.

For example, [MoveBot](#) is the name of the intent. It is configured with one custom-slot with a type as RoomType. RoomType can have values such as Kitchen, Living room, Dining and Sitting. It is also configured with a prompt to elicit the slot. Following Utterances are configured with

MoveBot 9 ▼ Edit

▼ Sample utterances ⓘ

find my keys in {place}

move to {place}

hello

hi

bring me coffee from {place}

move the robot

go to {place}

intent.

STEP 3

The Bot can be configured in a way that validation and fulfillment is performed by Lambda function or return responses and details to the client application for fulfillment.

In the editor, select Lambda function as Dialog code hook and Fulfillment code hook as shown below. **MyFirstLambda** is the name of Lambda Function which is already defined in a way to receive input from lex and respond back to lex.

```
function elicitSlot(sessionAttributes, intentName, slots, slotToElicit, message) {
  return {
    sessionAttributes,
    dialogAction: {
      type: 'ElicitSlot',
      intentName,
      slots,
      slotToElicit,
      message,
    },
  };
}
```

```
function close(sessionAttributes, fulfillmentState, message) {
  return {
    sessionAttributes,
    dialogAction: {
      type: 'Close',
      fulfillmentState,
      message,
    },
  };
}
```

```
// ----- Helper Functions -----
```

```
function buildValidationResult(isValid, violatedSlot, messageContent) {
  if (messageContent == null) {
    return {
      isValid,
      violatedSlot,
    };
  }
  return {
    isValid,
    violatedSlot,
    message: { contentType: 'PlainText', content: messageContent },
  };
}
```

```
function validateMoveBot(place) {
  var placeType = ['kitchen', 'hall', 'room', 'dining', 'sitting'];
```

```

    if (place === null){
        return buildValidationResult(false, 'place', 'Hi.How can I help you ?You can move the robot
to KITCHEN,HALL,ROOM,DINING,SITTING');

        //To provide the builtin prompt.
        //return buildValidationResult(false, 'place', null);

    }

    if (place && placeType.indexOf(place) === -1 ) {
        return buildValidationResult(false, 'place', `We do not have ${place}, would you like
to move to any one of the rooms KITCHEN,HALL,BROOM,DINING,SITTING`);
    }

    return buildValidationResult(true, null, null);
}

// ----- Functions that control the bot's behavior -----

/**
 * Performs dialog management and fulfillment
 *
 * Beyond fulfillment, the implementation of this intent demonstrates the use of the elicitSlot
dialog action
 * in slot validation and re-prompting.
 */
function MoveBot(intentRequest, callback) {
    const place = intentRequest.currentIntent.slots.place;
    const source = intentRequest.invocationSource;

    if (source === 'DialogCodeHook') {
        // Perform basic validation on the supplied input slots. Use the elicitSlot dialog action to
re-prompt for the first violation detected.
        const slots = intentRequest.currentIntent.slots;
        const validationResult = validateMoveBot(place);
        if (!validationResult.isValid) {
            slots[`${validationResult.violatedSlot}`] = null;
            callback(elicitSlot(intentRequest.sessionAttributes, intentRequest.currentIntent.name,
slots, validationResult.violatedSlot, validationResult.message));
            return;
        }
    }
}

```

```

    if(place === 'kitchen'){
      callback(close(intentRequest.sessionAttributes, 'Fulfilled',
        { contentType: 'PlainText', content: `moving to ${place} ,value:'1' `}));
    }

    if(place === 'hall'){
      callback(close(intentRequest.sessionAttributes, 'Fulfilled',
        { contentType: 'PlainText', content: `moving to ${place} ,value:'2' `}));
    }

    if(place === 'room'){
      callback(close(intentRequest.sessionAttributes, 'Fulfilled',
        { contentType: 'PlainText', content: `moving to ${place} ,value:'3' `}));
    }

    if(place === 'dining'){
      callback(close(intentRequest.sessionAttributes, 'Fulfilled',
        { contentType: 'PlainText', content: `moving to ${place} ,value:'4' `}));
    }

    if(place === 'sitting'){
      callback(close(intentRequest.sessionAttributes, 'Fulfilled',
        { contentType: 'PlainText', content: `moving to ${place} ,value:'5' `}));
    }
  }
  // ----- Intents -----
  /**
   * Called when the user specifies an intent for this skill.
   */
  function dispatch(intentRequest, callback) {
    console.log(`dispatch                                userId=${intentRequest.userId},
intentName=${intentRequest.currentIntent.name}`);
    const intentName = intentRequest.currentIntent.name;

    // Dispatch to your skill's intent handlers
    if (intentName === 'MoveBot') {
      return MoveBot(intentRequest, callback);
    }
    throw new Error(`Intent with name ${intentName} not supported`);
  }

  // ----- Main handler -----

  // Route the incoming request based on intent.
  // The JSON body of the request is provided in the event slot.
  exports.handler = (event, context, callback) => {
    try {

```

```

console.log(`event.bot.name=${event.bot.name}`);

if (event.bot.name !== 'Test') {
  callback('Invalid Bot Name');
}

dispatch(event, (response) => callback(null, response));
} catch (err) {
  callback(err);
}
};

```

▼ Fulfillment ⓘ

☒ AWS Lambda function
 ☐ Return parameters to client

Lambda function

MyFirstLambda ▼

[View in Lambda console](#) 



Version or alias

Latest ▼

The other way for fulfillment is returning parameters to the client. We can add messages or Custom Markup to close the intent.

▼ Response ⓘ

 [Preview](#)

||
☒ Message
☐ Custom Markup



One of these messages will be presented at random.

e.g. Thank you. Your {Drink_Name} has been ordered.

+

Stopping the bot

×

+


 Add Message


STEP 4

Lex bot can be configured with Clarification prompts and messages. Lex uses these prompts at runtime when it could not understand user intent.

Error handling

☒ Clarification prompts


e.g. Sorry, can you please repeat that? 

Sorry, can you please repeat that? 

Maximum number of retries

2

Hang-up phrase

e.g. Sorry, I could not understand. Please contact custome 

Sorry, I could not understand. Goodbye. 

STEP 5

Save the intent, Build and Publish the bot by giving an Alias name as “[TestBot](#)”. After the full build is completed, we are ready to test the bot by giving text or audio input.

> Test bot (Latest)
Ready. Build complete.

hi

Hi.How can I help you ?You can move the robot to KITCHEN,HALL,ROOM,DINING,SITTING

move to hall

moving to hall ,value:'2'

move to balcony

We do not have balcony, would you like to move to any one of the rooms KITCHEN,HALL,BROOM,DINING,SITTING

Clear chat history

Chat with your bot...

> Test bot (Latest)
Ready. Build complete.

aw fd hw c th

Sorry, can you please repeat that?

Audio input

Sorry, can you please repeat that?

Audio input

Sorry, I could not understand. Goodbye.

How can we share our chatbot:

A bot, intent or slot type can be imported or exported. This is helpful in cases where we want to share/reuse our bot with users with a different AWS account. We can export it and then share. Please follow instructions in this link:
<https://docs.aws.amazon.com/lex/latest/dg/export-to-lex.html>

We can export bots in either Amazon Lex or an Alexa skill format but can import only in Amazon Lex format.

Bot can be exported in a format compatible with Alexa skill and can be imported using Alexa Skills Kit that makes our bot available with Alexa.

While exporting a bot, its resources are written in a JSON format file. While exporting we can make use of the Amazon Lex console.

References

[Amazon lex documentation](#)
[AWS lambda documentation](#)