aws

開発者ガイド

Amazon Lex V1

Amazon Lex V1: 開発者ガイド

Table of Contents

Amazon Lex V2 を使用している場合は、代わりに Amazon Lex V2 ガイドを参照してください。

Amazon Lex V1 を使用している場合は、ボットを Amazon Lex V2 にアップグレードすることをお勧めします。V1 には新機能を追加されませんので、すべての新しいボットには V2 を使用することを強くお勧めします。

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。

Amazon Lex とは

Amazon Lex は、音声やテキストを使用した会話型インターフェイスをさまざまなアプリケーションに構築するための AWS のサービスです。Amazon Lex では、Amazon Alexa と同じ会話型エンジンが使用可能になり、デベロッパーは新規および既存のアプリケーションに高度な自然言語の chatbot を構築できるようになりました。Amazon Lex では、機能性と柔軟性が高い自然言語理解 (NLU) と自動音声認識 (ASR) が使用できるため、リアルな会話のやり取りができる、ユーザーにとって使いやすく魅力的なアプリケーションを構築し、新しいカテゴリの製品を生み出すことができます。

Amazon Lex を使用することで、デベロッパーは会話型の chatbot をすばやく構築できます。Amazon Lex では深層学習の専門知識は必要ありません。ボットを作成するには、Amazon Lex コンソールで基本的な会話フローを指定するだけです。Amazon Lex は会話を管理し、会話のレスポンスを動的に調整します。コンソールを使用して、テキストまたは音声の chatbot を構築、テスト、公開できます。次に、モバイルデバイス、ウェブアプリケーション、チャットプラットフォーム (Facebook Messenger など) で、会話型インターフェイスをボットに追加できます。

Amazon Lex では、AWS Lambda との統合が構築済みであり、AWS プラットフォームの Amazon Cognito、AWS Mobile Hub、Amazon CloudWatch、Amazon DynamoDB などの他の多くのサービスとも簡単に連携できます。Lambda との統合により、構築済みのサーバーレスのエンタープライズコネクターにボットからアクセスし、Salesforce、HubSpot、Marketo などの SaaS アプリケーションのデータにリンクできます。

Amazon Lex を使用する利点のいくつかを以下に示します。

- シンプル – Amazon Lex では、コンソールを使用して独自の chatbot を数分で作成できます。いくつかのフレーズの例を提供するだけで、Amazon Lex によって自然言語のモデルが完成されます。このモデルを通じて、ボットは音声やテキストを使用したやり取りで質問を行い、回答を取得し、高度なタスクを実行できます。

- 一般化された深層学習のテクノロジー – Alexa と同じテクノロジーを使用する Amazon Lex には、音声言語認識 (SLU) システムを作成するための ASR および NLU テクノロジーが搭載されています。SLU によって、Amazon Lex は自然言語の音声やテキストの入力を受け取り、入力の背後のインテントを理解し、適切なビジネス関数を呼び出してユーザーのインテントを達成します。

音声認識と自然言語理解は、コンピュータサイエンスで解決すべき最も難しい問題の一部であり、膨大な量のデータとインフラストラクチャでトレーニングされた高度な深層学習アルゴリズムが必要です。Amazon Lex によって、すべての開発者の手の届くところに Alexa のテクノロジーを提供することで、深層学習テクノロジーを誰もが使えるようになります。Amazon Lex の chatbot では、音声をテキストに変換し、ユーザーのインテントを理解してインテリジェントなレスポンスを生成します。開発者は、顧客に合わせて付加価値を差別化したボットを構築することに集中し、会話型インターフェイスを使って可能になるまったく新しいカテゴリの製品を定義できます。

- シームレスなデプロイとスケーリング – Amazon Lex を使用することで、Amazon Lex コンソールから直接 chatbot を構築、テスト、デプロイすることができます。Amazon Lex では、モバイルデバイス、ウェブアプリ、チャットサービス (Facebook Messenger など) で使用する音声やテキストの chatbot を簡単にパブリッシュできます。Amazon Lex は自動的にスケールされるため、ボット体験を実現するためにハードウェアのプロビジョニングやインフラストラクチャの管理について心配する必要はありません。

- AWS プラットフォームとの組み込み統合 – Amazon Lex には、Amazon Cognito、AWS Lambda、Amazon CloudWatch、AWS Mobile Hub など、他の AWS のサービスと相互に連携する機能が組み込まれています。AWS プラットフォームの能力を活用して、セキュリティ、モニタリング、ユーザー認証、ビジネスロジック、ストレージ、モバイルアプリケーションを開発できます。

- 高いコスト効率 – Amazon Lex には初期費用や最低料金はありません。実際に行ったテキストや音声のリクエストに対してのみ料金がかかります。従量制料金とリクエストごとの低コストは、会話型インターフェイスを構築するためのコスト効率の高い方法です。Amazon Lex の無料利用枠を利用すると、一切の初期投資なしで Amazon Lex を簡単に試すことができます。

Amazon Lex を初めてお使いになるユーザー向けの情報

Amazon Lex を初めて使用するユーザーには、次のセクションを順に読むことをお勧めします。

1. Amazon Lex の開始方法 – このセクションでは、アカウントをセットアップして Amazon Lex をテストします。
2. API リファレンス – このセクションでは、Amazon Lex の学習に役立つその他の例を示します。

Amazon Lex: 仕組み

Amazon Lex を使用すると、Amazon Alexa に採用されているのと同じテクノロジーを利用し、音声またはテキストのインターフェイスを使用するアプリケーションを構築できるようになります。Amazon Lex を使用する際に実行する一般的な手順を以下に示します。

1. ボットを作成し、サポート対象となる 1 つ以上のインテントを使用してそれを設定します。ボットがユーザーの目的 (インテント) を理解すること、そしてユーザーとの会話から情報を引き出し、ユーザーのインテントを達成できるように設定します。
2. ボットをテストします。Amazon Lex コンソールで提供されているテストウィンドウクライアントを使用できます。
3. バージョンを発行してエイリアスを作成します。
4. ボットをデプロイします。ボットは、モバイルアプリケーションなどのプラットフォームまたはメッセージングプラットフォーム (Facebook Messenger など) にデプロイできます。

開始する前に、以下の Amazon Lex の主要概念と用語を理解してください。

- ボット – ボットは、ピザの注文、ホテルの予約、花の注文などの自動化されたタスクを実行します。Amazon Lex のボットでは、自動音声認識 (ASR) 機能と自然言語理解 (NLU) 機能を使用しています。各ボットは、アカウント内で一意の名前を持つ必要があります。

  Amazon Lex ボットは、テキストまたは音声のユーザー入力を理解し、自然言語で会話できます。Lambda 関数を作成してコードフックとしてインテント設定に追加することで、ユーザーデータの検証とフルフィルメントタスクを実行できます。

- インテント – インテントは、ユーザーが実行したいアクションを表します。1 つ以上の関連するインテントをサポートするには、ボットを作成します。例えば、ピザと飲み物を注文するボットを作成できます。各インテントでは、以下の必要な情報を指定します。

  - インテント名 – インテントのわかりやすい名前。例えば、**`OrderPizza`** です。インテント名はアカウント内で一意でなければなりません。

- サンプル発話 – ユーザーがインテントを伝える方法。例: ユーザーが「ピザの注文をお願いします」や「ピザを注文します」と言った場合。
- インテントを達成する方法 – 必要な情報をユーザーが指定した後に、そのインテントを達成する方法 (例: 最寄りのピザ店に注文する)。インテントを達成する方法として Lambda 関数を作成することが推奨されます。

  オプションとして、Amazon Lex からクライアントアプリケーションに単に情報を返して目的を達成するように、インテントを設定することもできます。

Amazon Lex はまた、ピザの注文などのカスタムインテントを使用できるだけでなく、組み込みインテントを使用してボットを迅速にセットアップすることもできます。詳細については、「組み込みのインテントとスロットタイプ」を参照してください。

- スロット – インテントでは 0 個以上のスロット (パラメータ) を使用します。インテント設定の一部としてスロットを追加します。実行時に、Amazon Lex は特定のスロット値を指定するようにユーザーに求めます。Amazon Lex がインテントを達成するには、ユーザーがすべての必須スロットの値を指定する必要があります。

  例えば、`OrderPizza` インテントではピザのサイズ、クラストタイプ、ピザの枚数などが必須スロットです。これらのスロットはインテント設定で追加します。スロットごとに、スロットタイプとプロンプトを指定します。プロンプトは、ユーザーからデータを引き出すために Amazon Lex からクライアントに送信されます。ユーザーは「ラージサイズのピザにしてください」や「スモールサイズにします」などの追加の言葉を含めたスロット値で応答できます。Amazon Lex はまだ意図したスロット値を理解できます。

- スロットタイプ – 各スロットにはタイプがあります。カスタムスロットタイプを作成するか、組み込みスロットタイプを使用できます。各スロットタイプの名前は、アカウント内で一意でなければなりません。例えば、`OrderPizza` インテントでは以下のスロットタイプを作成して使用できます。

- Size – 列挙値は `Small`、`Medium`、`Large` です。
- Crust – 列挙値は `Thick`、`Thin` です。

Amazon Lex はまた、組み込みスロットタイプも用意されています。例えば、`AMAZON.NUMBER` はピザの注文数に使用できる組み込みスロットタイプです。詳細については、「組み込みのインテントとスロットタイプ」を参照してください。

Amazon Lex が利用可能な AWS リージョンの一覧については「Amazon Web Services General Reference」(Amazon Web Services 全般のリファレンス) の「AWS Regions and Endpoints」(AWS リージョンおよびエンドポイント) を参照してください。

ここで示している各トピックで、さらに詳しく学習できます。それらを順に確認した後に、「Amazon Lex の開始方法」に進むことをお勧めします。

トピック

Amazon Lex でサポートされている言語

Amazon Lex V1 は、さまざまな言語と口ケールをサポートしています。次の表に、サポートされている言語とサポートされる機能を示します。

Amazon Lex V2 はその他の言語もサポートしています。「Amazon Lex V2 でサポートされている言語」を参照してください。

サポートされている言語と口ケール

Amazon Lex V1 では、次の言語と口ケールがサポートされています。

コード	言語と口ケール
de-DE	ドイツ語 (ドイツ)
en-AU	英語 (オーストラリア)
en-GB	英語 (英国)
en-IN	英語 (インド)
en-US	英語 (米国)
es-419	スペイン語 (南米)
es-ES	スペイン語 (スペイン)
es-US	スペイン語 (米国)
fr-CA	フランス語 (カナダ)
fr-FR	フランス語 (フランス)
it-IT	イタリア語 (イタリア)
ja-JP	日本語 (日本)
ko-KR	韓国語 (韓国)

Amazon Lex の機能でサポートされている言語と口ケール

Amazon Lex のすべての機能は、次の表に示す以外のすべての言語と口ケールでサポートされています。

機能	サポートされている言語と口ケール
インテントコンテキストの設定	英語 (米国) (en-US)

プログラミングモデル

ボットは、Amazon Lex のプライマリリソースタイプです。Amazon Lex には、他のリソースタイプとして、インテント、スロットタイプ、エイリアス、ボットチャンネル関連付けがあります。

ボットを作成するには、Amazon Lex コンソールまたはモデル構築 API を使用します。コンソールのグラフィカルユーザーインターフェイスでは、アプリケーションの本番稼働準備が整ったボットを構築できます。必要に応じて、AWS CLI を介したモデル構築 API や独自のカスタムプログラムを使用してボットを作成することもできます。

作成したボットは、サポートされているプラットフォームのいずれかにデプロイします。または、独自のアプリケーションに統合します。ユーザーがボットとやり取りする場合、クライアントアプリケーションは Amazon Lex ランタイム API を使用してボットにリクエストを送信します。例えば、ユーザーが「ピザを注文したい」と言うと、クライアントはランタイム API オペレーションのいずれかを使用して Amazon Lex にこの入力を送信します。ユーザーは、入力を音声またはテキストで提供できます。

インテントでは、Lambda 関数を作成して使用することもできます。これらの Lambda 関数コードフックを使用して、初期化、ユーザー入力の検証、インテントの達成などのランタイムアクティビティを実行できます。ここで示している各セクションで、さらに詳しく学習できます。

トピック

- モデル構築 API オペレーション
- ランタイム API オペレーション
- コードフックとしての Lambda 関数

モデル構築 API オペレーション

ボット、インテント、スロットタイプをプログラムで作成するには、モデル構築 API オペレーションを使用します。ボットのリソースの管理、更新、削除にもモデル構築 API を使用できます。モデル構築 API オペレーションには以下が含まれます。

- PutBot、PutBotAlias、PutIntent、PutSlotType: それぞれ、ボット、ボットのエイリアス、インテント、スロットタイプを作成して更新します。
- CreateBotVersion、CreateIntentVersion、CreateSlotTypeVersion: それぞれ、ボット、インテント、スロットタイプのバージョンを作成して発行します。
- GetBot、GetBots: それぞれ、作成した特定のボット、複数のボットのリストを取得します。
- GetIntent、GetIntents: それぞれ、作成した特定のインテント、複数のインテントのリストを取得します。
- GetSlotType、GetSlotTypes: それぞれ、作成した特定のスロットタイプ、複数のスロットタイプのリストを取得します。
- GetBuiltinIntent、GetBuiltinIntents、GetBuiltinSlotTypes: それぞれ、ボットで使用できる特定の Amazon Lex 組み込みインテント、複数の Amazon Lex 組み込みインテントのリスト、複数の組み込みスロットタイプのリストを取得します。
- GetBotChannelAssociation、GetBotChannelAssociations: それぞれ、ボットと特定のメッセージングプラットフォームの関連付け、ボットと複数のメッセージングプラットフォームの関連付けを取得します。
- DeleteBot、DeleteBotAlias、DeleteBotChannelAssociation、DeleteIntent、DeleteSlotType: アカウントから不要なリソースを削除します。

モデル構築 API では、Amazon Lex リソースを管理するためのカスタムツールを作成できます。例えば、ボット、インテント、スロットタイプごとのバージョン数は 100 に制限されています。モデル構築 API を使用すると、ボットが制限に近づいたときに、古いバージョン自動的に削除するツールを構築できます。

リソースの更新が一度に 1 つのオペレーションでのみ実行されるように、Amazon Lex ではチェックサムを使用します。Put API オペレーション (PutBot、PutBotAlias、PutIntent、または PutSlotType) を使用してリソースを更新する場合は、リソースの最新のチェックサムをリクエストで渡す必要があります。2 つのツールで同時にリソースを更新しようとすると、両方から同じ最新のチェックサムが提供されます。最初に Amazon Lex に到達するリクエストは、リソースの最新のチェックサムと一致します。2 番目のリクエストが到着するまでには、チェックサムが異なっています。2 番目のツールは、`PreconditionFailedException` 例外を受け取り、更新は終了します。

Get オペレーション (GetBot、GetIntent、GetSlotType) には結果整合性があります。いずれかの Put オペレーションを使用してリソースを作成または変更した直後に Get オペレーションを使用すると、その更新は返されない場合があります。Get オペレーションは、前回の更新を返した後で再度リソースが変更されるまでの間に、必ず今回の更新されたリソースを返します。更新されたリソースが返されたかどうかは、チェックサムを調べて確認できます。

ランタイム API オペレーション

クライアントアプリケーションは、以下のランタイム API オペレーションを使用して Amazon Lex と通信します。

- PostContent – 音声またはテキストの入力を受け取り、インテント情報とユーザーに伝えるテキストまたは音声のメッセージを返します。現在、Amazon Lex は以下の音声形式をサポートしています。

  入力音声形式 – LPCM および Opus

  出力音声形式 – MPEG、OGG、および PCM

  PostContent オペレーションは 8 kHz と 16 kHz のオーディオ入力をサポートしています。エンドユーザーが電話で Amazon Lex と話す形式のアプリケーション (自動化されたコールセンターなど) は、8 kHz のオーディオを直接渡すことができます。

- PostText – テキストを入力として受け取り、インテント情報とユーザーに伝えるテキストメッセージを返します。

クライアントアプリケーションは、ランタイム API を使用して特定の Amazon Lex ボットを呼び出し、発話 (ユーザーによるテキストまたは音声の入力) を処理します。例えば、ユーザーが「ピザをください」と言ったとします。クライアントは、Amazon Lex のランタイム API オペレーションのいずれかを使用して、このユーザー入力をボットに送信します。Amazon Lex は、ユーザー入力から、ボットに定義されている OrderPizza インテントに関するリクエストであることを認識します。Amazon Lex は、ユーザーとの会話から必要な情報 (ピザのサイズ、トッピング、枚数など) をスロットデータとして集めます。ユーザーがすべての必要なスロットデータを指定すると、Amazon

Lex はインテント設定に応じ Lambda 関数のコードフックを呼び出してインテントを達成するか、インテントデータをクライアントに返します。

ボットで音声入力を使用している場合は、PostContent オペレーションを使用します。例えば、自動化されたコールセンターアプリケーションでは音声を Amazon Lex ボットに送信することができるので、エージェントが顧客の問い合わせに対応する必要がありません。電話から Amazon Lex に音声を直接送信するには、8 kHz オーディオ形式を使用できます。

Amazon Lex コンソールのテストウィンドウでは、PostContent API を使用してテキストおよび音声のリクエストを Amazon Lex に送信します。このテストウィンドウは「Amazon Lex の開始方法」の演習で使用します。

コードフックとしての Lambda 関数

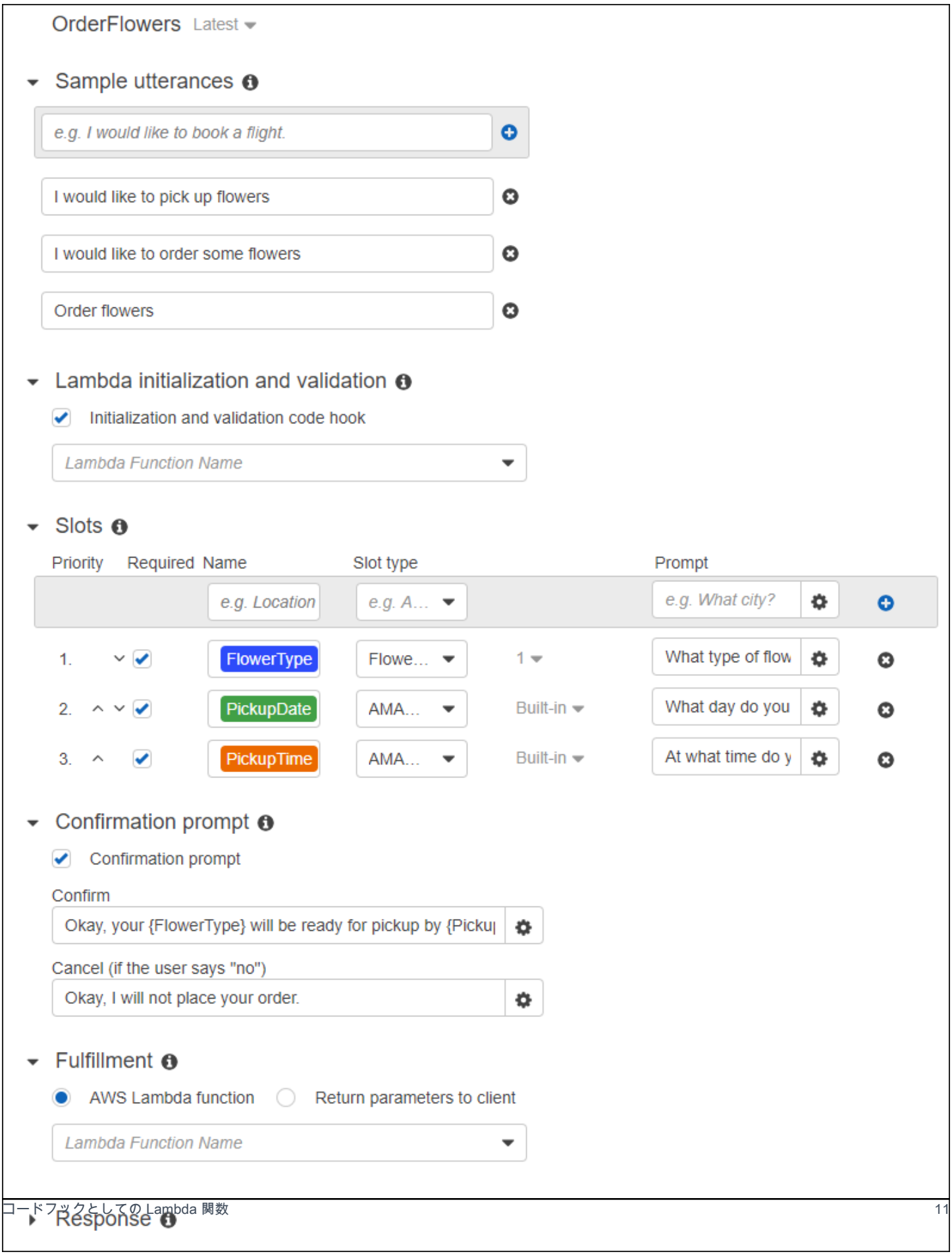
Lambda 関数をコードフックとして呼び出すように Amazon Lex ボットを設定できます。コードフックは複数の目的で使用できます。

- ユーザーとのやり取りをカスタマイズする - 例えば、Joe からピザのトッピングの種類を尋ねられたときに、Joe の好みに関する過去の知識に基づいてトッピングのサブセットを表示できます。
- ユーザーの入力を検証する - Jen が営業時間外に花の受け取りを希望しているとします。Jen が入力した時間を検証して、適切なレスポンスを送信できます。
- ユーザーのインテントを達成する - Joe からピザの注文に必要なすべての情報を得た後で、Amazon Lex は Lambda 関数を呼び出して最寄りのピザ店に注文できます。

インテントを設定する際に、以下の場所で Lambda 関数をコードフックとして指定します。

- 初期化/検証用のダイアログのコードフック – この Lambda 関数はユーザー入力ごとに呼び出されます (Amazon Lex がユーザーのインテントを理解しているものとします)。
- フルフィルメントコードフック – この Lambda 関数は、インテントを達成するために必要なすべてのスロットデータがユーザーから提供された後で呼び出されます。

次のスクリーンショットに示すように、インテントを選択して Amazon Lex コンソールでコードフックを設定します。


OrderFlowers
Latest
Sample utterances
e.g. I would like to book a flight.
I would like to pick up flowers
I would like to order some flowers
Order flowers
Lambda initialization and validation
Initialization and validation code hook
Lambda Function Name
Slots
Priority
Required
Name
Slot type
Prompt
e.g. Location
e.g. A...
e.g. What city?
1.
FlowerType
Flowe...
1
What type of flow
2.
PickupDate
AMA...
Built-in
What day do you
3.
PickupTime
AMA...
Built-in
At what time do y
Confirmation prompt
Confirmation prompt
Confirm
Okay, your {FlowerType} will be ready for pickup by {Picku
Cancel (if the user says "no")
Okay, I will not place your order.
Fulfillment
AWS Lambda function
Return parameters to client
Lambda Function Name
Response

コードフックは、PutIntent オペレーションの dialogCodeHook フィールドおよび fulfillmentActivity フィールドでも設定できます。

1 つの Lambda 関数で、初期化、検証、およびフルフィルメントを実行できます。Lambda 関数が受信するイベントデータには、呼び出し元をダイアログまたはフルフィルメントのコードフックとして識別するフィールドがあります。この情報を使用して、コードの該当部分を実行できます。

Lambda 関数では、複雑なダイアログ間をナビゲートできるボットを構築できます。Amazon Lex に特定のアクションを実行するように指示するには、Lambda 関数のレスポンスの dialogAction フィールドを使用します。例えば、必須ではないスロット値をユーザーから取得するよう Amazon Lex に指示するには、ElicitSlot ダイアログアクションを使用します。明確化プロンプトが定義されている場合、ElicitIntent ダイアログアクションを使用して、ユーザーが前のインテントを終了したときに、別のインテントを引き出すことができます。

詳細については、「Lambda 関数を使用する」を参照してください。

メッセージの管理

トピック

- メッセージのタイプ
- メッセージの設定のコンテキスト
- サポートされているメッセージ形式
- メッセージグループ
- レスポンスカード

ボットを作成する際に、クライアントに送信したい明確メッセージや情報メッセージを設定できます。次の例を考えます。

- 次の明確化プロンプトをボットに設定できます。

  ```
  I don't understand. What would you like to do?
  ```

  Amazon Lex は、ユーザーのインテントを理解できない場合に、このメッセージをクライアントに送信します。

- OrderPizza というインテントをサポートするボットを作成するとします。ピザの注文では、ピザのサイズ、トッピング、クラストタイプなどの情報を指定するようにユーザーに求めます。以下のプロンプトを設定することもできます。

  ```
  What size pizza do you want?
  What toppings do you want?
  Do you want thick or thin crust?
  ```

  Amazon Lex はユーザーのインテントがピザの注文であると判断すると、これらのメッセージをクライアントに送信してユーザーから情報を取得します。

このセクションでは、ボットの設定でユーザーのやり取りをデザインする方法について説明します。

メッセージのタイプ

メッセージはプロンプトまたはステートメントにすることができます。

- 通常、プロンプトは質問でユーザーからのレスポンスを想定します。
- ステートメントは情報を表示します。これはレスポンスを想定していません。

メッセージには、スロット、セッション属性、リクエスト属性への参照を含めることができます。実行時に、Amazon Lex は、これらの参照を実際の値で置き換えます。

設定されたスロット値を参照するには、次の構文を使用します。

```
{SlotName}
```

セッション属性を参照するには、次の構文を使用します。

```
[SessionAttributeName]
```

リクエスト属性を参照するには、以下の構文を使用します。

```
((RequestAttributeName))
```

メッセージには、スロット値、セッション属性、リクエスト属性を含めることができます。

例えば、次のメッセージをボットの OrderPizza インテントに設定したとします。

```
"Hey [FirstName], your {PizzaTopping} pizza will arrive in [DeliveryTime] minutes."
```

このメッセージでは、スロット (PizzaTopping) とセッション属性 (FirstName と DeliveryTime) の両方が参照されています。実行時に、Amazon Lex はこれらのプレースホルダを値に置き換えて、次のメッセージをクライアントに返します。

```
"Hey John, your cheese pizza will arrive in 30 minutes."
```

メッセージに角括弧 ([]) または中括弧 ({}) を入れるには、バックスラッシュ (\) のエスケープ文字を使用します。例えば、次のメッセージには中括弧と角括弧が含まれています。

```
\{Text\} \[Text\]
```

クライアントアプリケーションに返されたテキストは次のようになります。

```
{Text} [Text]
```

セッション属性の詳細については、ランタイム API オペレーションの「PostText」および「PostContent」を参照してください。例については、「旅行を予約する」を参照してください。

Lambda 関数は、メッセージを生成して Amazon Lex に返し、ユーザーに送信することもできます。インテントの設定時に Lambda 関数を追加すると、メッセージを動的に作成できます。ボットの設定時にメッセージを指定すると、Lambda 関数でプロンプトを構築する必要がなくなります。

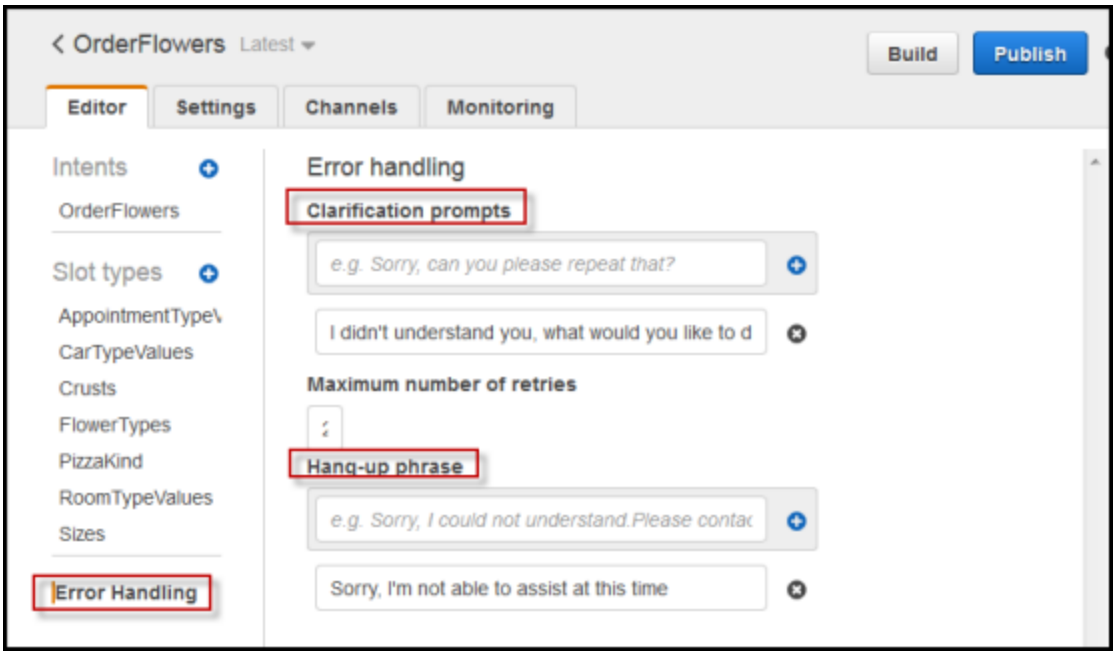
メッセージの設定のコンテキスト

ボットを作成する場合、ボットの明確化プロンプト、スロット値のプロンプト、インテントからのメッセージなど、異なるコンテキストでメッセージを作成することができます。Amazon Lex は各コンテキストで適切なメッセージを選択してユーザーに戻します。各コンテキストにメッセージのグループを指定することができます。この場合、Amazon Lex はグループから 1 つのメッセージをランダムに選択します。メッセージの形式を指定したり、メッセージをグループにまとめることもできます。詳細については、「サポートされているメッセージ形式」を参照してください。

インテントと関連付けている Lambda 関数がある場合は、構築時に設定したメッセージをどれでも上書きできます。ただし、こうしたメッセージを使用する上で Lambda 関数は必須ではありません。

ボットメッセージ

ボットに明確化プロンプトおよびセッション終了メッセージを設定できます。実行時にユーザーのインテントを理解できない場合、Amazon Lex は明確化プロンプトを使用します。Amazon Lex がセッション終了メッセージを送信する前に、説明を要求する回数を設定できます。次の画像のように、Amazon Lex コンソールの [エラー処理] セクションでボットレベルのメッセージを設定できます。



API で `clarificationPrompt` と `abortStatement` フィールドを設定してメッセージを設定します。このフィールドは PutBot オペレーションにあります。

インテントで Lambda 関数を使用する場合は、Lambda 関数がユーザーのインテントを尋ねるために Amazon Lex に向けたレスポンスを戻す場合があります。Lambda 関数がそうしたメッセージを提供しない場合は、Amazon Lex が明確化プロンプトを使用します。

スロットプロンプト

インテントの必須スロットのそれぞれに少なくとも 1 つのプロンプトメッセージを指定する必要があります。実行時に、Amazon Lex はこれらのメッセージのいずれかを使用して、スロットの値を指定することをユーザーに求めます。次は、`cityName` スロットに対する有効なプロンプトの例です。

```
Which city would you like to fly to?
```

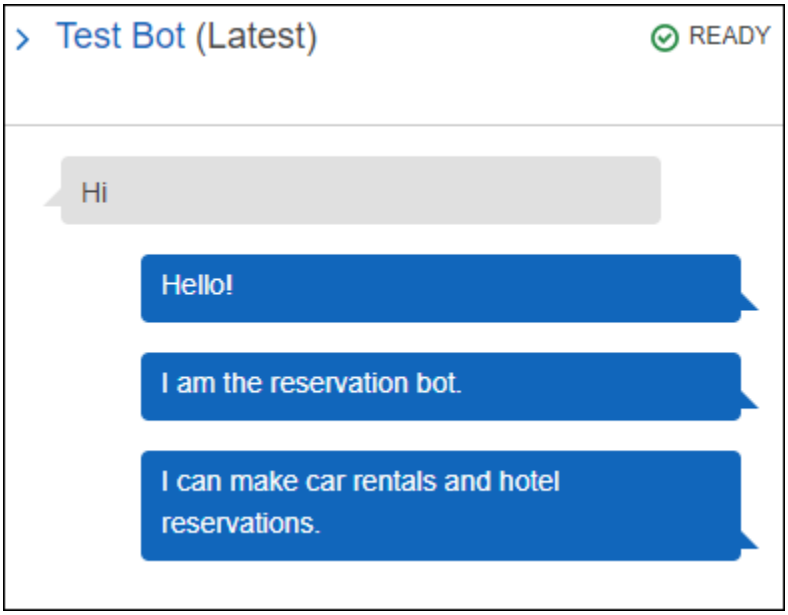
コンソールを使用して各スロットの 1 つまたは複数のプロンプトを設定できます。プロンプトのグループは、PutIntent オペレーションを使用して作成することもできます。詳細については、「メッセージグループ」を参照してください。

レスポンス

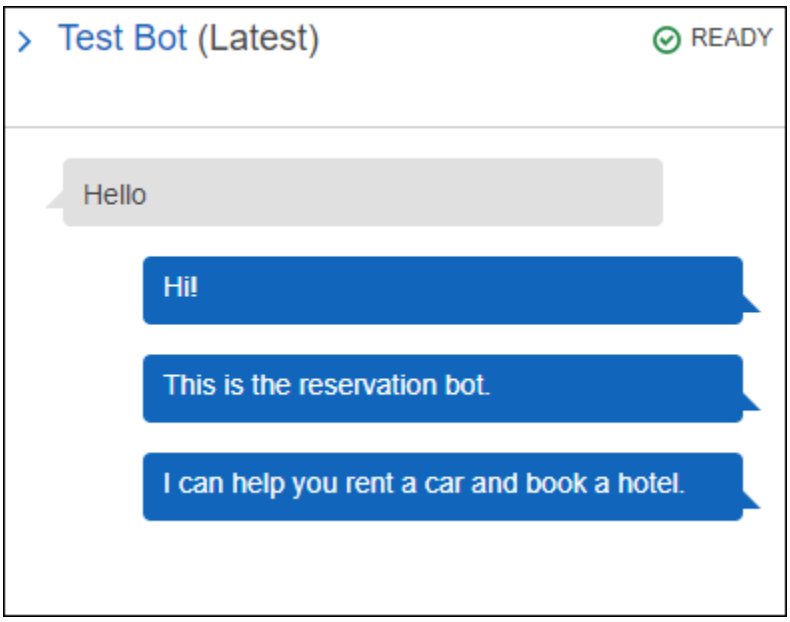
コンソールで [Responses (レスポンス)] セクションを使用し、ボットに動的で柔軟性の高い会話を構築します。レスポンスに 1 つ以上のメッセージグループを作成することができます。実行時に、Amazon Lex は各メッセージグループからメッセージを 1 つ選択することでレスポンスを構築します。メッセージグループの詳細については、「メッセージグループ」を参照してください。

例えば、最初のメッセージグループに「初めまして」、「こんにちは」、「ようこそ」といったように異なる応答メッセージを含めることができます。2 番目のメッセージグループには「私は予約ボットです」や「これは予約ボットです」といった異なる紹介メッセージを含めることもできます。3 番目のメッセージグループでは「レンタカーとホテルの予約をお手伝いします」、「レンタカーやホテルを予約することができます」、「レンタカーやホテルの予約をお手伝いできます」といったボットの通信機能を利用することができます。

Lex は各メッセージグループのメッセージを使用して、会話のレスポンスを動的に構築します。例えば、1 つのやり取りを以下に示します。

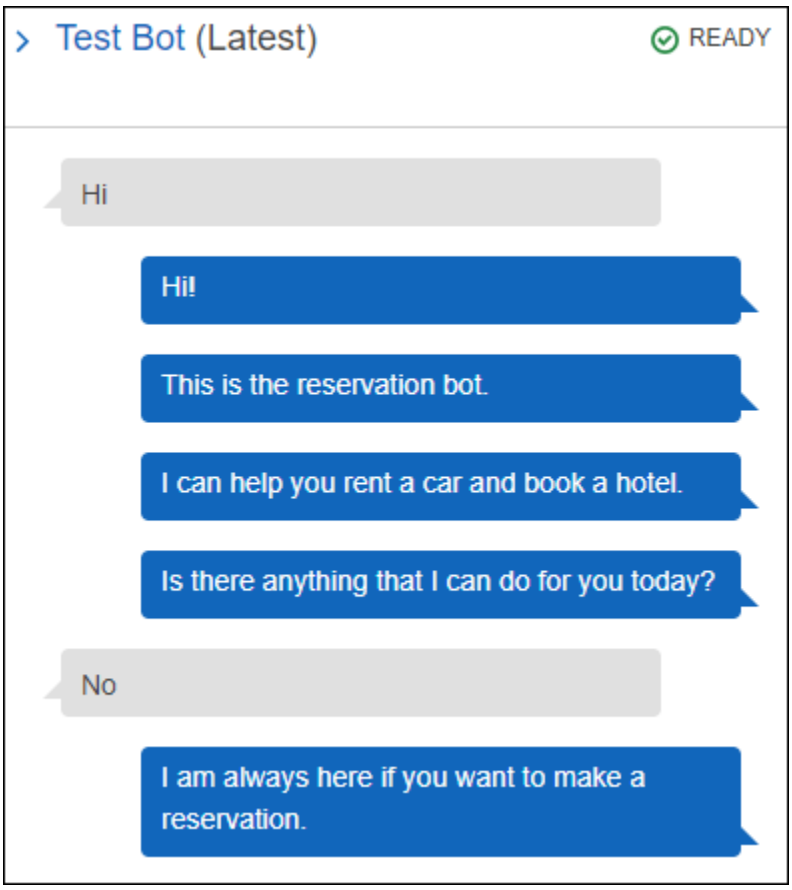


もう 1 つの例としては、次のようなものがあります。

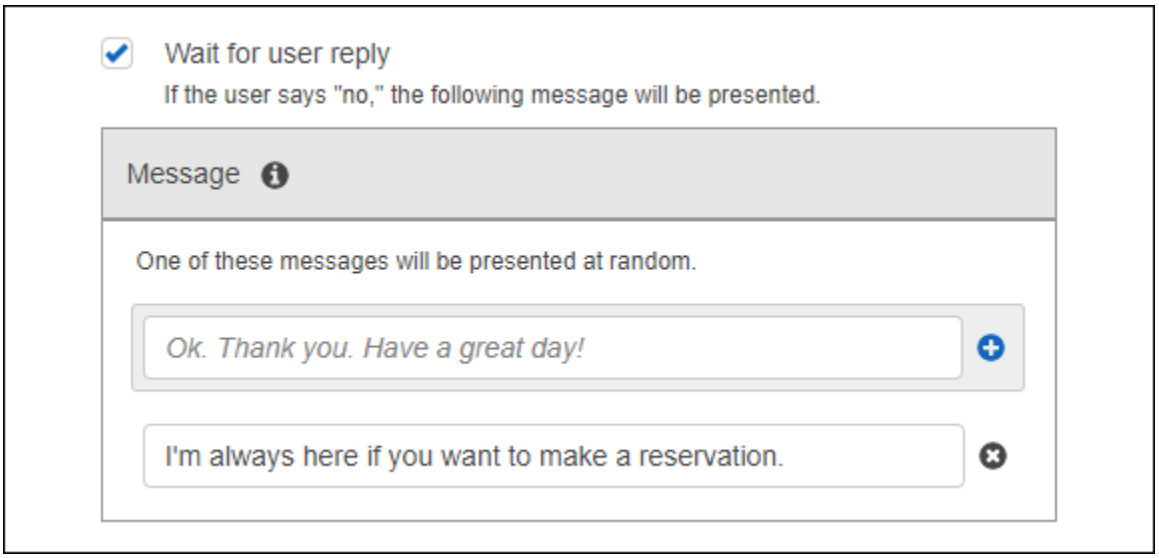


どちらの場合も、ユーザーは BookCar や BookHotel などで新しいインテントを使用して応答することができます。

レスポンスでフォローアップの質問をするようにボットをセットアップできます。例えば、前述のインタラクションでは「レンタカーやホテルの予約をお手伝いしましょうか?」、「今すぐ予約を入れますか?」、「他にお手伝いできることはありますか?」といった質問を追加して 4 番目のメッセージを作成することもできます。レスポンスとして「いいえ」が含まれるメッセージには、フォローアップのプロンプトを作成できます。次の画像に例を示します。



フォローアップのプロンプトを作成するには [Wait for user reply] を選択します。次に、メッセージを入力またはユーザーが「いいえ」と回答した場合に送信する複数のメッセージを入力します。フォローアップのプロンプトとして使用するレスポンスを作成する場合は、回答が「いいえ」だった場合に適切なステートメントを特定する必要があります。例については、次の画像を参照してください。



API を使用してインテントに応答を追加するには、PutIntent オペレーションを使用します。レスポンスを特定するには [conclusionStatement] フィールドを PutIntent リクエストで設定します。フォローアップのプロンプトを設定するには、[followUpPrompt] フィールドを設定します。これにはユーザーが「いいえ」と回答した場合に送信するステートメントを含めてください。[conclusionStatement] フィールドと [followUpPrompt] フィールドの両方を同じインテントで設定することはできません。

サポートされているメッセージ形式

PostText オペレーションを使用する場合、または PostContent オペレーションを使用する場合に Accept ヘッダーを text/plain;charset=utf8 に設定している場合、Amazon Lex は次の形式でメッセージをサポートします。

- PlainText - メッセージには UTF-8 形式テキストが含まれています。
- SSML - メッセージには音声出力のテキスト形式が含まれています。
- CustomPayload - メッセージにはクライアント向けに作成したカスタム形式が含まれています。アプリケーションのニーズを満たすようにペイロードを定義することができます。
- Composite - メッセージは各メッセージグループから 1 つずつ取り出したメッセージを集約したものです。メッセージグループの詳細については、「メッセージグループ」を参照してください。

デフォルトでは、Amazon Lex が特定のプロンプトに定義されているメッセージのうち 1 つを返します。例えば、スロット値を引き出すために 5 つのメッセージを定義した場合、Amazon Lex はランダムにいずれかのメッセージを選びクライアントに返します。

Amazon Lex がランタイムリクエストでクライアントに指定のメッセージタイプを返すには、x-amzn-lex:accept-content-types リクエストパラメータを設定します。レスポンスはリクエストされたタイプに制限されます。指定したタイプのメッセージが複数ある場合は、Amazon Lex がランダムに 1 つ返します。x-amz-lex:accept-content-types ヘッダーの詳細については、「レスポンスタイプの設定」を参照してください。

メッセージグループ

メッセージグループ は特定のプロンプトに対する一連の適切なレスポンスです。ボットに会話のレスポンスを動的に構築させるにはメッセージグループを使用します。Amazon Lex がクライアントアプリケーションにレスポンスを返すと、各グループからメッセージを 1 つランダムに選択します。各レスポンスで最大 5 件のメッセージを作成できます。各グループには最大 5 件のメッセージを含

めることができます。コンソールでメッセージグループを作成する例については「レスポンス」を参照してください。

メッセージグループを作成するには、コンソールを使用するか、PutBot、PutIntent、または PutSlotType オペレーションを使用して、メッセージにグループ数を割り当てることができます。メッセージグループを作成しない場合やメッセージグループを 1 つだけ作成した場合、Amazon Lex は Message フィールドでメッセージを 1 件送信します。コンソールでメッセージグループを複数作成した場合、または PutIntent オペレーションを使用してインテントを作成または更新したメッセージグループを複数作成した場合に限り、クライアントアプリケーションはレスポンスで複数のメッセージを受け取ります。

Amazon Lex がグループからメッセージを送信すると、レスポンスの Message フィールドにはメッセージを含むエスケープした JSON オブジェクトがあります。次の例は複数のメッセージを含む [Message] フィールドのコンテンツを示しています。

> **Note**
>
> このファイルは、読みやすいようにフォーマットされています。レスポンスに改行 (CR) は含まれていません。

```
{\"messages\":[
   {\"type\":\"PlainText\",\"group\":0,\"value\":\"Plain text\"},
   {\"type\":\"SSML\",\"group\":1,\"value\":\"SSML text\"},
   {\"type\":\"CustomPayload\",\"group\":2,\"value\":\"Custom payload\"}
]}
```

メッセージの形式を設定できます。形式は次のいずれかになります。

- プレーンテキスト—UTF-8 形式テキストのメッセージです。
- SSML—音声合成マークアップ言語 (SSML) のメッセージです。
- CustomPayload—指定したカスタム警視のメッセージです。

PostContent と PostText オペレーションが [Message] フィールドで返すメッセージの形式を管理するには、x-amz-lex:accept-content-types リクエストの属性を設定します。例えば次のようにヘッダーを設定すると、レスポンスでプレーンテキストと SSML メッセージのみを受信するようになります。

```
x-amz-lex:accept-content-types: PlainText,SSML
```

特定のメッセージ形式をリクエストし、メッセージグループにその形式のメッセージが含まれていない場合は NoUsableMessageException 例外を受信します。メッセージグループを使用してタイプ別にメッセージをグループ化する場合は、x-amz-lex:accept-content-types ヘッダーを使用しないでください。

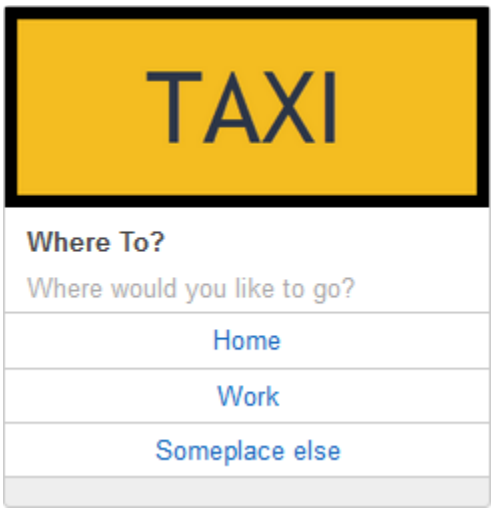
x-amz-lex:accept-content-types ヘッダーの詳細については、「レスポンスタイプの設定」を参照してください。

レスポンスカード

> **Note**
>
> レスポンスカードは Amazon Connect チャットでは機能しません。ただし、同様の機能については、「対話型メッセージをチャットに追加する」を参照してください。

レスポンスカードにはプロンプトに対する適切なレスポンスの一連が含まれています。レスポンスカードを使用すると、ユーザーとのやり取りが簡素化され、テキストのやり取りで入力ミスが減り、ボットの精度が向上します。Amazon Lex からクライアントアプリケーションに送信される各プロンプトに対して、レスポンスカードを送信できます。レスポンスカードは、Facebook Messenger、Slack、Twilio、および独自のクライアントアプリケーションで使用できます。

例えば、タクシーアプリケーションで、レスポンスカードに「自宅」のオプションを設定し、その値としてユーザーの自宅の住所を設定できます。このオプションをユーザーが選択すると、Amazon Lex では入力テキストとして住所全体を受け取ります。次の画像を参照してください。



レスポンスカードは以下のプロンプトに対して定義できます。

- 結論ステートメント
- 確認プロンプト
- フォローアッププロンプト
- 拒否ステートメント
- スロットタイプ発話

プロンプトごとに 1 つレスポンスカードのみ定義できます。

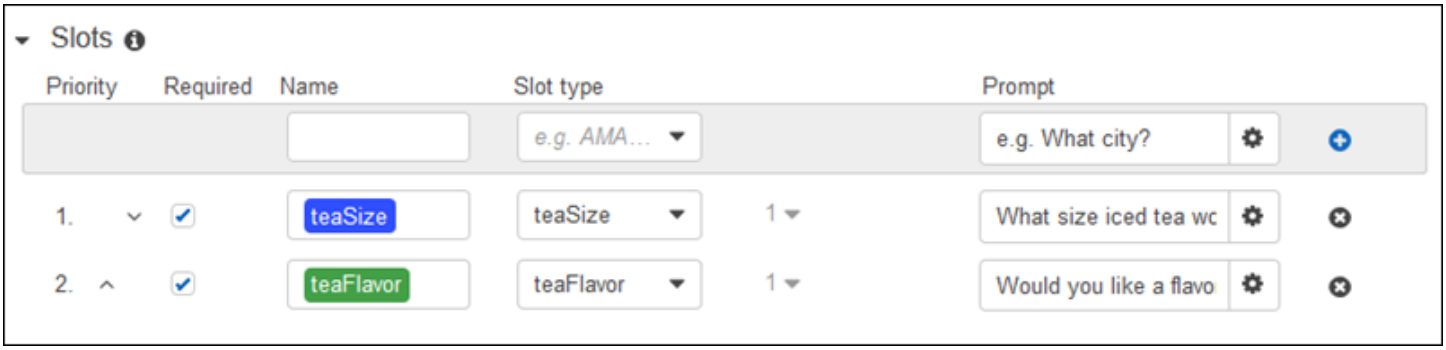
レスポンスカードは、インテントを作成するときに設定します。コンソールまたは PutIntent オペレーションを使用して静的レスポンスカードを構築時に定義できます。または、Lambda 関数でランタイムに動的レスポンスを定義できます。静的レスポンスカードと動的レスポンスカードの両方を定義すると、動的レスポンスカードが優先されます。

Amazon Lex は、クライアントが理解する形式でレスポンスカードを送信します。レスポンスカードは、Facebook Messenger、Slack、Twilio に応じて変換されます。その他のクライアントの場合、Amazon Lex は PostText レスポンスで JSON 構造を送信します。例えば、クライアントが Facebook Messenger である場合、Amazon Lex はレスポンスカードを一般テンプレートに変換します。Facebook Messenger の一般テンプレートの詳細については、Facebook ウェブサイトの「一般テンプレート」を参照してください。JSON 構造の例については、「レスポンスカードの動的な生成」を参照してください。

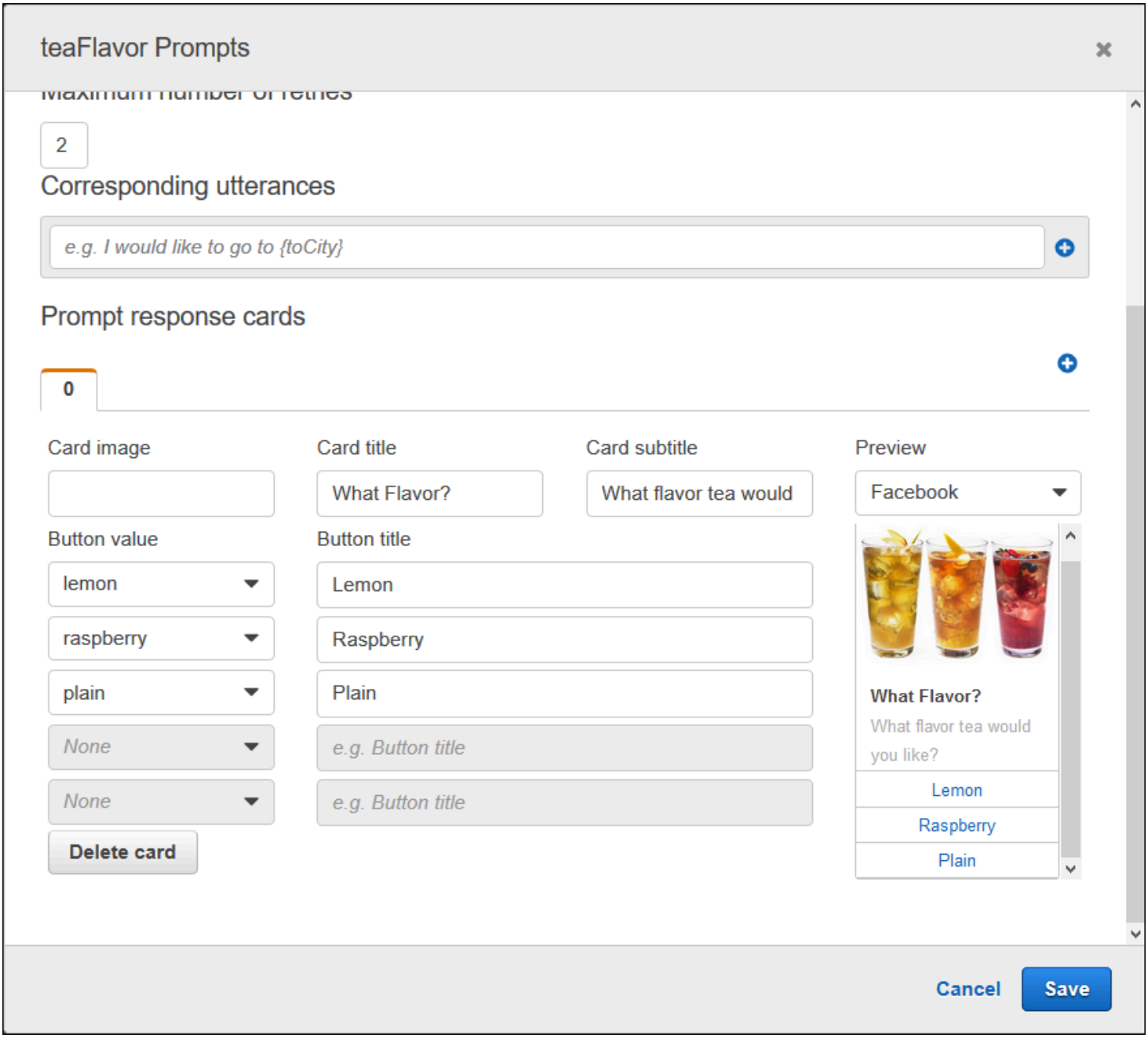
レスポンスカードは、PostText オペレーションでのみ使用できます。レスポンスカードを PostContent オペレーションで使用することはできません。

静的レスポンスカードの定義

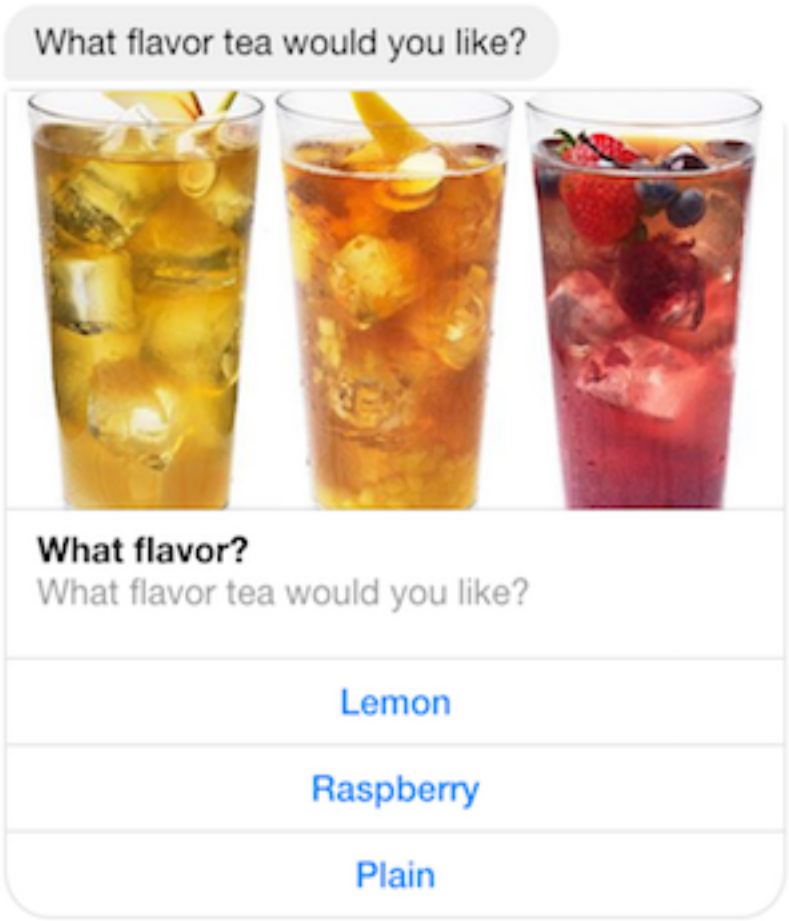
静的レスポンスカードは、インテントの作成時に PutBot オペレーションまたは Amazon Lex コンソールで定義します。静的レスポンスカードはインテントと同時に定義されます。静的レスポンスカードはレスポンスが固定されているときに使用します。例えば、風味のスロットがあるインテントのボットを作成するとします。風味のスロットを定義する場合、次のコンソールのスクリーンショットに示すようなプロンプトを指定します。



プロンプトを定義する場合は、必要に応じてレスポンスカードを関連付け、PutBot オペレーションまたは Amazon Lex コンソール (次の例を参照) で詳細を定義できます。



ここで、ボットを Facebook Messenger と統合したとします。ユーザーは、次の図に示すように、ボタンをクリックして風味を選択できます。



レスポンスカードのコンテンツをカスタマイズするには、セッション属性を参照できます。実行時に、Amazon Lex は、このリファレンスをセッション属性の該当する値に置き換えます。詳細については、「セッション属性の設定」を参照してください。例については、「レスポンスカードの使用」を参照してください。

レスポンスカードの動的な生成

レスポンスカードをランタイムに動的に生成するには、インテントで初期化および検証の Lambda 関数を使用します。動的レスポンスカードは、Lambda 関数でレスポンスがランタイムに確定される場合に使用します。Lambda 関数は、ユーザー入力に応答してレスポンスカードを生成し、それをレスポンスの `dialogAction` セクションで返します。詳細については、「レスポンスの形式」を参照してください。

Lambda 関数内の `responseCard` エレメントを示す部分は以下のとおりです。これにより、前のセクションで示したようなユーザーエクスペリエンスが生成されます。

```
responseCard: {
  "version": 1,
  "contentType": "application/vnd.amazonaws.card.generic",
  "genericAttachments": [
    {
      "title": "What Flavor?",
      "subtitle": "What flavor do you want?",
      "imageUrl": "Link to image",
      "attachmentLinkUrl": "Link to attachment",
      "buttons": [
        {
          "text": "Lemon",
          "value": "lemon"
        },
        {
          "text": "Raspberry",
          "value": "raspberry"
        },
        {
          "text": "Plain",
          "value": "plain"
        }
      ]
    }
  ]
}
```

例については、「予約のスケジュール」を参照してください。

会話コンテキストの管理

会話コンテキストは、インテントを達成するためにユーザー、アプリケーション、または Lambda 関数から Amazon Lex ボットに提供される情報です。会話コンテキストには、ユーザーが提供するスロットデータ、クライアントアプリケーションによって設定されるリクエスト属性、クライアントアプリケーションと Lambda 関数で作成するセッション属性が含まれます。

トピック

- インテントコンテキストの設定
- デフォルトのスロット値を使用する
- セッション属性の設定

- リクエスト属性の設定
- セッションタイムアウトの設定
- インテント間での情報の共有
- 複雑な属性の設定

インテントコンテキストの設定

Amazon Lex はコンテキストに基づいてインテントをトリガーすることができます。コンテキストは、ボットを定義するときにインテントに関連付けることができる状態変数です。

コンソールを使用して、または PutIntent オペレーションを使用してインテントを作成するときに、インテント用のコンテキストを設定します。英語 (US) (en-US) ロケールで、PutBot オペレーションでボットを作成した際に、`enableModelImprovements` パラメータを `true` に設定した場合のみ、コンテクストを使用することができます。

コンテキストには、出力コンテキストと出力コンテキストの 2 種類の関係があります。出力コンテキストは、関連するインテントが満たされたときにアクティブになります。出力コンテキストは、PostText オペレーション、または PostContent オペレーションの応答でアプリケーションに返され、現在のセッションに設定されます。コンテキストがアクティブになった後、コンテキストが定義されたときに設定されたターン数または時間制限の間、アクティブなままになります。

入力コンテキストは、インテントを認識するための条件を指定します。会話中にインテントを認識できるのは、その入力コンテキストがすべてアクティブになっているときだけです。入力コンテクストがないインテントは、常に認識の対象となります。

Amazon Lex は、出力コンテキストでインテントを満たすことによってアクティブになったコンテキストのライフサイクルを自動的に管理します。また、`PostContent` オペレーション、または `PostText` のオペレーションの呼び出しでアクティブなコンテキストを設定することができます。

また、インテントの Lambda 関数を使用して会話のコンテキストを設定することができます。Amazon Lex からの出力コンテキストは、Lambda 関数の入力イベントに送信されます。Lambda 関数は、そのレスポンスでコンテキストを送信することができます。詳細については、「Lambda 関数の入力イベントとレスポンスの形式」を参照してください。

例えば、レンタカーを予約するインテントがあり「book_car_fulfilled」という出力コンテキストを返すよう設定されているとします。インテントが達成すると、Amazon Lex は出力コンテキスト変数「book_car_fulfilled」を設定します。「book_car_filded」はアクティブなコンテキストであるため、

ユーザーの発話がそのインテントを引き出す試みとして認識される限り、「book_car_filded」コンテキストを入力コンテキストとして設定したインテントが認識対象として考慮されます。これは、領収書のメール送信や予約の変更など、車を予約した後にのみ意味を持つインテントに使用できます。

出力コンテキスト

Amazon Lex は、インテントが実行されたときに、インテントの出力コンテキストをアクティブにします。出力コンテキストを使用して、現在のインテントをフォローアップできるインテントを制御できます。

各コンテキストは、セッションで保持されるパラメータのリストを持っています。パラメータは、履行されたインテントのスロット値です。これらのパラメータを使用して、他のインテントのスロット値を事前に入力することができます。詳細については、「デフォルトのスロット値を使用する」を参照してください。

出力コンテキストは、コンソールまたは PutIntent オペレーションでインテントを作成するときに設定します。1 つのインテントに複数の出力コンテキストを設定することができます。インテントが実行されると、すべての出力コンテキストがアクティブになり、PostText または PostContent 応答で返されます。

次に、コンソールを使用してインテントに出力コンテキストを割り当てる方法を示します。

出力コンテキストを定義するときは、そのコンテキストの有効期限 (TTL)、つまり Amazon Lex からの応答に含まれる時間の長さまたはターン数も定義します。ある順番は、アプリケーションから Amazon Lex への 1 つのリクエストです。ターン数または時間が経過すると、コンテキストはアクティブでなくなります。

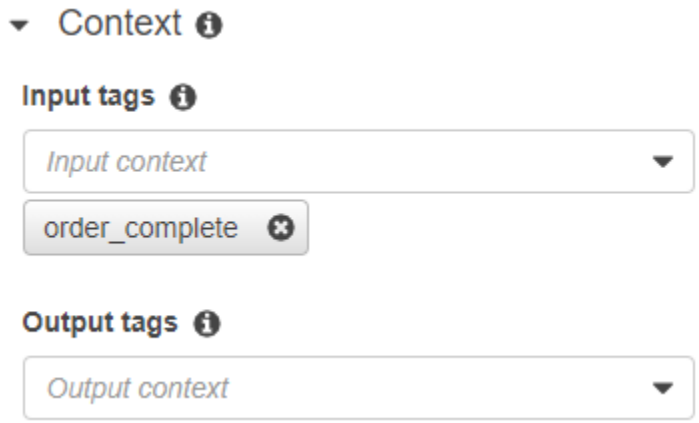
アプリケーションは必要に応じて、出力コンテキストを使用することができます。例えば、アプリケーションは出力コンテキストを次のように使用できます。

- コンテキストに基づき、アプリケーションの動作を変更します。例えば、旅行アプリケーションでは、コンテキストの「book_car_filled」に対して「rental_hotel_filded」とは異なるアクションを設定することができます。
- 出力コンテキストを、次の発話の入力コンテキストとして Amazon Lex に返します。Amazon Lex がその発話をインテントを引き出す試みと認識した場合、そのコンテキストを使用して、返されるインテントを指定されたコンテキストを持つものに限定します。

コンテキストを入力する

入力コンテキストを設定することで、会話の中でインテントが認識されるポイントを限定することができます。入力コンテキストがないインテントは、常に認識対象となります。

インテントが応答する入力コンテキストは、コンソールまたは `PutIntent` オペレーションを使用して設定します。インテントは複数の入力コンテキストを持つことができます。次に、コンソールを使用してインテントに入力コンテキストを割り当てる方法を示します。



複数の入力コンテキストを持つインテントでは、インテントをトリガーするには、すべてのコンテキストがアクティブである必要があります。PostText オペレーション、PostContent オペレーション、または PutSession のオペレーションを呼び出すと、入力コンテキストを設定することができます。

インテント内のスロットは、現在アクティブなコンテキストからデフォルト値を取るように設定することができます。デフォルト値は、Amazon Lex が新しいインテントを認識するが、スロット値を受信しない場合に使用されます。スロットを定義する際に、コンテキスト名とスロット名を `#context-name.parameter-name` という形で指定します。詳細については、「デフォルトのスロット値を使用する」を参照してください。

デフォルトのスロット値を使用する

デフォルト値を使用する場合、ユーザーの入力によってスロットが提供されない場合に、新しいインテントで入力されるスロット値のソースを指定します。このソースは、以前のダイアログ、リクエスト、またはセッション属性、またはビルド時に設定した固定値にすることができます。

デフォルト値のソースとして、以下のものを使用することができます。

- 以前のダイアログ(コンテキスト) - #context-name.parameter-name
- セッション属性 - [attribute-name]
- リクエスト属性 - <attribute-name>
- 固定値 - 前の値と一致しない値

PutIntent オペレーションでインテントにスロットを追加する場合、デフォルト値のリストを追加することができます。デフォルト値は、記載されている順序に沿って使用されます。例えば、次のような定義のスロットを持つインテントがあるとします。

```
"slots": [
    {
        "name": "reservation-start-date",
        "defaultValueSpec": {
            "defaultValueList": [
                {
                    "defaultValue": "#book-car-fulfilled.startDate"
                },
                {
                    "defaultValue": "[reservationStartDate]"
                }
            ]
        },
        Other slot configuration settings
    }
]
```

インテントが認識されると、「reservation-start-date」という名前のスロットは、その値が以下のいずれかに設定されます。

1. 「book-car-fulfilled」コンテキストがアクティブな場合、「startDate」パラメータの値はデフォルト値として使用されます。

2. 「book-car-fulfilled」コンテキストがアクティブでない場合、または「startDate」パラメータが設定されていない場合、「reservationStartDate」セッション属性の値がデフォルト値として使用されます。
3. もし最初の 2 つのデフォルト値のどちらも使用されない場合、スロットにはデフォルト値がなく、Amazon Lex は通常通り値を引き出します。

スロットにデフォルト値が使われている場合、そのスロットが必要であっても引き出されることはありません。

セッション属性の設定

セッション属性には、セッション中にボットとクライアントアプリケーションの間でやり取りされるアプリケーション固有の情報が含まれます。Amazon Lex は、ボットに設定されたすべての Lambda 関数にセッション属性を渡します。Lambda 関数でセッション属性が追加または更新されると、Amazon Lex からクライアントアプリケーションに新しい情報が返されます。例:

- 演習 1: 設計図を使用して Amazon Lex ボットを作成する (コンソール) で、サンプルボットは `price` セッション属性を使用して花の価格を保持しています。Lambda 関数は、注文された花の種類に基づいて、この属性を設定します。詳細については、「ステップ 5 (オプション): 情報フローの詳細を確認する (コンソール)」を参照してください。
- 旅行を予約する で、サンプルボットは `currentReservation` セッション属性を使用して、会話中のスロットタイプデータのコピーを保持し、ホテルやレンタカーを予約します。詳細については、「情報フローの詳細」を参照してください。

ボットの初期化、プロンプトやレスポンスカードのカスタマイズには、Lambda 関数のセッション属性を使用します。例:

- 初期化 - ピザの注文ボットにおいて、PostContent オペレーションまたは PostText オペレーションへの最初の呼び出しで、クライアントアプリケーションはユーザーの場所をセッション属性として渡します。例えば、 `"Location": "111 Maple Street"`。Lambda 関数は、この情報に基づいて最寄りのピザ屋を見つけ、注文を行います。
- プロンプトのカスタマイズ - セッション属性を参照するようにプロンプトとレスポンスカードを設定します。例: 「[FirstName] 様、トッピングは何になさいますか?」 ユーザーの名前をセッション属性 (`{"FirstName": "Jo"}`) として渡すと、Amazon Lex はプレースホルダをその名前に置き換えます。次に、カスタマイズしたプロンプトをユーザーに送信します:「Jo 様、トッピングは何になさいますか?」

セッション属性は、セッションの期間にわたって保持されます。Amazon Lex では、セッションが終わるまで、セッション属性を暗号化されたデータストアに保存します。クライアントは、PostContent オペレーションまたは PostText オペレーションを呼び出し、sessionAttributes フィールドに値を設定することで、リクエストのセッション属性を作成できます。Lambda 関数は、レスポンスのセッション属性を作成できます。クライアントまたは Lambda 関数でセッション属性を作成すると、クライアントアプリケーションで Amazon Lex へのリクエストに sessionAttribute フィールドを指定しない場合に、いつでも保存された属性値が使用されます。

例えば、2 つのセッション属性 {"x": "1", "y": "2"} があるとします。クライアントが PostContent オペレーションまたは PostText オペレーションを呼び出すときに sessionAttributes フィールドを指定しない場合、Amazon Lex は保存されたセッション属性 ({"x": 1, "y": 2}) を使用して Lambda 関数を呼び出します。Lambda 関数からセッション属性が返されない場合、Amazon Lex は保存されたセッション属性をクライアントアプリケーションに返します。

クライアントアプリケーションまたは Lambda 関数のいずれかがセッション属性を渡すと、Amazon Lex は保存されたセッション属性を更新します。既存の値　{"x": 2} などを渡すと、保存された値が更新されます。新しい一連のセッション属性 ({"z": 3} など) を渡すと、既存の値は削除され、新しい値のみが保持されます。空のマップ {} を渡すと、保存された値が消去されます。

セッション属性を Amazon Lex に送信するには、属性の文字列間マップを作成します。セッション属性のマッピング方法を以下に示します。

```
{
   "attributeName": "attributeValue",
   "attributeName": "attributeValue"
}
```

PostText オペレーションの場合は、次に示すように、[sessionAttributes] フィールドを使用してリクエストの本文にマップを挿入します。

```
"sessionAttributes": {
   "attributeName": "attributeValue",
   "attributeName": "attributeValue"
}
```

PostContent オペレーションの場合は、マップを base64 エンコードし、それを x-amz-lex-session-attributes ヘッダーとして送信します。

バイナリまたは構造化されたデータをセッション属性で送信する場合は、最初にデータを単純な文字列に変換する必要があります。詳細については、「複雑な属性の設定」を参照してください。

リクエスト属性の設定

リクエスト属性は、リクエスト固有の情報を示し、現在のリクエストにのみ適用されます。クライアントアプリケーションは、この情報を Amazon Lex に送信します。セッション全体を通しては保持する必要がない情報は、リクエスト属性を使用して渡します。リクエスト属性は、独自に作成することも、事前定義されたものを使用することもできます。リクエスト属性を送信するには、the section called “PostContent” の `x-amz-lex-request-attributes` ヘッダーを使用するか、the section called “PostText” リクエストの `requestAttributes` フィールドを使用します。セッション属性とは異なり、リクエスト属性は複数のリクエストにわたって保持されないため、`PostContent` レスポンスや `PostText` レスポンスで返されることはありません。

> **Note**
>
> 複数のリクエストにわたって保持される情報を送信するには、セッション属性を使用します。

名前空間 `x-amz-lex:` は、事前定義されたリクエスト属性用に予約されています。リクエスト属性をプレフィックス `x-amz-lex:` で作成しないでください。

事前定義されたリクエスト属性の設定

Amazon Lex には、ボットに送信される情報の処理方法を管理するための事前定義されたリクエスト属性があります。事前定義されたリクエスト属性は、セッション全体にわたって保持されないため、リクエストごとに属性を送信する必要があります。すべての事前定義された属性は `x-amz-lex:` 名前空間にあります。

Amazon Lex には、以下の定義済み属性に加えて、メッセージングプラットフォーム用の定義済み属性が用意されています。これらの属性のリストについては、「メッセージングプラットフォームで Amazon Lex ボットをデプロイする」を参照してください。

レスポンスタイプの設定

異なる機能を持つクライアントアプリケーションが 2 つある場合は、レスポンスのメッセージ形式の制限が必要な場合もあります。例えば、ウェブクライアントに送信するメッセージをプレーンテ

キストに制限し、モバイルクライアントではプレーンテキストと音声合成マークアップ言語 (SSML) の両方を使用できるようにしたいといった場合も考えられます。PostContent と PostText オペレーションが返すメッセージの形式を設定するには、x-amz-lex:accept-content-types" リクエストの属性を使用します。

次のメッセージタイプを任意に組み合わせて属性を設定することができます。

- PlainText - メッセージには UTF-8 形式テキストが含まれています。
- SSML - メッセージには音声出力のテキスト形式が含まれています。
- CustomPayload - メッセージにはクライアント向けに作成したカスタム形式が含まれています。アプリケーションのニーズを満たすようにペイロードを定義することができます。

Amazon Lex はレスポンスの [Message] フィールドで指定したタイプを使用するメッセージのみを返します。カンマで区切れば複数の値を設定できます。メッセージグループを使用している場合は、各メッセージグループで少なくとも 1 つ指定したタイプのメッセージが含まれている必要があります。それ以外の場合は、NoUsableMessageException エラーが発生します。詳細については、「メッセージグループ」を参照してください。

> **Note**
>
> x-amz-lex:accept-content-types リクエストの属性は、HTML 本文のコンテンツには影響しません。PostText オペレーションのレスポンスの内容は常に UTF-8 形式テキストです。PostContent オペレーションレスポンスの本文には、リクエストの Accept ヘッダーで設定した形式のデータが含まれています。

優先タイムゾーンの設定

ユーザーのタイムゾーンを基準として日付を解決するようにタイムゾーンを設定するには、x-amz-lex:time-zone リクエスト属性を使用します。x-amz-lex:time-zone 属性にタイムゾーンを指定しないと、ボットで使用しているリージョンに応じたデフォルトのタイムゾーンが使用されます。

リージョン	デフォルトのタイムゾーン
米国東部 (バージニア北部)	America/New_York
米国西部 (オレゴン)	America/Los_Angeles

リージョン	デフォルトのタイムゾーン
アジアパシフィック (シンガポール)	Asia/Singapore
アジアパシフィック (シドニー)	Australia/Sydney
アジアパシフィック (東京)	Asia/Tokyo
欧州 (フランクフルト)	Europe/Berlin
ヨーロッパ (アイルランド)	Europe/Dublin
欧州 (ロンドン)	Europe/London

例えば、「何日にパッケージを配達しましょうか?」というプロンプトに対して、ユーザーがレスポンスで tomorrow と回答した場合、パッケージを配達する実際の日付は、ユーザーのタイムゾーンによって異なります。例えば、ニューヨークの 9 月 16 日 01:00 時は、ロサンゼルスの 9 月 15 日 22:00 時です。米国東部 (バージニア北部 リージョンにいる人物が、デフォルトタイムゾーンを使用してパッケージの配達日を「明日」に指定した場合、パッケージは 16 日ではなく、17 日に配達されます。x-amz-lex:time-zone リクエスト属性を America/Los_Angeles に設定すると、パッケージは 16 日に配達されます。

属性は、IANA (Internet Assigned Number Authority) のタイムゾーン名のいずれかに設定できます。タイムゾーン名のリストについては Wikipedia の「List of tz database time zones」をご覧ください。

ユーザー定義のリクエスト属性の設定

ユーザー定義のリクエスト属性は各リクエストでボットに送信するデータです。この情報を送信するには、PostContent リクエストの amz-lex-request-attributes ヘッダーを使用するか、PostText リクエストの requestAttributes フィールドを使用します。

リクエスト属性を Amazon Lex に送信するには、属性の文字列間マップを作成します。リクエスト属性のマッピング方法を以下に示します。

```
{
   "attributeName": "attributeValue",
   "attributeName": "attributeValue"
}
```

PostText オペレーションの場合は、次に示すように、[requestAttributes] フィールドを使用してリクエストの本文にマップを挿入します。

```
"requestAttributes": {
   "attributeName": "attributeValue",
   "attributeName": "attributeValue"
}
```

PostContent オペレーションの場合は、マップを base64 エンコードし、それを x-amz-lex-request-attributes ヘッダーとして送信します。

バイナリまたは構造化されたデータをリクエスト属性で送信する場合は、最初にデータを単純な文字列に変換する必要があります。詳細については、「複雑な属性の設定」を参照してください。

セッションタイムアウトの設定

会話セッションが終了するまで、Amazon Lex はコンテキスト情報 (スロットデータとセッション属性) を保持します。ボットのセッションの長さを制御するには、セッションタイムアウトを設定します。デフォルトでは、セッションの所要時間は 5 分ですが、0〜1,440 分 (24 時間) の間で任意の所要時間を指定できます。

例えば、ShoeOrdering や OrderShoes などのインテントをサポートする GetOrderStatus ボットを作成したとします。Amazon Lex は、ユーザーのインテントが靴の注文であることを検出すると、スロットデータを求めます。例えば、靴のサイズ、色、ブランドなどを求めます。ユーザーがスロットデータの一部のみを指定して靴の購入を完了しない場合、Amazon Lex はセッションが終わるまで、すべての指定されたスロットデータとセッション属性を記憶します。ユーザーは、セッションの有効期限が切れる前にセッションに戻った場合、残りのスロットデータを指定して購入を完了できます。

セッションタイムアウトは、ボットの作成時に Amazon Lex コンソールで設定します。AWS Command Line Interface (AWS CLI) または API では、PutBot オペレーションを使用してボットを作成または更新するときに、[idleSessionTTLInSeconds] フィールドを設定してタイムアウトを設定します。

インテント間での情報の共有

Amazon Lex では、インテント間で情報を共有できます。インテント間で共有するには、セッション属性を使用します。

例えば、ShoeOrdering ボットのユーザーが靴の注文を開始したとします。ボットは、ユーザーと会話することで靴のサイズ、色、ブランドなどのスロットデータを集めます。ユーザーが注文を行うと、その注文を処理する Lambda 関数では、注文番号を含む orderNumber セッション属性を設定します。注文のステータスを取得するために、ユーザーは GetOrderStatus インテントを使用します。ボットは、ユーザーに発注番号や注文日などのスロットデータを求めます。ボットは、必要な情報を入手すると、注文のステータスを返します。

ユーザーが同じセッション中にインテントを変更すると予想される場合は、最新の注文のステータスを返すようにボットを設計できます。ユーザーに対して注文情報を再度求める代わりに、orderNumber セッション属性を使用してインテント間で情報を共有し、GetOrderStatus インテントを達成できます。ボットでは、そのためにユーザーの最後の注文のステータスを返します。

クロスインテント情報共有の例については、「旅行を予約する」を参照してください。

複雑な属性の設定

セッション属性およびリクエスト属性は、属性と値の文字列間マップです。多くの場合、文字列マップを使用してクライアントアプリケーションとボットの間で属性値を転送できます。ただし、場合によっては、文字列マップに容易に変換できないバイナリデータや複雑な構造の転送が必要になることもあります。例えば、次の JSON オブジェクトは米国の最も人口が多い 3 つの都市の配列を示しています。

```
{
   "cities": [
      {
         "city": {
            "name": "New York",
            "state": "New York",
            "pop": "8537673"
         }
      },
      {
         "city": {
            "name": "Los Angeles",
            "state": "California",
            "pop": "3976322"
         }
      },
      {
         "city": {
            "name": "Chicago",
```

```
            "state": "Illinois",
            "pop": "2704958"
         }
      }
   ]
}
```

このデータの配列は、文字列間マップに適切に変換されません。このような場合は、オブジェクトを単純な文字列に変換し、その文字列を PostContent オペレーションと PostText オペレーションを使用してボットに送信できます。

例えば、JavaScript を使用している場合は、JSON.stringify オペレーションを使用してオブジェクトを JSON に変換し、JSON.parse オペレーションを使用して JSON テキストを JavaScript オブジェクトに変換します。

```
// To convert an object to a string.
var jsonString = JSON.stringify(object, null, 2);
// To convert a string to an object.
var obj = JSON.parse(JSON string);
```

セッション属性を PostContent オペレーションで送信するには、次の JavaScript コードに示すように、セッション属性を base64 エンコードしてからリクエストヘッダーに追加する必要があります。

```
var encodedAttributes = new Buffer(attributeString).toString("base64");
```

バイナリデータを PostContent オペレーションと PostText オペレーションに送信する場合は、最初にバイナリデータを base64 エンコードされた文字列に変換し、次にその文字列をセッション属性の値として送信します。

```
"sessionAttributes" : {
   "binaryData": "base64 encoded data"
}
```

信頼スコアの使用

ユーザーが発話を行うと、Amazon Lex は自然言語理解 (NLU) を使用してユーザーの要求を理解し、適切なインテントを返します。デフォルトでは、Amazon Lex は、ボットによって定義された最も可能性の高いインテントを返します。

場合によっては、Amazon Lex が最も可能性の高いインテントを判断するのが困難なことがあります。例えば、ユーザーが曖昧な発話をした場合や、似たようなインテントが2つある場合などです。適切なインテントを決定するために、代替えインテントのリストで、ドメイン知識と信頼性スコアを組み合わせることができます。信頼スコアは、Amazon Lex が提供する評価で、インテントが正しいインテントであるという確信度を示します。

代替えインテントの 2 つのインテントの差を判断するには、信頼度スコアを比較します。例えば、あるインテントの信頼スコアが 0.95 で、別のインテントのスコアが0.65の場合、最初のインテントはおそらく正しいでしょう。ただし、あるインテントのスコアが 0.75 で、別のインテントのスコアが 0.72 の場合、2 つのインテントの間には曖昧さがあり、アプリケーションでドメインナレッジを使用して識別できます。

また、信頼度スコアを使用して、インテントの発話に対する変更がボットの動作に違いをもたらすかどうかを判断するテストアプリケーションを作成することもできます。例えば、一連の発話を使用してボットのインテントの信頼度スコアを取得し、新しい発話でインテントを更新することができます。その後、信頼スコアをチェックして、改善があったかどうかを確認することができます。

Amazon Lex が返す信頼スコアは、比較のための値です。絶対的なスコアとして信頼するべきではありません。この値は、Amazon Lex の改善に基づいて変更される場合があります。

信頼度スコアを使用した場合、Amazon Lex は、各応答において、最も可能性の高いインテントと、最大 4 つの代替インテントを、それぞれの関連スコアとともに返します。すべての信頼度スコアがしきい値よりも小さい場合、Amazon Lex は `AMAZON.FallbackIntent`、`AMAZON.KendraSearchIntent`、またはその両方 (設定されている場合) を含みます。デフォルトのしきい値を使用することもできますし、独自のしきい値を設定することもできます。

次のJSONコードは、PostText オペレーションのレスポンスに含まれる `alternativeIntents` フィールドを示しています。

```
"alternativeIntents": [
   {
      "intentName": "string",
```

```
        "nluIntentConfidence": {
           "score": number
        },
        "slots": {
           "string" : "string"
        }
     }
  ],
```

ボットを作成または更新するときにしきい値を設定します。API または Amazon Lex コンソールのいずれかを使用できます。以下にリストされているリージョンでは、精度の向上と信頼度スコアを有効にするためにオプトインする必要があります。コンソールでは、「詳細オプション」(Advanced Options) セクションで [信頼スコア] (confidence scores) を選択します。API を使用して、PutBot オペレーションを呼び出す際に `enableModelImprovements` パラメータを設定します。

- 米国東部 (バージニア北部) (us-east-1)
- 米国西部 (オレゴン) (us-west-2)
- アジアパシフィック (シドニー) (ap-southeast-2)
- 欧州 (アイルランド) (eu-west-1)

その他の地域では、精度の向上と信頼性スコアのサポートがデフォルトで利用できます。

信頼度しきい値を変更するには、コンソールで設定するか、PutBot オペレーションで設定します。しきい値は 1.00 から 0.00 の間の数値である必要があります。

コンソールを使用するには、ボットの作成時または更新時に信頼度のしきい値を設定します。

ボットの作成時に信頼度しきい値を設定するには (コンソール)

- [Create your bot] (ボットの作成) で、[Confidence score threshold] (信頼度スコアのしきい値) フィールドに値を入力します。

信頼性のしきい値を更新するには (コンソール)

1. ボットの一覧から、エクスポートするボットを選択します。
2. [Settings] (設定) タブを選択します。
3. 左側のナビゲーションペインで [General] (全般) を選択します。
4. [Confidence score threshold] (信頼スコアのしきい値) フィールドの値を更新します。

信頼度しきい値 (SDK) を設定または更新するには

- PutBot のオペレーションの nluIntentConfidenceThreshold のパラメータを設定します。次の JSON コードは、パラメータが設定されていることを示しています。

```
"nluIntentConfidenceThreshold": 0.75,
```

セッションの管理

Amazon Lex がユーザーとの会話で使用するインテントを変更するには、ダイアログコードフックの Lambda 関数からの応答を使用するか、カスタムアプリケーションでセッション管理 API を使用します。

Lambda 関数を使用する

Lambda 関数を使用する場合、Amazon Lex は関数への入力を含む JSON 構造で呼び出します。JSON 構造は、Amazon Lex がユーザーの発話の最も可能性の高いインテントとして特定したインテントを含む、currentIntent と呼ばれるフィールドを含んでいます。JSON 構造は、ユーザーのインテントを満たす可能性のある最大 4 つの追加インテントを含む alternativeIntents フィールドも含んでいます。各インテントには、Amazon Lex がそのインテントに割り当てた信頼度スコアを含む nluIntentConfidenceScore というフィールドが含まれます。

別のインテントを使用するには、Lambda 関数の ConfirmIntent または ElicitSlot ダイアログアクションで指定します。

詳細については、「Lambda 関数を使用する」を参照してください。

セッション管理 API を使用する

現在のインテントと異なるインテントを使用するには、PutSession オペレーションを使用します。例えば、Amazon Lex が選択したインテントよりも、最初の選択肢が望ましいと判断した場合、PutSession オペレーションを使用してインテントを変更し、ユーザーが次に対話するインテントが選択したものになるようにすることが可能です。

詳細については、「Amazon Lex API を使用したセッションの管理」を参照してください。

会話ログ

会話ログを有効にして、ボットとのやりとりを保存します。これらのログを使用して、ボットのパフォーマンスを確認し、会話に関する問題のトラブルシューティングを行うことができます。PostText オペレーションのテキストをログに記録できます。PostContent オペレーションのテキストとオーディオの両方を記録できます。会話ログを有効にすると、ユーザーとボットとの会話の詳細ビューが表示されます。

例えば、ボットとのセッションにはセッション ID があります。この ID を使用して、ユーザー発話および対応するボットの応答を含む会話のトランスクリプトを取得できます。また、発話のインテント名やスロット値などのメタデータも取得します。

> **Note**
>
> 児童オンラインプライバシー保護法 (COPPA) の対象となるボットでは、会話ログを使用することはできません。

会話ログは、エイリアスに対して設定されます。各エイリアスで、テキストログとオーディオログに対して異なる設定を使用できます。テキストログ、オーディオログ、またはその両方をエイリアスごとに有効にできます。テキストログでは、テキスト入力、オーディオ入力の書き起こし、および関連するメタデータが CloudWatch Logs に保存されます。オーディオログは、Amazon S3 にオーディオ入力を保存します。AWS KMS カスタマー管理の CMK を使用して、テキストログとオーディオログの暗号化を有効にできます。

ログ記録を設定するには、コンソールまたは PutBotAlias オペレーションを使用します。ボットの $LATEST エイリアス、または Amazon Lex コンソールで利用可能なテストボットの会話をログに記録することはできません。エイリアスの会話ログを有効にすると、PostContent、またはそのエイリアスの PostText オペレーションが、設定された CloudWatch Logs ロググループまたは S3 バケット内のテキストまたは音声発話をログに記録します。

トピック

- 会話ログの IAM ポリシー
- 会話ログの設定
- 会話ログの暗号化
- Amazon CloudWatch Logs のテキストログの表示
- Amazon S3 のオーディオログへのアクセス

- CloudWatch メトリクスを使用した会話ログのステータスのモニタリング

会話ログの IAM ポリシー

選択したログ記録のタイプに応じて、Amazon Lex には、ログを保存するために Amazon CloudWatch Logs と Amazon Simple Storage Service (S3) バケットを使用するアクセス許可が必要です。これらのリソースに Amazon Lex がアクセスできるようにするには、AWS Identity and Access Management ロールとアクセス許可を作成する必要があります。

会話ログ用の IAM ロールとポリシーの作成

会話ログを有効にするには、CloudWatch Logs および Amazon S3 の書き込み権限を付与する必要があります。S3 オブジェクトのオブジェクト暗号化を有効にする場合は、オブジェクトの暗号化に使用する AWS KMS キーへのアクセス許可を付与する必要があります。

IAM AWS Management Console、IAM API あるいは AWS Command Line Interface を使用して、ロールポリシーを作成して埋め込むことができます。この手順では、AWS CLI を使用してロールとポリシーを作成します。コンソールを使用しての ポリシーの作成の詳細については、「AWS Identity and Access Management User Guide」(AWS Identity and Access Management ユーザーガイド) の「Creating policies on the JSON tab」(JSON タブでポリシーを作成する) を参照してください。

> **Note**
>
> 次のコードは、Linux と MacOS 用にフォーマットされています。Windows の場合、Linux 行連結記号 (\) をキャレット (^) に置き換えます。

会話ログの IAM ロールを作成するには

1. **LexConversationLogsAssumeRolePolicyDocument.json** という現在のディレクトリにドキュメントを作成し、次のコードを追加して保存します。このポリシードキュメントは、信頼されたエンティティとしてロールに Amazon Lex を追加します。これにより、Lex は、会話ログ用に設定されたリソースにログを配信するロールを引き受けることができます。

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lex.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

2. AWS CLI で、次のコマンドを実行して、会話ログの IAM ロールを作成します。

```
aws iam create-role \
    --role-name role-name \
    --assume-role-policy-document file://
LexConversationLogsAssumeRolePolicyDocument.json
```

次に、Amazon Lex が CloudWatch Logs に書き込むことができるロールにポリシーを作成し、アタッチします。

会話テキストを CloudWatch Logs にログ記録するための IAM ポリシーを作成するには

1. **LexConversationLogsCloudWatchLogsPolicy.json** という現在のディレクトリにドキュメントを作成し、次の IAM ポリシーを追加して保存します。

```
{
  "Version": "2012-10-17",
  "Statement": [
      {
          "Effect": "Allow",
          "Action": [
              "logs:CreateLogStream",
              "logs:PutLogEvents"
          ],
          "Resource": "arn:aws:logs:region:account-id:log-group:log-group-name:*"
      }
  ]
}
```

2. AWS CLI で、CloudWatch Logs ロググループに書き込み権限を付与する IAM ポリシーを作成します。

```
aws iam create-policy \
    --policy-name cloudwatch-policy-name \
    --policy-document file://LexConversationLogsCloudWatchLogsPolicy.json
```

3. 会話ログ用に作成した IAM ロールにポリシーをアタッチします。

```
aws iam attach-role-policy \
    --policy-arn arn:aws:iam::account-id:policy/cloudwatch-policy-name \
    --role-name role-name
```

オーディオを S3 バケットにログ記録する場合は、Amazon Lex がバケットに書き込むことを可能にするポリシーを作成します。

S3 バケットへのオーディオログ記録のための IAM ポリシーを作成するには

1. **LexConversationLogsS3Policy.json** という現在のディレクトリにドキュメントを作成し、次のポリシーを追加して保存します。

```
{
  "Version": "2012-10-17",
  "Statement": [
      {
          "Effect": "Allow",
          "Action": [
              "s3:PutObject"
          ],
          "Resource": "arn:aws:s3:::bucket-name/*"
      }
  ]
}
```

2. AWS CLI で、S3 バケットへの書き込みアクセス権限を付与する IAM ポリシーを作成します。

```
aws iam create-policy \
    --policy-name s3-policy-name \
    --policy-document file://LexConversationLogsS3Policy.json
```

3. 会話ログ用に作成したロールにポリシーをアタッチします。

```
aws iam attach-role-policy \
```

```
    --policy-arn arn:aws:iam::account-id:policy/s3-policy-name \
    --role-name role-name
```

IAM ロールを渡すアクセス許可の付与

コンソール、AWS Command Line Interface、または AWS SDK を使用して会話ログに使用する IAM ロールを指定する場合、会話ログ IAM ロールを指定するユーザーには、ロールを Amazon Lex に渡すアクセス許可が必要です。ユーザーが Amazon Lex サービスにロールを渡すには、ユーザー、ロール、またはグループに PassRole アクセス許可を付与する必要があります。

次のポリシーは、ユーザー、ロール、またはグループに付与するアクセス許可を定義します。iam:AssociatedResourceArn 条件キーと iam:PassedToService 条件キーを使用して、アクセス許可の範囲を制限できます。詳細については、「AWS Identity and Access Management ユーザーガイド」の「AWS サービスに IAM と AWS STS Condition Context Keys ロールを渡す許可をユーザーに付与する」をご覧ください。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::account-id:role/role-name",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "lex.amazonaws.com"
                },
                "StringLike": {
                    "iam:AssociatedResourceARN": "arn:aws:lex:region:account-
id:bot:bot-name:bot-alias"
                }
            }
        }
    ]
}
```

会話ログの設定

会話ログを有効または無効にするには、コンソールまたは PutBotAlias オペレーションの conversationLogs フィールドを使用します。オーディオログ、テキストログ、またはその両方を

オンまたはオフにできます。新しいボットセッションでログ記録が開始されます。ログ設定への変更は、アクティブなセッションでは反映されません。

テキストログを保存するには、AWS アカウントで Amazon CloudWatch Logs ロググループを使用します。任意の有効なロググループを使用できます。ロググループは、Amazon Lex ボットと同じリージョンに存在する必要があります。CloudWatch Logs ロググループの詳細については、「Amazon CloudWatch Logs User Guide」(Amazon CloudWatch Logs ユーザーガイド) の「Working with Log Groups and Log Streams」(ロググループとログストリームの使用) を参照してください。

オーディオログを保存するには、AWS アカウントで Amazon S3 バケットを使用します。任意の有効な S3 バケットを使用できます。バケットは Amazon Lex ボットと同じリージョンにある必要があります。S3 バケットの作成の詳細については、「Amazon Simple Storage Service Getting Started Guide」(Amazon Simple Storage Service 入門ガイド) の「Create a Bucket」(バケットの作成) を参照してください。

設定済みのロググループまたはバケットへの Amazon Lex による書き込みを有効にするポリシーのある IAM ロールを指定する必要があります。詳細については、「会話ログ用の IAM ロールとポリシーの作成」を参照してください。

AWS Command Line Interface を使用してサービスにリンクされたロールを作成する場合は、以下のように `custom-suffix` オプションを使用してロールにカスタムサフィックスを追加する必要があります。

```
aws iam create-service-linked-role \
    --aws-service-name lex.amazon.aws.com \
    --custom-suffix suffix
```

会話ログを有効にするために使用する IAM ロールには、`iam:PassRole` アクセス許可が必要です。以下のポリシーをロールにアタッチする必要があります。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::account:role/role"
        }
    ]
}
```

会話ログの有効化

コンソールを使用してログを有効にするには

1. Amazon Lex コンソール (https://console.aws.amazon.com/lex) を開きます。
2. リストからボットを選択します。
3. [Settings] (設定) タブを選択し、左側のメニューから [Conversation logs] (会話ログ) を選択します。
4. エイリアスのリストで、会話ログを設定するエイリアスの設定アイコンを選択します。
5. テキスト、オーディオ、またはその両方をログに記録するかどうかを選択します。
6. テキストログの場合は、Amazon CloudWatch Logs のロググループ名を入力します。
7. オーディオのログ記録の場合は、S3 バケット情報を入力します。
8. オプション。オーディオログを暗号化するには、暗号化に使用する AWS KMS キーを選択します。
9. 必要な権限を持つ IAM ロールを選択します。
10. [Save] (保存) を選択して、会話のログ記録を開始します。

API を使用してテキストログを有効にするには

1. `conversationLogs` フィールドの `logSettings` メンバーのエントリを使用して PutBotAlias オペレーションを呼び出す
   - `destination` メンバーを `CLOUDWATCH_LOGS` に設定する
   - `logType` メンバーを `TEXT` に設定する
   - ログの送信先である CloudWatch Logs ロググループの Amazon リソースネーム (ARN) に `resourceArn` メンバーを設定する
2. `conversationLogs` フィールドの `iamRoleArn` メンバーを、指定したリソースで会話ログを有効にするために必要なアクセス権限を持つ IAM ロールの Amazon リソースネーム (ARN) に設定します。

API を使用してオーディオログを有効にするには

1. `conversationLogs` フィールドの `logSettings` メンバーのエントリを使用して PutBotAlias オペレーションを呼び出す

- destination メンバーを S3 に設定する
- logType メンバーを AUDIO に設定する
- resourceArn メンバーをオーディオログが保存されている Amazon S3 バケットの ARN に設定する
- オプション。特定の AWS KMS キーでオーディオログを暗号化するには、暗号化に使用するキーの ARN の kmsKeyArn メンバーを設定します。

2. conversationLogs フィールドの iamRoleArn メンバーを、指定したリソースで会話ログを有効にするために必要なアクセス権限を持つ IAM ロールのAmazon リソースネーム (ARN) に設定します。

会話ログの無効化

コンソールを使用してログをオフにするには

1. Amazon Lex コンソール (https://console.aws.amazon.com/lex) を開きます。
2. リストからボットを選択します。
3. [Settings] (設定) タブを選択し、左側のメニューから [Conversation logs] (会話ログ) を選択します。
4. エイリアスのリストで、会話ログを設定するエイリアスの設定アイコンを選択します。
5. ログ記録をオフにするには、テキスト、オーディオ、またはその両方からチェックを外します。
6. 会話のログ記録を停止するには、[Save] (保存) を選択します。

API を使用してログをオフにするには

- conversationLogs フィールドを使用しないで PutBotAlias オペレーションを呼び出します。

API を使用してテキストログを無効にするには

- - オーディオを記録する場合
    - AUDIO の logSettings エントリのみを使用して PutBotAlias オペレーションを呼び出します。
    - PutBotAlias オペレーションの呼び出しには、TEXT の logSettings エントリを含めることはできません。

- オーディオを口グ記録しない場合
  - conversationLogs フィールドを使用しないで PutBotAlias オペレーションを呼び出します。

API を使用してオーディオログをオフにするには

- • テキストを口グ記録する場合
  - TEXT の logSettings エントリのみを使用して PutBotAlias オペレーションを呼び出します。
  - PutBotAlias オペレーションの呼び出しには、AUDIO の logSettings エントリを含めることはできません。
- テキストを口グ記録していない場合
  - conversationLogs フィールドを使用しないで PutBotAlias オペレーションを呼び出します。

会話口グの暗号化

暗号化を使用すると、会話口グの内容を保護できます。テキストログとオーディオログの場合、AWS KMS カスタマー管理の CMK を使用して、CloudWatch Logs ロググループと S3 バケットのデータを暗号化できます。

> **Note**
>
> Amazon Lex は、シンメトリック CMKのみをサポートします。非対称 CMK を使用して、データを暗号化することはできません。

Amazon Lex がテキストログに使用する CloudWatch Logs ロググループの AWS KMS キーを使用して、暗号化を有効にします。口グ設定に AWS KMS キーを指定して、ロググループの AWS KMS 暗号化を有効にすることはできません。詳しくは、Amazon CloudWatch Logs ユーザーガイドの「AWS KMS を使用して CloudWatch Logs の口グデータを暗号化する」を参照してください。

オーディオログの場合は、S3 バケットでデフォルトの暗号化を使用するか、オーディオオブジェクトを暗号化する AWS KMS キーを指定します。S3 バケットでデフォルトの暗号化が使用されている場合でも、別の AWS KMS キーを指定してオーディオオブジェクトを暗号化できます。詳細につい

ては、Amazon Simple Storage Service 開発者ガイドの「S3 バケットの Amazon S3 デフォルト暗号化」を参照してください。

Amazon Lex では、オーディオログの暗号化を選択した場合に AWS KMS 許可が必要です。会話ログに使用する IAM ロールに追加のポリシーをアタッチする必要があります。S3 バケットでデフォルトの暗号化を使用する場合、ポリシーでそのバケットに設定された AWS KMS キーへのアクセスを許可する必要があります。オーディオログ設定で AWS KMS キーを指定する場合、そのキーへのアクセスを許可する必要があります。

会話ログのロールを作成していない場合は、「会話ログの IAM ポリシー」を参照してください。

オーディオログの暗号化に AWS KMS キーを使用するための IAM ポリシーを作成するには

1. **LexConversationLogsKMSPolicy.json** という現在のディレクトリにドキュメントを作成し、次のポリシーを追加して保存します。

```
{
  "Version": "2012-10-17",
  "Statement": [
      {
          "Effect": "Allow",
          "Action": [
              "kms:GenerateDataKey"
          ],
          "Resource": "kms-key-arn"
      }
  ]
}
```

2. AWS CLI で、オーディオログの暗号化に AWS KMS キーを使用するアクセス権限を付与する IAM ポリシーを作成します。

```
aws iam create-policy \
    --policy-name kms-policy-name \
    --policy-document file://LexConversationLogsKMSPolicy.json
```

3. 会話ログ用に作成したロールにポリシーをアタッチします。

```
aws iam attach-role-policy \
    --policy-arn arn:aws:iam::account-id:policy/kms-policy-name \
    --role-name role-name
```

Amazon CloudWatch Logs のテキストログの表示

Amazon Lex では、Amazon CloudWatch Logs で会話のテキストログが保存されます。ログを表示するには、CloudWatch Logs コンソールまたは API を使用します。詳細については、「Amazon CloudWatch Logs ユーザーガイド」の「フィルターパターンを使用したログデータの検索」と「CloudWatch Logs インサイトクエリ構文」を参照してください。

Amazon Lex コンソールを使用してログを表示するには

1. Amazon Lex コンソール (https://console.aws.amazon.com/lex) を開きます。
2. リストからボットを選択します。
3. [Settings] (設定) タブを選択し、左側のメニューから [Conversation logs] (会話ログ) を選択します。
4. [Text logs] (テキストログ) の下のリンクを選択して、エイリアスのログを CloudWatch コンソールに表示します。

CloudWatch コンソールまたは API を使用してログエントリを表示することもできます。ログエントリを見つけるには、エイリアスに対して設定したロググループに移動します。Amazon Lex コンソールでログのログストリームプレフィックスを見つけるか、GetBotAlias オペレーションを使用します。

ユーザー発話のログエントリは、複数のログストリームにあります。会話内の発話には、指定されたプレフィックスを持つログストリームの 1 つにエントリがあります。ログストリームのエントリには、次の情報が含まれます。

```
{
   "messageVersion": "1.0",
   "botName": "bot name",
   "botAlias": "bot alias",
   "botVersion": "bot version",
   "inputTranscript": "text used to process the request",
   "botResponse": "response from the bot",
   "intent": "matched intent",
   "nluIntentConfidence": "number",
   "slots": {
       "slot name": "slot value",
       "slot name": null,
       "slot name": "slot value"
       ...
```

```
  },
  "alternativeIntents": [
      {
          "name": "intent name",
          "nluIntentConfidence": "number",
          "slots": {
              "slot name": slot value,
              "slot name": null,
              "slot name": slot value
              ...
          }
      },
      {
          "name": "intent name",
          "nluIntentConfidence": number,
          "slots": {}
      }
  ],
  "developerOverride": "true" | "false",
  "missedUtterance": true | false,
  "inputDialogMode": "Text" | "Speech",
  "requestId": "request ID",
  "s3PathForAudio": "S3 path to audio file",
  "userId": "user ID",
  "sessionId": "session ID",
  "sentimentResponse": {
      "sentimentScore": "{Positive: number, Negative: number, Neutral: number,
Mixed: number}",
      "sentimentLabel": "Positive" | "Negative" | "Neutral" | "Mixed"
  },
  "slotToElicit": "slot name",
  "dialogState": "ElicitIntent" | "ConfirmIntent" | "ElicitSlot" | "Fulfilled" |
"ReadyForFulfillment" | "Failed",
  "responseCard": {
      "genericAttachments": [
          ...
      ],
      "contentType": "application/vnd.amazonaws.card.generic",
      "version": 1
   },
  "locale": "locale",
  "timestamp": "ISO 8601 UTC timestamp",
  "kendraResponse": {
     "totalNumberOfResults": number,
```

```
    "resultItems": [
        {
            "id": "query ID",
            "type": "DOCUMENT" | "QUESTION_ANSWER" | "ANSWER",
            "additionalAttributes": [
                {
                   ...
                }
            ],
            "documentId": "document ID",
            "documentTitle": {
                "text": "title",
                "highlights": null
            },
            "documentExcerpt": {
                "text": "text",
                "highlights": [
                    {
                        "beginOffset": number,
                        "endOffset": number,
                        "topAnswer": true | false
                    }
                ]
            },
            "documentURI": "URI",
            "documentAttributes": []
        }
    ],
    "facetResults": [],
    "sdkResponseMetadata": {
        "requestId": "request ID"
    },
    "sdkHttpMetadata": {
        "httpHeaders": {
            "Content-Length": "number",
            "Content-Type": "application/x-amz-json-1.1",
            "Date": "date and time",
            "x-amzn-RequestId": "request ID"
        },
        "httpStatusCode": 200
    },
    "queryId": "query ID"
},
"sessionAttributes": {
```

```
        "attribute name": "attribute value"
        ...
    },
   "requestAttributes": {
        "attribute name": "attribute value"
        ...
    }
}
```

ログエントリの内容は、トランザクションの結果、およびボットとリクエストの設定によって異なります。

- missedUtterance フィールドが true の場合、intent、slots、および slotToElicit フィールドはエントリに表示されません。
- オーディオログが無効になっている場合、または inputDialogMode フィールドが Text の場合、s3PathForAudio フィールドは表示されません。
- responseCard フィールドは、ボットの応答カードを定義した場合にのみ表示されます。
- requestAttributes マップは、リクエストでリクエスト属性を指定した場合にのみ表示されます。
- この kendraResponse フィールドが存在するのは、AMAZON.KendraSearchIntent Amazon Kendra インデックスを検索するリクエストを作成したときのみです。
- ボットの Lambda 関数で代替インテントが指定されている場合、この developerOverride フィールドは true です。
- sessionAttributes マップは、リクエストでセッション属性を指定した場合にのみ表示されます。
- sentimentResponse マップは、センチメント値を返すようにボットを設定した場合のみ表示されます。

> **Note**
>
> 入力形式は変わる場合があり、この変更は対応する messageVersion に反映されないことがあります。新しいフィールドが追加されても、コードでエラーがスローされないようにします。

Amazon Lex の CloudWatch Logs への書き込みを有効にするロールとポリシーが必要です。詳細については、「会話ログの IAM ポリシー」を参照してください。

Amazon S3 のオーディオログへのアクセス

Amazon Lex は、会話のオーディオログを S3 バケットに保存します。

コンソールを使用してオーディオログにアクセスするには

1. Amazon Lex コンソール (https://console.aws.amazon.com/lex) を開きます。
2. リストからボットを選択します。
3. [Settings] (設定) タブを選択し、左側のメニューから [Conversation logs] (会話ログ) を選択します。
4. [Audio logs] (オーディオログ) の下のリンクを選択して、Amazon S3 コンソールでエイリアスのログにアクセスします。

Amazon S3 コンソールまたは API を使用してオーディオログにアクセスすることもできます。オーディオファイルの S3 オブジェクトキープレフィックスは、Amazon Lex コンソールまたは `GetBotAlias` オペレーションレスポンスの `resourcePrefix` フィールドに表示されます。

CloudWatch メトリクスを使用した会話ログのステータスのモニタリング

会話ログの配信メトリクスをモニタリングするために Amazon CloudWatch を使用します。メトリクスにアラームを設定して、ログに問題が発生した場合にその問題を認識できます。

Amazon Lex は、会話ログ用の `AWS/Lex` 名前空間に 4 つのメトリクスを提供します。

- `ConversationLogsAudioDeliverySuccess`
- `ConversationLogsAudioDeliveryFailure`
- `ConversationLogsTextDeliverySuccess`
- `ConversationLogsTextDeliveryFailure`

詳細については、「会話ログの CloudWatch メトリクス」を参照してください。

成功のメトリクスは、Amazon Lex がオーディオログまたはテキストログを送信先に正常に書き込んだことを示しています。

失敗のメトリクスは、Amazon Lex が指定された送信先にオーディオログまたはテキストログを配信できなかったことを示しています。通常、これは設定エラーです。失敗のメトリクスがゼロを超える場合は、次の点を確認してください。

- Amazon Lex が IAM ロールの信頼されたエンティティであることを確認します。
- テキストログ記録の場合は、CloudWatch Logs ロググループが存在することを確認します。オーディオログ記録の場合は、S3 バケットが存在することを確認します。
- Amazon Lex が CloudWatch Logs ロググループまたは S3 バケットにアクセスするために使用する IAM ロールに、ロググループまたはバケットに対する書き込み権限があることを確認します。
- S3 バケットが Amazon Lex ボットと同じリージョンに存在し、アカウントに属していることを確認します。
- S3 暗号化に AWS KMS キーを使用している場合は、Amazon Lex がキーを使用することを妨げるポリシーがないことを確認し、提供する IAM ロールに必要な AWS KMS アクセス許可があることを確認します。詳細については、「会話ログの IAM ポリシー」を参照してください。

Amazon Lex API を使用したセッションの管理

ユーザーがボットとの会話を開始すると、Amazon Lex によりセッションが作成されます。アプリケーションと Amazon Lex 間で交換される情報は、会話のセッション状態を構成します。リクエストを行うと、セッションはボット名と指定したユーザー識別子の組み合わせによって識別されます。ユーザー識別子の詳細については、PostContent または PostText オペレーションの「`userId`」フィールドを参照してください。

セッションオペレーションからのレスポンスには、ユーザーとの特定のセッションを識別する一意のセッション識別子が含まれます。この識別子はテスト中やボットのトラブルシューティングに使用できます。

アプリケーションとボット間で送信されるセッション状態を変更できます。例えば、セッションに関するカスタム情報を含むセッション属性を作成および変更できます。また、次の発話を解釈するダイアログコンテキストを設定することで、会話のフローを変更できます。

セッション状態を更新する方法は 2 つあります。最初の方法は、会話の各ターンの後に呼び出される `PostContent` または `PostText` オペレーションで Lambda 関数を使用することです。詳細については、「Lambda 関数を使用する」を参照してください。もう 1 つの方法は、アプリケーションでセッション状態を変更する Amazon Lex ランタイム API を使用することです。

Amazon Lex ランタイム API は、ボットとの会話のセッション情報を管理できるオペレーションを提供します。これらのオペレーションは PutSession、GetSession、DeleteSession です。これらのオペレーションを使用して、ボットとのユーザーセッションの状態に関する情報を取得し、その状態をきめ細かく制御します。

セッションの現在の状態を取得するには、GetSession オペレーションを使用します。このオペレーションは、ユーザーとのダイアログの状態、設定されているセッション属性、ユーザーが操作した最後の 3 つのインテントのスロット値など、セッションの現在の状態を返します。

PutSession オペレーションにより、現在のセッション状態を直接操作できます。ボットによって次に実行されるダイアログアクションのタイプを設定できます。これにより、ボットとの会話のフローを制御できます。ダイアログアクションの type フィールドを Delegate に設定して、Amazon Lex にボットの次のアクションを決定させます。

PutSession オペレーションを使用して、ボットとの新しいセッションを作成し、ボットが開始されるインテントを設定できます。PutSession オペレーションを使用して、あるインテントから別のインテントに変更することもできます。セッションの作成時またはインテントの変更時に、スロット値やセッション属性などのセッション状態を設定することもできます。新しいインテントが終了したら、前のインテントを再開するオプションがあります。GetSession オペレーションを使用して、前のインテントのダイアログ状態を Amazon Lex から取得し、その情報を使用してインテントのダイアログ状態を設定できます。

PutSession オペレーションからのレスポンスには、PostContent オペレーションと同じ情報が含まれます。PostContent オペレーションからのレスポンスの場合と同様に、この情報を使用して、ユーザーに次の情報を求めることができます。

DeleteSession オペレーションを使用して既存のセッションを削除し、新しいセッションからやり直します。例えば、ボットをテストするときは、DeleteSession オペレーションを使用してボットからテストセッションを削除できます。

セッションオペレーションはフルフィルメント Lambda 関数と連携します。例えば、Lambda 関数がフルフィルメント状態として Failed を返す場合、PutSession オペレーションを使用してダイアログアクションタイプを close に設定し、fulfillmentState を ReadyForFulfillment に設定して、フルフィルメントステップを再試行できます。

セッションオペレーションでは以下のことが可能です。

- ユーザーを待たずにボットに会話を開始させる。
- 会話中にインテントを切り替える。

- 前のインテントに戻る。
- 操作の途中で会話を開始または再開する。
- スロット値を検証し、無効であればボットに値の再入力を求めさせる。

これらのそれぞれについて、以下で詳しく説明します。

インテントの切り替え

PutSession オペレーションを使用して、あるインテントから別のインテントに切り替えることができます。このオペレーションを使用して、前のインテントに戻ることもできます。PutSession オペレーションを使用して、新しいインテントのセッション属性またはスロット値を設定できます。

- PutSession オペレーションを呼び出します。インテント名を新しいインテントの名前に設定し、ダイアログアクションを Delegate に設定します。新しいインテントに必要なスロット値またはセッション属性を設定することもできます。
- Amazon Lex は、新しいインテントを使用してユーザーとの会話を開始します。

前のインテントの再開

前のインテントを再開するには、GetSession オペレーションを使用してインテントの要約を取得してから、PutSession オペレーションを使用してインテントを前のダイアログ状態に設定します。

- GetSession オペレーションを呼び出します。このオペレーションからのレスポンスには、ユーザーが操作した最後の 3 つのインテントのダイアログ状態の要約が含まれます。
- インテントの要約からの情報を使用して、PutSession オペレーションを呼び出します。これにより、ユーザーは会話内の同じ場所で前のインテントに戻ります。

場合によっては、ユーザーのボットとの会話を再開する必要があります。例えば、カスタマーサービスボットを作成したとします。アプリケーションは、ユーザーがカスタマーサービス担当者と話す必要があると判断します。ユーザーと話した後、担当者は収集した情報を使用して会話をボットに戻すことができます。

セッションを再開するには、以下のような手順を使用します。

- アプリケーションは、ユーザーがカスタマーサービス担当者と話す必要があると判断します。

- GetSession オペレーションを使用して、インテントの現在のダイアログ状態を取得します。
- カスタマーサービス担当者はユーザーと話し、問題を解決します。
- PutSession オペレーションを使用して、インテントのダイアログ状態を設定します。さらに、スロット値やセッション属性の設定、インテントの変更を行う場合もあります。
- ボットはユーザーとの会話を再開します。

PutSession オペレーションの checkpointLabel パラメータを使用してインテントにラベルを付け、後で検索することができます。例えば、顧客に情報を求めるボットは、顧客が情報を収集している間に Waiting インテントに入ることがあります。ボットは、現在のインテントのチェックポイントラベルを作成し、Waiting インテントを開始します。顧客が戻ると、ボットはチェックポイントラベルを使用して以前のインテントを見つけ、元に戻すことができます。

インテントは、GetSession オペレーションによって返される recentIntentSummaryView 構造体に存在する必要があります。GetSession オペレーションリクエストでチェックポイントラベルを指定すると、そのチェックポイントラベルを持つ最大 3 つのインテントが返されます。

- GetSession オペレーションを使用して、セッションの現在の状態を取得します。
- PutSession オペレーションを使用して、最後のインテントにチェックポイントラベルを追加します。必要に応じて、この PutSession 呼び出しを使用して別のインテントに切り替えることができます。
- ラベル付けされたインテントに戻るときは、GetSession オペレーションを呼び出して最近のインテントリストを返します。checkpointLabelFilter パラメータを使用し、指定したチェックポイントラベルを持つインテントのみを Amazon Lex で返すことができます。

新しいセッションの開始

ボットにユーザーとの会話を開始させる場合は、PutSession オペレーションを使用できます。

- スロットのない挨拶のインテントと、ユーザーにインテントの指定を求める結びのメッセージを作成します。例えば、「ご注文は何になさいますか? 飲み物になさいますか、ピザになさいますか」とします。
- PutSession オペレーションを呼び出します。インテント名を挨拶のインテントの名前に設定し、ダイアログアクションを Delegate に設定します。
- Amazon Lex は、ユーザーとの会話を開始する挨拶のインテントのプロンプトで応答します。

スロット値の検証

クライアントアプリケーションを使用して、ボットへのレスポンスを検証できます。レスポンスが有効でない場合、PutSession オペレーションを使用してユーザーから新しいレスポンスを取得できます。例えば、花の注文ボットがチューリップ、バラ、ユリのみを販売できるとします。ユーザーがカーネーションを注文すると、アプリケーションは以下のことができます。

- PostText または PostContent レスポンスから返されたスロット値を調べます。
- スロット値が無効な場合は、PutSession オペレーションを呼び出します。アプリケーションはスロット値をクリアし、slotToElicit フィールドを設定して、dialogAction.type 値を elicitSlot に設定する必要があります。オプションで、Amazon Lex によってスロット値の引き出しに使用されるメッセージを変更する場合は、message および messageFormat フィールドを設定できます。

ボットのデプロイメントオプション

現在、Amazon Lex では以下のボットのデプロイメントオプションが用意されています。

- AWS Mobile SDK – AWS Mobile SDK を使用すると、Amazon Lex と通信するモバイルアプリケーションを構築できます。
- Facebook Messenger – Facebook Messenger ページと Amazon Lex ボットを統合すると、エンドユーザーは Facebook でボットと通信できます。現在の実装では、この統合でサポートされるのはテキスト入力メッセージのみです。
- Slack – Amazon Lex ボットを Slack メッセージングアプリケーションと統合できます。
- Twilio – Amazon Lex ボットを Twilio Simple Messaging Service (SMS) と統合できます。

例については、「Amazon Lex ボットのデプロイ」を参照してください。

組み込みのインテントとスロットタイプ

Amazon Lex では、Alexa に標準で組み込まれているインテントおよびスロットタイプを使用してボットを簡単に作成できます。

トピック

- 組み込みインテント
- 組み込みスロットタイプ

組み込みインテント

一般的なアクションに対しては、標準の組み込みインテントライブラリを使用できます。組み込みインテントからインテントを作成するには、コンソールで組み込みインテントを選択し、新しい名前を付けます。新しいインテントは、サンプル発話など、元のインテントの設定を継承します。

現在の実装では、以下の操作は実行できません。

- 元のインテントのサンプル発話を追加または削除する
- 組み込みインテントのスロットを設定する

ボットに組み込みインテントを追加するには

1. AWS Management Console にサインインし、Amazon Lex コンソール (https://console.aws.amazon.com/lex/) を開きます。
2. 組み込みインテントを追加するボットを選択します。
3. ナビゲーションペインで、[インテント] の横のプラス (+) を選択します。
4. [インテントの追加] で、[既存のインテントの検索] を選択します。
5. [Search intents] (インテントの検索) ボックスに、ボットに追加する組み込みインテントの名前を入力します。
6. [Copy built-in intent] (組み込みインテントのコピー) で、インテントに名前を付け、[Add] (追加) を選択します。
7. インテントエディタを使用して、ボットに必要なインテントを構成します。

トピック

- AMAZON.CancelIntent
- AMAZON.FallbackIntent
- AMAZON.HelpIntent
- AMAZON.KendraSearchIntent
- AMAZON.PauseIntent
- AMAZON.RepeatIntent
- AMAZON.ResumeIntent
- AMAZON.StartOverIntent
- AMAZON.StopIntent

> **Note**
>
> 英語 (米国) (en-US) ロケールの場合、Amazon Lex は Alexa 標準の組み込みインテントからのインテントをサポートしています。組み込みインテントの一覧については、「Alexa Skills Kit」の「Standard Built-in Intents」を参照してください。
> Amazon Lex では、以下のインテントはサポートされません。
>
> - `AMAZON.YesIntent`
> - `AMAZON.NoIntent`
> - 「Alexa Skills Kit」の「Built-in Intent Library」のインテント

AMAZON.CancelIntent

ユーザーが現在の対話をキャンセルしたいことを示す単語やフレーズに応答します。アプリケーションはこのインテントを使用して、ユーザーとの対話を終了する前に、スロットタイプの値やその他の属性を削除できます。

一般的な発話:

- キャンセル
- 気にしないで
- 忘れて

AMAZON.FallbackIntent

ユーザーがインテントに入力した内容がボットの想定通りでない場合、Amazon Lex がフォールバックインテントを呼び出すように設定できます。例えば、ユーザー入力「I'd to order candy」が `OrderFlowers` ボットのインテントと一致しない場合、Amazon Lex はフォールバックインテントを呼び出してレスポンスを処理します。

組み込みの `AMAZON.FallbackIntent` インテントタイプをボットに追加することで、フォールバックインテントを追加します。インテントを指定するには、PutBot オペレーションを使用するか、コンソールの組み込みインテントのリストからインテントを選択します。

フォールバックインテントを呼び出すには、2 つのステップを使用します。最初のステップでは、フォールバックインテントはユーザーからの入力に基づいてマッチングされます。フォールバックインテントが一致した場合、ボットの動作は、プロンプトに設定された再試行回数によって異なりま

す。例えば、インテントを決定する最大試行回数が 2 の場合、ボットはフォールバックインテントを呼び出す前に、ボットの明確化プロンプトを 2 回返します。

Amazon Lex は、次のような状況でフォールバックインテントを一致させます。

- インテントへのユーザーの入力が、ボットが想定する入力と一致しません
- オーディオ入力がノイズであるか、テキスト入力が単語として認識されません。
- ユーザーの入力があいまいで、Amazon Lex が呼び出すインテントを判断できません。

フォールバックインテントは、次の場合に呼び出されます。

- 会話の開始時に明確化のための試行回数を設定した後で、ボットがインテントとしてユーザー入力を認識しない場合。
- 設定された試行回数後に、インテントがユーザー入力をスロット値として認識しない場合。
- 設定された試行回数後に、インテントが確認プロンプトへの応答としてユーザー入力を認識しない場合。

フォールバックインテントは以下と使用できます。

- フルフィルメント Lambda 関数
- 結論ステートメント
- フォローアッププロンプト

フォールバックインテントに以下を追加することはできません。

- 発話
- スロット
- 初期化および検証 Lambda 関数
- 確認プロンプト

ボットに対して中止ステートメントとフォールバックインテントの両方を設定している場合、Amazon Lex はフォールバックインテントを使用します。ボットに中止ステートメントが必要な場合は、フォールバックインテントのフルフィルメント関数を使用して、中止ステートメントと同じ動作を提供できます。詳細については、「`abortStatement` オペレーション」の「PutBot パラメータ」を参照してください。

明確化プロンプトの使用

ボットに明確化プロンプトを指定する場合、プロンプトはユーザーから有効なインテントを求めるために使用されます。明確化プロンプトは、設定した回数だけ繰り返されます。その後、フォールバックインテントが呼び出されます。

ボットの作成時に明確化プロンプトを設定せず、ユーザーが有効なインテントで会話を開始しない場合、Amazon Lex は直ちにフォールバックインテントを呼び出します。

明確化プロンプトなしでフォールバックインテントを使用すると、Amazon Lex は次の状況下ではフォールバックを呼び出しません。

- ユーザーがフォローアッププロンプトに応答しても、インテントを提供しない場合。例えば、「今日は他に何か好きですか？」というフォローアッププロンプトに応答して、 とすると、ユーザーは「はい」と言います。Amazon Lex には、ユーザーからインテントを取得するための明確化プロンプトがないため、400 Bad Request 例外が返されます。
- AWS Lambda 関数を使用するときは、 `ElicitIntent` ダイアログタイプを返します。Amazon Lex には、ユーザーからインテントを取得するための明確化プロンプトがないため、400 Bad Request 例外が返されます。
- `PutSession` オペレーションを使用するときは、`ElicitIntent` ダイアログタイプを送信します。Amazon Lex には、ユーザーからインテントを取得するための明確化プロンプトがないため、400 Bad Request 例外が返されます。

フォールバックインテントでの Lambda 関数の使用

フォールバックインテントが呼び出されると、レスポンスは PutIntent オペレーションに対する `fulfillmentActivity` パラメータの設定によって異なります。ボットは、次のいずれかを実行します。

- クライアントアプリケーションにインテント情報を返します。
- フルフィルメント Lambda 関数を呼び出します。セッションに設定されたセッション変数を使用して関数を呼び出します。

フォールバックインテントが呼び出されたときのレスポンスの設定の詳細については、「`fulfillmentActivity` オペレーション」の「PutIntent パラメータ」を参照してください。

フォールバックインテントでフルフィルメント Lambda 関数を使用する場合、この関数を使用して、別のインテントを呼び出す、またはコールバック番号の収集やカスタマーサービス担当者とのセッションの開始など、ユーザーとの何らかの通信を行うことができます。

フルフィルメント関数で他のインテントに実行できる任意のアクションをフォールバックインテント Lambda 関数で実行できます。AWS Lambda を使用したフルフィルメント関数の作成の詳細については、「Lambda 関数を使用する」を参照してください 。

フォールバックインテントは、同じセッションで複数回呼び出すことができます。例えば、Lambda 関数で `ElicitIntent` ダイアログアクションを使用して、ユーザーに別のインテントの入力を求めるとします。設定された試行回数の後に Amazon Lex がユーザーのインテントを推測できない場合、フォールバックインテントを再度呼び出します。また、試行回数設定後にユーザーが有効なスロット値で応答しない場合に、フォールバックインテントを呼び出します。

セッション変数を使用して、フォールバックインテントが呼び出された回数を追跡するように Lambda 関数を設定できます。Lambda 関数で設定したしきい値を超えて呼び出された場合、Lambda 関数は別のアクションを実行できます。セッション変数の詳細については、「セッション属性の設定」を参照してください。

AMAZON.HelpIntent

ボットとのやりとり中にユーザーが助けを必要としていることを示す単語やフレーズに応答します。このインテントが呼び出されると、Lambda 関数またはアプリケーションを設定して、ボットの機能に関する情報を提供したり、ヘルプの領域に関するフォローアップ質問をしたり、インタラクションを人間のエージェントに渡すことができます。

一般的な発話:

- help (ヘルプ)
- 助けて
- 助けてくれますか

AMAZON.KendraSearchIntent

Amazon Kendra でインデックス付けしたドキュメントを検索するには、`AMAZON.KendraSearchIntent` インテントを使用します。Amazon Lex がユーザーとの会話の次のアクションを決定できない場合、検索インテントをトリガーします。

AMAZON.KendraSearchIntent は、英語 (米国) (en-US)、および米国東部 (バージニア北部)、米国東部 (バージニア北部)、米国西部 (オレゴン)、欧州 (アイルランド) の各リージョンでのみ利用できます。

Amazon Kendra は、PDF ドキュメントや Microsoft Word ファイルなどの自然言語ドキュメントにインデックス付けする機械学習ベースの検索サービスです。インデックス付けされたドキュメントを検索し、質問に対して以下のタイプのレスポンスを返すことができます。

- 回答
- 質問への回答になる可能性がある FAQ のエントリ
- 質問に関連するドキュメント

AMAZON.KendraSearchIntent の使用例については、「例: Amazon Kendra インデックスを使用する FAQ ボットを作成する」を参照してください。

ボットに AMAZON.KendraSearchIntent インテントを設定した場合、Amazon Lex は、スロットまたはインテントのユーザー発話を判別できないときは常に、そのインテントを呼び出します。例えば、ボットが「ピザのトッピング」というスロットタイプのレスポンスを引き出し、ユーザーが「ピザって何?」と言った場合、Amazon Lex は AMAZON.KendraSearchIntent を呼び出してその質問を処理します。Amazon Kendra からのレスポンスがない場合、会話はボットで設定されたとおりに進みます。

同じボットで AMAZON.KendraSearchIntent と AMAZON.FallbackIntent の両方を使用する場合、Amazon Lex は以下のようにインテントを使用します。

1. Amazon Lex は AMAZON.KendraSearchIntent を呼び出します。インテントは Amazon Kendra Query オペレーションを呼び出します。
2. Amazon Kendra がレスポンスを返す場合、Amazon Lex はユーザーに結果を表示します。
3. Amazon Kendra からのレスポンスがない場合、Amazon Lex はユーザーに再度プロンプトを表示します。以下のアクションは、ユーザーからのレスポンスによって異なります。
   - ユーザーからのレスポンスに、スロット値の入力やインテントの確認など、Amazon Lex が認識する発話が含まれている場合、ユーザーとの会話はボットで設定されたとおりに進みます。
   - ユーザーからのレスポンスに Amazon Lex が認識する発話が含まれていない場合、Amazon Lex は Query オペレーションを再度呼び出します。
4. 設定された再試行回数の後にレスポンスがない場合、Amazon Lex は AMAZON.FallbackIntent を呼び出し、ユーザーとの会話を終了します。

AMAZON.KendraSearchIntent を使用して Amazon Kendra へのリクエストを作成するには、3 つの方法があります。

- 検索インテントからリクエストを作成します。Amazon Lex では、ユーザーの発話を検索文字列として Amazon Kendra を呼び出します。インテントを作成するときに、Amazon Kendra が返すレスポンスの数を制限するクエリフィルタ文字列を定義できます。Amazon Lex は、クエリリクエストでフィルターを使用します。
- ダイアログ Lambda 関数を使用して、追加のクエリパラメータをリクエストに追加します。Amazon Kendra クエリパラメータを含む kendraQueryFilterString フィールドを delegate ダイアログアクションに追加します。Lambda 関数を使用してクエリパラメータをリクエストに追加すると、それらのパラメータは、インテントを作成したときに定義したクエリフィルタよりも優先されます。
- ダイアログ Lambda 関数を使用して、新しいクエリを作成します。Amazon Lex によって送信される完全な Amazon Kendra クエリリクエストを作成できます。delegate ダイアログアクションの kendraQueryRequestPayload フィールドでクエリを指定します。kendraQueryRequestPayload フィールドは kendraQueryFilterString フィールドよりも優先されます。

ボットを作成するときに queryFilterString パラメータを指定したり、ダイアログ Lambda 関数で delegate アクションを呼び出すときに kendraQueryFilterString フィールドを指定したりするには、Amazon Kendra クエリの属性フィルターとして使用する文字列を指定します。文字列が有効な属性フィルターでないと、実行時に InvalidBotConfigException 例外が発生します。属性ドキュメントの詳細については、「Amazon Kendra Developer Guide」(Amazon Kendra デベロッパーガイド) の「Using document attributes to filter queries」(ドキュメント属性を使用してクエリをフィルタリングする) を参照してください。

Amazon Lex が Amazon Kendra に送信するクエリを制御するには、ダイアログ Lambda 関数の kendraQueryRequestPayload フィールドでクエリを指定できます。クエリが有効でない場合、Amazon Lex は InvalidLambdaResponseException の例外を返します。詳細については、「Amazon Kendra Developer Guide」(Amazon Kendra デベロッパーガイド) の「Query operation」(クエリオペレーション) を参照してください。

AMAZON.KendraSearchIntent の使用方法の例については、「例: Amazon Kendra インデックスを使用する FAQ ボットを作成する」を参照してください。

Amazon Kendra 検索の IAM ポリシー

AMAZON.KendraSearchIntent インテントを使用するには、AWS Identity and Access Management (IAM) ポリシーがアタッチされたロールを使用して、Amazon Kendra Query インテントを呼び出すアクセス許可を持つランタイムロールを Amazon Lex が引き受けられるようにする必要があります。使用する IAM 設定は、Amazon Lex コンソールを使用して AMAZON.KendraSearchIntent を作成するか、AWS SDK や AWS Command Line Interface (AWS CLI) を使用して作成するかによって異なります。コンソールを使用する場合、Amazon Lex サービスにリンクされたロールに Amazon Kendra を呼び出すアクセス許可を追加するか、Amazon Kendra Query オペレーションを呼び出すための専用のロールを使用するかを選択できます。AWS CLI または SDK を使用してインテントを作成するときは、Query オペレーションを呼び出すための専用のロールを使用する必要があります。

アクセス許可のアタッチ

コンソールを使用して、Amazon Kendra Query オペレーションに対するアクセス許可をデフォルトの Amazon Lex サービスにリンクされたロールにアタッチできます。サービスにリンクされたロールにアクセス許可をアタッチする場合は、Amazon Kendra インデックスに接続するための専用のランタイムロールを作成して管理する必要はありません。

Amazon Lex コンソールへのアクセスに使用するユーザー、ロール、またはグループには、ロールポリシーを管理するアクセス許可が必要です。以下の IAM ポリシーをコンソールのアクセスロールにアタッチします。これらのアクセス許可を付与すると、既存のサービスにリンクされたロールポリシーを変更するアクセス許可がロールに付与されます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:AttachRolePolicy",
                "iam:PutRolePolicy",
                "iam:GetRolePolicy"
            ],
            "Resource": "arn:aws:iam::*:role/aws-service-role/lex.amazonaws.com/
AWSServiceRoleForLexBots"
        },
        {
            "Effect": "Allow",
            "Action": "iam:ListRoles",
```

```
            "Resource": "*"
        }
    ]
}
```

ロールの指定

コンソール、AWS CLI、または API を使用して、Amazon Kendra Query オペレーションを呼び出すときに使用するランタイムロールを指定できます。

ランタイムロールの指定に使用する ユーザー、ロール、またはグループには、iam:PassRole アクセス許可が必要です。以下のポリシーでは、このアクセス許可を定義しています。iam:AssociatedResourceArn および iam:PassedToService 条件コンテキストキーを使用して、アクセス許可の範囲をさらに制限できます。詳細については、[AWS STS ユーザーガイド] で [IAM と AWS Identity and Access Management 条件コンテキストキー] を参照してください。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::account:role/role"
        }
    ]
}
```

Amazon Lex が Amazon Kendra の呼び出しに使用する必要があるランタイムロールには、kendra:Query アクセス許可が必要です。Amazon Kendra Query オペレーションを呼び出すアクセス許可に既存の IAM ロールを使用する場合、そのロールには以下のポリシーがアタッチされている必要があります。

IAM コンソール、IAM API、または AWS CLI を使用して、ポリシーを作成し、ロールにアタッチすることができます。以下の手順では、AWS CLI を使用してロールとポリシーを作成します。

> **Note**
>
> 次のコードは、Linux と MacOS 用にフォーマットされています。Windows の場合、Linux 行連結記号 (\) をキャレット (^) に置き換えます。

Query オペレーションのアクセス許可をロールに追加するには

1. 現在のディレクトリに **KendraQueryPolicy.json** という名前でドキュメントを作成し、以下のコードを追加して保存します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kendra:Query"
            ],
            "Resource": [
                "arn:aws:kendra:region:account:index/index ID"
            ]
        }
    ]
}
```

2. AWS CLI で次のコマンドを実行して、Amazon Kendra Query オペレーションを実行するための IAM ポリシーを作成します。

```
aws iam create-policy \
    --policy-name query-policy-name \
    --policy-document file://KendraQueryPolicy.json
```

3. Query オペレーションの呼び出しに使用している IAM ロールに、そのポリシーをアタッチします。

```
aws iam attach-role-policy \
    --policy-arn arn:aws:iam::account-id:policy/query-policy-name
    --role-name role-name
```

Amazon Lex サービスにリンクされたロールを更新するか、ボット用に AMAZON.KendraSearchIntent を作成したときに作成したロールを使用するかを選択できます。以下の手順は、使用する IAM ロールを選択する方法を示しています。

AMAZON.KendraSearchIntent のランタイムロールを指定するには

1. AWS Management Console にサインインし、Amazon Lex コンソール (https://console.aws.amazon.com/lex/) を開きます。
2. `AMAZON.KendraSearchIntent` を追加するボットを選択します。
3. [インテント] の横のプラス (+) を選択します。
4. [インテントの追加] で、[既存のインテントの検索] を選択します。
5. [インテントの検索] に「**`AMAZON.KendraSearchIntent`**」と入力し、[追加] を選択します。
6. [組み込みインテントのコピー] にインテントの名前 (「**`KendraSearchIntent`**」など) を入力し、[追加] を選択します。
7. [Amazon Kendra クエリ] セクションを開きます。
8. [IAM ロール] で、以下のいずれかのオプションを選択します。

   - ボットが Amazon Kendra インデックスをクエリできるように Amazon Lex サービスにリンクされたロールを更新するには、[Add Amazon Kendra permissions] (Amazon Kendra アクセス許可の追加) を選択します。
   - Amazon Kendra `Query` オペレーションを呼び出すアクセス許可を持つロールを使用するには、[Use an existing role] (既存のロールを使用) を選択します。

フィルタとしてのリクエスト属性とセッション属性の使用

Amazon Kendra から現在の会話に関連するアイテムへのレスポンスをフィルター処理するには、ボットの作成時に `queryFilterString` パラメータを追加して、セッション属性とリクエスト属性をフィルターとして使用します。インテントを作成するときに属性のプレースホルダーを指定します。それにより、Amazon Lex V2 が Amazon Kendra を呼び出す前にプレースホルダーを値に置き換えます。リクエスト属性の詳細については、「リクエスト属性の設定」を参照してください。セッション属性の詳細については、「セッション属性の設定」を参照してください。

以下に示しているのは、string to filter というリクエスト属性を使用して Amazon Kendra クエリをフィルター処理する `queryFilterString` パラメータの例です。

```
"{"equalsTo": {"key": "City", "value": {"stringValue": "Seattle"}}}"
```

以下に示しているのは、"SourceURI" というセッション属性を使用して Amazon Kendra クエリをフィルター処理する `queryFilterString` パラメータの例です。

```
"{"equalsTo": {"key": "SourceURI","value": {"stringValue": "[FileURL]"}}}"
```

以下に示しているのは、"DepartmentName" というリクエスト属性を使用して Amazon Kendra クエリをフィルター処理する queryFilterString パラメータの例です。

```
"{"equalsTo": {"key": "Department","value": {"stringValue": "((DepartmentName))"}}}"
```

AMAZON.KendraSearchInteng フィルターは Amazon Kendra 検索フィルターと同じ形式を使用します。詳細については、[Amazon Kendra デベロッパーガイド] の [ドキュメント属性を使用して検索結果をフィルターする] を参照してください。

AMAZON.KendraSearchIntent で使用されるクエリフィルタ文字列は、各フィルターの最初の文字には小文字を使用する必要があります。例えば、次は AMAZON.KendraSearchIntent の有効なクエリフィルターです。

```
{
    "andAllFilters": [
        {
            "equalsTo": {
                "key": "City",
                "value": {
                    "stringValue": "Seattle"
                }
            }
        },
        {
            "equalsTo": {
                "key": "State",
                "value": {
                    "stringValue": "Washington"
                }
            }
        }
    ]
}
```

検索レスポンスの使用

Amazon Kendra は、インテントの conclusion ステートメントでの検索に対するレスポンスを返します。フルフィルメント Lambda 関数が結論メッセージを生成しない限り、インテントには conclusion ステートメントが必要です。

Amazon Kendra には 4 タイプのレスポンスがあります。

- `x-amz-lex:kendra-search-response-question_answer-question-<N>` - 検索に一致する FAQ からの質問。
- `x-amz-lex:kendra-search-response-question_answer-answer-<N>` - 検索に一致する FAQ からの回答。
- `x-amz-lex:kendra-search-response-document-<N>` - 発話のテキストに関連するインデックス内のドキュメントからの抜粋。
- `x-amz-lex:kendra-search-response-document-link-<N>` - 発話のテキストに関連するインデックス内のドキュメントからの抜粋。
- `x-amz-lex:kendra-search-response-answer-<N>` - 質問への回答があるインデックス内のドキュメントからの抜粋。

レスポンスは `request` 属性で返されます。各属性には、最大 5 つのレスポンスがあり 1~5 の番号が付けられます。レスポンスの詳細については、[Amazon Kendra デベロッパーガイド] の [レスポンスのタイプ] を参照してください。

`conclusion` ステートメントには、1 つ以上のメッセージグループが必要です。各メッセージグループには、1 つ以上のメッセージが含まれます。各メッセージには、Amazon Kendra からのレスポンスでリクエスト属性によって置き換えられる 1 つ以上のプレースホルダー変数を含めることができます。メッセージ内のすべての変数がランタイムレスポンスのリクエスト属性値で置き換えられるメッセージグループには、1 つ以上のメッセージが必要です。プレースホルダー変数のないメッセージグループには、1 つのメッセージが必要です。リクエスト属性は二重かっこ (( )) で囲みます。以下のメッセージグループのメッセージは Amazon Kendra からのレスポンスに一致します。

- 「FAQ の質問 ((x-amz-lex:kendra-search-response-question_answer-question-1)) を見つけました。回答は ((x-amz-lex:kendra-search-response-question_answer -answer-1)) です」
- 「参考になるドキュメント ((x-amz-lex:kendra-search-response-document-1)) からの抜粋を見つけました」
- 「質問への回答はおそらく ((x-amz-lex:kendra-search-response-answer-1)) です」

Lambda 関数を使用したリクエストとレスポンスの管理

`AMAZON.KendraSearchIntent` インテントでは、ダイアログコードフックとフルフィルメントコードフックを使用して、Amazon Kendra へのリクエストとレスポンスを管理できます。Amazon

Kendra に送信するクエリを変更する場合はダイアログコードフック Lambda 関数を使用し、レスポンスを変更する場合はフルフィルメントコードフック Lambda 関数を使用します。

ダイアログコードフックを使用したクエリの作成

ダイアログコードフックを使用して、Amazon Kendra に送信するクエリを作成できます。ダイアログコードフックを使用するかどうかはオプションです。ダイアログコードフックを指定しない場合、Amazon Lex によってユーザー発話からクエリが作成され、`queryFilterString` が使用されます (インテントの設定時に指定した場合)。

ダイアログコードフックレスポンスで 2 つのフィールドを使用して、Amazon Kendra へのリクエストを変更できます。

- `kendraQueryFilterString` - この文字列を使用して、Amazon Kendra リクエストの属性フィルタを指定します。インデックスで定義されたインデックスフィールドのいずれかを使用して、クエリをフィルタ処理できます。フィルター文字列の構造については、「Amazon Kendra Developer Guide」(Amazon Kendra デベロッパーガイド) の「Using document attributes to filter queries」(ドキュメント属性を使用してクエリをフィルタリングする) を参照してください。指定したフィルタ文字列が有効でないと、`InvalidLambdaResponseException` 例外が発生します。`kendraQueryFilterString` 文字列は、インテント用に設定された `queryFilterString` で指定されたクエリ文字列を上書きします。
- `kendraQueryRequestPayload` - この文字列を使用して、Amazon Kendra クエリを指定します。クエリでは、Amazon Kendra の任意の機能を使用できます。有効なクエリを指定しないと、`InvalidLambdaResponseException` 例外が発生します。これらの制限の詳細については、「Amazon Kendra Developer Guide」(Amazon Kendra デベロッパーガイド) の「Query」(クエリ) を参照してください。

フィルターまたはクエリ文字列を作成したら、レスポンスの `dialogAction` フィールドを `delegate` に設定してAmazon Lex にレスポンスを送信します。Amazon Lex は、クエリを Amazon Kendra に送信し、クエリレスポンスをフルフィルメントコードフックに返します。

レスポンスでのフルフィルメントコードフックの使用

Amazon Lex がクエリを Amazon Kendra に送信すると、クエリレスポンスが `AMAZON.KendraSearchIntent` フルフィルメント Lambda 関数に返されます。コードフックへの入力イベントには、Amazon Kendra からの完全なレスポンスが含まれます。クエリデータは、Amazon Kendra `Query` オペレーションによって返されるものと同じ構造になります。詳細については、[Amazon Kendra デベロッパーガイド] の [クエリレスポンス構文] を参照してください。

フルフィルメントコードフックはオプションです。フルフィルメントコードフックがない場合、またはレスポンスでメッセージを返さない場合、Amazon Lex はレスポンスに `conclusion` ステートメントを使用します。

例: Amazon Kendra インデックスを使用する FAQ ボットを作成する

この例では、Amazon Kendra インデックスを使用してユーザーの質問への回答を返す Amazon Lex ボットを作成します。FAQ ボットはユーザーのダイアログを管理します。`AMAZON.KendraSearchIntent` インテントを使用して、インデックスをクエリし、ユーザーにレスポンスを返します。ボットを作成するには、以下の操作を行います。

1. 顧客と対話して回答を返すボットを作成します。
2. カスタムインテントを作成します。ボットには、少なくとも 1 つのインテントと 1 つの発話が必要です。このインテントはボットのビルドに使用されるだけで、それ以外には使用されません。
3. ボットに `KendraSearchIntent` インテントを追加し、Amazon Kendra インデックスで使用されるように設定します。
4. ボットをテストするには、Amazon Kendra インデックス内のドキュメントに回答がある質問をします。

この例を使用する前に、Amazon Kendra インデックスを作成する必要があります。詳細については、「Amazon Kendra Developer Guide」(Amazon Kendra デベロッパーガイド) の「Getting started with an S3 bucket (console)」(S3 バケットの使用を開始する (コンソール)) を参照してください。

FAQ ボットを作成するには

1. AWS Management Console にサインインし、Amazon Lex コンソール (https://console.aws.amazon.com/lex/) を開きます。
2. ナビゲーションペインで、[ボット] を選択します。
3. [Create] (作成) を選択します。
4. [Custom bot] (カスタムボット) を選択します。以下のようにボットを設定します。
   - [Bot name] (ボット名) - ボットには、**`KendraTestBot`** などの目的を示す名前を付けます。
   - [Output voice] (音声出力) - [None] (なし) を選択します。
   - [Session timeout] (セッションタイムアウト) -「**5**」と入力します。
   - [Sentiment analysis] (センチメント分析) - [No] (いいえ) を選択します。
   - [COPPA] - [No] (いいえ) を選択します。

- [User utterance storage] (ユーザー発話の保存) - [Do not store] (保存しない) を選択します。

5. [Create] (作成) を選択します。

ボットを正常にビルドするには、1 つ以上のインテントと 1 つ以上のサンプル発話を作成する必要があります。このインテントは Amazon Lex ボットのビルドに必要ですが、FAQ のレスポンスには使用されません。このインテントの発話は、顧客が尋ねるどの質問にも適用されないようにしてください。

必要なインテントを作成するには

1. [ボットの開始方法] ページで、[インテントの作成] を選択します。
2. [インテントの追加] で、[インテントの作成] を選択します。
3. [インテントの作成] ダイアログボックスで、インテントに「**RequiredIntent**」などの名前を付けます。
4. [サンプル発話] に「**Required utterance**」などの発話を入力します。
5. [インテントの保存] を選択します。

次は、Amazon Kendra インデックスを検索するインテント、および返すレスポンスメッセージを作成します。

AMAZON.KendraSearchIntent インテントとレスポンスメッセージを作成するには

1. ナビゲーションペインで、[インテント] の横のプラス (+) を選択します。
2. [インテントの追加] で、[既存のインテントの検索] を選択します。
3. [Search intents] (インテントの検索) ボックスに **AMAZON.KendraSearchIntent** と入力し、リストからそのインテントを選択します。
4. [組み込みインテントのコピー] で、インテントに「**KendraSearchIntent**」などの名前を付け、[追加] を選択します。
5. インテントエディタで、[Amazon Kendra クエリ] を選択してクエリオプションを開きます。
6. [Amazon Kendra インデックス] メニューから、検索するインデックスを選択します。
7. [レスポンス] セクションで、以下の 3 つのメッセージを追加します。

```
I found a FAQ question for you: ((x-amz-lex:kendra-search-response-question_answer-question-1)) and the answer is ((x-amz-lex:kendra-search-response-question_answer-answer-1)).
```

```
I found an excerpt from a helpful document: ((x-amz-lex:kendra-search-response-
document-1)).
I think the answer to your questions is ((x-amz-lex:kendra-search-response-
answer-1)).
```

8. [インテントの保存] を選択してから、[ビルド] を選択してボットをビルドします。

最後に、コンソールテストウィンドウを使用して、ボットからのレスポンスをテストします。質問は、インデックスがサポートするドメインにあることが必要です。

FAQ ボットをテストするには

1. コンソールテストウィンドウで、インデックスに対する質問を入力します。
2. テストウィンドウのレスポンスセクションで、回答を確認します。
3. 別の質問のためにテストウィンドウをリセットするには、[チャットの履歴をクリア] を選択します。

AMAZON.PauseIntent

ユーザーがボットとの対話を一時停止して、後で再開できるようにするための単語やフレーズに応答します。Lambda関数やアプリケーションでは、セッション変数にインテントデータを保存したり、現在のインテントを再開する際に GetSession オペレーションでインテントデータを取得したりする必要があります。

一般的な発話:

- 一時停止
- 一時停止します

AMAZON.RepeatIntent

ユーザーが前のメッセージを繰り返すことができる単語やフレーズに応答します。お客様のアプリケーションでは、Lambda関数を使用して前回のインテント情報をセッション変数に保存するか、GetSession オペレーションを使用して前回のインテント情報を取得する必要があります。

一般的な発話:

- 繰り返し

- もう一回言う
- 繰り返す

AMAZON.ResumeIntent

単語や語句に応答して、ユーザーが以前一時停止したインテントを再開できるようにします。Lambda 関数またはアプリケーションは、前のインテントを再開するために必要な情報を管理する必要があります。

一般的な発話:

- 再開する
- 継続する
- 続ける

AMAZON.StartOverIntent

ユーザーが現在のインテントの処理を停止し、最初からやり直しを有効にする単語やフレーズに応答します。Lambda 関数または `PutSession` オペレーションを使用して、最初のスロット値を再び取得できます。

一般的な発話:

- 最初からやり直す
- 再起動
- 再開

AMAZON.StopIntent

ユーザーが現在のインテントの処理を停止し、ボットとの対話を終了したいことを示す単語やフレーズに応答します。Lambda 関数またはアプリケーションは、既存の属性とスロットタイプの値をクリアしてから、対話を終了する必要があります。

一般的な発話:

- stop
- オフ

- 黙って

組み込みスロットタイプ

Amazon Lex では、スロット内のデータの認識と処理方法を定義する組み込みスロットタイプがサポートされています。これらのスロットタイプはインテントで作成できます。これにより、よく使用されるスロットデータの列挙値 (日付、時刻、場所など) を作成する必要がなくなります。組み込みスロットタイプにはバージョンがありません。

スロットタイプ	短い説明	サポート対象ロケール	
AMAZON.Airport	空港を表す単語を認識します。	すべてのロケール	
AMAZON.AlphaNumeric	文字と数字で構成される単語を認識します。	韓国語 (ko-KR) を除くすべてのロケール	
AMAZON.City	都市を表す単語を認識します。	すべてのロケール	
AMAZON.Country	国を表す単語を認識します。	すべてのロケール	
AMAZON.DATE	日付を表す単語を認識し、標準形式に変換します。	すべてのロケール	
AMAZON.DURATION	継続時間を表す単語を認識し、標準形式に変換します。	すべてのロケール	
AMAZON.EmailAddress	E メールアドレスを表す単語を認識して、標準の E メールアドレスに変換	すべてのロケール	

スロットタイプ	短い説明	サポート対象ロケール	
AMAZON.FirstName	名を表す単語を認識します。	すべてのロケール	
AMAZON.LastName	名を表す単語を認識します。	すべてのロケール	
AMAZON.NUMBER	数語を認識し、数字に変換します。	すべてのロケール	
AMAZON.Percentage	パーセンテージを表す単語を認識して、数値とパーセント記号 (%) に変換	すべてのロケール	
AMAZON.PhoneNumber	電話番号を表す単語を認識して、数値の文字列に変換	すべてのロケール	
AMAZON.SpeedUnit	速度の単位を表す単語を認識して、標準の略語に変換	英語 (米国) (en-US)	
AMAZON.State	州を表す単語を認識します。	すべてのロケール	
AMAZON.StreetName	区画の名前を表す単語を認識します。	英語 (米国) (en-US) を除くすべてのロケール	
AMAZON.TIME	時間を示す単語を時間形式に変換します。	すべてのロケール	
AMAZON.WeightUnit	重量の単位を表す単語を認識して、標準の略語に変換	英語 (米国) (en-US)	

> **Note**
>
> 英語 (米国) (en-US) ロケールの場合、Amazon Lex はAlexaスキルキットのスロットタイプをサポートしています。使用可能な組み込みスロットタイプの一覧については、Alexa スキルキットのドキュメントの「スロットタイプリファレンス」を参照してください。
>
> - Amazon Lex では、`AMAZON.LITERAL` または `AMAZON.SearchQuery` 組み込みスロットタイプをサポートしていません。

AMAZON.Airport

空港の一覧を示します。その例を以下に示します。

- ジョン・F・ケネディ国際空港
- メルボルン空港

AMAZON.AlphaNumeric

文字と数字で構成される文字列 (**APQ123** など) を認識します。

このスロットタイプは、韓国語 (ko-KR) ロケールでは使用できません。

以下を含む文字列には、`AMAZON.AlphaNumeric` スロットタイプを使用できます。

- 英字 (**ABC** など)
- 数値 (**123** など)
- 英数字の組み合わせ (**ABC123** など)

スロットに入力された値を検証するために、`AMAZON.AlphaNumeric` スロットタイプに正規表現を追加できます。例えば、正規表現を使用して次のことを検証できます。

- イギリスまたはカナダの郵便番号
- 運転免許証番号
- 車両識別番号

標準の正規表現を使用します。Amazon Lex は、正規表現で以下の文字をサポートします。

- A ~ Z、a ~ z
- 0-9

Amazon Lex はまた、正規表現で Unicode 文字をサポートします。その形式は \u*Unicode* です。Unicode 文字を表すには、4 桁の数字を使用します。例えば、[\u0041-\u005A] は [A-Z] と同じです。

次の正規表現演算子はサポートされていません。

- 無限リピーター: *、+、または 上限のない {x,}
- ワイルドカード (.)

正規表現の最大長は 300 文字です。正規表現を使用する AMAZON.AlphaNumeric スロットタイプに保存される文字列の最大長は 30 文字です。

正規表現の例を次に示します。

- **APQ123** または **APQ1** などの英数字の文字列: `[A-Z]{3}[0-9]{1,3}` またはより制約がある `[A-DP-T]{3} [1-5]{1,3}`
- 米国国際プライオリティー郵便の形式 (**CP123456789US** など): `CP[0-9]{9}US`
- 銀行ルーティング番号 (**123456789** など): `[0-9]{9}`

スロットタイプの正規表現を設定するには、コンソールまたは PutSlotType オペレーションを使用します。スロットタイプを保存するときに、正規表現が検証されます。正規表現が有効でない場合、Amazon Lex はエラーメッセージを返します。

スロットタイプで正規表現を使用するときに、Amazon Lex はそのタイプのスロットへの入力を正規表現と照合します。入力が式と一致する場合、値はスロットに対して受け入れられます。入力が一致しない場合、Amazon Lex は入力を繰り返すようユーザーに要求します。

AMAZON.City

ローカルおよび世界の都市のリストを提供します。スロットタイプは、都市名の共通のバリエーションを認識します。Amazon Lex はバリエーションから正式名称に変換されません。

例:

- ニューヨーク

- レイキャビク
- 東京
- ヴェルサイユ

AMAZON.Country

世界の国の名前。例:

- オーストラリア
- ドイツ
- 日本
- アメリカ
- ウルグアイ

AMAZON.DATE

日付を表す単語を日付形式に変換します。

日付は ISO-8601 の日付形式でインテントに提供されます。スロットでインテントが受け取る日付は、ユーザーが発した特定のフレーズによって異なります。

- 「今日」、「今」、「11 月 25 日」など、特定の日付にマップされる発話は、完全な日付に変換されます `2020-11-25`。デフォルトでは、[現在または以降] の日付になります。
- 「今週」や「来週」など、特定の週に対応する発話は、その週の 1 日目の日付に変換されます。ISO-8601 形式では、週は月曜日に始まり、日曜日に終了します。例えば、今日が 2020-11-25 の場合、「来週」は `2020-11-30` に変換されます。
- 「来月」などの特定の日付ではなく、月にマップされる発話は、その月の最終日に変換されます。例えば、今日が2020-11-25 の場合、「来月」は `2020-12-31` に変換されます。
- 「来年」など、特定の月や日ではなく、年にマップされる発話は、翌年の最終日に変換されます。例えば、今日が2020-11-25 の場合、「来年」は `2021-12-31` に変換されます。

AMAZON.DURATION

期間を示す単語を数値の期間に変換します。

期間は、[ISO-8601 期間形式] に基づいた形式 PnYnMnWnDTnHnMnS に変換されます。P は期間であることを示し、n は数値で、次の大文字 n は、特定の日付または時刻の要素です。例えば、P3D は、3 日を意味します。T は、残りの値が日付要素ではなく時間要素を表していることを示すために使用されます。

例:

- 「10 分」: PT10M
- 「5 時間」: PT5H
- 「3 日間」: P3D
- 「45 秒」 : PT45S
- 「8 週間」: P8W
- 「7 年間」: P7Y
- 「5 時間 10 分」: PT5H10M
- 「2 年 3 時間 10 分」 : P2YT3H10M

AMAZON.EmailAddress

username@domain としての E メールアドレスを表す単語を認識します。アドレスのユーザー名には、特殊文字のアンダースコア (_)、ハイフン (-)、ピリオド (.)、プラス記号 (+) を含めることができます。

AMAZON.FirstName

よく使われる名前。このスロットタイプは、正式な名前と非公式のニックネームの両方を認識します。インテントに送信される名前は、ユーザーが送信した値です。Amazon Lex はニックネームから正式名に変換しません。

名前が似ているがスペルが異なる場合、Amazon Lex からインテントに共通フォームが送信されます。

英語 (米国) (en-US) ロケールのスロット名 AMAZON.US_First_Name を使用してください。

例:

- エミリー
- ジョン
- ソフィー

AMAZON.LastName

一般的に使用される姓。同じ名前のスペルが異なる場合、Amazon Lex からインテントに共通フォームが送信されます。

英語 (米国) (en-US) ロケールのスロット名 AMAZON.US_Last_Name を使用してください。

例:

- ブロスキー
- ダッシャー
- エバース
- パレス
- ウェルト

AMAZON.NUMBER

数値を表す単語や数値を少数を含む数値に変換します。次の表は、AMAZON.NUMBER スロットタイプでの数値語の変換方法を示しています。

入力	レスポンス
one hundred twenty three point four five	123.45
one hundred twenty three dot four five	123.45
point four two	0.42
point forty two	0.42
232.998	232.998
50	50

AMAZON.Percentage

パーセンテージを表す単語と記号を数値とパーセント記号 (%) に変換します。

パーセント記号や単語「percent」を使わずに数値を入力すると、スロット値は数値に設定されます。次の表は、AMAZON.Percentage スロットタイプでのパーセンテージの変換方法を示しています。

入力	レスポンス
50 percent	50%
0.4 パーセント	0.4%
23.5%	23.5%
25 パーセント	25%

AMAZON.PhoneNumber

電話番号を表す数値や単語を、次に示すように、区切り文字を使用しない文字列形式に変換します。

型	説明	入力	結果
先頭にプラス記号 (+) が付いた国際番号	先頭にプラス記号が付いた 11 桁の番号	+61 7 4445 1061 +1 (509) 555-1212	+61744431061 +15095551212
先頭にプラス記号 (+) がない国際番号	先頭にプラス記号がない 11 桁の番号	1 (509) 555-1212 61 7 4445 1061	15095551212 61744451061
国内番号	国番号がない 10 桁の番号	(03) 5115 4444 (509) 555-1212	0351154444 5095551212
ローカル番号	国番号や市外局番がない 7 桁の電話番号	555-1212	5551212

AMAZON.SpeedUnit

速度の単位を表す単語を、対応する略語に変換します。例えば、「miles per hour」は mph に変換されます。

スロットタイプは、英語 (英国) (en-GB) ロケールのみで使用できます。

次の例は、AMAZON.SpeedUnit スロットタイプでの速度単位の変換方法を示しています。

速度の単位	略語
miles per hour、mph、MPH、m/h	mph
kilometers per hour、km per hour、kmph、KMPH、km/h	kmph
meters per second、mps、MPS、m/s	mps
nautical miles per hour、knots、knot	knot

AMAZON.State

国内の地理的および政治的地域の名前。

例:

- バイエルン州
- 福島県
- 太平洋北西部
- クイーンズランド
- ウェールズ

AMAZON.StreetName

一般的な住所内の通りの名前。これには通りの名前だけが含まれ、番地は含まれません。

スロットタイプは、英語 (英国) (en-GB) ロケールのみで使用できます。

例:

- キャンベラアベニュー
- フロントストリート
- マーケットロード

AMAZON.TIME

時刻を表す単語を時刻値に変換します。あいまいな時間の解決を含みます。ユーザーがあいまいな時刻を入力すると、Amazon Lex は Lambda イベントの `slotDetails` 属性を使用して、あいまいな時刻の解決を Lambda 関数に渡します。例えば、ボットからユーザーに配達時間を尋ねたときにユーザーが「10 時」と答えると、この時刻はあいまいです。午前 10:00 なのか午後 10:00 なのかが不明です。この場合、`slots` マップの値は `null` となり、`slotDetails` エンティティに 2 つの可能な時刻の解決が含まれます。Amazon Lex は Lambda 関数に次のように入力します。

```
"slots": {
   "deliveryTime": null
},
"slotDetails": {
   "deliveryTime": {
      "resolutions": [
         {
            "value": "10:00"
         },
         {
            "value": "22:00"
         }
      ]
   }
}
```

ユーザーからあいまいではない時刻が返されると、Amazon Lex は Lambda イベントの `slots` 属性で時刻を Lambda 関数に送信し、`slotDetails` 属性は空になります。例えば、ユーザーが配達時間を求めるプロンプトに対して「10:00 PM」と返答すると、Amazon Lex は Lambda 関数に次のように入力します。

```
"slots": {
   "deliveryTime": "22:00"
}
```

Amazon Lex から Lambda 関数に送信されるデータの詳細については、「入力イベントの形式」を参照してください。

AMAZON.WeightUnit

重量の単位を表す単語を対応する略語に変換します。例えば、「kilogram」は kg に変換されます。

スロットタイプは、英語 (英国) (en-GB) ロケールのみで使用できます。

以下の例は、AMAZON.WeightUnit スロットタイプでの重量単位の変換方法を示しています。

重量の単位	略語
kilograms、kilos、kgs、KGS	kg
grams、gms、gm、GMS、g	g
milligrams、mg、mgs	mg
pounds、lbs、LBS	lbs
ounces、oz、OZ	oz
tonne、ton、t	t
kiloton、kt	kt

カスタムスロットタイプ

インテントごとに、ユーザーのリクエストの達成に必要な情報を示すパラメータを指定できます。これらのパラメータ (スロット) にはタイプがあります。スロットタイプは、スロット値を認識するように機械学習モデルをトレーニングするために Amazon Lex で使用する値のリストです。例えば、「Genres.」というスロットタイプを定義できます。このスロットタイプの各値は、ジャンル名を示す「comedy」、「adventure」、「documentary」などです。スロットタイプ値にはシノニムを定義できます。例えば、「comedy」値のシノニムとして「funny」や「humorous」を定義できます。

スロット値に解決される候補を制限するようにスロットタイプを設定できます。スロット値は列挙値として使用されます。ユーザーが入力した値は、スロット値のいずれかまたはシノニムと一致した場合にのみ、スロット値として解決されます。シノニムは、対応するスロット値に解決されます。例えば、ユーザーが「funny」と入力した場合、これはスロット値「comedy」に解決されます。

また、値を拡張するようにスロットタイプを設定することもできます。スロット値はトレーニングデータとして使用され、スロットはスロット値やシノニムに類似している場合に、ユーザーが指定した値に解決されます。これがデフォルトの動作です。

Amazon Lex には、スロットの解決の候補リストが保持されています。リスト内のエントリごとに、Amazon Lex でスロットの追加候補として認識された解決値があります。解決値は、スロット値と一致させるための最適な方法です。リストには最大 5 つの値が含まれます。

ユーザーが入力した値がシノニムである場合、解決値のリストの最初のエントリはスロットタイプ値になります。例えば、ユーザーが「funny」と入力すると、`slots` フィールドに「funny」が入り、`slotDetails` フィールドの最初のエントリが「comedy」になります。PutSlotType オペレーションを使用してスロットタイプを作成または更新するときに、スロット値が解決リストの最初の値になるように、`valueSelectionStrategy` を設定できます。

Lambda 関数を使用している場合は、関数への入力イベントに `slotDetails` という解決リストが含まれています。次の例は、Lambda 関数への入力のスロットとスロット詳細のセクションを示しています。

```
"slots": {
   "MovieGenre": "funny";
},
"slotDetails": {
   "Movie": {
      "resolutions": [
         "value": "comedy"
      ]
   }
}
```

スロットタイプごとに、最大 10,000 個の値とシノニムを定義できます。各ボットには、合計 50,000 個のスロットタイプ値とシノニムを含めることができます。例えば、5,000 の値と 5,000 のシノニムを含む 5 個のスロットタイプを使用するか、2,500 の値と 2,500 のシノニムを含む 10 個のスロットタイプを使用できます。これらの制限を超えると、PutBot オペレーションを呼び出したときに `LimitExceededException` が返されます。

スロットの難読化

Amazon Lex では、スロットの内容を難読化または非表示にして、コンテンツが表示されないようにすることができます。スロット値としてキャプチャされた機密データを保護するために、スロットの難読化を有効にして、会話ログのためにこれらの値をマスクできます。

スロット値を難読化することを選択した場合、Amazon Lex はスロット値を会話ログ内のスロットの名前に置き換えます。full_name と呼ばれるスロットの場合、スロットの値は次のように難読化されます。

```
Before obfuscation:
    My name is John Stiles
After obfuscation:
    My name is {full_name}
```

発話に括弧文字 ({}) が含まれている場合、Amazon Lex は括弧文字を 2 つのバックスラッシュ (\\) でエスケープします。例えば、テキスト {John Stiles} は次のように難読化されます。

```
Before obfuscation:
    My name is {John Stiles}
After obfuscation:
    My name is \\{{full_name}\\}
```

スロット値は会話ログで難読化されます。スロット値は、PostContent および PostText オペレーションからのレスポンスでも使用できます。スロット値は、検証およびフルフィルメント Lambda 関数で使用できます。プロンプトまたはレスポンスでスロット値を使用している場合、これらのスロット値は会話ログで難読化されません。

会話の最初のターンで、発話のスロットとスロットの値を認識すると、Amazon Lex はスロット値を難読化します。スロット値が認識されない場合、Amazon Lex は発話を難読化しません。

2 回目以降のターンでは、Amazon Lex は引き出すスロットを認識し、スロット値を難読化する必要があるかどうかを認識します。Amazon Lex がスロット値を認識すると、値は難読化されます。Amazon Lex が値を認識しない場合、発話全体が難読化されます。認識されなかった発話のスロット値は難読化されません。

Amazon Lex はまた、リクエスト属性またはセッション属性に保存するスロット値を難読化しません。難読化する必要があるスロット値を属性として保存する場合は、値を暗号化するか、難読化する必要があります。

Amazon Lex はオーディオのスロット値を難読化しません。これは、オーディオの書き起こしのスロット値を難読化します。

ボット内のすべてのスロットを難読化する必要はありません。どのスロットを難読化するかは、コンソールまたは Amazon Lex API を使用して選択できます。コンソールのスロットの設定で [Slot

obfuscation] (スロットの難読化) を選択します。API を使用している場合は、PutIntent オペレーションを呼び出すときに、スロットの obfuscationSetting フィールドを DEFAULT_OBFUSCATION に設定します。

センチメント分析

センチメント分析を使用して、ユーザー発話で表現されるセンチメントを判断できます。センチメント情報を使用して、会話フローを管理したり、コール後の分析を実行できます。例えば、ユーザーのセンチメントがネガティブな場合、会話をヒューマンエージェントに引き渡すフローを作成できます。

Amazon Lex は、Amazon Comprehend と統合して、ユーザーのセンチメントを検出します。Amazon Comprehend からの応答は、テキストの全体的なセンチメントがポジティブ、中立、ネガティブ、または混合のいずれであるかを示します。応答には、ユーザー発話における最も可能性の高いセンチメントおよびセンチメントカテゴリごとのスコアが含まれます。スコアはセンチメントが正しく検出された可能性を表します。

コンソールまたは Amazon Lex API を使用して、ボットのセンチメント分析を有効にします。Amazon Lex コンソールで、ボットの[設定] タブを選択し、[Sentiment Analysis] (センチメント分析) オプションを [はい] に設定します。API を使用している場合は、detectSentiment フィールドを true に設定して PutBot オペレーションを呼び出します。

センチメント分析を有効にしている場合、PostContent および PostText オペレーションは、ボットの応答で sentimentResponse という名前のフィールドを他のメタデータとともに返します。sentimentResponse フィールドには、センチメント分析の結果を含む SentimentLabel および SentimentScore の 2 つのフィールドがあります。Lambda 関数を使用している場合、sentimentResponse フィールドは関数に送信されるイベントデータに含まれます。

以下に、PostText または PostContent 応答の一部として返される sentimentResponse フィールドの例を示します。SentimentScore フィールドは、レスポンスのスコアを含む文字列です。

```
{
    "SentimentScore":
        "{
        Mixed: 0.030585512690246105,
        Positive: 0.94992071056365967,
        Neutral: 0.0141543131828308,
        Negative: 0.00893945890665054
        }",
```

```
    "SentimentLabel": "POSITIVE"
}
```

Amazon Lex は自動的に Amazon Comprehend を呼び出し、ボットによって処理されるすべての発話のセンチメントを判断します。センチメント分析を有効にすると、Amazon Comprehend のサービス利用規約および契約に同意したことになります。Amazon Comprehend の料金の詳細については、「Amazon Comprehend Pricing」(Amazon Comprehend の料金) を参照してください。

Amazon Comprehend のセンチメント分析の仕組みについては、「Amazon Comprehend Developer Guide」(Amazon Comprehend デベロッパーガイド) の「Determine the Sentiment」(センチメントを決定する) を参照してください。

Amazon Lex リソースのタグ付け

Amazon Lex ボット、ボットエイリアス、ボットチャンネルを管理しやすくするために、各リソースにメタデータをタグとして割り当てることができます。タグとは、AWS リソースに割り当てるラベルです。各タグは、キーと値から構成されます。

タグを使用すると、AWS リソースを目的、所有者、アプリケーションなどさまざまな方法で分類することができます。タグを使用すると、次のことができます。

- AWS リソースを識別および整理します。多くの AWS のリソースではタグ付けがサポートされるため、さまざまなサービスのリソースに同じタグを割り当てて、リソースの関連を示すことができます。例えば、ボットとボットが使用する Lambda 関数に同じタグを付けることができます。
- コストの割り当て。タグは、AWS Billing and Cost Management ダッシュボードでアクティベートします。AWS では、タグを使用してコストを分類し、毎月のコスト割り当てレポートを設定することができます。Amazon Lex では、$LATEST エイリアス以外のエイリアスに固有のタグを使用して、エイリアスごとにコストを割り当てることができます。Amazon Lex ボットのタグを使用して、$LATEST エイリアスのコストを割り当てます。詳細については、「AWS Billing and Cost Management ユーザーガイド」の「コスト配分タグの使用」(Use Cost Allocation Tags) を参照してください。
- リソースへのアクセスを制御します。Amazon Lex へのタグを使用して、Amazon Lex リソースへのアクセスを制御するポリシーを作成することができます。これらのポリシーを IAM ロールまたはユーザーにアタッチして、タグベースのアクセスコントロールを有効にできます。詳細については、「Amazon Lex での ABAC」を参照してください。リソースのタグに基づいてリソースへのアクセスを制限するためのアイデンティティベースのポリシーの例を表示するには、「タグを使用したリソースへのアクセス」を参照してください。

AWS Management Console、AWS Command Line Interface、および Amazon Lex API を使用してタグを操作できます。

リソースのタグ付け

Amazon Lex コンソールを使用している場合は、作成時にリソースにタグを付けることも、後でタグを追加することもできます。コンソールを使用して、既存のタグを更新または削除することもできます。

AWS CLI または Amazon Lex API を使用している場合は、以下のオペレーションを使用してリソースのタグを管理します。

- ListTagsForResource - リソースに関連付けられたタグを表示します。
- PutBot および PutBotAlias - ボットまたはボットエイリアスを作成するときにタグを適用します。
- TagResource - 既存のリソースにタグを追加および変更します。
- UntagResource - リソースからタグを削除します

Amazon Lex の以下のリソースがタグ付けをサポートしています。

- ボット - 次のような Amazon リソースネーム (ARN) を使用します。
  - `arn:${partition}:lex:${region}:${account}:bot:${bot-name}`
- ボットエイリアス - 次のような ARN を使用します。
  - `arn:${partition}:lex:${region}:${account}:bot:${bot-name}:${bot-alias}`
- ボットチャンネル - 次のような ARN を使用します。
  - `arn:${partition}:lex:${region}:${account}:bot-channel:${bot-name}:${bot-alias}:${channel-name}`

タグの制限

Amazon Lex リソースのタグには、以下の基本的な制限が適用されます。

- タグの最大数 - 50
- キーの最大長 - 128 文字
- 最大値の長さ - 256 文字
- キーと値の有効な文字 – a-z、A-Z、0-9、スペース、および特殊文字 (_ . : / = + - @)

- キーと値は大文字と小文字が区別されます。
- aws: をキーのプレフィックスとしてを使用しないでください。AWS 用に予約済みです。

リソースのタグ付け (コンソール)

コンソールを使用して、ボット、ボットエイリアス、またはボットチャンネルリソースのタグを管理できます。リソースの作成時にタグを追加することも、既存のリソースからタグを追加、変更、または削除することもできます。

ボットの作成時にタグを追加するには

1. AWS Management Console にサインインし、Amazon Lex コンソール (https://console.aws.amazon.com/lex/) を開きます。
2. [Create] を選択して、新しいボットを作成します。
3. [Create your bot] ページの下部にある [Tags] を選択します。
4. [Add tag] を選択し、ボットに 1 つ以上のタグを追加します。最大 50 個のタグを追加できます。

ボットエイリアスの作成時にタグを追加するには

1. AWS Management Console にサインインし、Amazon Lex コンソール (https://console.aws.amazon.com/lex/) を開きます。
2. ボットエイリアスを追加するボットを選択します。
3. [Settings] (設定) を選択します。
4. エイリアス名を追加し、ボットのバージョンを選択して、[Add tags] を選択します。
5. [Add tag] を選択し、ボットエイリアスに 1 つ以上のタグを追加します。最大 50 個のタグを追加できます。

ボットチャンネルの作成時にタグを追加するには

1. AWS Management Console にサインインし、Amazon Lex コンソール (https://console.aws.amazon.com/lex/) を開きます。
2. ボットチャンネルを追加するボットを選択します。
3. [Channels] を選択し、追加するチャンネルを選択します。
4. ボットチャンネルの詳細を追加し、[Tags] を選択します。

5. [Add tag] を選択し、ボットチャンネルに 1 つ以上のタグを追加します。最大 50 個のタグを追加できます。

ボットをインポートするときにタグを追加するには

1. AWS Management Console にサインインし、Amazon Lex コンソール (https://console.aws.amazon.com/lex/) を開きます。
2. [アクション] を選択してから、[インポート] を選択します。
3. ボットをインポートするための zip ファイルを選択します。
4. [Tags] を選択し、[Add tag] を選択して、ボットに 1 つ以上のタグを追加します。最大 50 個のタグを追加できます。

既存のボットのタグを追加、削除、または変更するには

1. AWS Management Console にサインインし、Amazon Lex コンソール (https://console.aws.amazon.com/lex/) を開きます。
2. 左側のメニューから [Bots] を選択し、変更するボットを選択します。
3. [Settings] を選択し、左側のメニューから [General] を選択します。
4. [Tags] を選択し、ボットのタグを追加、変更、または削除します。

ボットエイリアスのタグを追加、削除、または変更するには

1. AWS Management Console にサインインし、Amazon Lex コンソール (https://console.aws.amazon.com/lex/) を開きます。
2. 左側のメニューから [Bots] を選択し、変更するボットを選択します。
3. [Settings] を選択し、左側のメニューから [Aliases] を選択します。
4. 変更するエイリアスの [Manage tags] を選択し、ボットエイリアスのタグを追加、変更、または削除します。

既存のボットチャンネルでタグを追加、削除、または変更するには

1. AWS Management Console にサインインし、Amazon Lex コンソール (https://console.aws.amazon.com/lex/) を開きます。
2. 左側のメニューから [Bots] を選択し、変更するボットを選択します。

3. [Channels] を選択します。
4. [Tags] を選択し、ボットチャンネルのタグを追加、変更、または削除します。

リソースのタグ付け (AWS CLI)

AWS CLI を使用して、ボット、ボットエイリアス、またはボットチャンネルリソースのタグを管理できます。ボットまたはボットエイリアスの作成時にタグを追加したり、ボット、ボットエイリアス、ボットチャンネルからタグを追加、変更、または削除したりできます。

すべての例は、Linux および macOS 用にフォーマットされています。Windows でコマンドを使用するには、Linux の継続文字 (\) をキャレット (^) に置き換えます。

ボットの作成時にタグを追加するには

- 以下の省略形 put-bot AWS CLI コマンドは、ボットの作成時にタグを追加するために使用する必要があるパラメータを示しています。実際にボットを作成するには、他のパラメータを指定する必要があります。詳細については、「ステップ 4: ご利用開始にあたって (AWS CLI)」を参照してください。

```
aws lex-models put-bot \
    --tags '[{"key": "key1", "value": "value1"}, \
             {"key": "key2", "value": "value2"}]'
```

ボットエイリアスの作成時にタグを追加するには

- 以下の省略形 put-bot-alias AWS CLI コマンドは、ボットエイリアスの作成時にタグを追加するために使用する必要があるパラメータを示しています。実際にボットエイリアスを作成するには、他のパラメータを指定する必要があります。詳細については、「演習 5: エイリアスを作成する (AWS CLI)」を参照してください。

```
aws lex-models put-bot \
    --tags '[{"key": "key1", "value": "value1"}, \
             {"key": "key2", "value": "value2"}]"
```

リソースのタグを一覧表示するには

- `list-tags-for-resource` AWS CLI コマンドを使用して、ボット、ボットエイリアス、ボットチャンネルに関連付けられたリソースを表示します。

```
aws lex-models list-tags-for-resource \
    --resource-arn bot, bot alias, or bot channel ARN
```

リソースのタグを追加または変更するには

- `tag-resource` AWS CLI コマンドを使用して、ボット、ボットエイリアス、ボットチャンネルを追加または変更します。

```
aws lex-models tag-resource \
    --resource-arn bot, bot alias, or bot channel ARN \
    --tags '[{"key": "key1", "value": "value1"}, \
             {"key": "key2", "value": "value2"}]'
```

リソースからタグを削除するには

- `untag-resource` AWS CLI コマンドを使用して、ボット、ボットエイリアス、ボットチャンネルからタグを削除します。

```
aws lex-models untag-resource \
    --resource-arn bot, bot alias, or bot channel ARN \
    --tag-keys '["key1", "key2"]'
```

Amazon Lex の開始方法

Amazon Lex には、既存のアプリケーションと統合できる API オペレーションがあります。サポートされているオペレーションのリストについては、「API リファレンス」を参照してください。以下のいずれかのオプションを使用できます。

- AWS SDK - SDK を使用するとき、Amazon Lex へのリクエストは自動的に署名され、指定した認証情報を使用して認証されます。これは、アプリケーション構築に推奨される選択肢です。
- AWS CLI — を使用して AWS CLI 、コードを記述しなくても任意の Amazon Lex 機能にアクセスできます。
- AWS コンソール - コンソールは Amazon Lex をテストして使用を開始する最も簡単な方法です。

Amazon Lex を初めて使用する場合は、まず「Amazon Lex: 仕組み」を読むことをお勧めします。

トピック

ステップ 1: AWS アカウントを設定し、管理者ユーザーを作成する

Amazon Lex を初めて使用する場合は、事前に以下のタスクをすべて実行してください。

1. にサインアップする AWS
2. ユーザーの作成

にサインアップする AWS

AWS アカウントを既にお持ちの場合は、このタスクをスキップしてください。

Amazon Web Services (AWS) にサインアップすると AWS、Amazon Lex を含む のすべてのサービスに AWS アカウントが自動的にサインアップされます。料金は、使用するサービスの料金のみが請求されます。

Amazon Lex の場合、使用したリソースに対してのみ料金を支払います。 AWS の新規のお客様の場合、無料で Amazon Lex の使用を開始できます。詳細については、「AWS 無料利用枠」を参照してください。

AWS アカウントをすでにお持ちの場合は、次のタスクに進んでください。 AWS アカウントをお持ちでない場合は、以下の手順に従ってアカウントを作成してください。

AWS アカウントを作成するには

1. https://portal.aws.amazon.com/billing/signup を開きます。
2. オンラインの手順に従います。

   サインアップ手順の一環として、通話呼び出しを受け取り、電話キーパッドで検証コードを入力するように求められます。

   にサインアップすると AWS アカウント、 AWS アカウントのルートユーザーが作成されます。ルートユーザーには、アカウントのすべての AWS のサービス とリソースへのアクセス権があります。セキュリティのベストプラクティスとして、ユーザーに管理アクセスを割り当て、ルートユーザーのみを使用してルートユーザーアクセスが必要なタスクを実行してください。

次のタスクで必要になるため、 AWS アカウント ID を書き留めます。

ユーザーの作成

Amazon Lex などの のサービスでは AWS、アクセス時に認証情報を指定する必要があります。これにより、サービスが所有するリソースにアクセスするためのアクセス許可があるかどうかをサービスが判断できるようになります。コンソールを使用するにはパスワードが必要です。ただし、 AWS アカウントの認証情報 AWS を使用して にアクセスすることはお勧めしません。代わりに、以下をお勧めします。

- AWS Identity and Access Management (IAM) を使用してユーザーを作成する
- 管理権限を持つ IAM グループにユーザーを追加する
- 作成した ユーザーに管理権限を付与します。

その後、特別な URL とユーザーの認証情報 AWS を使用して にアクセスできます。

このガイドの「使用開始」実習では、管理者権限を持つユーザー (adminuser) が存在すること想定しています。手順に従ってアカウントに adminuser を作成します。

管理者ユーザーを作成し、コンソールにサインインするには

1. AWS アカウントに adminuser という管理者ユーザーを作成します。手順については、「IAM ユーザーガイド」の「最初のユーザーと管理者グループの作成」を参照してください。
2. ユーザーとして、特別な URL AWS Management Console を使用して にサインインできます。詳細については、「IAM ユーザーガイド」の「ユーザーがアカウントにサインインする方法」を参照してください。

IAM の詳細については、以下を参照してください。

- AWS Identity and Access Management (IAM)
- IAM の使用開始
- IAM ユーザーガイド

次のステップ

ステップ 2: を設定する AWS Command Line Interface

ステップ 2: を設定する AWS Command Line Interface

AWS Command Line Interface （ AWS CLI) で Amazon Lex を使用する場合は、ダウンロードして設定します。

> **Important**
>
> 入門演習のステップを実行する AWS CLI ために は必要ありません。ただし、このガイドの後半の演習の一部では AWS CLIを使用します。コンソールを使用して開始する場合は、このステップをスキップして「ステップ 3: 開始方法 (コンソール)」に進んでください。後で、が必要な場合は AWS CLI、こちらに戻ってセットアップします。

を設定するには AWS CLI

1. AWS CLIをダウンロードして設定します。手順については、AWS Command Line Interface ユーザーガイドの次のトピックを参照してください。
   - のセットアップ AWS Command Line Interface

- AWS Command Line Interfaceの設定

2. 管理者ユーザーの名前付きプロファイルを設定 AWS CLI ファイルの末尾に追加します。このプロファイルは AWS CLI 、コマンドを実行するときに使用します。名前付きプロファイルの詳細については、AWS Command Line Interface ユーザーガイドの「名前付きプロファイル」を参照してください。

```
[profile adminuser]
aws_access_key_id = adminuser access key ID
aws_secret_access_key = adminuser secret access key
region = aws-region
```

利用可能な AWS リージョンのリストについては、「」の「リージョンとエンドポイント」を参照してくださいAmazon Web Services 全般のリファレンス。

3. コマンドプロンプトに Help コマンドを入力して、セットアップを検証します。

```
aws help
```

ステップ 3: 開始方法 (コンソール)

ステップ 3: 開始方法 (コンソール)

Amazon Lex の使用開始方法を学ぶ最も簡単な方法は、コンソールを使用することです。開始するために、以下の作成済みの演習を利用できます。すべての演習でコンソールを使用します。

- 演習 1 - 設計図を使用して Amazon Lex ボットを作成します。設計図は、すべての必要なボット設定を提供する事前定義されたボットです。 end-to-end セットアップのテストには最小限の作業しか行いません。

  さらに、 が提供する Lambda 関数の設計図を使用して AWS Lambda、Lambda 関数を作成します。関数は、ボットと互換性のある事前定義済みのコードを使用するコードフックです。
- 演習 2 - ボットを手動で作成して設定し、カスタムボットを作成します。また、コードフックとして Lambda 関数を作成します。サンプルコードが用意されています。
- 演習 3 - ボットを発行し、このボットの新しいバージョンを作成します。この演習では、ボットのバージョンを参照するエイリアスを作成します。

トピック

演習 1: 設計図を使用して Amazon Lex ボットを作成する (コンソール)

この演習では、以下のことを行います。

- 最初の Amazon Lex ボットを作成して Amazon Lex コンソールでテストします。

  この演習では OrderFlowers 設計図を使用します。設計図については、「Amazon Lex および AWS Lambda の設計図」を参照してください。

- AWS Lambda 関数を作成して、Lambda コンソールでテストします。リクエストの処理中に、この Lambda 関数をボットから呼び出します。この演習では、AWS Lambda コンソールに用意されている Lambda の設計図 (lex-order-flowers-python) を使用して Lambda 関数を作成します。設計図コードは、同じ Lambda 関数を使用して初期化と検証を行う方法、および `OrderFlowers` インテントを達成する方法を示しています。

- ボットを更新し、インテントを達成するためのコードフックとして Lambda 関数を追加します。エンドツーエンドエクスペリエンスをテストします。

以下のセクションでは設計図の機能について説明します。

Amazon Lex ボット: 設計図の概要

OrderFlowers 設計図を使用して Amazon Lex ボットを作成します。ボットの構造の詳細については、「Amazon Lex: 仕組み」を参照してください。このボットは次のように事前設定されています。

- インテント – OrderFlowers
- スロットタイプ – 1 つのカスタムスロットタイプ (`FlowerTypes`) と列挙値 (`roses`、`lilies`、`tulips`)。

- スロット – ボットでインテントを達成する前に、インテントには以下の情報 (つまり、スロット) が必要です。
  - `PickupTime` (AMAZON.TIME 組み込みタイプ)
  - `FlowerType` (FlowerTypes カスタムタイプ)
  - `PickupDate` (AMAZON.DATE 組み込みタイプ)
- 発話 – 以下のサンプル発話はユーザーのインテントを示しています。
  - 「花をピックアップしたい」
  - 「花を注文したい」
- プロンプト – ボットは、インテントを識別した後で、以下のプロンプトを使用してスロットを満たします。
  - `FlowerType` スロットのプロンプト –「どの花を注文なさいますか?」
  - `PickupDate` スロットのプロンプト –「何日に {FlowerType} をピックアップなさいますか?」
  - `PickupTime` スロットのプロンプト –「何時に {FlowerType} をピックアップなさいますか?」
  - 確認ステートメント –「了解いたしました。お客様の {FlowerType} は {PickupDate} の {PickupTime} までにご用意させていただきます。それでよろしいでしょうか?」

AWS Lambda 関数: 設計図の概要

この演習の Lambda 関数では、初期化と検証、およびフルフィルメントの両方のタスクを実行します。したがって、Lambda 関数を作成した後に、初期化/検証とフルフィルメントタスクの両方を処理するコードフックとして同じ Lambda 関数を指定することで、インテント設定を更新します。

- 初期化および検証のコードフックとして、Lambda 関数は基本的な検証を実行します。例えば、ユーザーが指定したピックアップの時間が通常の営業時間外である場合、Lambda 関数は時間を指定し直すことをユーザーに再度プロンプトするように Amazon Lex に指示します。
- フルフィルメントコードフックの一環として、Lambda 関数は花の注文が確定した (つまり、インテントが達成された) ことを示す概要メッセージを返します。

次のステップ

[ステップ 1: Amazon Lex ボット (コンソール) を作成する](#)

ステップ 1: Amazon Lex ボット (コンソール) を作成する

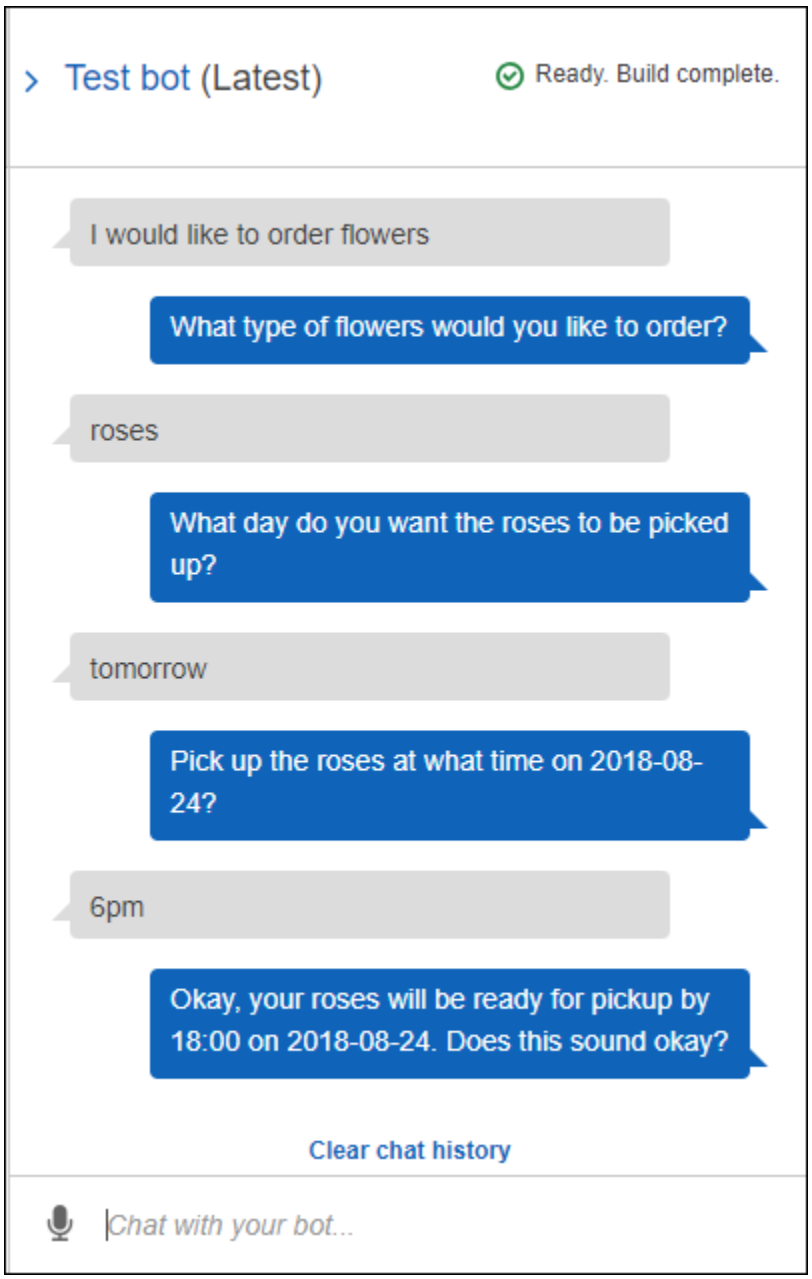
この演習では、花を注文するボットとして OrderFlowersBot を作成します。

Amazon Lex ボット (コンソール) を作成するには

1. AWS Management Console にサインインし、Amazon Lex コンソール (https://console.aws.amazon.com/lex/) を開きます。
2. 初めてのボットの場合には [Get Started] (ご利用開始にあたって)、それ以外の場合は [ボット] ページを選択し、続いて [作成] を選択します。
3. [Create your Lex bot] ページで、以下の情報を指定して [Create] を選択します。
   - [OrderFlowers] 設計図を選択します。
   - ボット名 (OrderFlowers) はデフォルトのままにしておきます。
   - [COPPA] で [**No**] を選択します。
   - ユーザーの発話ストレージで、適切なレスポンスを選択します。
4. [Create] (作成) を選択します。コンソールは Amazon Lex に対して必要なリクエストを行い、設定を保存します。次に、コンソールはボットエディタウィンドウを表示します。
5. ボットが作成されたことが確認されるまで待機します。
6. ボットをテストします。

   > **Note**
   >
   > ボットをテストするには、テストウィンドウにテキストを入力するか、テストウィンドウにあるマイクボタンを選択して話しかけます (対応しているブラウザの場合)。

   次のサンプルテキストを使用してボットと会話し、花を注文します。



この入力により、ボットは `OrderFlowers` インテントを推測し、スロットデータのプロンプトを表示します。すべての必要なスロットデータを提供すると、ボットはすべての情報をクライアントアプリケーション (この例ではコンソール) に返すことで、インテント (`OrderFlowers`) を達成します。コンソールは、テストウィンドウに情報を表示します。

具体的には次のとおりです。

- 「何日にバラをピックアップなさいますか?」の文で「バラ」という単語が使用されているのは、`pickupDate` スロットのプロンプトが `{FlowerType}` を置き換えるように設定されているためです。これはコンソールで確認できます。

- 「了解いたしました。お客様のバラは ...」という文は、確認プロンプトとして設定したものです。
- 最後の文 (「`FlowerType:roses...`」) は、クライアント (この例ではテストウィンドウ) に単純に返されるスロットデータです。次の演習では、Lambda 関数を使用してインテントを達成します。インテントが達成されると、注文が確定されたことを示すメッセージが表示されます。

次のステップ

ステップ 2 (オプション): 情報フローの詳細を確認する (コンソール)

ステップ 2 (オプション): 情報フローの詳細を確認する (コンソール)

このセクションでは、クライアントと Amazon Lex 間の情報フローについて、サンプル会話のユーザー入力別に説明します。

この例では、ボットとの会話にコンソールテストウィンドウを使用します。

Amazon Lex のテストウィンドウを開くには

1. AWS Management Console にサインインし、Amazon Lex コンソール (https://console.aws.amazon.com/lex/) を開きます。
2. テストするボットを選択します。
3. コンソールの右側で、[Test chatbot] (chatbot のテスト) を選択します。

次のいずれかのトピックを選択し、音声または入力による情報フローを確認してください。

トピック

- ステップ 2a (オプション): 音声による情報フローの詳細を確認する (コンソール)
- ステップ 2b (オプション): 入力による情報フローの詳細を確認する (コンソール)

ステップ 2a (オプション): 音声による情報フローの詳細を確認する (コンソール)

このセクションでは、クライアントが音声を使用してリクエストを送信した場合の、クライアントと Amazon Lex 間の情報フローについて説明します。詳細については、「PostContent」を参照してください。

1. ユーザーの発話: 花を注文したい。

a. クライアント (コンソール) は以下の PostContent リクエストを Amazon Lex に送信します。

```
POST /bot/OrderFlowers/alias/$LATEST/user/4o9wwdhx6nlheferh6a73fujd3118f5w/
content HTTP/1.1
x-amz-lex-session-attributes: "e30="
Content-Type: "audio/x-l16; sample-rate=16000; channel-count=1"
Accept: "audio/mpeg"


Request body
input stream
```

リクエストの URI と本文の両方で Amazon Lex に情報が提供されています。

- リクエスト URI ボット名 (OrderFlowers)、ボットエイリアス ($LATEST)、ユーザー名 (ユーザーを識別するランダムな文字列)。content はこれが PostContent API リクエストである (PostText リクエストではない) と示します。
- リクエストヘッダー
  - x-amz-lex-session-attributes – 「{}」を表す base64 エンコード値。クライアントが最初のリクエストを行うときにはセッション属性はありません。
  - Content-Type – オーディオ形式が反映されています。
- リクエストボディ – ユーザー入力のオーディオストリーム (「花を注文したい」)。

> **Note**
>
> 話すのではなく PostContent API にテキスト (「花を注文したい」) を送信することをユーザーが選択した場合、リクエストボディはユーザー入力です。それに応じて Content-Type ヘッダーが設定されています。
>
> ```
> POST /bot/OrderFlowers/alias/$LATEST/
> user/4o9wwdhx6nlheferh6a73fujd3118f5w/content HTTP/1.1
> x-amz-lex-session-attributes: "e30="
> Content-Type: "text/plain; charset=utf-8"
> Accept: accept
>
> Request body
> ```

```
input stream
```

b. 入力ストリームから、Amazon Lex はインテント (OrderFlowers) を検出します。次に、そのインテントのスロットのいずれか (この場合は FlowerType)、およびその値を引き出すプロンプトの 1 つ を選択し、以下のヘッダーを付けてレスポンスを送信します。

```
x-amz-lex-dialog-state:ElicitSlot
x-amz-lex-input-transcript:I would like to order some flowers.
x-amz-lex-intent-name:OrderFlowers
x-amz-lex-message:What type of flowers would you like to order?
x-amz-lex-session-attributes:e30=
x-amz-lex-slot-to-elicit:FlowerType
x-amz-lex-
slots:eyJQaWNrdXBUaW1lIjpudWxsLCJGbG93ZXJUeXBlIjpudWxsLCJQaWNrdXBEYXRlIjpudWxsfQ==
```

ヘッダー値では以下の情報が指定されています。

- x-amz-lex-input-transcript – リクエスト内の音声 (ユーザー入力) のトランスクリプト
- x-amz-lex-message – Amazon Lex がレスポンスで返した音声のトランスクリプト
- x-amz-lex-slots – スロットと値のペアを表す次の文字列を base64 でエンコードした値

```
{"PickupTime":null,"FlowerType":null,"PickupDate":null}
```

- x-amz-lex-session-attributes – セッション属性を表す文字列 ({}) を base64 でエンコードした値

クライアントはレスポンス本文内の音声を再生します。

2. ユーザーの発声: バラ

a. クライアント (コンソール) は以下の PostContent リクエストを Amazon Lex に送信します。

```
POST /bot/OrderFlowers/alias/$LATEST/user/4o9wwdhx6nlheferh6a73fujd3118f5w/
content HTTP/1.1
x-amz-lex-session-attributes: "e30="
Content-Type: "audio/x-l16; sample-rate=16000; channel-count=1"
```

```
Accept: "audio/mpeg"


Request body
input stream ("roses")
```

リクエストボディはユーザー入力の音声ストリーム (「バラ」) です。sessionAttributes は空のままです。

b. Amazon Lex は現在のインテントのコンテキストで入力ストリームを解釈します (FlowerType スロットに関する情報をこのユーザーに求めていたことを覚えています)。Amazon Lex はまず現在のインテントのスロット値を更新します。次に、別のスロット (PickupDate) をそのプロンプトメッセージの 1 つ (いつバラをピックアップなさいますか?) と共に選択し、以下のヘッダーを付けてレスポンスを返します。

```
x-amz-lex-dialog-state:ElicitSlot
x-amz-lex-input-transcript:roses
x-amz-lex-intent-name:OrderFlowers
x-amz-lex-message:When do you want to pick up the roses?
x-amz-lex-session-attributes:e30=
x-amz-lex-slot-to-elicit:PickupDate
x-amz-lex-
slots:eyJQaWNrdXBUaW1lIjpudWxsLCJGbG93ZXJUeXBlIjoicm9zaSdzIiwiUGlja3VwRGF0ZSI6bnVsbH0=
```

ヘッダー値では以下の情報が指定されています。

- x-amz-lex-slots – スロットと値のペアを表す次の文字列を base64 でエンコードした値

  ```
  {"PickupTime":null,"FlowerType":"roses","PickupDate":null}
  ```

- x-amz-lex-session-attributes – セッション属性を表す文字列 ({}) を base64 でエンコードした値

クライアントはレスポンス本文内の音声を再生します。

3. ユーザーの発声: 明日

a. クライアント (コンソール) は以下の PostContent リクエストを Amazon Lex に送信します。

```
POST /bot/OrderFlowers/alias/$LATEST/user/4o9wwdhx6nlheferh6a73fujd3118f5w/
content HTTP/1.1
x-amz-lex-session-attributes: "e30="
Content-Type: "audio/x-l16; sample-rate=16000; channel-count=1"
Accept: "audio/mpeg"


Request body
input stream ("tomorrow")
```

リクエストボディはユーザー入力の音声ストリーム (「明日」) です。sessionAttributes は空のままです。

b. Amazon Lex は現在のインテントのコンテキストで入力ストリームを解釈します (PickupDate スロットに関する情報をこのユーザーに求めていたことを覚えています)。Amazon Lex は現在のインテントのスロット (PickupDate) の値を更新します。次に、値を引き出す別のスロット (PickupTime)、およびその値を引き出すいずれかのプロンプト (「2017 年 3 月 18 日の何時にバラをピックアップなさいますか」) を選択し、以下のヘッダーを付けてレスポンスを返します。

```
x-amz-lex-dialog-state:ElicitSlot
x-amz-lex-input-transcript:tomorrow
x-amz-lex-intent-name:OrderFlowers
x-amz-lex-message:When do you want to pick up the roses on 2017-03-18?
x-amz-lex-session-attributes:e30=
x-amz-lex-slot-to-elicit:PickupTime
x-amz-lex-
slots:eyJQaWNrdXBUaW1lIjpudWxsLCJGbG93ZXJUeXBlIjoicm9zaSdzIiwiUGlja3VwRGF0ZSI6IjIwMTctM
x-amzn-RequestId:3a205b70-0b69-11e7-b447-eb69face3e6f
```

ヘッダー値では以下の情報が指定されています。

- x-amz-lex-slots – スロットと値のペアを表す次の文字列を base64 でエンコードした値

  ```
  {"PickupTime":null,"FlowerType":"roses","PickupDate":"2017-03-18"}
  ```

- x-amz-lex-session-attributes – セッション属性を表す文字列 ({}) を base64 でエンコードした値

クライアントはレスポンス本文内の音声を再生します。

4. ユーザーの発声: 午後 6 時

   a. クライアント (コンソール) は以下の PostContent リクエストを Amazon Lex に送信します。

```
POST /bot/OrderFlowers/alias/$LATEST/user/4o9wwdhx6nlheferh6a73fujd3118f5w/
content HTTP/1.1
x-amz-lex-session-attributes: "e30="
Content-Type: "text/plain; charset=utf-8"
Accept: "audio/mpeg"


Request body
input stream ("6 pm")
```

   リクエストボディはユーザー入力の音声ストリーム (「午後 6 時」) です。sessionAttributes は空のままです。

   b. Amazon Lex は現在のインテントのコンテキストで入力ストリームを解釈します (PickupTime スロットに関する情報をこのユーザーに求めていたことを覚えています)。まず現在のインテントのスロット値を更新します。

   ここで、Amazon Lex はすべてのスロットのデータがそろっていることを検出します。ただし、OrderFlowers インテントには確認メッセージが設定されています。そのため、Amazon Lex はインテントの達成に進む前に、ユーザーからの明示的な確認を必要とします。花を注文する前に確認を要求する以下のヘッダーを付けてレスポンスを送信します。

```
x-amz-lex-dialog-state:ConfirmIntent
x-amz-lex-input-transcript:six p. m.
x-amz-lex-intent-name:OrderFlowers
x-amz-lex-message:Okay, your roses will be ready for pickup by 18:00 on
 2017-03-18.  Does this sound okay?
x-amz-lex-session-attributes:e30=
x-amz-lex-
slots:eyJQaWNrdXBUaW1lIjoiMTg6MDAiLCJGbG93ZXJUeXBlIjoicm9zaSdzIiwiUGlja3VwRGF0ZSI6IjIwM
x-amzn-RequestId:083ca360-0b6a-11e7-b447-eb69face3e6f
```

   ヘッダー値では以下の情報が指定されています。

- x-amz-lex-slots – スロットと値のペアを表す次の文字列を base64 でエンコードした値

```
{"PickupTime":"18:00","FlowerType":"roses","PickupDate":"2017-03-18"}
```

- x-amz-lex-session-attributes – セッション属性を表す文字列 ({}) を base64 でエンコードした値

クライアントはレスポンス本文内の音声を再生します。

5. ユーザーの発声: はい

a. クライアント (コンソール) は以下の PostContent リクエストを Amazon Lex に送信します。

```
POST /bot/OrderFlowers/alias/$LATEST/user/4o9wwdhx6nlheferh6a73fujd3118f5w/
content HTTP/1.1
x-amz-lex-session-attributes: "e30="
Content-Type: "audio/x-l16; sample-rate=16000; channel-count=1"
Accept: "audio/mpeg"


Request body
input stream ("Yes")
```

リクエストボディはユーザー入力の音声ストリーム (「はい」) です。sessionAttributes は空のままです。

b. Amazon Lex は入力ストリームを解釈し、注文を進めることをユーザーが望んでいることを理解します。OrderFlowers インテントには、フルフィルメントアクティビティとして ReturnIntent が設定されています。これにより、Amazon Lex からすべてのインテントデータがクライアントに返されます。Amazon Lex は以下の内容でレスポンスを返します。

```
x-amz-lex-dialog-state:ReadyForFulfillment
x-amz-lex-input-transcript:yes
x-amz-lex-intent-name:OrderFlowers
x-amz-lex-session-attributes:e30=
x-amz-lex-
slots:eyJQaWNrdXBUaW1lIjoiMTg6MDAiLCJGbG93ZXJUeXBlIjoicm9zaSdzIiwiUGlja3VwRGF0ZSI6IjIwM
```

`x-amz-lex-dialog-state` レスポンスヘッダーは `ReadyForFulfillment` に設定されています。これで、クライアントはインテントを達成できます。

6. ここで、ボットを再テストします。新しい (ユーザー) コンテキストを確立するには、コンソールで [Clear] リンクを選択します。`OrderFlowers` インテントに指定するデータに、以下のような無効なデータを含めます。例:

   - 花の種類として「Jasmine」 (サポートされている花の種類ではない)
   - 花をピックアップする日付として「昨日」

   ボットではこれらの値が受け付けられることがわかります。これは、ユーザーデータを初期化および検証するコードがないためです。次のセクションでは、その処理を行う Lambda 関数を追加します。Lambda 関数について、以下の点に注意してください。

   - ユーザー入力のたびにスロットデータを検証します。インテントは最後に達成されます。つまり、ボットはスロットデータをクライアントに返すだけではなく、花の注文を処理してユーザーにメッセージを返します。詳細については、「Lambda 関数を使用する」を参照してください。
   - セッション属性も設定します。セッション属性の詳細については、「PostText」を参照してください。

     ご利用開始のセクションを完了したら、その他の演習 (その他の例: Amazon Lex ボットの作成) を行うことができます。旅行を予約する は、セッション属性を使用してクロスインテント情報を共有し、ユーザーと動的に会話します。

次のステップ

ステップ 3: Lambda 関数を作成する (コンソール)

ステップ 2b (オプション): 入力による情報フローの詳細を確認する (コンソール)

このセクションでは、クライアントが `PostText` API を使用してリクエストを送信する場合の、クライアントと Amazon Lex の間の情報のフローについて説明します。詳細については、「PostText」を参照してください。

1. ユーザーの入力: 「花を注文したい」

   a. クライアント (コンソール) は以下の PostText リクエストを Amazon Lex に送信します。

```
POST /bot/OrderFlowers/alias/$LATEST/user/4o9wwdhx6nlheferh6a73fujd3118f5w/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
    "inputText": "I would like to order some flowers",
    "sessionAttributes": {}
}
```

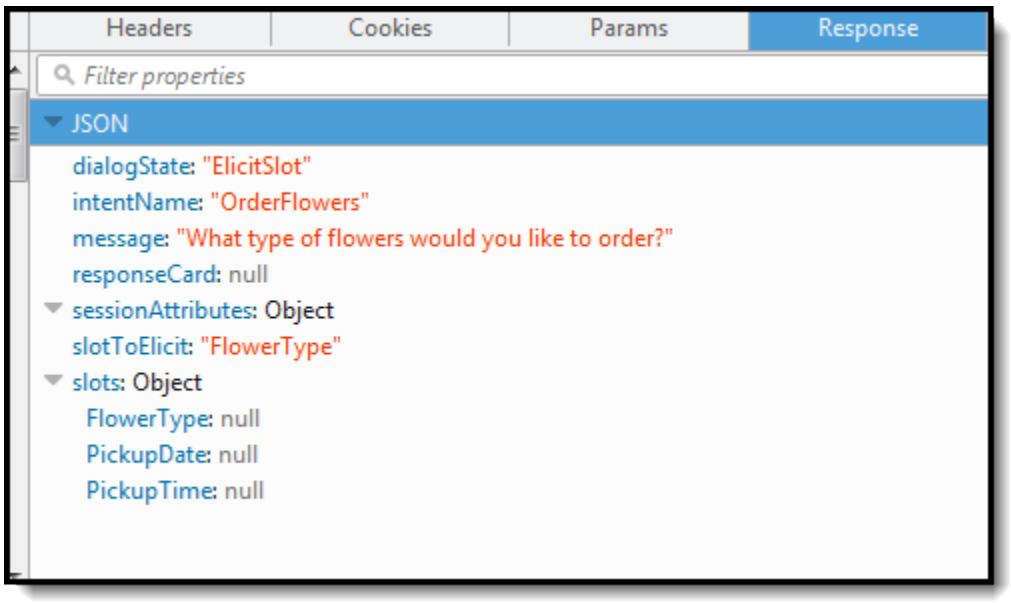
リクエストの URI と本文の両方で Amazon Lex に情報が提供されています。

- リクエスト URI – ボット名 (OrderFlowers)、ボットのエイリアス ($LATEST)、およびユーザー名 (ユーザーを識別するランダムな文字列) を提供します。末尾の text では、これが PostText API リクエストである (PostContent ではない) ことが示されています。

- リクエストボディ – ユーザー入力 (inputText) と空の sessionAttributes が含まれています。クライアントが最初のリクエストを行うときにはセッション属性はありません。Lambda 関数は後でセッション属性を使用します。

b. inputText から、Amazon Lex はインテント (OrderFlowers) を検出します。このインテントには、ユーザー入力の初期化と検証、あるいはフルフィルメントを行うためのコードフック (Lambda 関数) がありません。

   Amazon Lex は、値を引き出すインテントのスロットのいずれか (この場合は FlowerType) を選択します。また、そのスロットの値を引き出すプロンプトの 1 つ (すべてインテント設定の一部) を選択し、以下のレスポンスをクライアントに返します。コンソールに、ユーザーへのレスポンスのメッセージが表示されます。



クライアントはレスポンス内のメッセージを表示しています。

2. ユーザーの入力: 「バラ」

   a. クライアント (コンソール) は以下の PostText リクエストを Amazon Lex に送信します。

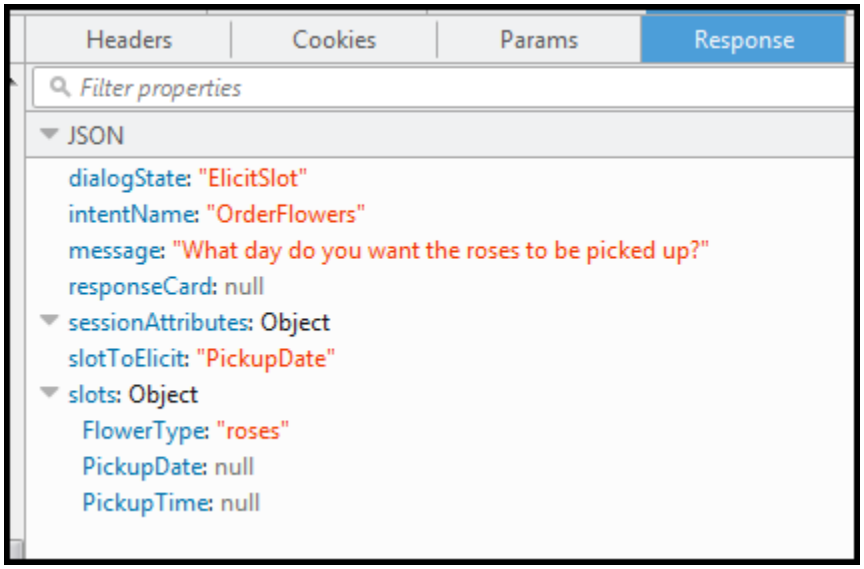
```
POST /bot/OrderFlowers/alias/$LATEST/user/4o9wwdhx6nlheferh6a73fujd3118f5w/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
    "inputText": "roses",
    "sessionAttributes": {}
}
```

   リクエストボディの `inputText` は、ユーザー入力を示します。`sessionAttributes` は空のままです。

   b. Amazon Lex はまず、現在のインテントのコンテキストの `inputText` を解釈します (このサービスでは `FlowerType` スロットに関する情報を特定のユーザーに求めていたことが記憶されています)。Amazon Lex は現在のインテントのスロット値を更新し、別のスロット (`PickupDate`) をそのスロットのプロンプトメッセージの 1 つ (「何日にバラをピックアップなさいますか?」) と共に選択します。

   次に、Amazon Lex は以下のレスポンスを返します。



クライアントはレスポンス内のメッセージを表示しています。

3. ユーザーの入力: 「明日」

   a. クライアント (コンソール) は以下の PostText リクエストを Amazon Lex に送信します。

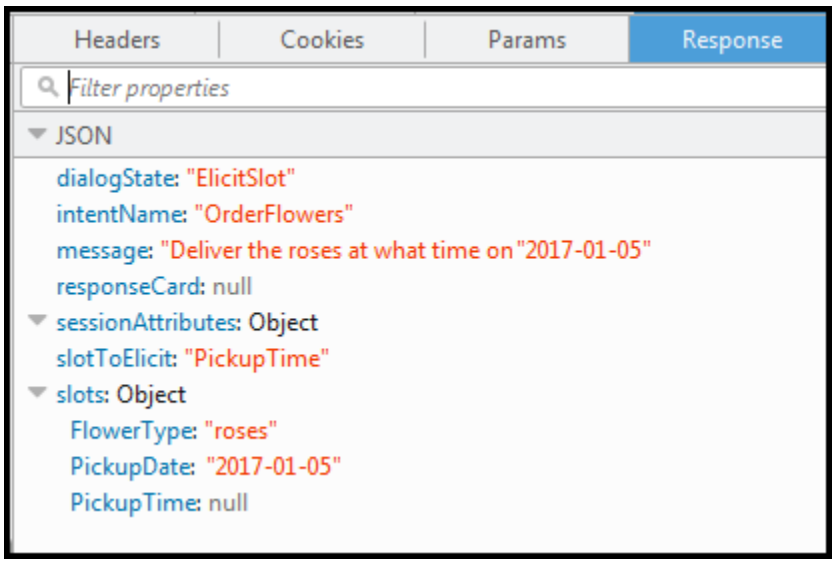
```
POST /bot/OrderFlowers/alias/$LATEST/user/4o9wwdhx6nlheferh6a73fujd3118f5w/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
    "inputText": "tomorrow",
    "sessionAttributes": {}
}
```

   リクエストボディの inputText は、ユーザー入力を示します。sessionAttributes は空のままです。

   b. Amazon Lex はまず、現在のインテントのコンテキストの inputText を解釈します (このサービスでは PickupDate スロットに関する情報を特定のユーザーに求めていたことが記憶されています)。Amazon Lex は現在のインテントのスロット (PickupDate) の値を更新します。値を引き出す別のスロット (PickupTime) を選択します。値を引き出すプロンプトの 1 つ (2017 年 1 月 5 日の何時にバラを配達しますか?) がクライアントに返されます。

   次に、Amazon Lex は以下のレスポンスを返します。



クライアントはレスポンス内のメッセージを表示しています。

4. ユーザーの入力: 「午後 6 時」

   a. クライアント (コンソール) は以下の PostText リクエストを Amazon Lex に送信します。

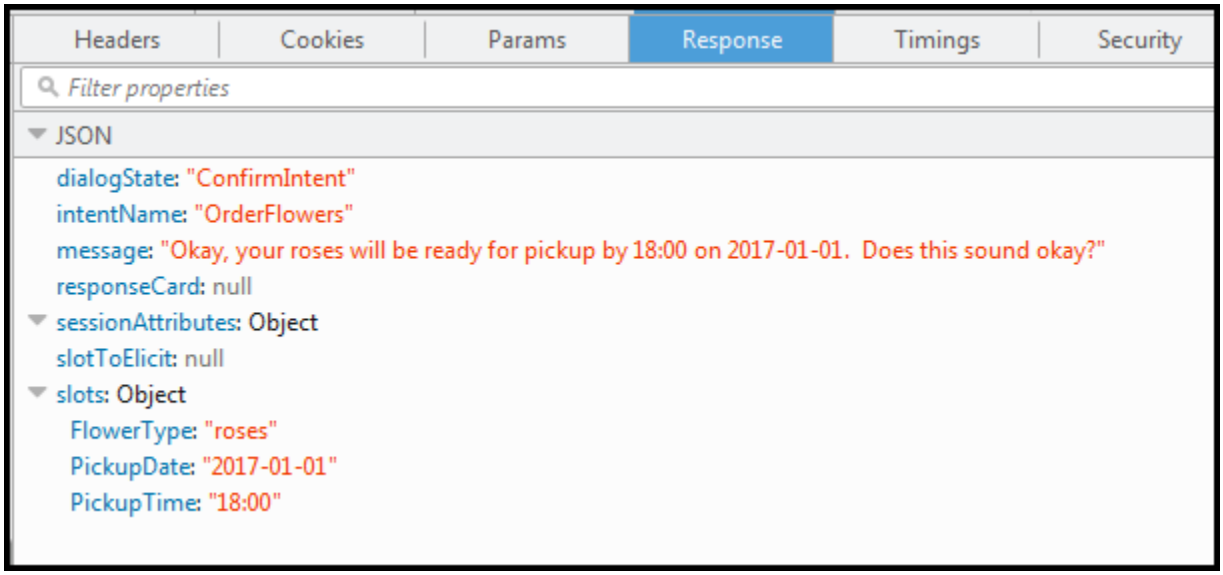
```
POST /bot/OrderFlowers/alias/$LATEST/user/4o9wwdhx6nlheferh6a73fujd3118f5w/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
    "inputText": "6 pm",
    "sessionAttributes": {}
}
```

   リクエストボディの `inputText` は、ユーザー入力を示します。`sessionAttributes` は空のままです。

   b. Amazon Lex はまず、現在のインテントのコンテキストの `inputText` を解釈します (このサービスでは `PickupTime` スロットに関する情報を特定のユーザーに求めていたことが記憶されています)。Amazon Lex はまず現在のインテントのスロット値を更新します。ここで、Amazon Lex はすべてのスロットのデータがそろっていることを検出します。

   `OrderFlowers` インテントには確認メッセージが設定されています。そのため、Amazon Lex はインテントの達成に進む前に、ユーザーからの明示的な確認を必要とします。Amazon Lex は、花を注文する前に確認を要求する以下のメッセージをクライアントに送信します。



クライアントはレスポンス内のメッセージを表示しています。

5. ユーザーの入力: 「はい」

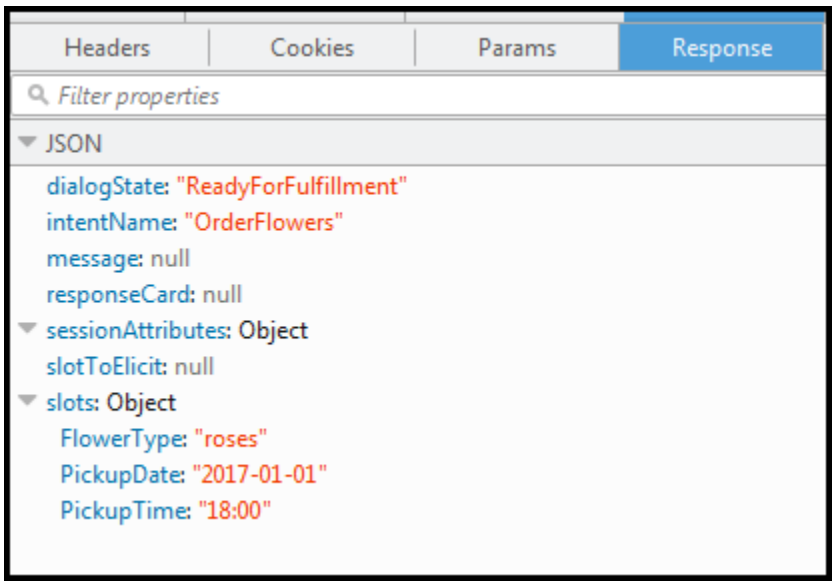
   a. クライアント (コンソール) は以下の PostText リクエストを Amazon Lex に送信します。

```
POST /bot/OrderFlowers/alias/$LATEST/user/4o9wwdhx6nlheferh6a73fujd3118f5w/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
    "inputText": "Yes",
    "sessionAttributes": {}
}
```

   リクエストボディの inputText は、ユーザー入力を示します。sessionAttributes は空のままです。

   b. Amazon Lex は、現在のインテントの確認のコンテキストで inputText を解釈して、注文を進めることをユーザーが望んでいることを理解します。OrderFlowers インテントには、フルフィルメントアクティビティとして ReturnIntent が設定されています (インテントを達成するための Lambda 関数はありません)。したがって、Amazon Lex は次のスロットデータをクライアントに返します。



Amazon Lex は `dialogState` を `ReadyForFulfillment` に設定します。これで、クライアントはインテントを達成できます。

6. ここで、ボットをもう一度テストします。そのためには、コンソールで [Clear] リンクを選択して、新しい (ユーザー) コンテキストを確立する必要があります。ここで、花の注文のデータを提供するときに、無効なデータを指定してみます。例:

   - 花の種類として「Jasmine」 (サポートされている花の種類ではない)
   - 花をピックアップする日付として「昨日」

   ボットではこれらの値が受け付けられることがわかります。これは、ユーザーデータを初期化/検証するコードがないためです。次のセクションでは、その処理を行う Lambda 関数を追加します。Lambda 関数について、以下の点に注意してください。

   - Lambda 関数はユーザー入力のたびにスロットデータを検証します。インテントは最後に達成されます。つまり、ボットはスロットデータをクライアントに返すだけではなく、花の注文を処理してユーザーにメッセージを返します。詳細については、「Lambda 関数を使用する」を参照してください。
   - Lambda 関数はセッション属性も設定します。セッション属性の詳細については、「PostText」を参照してください。

   ご利用開始のセクションを完了したら、その他の演習 (その他の例: Amazon Lex ボットの作成) を行うことができます。旅行を予約する は、セッション属性を使用してクロスインテント情報を共有し、ユーザーと動的に会話します。

次のステップ

[ステップ 3: Lambda 関数を作成する (コンソール)]

ステップ 3: Lambda 関数を作成する (コンソール)

Lambda 関数を作成し (lex-order-flowers-python 設計図を使用)、 AWS Lambda コンソールでサンプルのイベントデータを使用して呼び出しのテストを行います。

Amazon Lex コンソールに戻り、前のセクションで作成した `OrderFlowersBot` の `OrderFlowers` インテントを達成するコードフックとして Lambda 関数を追加します。

Lambda 関数を作成するには (コンソール)

1. AWS Management Console にサインインして AWS Lambda コンソール (https://console.aws.amazon.com/lambda/) を開きます。
2. [Create function] (関数の作成) を選択します。
3. [Create function] (関数の作成) ページで、[Blueprints] (設計図) を選択します。フィルタテキストボックスに「**`lex-`**」と入力し、`Enter` キーを押して設計図を見つけ、[`lex-order-flowers-python` 設計図] を選択します。

   Lambda 関数の設計図は Node.js と Python の両方で提供されています。この演習では Python ベースの設計図を使用します。
4. [基本的な情報] ページでは、以下を実行します。
   - Lambda 関数の名前 (`OrderFlowersCodeHook`) を入力します。
   - 実行ロールについては、[Create a new role with basic Lambda permissions] (基本的な Lambda アクセス権限で新しいロールを作成) を選択します。
   - 他はデフォルト値のままにしておきます。
5. [Create function] (関数の作成) を選択します。
6. 英語 (US) (en-US) 以外のロケールを使用している場合は、特定の口ケールの設計図の更新 の説明に従ってインテント名を更新します。
7. Lambda 関数をテストします。
   a. [テストイベントの選択]、[テストイベント設定] の順に選択します。
   b. [Event template] (イベントテンプレート) リストから [Amazon Lex-Order Flowers] (Amazon Lex - お花の注文) を選択します。このサンプルイベントは Amazon Lex のリクエスト/レ

スポンスモデル (「Lambda 関数を使用する」を参照) と一致します。テストイベント名 (LexOrderFlowersTest) を指定します。

c. [Create] (作成) を選択します。

d. [テスト] を選択してコードフックをテストします。

e. Lambda 関数が正常に実行されたことを確認します。この例のレスポンスは、Amazon Lex レスポンスモデルと一致します。

次のステップ

ステップ 4: Lambda 関数をコードフックとして追加する (コンソール)

ステップ 4: Lambda 関数をコードフックとして追加する (コンソール)

このセクションでは、次のように Lambda 関数を使用するように OrderFlowers インテントの設定を更新します。

- 最初に OrderFlowers インテントを達成するためのコードフックとして Lambda 関数を使用します。ボットをテストして、Lambda 関数からフルフィルメントメッセージを受信していることを確認します。Amazon Lex は、花の注文に必要なすべてのスロットのデータが提供された後にのみ、Lambda 関数を呼び出します。
- 初期化と検証を行うために同じ Lambda 関数をコードフックとして設定します。Lambda 関数が、(スロットデータが提供されたときに) 検証を行っていることをテストして確認します。

Lambda 関数をコードフックとして追加するには (コンソール)

1. Amazon Lex コンソールで、[OrderFlowers] ボットを選択します。コンソールに [OrderFlowers] インテントが表示されます。インテントのバージョンが $LATEST に設定されていることを確認します。変更できるのはこのバージョンだけです。
2. Lambda 関数をフルフィルメントのコードフックとして追加してテストします。

   a. エディタで、[AWS Lambda function] (関数) を [Fulfillment] (フルフィルメント) として選択し、前のステップで作成した Lambda 関数を選択します (OrderFlowersCodeHook)。[OK] を選択して、Lambda 関数を呼び出すアクセス権限を Amazon Lex に付与します。

   この Lambda 関数は、インテントを達成するためのコードフックとして設定しています。この関数を Amazon Lex が呼び出すのは、インテントを達成するために必要なすべてのスロットデータをユーザーから取得した後のみです。

b. [Goodbye message] を指定します。

c. [Build] を選択します。

d. 前の会話を使用してボットをテストします。

最後の文章である「ありがとうございました。お客様のバラは ...」は、コードフックとして設定した Lambda 関数からのレスポンスです。前のセクションでは、Lambda 関数はありませんでした。ここでは、Lambda 関数を使用して実際に `OrderFlowers` インテントを達成します。

3. Lambda 関数を初期化および検証のコードフックとして追加してテストします。

Lambda 関数のサンプルコードを使用して、ユーザー入力の検証とフルフィルメントの両方を行うことができます。Lambda 関数が受け取る入力イベントには、どのコード部分を実行するかをコードで判断するためのフィールド (`invocationSource`) があります。詳細については、「Lambda 関数の入力イベントとレスポンスの形式」を参照してください。

a. `OrderFlowers` の $LATEST バージョンを選択します。これは更新できる唯一のバージョンです。

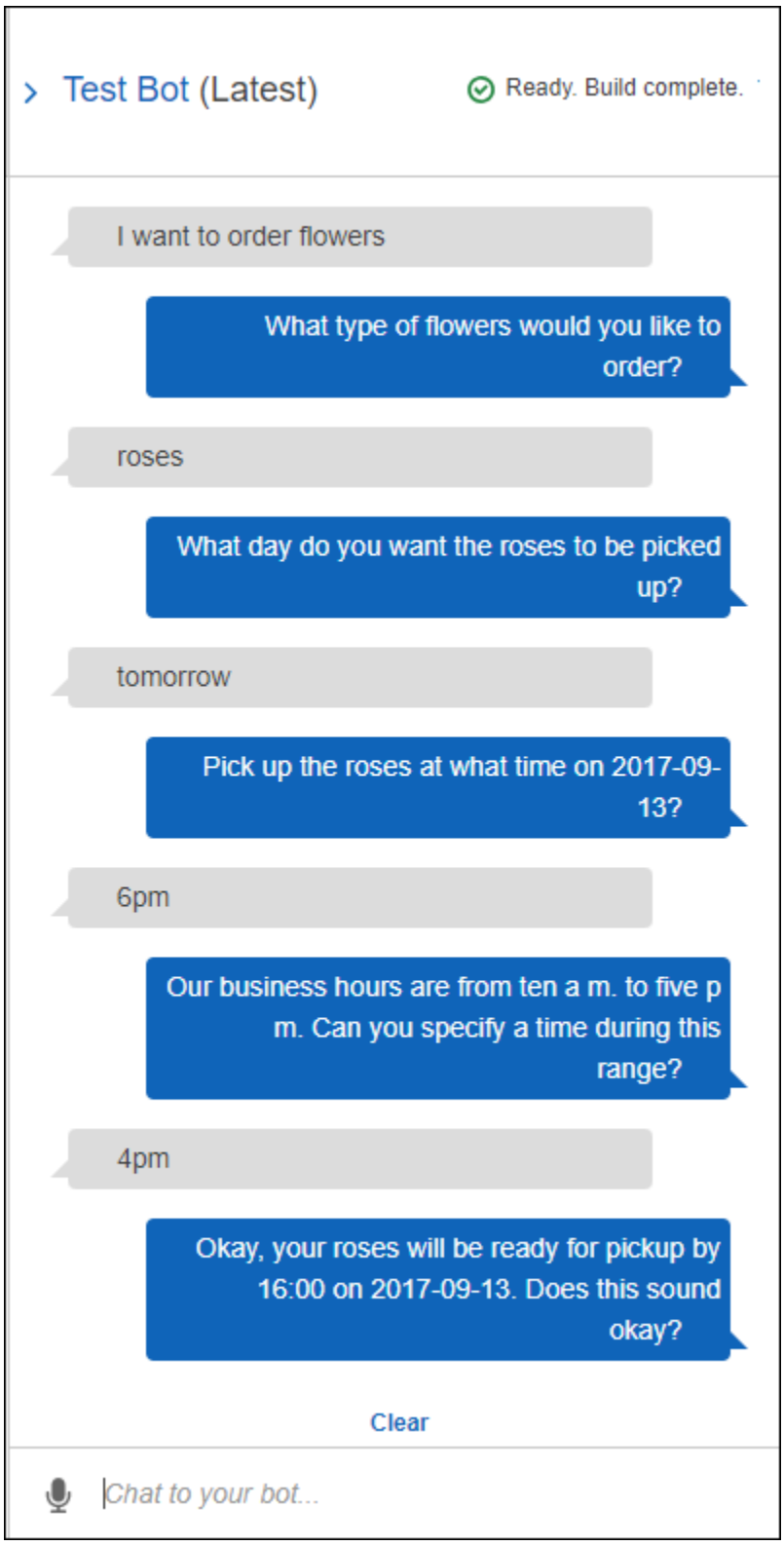
b. エディタで、[オプション] の [Initialization and validation] (初期化と検証) を選択します。

c. ここでも、同じ Lambda 関数を選択します。

d. [Build] を選択します。

e. ボットをテストします。

次の画像ように Amazon Lex と対話する準備が整いました。検証部分をテストするには、時間として 6 PM を選択します。Lambda 関数からレスポンス (「当社の営業時間は午前 10 時から午後 5 時までとなっております」) が返され、選択し直すことを求められます。すべての有効なスロットデータを提供すると、Lambda 関数によって注文が確定されます。


Test Bot (Latest)
Ready. Build complete.
I want to order flowers
What type of flowers would you like to order?
roses
What day do you want the roses to be picked up?
tomorrow
Pick up the roses at what time on 2017-09-13?
6pm
Our business hours are from ten a m. to five p m. Can you specify a time during this range?
4pm
Okay, your roses will be ready for pickup by 16:00 on 2017-09-13. Does this sound okay?
Clear
Chat to your bot...

次のステップ

ステップ 5 (オプション): 情報フローの詳細を確認する (コンソール)

ステップ 5 (オプション): 情報フローの詳細を確認する (コンソール)

このセクションでは、各ユーザー入力に対する、クライアントと Amazon Lex の間の情報のフローについて、Lambda 関数の統合も含めて説明します。

> **Note**
>
> このセクションでは、クライアントが PostText ランタイム API を使用して Amazon Lex にリクエストを送信することを前提としていて、それに応じてリクエストとレスポンスの詳細を示しています。クライアントが PostContent API を使用する場合の、クライアントと Amazon Lex の間の情報フローの例については、「ステップ 2a (オプション): 音声による情報フローの詳細を確認する (コンソール)」を参照してください。

PostText ランタイム API、および以下のステップで示しているリクエストとレスポンスに関するその他の詳細については、「PostText」を参照してください。

1. ユーザー:「花を注文したい」

   a. クライアント (コンソール) は以下の PostText リクエストを Amazon Lex に送信します。

```
POST /bot/OrderFlowers/alias/$LATEST/user/ignw84y6seypre4xly5rimopuri2xwnd/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
    "inputText": "I would like to order some flowers",
    "sessionAttributes": {}
}
```

   リクエストの URI と本文の両方で Amazon Lex に情報が提供されています。

   - リクエスト URI – ボット名 (OrderFlowers)、ボットのエイリアス ($LATEST)、およびユーザー名 (ユーザーを識別するランダムな文字列) を提供します。末尾の text では、これが PostText API リクエストである (PostContent ではない) ことが示されています。

- リクエストボディ – ユーザー入力 (inputText) と空の sessionAttributes が含まれています。クライアントが最初のリクエストを行うときにはセッション属性はありません。Lambda 関数は後でセッション属性を使用します。

b. inputText から、Amazon Lex はインテント (OrderFlowers) を検出します。このインテントには、ユーザーデータの初期化/検証を行うためのコードフックとして Lambda 関数が設定されています。そのため、Amazon Lex は以下の情報をイベントデータとして渡すことで、その Lambda 関数を呼び出します。

```
{
    "messageVersion": "1.0",
    "invocationSource": "DialogCodeHook",
    "userId": "ignw84y6seypre4xly5rimopuri2xwnd",
    "sessionAttributes": {},
    "bot": {
        "name": "OrderFlowers",
        "alias": null,
        "version": "$LATEST"
    },
    "outputDialogMode": "Text",
    "currentIntent": {
        "name": "OrderFlowers",
        "slots": {
            "PickupTime": null,
            "FlowerType": null,
            "PickupDate": null
        },
        "confirmationStatus": "None"
    }
}
```

詳細については、「入力イベントの形式」を参照してください。

クライアントが送信する情報に加えて、Amazon Lex はまた、以下の追加データを含めます。

- messageVersion – 現在 Amazon Lex でサポートしているのは 1.0 バージョンだけです。
- invocationSource – Lambda 関数呼び出しの目的を示しています。この場合は、ユーザーデータの初期化および検証を行うことです。この時点で、Amazon Lex はインテント

を達成するためのスロットデータの一部をユーザーがまだ指定していないことを知っています。

- currentIntent の情報。すべてのスロット値は null に設定されています。

c. この時点では、すべてのスロット値は null です。Lambda 関数が検証する対象はありません。Lambda 関数は以下のレスポンスを Amazon Lex に返します。

```
{
    "sessionAttributes": {},
    "dialogAction": {
        "type": "Delegate",
        "slots": {
            "PickupTime": null,
            "FlowerType": null,
            "PickupDate": null
        }
    }
}
```

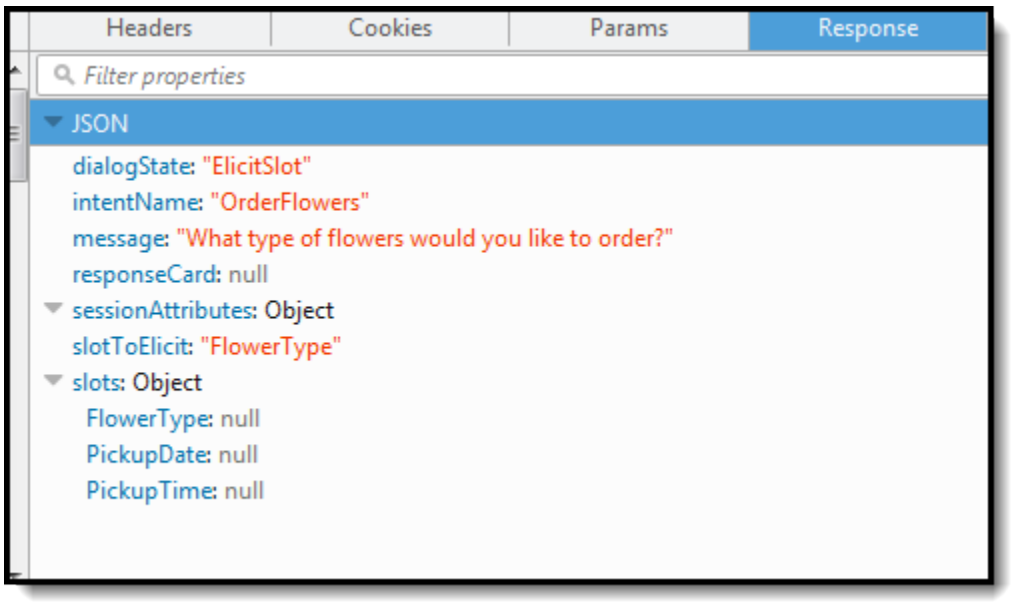
レスポンスの形式については、「レスポンスの形式」を参照してください。

次の点に注意してください。

- dialogAction.type – この値を Delegate に設定することで、Lambda 関数は次の一連のアクションを決定する責任を Amazon Lex に委任します。

> **Note**
>
> Lambda 関数は、ユーザーデータの検証で何かを検出した場合に、次に何をするかを Amazon Lex に指示します。それについては以下のステップで説明します。

d. dialogAction.type に従って、Amazon Lex は次の一連のアクションを決定します。どのスロットも入力されていないため、FlowerType スロットの値を引き出すことを決定します。値を引き出すプロンプトの 1 つ (「どの種類の花を注文しますか?」) をこのスロットに選択し、以下のレスポンスをクライアントに返します。



クライアントはレスポンス内のメッセージを表示しています。

2. ユーザー: 「バラ」

   a. クライアント は以下の PostText リクエストを Amazon Lex に送信します。

```
POST /bot/OrderFlowers/alias/$LATEST/user/ignw84y6seypre4xly5rimopuri2xwnd/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
    "inputText": "roses",
    "sessionAttributes": {}
}
```

   リクエストボディの `inputText` でユーザー入力が提供されています。`sessionAttributes` は空のままです。

   b. Amazon Lex はまず現在のインテントのコンテキストで `inputText` を解釈します。サービスでは `FlowerType` スロットに関する情報をこのユーザーに求めていたことが記憶されています。現在のインテントのスロット値を更新し、以下のイベントデータを使用して Lambda 関数を呼び出します。

```
{
    "messageVersion": "1.0",
    "invocationSource": "DialogCodeHook",
```

```
    "userId": "ignw84y6seypre4xly5rimopuri2xwnd",
    "sessionAttributes": {},
    "bot": {
        "name": "OrderFlowers",
        "alias": null,
        "version": "$LATEST"
    },
    "outputDialogMode": "Text",
    "currentIntent": {
        "name": "OrderFlowers",
        "slots": {
            "PickupTime": null,
            "FlowerType": "roses",
            "PickupDate": null
        },
        "confirmationStatus": "None"
    }
}
```

次の点に注意してください。

- invocationSource – DialogCodeHook のままです (ユーザーデータを検証しているだけです)。
- currentIntent.slots – Amazon Lex は FlowerType スロットを「バラ」に更新しています。

c. invocationSource の DialogCodeHook の値に従って、Lambda 関数はユーザーデータの検証を実行します。この関数は roses を有効なスロット値として認識し (また、Price をセッション属性として設定して)、以下のレスポンスを Amazon Lex に返します。

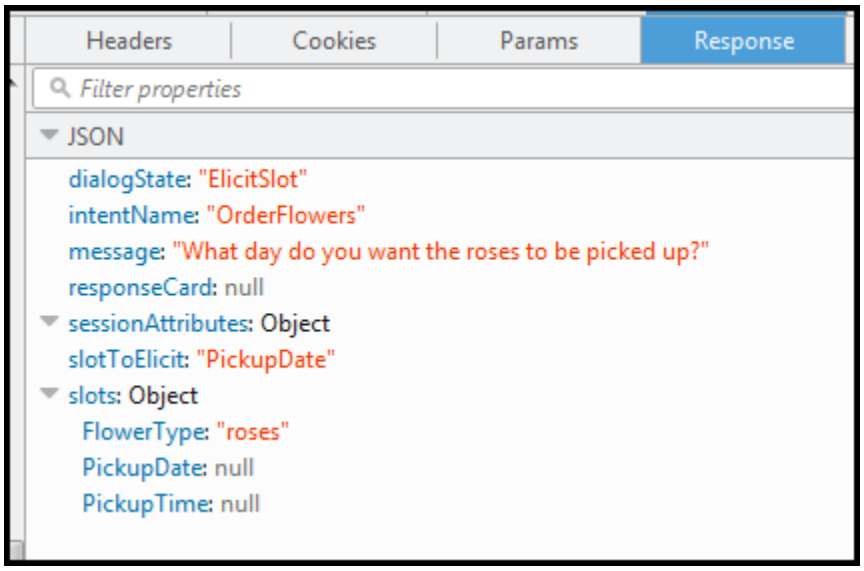
```
{
    "sessionAttributes": {
        "Price": 25
    },
    "dialogAction": {
        "type": "Delegate",
        "slots": {
            "PickupTime": null,
            "FlowerType": "roses",
            "PickupDate": null
        }
    }
```

```
}
```

次の点に注意してください。

- `sessionAttributes` – Lambda 関数は (バラの) `Price` をセッション属性として追加しています。
- `dialogAction.type` – `Delegate` に設定されます。ユーザーデータは有効であるため、Lambda 関数は次の一連のアクションを選択するように Amazon Lex に指示します。

d. `dialogAction.type` に従って、Amazon Lex は次の一連のアクションを選択します。Amazon Lex は、より多くのスロットデータが必要であることを知っているため、インテント設定に従って最も優先度が高い次の未指定スロット (`PickupDate`) を選択します。Amazon Lex は、このスロットにプロンプトメッセージの 1 つ (「何日にバラを受け取りたいですか?」) をインテント設定に応じて選択し、クライアントに次のレスポンスを返答として送信します。



クライアントには、レスポンスのメッセージ (「何日にバラをピックアップなさいますか?」) だけが表示されます。

3. ユーザー: 「明日」

a. クライアント は以下の PostText リクエストを Amazon Lex に送信します。

```
POST /bot/OrderFlowers/alias/$LATEST/user/ignw84y6seypre4xly5rimopuri2xwnd/text
"Content-Type":"application/json"
```

```
"Content-Encoding":"amz-1.0"

{
    "inputText": "tomorrow",
    "sessionAttributes": {
        "Price": "25"
    }
}
```

リクエストボディの inputText でユーザー入力が提供され、クライアントはセッション属性をサービスに返します。

b. Amazon Lex は、これが PickupDate スロットに対して引き出されているデータであるというコンテキストを覚えています。このコンテキストでは、inputText が PickupDate スロットに対する値であることを知っています。Amazon Lex は以下のイベントを送信することで Lambda 関数を呼び出します。

```
{
    "messageVersion": "1.0",
    "invocationSource": "DialogCodeHook",
    "userId": "ignw84y6seypre4xly5rimopuri2xwnd",
    "sessionAttributes": {
        "Price": "25"
    },
    "bot": {
        "name": "OrderFlowersCustomWithRespCard",
        "alias": null,
        "version": "$LATEST"
    },
    "outputDialogMode": "Text",
    "currentIntent": {
        "name": "OrderFlowers",
        "slots": {
            "PickupTime": null,
            "FlowerType": "roses",
            "PickupDate": "2017-01-05"
        },
        "confirmationStatus": "None"
    }
}
```

Amazon Lex は currentIntent.slots の値を設定することで、PickupDate を更新しています。また、サービスによって sessionAttributes がそのまま Lambda 関数に渡されています。

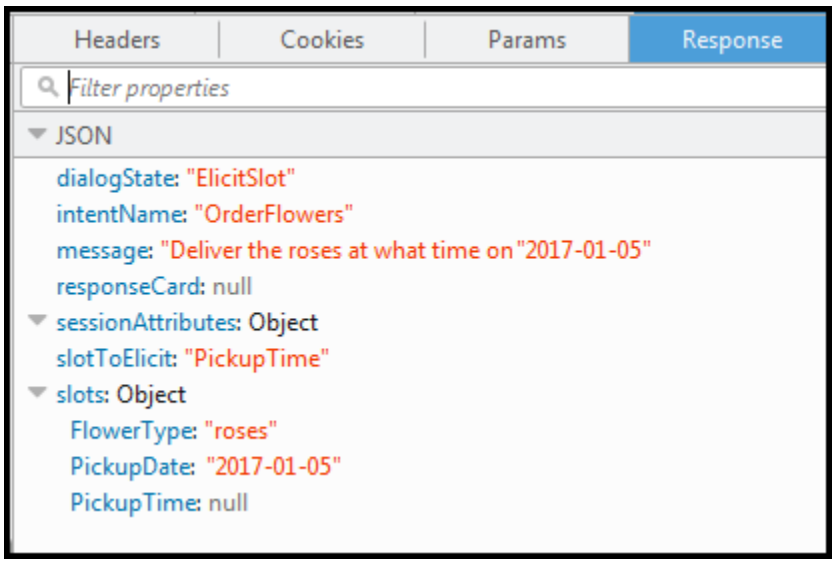
c. invocationSource の DialogCodeHook の値に従って、Lambda 関数はユーザーデータの検証を行います。この関数は PickupDate のスロット値が有効であることを認識し、以下のレスポンスを Amazon Lex に返します。

```
{
    "sessionAttributes": {
        "Price": 25
    },
    "dialogAction": {
        "type": "Delegate",
        "slots": {
            "PickupTime": null,
            "FlowerType": "roses",
            "PickupDate": "2017-01-05"
        }
    }
}
```

次の点に注意してください。

- sessionAttributes – 変更しません。
- dialogAction.type – Delegate に設定されます。ユーザーデータは有効であるため、Lambda 関数は次の一連のアクションを選択するように Amazon Lex に指示します。

d. dialogAction.type に従って、Amazon Lex は次の一連のアクションを選択します。Amazon Lex は、より多くのスロットデータが必要であることを知っているため、インテント設定に従って最も優先度が高い次の未指定スロット (PickupTime) を選択します。Amazon Lex はプロンプトのメッセージ (「2017 年 1 月 5 日の何時にバラを配達いたしましょうか?」) のいずれかを選択し、 以下のレスポンスをクライアントに送信します。



クライアントはレスポンス内のメッセージ (「2017 年 1 月 5 日の何時にバラを配達いたしましょうか?」) を表示します。

4. ユーザー: 「午後 4 時」

   a. クライアント は以下の PostText リクエストを Amazon Lex に送信します。

```
POST /bot/OrderFlowers/alias/$LATEST/user/ignw84y6seypre4xly5rimopuri2xwnd/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
    "inputText": "4 pm",
    "sessionAttributes": {
        "Price": "25"
    }
}
```

   リクエストボディの inputText でユーザー入力が提供されています。クライアントはそのリクエストで sessionAttributes を渡します。

   b. Amazon Lex はコンテキストを理解しています。PickupTime スロットに対するデータを引き出していたというコンテキストを理解しています。このコンテキストでは、inputText が PickupTime スロットに対する値であることを知っています。Amazon Lex は以下のイベントを送信することで Lambda 関数を呼び出します。

```
{
    "messageVersion": "1.0",
```

```
    "invocationSource": "DialogCodeHook",
    "userId": "ignw84y6seypre4xly5rimopuri2xwnd",
    "sessionAttributes": {
        "Price": "25"
    },
    "bot": {
        "name": "OrderFlowersCustomWithRespCard",
        "alias": null,
        "version": "$LATEST"
    },
    "outputDialogMode": "Text",
    "currentIntent": {
        "name": "OrderFlowers",
        "slots": {
            "PickupTime": "16:00",
            "FlowerType": "roses",
            "PickupDate": "2017-01-05"
        },
        "confirmationStatus": "None"
    }
}
```

Amazon Lex は currentIntent.slots の値を設定することで、PickupTime を更新しています。

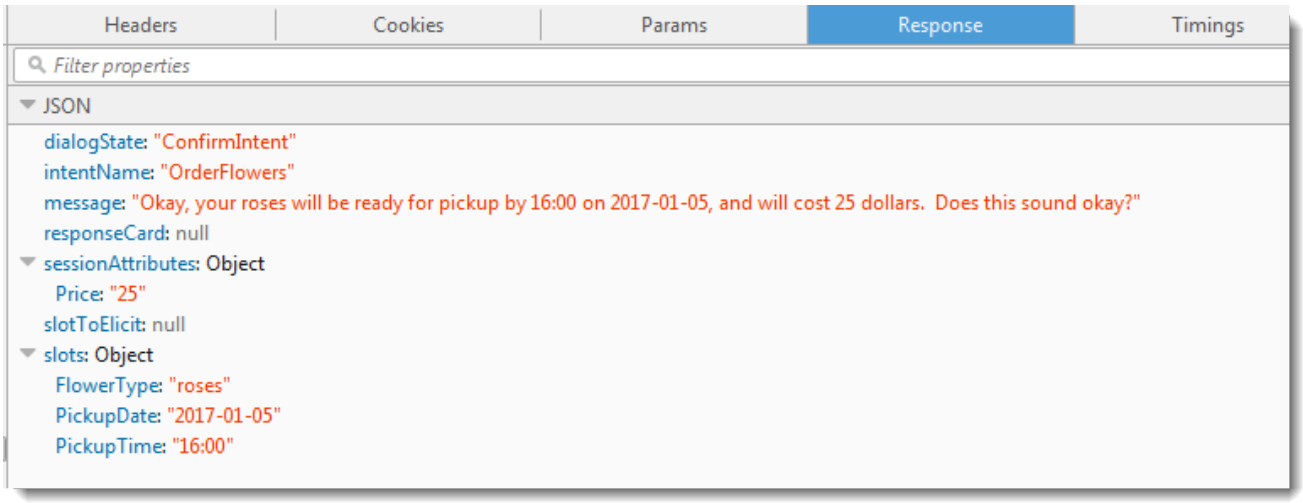
c. invocationSource の DialogCodeHook の値に従って、Lambda 関数はユーザーデータの検証を実行します。この関数は PickupDate のスロット値が有効であることを認識し、以下のレスポンスを Amazon Lex に返します。

```
{
    "sessionAttributes": {
        "Price": 25
    },
    "dialogAction": {
        "type": "Delegate",
        "slots": {
            "PickupTime": "16:00",
            "FlowerType": "roses",
            "PickupDate": "2017-01-05"
        }
    }
}
```

次の点に注意してください。

- sessionAttributes – セッション属性は変更されていません。
- dialogAction.type - Delegate に設定されます。ユーザーデータは有効であるため、Lambda 関数は次の一連のアクションを選択するように Amazon Lex に指示します。

d. この時点で、Amazon Lex はすべてのスロットデータがそろっていることを知っています。このインテントには確認プロンプトが設定されています。そのため、Amazon Lex は、インテントを達成する前にユーザーの確認を求める以下のレスポンスを送信します。



クライアントはレスポンス内のメッセージをそのまま表示し、ユーザーの応答を待ちます。

5. ユーザー:「はい」

a. クライアント は以下の PostText リクエストを Amazon Lex に送信します。

```
POST /bot/OrderFlowers/alias/$LATEST/user/ignw84y6seypre4xly5rimopuri2xwnd/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
    "inputText": "yes",
    "sessionAttributes": {
        "Price": "25"
    }
}
```

b. Amazon Lex は、現在のインテントの確認のコンテキストで inputText を解釈して、Amazon Lexは、注文を進めることをユーザーが望んでいることを理解しま

す。Amazon Lex は今回は、以下のイベントを送信することで、インテントを達成するために Lambda 関数を呼び出します。Lambda 関数に送信するイベントで invocationSource を FulfillmentCodeHook に設定しています。Amazon Lex はまた、confirmationStatus から Confirmed に設定します。

```
{
    "messageVersion": "1.0",
    "invocationSource": "FulfillmentCodeHook",
    "userId": "ignw84y6seypre4xly5rimopuri2xwnd",
    "sessionAttributes": {
        "Price": "25"
    },
    "bot": {
        "name": "OrderFlowersCustomWithRespCard",
        "alias": null,
        "version": "$LATEST"
    },
    "outputDialogMode": "Text",
    "currentIntent": {
        "name": "OrderFlowers",
        "slots": {
            "PickupTime": "16:00",
            "FlowerType": "roses",
            "PickupDate": "2017-01-05"
        },
        "confirmationStatus": "Confirmed"
    }
}
```

次の点に注意してください。

- invocationSource – Amazon Lex は今回は、この値を FulfillmentCodeHook に設定して、インテントを達成するように Lambda 関数に指示しています。
- confirmationStatus – Confirmed に設定されます。

c. 今回、Lambda 関数は OrderFlowers インテントを達成し、次のレスポンスを返します。

```
{
    "sessionAttributes": {
        "Price": "25"
    },
    "dialogAction": {
```

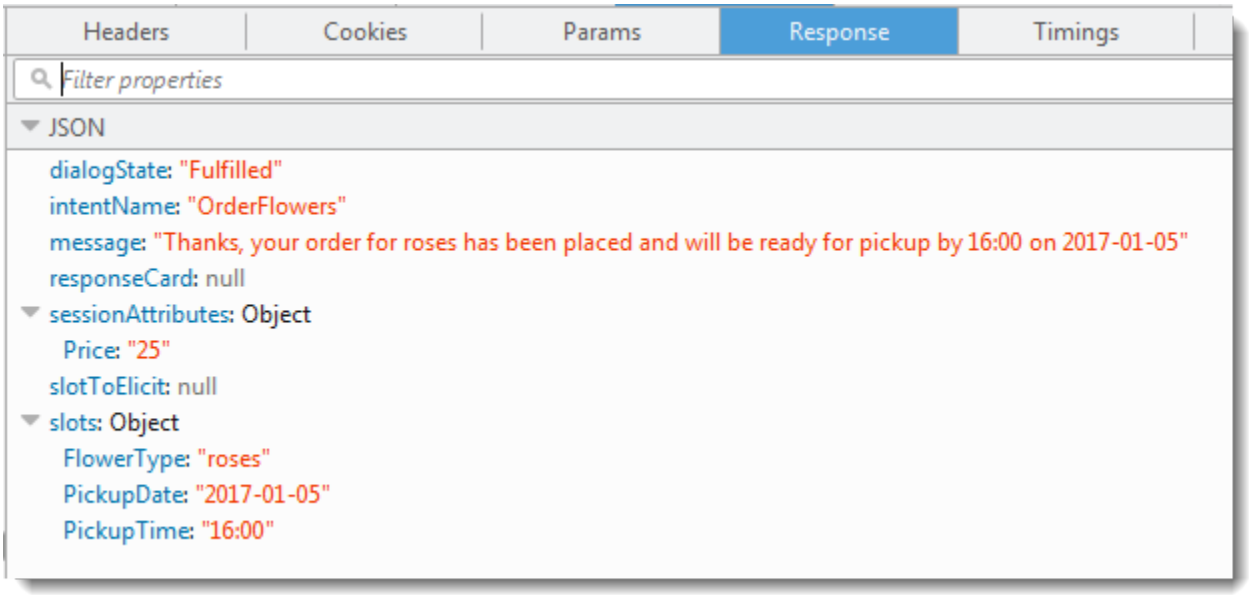
```
        "type": "Close",
        "fulfillmentState": "Fulfilled",
        "message": {
            "contentType": "PlainText",
            "content": "Thanks, your order for roses has been placed and will
 be ready for pickup by 16:00 on 2017-01-05"
        }
    }
}
```

次の点に注意してください。

- `dialogAction.type` を設定する – Lambda 関数はこの値を `Close` に設定し、ユーザーの応答を想定しないことを Amazon Lex に指示しています。
- `dialogAction.fulfillmentState` – Fulfilled に設定されていて、ユーザーに伝える適切なメッセージ (`message`) が含まれています。

d. Amazon Lex は `fulfillmentState` を確認し、以下のレスポンスをクライアントに返します。

Amazon Lex は以下のメッセージをクライアントに返しています。



以下の点に注意してください。

- `dialogState` – Amazon Lex はこの値を `fulfilled` に設定しています。
- `message` – Lambda 関数が提供したのと同じメッセージです。

クライアントはそのメッセージを表示します。

6. ここで、ボットをもう一度テストします。新しい (ユーザー) コンテキストを確立するには、テストウィンドウの [Clear] リンクを選択します。ここでは、`OrderFlowers` インテントに対して無効なスロットデータを指定します。Lambda 関数は今回は、データ検証を実行し、無効なスロットデータ値を null にリセットし、有効なデータをユーザーに求めるように Amazon Lex に依頼します。例えば、以下のことを試してみます。

   - 花の種類として「Jasmine」 (サポートされている花の種類ではない)
   - 花をピックアップする日付として「昨日」
   - 注文した後で、注文の確認に対して「はい」と応答する代わりに、花の種類を入力します。それに対して、Lambda 関数は、花の注文の現在の合計はそのままにして、セッション属性内の `Price` を更新します。

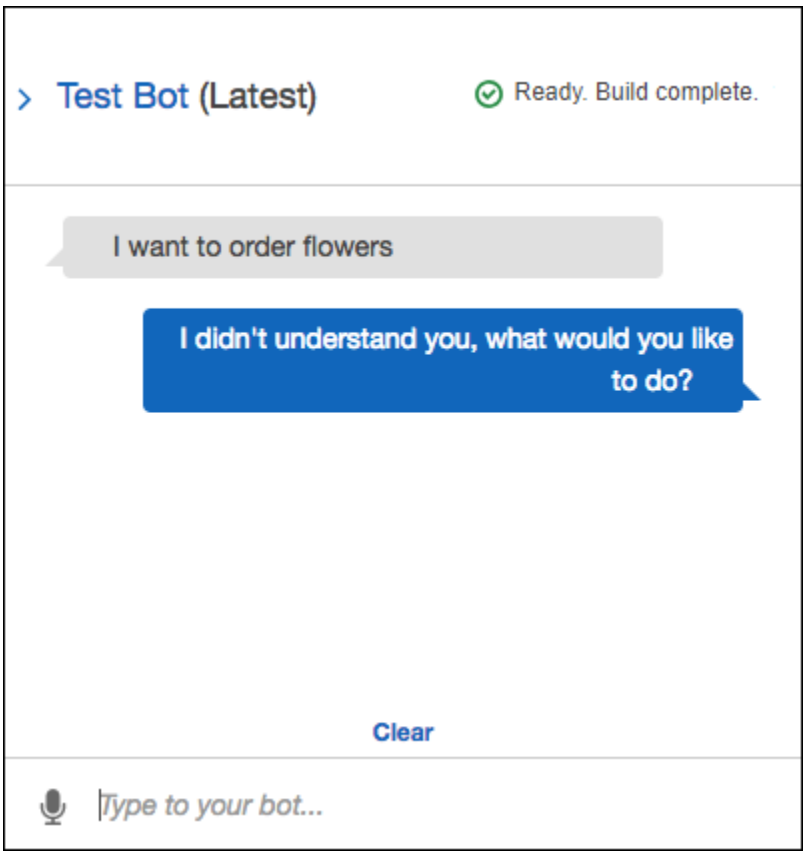
   Lambda 関数はフルフィルメントアクティビティも実行します。

次のステップ

[ステップ 6: インテント設定を更新して発話を追加する (コンソール)]

ステップ 6: インテント設定を更新して発話を追加する (コンソール)

`OrderFlowers` ボットには、2 つの発話のみが設定されています。これは、機械学習モデルを構築して、ユーザーのインテントを認識し応答するための制限された情報を Amazon Lex に提供します。次のテストウィンドウに「I want to order flowers」と入力してみてください。Amazon Lex は、このテキストを認識しないため、「I didn't understand you, what would you like to do?」と応答します。発話を追加することで、機械学習モデルを改善できます。



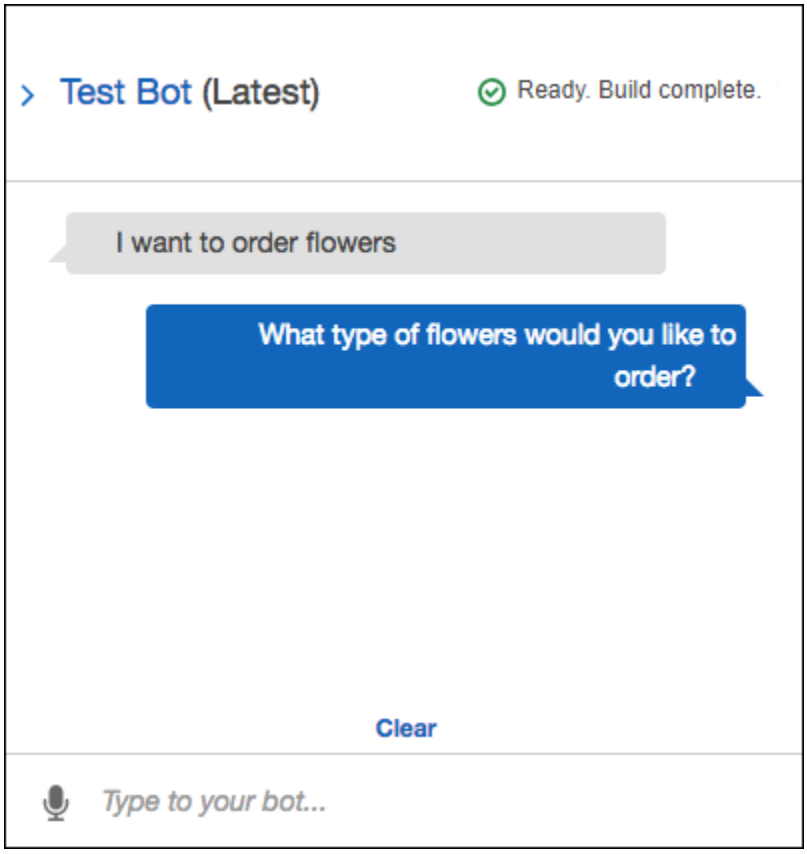
発話を追加するたびに Amazon Lex でユーザーに応答する方法に関する情報が増えます。正確な発話を追加する必要はありません。Amazon Lex は、提供されたサンプルから一般化を行い、正確な一致および類似する入力の両方を認識します。

発話を追加するには (コンソール)

1. 次の画像のように、インテントエディタの [サンプル発話] セクションに「I want flowers」と入力して発話をインテントに追加し、この新しい発話の横にあるプラスアイコンをクリックします。

2. ボットを構築して変更を反映します。[Build] を選択し、再度 [Build] を選択します。

3. ボットをテストし、新しい発話が認識されたことを確認します。次の画像のように、テストウィンドウに「I want to order flowers」と入力します。Amazon Lex は、この句を認識し、「What type of flowers would you like to order?」と応答します。



次のステップ

[ステップ 7 (オプション): クリーンアップする (コンソール)]

ステップ 7 (オプション): クリーンアップする (コンソール)

次に、作成したリソースを削除し、アカウントをクリーンアップします。

削除できるのは、使用中ではないリソースだけです。通常は、以下の順序でリソースを削除する必要があります。

- ボットを削除して、インテントのリソースを解放します。
- インテントを削除して、スロットタイプのリソースを解放します。
- 最後にスロットタイプを削除します。

アカウントをクリーンアップするには (コンソール)

1. AWS Management Console にサインインし、Amazon Lex コンソール (https://console.aws.amazon.com/lex/) を開きます。
2. ボットのリストで、[OrderFlowers] の横にあるチェックボックスをオンにし ます。
3. ボットを削除するには、[Delete] を選択し、確認ダイアログボックスで [Continue] を選択します。
4. 左のペインで [Intents] を選択します。
5. インテントのリストで、[OrderFlowersIntent] を選択します。
6. インテントを削除するには、[Delete] を選択し、確認ダイアログボックスで [Continue] を選択します。
7. 左のペインで、[Slot types] を選択します。
8. スロットタイプのリストで、[Flowers] を選択します。
9. スロットタイプを削除するには、[Delete] を選択し、確認ダイアログボックスで [Continue] を選択します。

これで、作成したすべてのAmazon Lex リソースが削除され、アカウントがクリーンアップされました。必要に応じて、Lambda コンソールを使用して、この演習で使用した Lambda 関数を削除します。

演習 2: Amazon Lex のカスタムボットを作成する

この演習では、Amazon Lex コンソールを使用してピザを注文するカスタムボット (`OrderPizzaBot`) を作成します。このボットを設定するには、カスタムインテント (`OrderPizza`) の追加、カスタムスロットタイプの定義、スロットの定義を通じて、ピザの注文 (ピザのクラストやサイズなど) を処理します。スロットタイプとスロットの詳細については、「Amazon Lex: 仕組み」を参照してください。

トピック

- ステップ 1: Lambda 関数を作成する
- ステップ 2: ボットを作成する
- ステップ 3: ボットを構築してテストする
- ステップ 4 (オプション): クリーンアップする

ステップ 1: Lambda 関数を作成する

最初に、ピザの注文を達成する Lambda 関数を作成します。この関数は、次のセクションで作成する Amazon Lex ボットで指定します。

Lambda 関数を作成するには

1. AWS Management Console にサインインして AWS Lambda コンソール (https://console.aws.amazon.com/lambda/) を開きます。
2. [Create function] を選択します。
3. [Create function] ページで、[Author from scratch] を選択します。

   この演習では、事前に用意されたカスタムコードを使用して Lambda 関数を作成します。したがって、最初から関数を作成するオプションを選択します。

   次のコマンドを実行します

   a. 名前 (`PizzaOrderProcessor`) を入力します。
   b. [ランタイム] で、最新バージョンの Node.js を選択します。
   c. [Role] で、[Create a new role from template(s)] を選択します。
   d. 新しいロール名 (`PizzaOrderProcessorRole`) を入力します。
   e. [Create function] (関数の作成) を選択します。
4. [関数] ページで、以下の作業を行います。

   [Function code] セクションで、[Edit code inline] を選択し、次の Node.js 関数コードをコピーしてウィンドウに貼り付けます。

```
'use strict';

// Close dialog with the customer, reporting fulfillmentState of Failed or
 Fulfilled ("Thanks, your pizza will arrive in 20 minutes")
function close(sessionAttributes, fulfillmentState, message) {
    return {
        sessionAttributes,
        dialogAction: {
            type: 'Close',
            fulfillmentState,
            message,
        },
```

```
    };
}

// --------------- Events -----------------------

function dispatch(intentRequest, callback) {
    console.log(`request received for userId=${intentRequest.userId}, intentName=
${intentRequest.currentIntent.name}`);
    const sessionAttributes = intentRequest.sessionAttributes;
    const slots = intentRequest.currentIntent.slots;
    const crust = slots.crust;
    const size = slots.size;
    const pizzaKind = slots.pizzaKind;

    callback(close(sessionAttributes, 'Fulfilled',
    {'contentType': 'PlainText', 'content': `Okay, I have ordered your ${size}
 ${pizzaKind} pizza on ${crust} crust`}));

}

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

// Route the incoming request based on intent.
// The JSON body of the request is provided in the event slot.
export const handler = (event, context, callback) => {
    try {
        dispatch(event,
            (response) => {
                callback(null, response);
            });
    } catch (err) {
        callback(err);
    }
};
```

5. [Save (保存)] を選択します。

サンプルイベントデータを使用した Lambda 関数のテスト

コンソールで、サンプルイベントデータを使用して手動で Lambda 関数を呼び出すことで、この関数をテストします。

Lambda 関数をテストするには:

1. AWS Management Console にサインインして AWS Lambda コンソール (https://console.aws.amazon.com/lambda/) を開きます。
2. [Lambda function] ページで、[Lambda function] (Lambda 関数) (`PizzaOrderProcessor`.) を選択します。
3. 関数のページで、テストイベントのリストから [Configure test events] を選択します。
4. [Configure test event] ページで、以下の操作を行います。

   a. [Create new test event] を選択します。

   b. [Event name] フィールドに、イベント名 (`PizzaOrderProcessorTest`) を入力します。

   c. 次の Amazon Lex イベントをウィンドウ内にコピーします。

```
{
  "messageVersion": "1.0",
  "invocationSource": "FulfillmentCodeHook",
  "userId": "user-1",
  "sessionAttributes": {},
  "bot": {
    "name": "PizzaOrderingApp",
    "alias": "$LATEST",
    "version": "$LATEST"
  },
  "outputDialogMode": "Text",
  "currentIntent": {
    "name": "OrderPizza",
    "slots": {
      "size": "large",
      "pizzaKind": "meat",
      "crust": "thin"
    },
    "confirmationStatus": "None"
  }
}
```

5. [Create] (作成) を選択します。

AWS Lambda によってテストが作成され、関数のページが再び表示されます。[Test] (テスト) を選択すると、Lambda 関数が実行されます。

結果ボックスで、[Details] を選択します。コンソールの [Execution result] ペインに、次の出力が表示されます。

```
{
  "sessionAttributes": {},
  "dialogAction": {
    "type": "Close",
    "fulfillmentState": "Fulfilled",
    "message": {
      "contentType": "PlainText",
      "content": "Okay, I have ordered your large meat pizza on thin crust."
    }
}
```

次のステップ

ステップ 2: ボットを作成する

ステップ 2: ボットを作成する

このステップでは、ピザの注文を処理するボットを作成します。

トピック

- ボットの作成
- インテントの作成
- スロットタイプの作成
- インテントの設定
- ボットの設定

ボットの作成

必要最低限の情報を使用して PizzaOrderingBot ボットを作成します。後で、このボットにインテント (ユーザーが実行するアクション) を追加します。

ボットを作成するには

1. AWS Management Console にサインインし、Amazon Lex コンソール (https://console.aws.amazon.com/lex/) を開きます。
2. ボットを作成します。

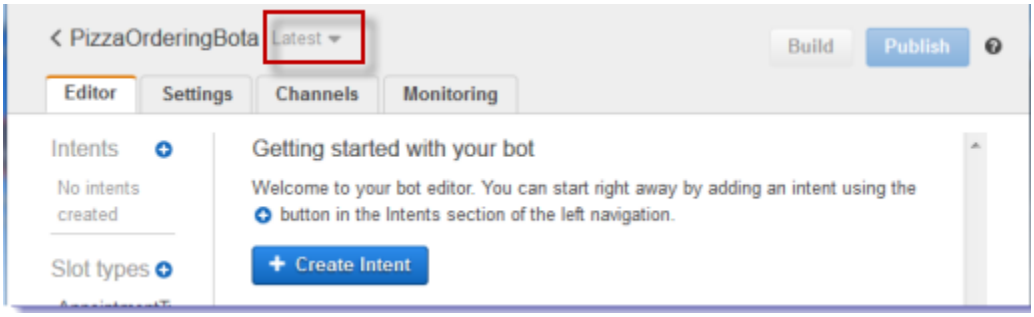
a. 初めてボットを作成する場合は、[Get Started] を選択します。それ以外の場合は、[Bots]、[Create] の順に選択します。

b. [Create your Lex bot] ページで、[Custom bot] を選択して、以下の情報を指定します。

- Bot name: PizzaOrderingBot
- 言語: ボットの言語とロケールを選択します。
- Output voice: Salli
- Session timeout: 5 分
- COPPA: 適切なレスポンスを選択します。
- ユーザーの発話ストレージ: 適切なレスポンスを選択します。

c. [Create] (作成) を選択します。

コンソールから Amazon Lex に新しいボットを作成するためのリクエストが送信されます。Amazon Lex でボットバージョンが $LATEST に設定されます。ボットの作成後に、次の画像に示すように、Amazon Lex に [エディタ] タブが表示されます。



- ボットバージョンの [Latest] は、コンソールでボット名の横に表示されます。Amazon Lex の新しいリソースのバージョンは $LATEST になります。詳細については、「バージョニングとエイリアス」を参照してください。
- インテントやスロットタイプはまだ作成していないため、何も表示されません。
- [Build] および [Publish] は、ボットレベルのアクティビティです。ボット全体を設定した後で、これらのアクティビティについて詳しく説明します。

次のステップ

インテントの作成

インテントの作成

次に、ユーザーが実行するアクションである `OrderPizza` インテントを必要最小限の情報で作成します。インテントのスロットタイプを追加し、後でインテントを設定します。

インテントを作成するには

1. Amazon Lex コンソールで、[Intents] (インテント) の横にあるプラス記号 (+) を選択し、[Create new intent] (新しいインテントの作成) を選択します。
2. [Create intent] ダイアログボックスに、インテントの名前 (`OrderPizza`) を入力し、[Add] を選択します。

コンソールは `OrderPizza` インテントを作成するために Amazon Lex にリクエストを送信します。この例では、スロットタイプの作成後にインテントのスロットを作成します。

次のステップ

スロットタイプの作成

スロットタイプの作成

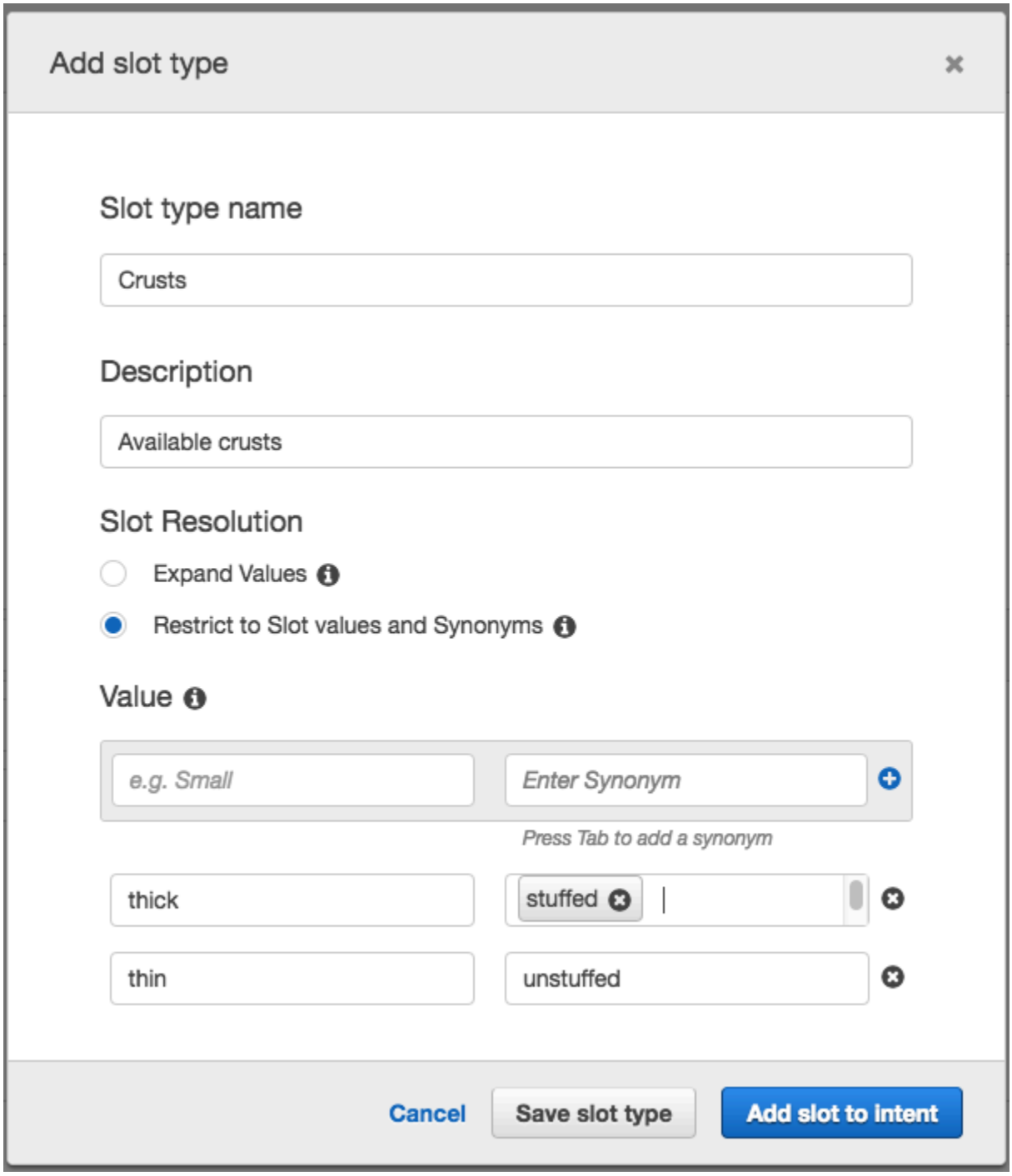
`OrderPizza` インテントで使用するスロットタイプ (パラメータ値) を作成します。

スロットタイプを作成するには

1. 左のメニューで、[Slot types] の横にあるプラス記号 (+) を選択します。
2. [Add slot type] ダイアログボックスで、以下を追加します。

   - [Slot type name] – Crusts
   - [Description] – Available crusts
   - [Restrict to Slot values and Synonyms] を選択します。
   - [Value] (値) - タイプ **`thick`**。タブを押して、[Synonym] フィールドに「**`stuffed`**」と入力します。プラス記号 (+) を選択します。「**`thin`**」と入力し、再びプラス記号 (+) を選択します。

   ダイアログは以下の画像のようになります。



3. [Add slot to intent] を選択します。
4. [Intent] ページで、[Required] を選択します。スロット名を「**slotOne**」から「**crust**」に変更します。プロンプトを **What kind of crust would you like?** に変更します。
5. 次の表の値を使用して Step 1～Step 4 を繰り返します。

名前	説明	値	スロット名	プロンプト
Sizes	Available sizes	small、medium、large	size	What size pizza?
PizzaKind	Available pizzas	veg、cheese	pizzaKind	Do you want a veg or cheese pizza?

次のステップ

インテントの設定

インテントの設定

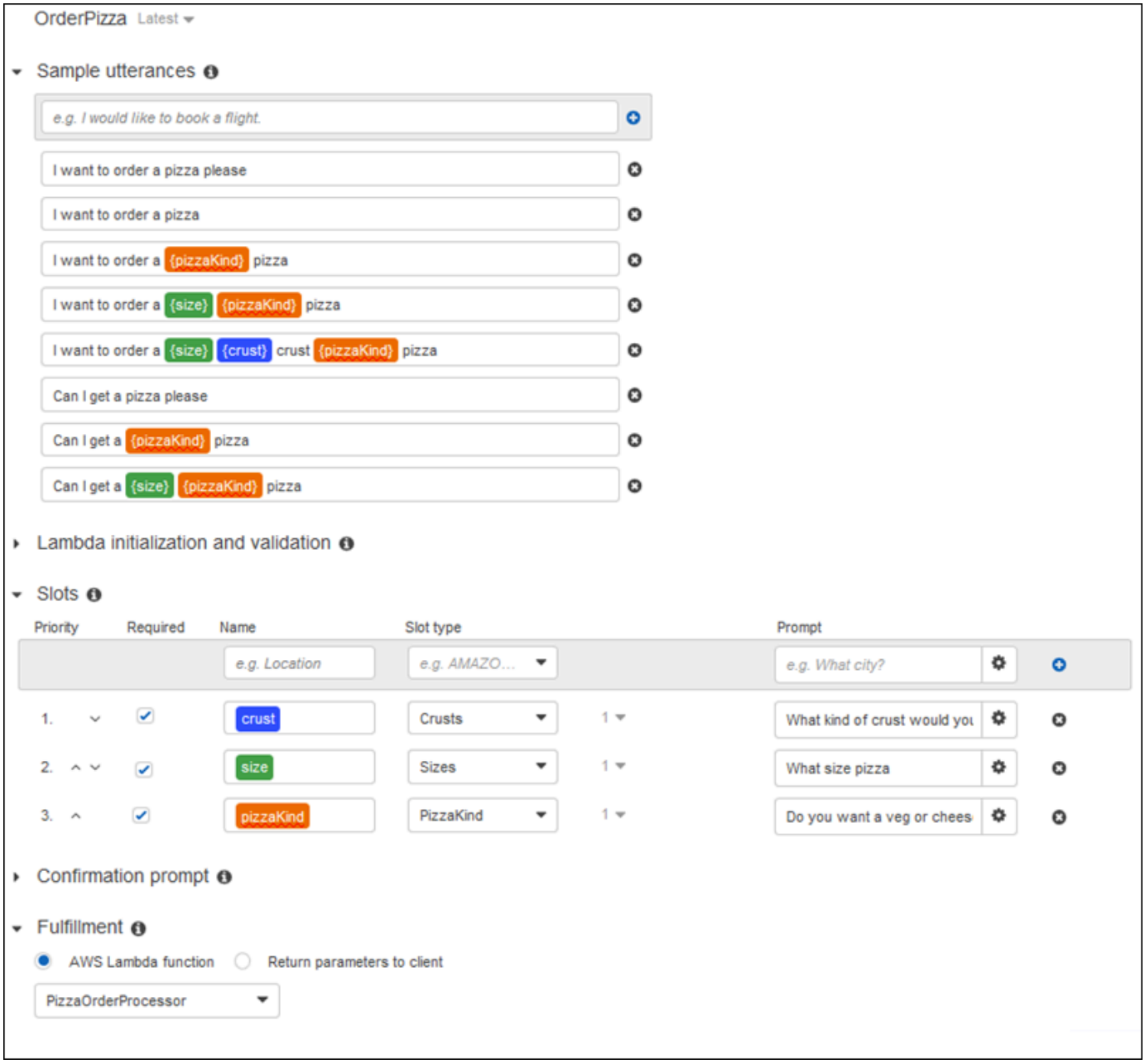
ユーザーのピザ注文のリクエストを処理するように `OrderPizza` インテントを設定します。

インテントを設定するには

- [OrderPizza] 設定ページで、次のようにインテントを設定します。
  - Sample utterances – 以下の文字列を入力します。中括弧 {} にはスロット名が入ります。
    - ピザを注文したいです
    - ピザを注文します
    - {pizzaKind} ピザを注文します
    - {size} {pizzaKind} ピザを注文します
    - {size} {crust} クラストの {pizzaKind} ピザをください
    - ピザをください
    - {pizzaKind} ピザをください
    - {size} {pizzaKind} ピザをください
  - Lambda initialization and validation – デフォルト設定のままにします。
  - Confirmation prompt – デフォルト設定のままにします。
  - フルフィルメント - 以下のタスクを実行します。
    - AWS Lambda 関数 を選択します。
    - **`PizzaOrderProcessor`** を選択します。

- [Add permission to Lambda function] (Lambda 関数に許可を追加する) ダイアログボックスが表示されている場合は、[OK] を選択して `OrderPizza` インテントに Lambda 関数 `PizzaOrderProcessor` を呼び出すアクセス許可を付与します。
- [None] は選択したままにします。

インテントは次のようになります。



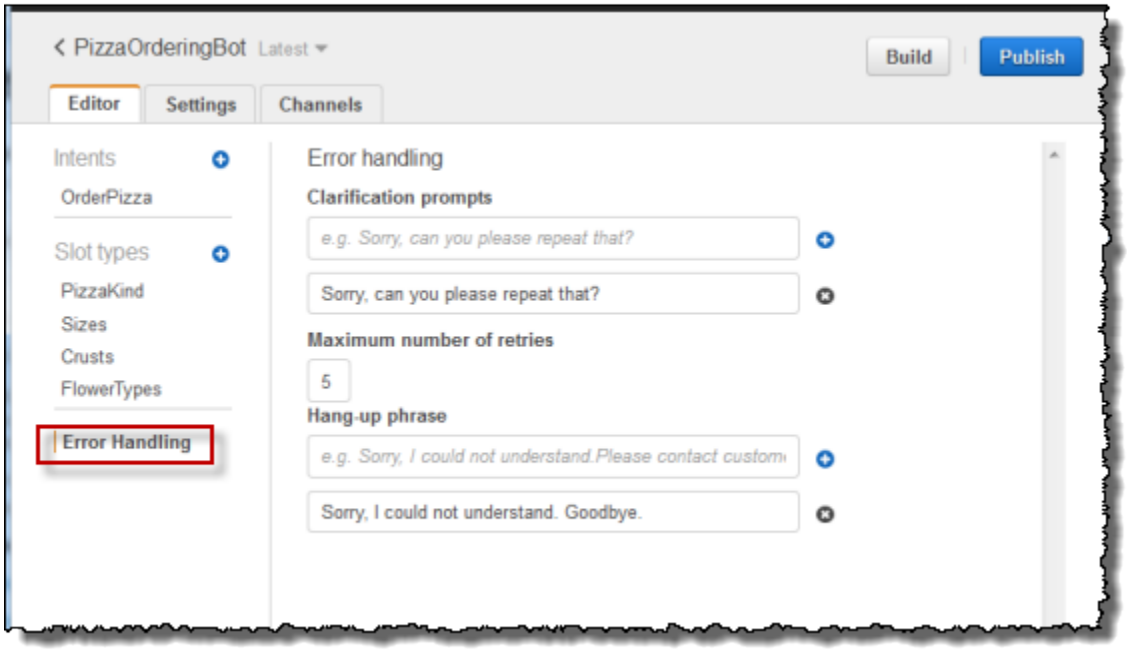
次のステップ

[ボットの設定]

ボットの設定

PizzaOrderingBot ボットのエラー処理を設定します。

1. PizzaOrderingBot ボットに移動します。[エディタ] を選択し、次の画像のように [エラー処理] を選択します。



2. [Editor] タブを使用してボットのエラー処理を設定します。

   - [Clarification Prompts] (明確化プロンプト) で指定する情報は、ボットの [clarificationPrompt] 設定にマッピングされます。

     Amazon Lex がユーザーのインテントを判断できない場合、サービスはこのメッセージを付けてレスポンスを返します。
   - [Hang-up phrase] (中断フレーズ) で指定する情報は、ボットの [abortStatement] 設定にマッピングされます。

     一連のリクエストを受信した後に、サービスがユーザーのインテントを判断できない場合、Amazon Lex はこのメッセージを付けてレスポンスを返します。

   デフォルト値はそのままにしておきます。

次のステップ

[ステップ 3: ボットを構築してテストする]

ステップ 3: ボットを構築してテストする

ボットを構築およびテストして、ボットが動作することを確認します。

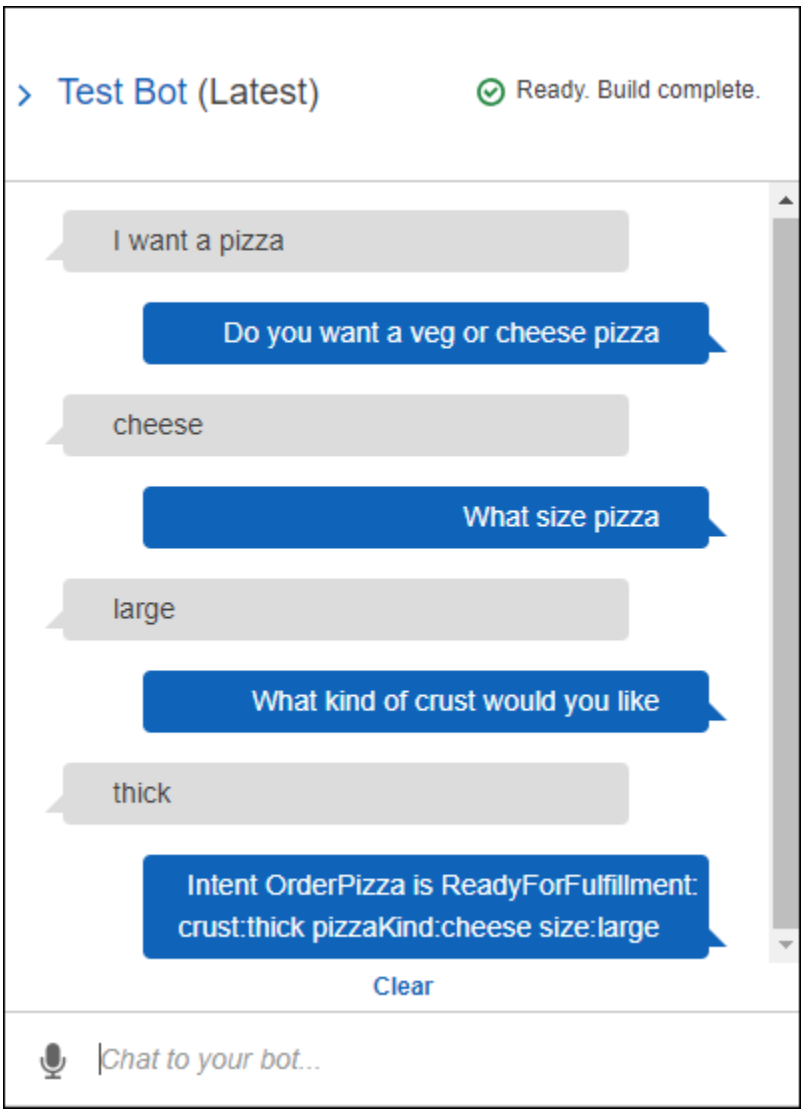
ボットを構築してテストするには

1. `PizzaOrderingBot` ボットを構築するには、[Build] を選択します。

   Amazon Lex はボットの機械学習モデルを構築します。ボットをテストする場合、コンソールではランタイム API を使用してユーザー入力を Amazon Lex に返します。Amazon Lex は機械学習モデルを使用してそのユーザー入力を解釈します。

   構築が完了するまでには時間がかかることがあります。
2. ボットをテストするには、[Test Bot] (ボットのテスト) ウィンドウで、Amazon Lex ボットとの通信を開始します。

   - 例えば、次のように言うか、入力します。



- `OrderPizza` インテントで設定したサンプル発話を使用してボットをテストします。例えば、以下は `PizzaOrder` インテントに設定したサンプル発話の 1 つです。

  ```
  I want a {size} {crust} crust {pizzaKind} pizza
  ```

  これをテストするには、次のように入力します。

  ```
  I want a large thin crust cheese pizza
  ```

「ピザを注文します」と入力すると、Amazon Lex はそのインテント (`OrderPizza`) を検出します。次に、Amazon Lex からスロット情報の入力を求められます。

すべてのスロット情報を指定すると、Amazon Lex はインテントに設定した Lambda 関数を呼び出します。

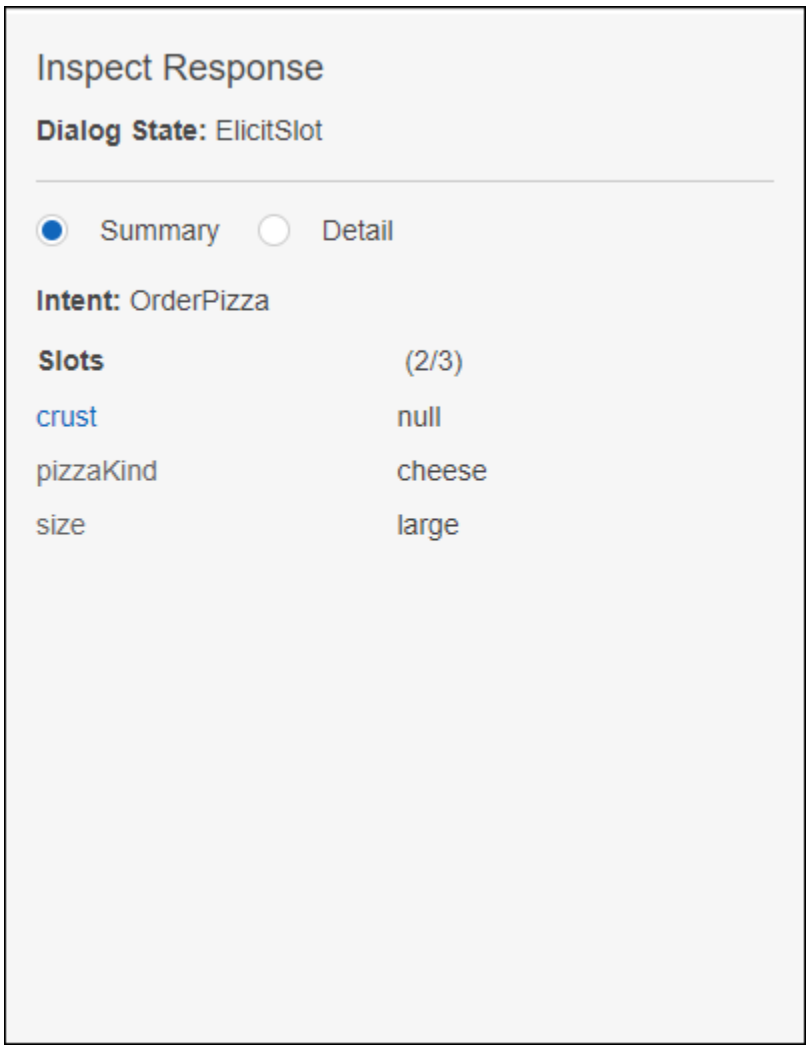
Lambda 関数はメッセージ (「かしこまりました。お客様の ... を注文いたしました」) を Amazon Lex に返し、Amazon Lex によって返信されます。

レスポンスの検査

チャットウィンドウの下のペインで、Amazon Lex からのレスポンスを検査できます。このペインには、ボットとのやり取りに応じて変わるボットの状態に関する全体情報が表示されます。ペインの情報は、オペレーションの現在の状態を示します。

- Dialog State – ユーザーとの会話の現在の状態。`ElicitIntent`、`ElicitSlot`、`ConfirmIntent`、`Fulfilled` のいずれかになります。

- Summary – ダイアログの簡素化されたビューであり、処理対象のインテントのスロット値が表示されます。これにより、情報フローを追跡できます。インテント名、スロット総数と入力済みスロット数、すべてのスロットおよび関連値の一覧が表示されます。次の画像を参照してください。



- Detail – chatbot の未加工の JSON レスポンスを表示します。これにより、chatbot のテストとデバッグを行う際に、ボットとのやり取りやダイアログの現在の状態をより深く把握できます。チャットウィンドウに入力すると、検査ペインに PostText オペレーションからの JSON レスポンスが表示されます。チャットウィンドウに話しかけると、検査ペインに PostContent オペレーションからのレスポンスヘッダーが表示されます。次の画像を参照してください。

次のステップ

[ステップ 4 (オプション): クリーンアップする]

ステップ 4 (オプション): クリーンアップする

作成したリソースを削除してアカウントをクリーンアップし、作成したリソースに対して料金が発生しないようにします。

削除できるのは、使用中ではないリソースだけです。例えば、インテントによって参照されているスロットは削除できません。ボットによって参照されているインテントは削除できません。

次の順序でリソースを削除します。

- ボットを削除して、インテントのリソースを解放します。

- インテントを削除して、スロットタイプのリソースを解放します。
- 最後にスロットタイプを削除します。

アカウントをクリーンアップするには

1. AWS Management Console にサインインし、Amazon Lex コンソール (https://console.aws.amazon.com/lex/) を開きます。
2. ボットのリストから [PizzaOrderingBot] を選択します。
3. ボットを削除するには、[Delete]、[Continue] の順に選択します。
4. 左のペインで [Intents] を選択します。
5. インテントのリストで、[OrderPizza] を選択します。
6. インテントを削除するには、[Delete]、[Continue] の順に選択します。
7. 左のメニューで、[Slot types] を選択します。
8. スロットタイプのリストで、[Crusts] を選択します。
9. スロットタイプを削除するには、[Delete]、[Continue] の順に選択します。
10. [Sizes] と [PizzaKind] の各スロットタイプで、「Step 8」と「Step 9」を繰り返します。

これで、作成したすべてのリソースが削除され、アカウントがクリーンアップされました。

次のステップ

- バージョンを発行してエイリアスを作成する
- AWS Command Line Interface を使って Amazon Lex ボットを作成する

演習 3: バージョンを発行してエイリアスを作成する

「開始方法」の演習 1 と 2 で、ボットを作成してテストしました。この演習では、以下のことを行います。

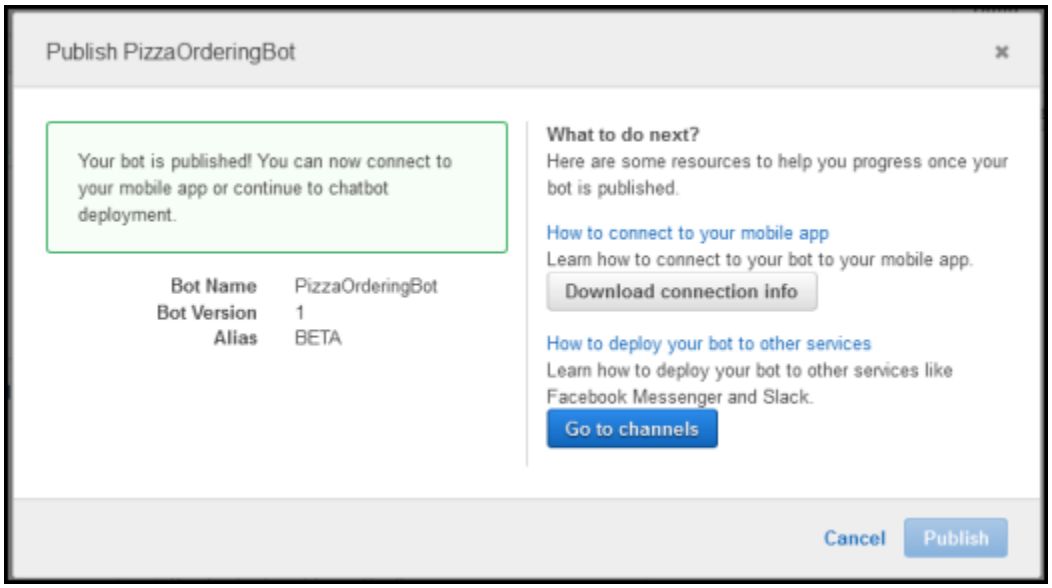
- ボットの新しいバージョンを発行します。Amazon Lex では、$LATEST バージョンのスナップショットを作成して新しいバージョンを発行します。
- 新しいバージョンを指すエイリアスを作成します。

バージョニングとエイリアスの詳細については、「バージョニングとエイリアス」を参照してください。

以下の手順に従って、この演習で作成したボットのバージョンを発行します。

1. Amazon Lex コンソールで、作成したボットの 1 つを選択します。

   コンソールで、ボット名の横にボットバージョンとして $LATEST と表示されていることを確認します。
2. [Publish] (発行) を選択します。
3. [Publish ***botname***] (ボット名の発行) ウィザードで、エイリアス (**BETA**) を指定し、[発行] を選択します。
4. 次の画像のように、Amazon Lex コンソールでボット名の横に新しいバージョンが表示されていることを確認します。



これでバージョンとエイリアスが発行されてボットが動作するようになったので、ボットをデプロイできます (モバイルアプリケーションでデプロイするか、ボットを Facebook Messenger と統合します)。例については、「Amazon Lex ボットと Facebook Messenger の統合」を参照してください。

ステップ 4: ご利用開始にあたって (AWS CLI)

このステップでは、AWS CLI を使用して Amazon Lex ボットを作成、テスト、および変更します。以下の演習を行うには、CLI の使い方を知っている必要があります。また、テキストエディタが必要

です。詳細については、「ステップ 2: を設定する AWS Command Line Interface」を参照してください。

- 演習 1 - Amazon Lex ボットを作成してテストします。この演習では、カスタムスロットタイプ、インテント、およびボットを作成するために必要なすべての JSON オブジェクトを提供します。詳細については、「Amazon Lex: 仕組み」を参照してください。
- 演習 2 - 演習 1 で作成したボットを更新し、新しいサンプル発話を追加します。Amazon Lex でサンプル発話を使用し、ボットの機械学習モデルを構築します。
- 演習 3 - 演習 1 で作成したボットを更新し、ユーザー入力を検証してインテントを達成するための Lambda 関数を追加します。
- 演習 4 - 演習 1 で作成したスロットタイプ、インテント、およびボットリソースのバージョンを発行します。バージョンは、リソースの変更できないスナップショットです。
- 演習 5 - 演習 1 で作成したボットのエイリアスを作成します。
- 演習 6 - 演習 1 で作成したスロットタイプ、インテント、ボットおよび演習 5 で作成したエイリアスを削除し、アカウントをクリーンアップします。

トピック

演習 1: Amazon Lex ボットを作成する (AWS CLI)

通常、ボットを作成するときは、以下のことを行います。

1. スロットタイプを作成し、ボットで扱う情報を定義します。
2. インテントを作成し、ボットでサポートするユーザーアクションを定義します。前に作成したカスタムスロットタイプを使用し、インテントに必要なスロット (パラメータ) を定義します。
3. 定義したインテントを使用するボットを作成します。

この演習では、CLI を使用して新しい Amazon Lex ボットを作成してテストします。ボットの作成には、用意されている JSON 構造を使用します。この演習のコマンドを実行するには、コマンドが実行されるリージョンを確認しておく必要があります。リージョンのリストについては、「モデル構築のクォータ」を参照してください。

トピック

- ステップ 1: サービスにリンクされたロールを作成する (AWS CLI)
- ステップ 2: カスタムスロットタイプを作成する (AWS CLI)
- ステップ 3: インテントを作成する (AWS CLI)
- ステップ 4: ボットを作成する (AWS CLI)
- ステップ 5: ボットをテストする (AWS CLI)

ステップ 1: サービスにリンクされたロールを作成する (AWS CLI)

Amazon Lex は、AWS Identity and Access Management サービスにリンクされたロールを引き受け、ボットに代わって AWS のサービスを呼び出します。ロールは、アカウント内で Amazon Lex ユースケースにリンクされ、アクセス権限が事前に定義されています。詳細については、「Amazon Lex のサービスリンクロールの使用」を参照してください。

コンソールで Amazon Lex ボットを作成済みである場合、サービスにリンクされたロールは自動的に作成されています。「ステップ 2: カスタムスロットタイプを作成する (AWS CLI)」へ進んでください。

サービスリンクロールの作成 (AWS CLI)

1. AWS CLI で、次のコマンドを入力します。

```
aws iam create-service-linked-role --aws-service-name lex.amazonaws.com
```

2. 次のコマンドを使用してポリシーをチェックします。

```
aws iam get-role --role-name AWSServiceRoleForLexBots
```

レスポンスは次のとおりです。

```
{
    "Role": {
        "AssumeRolePolicyDocument": {
```

```
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Action": "sts:AssumeRole",
                    "Effect": "Allow",
                    "Principal": {
                        "Service": "lex.amazonaws.com"
                    }
                }
            ]
        },
        "RoleName": "AWSServiceRoleForLexBots",
        "Path": "/aws-service-role/lex.amazonaws.com/",
        "Arn": "arn:aws:iam::account-id:role/aws-service-role/lex.amazonaws.com/
AWSServiceRoleForLexBots"
}
```

次のステップ

ステップ 2: カスタムスロットタイプを作成する (AWS CLI)

ステップ 2: カスタムスロットタイプを作成する (AWS CLI)

カスタムスロットタイプを作成し、注文できる花の種類を列挙値とします。次のステップで `OrderFlowers` インテントを作成するときに、このタイプを使用します。スロットタイプは、インテントのスロット (パラメータ) に指定できる値を定義します。

この演習のコマンドを実行するには、コマンドが実行されるリージョンを確認しておく必要があります。リージョンのリストについては、「 モデル構築のクォータ 」を参照してください。

カスタムスロットタイプを作成するには (AWS CLI)

1. **`FlowerTypes.json`** という名前のテキストファイルを作成します。このテキストファイル内に FlowerTypes.json の JSON コードをコピーします。
2. AWS CLI を使用して PutSlotType オペレーションを呼び出し、スロットタイプを作成します。例は、Unix、Linux、および macOS 用にフォーマットされています。Windows の場合は、各行末のバックスラッシュ (\) Unix 連結文字をキャレット (^) に置き換えてください。

```
aws lex-models put-slot-type \
    --region region \
    --name FlowerTypes \
```

```
    --cli-input-json file://FlowerTypes.json
```

サーバーからのレスポンスは次のとおりです。

```
{
    "enumerationValues": [
        {
            "value": "tulips"
        },
        {
            "value": "lilies"
        },
        {
            "value": "roses"
        }
    ],
    "name": "FlowerTypes",
    "checksum": "checksum",
    "version": "$LATEST",
    "lastUpdatedDate": timestamp,
    "createdDate": timestamp,
    "description": "Types of flowers to pick up"
}
```

次のステップ

ステップ 3: インテントを作成する (AWS CLI)

FlowerTypes.json

次のコードは、FlowerTypes カスタムスロットタイプを作成するために必要な JSON データです。

```
{
    "enumerationValues": [
        {
            "value": "tulips"
        },
        {
            "value": "lilies"
        },
        {
```

```
            "value": "roses"
        }
    ],
    "name": "FlowerTypes",
    "description": "Types of flowers to pick up"
}
```

ステップ 3: インテントを作成する (AWS CLI)

OrderFlowersBot ボットのインテントを作成し、3 つのスロット (パラメータ) を指定します。スロットを使用することで、ボットはインテントを達成できます。

- FlowerType は、注文できる花の種類を指定するカスタムスロットタイプです。
- AMAZON.DATE および AMAZON.TIME は、花の配送日時をユーザーから取得するための組み込みスロットタイプです。

この演習のコマンドを実行するには、コマンドが実行されるリージョンを確認しておく必要があります。リージョンのリストについては、「 モデル構築のクォータ 」を参照してください。

OrderFlowers インテントを作成するには (AWS CLI)

1. **OrderFlowers.json** という名前のテキストファイルを作成します。このテキストファイル内に OrderFlowers.json の JSON コードをコピーします。
2. AWS CLI で、PutIntent オペレーションを呼び出してインテントを作成します。例は、Unix、Linux、および macOS 用にフォーマットされています。Windows の場合は、各行末のバックスラッシュ (\) Unix 連結文字をキャレット (^) に置き換えてください。

   ```
   aws lex-models put-intent \
      --region region \
      --name OrderFlowers \
      --cli-input-json file://OrderFlowers.json
   ```

   サーバーは以下のように応答します。

   ```
   {
       "confirmationPrompt": {
           "maxAttempts": 2,
           "messages": [
               {
   ```

```
                "content": "Okay, your {FlowerType} will be ready for pickup by
{PickupTime} on {PickupDate}.  Does this sound okay?",
                "contentType": "PlainText"
            }
        ]
    },
    "name": "OrderFlowers",
    "checksum": "checksum",
    "version": "$LATEST",
    "rejectionStatement": {
        "messages": [
            {
                "content": "Okay, I will not place your order.",
                "contentType": "PlainText"
            }
        ]
    },
    "createdDate": timestamp,
    "lastUpdatedDate": timestamp,
    "sampleUtterances": [
        "I would like to pick up flowers",
        "I would like to order some flowers"
    ],
    "slots": [
        {
            "slotType": "AMAZON.TIME",
            "name": "PickupTime",
            "slotConstraint": "Required",
            "valueElicitationPrompt": {
                "maxAttempts": 2,
                "messages": [
                    {
                        "content": "Pick up the {FlowerType} at what time on
{PickupDate}?",
                        "contentType": "PlainText"
                    }
                ]
            },
            "priority": 3,
            "description": "The time to pick up the flowers"
        },
        {
            "slotType": "FlowerTypes",
            "name": "FlowerType",
```

```
            "slotConstraint": "Required",
            "valueElicitationPrompt": {
                "maxAttempts": 2,
                "messages": [
                    {
                        "content": "What type of flowers would you like to
 order?",
                        "contentType": "PlainText"
                    }
                ]
            },
            "priority": 1,
            "slotTypeVersion": "$LATEST",
            "sampleUtterances": [
                "I would like to order {FlowerType}"
            ],
            "description": "The type of flowers to pick up"
        },
        {
            "slotType": "AMAZON.DATE",
            "name": "PickupDate",
            "slotConstraint": "Required",
            "valueElicitationPrompt": {
                "maxAttempts": 2,
                "messages": [
                    {
                        "content": "What day do you want the {FlowerType} to be
 picked up?",
                        "contentType": "PlainText"
                    }
                ]
            },
            "priority": 2,
            "description": "The date to pick up the flowers"
        }
    ],
    "fulfillmentActivity": {
        "type": "ReturnIntent"
    },
    "description": "Intent to order a bouquet of flowers for pick up"
}
```

次のステップ

ステップ 4: ボットを作成する (AWS CLI)

OrderFlowers.json

次のコードは、OrderFlowers インテントを作成するために必要な JSON データです。

```
{
    "confirmationPrompt": {
        "maxAttempts": 2,
        "messages": [
            {
                "content": "Okay, your {FlowerType} will be ready for pickup by
 {PickupTime} on {PickupDate}.  Does this sound okay?",
                "contentType": "PlainText"
            }
        ]
    },
    "name": "OrderFlowers",
    "rejectionStatement": {
        "messages": [
            {
                "content": "Okay, I will not place your order.",
                "contentType": "PlainText"
            }
        ]
    },
    "sampleUtterances": [
        "I would like to pick up flowers",
        "I would like to order some flowers"
    ],
    "slots": [
        {
            "slotType": "FlowerTypes",
            "name": "FlowerType",
            "slotConstraint": "Required",
            "valueElicitationPrompt": {
                "maxAttempts": 2,
                "messages": [
                    {
                        "content": "What type of flowers would you like to order?",
                        "contentType": "PlainText"
                    }
```

```
                ]
            },
            "priority": 1,
            "slotTypeVersion": "$LATEST",
            "sampleUtterances": [
                "I would like to order {FlowerType}"
            ],
            "description": "The type of flowers to pick up"
        },
        {
            "slotType": "AMAZON.DATE",
            "name": "PickupDate",
            "slotConstraint": "Required",
            "valueElicitationPrompt": {
                "maxAttempts": 2,
                "messages": [
                    {
                        "content": "What day do you want the {FlowerType} to be picked
up?",
                        "contentType": "PlainText"
                    }
                ]
            },
            "priority": 2,
            "description": "The date to pick up the flowers"
        },
        {
            "slotType": "AMAZON.TIME",
            "name": "PickupTime",
            "slotConstraint": "Required",
            "valueElicitationPrompt": {
                "maxAttempts": 2,
                "messages": [
                    {
                        "content": "Pick up the {FlowerType} at what time on
{PickupDate}?",
                        "contentType": "PlainText"
                    }
                ]
            },
            "priority": 3,
            "description": "The time to pick up the flowers"
        }
    ],
```

```
    "fulfillmentActivity": {
        "type": "ReturnIntent"
    },
    "description": "Intent to order a bouquet of flowers for pick up"
}
```

ステップ 4: ボットを作成する (AWS CLI)

OrderFlowersBot ボットには、1 つのインテント (前のステップで作成した OrderFlowers インテント) があります。この演習のコマンドを実行するには、コマンドが実行されるリージョンを確認しておく必要があります。リージョンのリストについては、「 モデル構築のクォータ 」を参照してください。

> **Note**
>
> 次の AWS CLI の例は、Unix、Linux、および macOS 用にフォーマットされています。Windows の場合は、"\$LATEST" を $LATEST に変更してください。

OrderFlowersBot ボットを作成するには (AWS CLI)

1. **OrderFlowersBot.json** という名前のテキストファイルを作成します。このテキストファイル内に OrderFlowersBot.json の JSON コードをコピーします。
2. AWS CLI で、PutBot オペレーションを呼び出してボットを作成します。例は、Unix、Linux、および macOS 用にフォーマットされています。Windows の場合は、各行末のバックスラッシュ (\) Unix 連結文字をキャレット (^) に置き換えてください。

   ```
   aws lex-models put-bot \
       --region region \
       --name OrderFlowersBot \
       --cli-input-json file://OrderFlowersBot.json
   ```

   サーバーからのレスポンスは次のとおりです。ボットを作成または更新すると、status フィールドは BUILDING に設定されます。これは、ボットの使用準備が整っていないことを示します。ボットの使用準備が整ったことを確認するには、次のステップで GetBot オペレーションを使用します。

   ```
   {
       "status": "BUILDING",
   ```

```
    "intents": [
        {
            "intentVersion": "$LATEST",
            "intentName": "OrderFlowers"
        }
    ],
    "name": "OrderFlowersBot",
    "locale": "en-US",
    "checksum": "checksum",
    "abortStatement": {
        "messages": [
            {
                "content": "Sorry, I'm not able to assist at this time",
                "contentType": "PlainText"
            }
        ]
    },
    "version": "$LATEST",
    "lastUpdatedDate": timestamp,
    "createdDate": timestamp,
    "clarificationPrompt": {
        "maxAttempts": 2,
        "messages": [
            {
                "content": "I didn't understand you, what would you like to do?",
                "contentType": "PlainText"
            }
        ]
    },
    "voiceId": "Salli",
    "childDirected": false,
    "idleSessionTTLInSeconds": 600,
    "processBehavior": "BUILD",
    "description": "Bot to order flowers on the behalf of a user"
}
```

3. 新しいボットの使用準備が整っているかどうかを確認するには、次のコマンドを実行します。status フィールドから READY が返されるまで、このコマンドを繰り返します。例は、Unix、Linux、および macOS 用にフォーマットされています。Windows の場合は、各行末のバックスラッシュ (\) Unix 連結文字をキャレット (^) に置き換えてください。

```
aws lex-models get-bot \
```

```
    --region region \
    --name OrderFlowersBot \
    --version-or-alias "\$LATEST"
```

レスポンス内で status フィールドを探します。

```
{
    "status": "READY",

    ...

}
```

次のステップ

[ステップ 5: ボットをテストする (AWS CLI)]

OrderFlowersBot.json

次のコードは、Amazon Lex ボット OrderFlowers の構築に必要な JSON データを示しています。

```
{
    "intents": [
        {
            "intentVersion": "$LATEST",
            "intentName": "OrderFlowers"
        }
    ],
    "name": "OrderFlowersBot",
    "locale": "en-US",
    "abortStatement": {
        "messages": [
            {
                "content": "Sorry, I'm not able to assist at this time",
                "contentType": "PlainText"
            }
        ]
    },
    "clarificationPrompt": {
        "maxAttempts": 2,
        "messages": [
            {
```

```
                "content": "I didn't understand you, what would you like to do?",
                "contentType": "PlainText"
            }
        ]
    },
    "voiceId": "Salli",
    "childDirected": false,
    "idleSessionTTLInSeconds": 600,
    "description": "Bot to order flowers on the behalf of a user"
}
```

ステップ 5: ボットをテストする (AWS CLI)

ボットをテストするには、テキストベースまたは音声ベースのテストを使用できます。

トピック

- テキスト入力を使用してテストする (AWS CLI)
- 音声入力を使用してボットをテストする (AWS CLI)

テキスト入力を使用してテストする (AWS CLI)

テキスト入力でボットが正しく動作することを確認するには、PostText オペレーションを使用します。この演習のコマンドを実行するには、コマンドが実行されるリージョンを確認しておく必要があります。リージョンのリストについては、「ランタイム Service Quotas」を参照してください。

> **Note**
>
> 次の AWS CLI の例は、Unix、Linux、および macOS 用にフォーマットされています。Windows の場合は、"\$LATEST" を $LATEST に変更し、各行末のバックスラッシュ (\) 連結文字をキャレット (^) に置き換えてください。

テキストを使用してボットをテストするには (AWS CLI)

1. AWS CLI で、OrderFlowersBot ボットとの会話を開始します。例は、Unix、Linux、および macOS 用にフォーマットされています。Windows の場合は、各行末のバックスラッシュ (\) Unix 連結文字をキャレット (^) に置き換えてください。

   ```
   aws lex-runtime post-text \
       --region region \
   ```

```
    --bot-name OrderFlowersBot \
    --bot-alias "\$LATEST" \
    --user-id UserOne \
    --input-text "i would like to order flowers"
```

Amazon Lex は、ユーザーのインテントを認識し、次のレスポンスを返すことで会話を開始します。

```
{
    "slotToElicit": "FlowerType",
    "slots": {
        "PickupDate": null,
        "PickupTime": null,
        "FlowerType": null
    },
    "dialogState": "ElicitSlot",
    "message": "What type of flowers would you like to order?",
    "intentName": "OrderFlowers"
}
```

2. 以下のコマンドを実行して、ボットとの会話を終了します。

```
aws lex-runtime post-text \
    --region region \
    --bot-name OrderFlowersBot \
    --bot-alias "\$LATEST" \
    --user-id UserOne \
    --input-text "roses"
```

```
aws lex-runtime post-text  \
    --region region \
    --bot-name OrderFlowersBot \
    --bot-alias "\$LATEST" \
    --user-id UserOne \
    --input-text "tuesday"
```

```
aws lex-runtime post-text  \
    --region region \
    --bot-name OrderFlowersBot --bot-alias "\$LATEST" \
    --user-id UserOne \
    --input-text "10:00 a.m."
```

```
aws lex-runtime post-text  \
    --region region \
    --bot-name OrderFlowersBot \
    --bot-alias "\$LATEST" \
    --user-id UserOne \
    --input-text "yes"
```

注文を確認すると、Amazon Lex はフルフィルメントレスポンスを送信して会話を完了します。

```
{
    "slots": {
        "PickupDate": "2017-05-16",
        "PickupTime": "10:00",
        "FlowerType": "roses"
    },
    "dialogState": "ReadyForFulfillment",
    "intentName": "OrderFlowers"
}
```

次のステップ

音声入力を使用してボットをテストする (AWS CLI)

音声入力を使用してボットをテストする (AWS CLI)

音声ファイルを使用してボットをテストするには、PostContent オペレーションを使用します。音声ファイルは、Amazon Polly テキスト読み上げ機能のオペレーションを使用して生成します。

この演習のコマンドを実行するには、Amazon Lex および Amazon Polly コマンドが実行されるリージョンを確認しておく必要があります。Amazon Lex のリージョンのリストについては「ランタイム Service Quotas」を参照してください。Amazon Polly でサポートされているリージョンとエンドポイントの一覧については、Amazon Web Services 全般のリファレンスの「AWS リージョンとエンドポイント」を参照してください。

> **Note**
>
> 次の AWS CLI の例は、Unix、Linux、および macOS 用にフォーマットされています。Windows の場合は、"\$LATEST" を $LATEST に変更し、各行末のバックスラッシュ (\) 連結文字をキャレット (^) に置き換えてください。

音声入力を使用してボットをテストするには (AWS CLI)

1. AWS CLI で、Amazon Polly を使用して音声ファイルを作成します。例は、Unix、Linux、および macOS 用にフォーマットされています。Windows の場合は、各行末のバックスラッシュ (\) Unix 連結文字をキャレット (^) に置き換えてください。

```
aws polly synthesize-speech \
    --region region \
    --output-format pcm \
    --text "i would like to order flowers" \
    --voice-id "Salli" \
    IntentSpeech.mpg
```

2. 音声ファイルを Amazon Lex に送信するには、次のコマンドを実行します。Amazon Lex は、レスポンスの音声を指定の出力ファイルに保存します。

```
aws lex-runtime post-content \
    --region region \
    --bot-name OrderFlowersBot \
    --bot-alias "\$LATEST" \
    --user-id UserOne \
    --content-type "audio/l16; rate=16000; channels=1" \
    --input-stream IntentSpeech.mpg \
    IntentOutputSpeech.mpg
```

Amazon Lex は、レスポンスで最初のスロットをリクエストします。音声レスポンスは指定の出力ファイルに保存されます。

```
{
    "contentType": "audio/mpeg",
    "slotToElicit": "FlowerType",
    "dialogState": "ElicitSlot",
    "intentName": "OrderFlowers",
    "inputTranscript": "i would like to order some flowers",
    "slots": {
        "PickupDate": null,
        "PickupTime": null,
        "FlowerType": null
    },
    "message": "What type of flowers would you like to order?"
}
```

3. バラの花束を注文するには、次の音声ファイルを作成して Amazon Lex に送信します。

```
aws polly synthesize-speech \
    --region region \
    --output-format pcm \
    --text "roses" \
    --voice-id "Salli" \
    FlowerTypeSpeech.mpg
```

```
aws lex-runtime post-content \
    --region region \
    --bot-name OrderFlowersBot \
    --bot-alias "\$LATEST" \
    --user-id UserOne \
    --content-type "audio/l16; rate=16000; channels=1" \
    --input-stream FlowerTypeSpeech.mpg \
    FlowerTypeOutputSpeech.mpg
```

4. 配達日を設定するには、次の音声ファイルを作成して Amazon Lex に送信します:

```
aws polly synthesize-speech \
    --region region \
    --output-format pcm \
    --text "tuesday" \
    --voice-id "Salli" \
    DateSpeech.mpg
```

```
aws lex-runtime post-content \
    --region region \
    --bot-name OrderFlowersBot \
    --bot-alias "\$LATEST" \
    --user-id UserOne \
    --content-type "audio/l16; rate=16000; channels=1" \
    --input-stream DateSpeech.mpg \
    DateOutputSpeech.mpg
```

5. 配送時間を設定するには、次の音声ファイルを作成して Amazon Lex に送信します:

```
aws polly synthesize-speech \
    --region region \
    --output-format pcm \
```

```
    --text "10:00 a.m." \
    --voice-id "Salli" \
    TimeSpeech.mpg
```

```
aws lex-runtime post-content \
    --region region \
    --bot-name OrderFlowersBot \
    --bot-alias "\$LATEST" \
    --user-id UserOne \
    --content-type "audio/l16; rate=16000; channels=1" \
    --input-stream TimeSpeech.mpg \
    TimeOutputSpeech.mpg
```

6. 配達を確認するには、次の音声ファイルを作成して Amazon Lex に送信します。

```
aws polly synthesize-speech \
    --region region \
    --output-format pcm \
    --text "yes" \
    --voice-id "Salli" \
    ConfirmSpeech.mpg
```

```
aws lex-runtime post-content \
    --region region \
    --bot-name OrderFlowersBot \
    --bot-alias "\$LATEST" \
    --user-id UserOne \
    --content-type "audio/l16; rate=16000; channels=1" \
    --input-stream ConfirmSpeech.mpg \
    ConfirmOutputSpeech.mpg
```

配達を確認すると、Amazon Lex からインテントの達成を確認するレスポンスが送信されます。

```
{
    "contentType": "text/plain;charset=utf-8",
    "dialogState": "ReadyForFulfillment",
    "intentName": "OrderFlowers",
    "inputTranscript": "yes",
    "slots": {
        "PickupDate": "2017-05-16",
        "PickupTime": "10:00",
```

```
        "FlowerType": "roses"
    }
}
```

次のステップ

演習 2: 新しい発話を追加する (AWS CLI)

演習 2: 新しい発話を追加する (AWS CLI)

ユーザーからのリクエストを認識するために Amazon Lex が使用する機械学習モデルを向上するには、別のサンプル発話をボットに追加します。

新しい発話を追加するには 4 つのステップを使用します。

1. GetIntent オペレーションを使用して Amazon Lex からインテントを取得します。
2. インテントを更新します。
3. PutIntent オペレーションを使用して、更新したインテントを Amazon Lex に送り返します。
4. GetBot オペレーションと PutBot オペレーションを使用して、このインテントを使用するすべてのボットを再構築します。

この演習のコマンドを実行するには、コマンドが実行されるリージョンを確認しておく必要があります。リージョンのリストについては、「 モデル構築のクォータ 」を参照してください。

GetIntent オペレーションからのレスポンスには、インテントの特定のリビジョンを識別する checksum というフィールドが含まれています。PutIntent オペレーションを使用してインテントを更新するときに、このチェックサムの値を指定する必要があります。指定しないと、次のエラーメッセージが表示されます。

```
An error occurred (PreconditionFailedException) when calling
the PutIntent operation: Intent intent name already exists.
If you are trying to update intent name you must specify the
checksum.
```

> **Note**
>
> 次の AWS CLI の例は、Unix、Linux、および macOS 用にフォーマットされています。Windows の場合は、"\$LATEST" を $LATEST に変更し、各行末のバックスラッシュ (\) 連結文字をキャレット (^) に置き換えてください。

`OrderFlowers` インテントを更新するには (AWS CLI)

1. AWS CLI で、Amazon Lex からインテントを取得します。Amazon Lex は **`OrderFlowers-V2.json`.** というファイルにこの出力を送信します。

   ```
   aws lex-models get-intent \
       --region region \
       --name OrderFlowers \
       --intent-version "\$LATEST" > OrderFlowers-V2.json
   ```

2. テキストエディタで **`OrderFlowers-V2.json`** を開きます。

   1. `createdDate`、`lastUpdatedDate`、`version` の各フィールドを見つけて削除します。
   2. `sampleUtterances` フィールドに以下を追加します。

      ```
      I want to order flowers
      ```

   3. ファイルを保存します。

3. 次のコマンドを使用して、更新したインテントを Amazon Lex に送信します。

   ```
   aws lex-models put-intent  \
       --region region \
       --name OrderFlowers \
       --cli-input-json file://OrderFlowers-V2.json
   ```

   Amazon Lex から次のレスポンスが送信されます。

   ```
   {
       "confirmationPrompt": {
           "maxAttempts": 2,
           "messages": [
               {
   ```

```
                "content": "Okay, your {FlowerType} will be ready for pickup by
{PickupTime} on {PickupDate}.  Does this sound okay?",
                "contentType": "PlainText"
            }
        ]
    },
    "name": "OrderFlowers",
    "checksum": "checksum",
    "version": "$LATEST",
    "rejectionStatement": {
        "messages": [
            {
                "content": "Okay, I will not place your order.",
                "contentType": "PlainText"
            }
        ]
    },
    "createdDate": timestamp,
    "lastUpdatedDate": timestamp,
    "sampleUtterances": [
        "I would like to pick up flowers",
        "I would like to order some flowers",
        "I want to order flowers"
    ],
    "slots": [
        {
            "slotType": "AMAZON.TIME",
            "name": "PickupTime",
            "slotConstraint": "Required",
            "valueElicitationPrompt": {
                "maxAttempts": 2,
                "messages": [
                    {
                        "content": "Pick up the {FlowerType} at what time on
{PickupDate}?",
                        "contentType": "PlainText"
                    }
                ]
            },
            "priority": 3,
            "description": "The time to pick up the flowers"
        },
        {
            "slotType": "FlowerTypes",
```

```
            "name": "FlowerType",
            "slotConstraint": "Required",
            "valueElicitationPrompt": {
                "maxAttempts": 2,
                "messages": [
                    {
                        "content": "What type of flowers would you like to
 order?",
                        "contentType": "PlainText"
                    }
                ]
            },
            "priority": 1,
            "slotTypeVersion": "$LATEST",
            "sampleUtterances": [
                "I would like to order {FlowerType}"
            ],
            "description": "The type of flowers to pick up"
        },
        {
            "slotType": "AMAZON.DATE",
            "name": "PickupDate",
            "slotConstraint": "Required",
            "valueElicitationPrompt": {
                "maxAttempts": 2,
                "messages": [
                    {
                        "content": "What day do you want the {FlowerType} to be
 picked up?",
                        "contentType": "PlainText"
                    }
                ]
            },
            "priority": 2,
            "description": "The date to pick up the flowers"
        }
    ],
    "fulfillmentActivity": {
        "type": "ReturnIntent"
    },
    "description": "Intent to order a bouquet of flowers for pick up"
}
```

インテントを更新したので、このインテントを使用するすべてのボットを再構築します。

OrderFlowersBot ボットを再構築するには (AWS CLI)

1. AWS CLI で次のコマンドを使用し、OrderFlowersBot ボットの定義を取得してファイルに保存します。

```
aws lex-models get-bot \
    --region region \
    --name OrderFlowersBot \
    --version-or-alias "\$LATEST" > OrderFlowersBot-V2.json
```

2. テキストエディタで **OrderFlowersBot-V2.json** を開きます。createdDate、lastUpdatedDate、status、version の各フィールドを削除します。
3. テキストエディタで、ボットの定義に次の行を追加します。

```
"processBehavior": "BUILD",
```

4. AWS CLI で次のコマンドを実行し、ボットの新しいリビジョンを構築します。

```
aws lex-models put-bot \
    --region region \
    --name OrderFlowersBot \
    --cli-input-json file://OrderFlowersBot-V2.json
```

サーバーからのレスポンスは次のとおりです。

```
{
    "status": "BUILDING",
    "intents": [
        {
            "intentVersion": "$LATEST",
            "intentName": "OrderFlowers"
        }
    ],
    "name": "OrderFlowersBot",
    "locale": "en-US",
    "checksum": "checksum",
    "abortStatement": {
        "messages": [
            {
```

```
                "content": "Sorry, I'm not able to assist at this time",
                "contentType": "PlainText"
            }
        ]
    },
    "version": "$LATEST",
    "lastUpdatedDate": timestamp,
    "createdDate": timestamp
    "clarificationPrompt": {
        "maxAttempts": 2,
        "messages": [
            {
                "content": "I didn't understand you, what would you like to do?",
                "contentType": "PlainText"
            }
        ]
    },
    "voiceId": "Salli",
    "childDirected": false,
    "idleSessionTTLInSeconds": 600,
    "description": "Bot to order flowers on the behalf of a user"
}
```

次のステップ

演習 3: Lambda 関数を追加する (AWS CLI)

演習 3: Lambda 関数を追加する (AWS CLI)

ユーザー入力を検証し、ユーザーのインテントを達成する Lambda 関数をボットに追加します。

Lambda 表現を追加するには 5 つのステップを使用します。

1. Lambda の AddPermission 関数を使用して `OrderFlowers` インテントを有効にし、Lambda の Invoke オペレーションを呼び出します。
2. GetIntent オペレーションを使用して Amazon Lex からインテントを取得します。
3. Lambda 関数を追加してインテントを更新します。
4. PutIntent オペレーションを使用して、更新したインテントを Amazon Lex に送り返します。
5. GetBot オペレーションと PutBot オペレーションを使用して、このインテントを使用するすべてのボットを再構築します。

この演習のコマンドを実行するには、コマンドが実行されるリージョンを確認しておく必要があります。リージョンのリストについては、「モデル構築のクォータ」を参照してください。

InvokeFunction アクセス権限を追加する前に Lambda 関数をインテントに追加すると、次のエラーメッセージが表示されます。

```
An error occurred (BadRequestException) when calling the
PutIntent operation: Lex is unable to access the Lambda
function Lambda function ARN in the context of intent
intent ARN.  Please check the resource-based policy on
the function.
```

GetIntent オペレーションからのレスポンスには、インテントの特定のリビジョンを識別する checksum というフィールドが含まれています。PutIntent オペレーションを使用してインテントを更新するときに、このチェックサムの値を指定する必要があります。指定しないと、次のエラーメッセージが表示されます。

```
An error occurred (PreconditionFailedException) when calling
the PutIntent operation: Intent intent name already exists.
If you are trying to update intent name you must specify the
checksum.
```

この演習では、「演習 1: 設計図を使用して Amazon Lex ボットを作成する (コンソール)」の Lambda 関数を使用します。この Lambda 関数を作成する手順については、「ステップ 3: Lambda 関数を作成する (コンソール)」を参照してください。

> **Note**
>
> 次の AWS CLI の例は、Unix、Linux、および macOS 用にフォーマットされています。Windows の場合は、"\$LATEST" を $LATEST に変更してください。

Lambda 関数をインテントに追加するには

1. AWS CLI で InvokeFunction アクセス権限を OrderFlowers インテントに追加します。

```
aws lambda add-permission \
    --region region \
    --function-name OrderFlowersCodeHook \
    --statement-id LexGettingStarted-OrderFlowersBot \
    --action lambda:InvokeFunction \
    --principal lex.amazonaws.com \
    --source-arn "arn:aws:lex:region:account ID:intent:OrderFlowers:*"
    --source-account account ID
```

Lambda から次のレスポンスが送信されます。

```
{
    "Statement": "{\"Sid\":\"LexGettingStarted-OrderFlowersBot\",
      \"Resource\":\"arn:aws:lambda:region:account ID:function:OrderFlowersCodeHook
\",
      \"Effect\":\"Allow\",
      \"Principal\":{\"Service\":\"lex.amazonaws.com\"},
      \"Action\":[\"lambda:InvokeFunction\"],
      \"Condition\":{\"StringEquals\":
        {\"AWS:SourceAccount\": \"account ID\"},
        {\"AWS:SourceArn\":
          \"arn:aws:lex:region:account ID:intent:OrderFlowers:*\"}}}"
}
```

2. Amazon Lex からインテントを取得します。Amazon Lex は **OrderFlowers-V3.json** というファイルにこの出力を送信します。

```
aws lex-models get-intent \
    --region region \
    --name OrderFlowers \
    --intent-version "\$LATEST" > OrderFlowers-V3.json
```

3. テキストエディターで **OrderFlowers-V3.json** ファイルを開きます。

   1. `createdDate`、`lastUpdatedDate`、`version` の各フィールドを見つけて削除します。
   2. `fulfillmentActivity` フィールドを更新します。

```
    "fulfillmentActivity": {
        "type": "CodeHook",
        "codeHook": {
```

```
            "uri": "arn:aws:lambda:region:account
ID:function:OrderFlowersCodeHook",
            "messageVersion": "1.0"
        }
    }
```

3. ファイルを保存します。

4. AWS CLI で、更新したインテントを Amazon Lex: に送信します。

```
aws lex-models put-intent \
    --region region \
    --name OrderFlowers \
    --cli-input-json file://OrderFlowers-V3.json
```

インテントを更新したので、ボットを再構築します。

OrderFlowersBot ボットを再構築するには

1. AWS CLI で、OrderFlowersBot ボットの定義を取得してファイルに保存します。

```
aws lex-models get-bot \
    --region region \
    --name OrderFlowersBot \
    --version-or-alias "\$LATEST" > OrderFlowersBot-V3.json
```

2. テキストエディタで **OrderFlowersBot-V3.json** を開きます。createdDate、lastUpdatedDate、status、version の各フィールドを削除します。

3. テキストエディタで、ボットの定義に次の行を追加します。

```
"processBehavior": "BUILD",
```

4. AWS CLI で、ボットの新しいリビジョンを構築します。

```
aws lex-models put-bot \
    --region region \
    --name OrderFlowersBot \
    --cli-input-json file://OrderFlowersBot-V3.json
```

サーバーからのレスポンスは次のとおりです。

```
{
    "status": "READY",
    "intents": [
        {
            "intentVersion": "$LATEST",
            "intentName": "OrderFlowers"
        }
    ],
    "name": "OrderFlowersBot",
    "locale": "en-US",
    "checksum": "checksum",
    "abortStatement": {
        "messages": [
            {
                "content": "Sorry, I'm not able to assist at this time",
                "contentType": "PlainText"
            }
        ]
    },
    "version": "$LATEST",
    "lastUpdatedDate": timestamp,
    "createdDate": timestamp,
    "clarificationPrompt": {
        "maxAttempts": 2,
        "messages": [
            {
                "content": "I didn't understand you, what would you like to do?",
                "contentType": "PlainText"
            }
        ]
    },
    "voiceId": "Salli",
    "childDirected": false,
    "idleSessionTTLInSeconds": 600,
    "description": "Bot to order flowers on the behalf of a user"
}
```

次のステップ

[演習 4: バージョンを発行する (AWS CLI)]

演習 4: バージョンを発行する (AWS CLI)

次に演習 1 で作成したボットのバージョンを作成します。バージョンは、ボットのスナップショットです。一度作成したバージョンは変更できません。ボットのバージョンで更新できるのは $LATEST バージョンのみです。バージョンの詳細については、「バージョニングとエイリアス」を参照してください。

ボットのバージョンを発行する前に、そのバージョンで使用しているインテントを発行する必要があります。同様に、これらのインテントで参照しているスロットタイプを発行する必要があります。通常、ボットのバージョンを発行するには、以下の操作を行ないます。

1. CreateSlotTypeVersion オペレーションを使用してスロットタイプのバージョンを発行します。
2. CreateIntentVersion オペレーションを使用してインテントのバージョンを発行します。
3. CreateBotVersion オペレーションを使用してボットのバージョンを発行します。

この演習のコマンドを実行するには、コマンドが実行されるリージョンを確認しておく必要があります。リージョンのリストについては、「 モデル構築のクォータ 」を参照してください。

トピック

- ステップ 1: スロットタイプを発行する (AWS CLI)
- ステップ 2: インテントを発行する (AWS CLI)
- ステップ 3: ボットを発行する (AWS CLI)

ステップ 1: スロットタイプを発行する (AWS CLI)

スロットタイプを使用するインテントのバージョンを発行する前に、そのスロットタイプのバージョンを発行する必要があります。この例では、FlowerTypes を発行します。

> **Note**
>
> 次の AWS CLI の例は、Unix、Linux、および macOS 用にフォーマットされています。Windows の場合は、"\$LATEST" を $LATEST に変更し、各行末のバックスラッシュ (\) 連結文字をキャレット (^) に置き換えてください。

スロットタイプを発行するには (AWS CLI)

1. AWS CLI で、スロットタイプの最新バージョンを取得します。

```
aws lex-models get-slot-type \
    --region region \
    --name FlowerTypes \
    --slot-type-version "\$LATEST"
```

Amazon Lex からのレスポンスは次のとおりです。$LATEST バージョンの最新リビジョンのチェックサムを書き留めます。

```
{
    "enumerationValues": [
        {
            "value": "tulips"
        },
        {
            "value": "lilies"
        },
        {
            "value": "roses"
        }
    ],
    "name": "FlowerTypes",
    "checksum": "checksum",
    "version": "$LATEST",
    "lastUpdatedDate": timestamp,
    "createdDate": timestamp,
    "description": "Types of flowers to pick up"
}
```

2. スロットタイプのバージョンを発行します。前のステップで書き留めたチェックサムを使用します。

```
aws lex-models create-slot-type-version \
    --region region \
    --name FlowerTypes \
    --checksum "checksum"
```

Amazon Lex からのレスポンスは次のとおりです。次のステップのためにバージョン番号を書き留めます。

```
{
    "version": "1",
    "enumerationValues": [
        {
            "value": "tulips"
        },
        {
            "value": "lilies"
        },
        {
            "value": "roses"
        }
    ],
    "name": "FlowerTypes",
    "createdDate": timestamp,
    "lastUpdatedDate": timestamp,
    "description": "Types of flowers to pick up"
}
```

次のステップ

ステップ 2: インテントを発行する (AWS CLI)

ステップ 2: インテントを発行する (AWS CLI)

インテントを発行する前に、そのインテントで参照しているすべてのスロットタイプを発行する必要があります。スロットタイプは、$LATEST バージョンではなく、番号が付いたバージョンであることが必要です。

まず、OrderFlowers インテントを更新し、前のステップで発行した FlowerTypes スロットタイプのバージョンを使用します。次に、OrderFlowers インテントの新しいバージョンを発行します。

> **Note**
>
> 次の AWS CLI の例は、Unix、Linux、および macOS 用にフォーマットされています。Windows の場合は、"\$LATEST" を $LATEST に変更し、各行末のバックスラッシュ (\) 連結文字をキャレット (^) に置き換えてください。

インテントのバージョンを発行するには (AWS CLI)

1. AWS CLI で、OrderFlowers インテントの $LATEST バージョンを取得し、それをファイルに保存します。

```
aws lex-models get-intent \
    --region region \
    --name OrderFlowers \
    --intent-version "\$LATEST" > OrderFlowers_V4.json
```

2. テキストエディタで、**OrderFlowers_V4.json** ファイルを開きます。createdDate、lastUpdatedDate、version の各フィールドを削除します。FlowerTypes スロットタイプを見つけ、そのバージョンを前のステップで書き留めたバージョン番号に変更します。以下は、**OrderFlowers_V4.json** ファイルの中で変更箇所を示す部分です。

```
        {
            "slotType": "FlowerTypes",
            "name": "FlowerType",
            "slotConstraint": "Required",
            "valueElicitationPrompt": {
                "maxAttempts": 2,
                "messages": [
                    {
                        "content": "What type of flowers?",
                        "contentType": "PlainText"
                    }
                ]
            },
            "priority": 1,
            "slotTypeVersion": "version",
            "sampleUtterances": []
        },
```

3. AWS CLI で、インテントのリビジョンを保存します。

```
aws lex-models put-intent \
    --name OrderFlowers \
    --cli-input-json file://OrderFlowers_V4.json
```

4. インテントの最新リビジョンのチェックサムを取得します。

```
aws lex-models get-intent \
    --region region \
    --name OrderFlowers \
    --intent-version "\$LATEST" > OrderFlowers_V4a.json
```

以下は、レスポンスの中でインテントのチェックサムを示す部分です。次のステップのために、これを書き留めます。

```
    "name": "OrderFlowers",
    "checksum": "checksum",
    "version": "$LATEST",
```

5. インテントの新しいバージョンを発行します。

```
aws lex-models create-intent-version \
    --region region \
    --name OrderFlowers \
    --checksum "checksum"
```

以下は、レスポンスの中でインテントの新しいバージョンを示す部分です。次のステップのためにバージョン番号を書き留めます。

```
    "name": "OrderFlowers",
    "checksum": "checksum",
    "version": "version",
```

次のステップ

ステップ 3: ボットを発行する (AWS CLI)

ステップ 3: ボットを発行する (AWS CLI)

ボットで使用しているすべてのスロットタイプとインテントを発行したら、次にボットを発行できます。

OrderFlowersBot ボットを更新し、前のステップで更新した OrderFlowers インテントを使用します。次に、OrderFlowersBot ボットの新しいバージョンを発行します。

> **Note**
>
> 次の AWS CLI の例は、Unix、Linux、および macOS 用にフォーマットされています。Windows の場合は、"\$LATEST" を $LATEST に変更し、各行末のバックスラッシュ (\) 連結文字をキャレット (^) に置き換えてください。

ボットのバージョンを発行するには (AWS CLI)

1. AWS CLI で、OrderFlowersBot ボットの $LATEST バージョンを取得し、それをファイルに保存します。

   ```
   aws lex-models get-bot \
       --region region \
       --name OrderFlowersBot \
       --version-or-alias "\$LATEST" > OrderFlowersBot_V4.json
   ```

2. テキストエディタで、**OrderFlowersBot_V4.json** ファイルを開きます。createdDate、lastUpdatedDate、status、version の各フィールドを削除します。OrderFlowers インテントを見つけ、そのバージョンを前のステップで書き留めたバージョン番号に変更します。以下は、**OrderFlowersBot_V4.json** の中で変更箇所を示す部分です。

   ```
       "intents": [
           {
               "intentVersion": "version",
               "intentName": "OrderFlowers"
           }
   ```

3. AWS CLI で、ボットの新しいリビジョンを保存します。put-bot の呼び出しによって返されるバージョン番号をメモしておきます。

```
aws lex-models put-bot \
    --name OrderFlowersBot \
    --cli-input-json file://OrderFlowersBot_V4.json
```

4. ボットの最新リビジョンのチェックサムを取得します。ステップ 3 で返されたバージョン番号を使用します。

```
aws lex-models get-bot \
    --region region \
    --version-or-alias version \
    --name OrderFlowersBot > OrderFlowersBot_V4a.json
```

以下は、レスポンス内のボットのチェックサムを示す部分です。次のステップのために、これを書き留めます。

```
    "name": "OrderFlowersBot",
    "locale": "en-US",
    "checksum": "checksum",
```

5. ボットの新しいバージョンを発行します。

```
aws lex-models create-bot-version \
    --region region \
    --name OrderFlowersBot \
    --checksum "checksum"
```

以下は、レスポンス内でボットの新しいバージョンを示す部分です。

```
    "checksum": "checksum",
    "abortStatement": {
        ...
    },
    "version": "1",
    "lastUpdatedDate": timestamp,
```

次のステップ

演習 5: エイリアスを作成する (AWS CLI)

演習 5: エイリアスを作成する (AWS CLI)

エイリアスはボットの特定バージョンを参照するポインタです。エイリアスを使用すると、クライアントアプリケーションで使用しているバージョンを簡単に更新できます。詳細については「バージョニングとエイリアス」をご覧ください。この演習のコマンドを実行するには、コマンドが実行されるリージョンを確認しておく必要があります。リージョンのリストについては、「 モデル構築のクォータ 」を参照してください。

エイリアスを作成するには (AWS CLI)

1. AWS CLI で、演習 4: バージョンを発行する (AWS CLI) で作成した `OrderFlowersBot` ボットのバージョンを取得します。

   ```
   aws lex-models get-bot \
       --region region \
       --name OrderFlowersBot \
       --version-or-alias version > OrderFlowersBot_V5.json
   ```

2. テキストエディタで **`OrderFlowersBot_v5.json`** を開きます。バージョン番号を見つけて書き留めます。

3. AWS CLI で、ボットのエイリアスを作成します。

   ```
   aws lex-models put-bot-alias  \
       --region region \
       --name PROD \
       --bot-name OrderFlowersBot \
       --bot-version version
   ```

   サーバーからのレスポンスは次のとおりです。

   ```
   {
       "name": "PROD",
       "createdDate": timestamp,
       "checksum": "checksum",
       "lastUpdatedDate": timestamp,
       "botName": "OrderFlowersBot",
       "botVersion": "1"
   }}
   ```

次のステップ

演習 6: クリーンアップする (AWS CLI)

演習 6: クリーンアップする (AWS CLI)

作成したリソースを削除し、アカウントをクリーンアップします。

削除できるのは、使用中ではないリソースだけです。通常、以下の順序でリソースを削除します。

1. エイリアスを削除して、ボットのリソースを解放します。
2. ボットを削除して、インテントのリソースを解放します。
3. インテントを削除して、スロットタイプのリソースを解放します。
4. スロットタイプを削除します。

この演習のコマンドを実行するには、コマンドが実行されるリージョンを確認しておく必要があります。リージョンのリストについては、「 モデル構築のクォータ 」を参照してください。

アカウントをクリーンアップするには (AWS CLI)

1. AWS CLI コマンドラインで、エイリアスを削除します。

```
aws lex-models delete-bot-alias \
    --region region \
    --name PROD \
    --bot-name OrderFlowersBot
```

2. AWS CLI コマンドラインで、ボットを削除します。

```
aws lex-models delete-bot \
    --region region \
    --name OrderFlowersBot
```

3. AWS CLI コマンドラインで、インテントを削除します。

```
aws lex-models delete-intent \
    --region region \
    --name OrderFlowers
```

4. AWS CLI コマンドラインで、スロットタイプを削除します。

```
aws lex-models delete-slot-type \
    --region region \
    --name FlowerTypes
```

これで、作成したすべてのリソースが削除され、アカウントがクリーンアップされました。

バージョニングとエイリアス

Amazon Lex では、クライアントアプリケーションで使用する実装を制御できるように、ボット、インテント、スロットタイプのバージョンの発行をサポートしています。バージョン は、番号付きスナップショットであり、これをワークフローの開発、ベータデプロイ、本番稼働など、作業の段階別に発行して使用できます。

Amazon Lex ボットは、エイリアスもサポートしています。エイリアスはボットの特定バージョンを参照するポインタです。エイリアスを使用すると、クライアントアプリケーションが使用しているバージョンを簡単に更新できます。例えば、エイリアスにボットのバージョン 1 を参照させます。ボットを更新する準備ができたら、バージョン 2 を発行し、新しいバージョンを参照するようにエイリアスを変更します。アプリケーションは、特定のバージョンではなくエイリアスを使用しているため、すべてのクライアントが更新なしで新しい機能を使用できるようになります。

トピック

バージョニング

Amazon Lex リソースのスナップショットをバージョンとして作成すると、その作成時点のリソースを使用できます。作成したバージョンは、アプリケーションで作業を続けても変更されずに残ります。

$LATEST バージョン

Amazon Lex のボット、インテント、またはスロットタイプの作成時点では、バージョンは 1 つ ($LATEST バージョン) のみです。

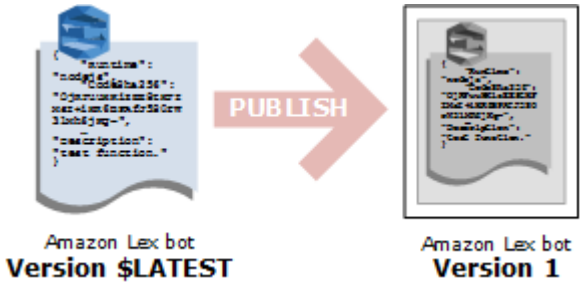
$LATEST は、リソースの作業コピーです。更新できるバージョンは $LATEST のみです。最初のバージョンを発行するまでは、$LATEST がリソースの唯一のバージョンです。

リソースの $LATEST バージョンでのみ、別のリソースの $LATEST バージョンを使用できます。例えば、ボットの $LATEST バージョンでは、インテントの $LATEST バージョンを使用できます。インテントの $LATEST バージョンでは、スロットタイプの $LATEST バージョンを使用できます。

$LATEST バージョンのボットは、手動テストにのみ使用してください。Amazon Lex では、$LATEST バージョンのボットに対するランタイムリクエストの数が制限されています。

Amazon Lex リソースバージョンの公開

リソースを発行すると、Amazon Lex は $LATEST バージョンのコピーを作成し、これを番号付きバージョンとして保存します。発行済みバージョンは変更できません。



Amazon Lex コンソールまたは CreateBotVersion オペレーションを使用してバージョンを作成し発行します。例については、「演習 3: バージョンを発行してエイリアスを作成する」を参照してください。

リソースの $LATEST バージョンを変更したら、新しいバージョンを発行して、その変更がクライアントアプリケーションで使用できるようにします。バージョンを発行するたびに、Amazon Lex が $LATEST バージョンをコピーして新しいバージョンを作成し、バージョン番号を 1 つずつ増加させます。バージョン番号が再利用されることはありません。例えば、バージョン 10 のリソースを削除して再作成すると、Amazon Lex が割り当てる次のバージョン番号はバージョン 11 になります。

ボットを発行する前に、ボットが使用しているインテントの番号付きバージョンをボットで参照する必要があります。インテントの $LATEST バージョンを使用しているボットの新バージョンを発行しようとすると、Amazon Lex から HTTP 400 Bad Request 例外が返されます。インテントの番号付きバージョンを発行する前に、インテントが使用しているスロットタイプの番号付きバージョンをインテントで参照する必要があります。そうしないと、HTTP 400 Bad Request 例外が発生します。

> **Note**
>
> Amazon Lex は、最後に発行したバージョンが $LATEST バージョンと異なる場合にのみ新しいバージョンを発行します。$LATEST バージョンを変更せずに発行しようとすると、Amazon Lex は新しいバージョンを作成せず、発行もしません。

Amazon Lex リソースの更新

Amazon Lex のボット、インテント、スロットタイプの $LATEST バージョンのみ更新できます。発行済みバージョンは変更できません。新しいバージョンは、コンソールを使用するか、CreateBotVersion オペレーション、CreateIntentVersion オペレーション、CreateSlotTypeVersion オペレーションのいずれかを使用してリソースを更新した後で、いつでも発行できます。

Amazon Lex のリソースまたはバージョンの削除

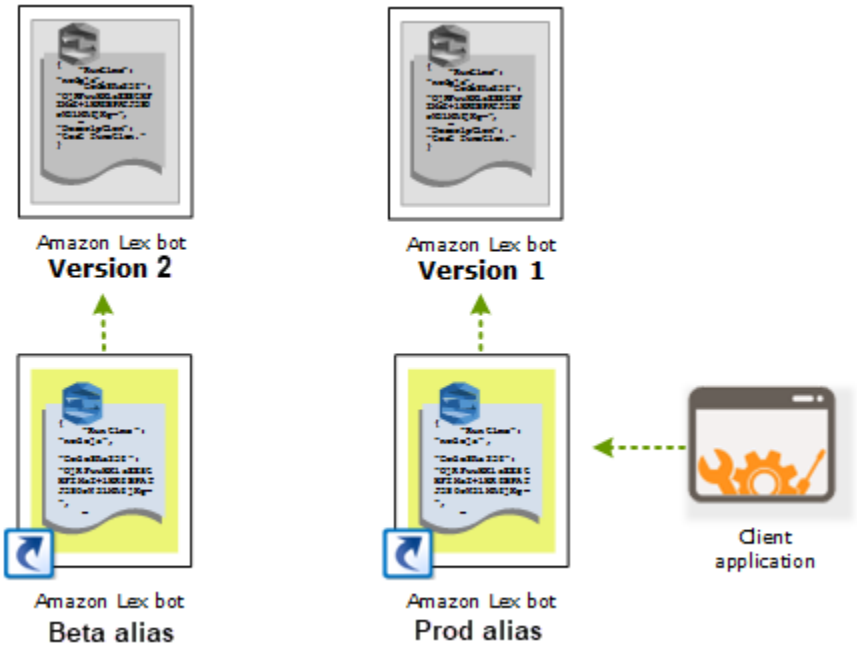
Amazon Lex では、コンソールを使用するか、API オペレーションのいずれかを使用して、リソースやバージョンを削除できます。

- DeleteBot
- DeleteBotVersion
- DeleteBotAlias
- DeleteBotChannelAssociation
- DeleteIntent
- DeleteIntentVersion
- DeleteSlotType
- DeleteSlotTypeVersion

エイリアス

エイリアスとは Amazon Lex ボットの特定バージョンを参照するポインタです。エイリアスを使用すると、クライアントアプリケーションはボットの特定のバージョンを追跡することなく、どのバージョンでも使えるようになります。

次の例では、Amazon Lex ボットの 2 つのバージョン (バージョン 1 とバージョン 2) を示しています。これらのボットのバージョンにはそれぞれ、BETA と PROD というエイリアスが関連付けられています。クライアントアプリケーションは、PROD エイリアスを使用してボットにアクセスします。



ボットの 2 番目のバージョンを作成したら、コンソールまたは PutBot オペレーションを使用して、ボットの新しいバージョンを参照するようにエイリアスを更新できます。エイリアスを変更すると、すべてのクライアントアプリケーションが新しいバージョンを使用します。新しいバージョンに問題がある場合は、そのバージョンを参照するエイリアスを変更するだけで、以前のバージョンにロールバックできます。

> **Note**
>
> $LATEST バージョンのボットはコンソールでテストできますが、ボットとクライアントアプリケーションを統合する場合は、最初にバージョンを発行してそのバージョンを参照するエイリアスを作成することをお勧めします。このセクションで説明した理由から、クライアントアプリケーションではエイリアスを使用してください。エイリアスを更新すると、Amazon Lex ではすべての現行セッションのセッションタイムアウトが期限切れになるまで待ってから、新しいバージョンの使用を開始します。セッションタイムアウトの詳細については、「the section called “セッションタイムアウトの設定”」を参照してください。

Lambda 関数を使用する

Amazon Lex のボットのコードフックとして使用する AWS Lambda 関数を作成できます。インテント設定で初期化/検証、フルフィルメント、またはその両方を実行する Lambda 関数を特定できます。

ボットのコードフックとして Lambda 関数を使用することをお勧めします。Lambda 関数を使用しないと、ボットはインテントを達成するための情報をクライアントアプリケーションに返します。

トピック

Lambda 関数の入力イベントとレスポンスの形式

このセクションでは、Amazon Lex から Lambda 関数に提供するイベントデータの構造について説明します。この情報は Lambda コードの入力の解析に使用します。また、Lambda 関数から Amazon Lex に返す必要があるレスポンスの形式についても説明します。

トピック

入力イベントの形式

以下に、Lambda 関数に渡される一般的な形式の Amazon Lex イベントを示します。この情報は、Lambda 関数の作成時に使用します。

> **Note**
>
> 入力形式は変わる場合があり、この変更は対応する `messageVersion` に反映されないことがあります。新しいフィールドが追加されても、コードでエラーがスローされないようにします。

```
{
  "currentIntent": {
    "name": "intent-name",
    "nluIntentConfidenceScore": score,
    "slots": {
      "slot name": "value",
      "slot name": "value"
    },
    "slotDetails": {
      "slot name": {
        "resolutions" : [
          { "value": "resolved value" },
          { "value": "resolved value" }
        ],
        "originalValue": "original text"
      },
      "slot name": {
        "resolutions" : [
          { "value": "resolved value" },
          { "value": "resolved value" }
        ],
        "originalValue": "original text"
      }
    },
    "confirmationStatus": "None, Confirmed, or Denied (intent confirmation, if
 configured)"
  },
  "alternativeIntents": [
    {
      "name": "intent-name",
      "nluIntentConfidenceScore": score,
      "slots": {
        "slot name": "value",
        "slot name": "value"
      },
      "slotDetails": {
        "slot name": {
          "resolutions" : [
            { "value": "resolved value" },
            { "value": "resolved value" }
          ],
          "originalValue": "original text"
        },
```

```
        "slot name": {
          "resolutions" : [
            { "value": "resolved value" },
            { "value": "resolved value" }
          ],
          "originalValue": "original text"
        }
      },
      "confirmationStatus": "None, Confirmed, or Denied (intent confirmation, if
configured)"
    }
 ],
 "bot": {
   "name": "bot name",
   "alias": "bot alias",
   "version": "bot version"
 },
 "userId": "User ID specified in the POST request to Amazon Lex.",
 "inputTranscript": "Text used to process the request",
 "invocationSource": "FulfillmentCodeHook or DialogCodeHook",
 "outputDialogMode": "Text or Voice, based on ContentType request header in runtime
API request",
 "messageVersion": "1.0",
 "sessionAttributes": {
    "key": "value",
    "key": "value"
 },
 "requestAttributes": {
    "key": "value",
    "key": "value"
 },
 "recentIntentSummaryView": [
   {
       "intentName": "Name",
       "checkpointLabel": Label,
       "slots": {
         "slot name": "value",
         "slot name": "value"
       },
       "confirmationStatus": "None, Confirmed, or Denied (intent confirmation, if
configured)",
       "dialogActionType": "ElicitIntent, ElicitSlot, ConfirmIntent, Delegate, or
Close",
       "fulfillmentState": "Fulfilled or Failed",
```

```
        "slotToElicit": "Next slot to elicit"
    }
  ],
   "sentimentResponse": {
      "sentimentLabel": "sentiment",
      "sentimentScore": "score"
   },
   "kendraResponse": {
       Complete query response from Amazon Kendra
   },
   "activeContexts": [
        {
            "timeToLive": {
                "timeToLiveInSeconds": seconds,
                "turnsToLive": turns
            },
            "name": "name",
            "parameters": {
                "key name": "value"
            }
        }
    ]
}
```

イベントフィールドに関して、次の追加情報に注意してください。

- currentIntent – インテントの name、slots、slotDetails、confirmationStatus の各フィールドを提供します。

  nluIntentConfidenceScore とは、現在のインテントがユーザーの現在のインテントに最も合致するものであると Amazon Lex が確信することです。

  slots は、スロット名のマップであり、インテントで Amazon Lex がユーザーとの会話で認識したスロット値に設定されます。スロット値は、ユーザーが値を指定するまで null のままです。

  入力イベントのスロット値は、スロットに設定済みの値のいずれとも一致しない場合があります。例えば、「どの色の車が好きですか?」というプロンプトに対して ユーザーが「ピザ」と答える

と、Amazon Lex は「ピザ」をスロット値として返します。関数では、値を検証し、値がコンテキストで意味をなすことを確認する必要があります。

slotDetails は、スロット値に関する追加の情報を提供します。resolutions 配列は、スロットで認識される追加の値のリストです。各スロットは、最大 5 個の値を持つことができます。

originalValue フィールドには、スロットのユーザーが入力した値が入ります。スロット値として最初の解決の値を返すようにスロットタイプを設定すると、originalValue は slots フィールドの値と異なる場合があります。

confirmationStatus は、確認のプロンプトが発生した場合、それに応じてユーザーレスポンスを提供します。例えば、Amazon Lex が「ラージサイズのチーズピザを注文しますか?」と質問した場合、ユーザーレスポンスに応じて、このフィールドの値は Confirmed または Denied となります。それ以外の場合、このフィールドの値は None になります。

ユーザーがインテントを確認した場合、Amazon Lex はこのフィールドを Confirmed に設定します。ユーザーがインテントを拒否した場合、Amazon Lex はこの値を Denied に設定します。

確認のレスポンスで、ユーザーの発話によりスロットの更新が提供される場合があります。例えば、ユーザーが「はい、サイズを M に変更します。」と言ったとします。この場合、以降の Lambda イベントではスロット値が更新され、PizzaSize が medium に設定されます。Amazon Lex は confirmationStatus を None に設定します。これはユーザーが一部のスロットデータを変更し、Lambda 関数はユーザーデータの検証を実行する必要があるためです。

- 代替インテント — 信頼度スコアを有効にすると、Amazon Lex は最大 4 つの代替インテントを返します。各インテントには、ユーザーの発話に基づくインテントが正しいインテントであるという Amazon Lex の信頼度を示すスコアが含まれます。

代替インテントの内容は、currentIntent フィールドの内容と同じです。詳細については、「信頼スコアの使用」を参照してください。

- bot – リクエストを処理したボットに関する情報です。
  - name – リクエストを処理したボットの名前。
  - alias – リクエストを処理したボットのバージョンのエイリアス。
  - version – リクエストを処理したボットのバージョン。

- userId – この値はクライアントアプリケーションから提供されます。Amazon Lex は、その値を Lambda 関数に渡します。

- inputTranscript – リクエストの処理に使用されるテキストです。

  入力がテキストであった場合、inputTranscript フィールドにはユーザーが入力したテキストが入ります。

  入力がオーディオストリームであった場合、inputTranscript フィールドにはオーディオストリームから抽出されたテキストが入ります。これは、インテントとスロット値を認識するために実際に処理されるテキストです。

- invocationSource – Amazon Lex が Lambda 関数を呼び出す理由を示すため、以下のいずれかの値に設定されます。
  - DialogCodeHook – Amazon Lex は、この値を設定することで、Lambda 関数に対して関数の初期化とユーザーが入力したデータの検証を指示します。

    初期化および検証のコードフックとして Lambda 関数を呼び出すようにインテントを設定すると、Amazon Lex がインテントを理解した後で、Amazon Lex はユーザー入力 (発話) ごとに指定された Lambda 関数を呼び出します。

> **Note**
>
> インテントが明確でない場合、Amazon Lex は Lambda 関数を呼び出すことができません。

- `FulfillmentCodeHook` – Amazon Lex はこの値を設定することで、Lambda 関数にインテントを達成するよう指示します。

  フルフィルメントコードフックとして Lambda 関数を呼び出すようインテントが設定されている場合、Amazon Lex は、インテントを達成するためにすべてのスロットデータが揃った後でのみ、`invocationSource` をこの値に設定します。

  インテントの設定では、2 つの異なる Lambda 関数を使用してユーザーデータを初期化および検証し、インテントを達成できます。1 つの Lambda 関数を使用して両方を行うこともできます。その場合、Lambda 関数は、`invocationSource` 値を使用して正しいコードパスをたどることができます。

- outputDialogMode – 各ユーザー入力について、クライアントはいずれかのランタイム API オペレーション、PostContent、または PostText を使用して、リクエストを Amazon Lex に送信します。Amazon Lex はリクエストパラメータを使用してクライアントへのレスポンスがテキストまたは音声であるかを判断し、それに応じてこのフィールドを設定します。

  Lambda 関数は、この情報を使用して適切なメッセージを生成できます。例えば、クライアントが音声応答を予期している場合、Lambda 関数はテキストの代わりに音声合成マークアップ言語 (SSML) を返すことができます。

- messageVersion – Lambda 関数に渡されるイベントデータの形式と Lambda 関数から返す必要があるレスポンスの形式を識別するメッセージのバージョン。

> **Note**
>
> インテントを定義するときに、この値を設定します。現在の実装では、メッセージバージョン 1.0 のみがサポートされています。そのため、コンソールではデフォルト値の 1.0 が想定され、メッセージのバージョンは表示されません。

- sessionAttributes – クライアントがリクエストで送信するアプリケーション固有のセッション属性。Amazon Lex でこれらの属性をクライアントへのレスポンスに含める場合、Lambda 関数はこれらの属性をレスポンスで Amazon Lex に返す必要があります。詳細については、「セッション属性の設定」を参照してください。

- requestAttributes – クライアントがリクエストで送信するリクエスト固有の属性。セッション全体を通しては保持する必要がない情報は、リクエスト属性を使用して渡します。リクエスト属性がない場合、値は null になります。詳細については、「リクエスト属性の設定」を参照してください。

- recentIntentSummaryView - インテントの状態に関する情報。最後に使用された 3 つのインテントに関する情報を表示できます。この情報を使用して、インテントの値を設定したり、前のインテントに戻ったりできます。詳細については、「Amazon Lex API を使用したセッションの管理」を参照してください。

- sentimentResponse - 最後の発話の Amazon Comprehend センチメント分析の結果。この情報を使用すると、ユーザーが表現したセンチメントに応じたボットの会話フローを管理できます。詳細については、「センチメント分析」を参照してください。

- kendraResponse - Amazon Kendra インデックスへのクエリの結果。フルフィルメントコードフックへの入力にのみ含まれ、インテントが `AMAZON.KendraSearchIntent` 組み込みインテントを継承する場合にのみ使用されます。このフィールドには、Amazon Kendra 検索からのレスポンス全体が含まれています。詳細については、「AMAZON.KendraSearchIntent」を参照してください。

- ActiveContexts — ユーザーとの会話のこのターン中にアクティブな 1 つ以上のコンテキスト。

- TimeTime Live — ユーザーとの会話の中で、コンテキストがアクティブのままである時間の長さやターン数。
- name – コンテキストの名前。
- parameters – コンテキストを起動したインテントのスロットの名前と値を含む、キーバリューのペアのリスト。

詳細については、「インテントコンテキストの設定」を参照してください。

レスポンスの形式

Amazon Lex は、以下の形式の Lambda 関数からのレスポンスを想定しています：

```
{
  "sessionAttributes": {
    "key1": "value1",
    "key2": "value2"
    ...
  },
  "recentIntentSummaryView": [
    {
       "intentName": "Name",
       "checkpointLabel": "Label",
       "slots": {
         "slot name": "value",
         "slot name": "value"
        },
       "confirmationStatus": "None, Confirmed, or Denied (intent confirmation, if
 configured)",
        "dialogActionType": "ElicitIntent, ElicitSlot, ConfirmIntent, Delegate, or
 Close",
        "fulfillmentState": "Fulfilled or Failed",
        "slotToElicit": "Next slot to elicit"
    }
  ],
  "activeContexts": [
     {
       "timeToLive": {
          "timeToLiveInSeconds": seconds,
          "turnsToLive": turns
      },
      "name": "name",
```

```
      "parameters": {
        "key name": "value"
      }
    }
  ],
  "dialogAction": {
    "type": "ElicitIntent, ElicitSlot, ConfirmIntent, Delegate, or Close",
    Full structure based on the type field. See below for details.
  }
}
```

レスポンスは 4 つのフィールドで構成されます。sessionAttributes、recentIntentSummaryView、activeContexts フィールドはオプションで、dialogAction フィールドは必須です。dialogAction フィールドの内容は、type の値によって異なります。詳細については、「dialogAction」を参照してください。

sessionAttributes

オプション。sessionAttributes フィールドを含める場合、このフィールドは空にすることができます。Lambda 関数でセッション属性が返らない場合は、API または Lambda 関数で渡された最後の既知の sessionAttributes は維持されます。詳細については、「PostContent オペレーション」と「PostText オペレーション」を参照してください。

```
  "sessionAttributes": {
     "key1": "value1",
     "key2": "value2"
  }
```

recentIntentSummaryView

オプション。含まれている場合、1 つ以上の最近のインテントの値を設定します。最大 3 つのインテントの情報を含めることができます。例えば、現在のインテントによって収集された情報に基づいて、以前のインテントの値を設定できます。概略の情報は、インテントに対して有効である必要があります。例えば、インテント名はボットのインテントでなければなりません。概略ビューにスロット値を含める場合、スロットはインテント内に存在する必要があります。レスポンスに recentIntentSummaryView を含めない場合、最近のインテントのすべての値は変更されません。詳細については、「PutSession オペレーション」または「IntentSummary データ型」を参照してください。

```
"recentIntentSummaryView": [
```

```
    {
       "intentName": "Name",
       "checkpointLabel": "Label",
       "slots": {
         "slot name": "value",
         "slot name": "value"
        },
       "confirmationStatus": "None, Confirmed, or Denied (intent confirmation, if
configured)",
        "dialogActionType": "ElicitIntent, ElicitSlot, ConfirmIntent, Delegate, or
Close",
        "fulfillmentState": "Fulfilled or Failed",
        "slotToElicit": "Next slot to elicit"
    }
 ]
```

activeContexts

オプション。含まれている場合は、1 つ以上のコンテクストの値を設定します。例えば、コンテキストを含めて、会話の次のターンでそのコンテキストを認識できる入力として持つ 1 つ以上のインテントを作成できます。

レスポンスに含まれていないアクティブなコンテキストは、有効期限 (TTL)の値が減少し、次のリクエストでもアクティブになる場合があります。

入力イベントに含まれていたコンテキストに対して有効期限 (TTL) を 0 に指定すると、次のリクエストでは非アクティブになります。

詳細については、「インテントコンテキストの設定」を参照してください。

dialogAction

必須。dialogAction フィールドは、次の一連のアクションを Amazon Lex に指示し、Amazon Lex からクライアントにレスポンスが返された後でユーザーから予期されることを示します。

type フィールドは、次の一連のアクションを示します。このフィールドにより、dialogAction 値の一部として Lambda 関数が提供すべき他のフィールドも決まります。

- Close - ユーザーからのレスポンスを予期しないように Amazon Lex に伝えます。例えば、「ピザのご注文を受け付けました」はレスポンスが不要です。

fulfillmentState フィールドは必須です。Amazon Lex は、この値を使用してクライアントアプリケーションに対する dialogState レスポンスまたは PostContent レスポンスの PostText フィールドを設定します。message および responseCard フィールドはオプションです。メッセージを指定しないと、Amazon Lex はインテント用に設定された終了メッセージまたはフォローアップメッセージを使用します。

```
"dialogAction": {
    "type": "Close",
    "fulfillmentState": "Fulfilled or Failed",
    "message": {
      "contentType": "PlainText or SSML or CustomPayload",
      "content": "Message to convey to the user. For example, Thanks, your pizza has
 been ordered."
    },
   "responseCard": {
      "version": integer-value,
      "contentType": "application/vnd.amazonaws.card.generic",
      "genericAttachments": [
          {
             "title":"card-title",
             "subTitle":"card-sub-title",
             "imageUrl":"URL of the image to be shown",
             "attachmentLinkUrl":"URL of the attachment to be associated with the
 card",
             "buttons":[
                 {
                    "text":"button-text",
                    "value":"Value sent to server on button click"
                 }
              ]
           }
       ]
     }
  }
```

- ConfirmIntent - ユーザーが現在のインテントを確認または拒否するために、「はい」または「いいえ」と答えることが予期されていることを Amazon Lex に知らせます。

intentName フィールドと slots フィールドを含める必要があります。slots フィールドには、指定されたインテントに入力された各スロットのエントリを含める必要があります。入力されていないスロットの slots フィールドにエントリを含める必要はありません。目的の confirmationPrompt フィールドが null の場合は、message フィールドを含める必要があります。Lambda 関数で返る message フィールドの内容は、インテントで指定される confirmationPrompt よりも優先されます。responseCard フィールドはオプションです。

```
"dialogAction": {
    "type": "ConfirmIntent",
    "message": {
      "contentType": "PlainText or SSML or CustomPayload",
      "content": "Message to convey to the user. For example, Are you sure you want a
 large pizza?"
    },
   "intentName": "intent-name",
   "slots": {
      "slot-name": "value",
      "slot-name": "value",
      "slot-name": "value"
   },
   "responseCard": {
      "version": integer-value,
      "contentType": "application/vnd.amazonaws.card.generic",
      "genericAttachments": [
          {
             "title":"card-title",
             "subTitle":"card-sub-title",
             "imageUrl":"URL of the image to be shown",
             "attachmentLinkUrl":"URL of the attachment to be associated with the
 card",
             "buttons":[
                 {
                    "text":"button-text",
                    "value":"Value sent to server on button click"
                 }
              ]
           }
       ]
     }
 }
```

- Delegate - ボットの設定に基づいて次の一連のアクションを選択するよう Amazon Lex に指示します。レスポンスにセッション属性が含まれていない場合は、Amazon Lex で既存の属性が保持されます。スロット値を null にする場合は、スロットフィールドをリクエストに含める必要はありません。フルフィルメント関数がスロットを削除せずに Delegate ダイアログアクションを返した場合は、DependencyFailedException 例外が発生します。

  kendraQueryRequestPayload および kendraQueryFilterString フィールドはオプションであり、インテントが AMAZON.KendraSearchIntent 組み込みインテントを継承する場合にのみ使用されます。詳細については、「AMAZON.KendraSearchIntent」を参照してください。

```
 "dialogAction": {
  "type": "Delegate",
  "slots": {
     "slot-name": "value",
     "slot-name": "value",
     "slot-name": "value"
  },
  "kendraQueryRequestPayload": "Amazon Kendra query",
  "kendraQueryFilterString": "Amazon Kendra attribute filters"
 }
```

- ElicitIntent - ユーザーはインテントを含む発話で応答することが予期されていることを Amazon Lex に知らせます。例えば、「ラージピザが欲しい」は OrderPizzaIntent を示します。一方、「ラージ」だけの発話は、Amazon Lex がユーザーのインテントを推論するには不十分です。

  message および responseCard フィールドはオプションです。メッセージを指定しない場合、Amazon Lex はボットの明確化プロンプトのいずれかを使用します。明確化プロンプトが定義されていない場合、Amazon Lex は 400 Bad Request 例外を返します。

```
{
  "dialogAction": {
    "type": "ElicitIntent",
    "message": {
      "contentType": "PlainText or SSML or CustomPayload",
      "content": "Message to convey to the user. For example, What can I help you
 with?"
    },
    "responseCard": {
```

```
      "version": integer-value,
      "contentType": "application/vnd.amazonaws.card.generic",
      "genericAttachments": [
          {
             "title":"card-title",
             "subTitle":"card-sub-title",
             "imageUrl":"URL of the image to be shown",
             "attachmentLinkUrl":"URL of the attachment to be associated with the
card",
             "buttons":[
                 {
                    "text":"button-text",
                    "value":"Value sent to server on button click"
                 }
              ]
           }
       ]
   }
}
```

- ElicitSlot - ユーザーがレスポンスでスロット値を提供することが予期されていることを Amazon Lex に知らせます。

  intentName、slotToElicit、slots の各フィールドは必須です。message および responseCard フィールドはオプションです。メッセージを指定しない場合、Amazon Lex はスロットに設定されているスロットを引き出すプロンプトのいずれかを使用します。

```
 "dialogAction": {
   "type": "ElicitSlot",
   "message": {
     "contentType": "PlainText or SSML or CustomPayload",
     "content": "Message to convey to the user. For example, What size pizza would
you like?"
   },
  "intentName": "intent-name",
  "slots": {
     "slot-name": "value",
     "slot-name": "value",
     "slot-name": "value"
  },
```

```
  "slotToElicit" : "slot-name",
  "responseCard": {
     "version": integer-value,
     "contentType": "application/vnd.amazonaws.card.generic",
     "genericAttachments": [
         {
            "title":"card-title",
            "subTitle":"card-sub-title",
            "imageUrl":"URL of the image to be shown",
            "attachmentLinkUrl":"URL of the attachment to be associated with the
card",
            "buttons":[
                {
                   "text":"button-text",
                   "value":"Value sent to server on button click"
                }
             ]
          }
       ]
    }
 }
```

Amazon Lex および AWS Lambda の設計図

Amazon Lex コンソールには、コンソールでボットをすばやく作成してテストできるように、設定済みのサンプルボット (ボットの設計図と呼ばれます) が用意されています。これらのボットの設計図ごとに、Lambda 関数の設計図も用意されています。これらの設計図には、対応するボットで使用できるサンプルコードが含まれています。これらの設計図を使用すると、Lambda 関数でコードフックとして設定されたボットをすばやく作成し、コードを記述することなくエンドツーエンドのセットアップをテストできます。

次の Amazon Lex ボットの設計図とそれに対応する AWS Lambda 関数の設計図を、ボットのコードフックとして使用できます。

- Amazon Lex 設計図 — `OrderFlowers`
  - AWS Lambda ブループリント — `lex-order-flowers-python`
- Amazon Lex 設計図 — `ScheduleAppointment`
  - AWS Lambda ブループリント — `lex-make-appointment-python`
- Amazon Lex 設計図 — `BookTrip`

- AWS Lambda ブループリント — `lex-book-trip-python`

設計図を使用してボットを作成し、Lambda 関数をコードフックとして使用するようボットを設定する方法については、「演習 1: 設計図を使用して Amazon Lex ボットを作成する (コンソール)」を参照してください。その他の設計図の使用例については、「その他の例: Amazon Lex ボットの作成」を参照してください。

特定のロケールの設計図の更新

英語 (US) (en-US) 以外のロケールで設計図を使用している場合は、ロケールを含めるようにインテントの名前を更新する必要があります。例えば、`OrderFlowers` の設計図を使用する場合、次のようにする必要があります。

- Lambda 関数のコードの最後に `dispatch` 関数を検索します。
- `dispatch` 関数では、インテントの名前を更新して、使用しているロケールを含めます。例えば、英語 (オーストラリア) (en-au) ロケールを使用している場合は、次の行を変更します。

  ```
  if intent_name == 'OrderFlowers':
  ```

  から

  ```
  if intent_name == 'OrderFlowers_enAU':
  ```

他の設計図では、別のインテント名を使用しているので、使用する前に上記のように更新する必要があります。

Amazon Lex ボットのデプロイ

このセクションでは、各種メッセージングプラットフォームおよびモバイルアプリケーションにおける Amazon Lex ボットのデプロイの例を示します。

トピック

メッセージングプラットフォームで Amazon Lex ボットをデプロイする

このセクションでは、Facebook、Slack、および Twilio の各メッセージングプラットフォームで Amazon Lex ボットをデプロイする方法について説明します。

> **Note**
>
> Facebook、Slack、Twilio の各設定を保存する際に、Amazon Lex は AWS Key Management Service カスタマーマスターキー (CMK) を使用して情報を暗号化します。これらのメッセージングプラットフォームのいずれかに対するチャンネルを初めて作成するときに、Amazon Lex はデフォルトの CMK (`aws/lex`) を作成します。独自のカスタマー管理キー (CMK) を AWS KMS で作成できます。これにより、キーの作成、更新、無効化ができるなど、より高い柔軟性が得られます。アクセスコントロールを定義し、データの保護に使用される暗号化キーを監査することもできます。詳細については、AWS Key Management Service デベロッパーガイドを参照してください。

メッセージングプラットフォームから Amazon Lex に送信されるリクエストには、プラットフォーム固有の情報が Lambda 関数へのリクエスト属性として含まれています。これらの属性を使用してボットの動作をカスタマイズします。詳細については、「リクエスト属性の設定」を参照してください。

すべての属性は、名前空間 `x-amz-lex:` をプレフィックスとして使用します。例えば、`user-id` 属性は `x-amz-lex:user-id` と呼ばれます。プラットフォーム別の固有の属性に加えて、すべてのメッセージングプラットフォームから送信される一般的な属性があります。以下の表は、メッセージングプラットフォームからボットの Lambda 関数に送信されるリクエスト属性の一覧です。

一般的なリクエスト属性

属性	説明
channel-id	Amazon Lex のチャンネルエンドポイント識別子。
channel-name	Amazon Lex のチャンネル名。
channel-type	次のいずれかの値になります。 • Facebook • Kik • Slack • Twilio-SMS
webhook-endpoint-url	チャンネルの Amazon Lex エンドポイント。

Facebook のリクエスト属性

属性	説明
user-id	送信者の Facebook 識別子。参照: https://developers.facebook.com/docs/messenger-platform/webhook-reference/message-received
facebook-page-id	受信者の Facebook ページ識別子。参照: https://developers.facebook.com/docs/messenger-platform/webhook-reference/message-received

Kik のリクエスト属性

属性	説明
kik-chat-id	ボットが関与する会話の識別子。詳細については、https://dev.kik.com/#/docs/messaging#message-formats を参照してください。

属性	説明
kik-chat-type	メッセージが発信された会話のタイプ。詳細については、https://dev.kik.com/#/docs/messaging#message-formats を参照してください。
kik-message-id	メッセージを識別する UUID。詳細については、https://dev.kik.com/#/docs/messaging#message-formats を参照してください。
kik-message-type	メッセージのタイプ。詳細については、https://dev.kik.com/#/docs/messaging#message-types を参照してください。

Twilio のリクエスト属性

属性	説明
user-id	送信者の電話番号 ("From")。参照: https://www.twilio.com/docs/api/rest/message
twilio-target-phone-number	受信者の電話番号 (「To」)。参照: https://www.twilio.com/docs/api/rest/message

Slack のリクエスト属性

属性	説明
user-id	Slack ユーザー識別子。参照: https://api.slack.com/types/user
slack-team-id	メッセージを送信したチームの識別子。参照: https://api.slack.com/methods/team.info
slack-bot-token	Slack API へのアクセス権限をボットに付与する開発者トークン。参照: https://api.slack.com/docs/token-types

Amazon Lex ボットと Facebook Messenger の統合

この演習では、Facebook Messenger と Amazon Lex ボットを統合する方法を示します。以下のステップを実行します。

1. Amazon Lex ボットを作成する
2. Facebook アプリケーションを作成する
3. Facebook Messenger と Amazon Lex ボットを統合する
4. 統合を検証する

トピック

ステップ 1: Amazon Lex ボットを作成する

Amazon Lex ボットをまだ持っていない場合には、作成してデプロイします。このトピックでは、「開始方法」の演習 1 で作成したボットを使用することを想定しています。ただし、このガイドで提供されている他のボット例を使用してもかまいません。「ご利用開始にあたって」の演習 1 については、「演習 1: 設計図を使用して Amazon Lex ボットを作成する (コンソール)」を参照してください。

1. Amazon Lex ボットを作成します。手順については、「演習 1: 設計図を使用して Amazon Lex ボットを作成する (コンソール)」を参照してください。
2. ボットをデプロイしてエイリアスを作成します。手順については、「演習 3: バージョンを発行してエイリアスを作成する」を参照してください。

ステップ 2: Facebook アプリケーションを作成する

Facebook 開発者ポータルで、Facebook アプリケーションと Facebook ページを作成します。手順については、Facebook Messenger プラットフォームのドキュメントで「クイックスタート」を参照してください。以下の内容を書き留めます。

- Facebook アプリの [App Secret]
- Facebook ページの [Page Access Token]

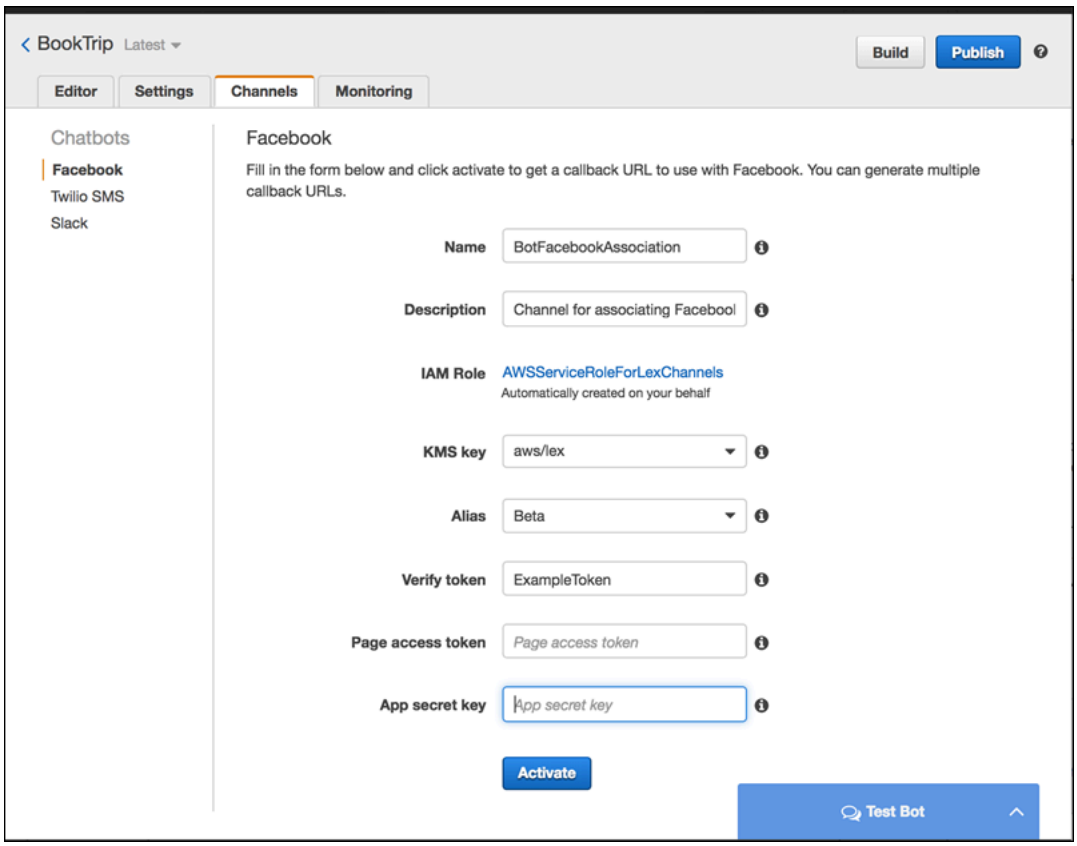
ステップ 3: Facebook Messenger と Amazon Lex ボットを統合する

このセクションでは、Facebook Messenger と Amazon Lex ボットを統合します。

このステップを完了すると、コンソールからコールバック URL が提供されます。この URL を書き留めます。

Facebook Messenger とボットを統合するには

1. a. AWS Management Console にサインインし、Amazon Lex コンソール (https://console.aws.amazon.com/lex/) を開きます。
   b. Amazon Lex ボットを選択してください。
   c. [Channels] を選択します。
   d. [Chatbots] の [Facebook] を選択します。コンソールに Facebook 統合ページが表示されます。
   e. Facebook 統合ページで、次の操作を行います。
      - 名前として「`BotFacebookAssociation`」と入力します。
      - [KMS key] で [aws/lex] を選択します。
      - [Alias] で、ボットのエイリアスを選択します。
      - [Verify token] にトークンを入力します。任意の文字列を選択できます (例: `ExampleToken`)。このトークンは、後で Webhook をセットアップするときに Facebook 開発者ポータルで使用します。
      - [Page access token] に、ステップ 2 で Facebook から取得したトークンを入力します。
      - [App secret key] に、ステップ 2 で Facebook から取得したキーを入力します。



f. [アクティブ化] を選択します。

コンソールによってボットチャンネル関連付けが作成され、コールバック URL が返されます。この URL を書き留めます。

2. Facebook 開発者ポータルで、アプリを選択します。
3. [Messenger] 製品を選択し、ページの [Webhooks] セクションで [Setup webhooks] を選択します。

手順については、Facebook Messenger プラットフォームのドキュメントで「クイックスタート」を参照してください。

4. サブスクリプションウィザードの [webhook] ページで、次の操作を行います。

- [Callback URL] (コールバック用 URL) に、前に Amazon Lex コンソールから返されたコールバック URL を入力します。
- [Verify Token] (トークンの検証) に、Amazon Lex で使用したトークンと同じものを入力します。

- [Subscription Fields] ([messages]、[messaging_postbacks]、および [messaging_optins]) を選択します。
- [Verify and Save] を選択します。これにより、Facebook と Amazon Lex のハンドシェイクが開始されます。

5. Webhooks 統合を有効にします。作成したページを選択し、[subscribe] を選択します。

> **Note**
>
> Webhook を更新または再作成する場合は、サブスクリプション解除してから再度サブスクライブする必要があります。

ステップ 4: 統合をテストする

これで Facebook Messenger から Amazon Lex ボットとの会話を開始できます。

1. Facebook ページを開き、[Message] を選択します。
2. [Messenger] ウィンドウで、「ステップ 1: Amazon Lex ボット (コンソール) を作成する」に記載されているのと同じテスト発話を使用します。

Amazon Lex ボットと Kik との統合

この演習では、Amazon Lex ボットと Kik メッセージングアプリケーションを統合する手順を示します。以下のステップを実行します。

1. Amazon Lex ボットを作成します。
2. Kik アプリとウェブサイトを使用する Kik ボットを作成します。
3. Amazon Lex コンソールを使用して Amazon Lex ボットと Kik ボットを統合します。
4. Kik を使用して Amazon Lex ボットとの会話を試して、Amazon Lex ボットと Kik との関連付けをテストします。

トピック

- ステップ 1: Amazon Lex ボットを作成する
- ステップ 2: Kik ボットの作成
- ステップ 3: Kik ボットと Amazon Lex ボットの統合

- ステップ 4: 統合をテストする

ステップ 1: Amazon Lex ボットを作成する

Amazon Lex ボットをまだ持っていない場合には、作成してデプロイします。このトピックでは、「開始方法」の演習 1 で作成したボットを使用することを想定しています。ただし、このガイドで提供されている他のボット例を使用してもかまいません。「ご利用開始にあたって」の演習 1 については、「演習 1: 設計図を使用して Amazon Lex ボットを作成する (コンソール)」を参照してください。

1. Amazon Lex ボットを作成します。手順については、「演習 1: 設計図を使用して Amazon Lex ボットを作成する (コンソール)」を参照してください。
2. ボットをデプロイしてエイリアスを作成します。手順については、「演習 3: バージョンを発行してエイリアスを作成する」を参照してください。

次のステップ

ステップ 2: Kik ボットの作成

ステップ 2: Kik ボットの作成

このステップでは、Kik ユーザーインターフェイスを使用して Kik ボットを作成します。ボットの作成中に生成された情報を使用して、Amazon Lex ボットに接続します。

1. Kik アプリをダウンロードしてインストールし、Kik アカウントにサインアップします (アカウントがない場合)。アカウントがある場合は、ログインします。
2. Kik ウェブサイト (https://dev.kik.com/) を開きます。ブラウザウィンドウは開いたままにします。
3. Kik アプリで、歯車アイコンを選択して設定を開き、[Your Kik Code] を選択します。
4. Kik ウェブサイトで Kik コードをスキャンし、Botsworth chatbot を開きます。[はい] を選択ししてボットダッシュボードを開きます。
5. Kik アプリで、[Create a Bot] を選択します。表示されるプロンプトに従って、Kik ボットを作成します。
6. ボットを作成したら、ブラウザで [Configuration] を選択します。新しいボットが選択されていることを確認します。
7. ボット名と API キーを書き留めます。次のセクションで必要になります。

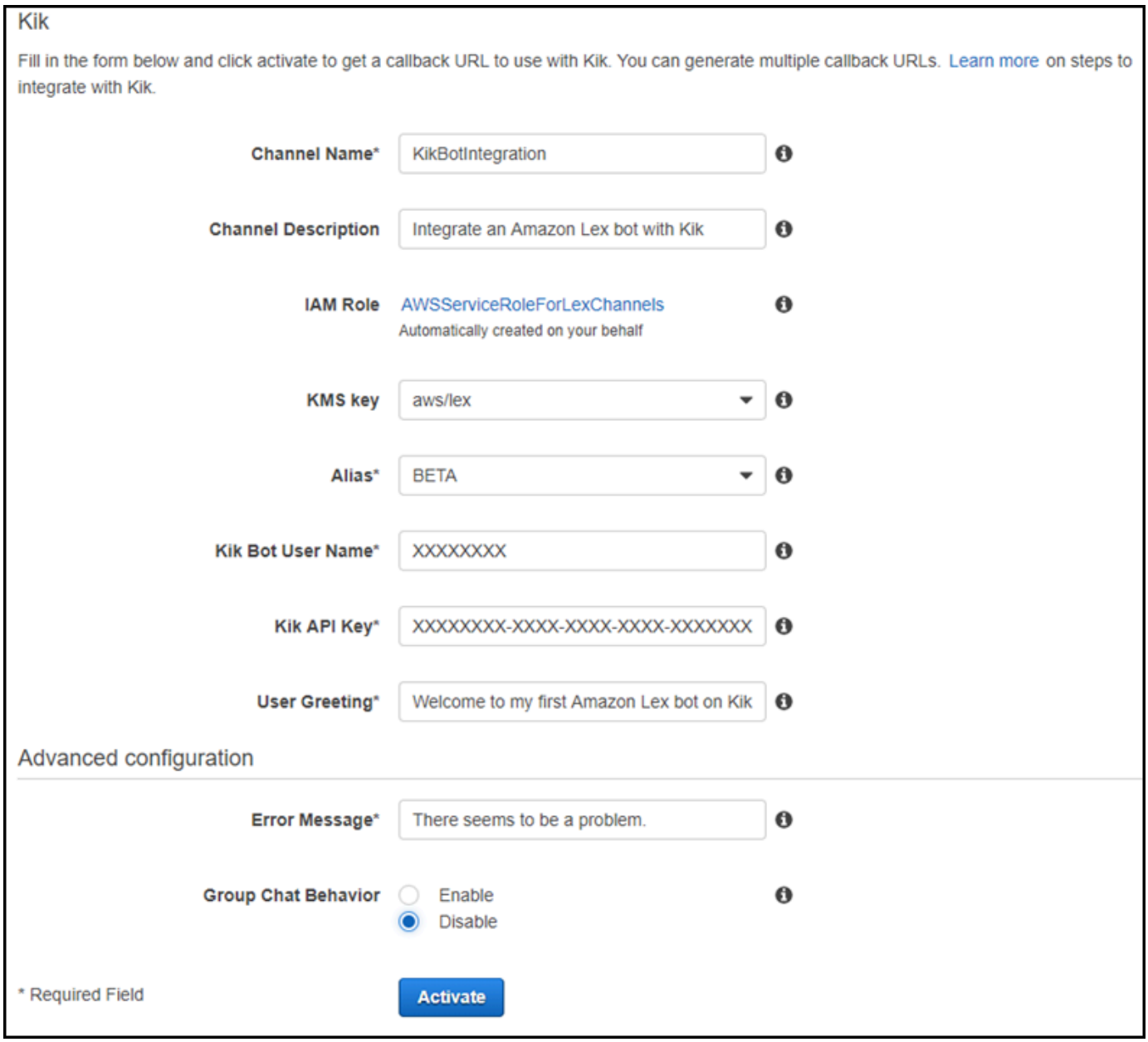
次のステップ

ステップ 3: Kik ボットと Amazon Lex ボットの統合

ステップ 3: Kik ボットと Amazon Lex ボットの統合

Amazon Lex ボットと Kik ボットの作成が完了すると、両者のチャンネル関連付けを Amazon Lex で作成できます。関連付けを有効化すると、Amazon Lex によって Kik のコールバック URL が自動的に設定されます。

1. AWS マネジメントコンソールにサインインし、Amazon Lex コンソール (https://console.aws.amazon.com/lex/) を開きます。
2. ステップ 1 で作成した Amazon Lex ボットを選択します。
3. [Channels] タブを選択します。
4. [Channels] セクションで、[Kik] を選択します。
5. Kik ページで、以下を指定します。
   - 名前を入力します。例えば、`BotKikIntegration` です。
   - 説明を入力します。
   - [KMS Key] ドロップダウンから [aws/lex] を選択します。
   - [Alias] で、ドロップダウンからエイリアスを選択します。
   - [Kik bot user name] に、Kik でボットに付けた名前を入力します。
   - [Kik API key] に、Kik でボットに割り当てた API キーを入力します。
   - [User greeting] に、ユーザーとの最初のチャットでボットから送信する挨拶を入力します。
   - [Error message] に、会話の一部が認識されないときにユーザーに表示するエラーメッセージを入力します。
   - [Group chat behavior] で、いずれかのオプションを選択します。
     - [Enable] – 単一の会話でチャットグループ全体がボットとやり取りできます。
     - [Disable] – 会話をチャットグループの特定のユーザーに制限します。
   - [Activate] を選択して関連付けを作成し、それを Kik ボットにリンクします。



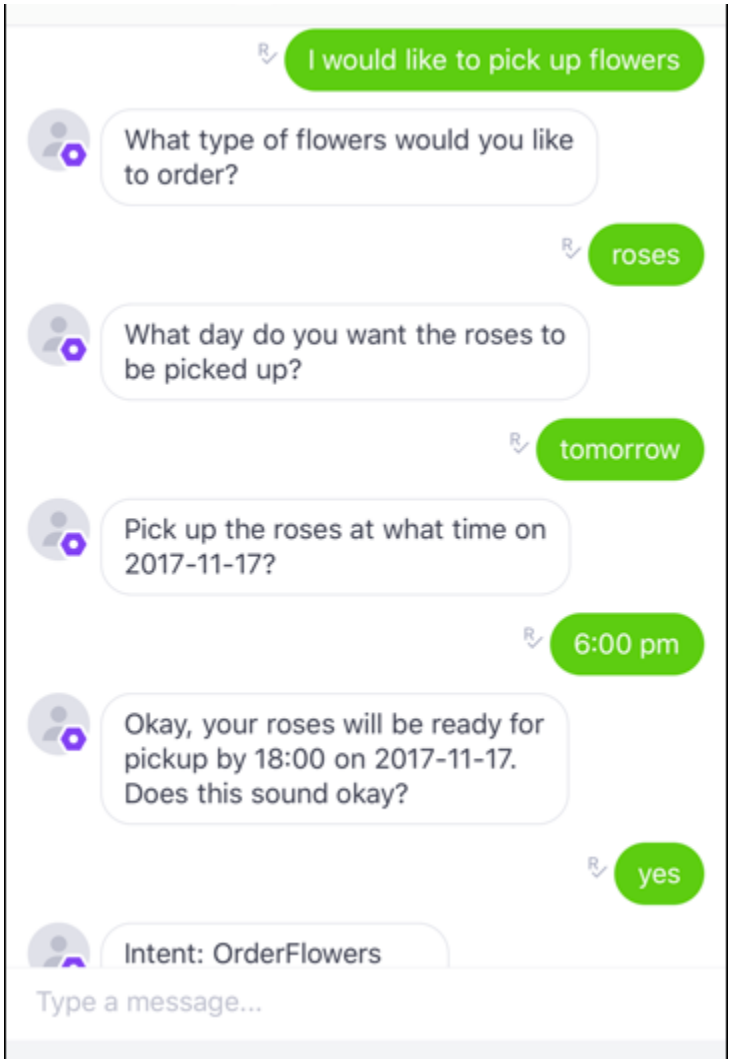
次のステップ

[ステップ 4: 統合をテストする]

ステップ 4: 統合をテストする

Amazon Lex ボットと Kik との関連付けを作成したので、次に Kik アプリを使用して関連付けをテストできます。

1. Kik アプリを起動してログインします。作成したボットを選択します。
2. 次の方法でボットをテストできます。



各フレーズを入力すると、Amazon Lex ボットは Kik を通じて応答し、各スロットの作成済みのプロンプトを表示します。

Amazon Lex ボットと Slack との統合

この演習では、Amazon Lex ボットと Slack メッセージングアプリケーションを統合する手順を示します。以下のステップを実行します。

1. Amazon Lex ボットを作成します。
2. Slack メッセージングアプリケーションを作成します。
3. Slack アプリケーションとボット Amazon Lex を統合します。

4. Amazon Lex ボットと会話して、統合をテストします。Slack アプリケーションを使用してメッセージを送信し、ブラウザウィンドウでテストします。

トピック

- ステップ 1: Amazon Lex ボットを作成する
- ステップ 2: Slack にサインアップして Slack チームを作成する
- ステップ 3: Slack アプリケーションを作成する
- ステップ 4: Slack アプリケーションと Amazon Lex ボットを統合する
- ステップ 5: Slack 統合を完了する
- ステップ 6: 統合をテストする

ステップ 1: Amazon Lex ボットを作成する

Amazon Lex ボットをまだ持っていない場合には、作成してデプロイします。このトピックでは、「開始方法」の演習 1 で作成したボットを使用することを想定しています。ただし、このガイドで提供されている他のボット例を使用してもかまいません。「ご利用開始にあたって」の演習 1 については、「演習 1: 設計図を使用して Amazon Lex ボットを作成する (コンソール)」を参照してください。

1. Amazon Lex ボットを作成します。手順については、「演習 1: 設計図を使用して Amazon Lex ボットを作成する (コンソール)」を参照してください。
2. ボットをデプロイしてエイリアスを作成します。手順については、「演習 3: バージョンを発行してエイリアスを作成する」を参照してください。

次のステップ

ステップ 2: Slack にサインアップして Slack チームを作成する

ステップ 2: Slack にサインアップして Slack チームを作成する

Slack アカウントにサインアップして Slack チームを作成します。手順については、「Slack の使用」を参照してください。次のセクションでは、Slack アプリケーションを作成します。このアプリケーションには、すべての Slack チームがインストールできます。

次のステップ

ステップ 3: Slack アプリケーションを作成する

ステップ 3: Slack アプリケーションを作成する

このセクションでは、以下の作業を行います。

1. Slack API コンソールで Slack アプリケーションを作成します。
2. ボットにインタラクティブメッセージングを追加するようアプリケーションを設定します。

このセクションの最後で、アプリケーションの認証情報 (クライアント ID、クライアントシークレット、および検証トークン) が提供されます。次のセクションでは、この情報を使用して Amazon Lex コンソールでボットチャンネル関連付けを設定します。

1. Slack API コンソール (http://api.slack.com) にサインインします。
2. アプリケーションを作成します。

   アプリケーションの作成が正常に完了すると、Slack にアプリケーションの [Basic Information] ページが表示されます。
3. アプリケーションの機能を次のように設定します。

   - 左のメニューで、[Interactivity & Shortcuts] (インタラクティブ性とショートカット) を選択します。

     - トグルを選択して、インタラクティブなコンポーネントをオンにします。
     - [Request URL] ボックスで、任意の有効な URL を指定します。たとえば、**`https://slack.com`** を使用できます。

       > **Note**
       >
       > ここでは、任意の有効な URL を入力し、次のステップで必要な検証トークンを取得します。この URL は、Amazon Lex コンソールでボットチャンネル関連付けを追加した後に更新します。

     - [Save changes] (変更の保存) をクリックします。
4. 左のメニューで、[Settings] の [Basic Information] を選択します。次のアプリケーション認証情報を記録します。

   - クライアント ID

- クライアントシークレット
- 検証トークン

次のステップ

ステップ 4: Slack アプリケーションと Amazon Lex ボットを統合する

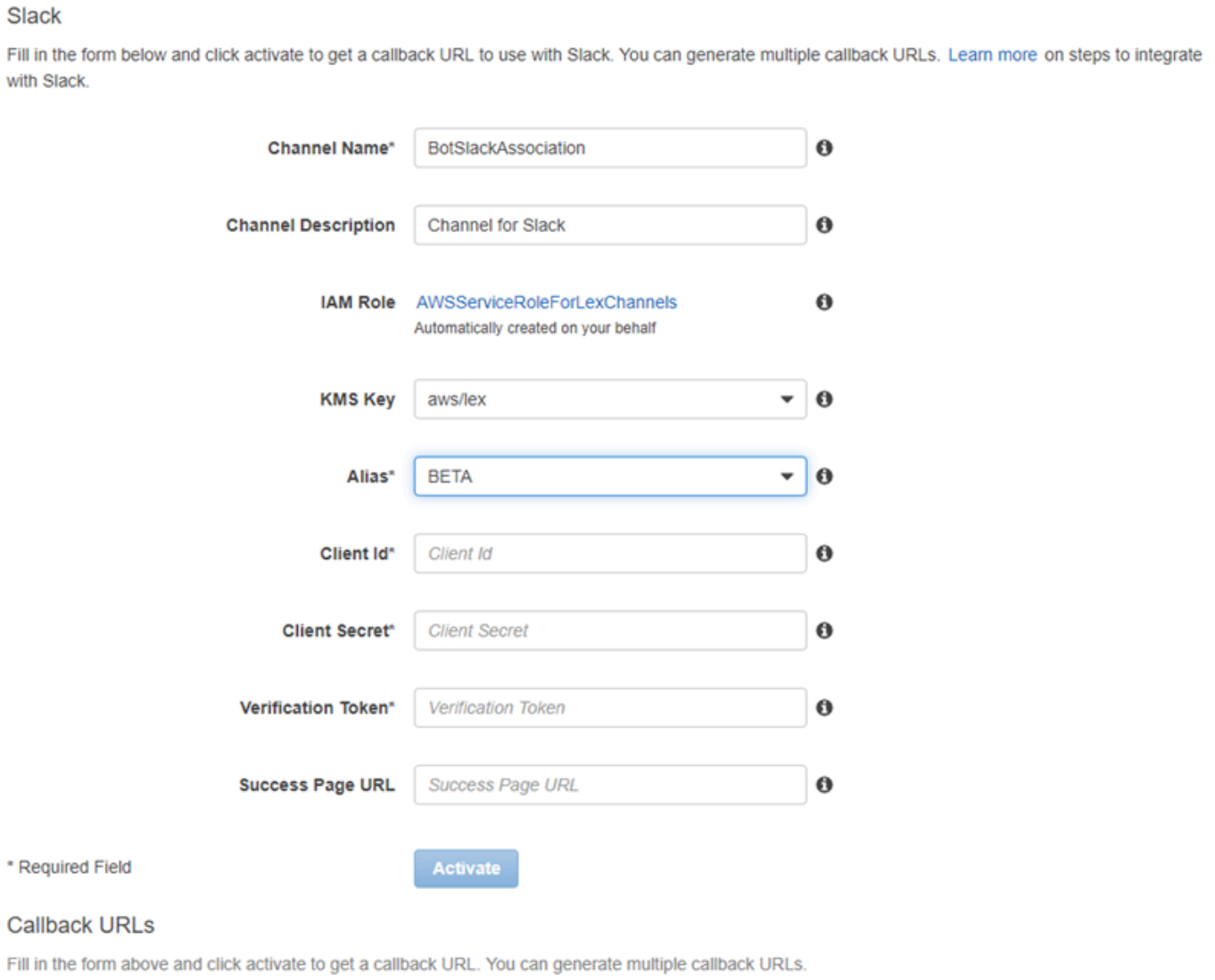
ステップ 4: Slack アプリケーションと Amazon Lex ボットを統合する

Slack アプリケーションの認証情報を取得したので、アプリケーションと Amazon Lex ボットを統合できます。Slack アプリケーションとボットを関連付けるには、Amazon Lex にボットチャンネル関連付けを追加します。

Amazon Lex コンソールで、ボットチャンネル関連付けを有効化し、ボットと Slack アプリケーションを関連付けます。ボットチャンネル関連付けが有効化されると、Amazon Lex は 2 つの URL (Postback URL と OAuth URL) を返します。後で必要になるため、これらの URL を記録します。

Slack アプリケーションと Amazon Lex ボットを統合するには

1. AWS マネジメントコンソールにサインインし、Amazon Lex コンソール (https://console.aws.amazon.com/lex/) を開きます。
2. ステップ 1 で作成した Amazon Lex ボットを選択します。
3. [Channels] タブを選択します。
4. 左側のメニューで、[Slack] を選択します。
5. [Slack] ページで、以下を指定します。
   - 名前を入力します。例えば、`BotSlackIntegration` です。
   - [KMS Key] ドロップダウンから [aws/lex] を選択します。
   - [Alias] で、ボットのエイリアスを選択します。
   - 前のステップで記録した [Client Id]、[Client secret]、[Verification Token] を入力します。これらは Slack アプリケーションの認証情報です。



6. [アクティブ化] を選択します。

   コンソールでは、ボットチャンネルの関連付けを行い、2 つの URL を返します (Postback URL と OAuth URL)。それを記録します。次のセクションでは、Slack アプリケーションの設定を次のように更新して、これらのエンドポイントを使用します。

   - Postback URL は、Slack イベントを待機する Amazon Lex ボットのエンドポイントです。この URL は以下の目的に使用します。
     - Slack アプリケーションの [Event Subscriptions] 機能でリクエスト URL として使用します。

- Slack アプリケーションの [Interactive Messages] 機能で、リクエスト URL のプレースホルダ値を置き換えます。
- OAuth URL は、Slack との OAuth ハンドシェイクにおいて Amazon Lex ボット側のエンドポイントです。

次のステップ

ステップ 5: Slack 統合を完了する

ステップ 5: Slack 統合を完了する

このセクションでは、Slack API コンソールを使用して Slack アプリケーションの統合を完了します。

1. Slack API コンソール (http://api.slack.com) にサインインします。「ステップ 3: Slack アプリケーションを作成する」で作成したアプリを選択します。
2. [OAuth & Permissions] 機能を次のように更新します。
   a. 左のメニューで、[OAuth & Permissions] を選択します。
   b. [Redirect URLs] (URL のリダイレクト) セクションで、前のステップで Amazon Lex から提供された OAuth URL を追加します。[Add a new Redirect URL]、[Save URLs] の順に選択します。
   c. Bot Token Scopes (ボットトークンスコープ) セクションでは、Add an OAuth Scope (OAuth スコープを追加する) ボタンで 2 つのパーミッションを追加します。次のテキストを使用してリストをフィルタリングします。
      - **`chat:write`**
      - **`team:read`**
3. [Request URL] (リクエスト URL) の値を、前のステップで Amazon Lex から返された Postback URL に更新して、[Interactive Components] (インタラクティブコンポーネント) 機能を更新します。ステップ 4 で保存した postback URL を参照し、[Save Changes] を選択します。
4. 次のように [Event Subscriptions] 機能にサブスクライブします。
   - [On] オプションを選択してイベントを有効化します。
   - [Request URL] (リクエスト URL) の値として、前のステップで Amazon Lex から返された Postback URL を設定します。

- [Subscribe to Bot Events] セクションで、`message.im` ボットイベントにサブスクライブして、エンドユーザーと Slack ボット間の直接メッセージングを有効にします。
- 変更を保存します。

5. [メッセージ] タブからのメッセージの送信を次のように有効にします。

   - 左のメニューで、[App Home] (アプリケーションホーム) をクリックします。
   - タブの表示 セクションで、[メッセージ] タブからユーザーが Slash コマンドとメッセージを送信することを許可する を選択します。

次のステップ

[ステップ 6: 統合をテストする](#)

ステップ 6: 統合をテストする

ここでは、ブラウザウィンドウを使用して、Slack と Amazon Lex ボットとの統合をテストします。

1. [Settings] の [Manage Distribution] を選択します。[Add to Slack] を選択してアプリケーションをインストールします。ボットがメッセージに応答することを承認します。
2. Slack チームにリダイレクトされます。左のメニューの [Direct Messages] セクションで、ボットを選択します。ボットが表示されていない場合は、[Direct Messages] の横にあるプラスアイコン (+) を選択してボットを探します。
3. Amazon Lex ボットにリンクされている Slack アプリケーションとチャットを開始します。ボットがメッセージに応答するようになりました。

   「ご利用開始にあたって」の演習 1 でボットを作成した場合は、その演習で提供されている会話例が使用できます。詳細については、「[ステップ 4: Lambda 関数をコードフックとして追加する (コンソール)](#)」を参照してください。

Amazon Lex ボットと Twilio プログラム可能 SMS の統合

この演習では、Amazon Lex ボットを Twilio 簡易メッセージングサービス (SMS) に統合する手順を示します。以下のステップを実行します。

1. Amazon Lex ボットを作成する
2. Twilio プログラム可能 SMS とボット Amazon Lex を統合する

3. 携帯電話の SMS サービスを使用して Amazon Lex ボットと交信し、セットアップをテストする
4. 統合をテストする

トピック

ステップ 1: Amazon Lex ボットを作成する

Amazon Lex ボットをまだ持っていない場合には、作成してデプロイします。このトピックでは、「開始方法」の演習 1 で作成したボットを使用することを想定しています。ただし、このガイドで提供されている他のボット例を使用してもかまいません。「ご利用開始にあたって」の演習 1 については、「演習 1: 設計図を使用して Amazon Lex ボットを作成する (コンソール)」を参照してください。

1. Amazon Lex ボットを作成します。手順については、「演習 1: 設計図を使用して Amazon Lex ボットを作成する (コンソール)」を参照してください。
2. ボットをデプロイしてエイリアスを作成します。手順については、「演習 3: バージョンを発行してエイリアスを作成する」を参照してください。

ステップ 2: Twilio SMS アカウントを作成する

Twilio アカウントにサインアップして、次のアカウント情報を書き留めます。

- ACCOUNT SID
- AUTH TOKEN

サインアップの手順については、「https://www.twilio.com/console」を参照してください。

ステップ 3: Twilio メッセージングサービスエンドポイントと Amazon Lex ボットを統合する

Twilio と Amazon Lex ボットを統合するには

1. Amazon Lex ボットを Twilio プログラム可能 SMS エンドポイントに関連付けるには、Amazon Lex コンソールでボットチャンネル関連付けを有効化します。ボットチャンネル関連付けが有効化されると、Amazon Lex からコールバック URL が返されます。このコールバック URL を書き留めます。後で必要になります。

   a. AWS Management Console にサインインし、Amazon Lex コンソール (https://console.aws.amazon.com/lex/) を開きます。

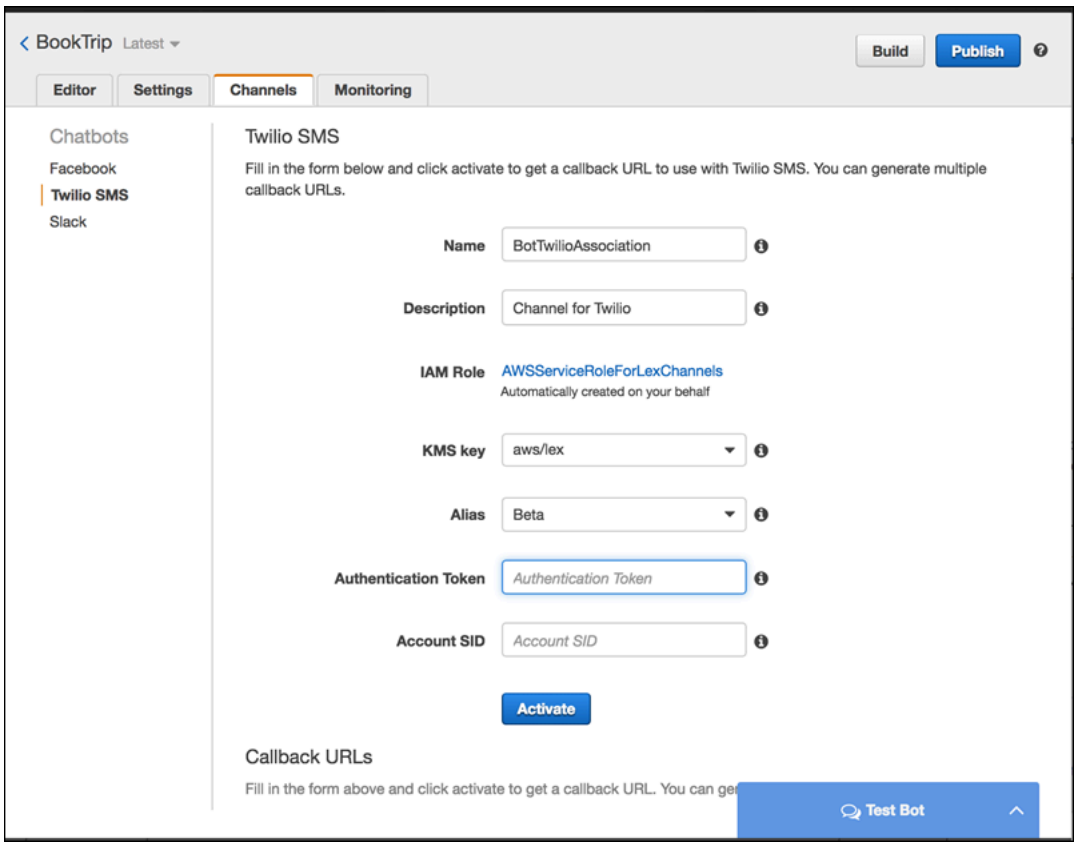
   b. ステップ 1 で作成した Amazon Lex ボットを選択します。

   c. [Channels] タブを選択します。

   d. [Chatbots] セクションで、[Twilio SMS] を選択します。

   e. [Twilio SMS] ページで、以下の情報を入力します。

      - 名前を入力します。例えば、`BotTwilioAssociation` です。
      - [KMS key] (KMS キー) で「aws/lex」を選択します。
      - [Alias] で、ボットのエイリアスを選択します。
      - [Authentication Token] に Twilio アカウントの AUTH TOKEN を入力します。
      - [Account SID] に Twilio アカウントの ACCOUNT SID を入力します。



f. [アクティブ化] を選択します。

コンソールによってボットチャンネル関連付けが作成され、コールバック URL が返されます。この URL を記録します。

2. Twilio コンソールで、Twilio SMS エンドポイントを Amazon Lex ボットに接続します。

a. Twilio コンソール (https://www.twilio.com/console) にサインインします。

b. Twilio SMS エンドポイントがない場合は、作成します。

c. [REQUEST URL] (リクエスト URL) の値として、前のステップで Amazon Lex から返されたコールバック URL を設定し、メッセージングサービスの [Inbound Settings] (インバウンド設定) 設定を更新します。

ステップ 4: 統合をテストする

携帯電話を使用して、Twilio SMS とボット間の統合をテストします。

統合をテストするには

1. Twilio コンソール (https://www.twilio.com/console) にサインインして、次の操作を行います。

   a. [Manage Numbers] に、メッセージングサービスに関連付けられた Twilio 番号が表示されていることを確認します。

      この番号に携帯電話からメッセージを送信し、Amazon Lex ボットと SMS で交信します。

   b. 携帯電話が [Verified Caller ID] (確認済み発信者 ID) としてリストに登録されていることを確認します。

      登録されていない場合は、Twilio コンソールの指示に従って、テストに使用する携帯電話を使用可能にします。

      これで、携帯電話を使用して、Amazon Lex ボットにマッピングされている Twilio SMS エンドポイントに、メッセージを送信できるようになりました。

2. 携帯電話を使用して、Twilio 番号にメッセージを送信します。

   Amazon Lex ボットが応答します。「ご利用開始にあたって」の演習 1 でボットを作成した場合は、その演習で提供されている会話例が使用できます。詳細については、「ステップ 4: Lambda 関数をコードフックとして追加する (コンソール)」を参照してください。

モバイルアプリケーションで Amazon Lex ボットをデプロイする

AWS Amplify を使用することで、Amazon Lex ボットをモバイルやウェブアプリケーションと統合することができます。詳細については、「AWS Amplify ドキュメント」の「Interactions – Getting started」(インタラクション — 入門方法) を参照してください。

Amazon Lex ボット、インテント、スロットタイプのインポートとエクスポート

ボット、インテント、スロットタイプをインポートあるいはエクスポートできます。例えば、異なる AWS アカウント間で同僚とボットを共有する場合には、エクスポートとして送信することができます。複数の発話をボットに追加するには、ボットをエクスポートして発話を追加し、それをアカウントに再度インポートできます。

ボット、インテントおよびスロットタイプのエクスポートは、Amazon Lex (共有あるいは変更のため) あるいは Alexa スキル形式の両方でできます。インポートは Amazon Lex 形式のみでできます。

リソースをエクスポートする場合、エクスポート先の Amazon Lex あるいは Alexa スキルキットのサービスと互換性のある形式でエクスポートする必要があります。ボットを Amazon Lex 形式でエクスポートする場合、自身のアカウントに再度インポートすることも、別のアカウントの Amazon Lex ユーザーがアカウントにそれをインポートすることもできます。ボットは、Alexa スキルと互換性がある形式でエクスポートすることもできます。そして、Alexa スキルキットを使用し、そのボットをインポートして、Alexa で利用できるようにします。詳細については、「Alexa スキルへのエクスポート」を参照してください。

スロットタイプをエクスポートする場合、そのリソースは JSON ファイルで書き込まれています。ボット、インテントあるいはスロットタイプをエクスポートするには、Amazon Lex コンソールあるいは GetExport オペレーションを使用できます。StartImport を使用して、ボット、インテントあるいはスロットタイプをインポートします。

トピック

- Amazon Lex 形式でのインポートとエクスポート
- Alexa スキルへのエクスポート

Amazon Lex 形式でのインポートとエクスポート

後に Amazon Lex に再インポートするために Amazon Lex からボット、インテント、スロットタイプをエクスポートするには、Amazon Lex 形式で JSON ファイルを作成します。このファイルではリソースを編集でき、Amazon Lex に再インポートできます。例えば、インテントに発話を追加し、変更したインテントをアカウントに再インポートできます。また、JSON 形式を使用してリソース

を共有することもできます。例えば、1 つの AWS リージョンからボットをエクスポートして、別のリージョンにインポートできます。または、同僚に JSON ファイルを送信してボットを共有できます。

トピック

Amazon Lex 形式でエクスポートする

AWS アカウントにインポートできる形式で、 Amazon Lex ボット、インテント、スロットタイプをエクスポートします。次のリソースをエクスポートできます。

- ボットに使用されるすべてのインテントおよびカスタムスロットを含むボット
- インテントに使用されるすべてのインテントおよびカスタムスロットを含むインテント
- そのスロットタイプのすべての値を含むカスタムスロットタイプ

番号付きのバージョンのリソースのみをエクスポートできます。リソースの $LATEST バージョンをエクスポートすることはできません。

エクスポートは非同期プロセスです。エクスポートが完了すると、Amazon S3 の署名付き URL を取得します。この URL は、エクスポートされた JSON 形式のリソースが含まれる .zip アーカイブの場所を示します。

ボット、インテント、カスタムスロットタイプをエクスポートするには、コンソールあるいは GetExport オペレーションを使用します。

ボット、インテント、あるいはスロットタイプをエクスポートする手順は同じです。次の手順で、インテントあるいはスロットタイプをボットに置き換えます。

ボットをエクスポートする

ボットをエクスポートするには

1. AWS マネジメントコンソールにサインインし、Amazon Lex コンソール (https://console.aws.amazon.com/lex/) を開きます。

2. [ボット] を選択し、エクスポートする 1 つのボットを選びます。
3. [アクション] メニューで、[エクスポート] を選択します。
4. [Export Bot] (ボットのエクスポート) ダイアログで、エクスポートするボットのバージョンを選択します。[プラットフォーム] で [Amazon Lex] を選択します。
5. [エクスポート] をクリックします。
6. .zip アーカイブをダウンロードして保存します。

Amazon Lex は、.zip アーカイブに含まれる JSON ファイルにボットをエクスポートします。ボットを更新するには、JSON テキストを変更し、それを Amazon Lex に再インポートします。

次のステップ

Amazon Lex 形式でインポートする

Amazon Lex 形式でインポートする

Amazon Lex 形式でリソースを JSON ファイルにエクスポートしたら、リソースが含まれる JSON ファイルを 1 つ以上 の AWS アカウントにインポートできます。例えば、ボットをエクスポートして、別の AWS リージョンにインポートできます。または、ボットを同僚に送信して、この同僚が自身のアカウントにボットをインポートすることもできます。

ボット、インテントあるいはスロットタイプをインポートするとき、インテントやスロットタイプなどの $LATEST バージョンをインポート中に上書きするか、またはアカウント内のリソースを維持するためにこのインポートを失敗するようにするかを決定する必要があります。例えば、編集されたバージョンのリソースをアカウントにアップロードする場合、$LATEST バージョンを上書きするかどうかを選択します。同僚から送信されたリソースをアップロードする場合、自身のリソースが置き換えられないために、リソース間に競合が発生した場合にこのインポートを失敗するように選択できます。

リソースをインポートするとき、インポートを実行するユーザーに割り当てられたアクセス許可から適用がリクエストされます。このユーザーは、アカウント内でこのインポートが影響するすべてのリソースに対してアクセス許可を有している必要があります。また、このユーザーは、GetBot、PutBot、GetIntent、PutIntent、GetSlotType、PutSlotType オペレーションへのアクセス許可も有している必要があります。権限の詳細については、「Amazon Lex で IAM を使用する方法」を参照してください。

インポートは、処理中に発生したエラーを報告します。一部のエラーはインポートが開始する前に報告され、その他はインポート処理中に報告されます。例えば、インテントが使用する Lambda 関数

を呼び出すアクセス許可がないインテントをアカウントでインポートする場合、このインポートはスロットタイプあるいはインテントが変更される前に失敗します。インポート処理中にインポートに失敗した場合、この処理が失敗する前にインポートされたすべてのインテントあるいはスロットタイプの $LATEST バージョンは変更されます。$LATEST バージョンに加えられた変更をロールバックすることはできません。

リソースをインポートする場合、すべての依存するリソースはリソースの $LATEST バージョンにインポートされ、番号付きバージョンを付与されます。例えば、ボットがインテントを使用する場合、このインテントには番号付きバージョンが付与されます。インテントはカスタムスロットタイプを使用する場合、このスロットタイプには番号付きバージョンが付与されます。

1 つのリソースは 1 度のみインポートされます。例えば、ボットに `OrderPizza` インテントおよび `OrderDrink` インテントが含まれ、どちらのインテントもカスタムスロットタイプ `Size` に依存している場合、`Size` スロットタイプは 1 度のみインポートされ、両方のインテントに使用されます。

> **Note**
>
> ボットをでエクスポートした場合 `enableModelImprovements` パラメータをに設定します。`false` では、ボット定義を含む.zip ファイルを開き、`enableModelImprovements` パラメータを `true` は以下のリージョンで指定します。
>
> - アジアパシフィック (シンガポール) (ap-southeast-1)
> - アジアパシフィック (東京) (ap-northeast-1)
> - 欧州 (フランクフルト) (eu-central-1)
> - 欧州 (ロンドン) (eu-west-2)

ボット、インテント、あるいはカスタムスロットタイプをインポートする手順は同じです。次の手順で、インテントあるいはスロットタイプを適当に置き換えます。

ボットのインポート

ボットをインポートするには

1. AWS マネジメントコンソールにサインインし、Amazon Lex コンソール (https://console.aws.amazon.com/lex/) を開きます。
2. [ボット] を選択し、インポートする 1 つのボットを選びます。新規のボットをインポートする場合、このステップはスキップします。

3. [アクション] で、[インポート] を選択します。
4. [Import Bot] (ボットのインポート) で、インポートするボットが含まれる JSON ファイルがある .zip アーカイブを選択します。マージする前にマージ競合を表示するには、[Notify me of merge conflicts] (マージ競合を通知) を選択します。競合チェックをオフにした場合、ボットが使用するすべてのリソースの $LATEST バージョンは上書きされます。
5. [Import] (インポート) を選択します。マージ競合の通知を選択し、競合が発生した場合、この競合の一覧を示すダイアログが表示されます。競合するすべてのリソースの $LATEST バージョンを上書きするには、[Overwrite and continue (上書きして続ける)] を選択します。インポートを停止するには、[キャンセル] を選択します。

これで、アカウントでボットをテストできるようになります。

インポートとエクスポートにおける JSON 形式

次の例では、スロットタイプ、インテントおよびボットを Amazon Lex 形式でエクスポート、インポートする JSON 構造を示しています。

スロットタイプ構造

カスタムスロットタイプの JSON 構造を以下に示します。スロットタイプをインポートあるいはエクスポートする場合、またカスタムスロットタイプに依存するインテントをエクスポートする場合にこの構造を使用します。

```
{
  "metadata": {
    "schemaVersion": "1.0",
    "importType": "LEX",
    "importFormat": "JSON"
  },
  "resource": {
    "name": "slot type name",
    "version": "version number",
    "enumerationValues": [
      {
        "value": "enumeration value",
        "synonyms": []
      },
      {
        "value": "enumeration value",
```

```
        "synonyms": []
      }
    ],
    "valueSelectionStrategy": "ORIGINAL_VALUE or TOP_RESOLUTION"
  }
}
```

インテントの構造

次に、インテントの JSON 構造を示します。インテントおよびインテントに依存するボットをインポート、エクスポートする場合にこの構造を使用します。

```
{
  "metadata": {
    "schemaVersion": "1.0",
    "importType": "LEX",
    "importFormat": "JSON"
  },
  "resource": {
    "description": "intent description",
    "rejectionStatement": {
      "messages": [
        {
          "contentType": "PlainText or SSML or CustomPayload",
          "content": "string"
        }
      ]
    },
    "name": "intent name",
    "version": "version number",
    "fulfillmentActivity": {
      "type": "ReturnIntent or CodeHook"
    },
    "sampleUtterances": [
      "string",
      "string"
    ],
    "slots": [
      {
        "name": "slot name",
        "description": "slot description",
        "slotConstraint": "Required or Optional",
        "slotType": "slot type",
```

```
        "valueElicitationPrompt": {
          "messages": [
            {
              "contentType": "PlainText or SSML or CustomPayload",
              "content": "string"
            }
          ],
          "maxAttempts": value
        },
        "priority": value,
        "sampleUtterances": []
      }
    ],
    "confirmationPrompt": {
      "messages": [
        {
          "contentType": "PlainText or SSML or CustomPayload",
          "content": "string"
        },
        {
          "contentType": "PlainText or SSML or CustomPayload",
          "content": "string"
        }
      ],
      "maxAttempts": value
    },
    "slotTypes": [
        List of slot type JSON structures.
        For more information, see #########.
    ]
  }
}
```

ボット構造

次に、ボットの JSON 構造を示します。ボットをインポートまたはエクスポートするときにこの構造を使用します。

```
{
  "metadata": {
    "schemaVersion": "1.0",
    "importType": "LEX",
    "importFormat": "JSON"
```

```
  },
  "resource": {
    "name": "bot name",
    "version": "version number",,
    "nluIntentConfidenceThreshold": 0.00-1.00,
    "enableModelImprovements": true | false,
    "intents": [
        List of intent JSON structures.
        For more information, see ########.
    ],
    "slotTypes": [
        List of slot type JSON structures.
        For more information, see #########.
    ],
    "voiceId": "output voice ID",
    "childDirected": boolean,
    "locale": "en-US",
    "idleSessionTTLInSeconds": timeout,
    "description": "bot description",
    "clarificationPrompt": {
      "messages": [
        {
          "contentType": "PlainText or SSML or CustomPayload",
          "content": "string"
        }
      ],
      "maxAttempts": value
    },
    "abortStatement": {
      "messages": [
        {
          "contentType": "PlainText or SSML or CustomPayload",
          "content": "string"
        }
      ]
    }
  }
}
```

Alexa スキルへのエクスポート

ボットスキーマは、Alexa Skill と互換性がある形式でエクスポートできます。ボットを JSON ファイルにエクスポートしたら、Skill Builder を使用してそれを Alexa にアップロードします。

ボットとそのスキーマをエクスポートするには (インタラクションモデル)

1. AWS Management Console にサインインし、Amazon Lex コンソール (https://console.aws.amazon.com/lex/) を開きます。
2. エクスポートするボットを選択します。
3. [アクション] で、[エクスポート] を選択します。
4. エクスポートするボットのバージョンを選択します。形式では [Alexa Skills Kit] を選択し、続いて [エクスポート] を選択します。
5. ダウンロードダイアログボックスが表示されたら、ファイルを保存する場所を選択し、[保存] を選択します。

ダウンロードされるファイルは、エクスポートしたボットと同じ名前の 1 つのファイルが含まれる .zip アーカイブです。このファイルには、Alexa スキルとしてボットをインポートするために必要な情報が含まれています。

> **Note**
>
> Amazon Lex および Alexa Skills Kit では、次の点が異なります。
>
> - 括弧 ([]) で示されるセッション属性は、Alexa Skills Kit ではサポートされません。セッション属性を使用しているプロンプトは更新する必要があります。
> - 句読点は、Alexa Skills Kit でサポートされません。句読点を使用している発話は更新する必要があります。

Alexa スキルにボットをアップロードするには

1. 開発者ポータル (https://developer.amazon.com/) にログインします。
2. [Alexa スキル] ページで、[スキルの作成] を選択します。
3. [新しいスキルの作成] ページに、スキル名と、スキルのデフォルト言語を入力します。スキルモデルに [カスタム] が選択されていることを確認し、[スキルの作成] を選択します。
4. [一から作成] が選択されていることを確認し、[選択] を選択します。
5. 左のメニューで [JSON テキストエディタ] を選択します。Amazon Lex からエクスポートした JSON ファイルを JSON テキストエディタにドラッグします。
6. [モデルの保存] を選択して、インタラクションモデルを保存します。

スキーマを Alexa スキル内にアップロードしたら、Alexa でスキルを実行するために必要な変更を行います。Alexa スキルを作成する方法の詳細については、「Alexa Skills Kit」の「Use Skill Builder (Beta)」を参照してください。

その他の例: Amazon Lex ボットの作成

以下のセクションでは、Amazon Lex の追加の演習の手順について説明します。

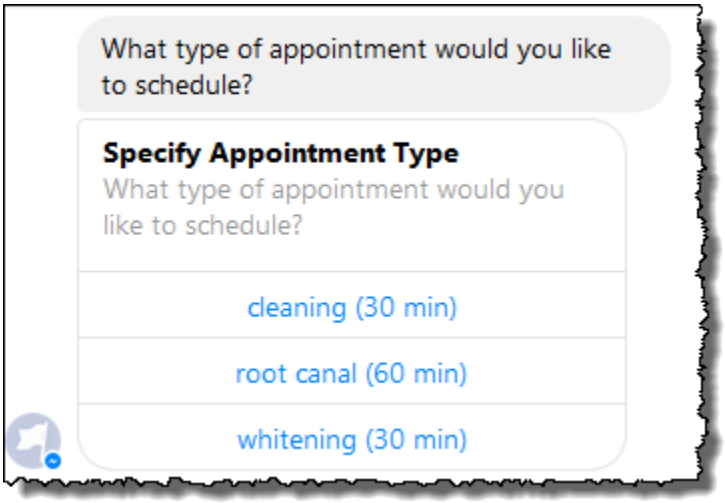
トピック

予約のスケジュール

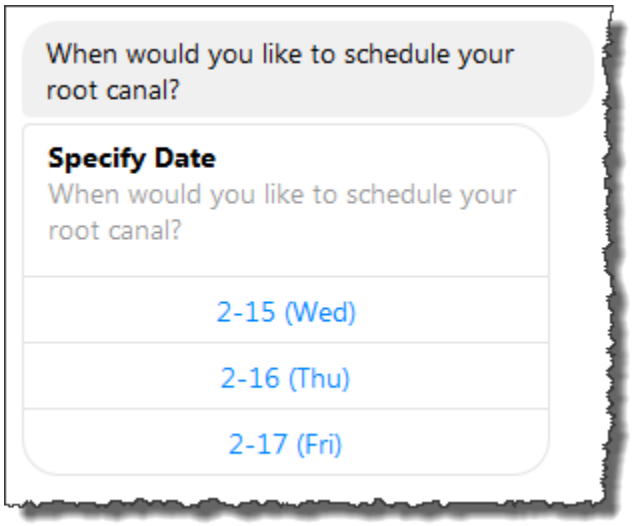
この演習のボット例では、歯科医院の予約を行います。この例では、レスポンスカードを使用してボタンでユーザー入力を取得する方法も示しています。具体的には、この例では、実行時に動的にレスポンスカードを生成しています。

構築時にレスポンスカードを設定するか (静的レスポンスカードとも呼ばれる)、または AWS Lambda 関数内で動的にレスポンスカードを生成できます。この例のボットでは、以下のレスポンスカードを使用します。

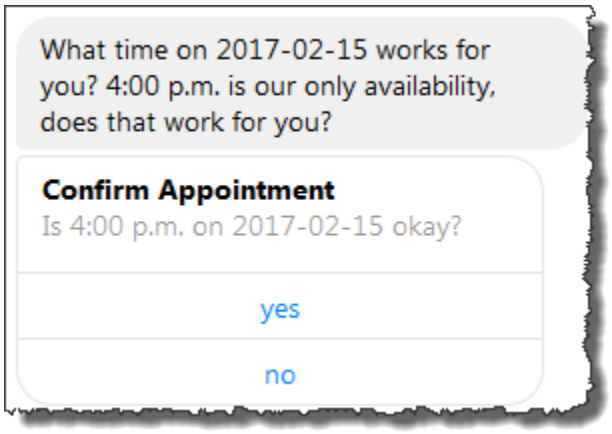
- 予約タイプのボタンをリストするレスポンスカード。例については、次の画像を参照してください。



- 予約日付のボタンをリストするレスポンスカード。例については、次の画像を参照してください。



- 提示された予約時刻を確認するボタンをリストするレスポンスカード。例については、次の画像を参照してください。



予約できる日付と時刻は変わるため、実行時にレスポンスカードを作成する必要があります。AWS Lambda 関数を使用してこれらのレスポンスカードを動的に生成します。Lambda 関数は、Amazon Lex へのレスポンス内でレスポンスカードを返します。Amazon Lex は、クライアントへのレスポンス内にレスポンスカードを含めます。

クライアント (例えば、Facebook Messenger) でレスポンスカードがサポートされている場合、ユーザーはボタンのリストから選択するか、またはレスポンスを入力します。サポートされていない場合、ユーザーはレスポンスを入力します。

前の例で示されているボタンに加えて、イメージ、添付ファイル、およびその他の役立つ情報をレスポンスカードに表示することもできます。レスポンスカードに関する情報については、「レスポンスカード」を参照してください。

この演習では、以下のことを行います。

- ボット (ScheduleAppointment 設計図を使用) を作成してテストします。この演習では、ボットの設計図を使用して、迅速にボットをセットアップし、テストします。使用可能な設計図の一覧については、「Amazon Lex および AWS Lambda の設計図」を参照してください。このボットには 1 つのインテント (MakeAppointment) が事前設定されています。

- Lambda 関数を作成してテストします (Lambda で提供されている lex-make-appointment-python 設計図を使用)。その Lambda 関数をコードフックとして使用して初期化、検証、およびフルフィルメント (達成) タスクを実行するように、MakeAppointment インテントを設定します。

  > **Note**
  >
  > 提供されている Lambda 関数の例では、歯科医予約のモックアップの予約可能日時に基づいて動的な会話を示しています。実際のアプリケーションでは、実際のカレンダーを使用して予定を設定できます。

- その Lambda 関数をコードフックとして使用するように、MakeAppointment インテント設定を更新します。次に、エンドツーエンドエクスペリエンスをテストします。
- 作動中のレスポンスカードを確認できるように、そのスケジュール予約ボットを Facebook Messenger に公開します (Amazon Lex コンソール内のクライアントではレスポンスカードは現在サポートされていません)。

以下のセクションでは、この演習で使用する設計図に関する概要情報を示しています。

トピック

- ボット設計図 (ScheduleAppointment) の概要
- Lambda 関数の設計図 (lex-make-appointment-python) の概要
- ステップ 1: Amazon Lex ボットを作成する
- ステップ 2: Lambda 関数を作成する
- ステップ 3: インテントの更新: コードフックを設定する

- ステップ 4: ボットを Facebook Messenger プラットフォームにデプロイする
- 情報フローの詳細

ボット設計図 (ScheduleAppointment) の概要

この演習でボットの作成に使用する ScheduleAppointment 設計図は以下のように事前設定されています。

- スロットタイプ – 1 つのカスタムスロットタイプ (`AppointmentTypeValue`) と列挙値 (`root canal`、`cleaning`、`whitening`)。
- インテント – 1 つのインテント (`MakeAppointment`)。次のように事前設定されています。
  - スロット – このインテントでは以下のスロットが設定されています。
    - スロット `AppointmentType`: `AppointmentTypes` カスタムタイプ
    - スロット `Date`: `AMAZON.DATE` 組み込みタイプ
    - スロット `Time`: `AMAZON.TIME` 組み込みタイプ
  - 発話 – このインテントでは以下の発話が事前設定されています。
    - 「予約をお願いします」
    - 「予約します」
    - 「{AppointmentType} を予約」

    ユーザーがこのいずれかを発声すると、Amazon Lex は `MakeAppointment` がインテントであると判断し、プロンプトを使用してスロットデータを引き出します。
  - プロンプト – このインテントでは以下のプロンプトが事前設定されています。
    - `AppointmentType` スロットのプロンプト –「どのタイプを予約なさいますか?」
    - `Date` スロットのプロンプト –「何日に {AppointmentType} を予約なさいますか?」
    - `Time` スロットのプロンプト –「何時に {AppointmentType} を予約なさいますか?」 and

      「{Date} の何時になさいますか?」
    - 確認プロンプト –「{Time} は予約できますが、この時刻で予約してよろしいでしょうか?」
    - キャンセルメッセージ –「かしこまりました、予約は行いません。」

Lambda 関数の設計図 (lex-make-appointment-python) の概要

Lambda 関数の設計図 (lex-make-appointment-python) は、ScheduleAppointment ボットの設計図を使用して作成したボット用のコードフックです。

この Lambda 関数の設計図のコードは、初期化/検証とフルフィルメントタスクの両方を実行できます。

- この Lambda 関数のコードは、歯科医予約のサンプル予約可能日時に基づいた動的会話を示しています (実際のアプリケーションではカレンダーを使用できます)。ユーザーが指定した曜日または日付について、このコードは次のように設定されています。
  - 予約可能な時間がない場合、Lambda 関数は、ユーザーに別の曜日または日付の入力を求めるように Amazon Lex に指示するレスポンスを返します (`dialogAction` タイプを `ElicitSlot`) に設定)。詳細については、「レスポンスの形式」を参照してください。
  - 指定された曜日または日付で予約可能な時間が 1 つだけの場合、Lambda 関数はその予約可能な時間をレスポンスで提示し、レスポンス内の `dialogAction` を `ConfirmIntent` に設定することで、ユーザーに確認するように Amazon Lex に指示します。これは、予約可能な時間を積極的に提示することでユーザーエクスペリエンスを改善する方法を示しています。
  - 予約可能な時間が複数ある場合、Lambda 関数は予約可能な時間のリストを Amazon Lex へのレスポンスで返します。Amazon Lex は、Lambda 関数からのメッセージを付けたレスポンスをクライアントに返します。
- フルフィルメントコードフックとして、Lambda 関数は予約が行われた (つまり、インテントが達成された) ことを示す概要メッセージを返します。

> **Note**
>
> この例ではレスポンスカードの使用方法を示しています。Lambda 関数はレスポンスカードを構築して Amazon Lex に返します。レスポンスカードには、予約可能な日付と時刻がボタンとしてリストされていて、そのリストから選択できます。Amazon Lex コンソールで提供されているクライアントを使用してボットをテストする場合は、レスポンスカードを確認できません。レスポンスカードを確認するには、ボットを Facebook Messenger などのメッセージングプラットフォームと連携する必要があります。手順については、「Amazon Lex ボットと Facebook Messenger の統合」を参照してください。レスポンスカードの詳細情報については、「メッセージの管理 」を参照してください。

Amazon Lex は Lambda 関数を呼び出す際にイベントデータを入力として渡します。イベントフィールドの 1 つは `invocationSource` であり、Lambda 関数はこのイベントフィールドを使用して入力検証とフルフィルメントのいずれかのアクティビティを選択します。詳細については、「入力イベントの形式」を参照してください。

次のステップ

ステップ 1: Amazon Lex ボットを作成する

ステップ 1: Amazon Lex ボットを作成する

このセクションでは、Amazon Lex コンソールで提供されている ScheduleAppointment 設計図を使用して Amazon Lex ボットを作成します。

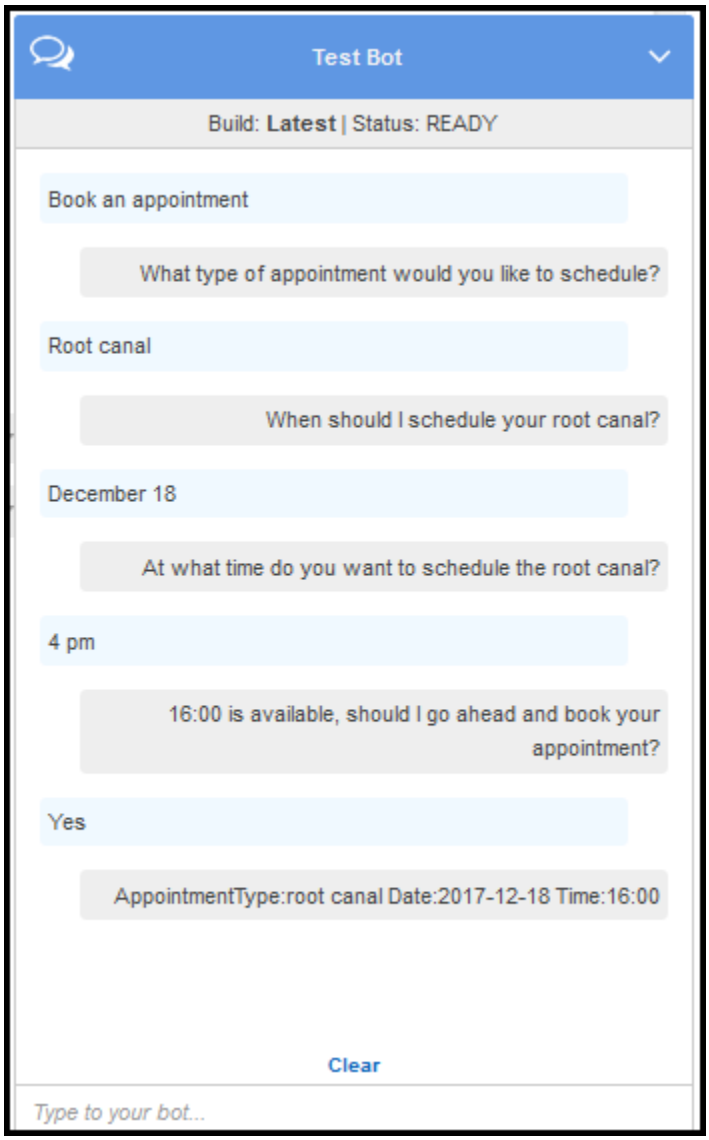
1. AWS Management Console にサインインし、Amazon Lex コンソール (https://console.aws.amazon.com/lex/) を開きます。
2. [Bots] ページで、[Create] を選択します。
3. [Create your Lex bot] ページで、次の操作を行います。
   - [ScheduleAppointment] 設計図を選択します。
   - ボット名 (ScheduleAppointment) はデフォルトのままにしておきます。
4. [Create] (作成) を選択します。

   このステップにより、ボットが保存および構築されます。構築プロセス中に、コンソールによって以下のリクエストが Amazon Lex に送信されます。

   - スロットタイプの新しいバージョンを ($LATEST バージョンから) 作成する。このボットの設計図で定義されているスロットタイプの詳細については、「ボット設計図 (ScheduleAppointment) の概要」を参照してください。
   - `MakeAppointment` インテントのバージョンを ($LATEST バージョンから) 作成する。場合によっては、新しいバージョンを作成する前に、`update` API オペレーションのリクエストがコンソールによって送信されます。
   - ボットの $LATEST バージョンを更新する。

     現時点では、Amazon Lex はボットの機械学習モデルを構築します。コンソールでボットをテストする場合、コンソールではランタイム API を使用してユーザー入力が Amazon Lex に返されます。Amazon Lex は機械学習モデルを使用してそのユーザー入力を解釈します。

5. コンソールで ScheduleAppointment ボットが表示されます。[Editor] タブで、事前設定されているインテント (MakeAppointment) の詳細を確認します。
6. テストウィンドウでボットをテストします。以下のスクリーンショットを使用して、ボットとのテスト会話を開始します。



次の点に注意してください。

- 最初のユーザー入力 (「予約します」) から、ボットはこのインテント (MakeAppointment) を推測します。
- ボットは、設定されているプロンプトを使用してユーザーからスロットデータを取得します。
- ボットの設計図では、MakeAppointment インテントに次の確認プロンプトが設定されています。

```
{Time} is available, should I go ahead and book your appointment?
```

ユーザーがすべてのスロットデータを提供すると、Amazon Lex は確認プロンプトをメッセージとして付けたレスポンスをクライアントに返します。クライアントはそのメッセージをユーザーに表示します。

```
16:00 is available, should I go ahead and book your appointment?
```

ユーザーデータを初期化または検証するためのコードがないため、ボットは任意の日付と時刻の予約を受け付けていることがわかります。次のセクションでは、その処理を行う Lambda 関数を追加します。

次のステップ

ステップ 2: Lambda 関数を作成する

ステップ 2: Lambda 関数を作成する

このセクションでは、Lambda コンソールで提供されている設計図 (lex-make-appointment-python) を使用して Lambda 関数を作成します。また、コンソールで提供されている Amazon Lex のサンプルイベントデータを使用して Lambda 関数を呼び出すことで、この関数をテストします。

1. AWS Management Console にサインインして AWS Lambda コンソール (https://console.aws.amazon.com/lambda/) を開きます。
2. [Create a Lambda function] (Lambda 関数の作成) を選択します。
3. [設計図の選択] に「**lex**」と入力して設計図を見つけます。次に、[lex-make-appointment-python] 設計図を選択します。
4. Lambda 関数を次のように設定します。
   - Lambda 関数の名前 (MakeAppointmentCodeHook) を入力します。
   - ロールとして [Create a new role from template(s)] を選択し、ロール名を入力します。
   - 他はデフォルト値のままにしておきます。
5. [Create Function] (関数の作成) を選択します。
6. 英語 (US) (en-US) 以外のロケールを使用している場合は、特定のロケールの設計図の更新 の説明に従ってインテント名を更新します。

7. Lambda 関数をテストします。

   a. [Actions]、[Configure test event] の順に選択します。

   b. [Sample event template] リストで、[Lex-Make Appointment (preview)] を選択します。このサンプルイベントでは、Amazon Lex のリクエスト/レスポンスモデルが使用されていて、この Amazon Lex ボットからのリクエストと一致するように値が設定されています。Amazon Lex のリクエスト/レスポンスモデルについては、「Lambda 関数を使用する」を参照してください。

   c. [Save and test] を選択します。

   d. Lambda 関数が正常に実行されたことを確認します。この例のレスポンスは、Amazon Lex レスポンスモデルと一致します。

次のステップ

ステップ 3: インテントの更新: コードフックを設定する

ステップ 3: インテントの更新: コードフックを設定する

このセクションでは、検証とフルフィルメントアクティビティのためのコードフックとして Lambda 関数を使用するように、`MakeAppointment` インテントの設定を更新します。

1. Amazon Lex コンソールで、[ScheduleAppointment] ボットを選択します。コンソールに [MakeAppointment] インテントが表示されます。インテント設定を次のように変更します。

   > **Note**
   >
   > 更新できるのは Amazon Lex リソース (インテントを含む) の $LATEST バージョンのみです。インテントのバージョンが $LATEST に設定されていることを確認します。このボットのバージョンはまだ発行していないため、コンソール内のボットは $LATEST バージョンのままです。

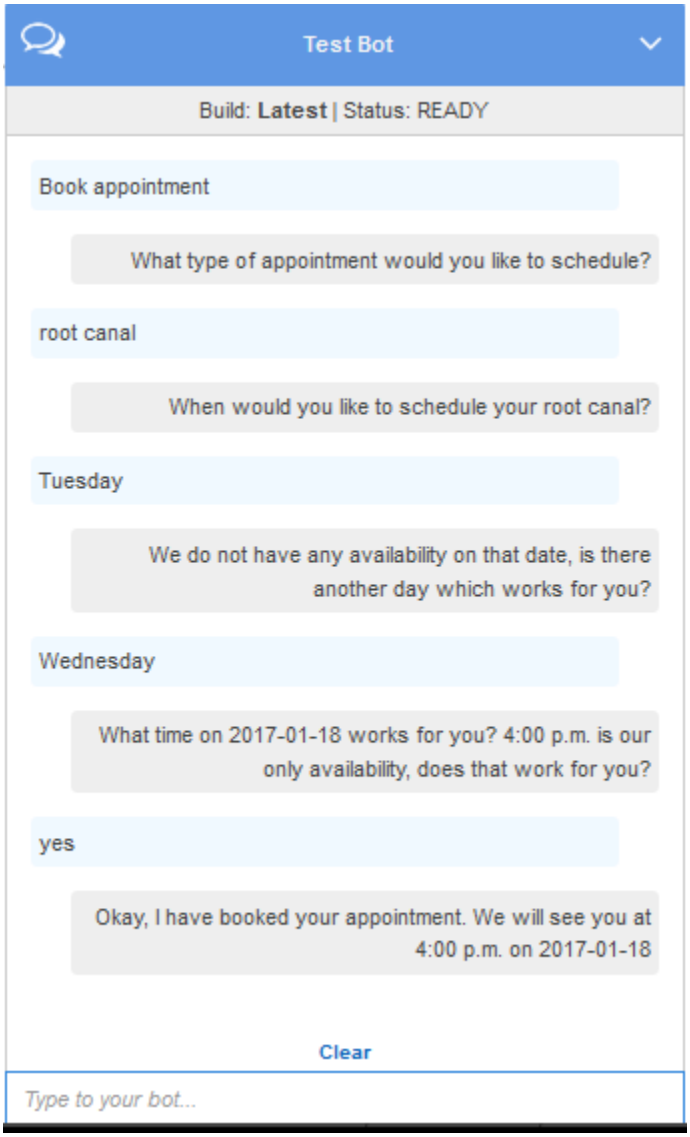
   a. [オプション] セクションでは、[初期化と検証コードのフック] を選択し、リストから Lambda 関数を選びます。

   b. [フルフィルメント] セクションで、[AWS Lambda 関数] を選択し、次にリストから Lambda 関数を選択します。

c. [Goodbye message] を選択し、メッセージを入力します。

2. [Save] を選択し、次に [Build] を選択します。

3. 以下の画像のように、ボットをテストします。



次のステップ

[ステップ 4: ボットを Facebook Messenger プラットフォームにデプロイする]

ステップ 4: ボットを Facebook Messenger プラットフォームにデプロイする

前のセクションでは、Amazon Lex コンソール内のクライアントを使用して ScheduleAppointment ボットをテストしました。現時点では、Amazon Lex コンソールはレスポンスカードをサポートしていません。ボットでサポートされている動的に生成されたレスポンスカードをテストするには、ボットを Facebook Messenger プラットフォームにデプロイしてテストします。

手順については、「Amazon Lex ボットと Facebook Messenger の統合」を参照してください。

次のステップ

情報フローの詳細

情報フローの詳細

`ScheduleAppointment` ボットの設計図では、動的に生成されたレスポンスカードの使用を主に示しています。この演習の Lambda 関数は、Amazon Lex へのレスポンスにレスポンスカードを含めています。Amazon Lex はそのレスポンスカードをクライアントへの応答に含めています。このセクションでは以下の両方について説明します。

- クライアントと Amazon Lex の間のデータフロー。

  このセクションでは、クライアントが `PostText` ランタイム API を使用して Amazon Lex にリクエストを送信することを前提としていて、それに応じたリクエスト/レスポンスの詳細を示しています。`PostText` ランタイム API の詳細については、「PostText」を参照してください。

  > **Note**
  >
  > クライアントが `PostContent` API を使用する場合の、クライアントと Amazon Lex の間の情報フローの例については、「ステップ 2a (オプション): 音声による情報フローの詳細を確認する (コンソール)」を参照してください。

- Amazon Lex と Lambda 関数の間のデータフロー。詳細については、「Lambda 関数の入力イベントとレスポンスの形式」を参照してください。

> **Note**
>
> この例では、Facebook Messenger クライアントを使用していることを前提としています。このクライアントは Amazon Lex へのリクエストでセッション属性を渡しません。したがって、このセクションで示しているリクエストの例では `sessionAttributes` は空です。Amazon Lex コンソールで提供されているクライアントを使用してボットをテストする場合、そのクライアントはセッション属性を含めます。

このセクションでは、各ユーザー入力の後に何が起こるかを説明します。

1. ユーザー: タイプ **`Book an appointment`**。

   a. クライアント (コンソール) は以下の PostContent リクエストを Amazon Lex に送信します。

   ```
   POST /bot/ScheduleAppointment/alias/$LATEST/
   user/bijt6rovckwecnzesbthrr1d7lv3ja3n/text
   "Content-Type":"application/json"
   "Content-Encoding":"amz-1.0"

   {
      "inputText":"book appointment",
      "sessionAttributes":{}
   }
   ```

   リクエストの URI と本文の両方で Amazon Lex に情報が提供されています。

   - リクエスト URI – ボット名 (ScheduleAppointment)、ボットのエイリアス ($LATEST)、およびユーザー名 ID。末尾の `text` は、このリクエストが `PostText` API リクエストである (`PostContent` API リクエストではない) ことを示しています。
   - リクエストボディ – ユーザー入力 (`inputText`) と空の `sessionAttributes` が含まれています。

   b. `inputText` から、Amazon Lex はインテント (`MakeAppointment`) を検出します。サービスによって、コードフックとして設定されている Lambda 関数が呼び出され、以下のイベントが渡されることで初期化および検証が実行されます。詳細については、「入力イベントの形式」を参照してください。

   ```
   {
   ```

```
    "currentIntent": {
        "slots": {
            "AppointmentType": null,
            "Date": null,
            "Time": null
        },
        "name": "MakeAppointment",
        "confirmationStatus": "None"
    },
    "bot": {
        "alias": null,
        "version": "$LATEST",
        "name": "ScheduleAppointment"
    },
    "userId": "bijt6rovckwecnzesbthrr1d7lv3ja3n",
    "invocationSource": "DialogCodeHook",
    "outputDialogMode": "Text",
    "messageVersion": "1.0",
    "sessionAttributes": {}
}
```

クライアントによって送信された情報に加えて、Amazon Lex には、以下の追加データが含まれます。

- `currentIntent` – 現在のインテントの情報を提供しています。
- `invocationSource` – Lambda 関数呼び出しの目的を表しています。この場合、目的はユーザーデータの初期化および検証を実行することです。(Amazon Lex はインテントを達成するためのスロットデータの一部をユーザーがまだ指定していないことを知っています)。
- `messageVersion` – 現在 Amazon Lex でサポートしているのは 1.0 バージョンだけです。

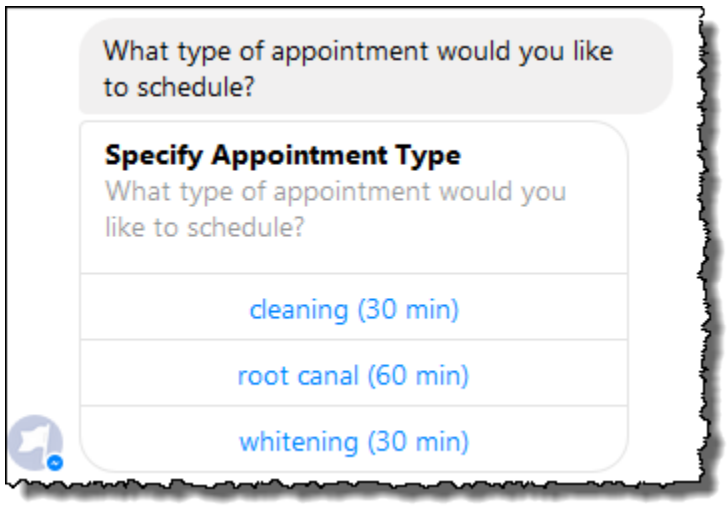
c. この時点では、すべてのスロット値は null です (検証する対象はありません)。Lambda 関数は、以下のレスポンスを Amazon Lex に返して、`AppointmentType` スロットの情報を引き出すようにサービスに指示します。レスポンスの形式については、「レスポンスの形式」を参照してください。

```
{
    "dialogAction": {
        "slotToElicit": "AppointmentType",
        "intentName": "MakeAppointment",
```

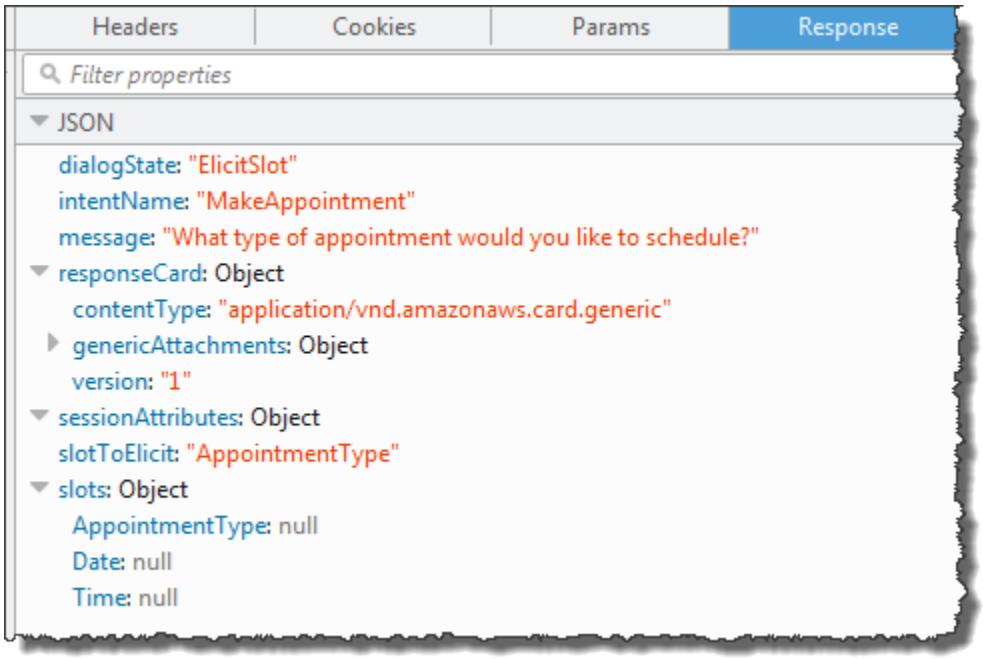
```
        "responseCard": {
            "genericAttachments": [
                {
                    "buttons": [
                        {
                            "text": "cleaning (30 min)",
                            "value": "cleaning"
                        },
                        {
                            "text": "root canal (60 min)",
                            "value": "root canal"
                        },
                        {
                            "text": "whitening (30 min)",
                            "value": "whitening"
                        }
                    ],
                    "subTitle": "What type of appointment would you like to
 schedule?",
                    "title": "Specify Appointment Type"
                }
            ],
            "version": 1,
            "contentType": "application/vnd.amazonaws.card.generic"
        },
        "slots": {
            "AppointmentType": null,
            "Date": null,
            "Time": null
        },
        "type": "ElicitSlot",
        "message": {
            "content": "What type of appointment would you like to schedule?",
            "contentType": "PlainText"
        }
    },
    "sessionAttributes": {}
}
```

このレスポンスには、dialogAction フィールドと sessionAttributes フィールドが含まれています。特に、dialogAction フィールドでは以下のフィールドが返されています。

- `type` – このフィールドを `ElicitSlot` に設定することで、Lambda 関数は、`slotToElicit` フィールドで指定しているスロットの値を引き出すように Amazon Lex に指示しています。Lambda 関数は、ユーザーに伝えるメッセージである `message` も提供しています。
- `responseCard` – `AppointmentType` スロットで使用可能な値のリストを特定しています。レスポンスカードをサポートしているクライアント (例えば、Facebook Messenger) では、ユーザーが予約タイプを選択できるレスポンスカードが次の画像のように表示されます。



d. Lambda 関数からのレスポンス内の `dialogAction.type` で示されているように、Amazon Lex は以下のレスポンスをクライアントに返します。



クライアントはレスポンスを読み取り、「どのタイプを予約なさいますか?」のメッセージと レスポンスカード (クライアントでサポートされている場合) を表示します。

2. ユーザー: ユーザーにはクライアントに応じて 2 つの選択肢があります。

   - レスポンスカードが表示されている場合は、[root canal (60 分)] あるいはタイプ **`root canal`** を選択します。
   - クライアントでレスポンスカードがサポートされていない場合は、「**`root canal`**」と入力します。

   a. クライアントは以下の PostText リクエスト (読みやすいように改行が追加されています) を Amazon Lex に送信します。

```
POST /bot/BookTrip/alias/$LATEST/user/bijt6rovckwecnzesbthrr1d7lv3ja3n/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
   "inputText": "root canal",
    "sessionAttributes": {}
}
```

   b. Amazon Lex は以下のイベントをパラメータとして送信することで、ユーザーデータの検証のために Lambda 関数を呼び出します。

```
{
    "currentIntent": {
        "slots": {
            "AppointmentType": "root canal",
            "Date": null,
            "Time": null
        },
        "name": "MakeAppointment",
        "confirmationStatus": "None"
    },
    "bot": {
        "alias": null,
        "version": "$LATEST",
        "name": "ScheduleAppointment"
    },
    "userId": "bijt6rovckwecnzesbthrr1d7lv3ja3n",
```

```
    "invocationSource": "DialogCodeHook",
    "outputDialogMode": "Text",
    "messageVersion": "1.0",
    "sessionAttributes": {}
}
```

イベントデータで次の点に注意してください。

- invocationSource は DialogCodeHook のままです。このステップではユーザーデータを検証しているだけです。
- Amazon Lex は AppointmentType スロットの currentIntent.slots フィールドを root canal に設定します。
- Amazon Lex はクライアントと Lambda 関数の間で sessionAttributes フィールドを渡すだけです。

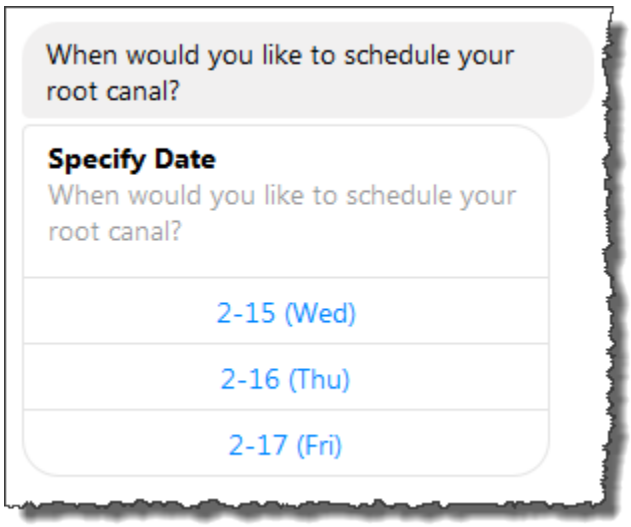
c. Lambda 関数はユーザー入力を検証し、予約日付の値を引き出すようにサービスに指示する、以下のレスポンスを Amazon Lex に返します。

```
{
    "dialogAction": {
        "slotToElicit": "Date",
        "intentName": "MakeAppointment",
        "responseCard": {
            "genericAttachments": [
                {
                    "buttons": [
                        {
                            "text": "2-15 (Wed)",
                            "value": "Wednesday, February 15, 2017"
                        },
                        {
                            "text": "2-16 (Thu)",
                            "value": "Thursday, February 16, 2017"
                        },
                        {
                            "text": "2-17 (Fri)",
                            "value": "Friday, February 17, 2017"
                        },
                        {
                            "text": "2-20 (Mon)",
                            "value": "Monday, February 20, 2017"
                        },
```

```
                        {
                            "text": "2-21 (Tue)",
                            "value": "Tuesday, February 21, 2017"
                        }
                    ],
                    "subTitle": "When would you like to schedule your root
 canal?",
                    "title": "Specify Date"
                }
            ],
            "version": 1,
            "contentType": "application/vnd.amazonaws.card.generic"
        },
        "slots": {
            "AppointmentType": "root canal",
            "Date": null,
            "Time": null
        },
        "type": "ElicitSlot",
        "message": {
            "content": "When would you like to schedule your root canal?",
            "contentType": "PlainText"
        }
    },
    "sessionAttributes": {}
}
```

ここでも、レスポンスに dialogAction フィールドと sessionAttributes フィールドが含まれています。特に、dialogAction フィールドでは以下のフィールドが返されています。

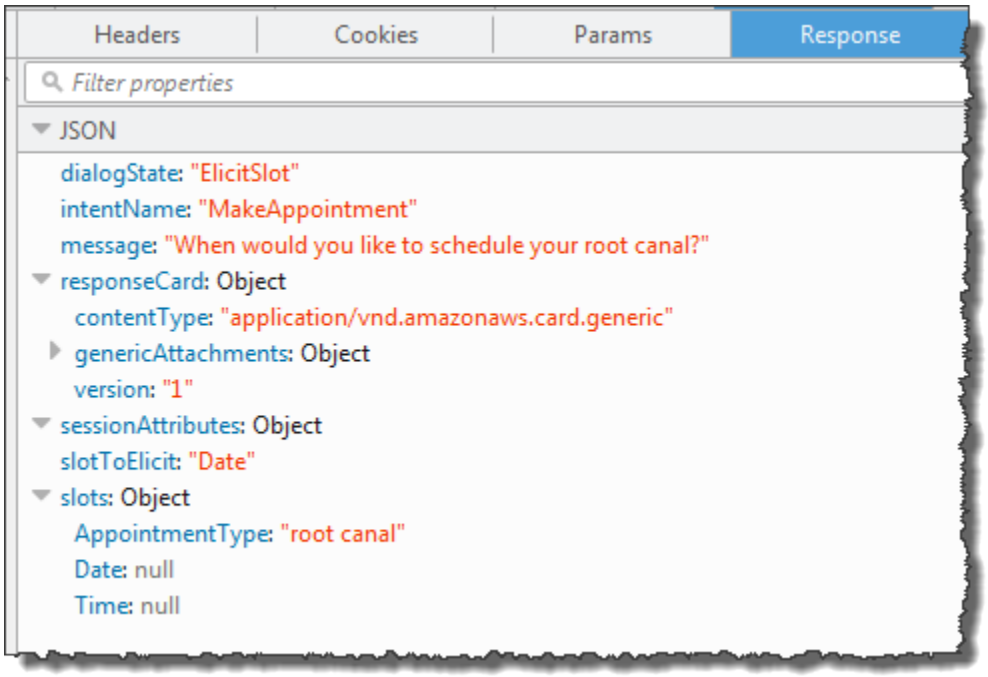
- type – このフィールドを ElicitSlot に設定することで、Lambda 関数は、slotToElicit フィールドで指定しているスロットの値を引き出すように Amazon Lex に指示しています。Lambda 関数は、ユーザーに伝えるメッセージである message も提供しています。
- responseCard – Date スロットで使用可能な値のリストを特定しています。レスポンスカードをサポートしているクライアント (例えば、Facebook Messenger) では、次の画像のように、ユーザーが予約日付を選択できるレスポンスカードが表示されます。



Lambda 関数は 5 つの日付を返していますが、このクライアント (Facebook Messenger) ではレスポンスカードのボタンが 3 つまでに制限されています。そのため、スクリーンショットでは最初の 3 つの値だけが表示されています。

これらの日付は Lambda 関数でハードコードされています。本稼働アプリケーションでは、カレンダーを使用して、予約可能な日付をリアルタイムで取得できます。日付は動的であるため、Lambda 関数でレスポンスカードを動的に生成する必要があります。

d. Amazon Lex は `dialogAction.type` に気づき、Lambda 関数のレスポンスからの情報を含む次のレスポンスをクライアントに返します。



クライアントには、「When would you like to schedule your root canal?」(root canal をいつ予約しますか ?) というメッセージ とレスポンスカード (クライアントでレスポンスカードがサポートされている場合) が表示されます。

3. ユーザー: タイプ **Thursday**。

   a. クライアントは以下の PostText リクエスト (読みやすいように改行が追加されています) を Amazon Lex に送信します。

```
POST /bot/BookTrip/alias/$LATEST/user/bijt6rovckwecnzesbthrr1d7lv3ja3n/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
   "inputText": "Thursday",
    "sessionAttributes": {}
}
```

   b. Amazon Lex は以下のイベントでパラメータとして送信して、ユーザーデータの検証のために Lambda 関数を呼び出します。

```
{
    "currentIntent": {
        "slots": {
            "AppointmentType": "root canal",
            "Date": "2017-02-16",
            "Time": null
        },
        "name": "MakeAppointment",
        "confirmationStatus": "None"
    },
    "bot": {
        "alias": null,
        "version": "$LATEST",
        "name": "ScheduleAppointment"
    },
    "userId": "u3fpr9gghj02zts7y5tpq5mm4din2xqy",
    "invocationSource": "DialogCodeHook",
    "outputDialogMode": "Text",
    "messageVersion": "1.0",
    "sessionAttributes": {}
}
```

イベントデータで次の点に注意してください。

- invocationSource は DialogCodeHook のままです。このステップではユーザーデータを検証しているだけです。
- Amazon Lex は Date スロットの currentIntent.slots フィールドを 2017-02-16 に設定します。
- Amazon Lex はクライアントと Lambda 関数の間で sessionAttributes を渡すだけです。

c. Lambda 関数はユーザー入力を検証します。今回は、Lambda 関数は指定された日付に予約可能な時間がないと判断します。この関数は、予約日付の値を再度引き出すようにサービスに指示する、以下のレスポンスを Amazon Lex に返します。

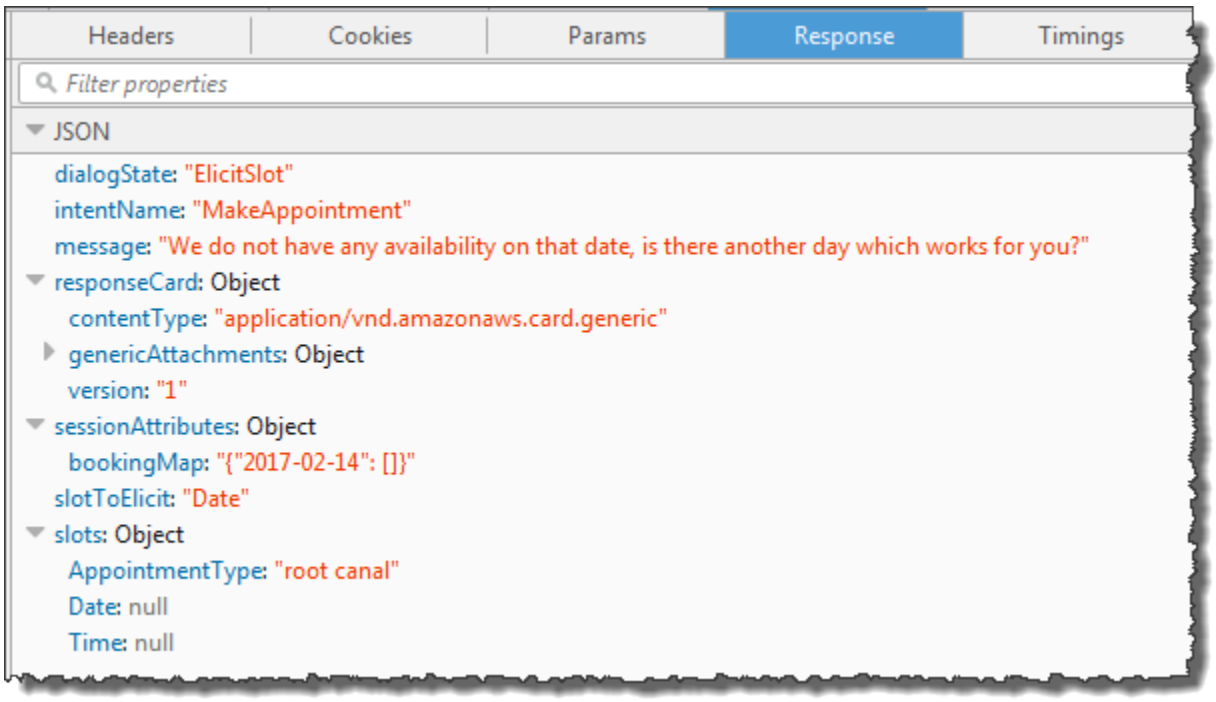
```
{
    "dialogAction": {
        "slotToElicit": "Date",
        "intentName": "MakeAppointment",
        "responseCard": {
            "genericAttachments": [
                {
                    "buttons": [
                        {
                            "text": "2-15 (Wed)",
                            "value": "Wednesday, February 15, 2017"
                        },
                        {
                            "text": "2-17 (Fri)",
                            "value": "Friday, February 17, 2017"
                        },
                        {
                            "text": "2-20 (Mon)",
                            "value": "Monday, February 20, 2017"
                        },
                        {
                            "text": "2-21 (Tue)",
                            "value": "Tuesday, February 21, 2017"
                        }
                    ],
                    "subTitle": "When would you like to schedule your root
 canal?",
                    "title": "Specify Date"
```

```
                }
            ],
            "version": 1,
            "contentType": "application/vnd.amazonaws.card.generic"
        },
        "slots": {
            "AppointmentType": "root canal",
            "Date": null,
            "Time": null
        },
        "type": "ElicitSlot",
        "message": {
            "content": "We do not have any availability on that date, is there
 another day which works for you?",
            "contentType": "PlainText"
        }
    },
    "sessionAttributes": {
     "bookingMap": "{\"2017-02-16\": []}"
   }
}
```

ここでも、レスポンスに dialogAction フィールドと sessionAttributes フィールドが含まれています。特に、dialogAction では以下のフィールドが返されています。

- dialogAction field:
  - type – Lambda 関数はこの値を ElicitSlot に設定し、slotToElicit フィールドを Date にリセットします。Lambda 関数は、ユーザーに伝える適切なメッセージである message も提供しています。
  - responseCard – Date スロットの値のリストを返します。
- sessionAttributes – 今回は、Lambda 関数は bookingMap セッション属性を含めています。その値は、要求された予約日付と予約可能な時間です (オブジェクトが空の場合は、予約可能な時間がないことを表します)。

d. Amazon Lex は dialogAction.type に気づき、Lambda 関数のレスポンスからの情報を含む次のレスポンスをクライアントに返します。



クライアントは「We do not have any availability on that date, is there another day which works for you?」(その日付では空きがありません。ご都合がよい別の日はありますか ?)のメッセージとレスポンスカード (クライアントでレスポンスカードがサポートされている場合) が表示されます。

4. ユーザー: ユーザーにはクライアントに応じて 2 つの選択肢があります。

   - レスポンスカードが表示されている場合は、[2-15 (Wed)] を選択するか、「**Wednesday**」と入力します。
   - クライアントでレスポンスカードがサポートされていない場合は、「**Wednesday**」と入力します。

   a. クライアント は以下の PostText リクエストを Amazon Lex に送信します。

```
POST /bot/BookTrip/alias/$LATEST/user/bijt6rovckwecnzesbthrr1d7lv3ja3n/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
   "inputText": "Wednesday",
    "sessionAttributes": {
   }
```

```
}
```

> **Note**
>
> Facebook Messenger クライアントではセッション属性は設定されません。リクエスト間のセッションステータスを保持する場合は、Lambda 関数内でこれを実行する必要があります。実際のアプリケーションでは、これらのセッション属性をバックエンドデータベースで保持する必要があります。

b. Amazon Lex は以下のイベントをパラメータとして送信することで、ユーザーデータの検証のために Lambda 関数を呼び出します。

```
{
    "currentIntent": {
        "slots": {
            "AppointmentType": "root canal",
            "Date": "2017-02-15",
            "Time": null
        },
        "name": "MakeAppointment",
        "confirmationStatus": "None"
    },
    "bot": {
        "alias": null,
        "version": "$LATEST",
        "name": "ScheduleAppointment"
    },
    "userId": "u3fpr9gghj02zts7y5tpq5mm4din2xqy",
    "invocationSource": "DialogCodeHook",
    "outputDialogMode": "Text",
    "messageVersion": "1.0",
    "sessionAttributes": {
    }
}
```

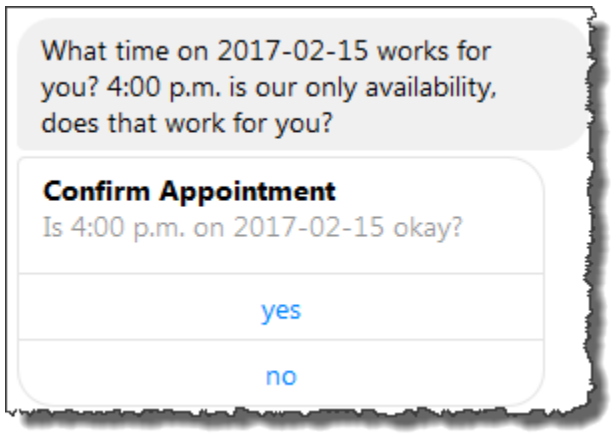
Amazon Lex は `currentIntent.slots` スロットを `Date` に設定することで `2017-02-15` を更新します。

c. Lambda 関数はユーザー入力を検証し、予約時刻の値を引き出すように指示する、以下のレスポンスを Amazon Lex に返します。

```
{
    "dialogAction": {
        "slots": {
            "AppointmentType": "root canal",
            "Date": "2017-02-15",
            "Time": "16:00"
        },
        "message": {
            "content": "What time on 2017-02-15 works for you? 4:00 p.m. is our
 only availability, does that work for you?",
            "contentType": "PlainText"
        },
        "type": "ConfirmIntent",
        "intentName": "MakeAppointment",
        "responseCard": {
            "genericAttachments": [
                {
                    "buttons": [
                        {
                            "text": "yes",
                            "value": "yes"
                        },
                        {
                            "text": "no",
                            "value": "no"
                        }
                    ],
                    "subTitle": "Is 4:00 p.m. on 2017-02-15 okay?",
                    "title": "Confirm Appointment"
                }
            ],
            "version": 1,
            "contentType": "application/vnd.amazonaws.card.generic"
        }
    },
    "sessionAttributes": {
        "bookingMap": "{\"2017-02-15\": [\"10:00\", \"16:00\", \"16:30\"]}"
    }
}
```

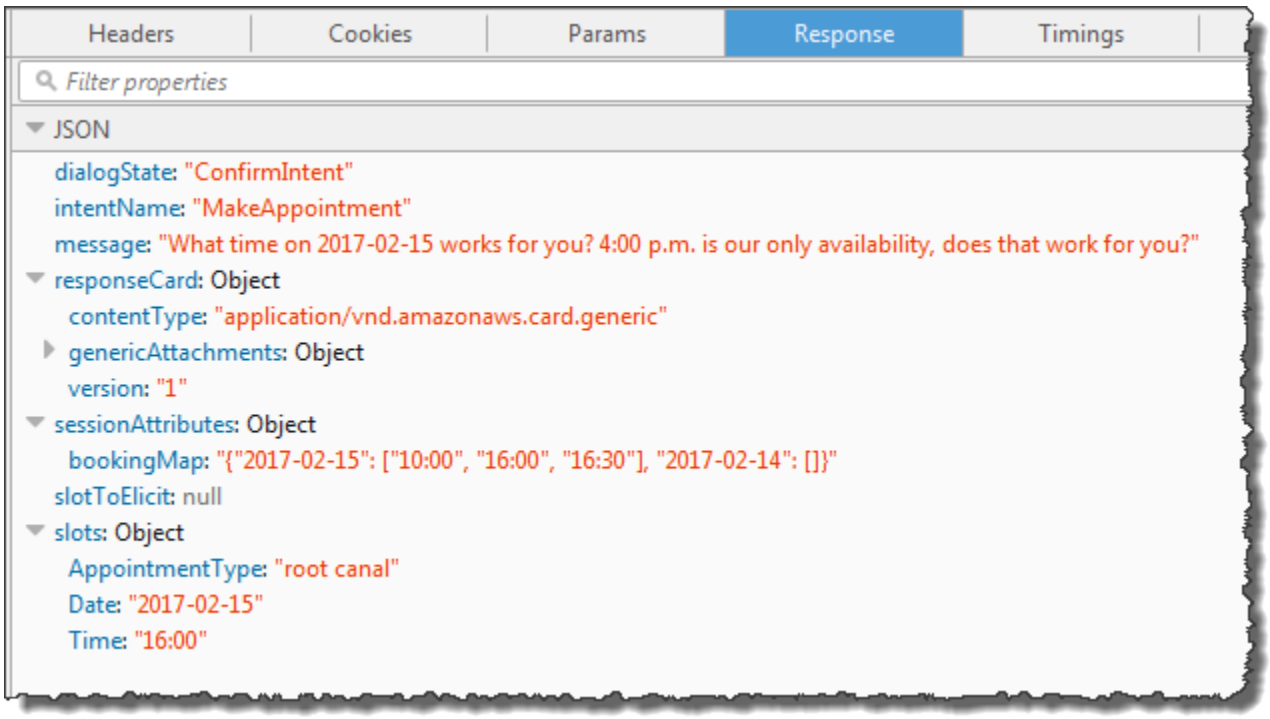
ここでも、レスポンスに dialogAction フィールドと sessionAttributes フィールドが含まれています。特に、dialogAction では以下のフィールドが返されています。

- dialogAction field:
  - type – Lambda 関数はこの値を ConfirmIntent に設定して、message で提示している予約時刻をユーザーに確認するように Amazon Lex に指示します。
  - responseCard – ユーザーが選択できる「はい/いいえ」の値のリストを返します。クライアントでレスポンスカードがサポートされている場合、クライアントはそのレスポンスカードを以下の例のように表示します。



  - sessionAttributes - Lambda 関数は bookingMap セッション属性を設定し、その値を予約日付とその日の予約可能な時間に設定します。この例では、予約時間は 30 分間です。1 時間かかる root canal の場合、予約できるのは 午後 4 時だけです。

d. Lambda 関数のレスポンス内の dialogAction.type で示されているように、Amazon Lex は以下のレスポンスをクライアントに返します。



クライアントは次のメッセージを表示します。2017 年 2 月 15 日は何時が都合がよろしいでしょうか？午後 4 時しか空いていませんがよろしいですか？

5. ユーザー : [**yes**] を選択し ます。

Amazon Lex は以下のイベントデータを使用して Lambda 関数を呼び出します。ユーザーが「**yes**」と応答しているため、`confirmationStatus` は `Confirmed` を `Time` に設定し、Amazon Lex の `currentIntent.slots` フィールドを `4 p.m` に設定します。

```
{
    "currentIntent": {
        "slots": {
            "AppointmentType": "root canal",
            "Date": "2017-02-15",
            "Time": "16:00"
        },
        "name": "MakeAppointment",
        "confirmationStatus": "Confirmed"
    },
    "bot": {
        "alias": null,
        "version": "$LATEST",
```

```
        "name": "ScheduleAppointment"
    },
    "userId": "u3fpr9gghj02zts7y5tpq5mm4din2xqy",
    "invocationSource": "FulfillmentCodeHook",
    "outputDialogMode": "Text",
    "messageVersion": "1.0",
    "sessionAttributes": {
   }
}
```

confirmationStatus が確認済みであるため、Lambda 関数はインテント (歯科予約) を処理し、以下のレスポンスを Amazon Lex に返します。

```
{
    "dialogAction": {
        "message": {
            "content": "Okay, I have booked your appointment.  We will see you at
 4:00 p.m. on 2017-02-15",
            "contentType": "PlainText"
        },
        "type": "Close",
        "fulfillmentState": "Fulfilled"
    },
    "sessionAttributes": {
        "formattedTime": "4:00 p.m.",
        "bookingMap": "{\"2017-02-15\": [\"10:00\"]}"
    }
}
```

次の点に注意してください。

- Lambda 関数が sessionAttributes を更新しています。
- dialogAction.type が Close に設定されていて、ユーザーの応答を想定しないことを Amazon Lex に指示しています。
- dialogAction.fulfillmentState が Fulfilled に設定されていて、インテントの達成が完了したことを示しています。

クライアントに次のメッセージを表示します。あなたの予約が完了しました。2017 年 2 月 15 日の午後 4 時にお待ちしています。

旅行を予約する

この例では、複数のインテントをサポートするように設定されているボットの作成を示しています。この例では、クロスインテント情報共有のためのセッション属性の使用方法も示しています。ボットを作成した後で、Amazon Lex コンソールでテストクライアントを使用してボット (BookTrip) をテストします。クライアントでは、PostText ランタイム API オペレーションを使用して、各ユーザー入力に対するリクエストが Amazon Lex に送信されます。

この例の BookTrip ボットは、2 つのインテント (BookHotel と BookCar) を使用して設定されています。例えば、ユーザーが最初にホテルを予約するとします。その操作中に、ユーザーはチェックイン日時、場所、宿泊数などの情報を指定します。インテントが達成されると、クライアントではセッション属性を使用してこの情報を保持できます。セッション属性の詳細については、「PostText」を参照してください。

次に、そのユーザーが引き続き車を予約するとします。前の BookHotel インテントでユーザーが提供した情報 (つまり、目的地、およびチェックイン/チェックアウトの日時) を使用して、BookCar インテントを初期化および検証するように設定されたコードフック (Lambda 関数) によって、BookCar インテント用のスロットデータ (つまり、目的地、受け取り場所、受け取り日付、および返却日時) が初期化されます。これは、クロスインテント情報共有によって、ユーザーと動的に会話できるボットを構築する方法を示しています。

この例では以下のセッション属性を使用しています。セッション属性を設定および更新できるのはクライアントと Lambda 関数だけであり、Amazon Lex はクライアントと Lambda 関数の間でセッション属性を渡すだけです。Amazon Lex では、セッション属性が保持および変更されることはありません。

- `currentReservation` – 進行中の予約とその他の関連情報のスロットデータが含まれています。クライアントから Amazon Lex へのリクエストの例を次に示します。この例では、リクエストボディに `currentReservation` セッション属性が含まれています。

```
POST /bot/BookTrip/alias/$LATEST/user/wch89kjqcpkds8seny7dly5x3otq68j3/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
   "inputText":"Chicago",
   "sessionAttributes":{
```

```
        "currentReservation":"{\"ReservationType\":\"Hotel\",
                               \"Location\":\"Moscow\",
                               \"RoomType\":null,
                               \"CheckInDate\":null,
                               \"Nights\":null}"
    }
}
```

- `lastConfirmedReservation` – 前のインテントでの類似した情報が含まれています (該当する場合)。例えば、ユーザーがホテルを予約した後に車を予約中である場合、このセッション属性には前の BookHotel インテントのスロットデータが格納されています。

- `confirmationContext` – Lambda 関数では、前の予約のスロットデータ (存在する場合) に基づいて一部のスロットデータを事前入力する際に、これが `AutoPopulate` に設定されます。これにより、クロスインテント情報共有が可能になります。例えば、ユーザーがホテルを予約した後に車を予約しようとしている場合、Amazon Lex では、ホテルの予約と同じ場所と日時で車を予約することを確認 (または拒否) するようにユーザーに指示できます。

この演習では、設計図を使用して Amazon Lex ボットと Lambda 関数を作成します。設計図の詳細については、「Amazon Lex および AWS Lambda の設計図」を参照してください。

次のステップ

ステップ 1: この演習で使用する設計図を確認する

ステップ 1: この演習で使用する設計図を確認する

トピック

- ボットの設計図の概要 (BookTrip)
- Lambda 関数の設計図 (lex-book-trip-python) の概要

ボットの設計図の概要 (BookTrip)

ボットの作成に使用する設計図 (BookTrip) では、以下が事前設定されています。

- スロットタイプ – 次の 2 つのカスタムスロットタイプ。
  - BookHotel インテントで使用される、列挙値 king、queen、および deluxe を持つ RoomTypes。
  - BookCar インテントで使用される、列挙値 economy、standard、midsize、full size、luxury、および minivan を持つ CarTypes。

- インテント 1 (BookHotel) – 次のように事前設定されています。
  - 事前設定スロット
    - RoomType: RoomTypes カスタムスロットタイプ
    - Location: AMAZON.US_CITY 組み込みスロットタイプ
    - CheckInDate: AMAZON.DATE 組み込みスロットタイプ
    - Nights: AMAZON.NUMBER 組み込みスロットタイプ
  - 事前設定発話
    - 「ホテルの予約」
    - 「ホテルを予約します」
    - 「{Location}で{Nights}泊の予約」

    ユーザーがこのいずれかを発声すると、Amazon Lex は BookHotel がインテントであると判断して、ユーザーにスロットデータを求めます。
  - 事前設定プロンプト
    - Location スロットのプロンプト –「どの都市に滞在されますか?」
    - CheckInDate スロットのプロンプト –「何日にチェックインされますか?」
    - Nights スロットのプロンプト –「何泊されますか?」
    - RoomType スロットのプロンプト –「部屋のタイプはクイーン、キング、デラックスのどれになさいますか?」
    - 確認ステートメント –「かしこまりました。{Location} での {CheckInDate} から {Nights} 泊の滞在を承りました。この内容で予約いたしましょうか?」
    - 拒否 –「かしこまりました。予約をキャンセルいたしました。」

- インテント 2 (BookCar) – 次のように事前設定されています。
  - 事前設定スロット

- PickUpCity: AMAZON.US_CITY 組み込みタイプ
- PickUpDate: AMAZON.DATE 組み込みタイプ
- ReturnDate: AMAZON.DATE 組み込みタイプ
- DriverAge: AMAZON.NUMBER 組み込みタイプ
- CarType: CarTypes カスタムタイプ

- 事前設定発話
  - 「車の予約」
  - 「レンタカーの予約」
  - 「車を予約します」

  ユーザーがこのいずれかを発声すると、Amazon Lex は BookCar がインテントであると判断して、ユーザーにスロットデータを求めます。
- 事前設定プロンプト
  - PickUpCity スロットのプロンプト –「どの都市でレンタカーが必要でしょうか?」
  - PickUpDate スロットのプロンプト –「レンタカーは何日からご使用なさいますか?」
  - ReturnDate スロットのプロンプト –「レンタカーは何日にご返却なさいますか?」
  - DriverAge スロットのプロンプト –「このレンタカーを運転される方の年齢を教えていただけますか?」
  - CarType スロットのプロンプト –「どのタイプのレンタカーをご希望でしょうか? 」 当社では小型車、中型車、高級車の中からお選びいただけます」
  - 確認ステートメント –「かしこまりました。{PickUpDate} から {ReturnDate} までの期間で、{PickUpCity} での {CarType} タイプのレンタカーのご使用を承りました。」 この内容で予約いたしましょうか?」
  - 拒否 –「かしこまりました。予約をキャンセルいたしました。」

Lambda 関数の設計図 (lex-book-trip-python) の概要

AWS Lambda には、ボットの設計図に加えて、この設計図でコードフックとして使用できる設計図 (lex-book-trip-python) が用意されています。ボットの設計図とそれに対応する Lambda 関数の設計図の一覧については、「Amazon Lex および AWS Lambda の設計図」を参照してください。

BookTrip 設計図を使用してボットを作成する際に、ユーザーデータの入力とインテントの達成の両方の初期化/検証のためのコードフックとしてこの Lambda 関数を追加して、両方のインテント (BookCar と BookHotel) の設定を更新します。

この Lambda 関数のコードでは、ユーザーに関する既知の情報 (セッション属性で保持されている) を使用してインテントのスロット値を初期化する、動的会話の例を示しています。詳細については、「会話コンテキストの管理」を参照してください。

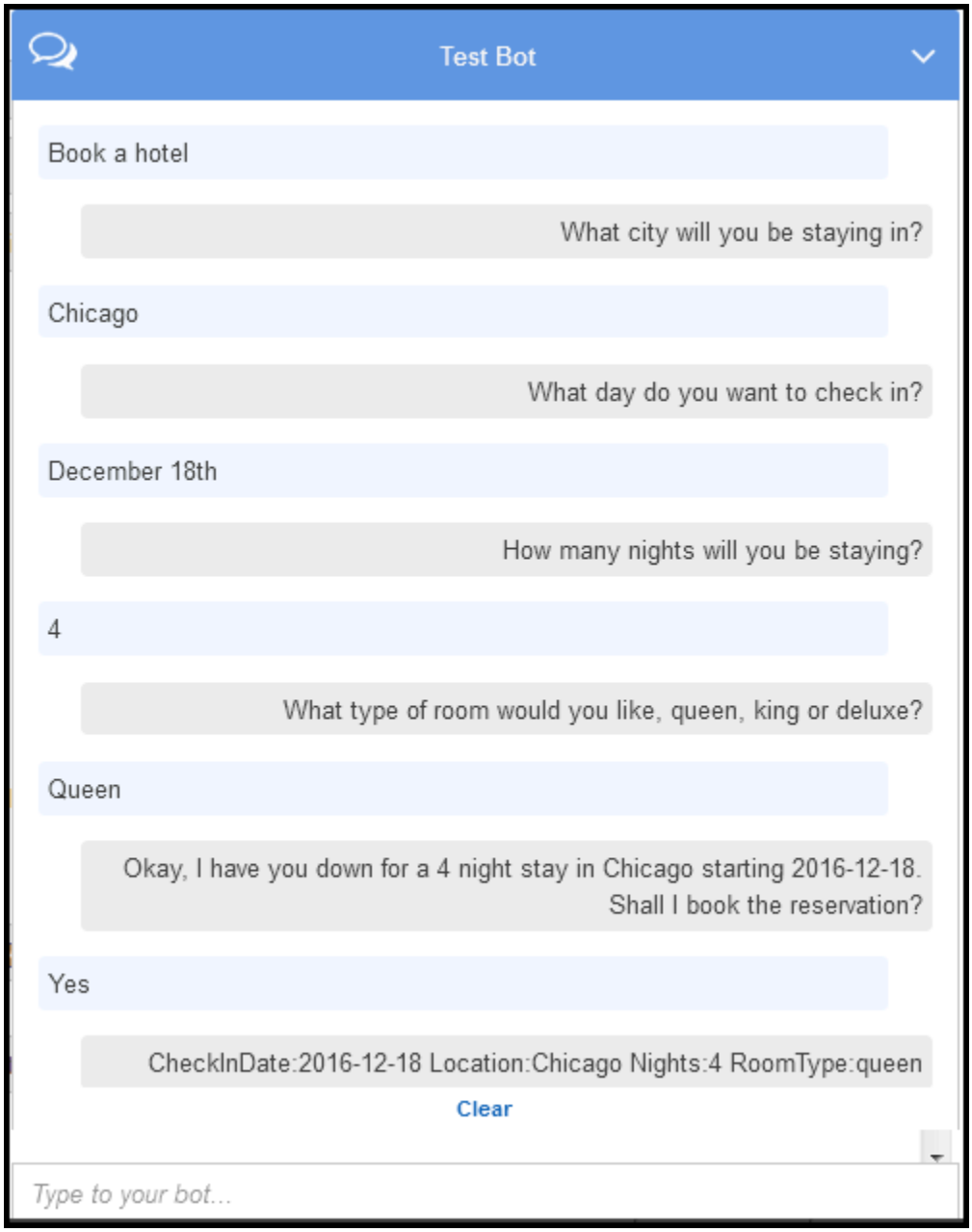
次のステップ

ステップ 2: Amazon Lex ボットを作成する

ステップ 2: Amazon Lex ボットを作成する

このセクションでは、Amazon Lex ボット (BookTrip) を作成します。

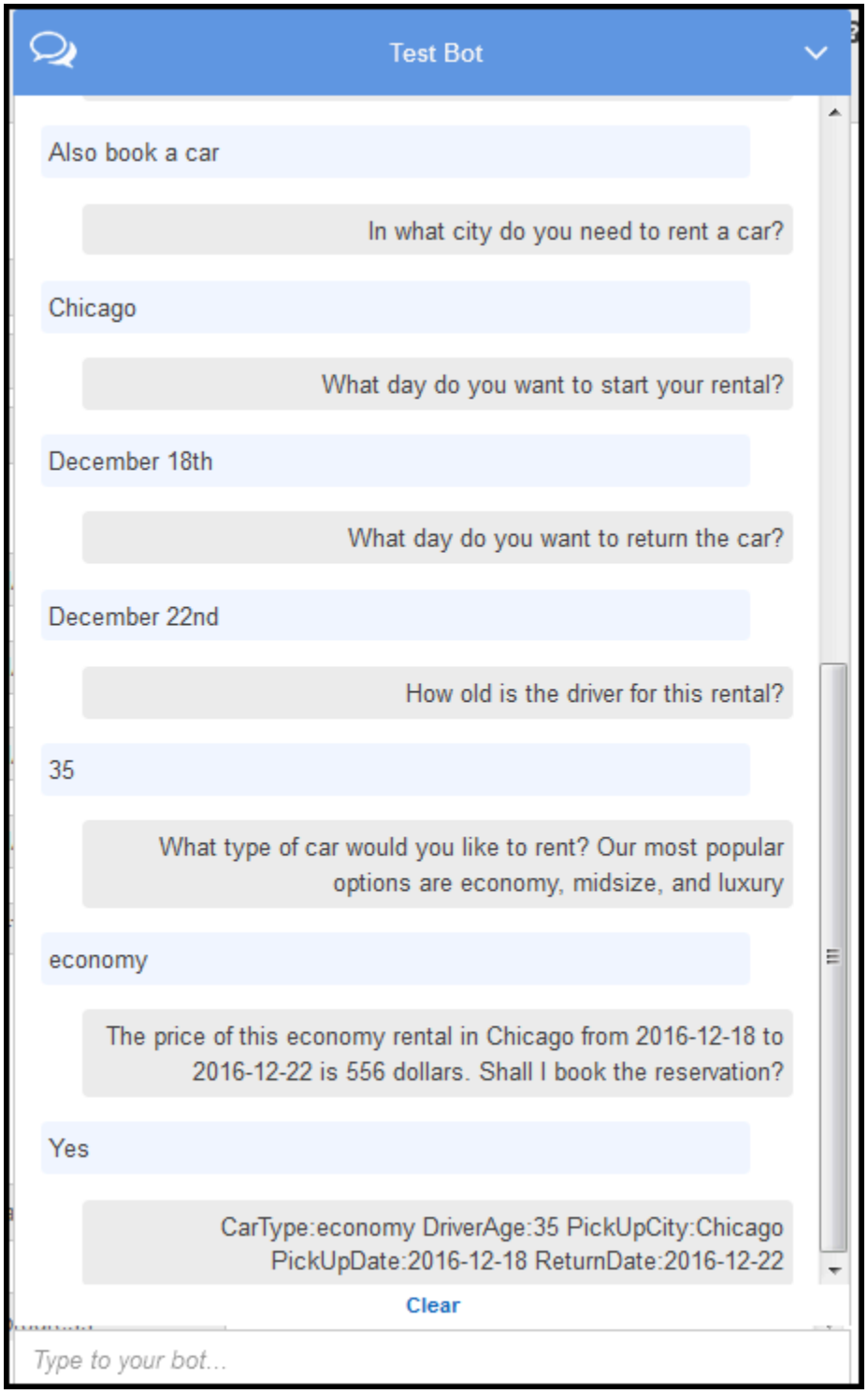
1. AWS Management Console にサインインし、Amazon Lex コンソール (https://console.aws.amazon.com/lex/) を開きます。
2. [Bots] ページで、[Create] を選択します。
3. [Create your Lex bot] ページで、以下の操作を行います。
   - [BookTrip] 設計図を選択します。
   - ボット名 (BookTrip) はデフォルトのままにしておきます。
4. [Create] (作成) を選択します。コンソールによって、ボットを作成するための一連のリクエストが Amazon Lex に送信されます。次の点に注意してください。
5. コンソールに BookTrip ボットが表示されます。[Editor] タブで、事前設定インテント (BookCar と BookHotel) の詳細を確認します。
6. テストウィンドウでボットをテストします。以下を使用して、ボットとのテスト会話を開始します。



最初のユーザー入力 (「ホテルの予約」) によって、Amazon Lex はインテント (BookHotel) を推測します。ボットは、このインテントに事前設定されているプロンプトを使用して、ユーザーからスロットデータを引き出します。ユーザーがすべてのスロットデータを提供すると、Amazon Lex は、すべてのユーザー入力をメッセージとして含むレスポンスをクライアントに返します。クライアントはレスポンス内のメッセージを次のように表示します。

```
CheckInDate:2016-12-18 Location:Chicago Nights:5 RoomType:queen
```

次に、会話を続行し、次の会話で車の予約を試みます。



次の点に注意してください。

- この時点ではユーザーデータの検証は行われません。例えば、ホテルを予約する都市を任意に指定できます。

- 同じ情報 (目的地、受け取り場所、受け取り日付、および返却日時) の一部を再度指定して車を予約します。動的会話では、ホテルの予約でユーザーが前に指定した情報に基づいて、ボットがこの情報の一部を初期化する必要があります。

次のセクションでは、一部のユーザーデータの検証と、セッション属性によるクロスインテント情報共有を使用した初期化を行う Lambda 関数を作成します。次に、ユーザー入力の初期化/検証およびインテントの達成を実行するコードフックとして Lambda 関数を追加して、インテント設定を更新します。

次のステップ

ステップ 3: Lambda 関数を作成する

ステップ 3: Lambda 関数を作成する

このセクションでは、AWS Lambda コンソールに用意されている設計図 (lex-book-trip-python) を使用して Lambda 関数を作成します。また、コンソールで提供されているサンプルイベントデータを使用して Lambda 関数を呼び出すことにより、この関数をテストします。

この Lambda 関数は Python で記述されます。

1. AWS Management Console にサインインして AWS Lambda コンソール (https://console.aws.amazon.com/lambda/) を開きます。
2. [Create function] (関数の作成) を選択します。
3. [設計図の使用] を選択します。「**lex**」と入力して設計図を検索し、`lex-book-trip-python` 設計図を選択します。
4. 次のように Lambda 関数を設定して、[Configure] (設定) を選択します。
   - Lambda 関数の名前 (`BookTripCodeHook`) を入力します。
   - ロールとして [Create a new role from template(s)] を選択し、ロール名を入力します。
   - 他はデフォルト値のままにしておきます。
5. [Create function] (関数の作成) を選択します。
6. 英語 (US) (en-US) 以外のロケールを使用している場合は、特定のロケールの設計図の更新 の説明に従ってインテント名を更新します。
7. Lambda 関数をテストします。ホテルの予約と車の予約の両方のサンプルデータを使用して、Lambda 関数を 2 回呼び出します。

a. [Configure test event] (テストイベント設定)、[Select a test event] (テストイベントの選択) の順に選択します。

b. [Sample event template] (サンプルイベント「テンプレート) リストで、[Amazon Lex Book Hotel] (Amazon Lex ホテルの予約) を選択します。

このサンプルイベントは Amazon Lex のリクエスト/レスポンスモデルと一致します。詳細については、「Lambda 関数を使用する」を参照してください。

c. [Save and test] を選択します。

d. Lambda 関数が正常に実行されたことを確認します。この例のレスポンスは、Amazon Lex レスポンスモデルと一致します。

e. このステップを繰り返します。今回は、[Sample event template] (サンプルイベント「テンプレート) リストで、[Amazon Lex Book Car] (Amazon Lex 車の予約) を選択します。Lambda 関数によって車の予約が処理されます。

次のステップ

ステップ 4: Lambda 関数をコードフックとして追加する

ステップ 4: Lambda 関数をコードフックとして追加する

このセクションでは、初期化/検証とフルフィルメントアクティビティのためのコードフックとして Lambda 関数を追加して、BookCar と BookHotel の両方のインテントの設定を更新します。更新できるのは Amazon Lex リソースの $LATEST バージョンのみであるため、インテントの $LATEST バージョンを選択していることを確認します。

1. Amazon Lex コンソールで、[BookTrip] ボットを選択します。
2. [Editor] タブで、[BookHotel] インテントを選択します。インテント設定を次のように更新します。

a. インテントのバージョン (インテント名の横にある) が $LATEST であることを確認します。

b. Lambda 関数を次のように初期化および検証のコードフックとして追加します。

- [Options] で、[Initialization and validation code hook] を選択します。

- リストから Lambda 関数を選択します。

c. Lambda 関数を次のようにフルフィルメントのコードフックとして追加します。

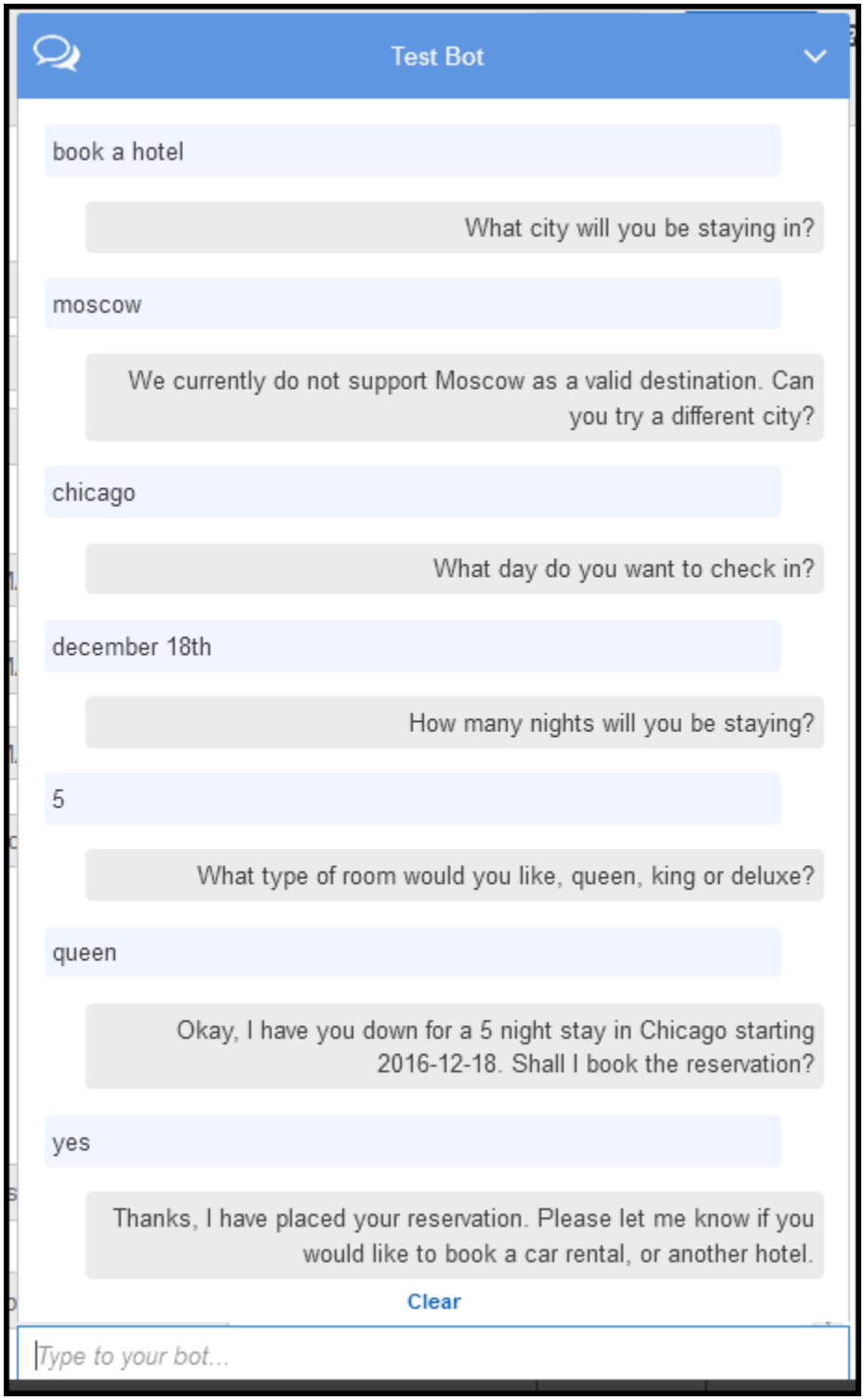
- [Fulfillment] で、[AWS Lambda function] を選択します。
- リストから Lambda 関数を選択します。
- [Goodbye message] を選択し、メッセージを入力します。

d. [Save (保存)] を選択します。

3. [Editor] タブで、BookCar インテントを選択します。前述のステップに従って、Lambda 関数を検証とフルフィルメントのコードフックとして追加します。

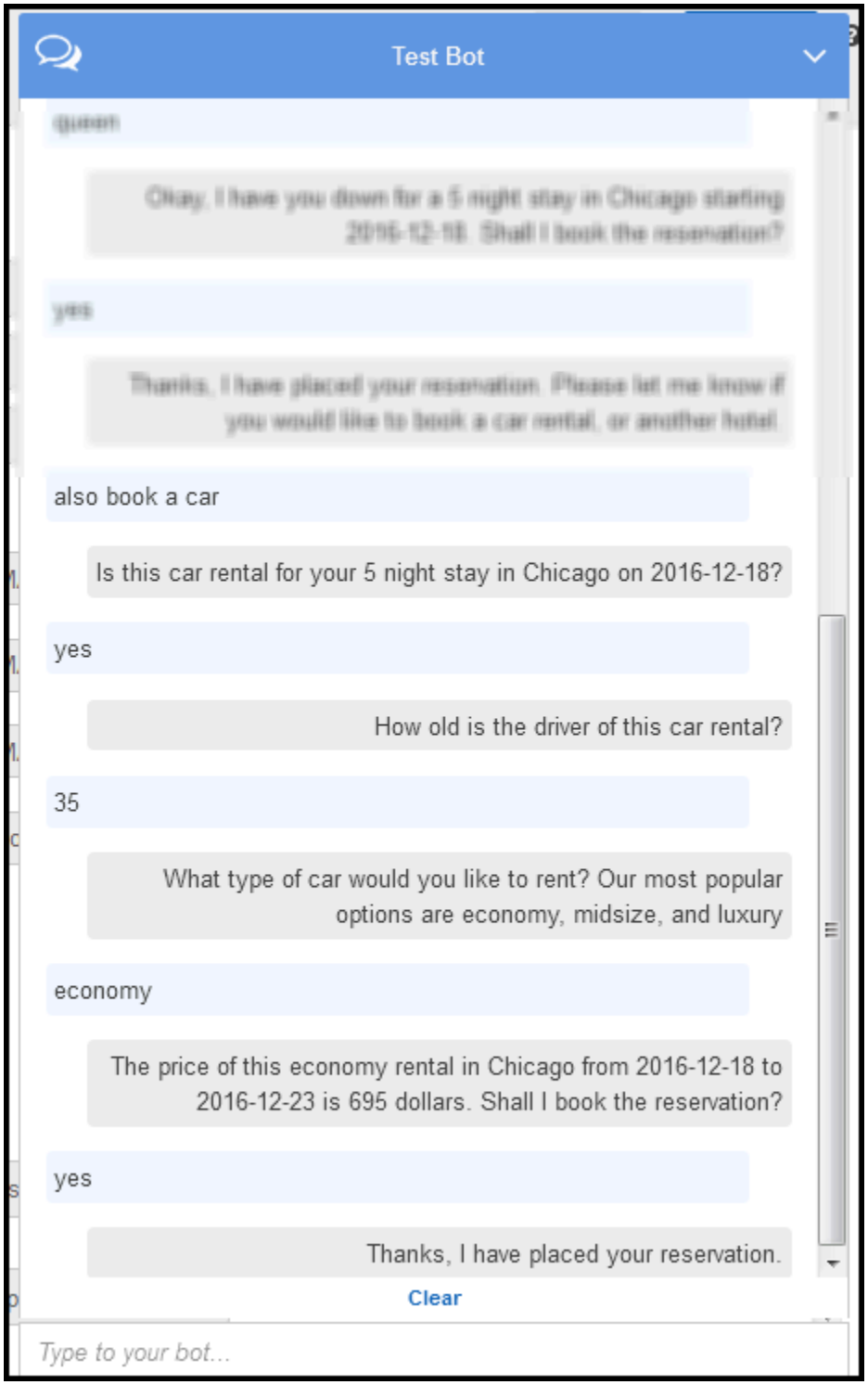
4. [Build] を選択します。コンソールによって、設定を保存するための一連のリクエストが Amazon Lex に送信されます。

5. ボットをテストします。これで初期化、ユーザーデータの検証、およびフルフィルメントを実行する Lambda 関数を作成したので、ユーザー操作での違いを次の会話で確認できます。



クライアント (コンソール) から Amazon Lex へのデータフローおよび Amazon Lex から Lambda 関数へのデータフローの詳細については、「データフロー: BookHotel インテント」を参照してください。

6. 会話を続行して、次の画像ように車を予約します。



車の予約を選択すると、(前の BookHotel での会話からの) セッション属性が含まれているリクエストが、クライアント (コンソール) によって Amazon Lex に送信されます。Amazon Lex はこの情報を Lambda 関数に渡し、その関数によって BookCar のスロットデータの一部 (PickUpDate、ReturnDate、および PickUpCity) が初期化 (事前入力) されます。

> **Note**
>
> これは、セッション属性を使用してインテント間でコンテキストを保持する方法を示しています。コンソールクライアントでは、テストウィンドウにある [Clear] リンクを使用して以前のセッション属性をクリアできます。

クライアント (コンソール) から Amazon Lex へのデータフローおよび Amazon Lex から Lambda 関数へのデータフローの詳細については、「データフロー: BookCar インテント」を参照してください。

情報フローの詳細

この演習では、Amazon Lex コンソールで提供されているテストウィンドウクライアントを使用して、Amazon Lex の BookTrip ボットと会話しました。このセクションでは次の項目について説明します。

- クライアントと Amazon Lex の間のデータフロー。

  このセクションでは、クライアントが `PostText` ランタイム API を使用して Amazon Lex にリクエストを送信することを前提としていて、それに応じてリクエストとレスポンスの詳細を示しています。`PostText` ランタイム API の詳細については、「PostText」を参照してください。

  > **Note**
  >
  > クライアントが `PostContent` API を使用する場合の、クライアントと Amazon Lex の間の情報フローの例については、「ステップ 2a (オプション): 音声による情報フローの詳細を確認する (コンソール)」を参照してください。

- Amazon Lex と Lambda 関数の間のデータフロー。詳細については、「Lambda 関数の入力イベントとレスポンスの形式」を参照してください。

トピック

- データフロー: BookHotel インテント
- データフロー: BookCar インテント

データフロー: BookHotel インテント

このセクションでは、各ユーザー入力の後に何が起こるかを説明します。

1. ユーザー:「ホテルの予約」

   a. クライアント (コンソール) は以下の PostText リクエストを Amazon Lex に送信します。

```
POST /bot/BookTrip/alias/$LATEST/user/wch89kjqcpkds8seny7dly5x3otq68j3/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
   "inputText":"book a hotel",
   "sessionAttributes":{}
}
```

   リクエストの URI と本文の両方で Amazon Lex に情報が提供されています。

   - リクエスト URI – ボット名 (BookTrip)、ボットのエイリアス ($LATEST)、およびユーザー名が提供されています。末尾の `text` では、これが `PostText` API リクエストである (`PostContent` ではない) ことが示されています。
   - リクエストボディ – ユーザー入力 (`inputText`) と空の `sessionAttributes` が含まれています。これは最初は空のオブジェクトであり、Lambda 関数によって最初にセッション属性が設定されます。

   b. Amazon Lex は `inputText` からインテント (BookHotel) を検出します。このインテントには、ユーザーデータの初期化/検証を行うためのコードフックとして Lambda 関数が設定されています。したがって、Amazon Lex は以下の情報をイベントパラメータ (「入力イベントの形式」を参照) として渡して Lambda 関数を呼び出します。

```
{
   "messageVersion":"1.0",
   "invocationSource":"DialogCodeHook",
```

```
   "userId":"wch89kjqcpkds8seny7dly5x3otq68j3",
   "sessionAttributes":{
   },
   "bot":{
      "name":"BookTrip",
      "alias":null,
      "version":"$LATEST"
   },
   "outputDialogMode":"Text",
   "currentIntent":{
      "name":"BookHotel",
      "slots":{
         "RoomType":null,
         "CheckInDate":null,
         "Nights":null,
         "Location":null
      },
      "confirmationStatus":"None"
   }
}
```

クライアントによって送信された情報に加えて、Amazon Lex はまた、以下の追加データも含めます。

- `messageVersion` – 現在 Amazon Lex でサポートしているのは 1.0 バージョンだけです。
- `invocationSource` – Lambda 関数呼び出しの目的を示しています。この場合は、ユーザーデータの初期化および検証を実行することが目的です (この時点で、Amazon Lex はインテントを達成するためのスロットデータの一部をユーザーが指定していないことを知っています)。
- `currentIntent` – すべてのスロット値は null に設定されています。

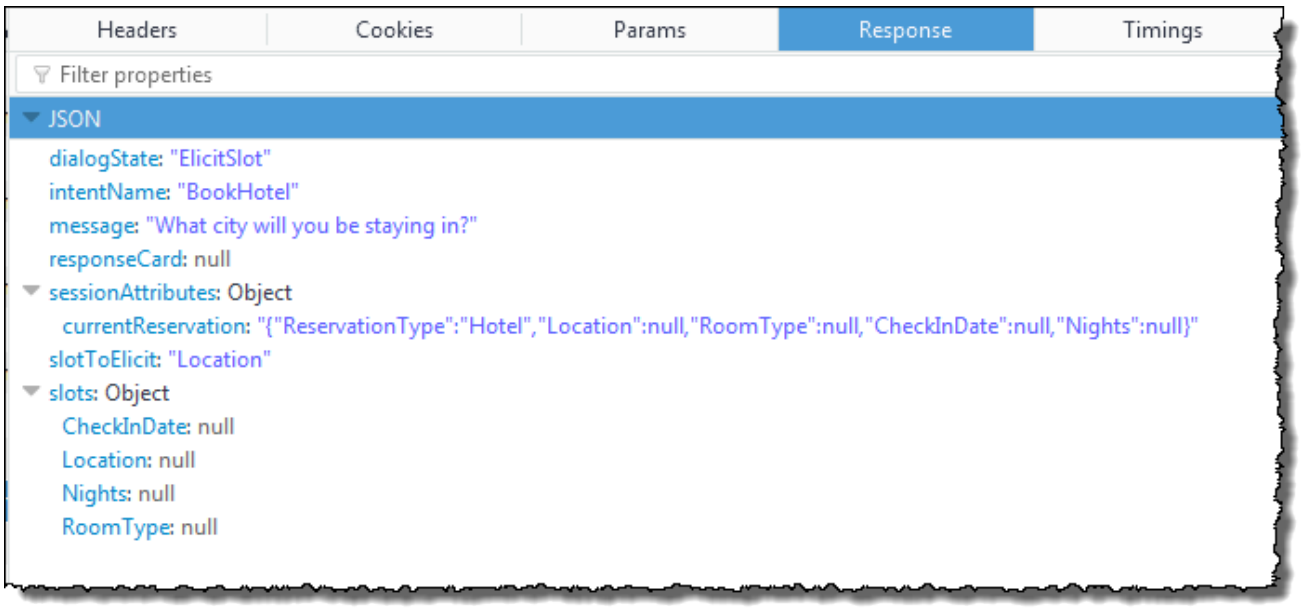
c. この時点では、すべてのスロット値は null です。Lambda 関数が検証する対象はありません。Lambda 関数は以下のレスポンスを Amazon Lex に返します。レスポンスの形式については、「レスポンスの形式」を参照してください。

```
{
   "sessionAttributes":{
      "currentReservation":"{\"ReservationType\":\"Hotel\",\"Location\":null,
\"RoomType\":null,\"CheckInDate\":null,\"Nights\":null}"
   },
```

```
   "dialogAction":{
      "type":"Delegate",
      "slots":{
         "RoomType":null,
         "CheckInDate":null,
         "Nights":null,
         "Location":null
      }
   }
}
```

> **Note**
>
> - `currentReservation` – Lambda 関数はこのセッション属性を含めます。この値は、現在のスロット情報と予約タイプのコピーです。
>
>   これらのセッション属性を更新できるのは Lambda 関数とクライアントだけであり、Amazon Lex はこれらの値を渡すだけです。
> - `dialogAction.type` – この値を `Delegate` に設定すると、Lambda 関数は次の一連のアクションの責任を Amazon Lex に委任します。
>
>   Lambda 関数は、ユーザーデータの検証で何かを検出すると、次に何をするかを Amazon Lex に指示します。

d. Amazon Lex は `dialogAction.type` に従って次の一連のアクションを判断します。つまり、ユーザーから `Location` スロットに対するデータを引き出します。このスロットにプロンプトメッセージの 1 つ (「どの都市に滞在しますか?」) をインテント設定に応じて選択し、ユーザーに次のレスポンスを送信します。



セッション属性がクライアントに渡されます。

クライアントはレスポンスを読み取り、そのメッセージ (「どの都市に滞在されますか?」) を表示します。

2. ユーザー:「モスクワ」

   a. クライアントは以下の PostText リクエスト (読みやすいように改行が追加されています) を Amazon Lex に送信します。

```
POST /bot/BookTrip/alias/$LATEST/user/wch89kjqcpkds8seny7dly5x3otq68j3/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
   "inputText":"Moscow",
   "sessionAttributes":{
       "currentReservation":"{\"ReservationType\":\"Hotel\",
                              \"Location\":null,
                              \"RoomType\":null,
                              \"CheckInDate\":null,
                              \"Nights\":null}"
   }
}
```

クライアントは、inputText に加えて、受信したのと同じ currentReservation セッション属性を含めます。

b. Amazon Lex はまず、現在のインテントのコンテキストの inputText を解釈します (このサービスでは Location スロットに関する情報を特定のユーザーに求めていたことが記憶されています)。現在のインテントのスロット値を更新し、以下のイベントを使用して Lambda 関数を呼び出します。

```
{
    "messageVersion": "1.0",
    "invocationSource": "DialogCodeHook",
    "userId": "wch89kjqcpkds8seny7dly5x3otq68j3",
    "sessionAttributes": {
        "currentReservation": "{\"ReservationType\":\"Hotel\",\"Location
\":null,\"RoomType\":null,\"CheckInDate\":null,\"Nights\":null}"
    },
    "bot": {
        "name": "BookTrip",
        "alias": null,
        "version": "$LATEST"
    },
    "outputDialogMode": "Text",
    "currentIntent": {
        "name": "BookHotel",
        "slots": {
            "RoomType": null,
            "CheckInDate": null,
            "Nights": null,
            "Location": "Moscow"
        },
        "confirmationStatus": "None"
    }
}
```

> **Note**
>
> - invocationSource は DialogCodeHook のままです。このステップではユーザーデータを検証しているだけです。
> - Amazon Lex はセッション属性を Lambda 関数に渡すだけです。

- currentIntent.slots については、Amazon Lex は Location スロットを Moscow に更新しています。

c. Lambda 関数は、ユーザーデータの検証を実行し、Moscow は無効な場所であると判断します。

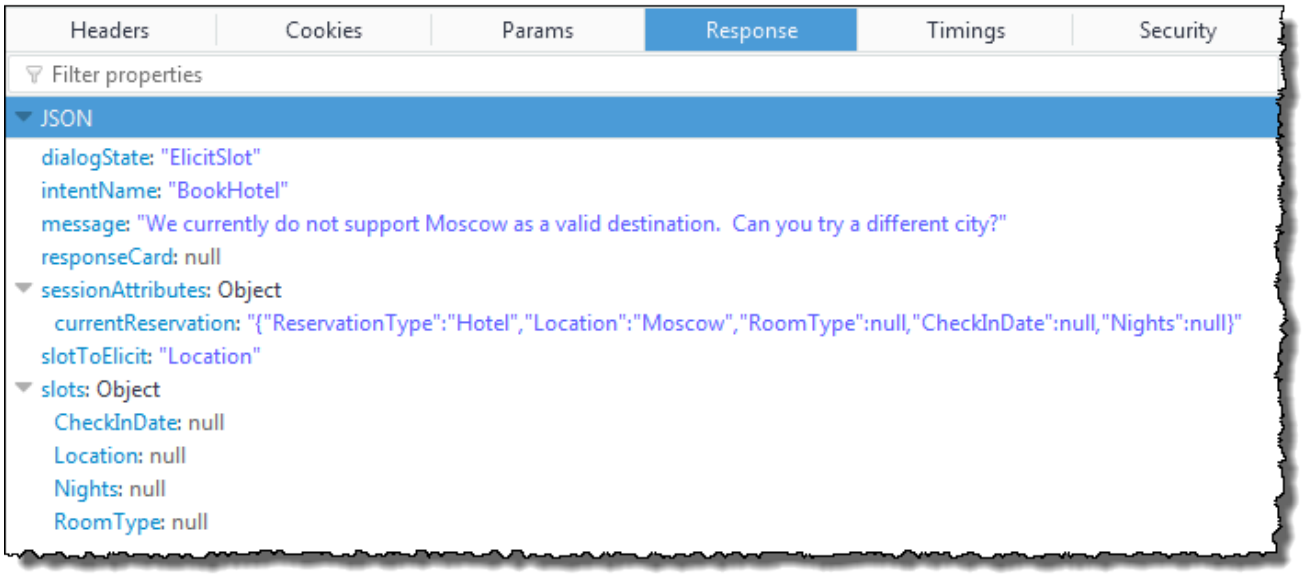
> **Note**
>
> この演習の Lambda 関数には、有効な都市の単純なリストがあり、Moscow はそのリストに存在しません。本稼働アプリケーションでは、この情報の取得にバックエンドデータベースを使用できます。

この関数は、スロット値を null にリセットし、以下のレスポンスを送信して、ユーザーに別の場所を指定し直すように求めることを Amazon Lex に指示します。

```
{
    "sessionAttributes": {
        "currentReservation": "{\"ReservationType\":\"Hotel\",\"Location\":
\"Moscow\",\"RoomType\":null,\"CheckInDate\":null,\"Nights\":null}"
    },
    "dialogAction": {
        "type": "ElicitSlot",
        "intentName": "BookHotel",
        "slots": {
            "RoomType": null,
            "CheckInDate": null,
            "Nights": null,
            "Location": null
        },
        "slotToElicit": "Location",
        "message": {
            "contentType": "PlainText",
            "content": "We currently do not support Moscow as a valid
 destination.  Can you try a different city?"
        }
    }
}
```

> **Note**
>
> - currentIntent.slots.Location は null にリセットされています。
> - dialogAction.type は ElicitSlot に設定されています。これにより、以下を指定してユーザーに再入力を求めるように Amazon Lex に指示しています。
>   - dialogAction.slotToElicit – ユーザーからデータを引き出すスロット。
>   - dialogAction.message – ユーザーに伝えるメッセージ (message)。

d. Amazon Lex は dialogAction.type に気づき、以下のレスポンスでその情報をクライアントに渡します。



クライアントは、「当社では現在、モスクワは有効な目的地ではありません。別の都市を指定いただけますか?」というメッセージを表示するだけです。

3. ユーザー:「シカゴ」

a. クライアント は以下の PostText リクエストを Amazon Lex に送信します。

```
POST /bot/BookTrip/alias/$LATEST/user/wch89kjqcpkds8seny7dly5x3otq68j3/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
   "inputText":"Chicago",
   "sessionAttributes":{
```

```
       "currentReservation":"{\"ReservationType\":\"Hotel\",
                              \"Location\":\"Moscow\",
                              \"RoomType\":null,
                              \"CheckInDate\":null,
                              \"Nights\":null}"
   }
}
```

b. Amazon Lex は、これが Location スロットに対して引き出されているデータであるというコンテキストを知っています。このコンテキストでは、inputText が Location スロットに対する値であることを知っています。Amazon Lex は以下のイベントを送信することで Lambda 関数を呼び出します。

```
{
    "messageVersion": "1.0",
    "invocationSource": "DialogCodeHook",
    "userId": "wch89kjqcpkds8seny7dly5x3otq68j3",
    "sessionAttributes": {
        "currentReservation": "{\"ReservationType\":\"Hotel\",\"Location
\":Moscow,\"RoomType\":null,\"CheckInDate\":null,\"Nights\":null}"
    },
    "bot": {
        "name": "BookTrip",
        "alias": null,
        "version": "$LATEST"
    },
    "outputDialogMode": "Text",
    "currentIntent": {
        "name": "BookHotel",
        "slots": {
            "RoomType": null,
            "CheckInDate": null,
            "Nights": null,
            "Location": "Chicago"
        },
        "confirmationStatus": "None"
    }
}
```

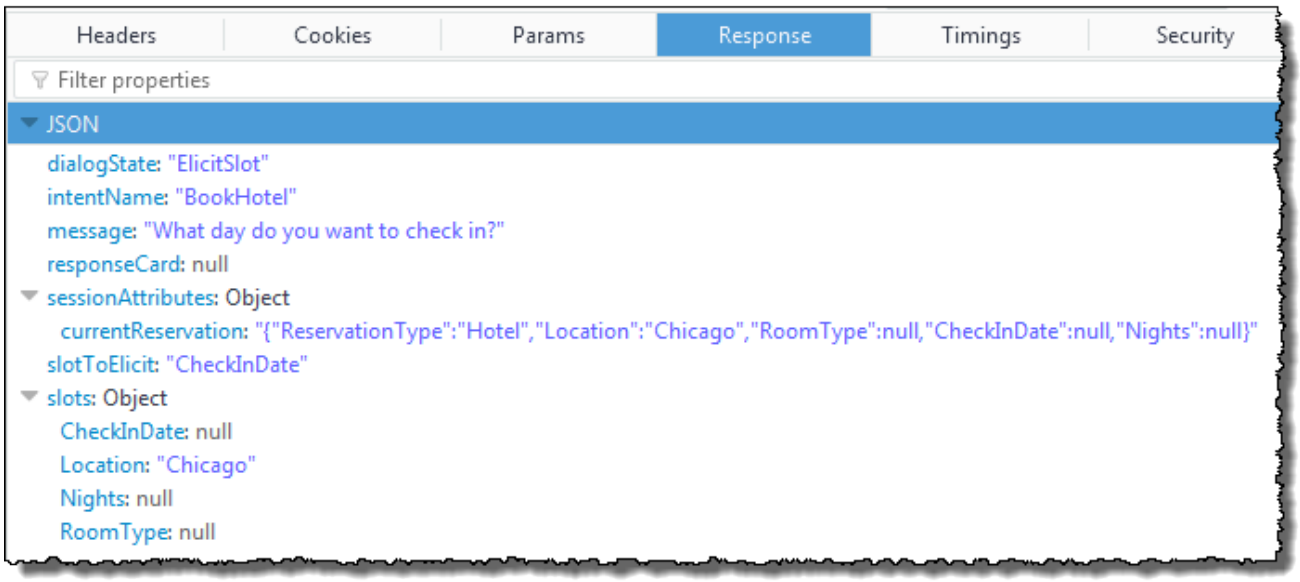
Amazon Lex は Location スロットを Chicago に設定して、currentIntent.slots を更新します。

c. invocationSource の DialogCodeHook の値に従って、Lambda 関数はユーザーデータの検証を実行します。この関数は、Chicago を有効なスロット値として認識し、これに応じてセッション属性を更新して、以下のレスポンスを Amazon Lex に返します。

```
{
    "sessionAttributes": {
        "currentReservation": "{\"ReservationType\":\"Hotel\",\"Location\":
\"Chicago\",\"RoomType\":null,\"CheckInDate\":null,\"Nights\":null}"
    },
    "dialogAction": {
        "type": "Delegate",
        "slots": {
            "RoomType": null,
            "CheckInDate": null,
            "Nights": null,
            "Location": "Chicago"
        }
    }
}
```

> **Note**
>
> - currentReservation – Lambda 関数は Location を Chicago に設定して、このセッション属性を更新します。
> - dialogAction.type - Delegate に設定されます。ユーザーデータは有効であり、Lambda 関数は、次の一連のアクションを選択するように Amazon Lex に指示します。

d. Amazon Lex は dialogAction.type に従って次の一連のアクションを選択します。Amazon Lex は、より多くのスロットデータが必要であることを知っているため、インテント設定に従って最も優先度が高い次の未指定スロット (CheckInDate) を選択します。このスロットにプロンプトメッセージの 1 つ (「何日にチェックインしますか?」) をインテント設定に応じて選択し、クライアントに次のレスポンスを返答として送信します。

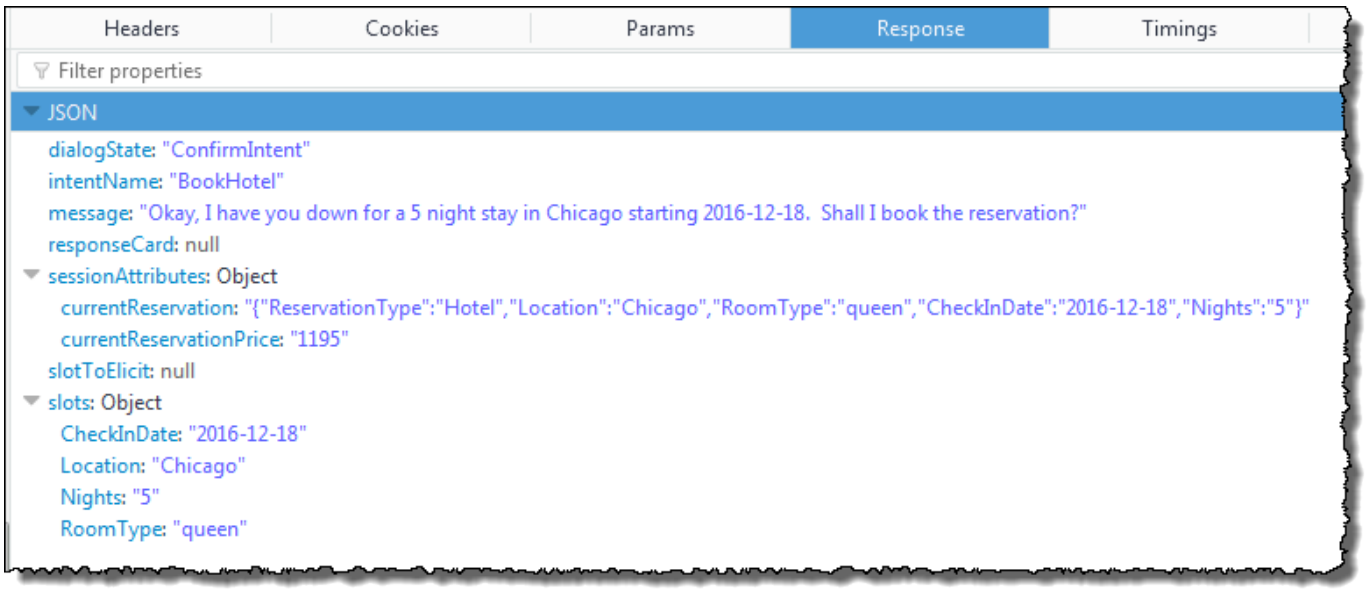


クライアントは「何日にチェックインされますか?」のメッセージを表示します。

4. ユーザーの操作が続行され、ユーザーがデータを提供し、Lambda 関数がデータを検証して次の一連のアクションを Amazon Lex に委任します。最終的に、ユーザーはすべてのスロットデータを提供し、Lambda 関数はすべてのユーザー入力を検証し、Amazon Lex はすべてのスロットデータがそろっていることを認識します。

> **Note**
>
> この演習では、ユーザーがすべてのスロットデータを提供すると、Lambda 関数はホテル予約の価格を計算し、それを別のセッション属性 (`currentReservationPrice`) として返します。

この時点で、このインテントが達成される準備ができていますが、BookHotel インテントでは、Amazon Lex がインテントを達成する前にユーザーの確認を必要とする確認プロンプトが設定されています。したがって、Amazon Lex は、ホテルを予約する前に確認をリクエストする以下のメッセージをクライアントに送信します。



クライアントは「かしこまりました。シカゴでの 2016 年 12 月 18 日から 5 泊の滞在を承りました」のメッセージを表示します。この内容で予約いたしましょうか?」

5. ユーザー:「はい」

   a. クライアント は以下の PostText リクエストを Amazon Lex に送信します。

```
POST /bot/BookTrip/alias/$LATEST/user/wch89kjqcpkds8seny7dly5x3otq68j3/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
   "inputText":"Yes",
   "sessionAttributes":{
       "currentReservation":"{\"ReservationType\":\"Hotel\",
                              \"Location\":\"Chicago\",
                              \"RoomType\":\"queen\",
                              \"CheckInDate\":\"2016-12-18\",
                              \"Nights\":\"5\"}",
      "currentReservationPrice":"1195"
   }
}
```

   b. Amazon Lex は、現在のインテントの確認のコンテキストで inputText を解釈して、Amazon Lex はユーザーが予約を進めることを望んでいることを理解しています。Amazon Lex は今回は、以下のイベントを送信して Lambda 関数を呼び出し、インテントを達成します。このイベントで invocationSource を FulfillmentCodeHook

に設定することで、Amazon Lex は Lambda 関数に送信します。Amazon Lex はまた、confirmationStatus から Confirmed に設定します。

```
{
    "messageVersion": "1.0",
    "invocationSource": "FulfillmentCodeHook",
    "userId": "wch89kjqcpkds8seny7dly5x3otq68j3",
    "sessionAttributes": {
        "currentReservation": "{\"ReservationType\":\"Hotel\",\"Location\":
\"Chicago\",\"RoomType\":\"queen\",\"CheckInDate\":\"2016-12-18\",\"Nights\":
\"5\"}",
        "currentReservationPrice": "956"
    },
    "bot": {
        "name": "BookTrip",
        "alias": null,
        "version": "$LATEST"
    },
    "outputDialogMode": "Text",
    "currentIntent": {
        "name": "BookHotel",
        "slots": {
            "RoomType": "queen",
            "CheckInDate": "2016-12-18",
            "Nights": "5",
            "Location": "Chicago"
        },
        "confirmationStatus": "Confirmed"
    }
}
```

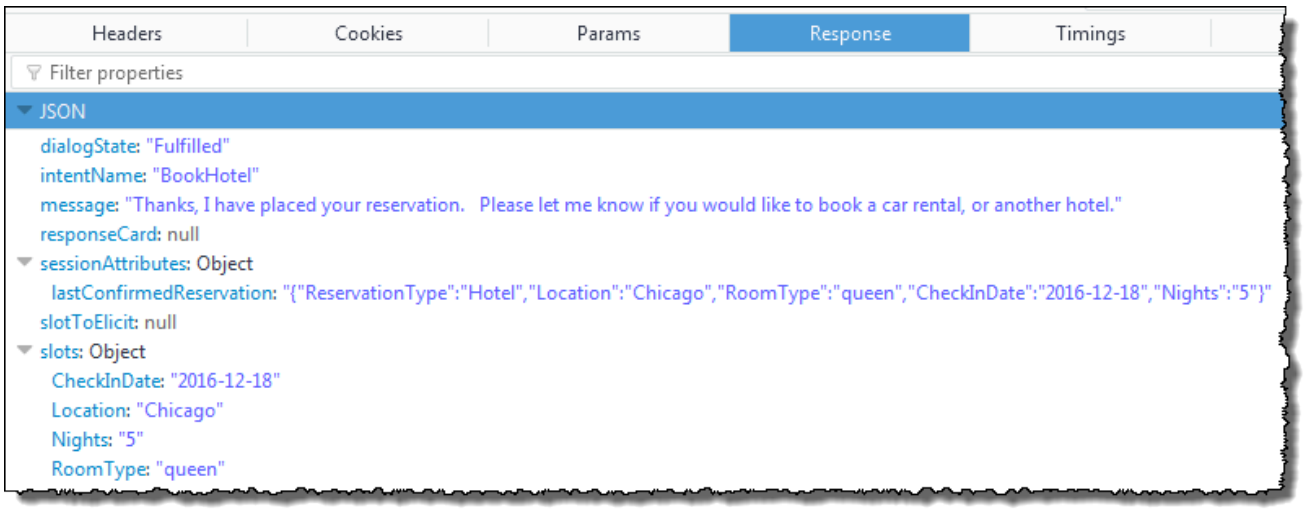
> **Note**
>
> - invocationSource – Amazon Lex は今回はこの値を FulfillmentCodeHook に設定して、インテントを達成するように Lambda 関数に指示しています。
> - confirmationStatus - Confirmed に設定されます。

c. Lambda 関数は今回は BookHotel インテントを達成し、Amazon Lex は予約を完了して、次のレスポンスを返します。

```
{
    "sessionAttributes": {
        "lastConfirmedReservation": "{\"ReservationType\":\"Hotel\",\"Location
\":\"Chicago\",\"RoomType\":\"queen\",\"CheckInDate\":\"2016-12-18\",\"Nights
\":\"5\"}"
    },
    "dialogAction": {
        "type": "Close",
        "fulfillmentState": "Fulfilled",
        "message": {
            "contentType": "PlainText",
            "content": "Thanks, I have placed your reservation.   Please let me
 know if you would like to book a car rental, or another hotel."
        }
    }
}
```

> **Note**
>
> - `lastConfirmedReservation` – Lambda 関数が追加した新しいセッション属性です (`currentReservation` や `currentReservationPrice` ではない)。
> - `dialogAction.type` – Lambda 関数はこの値を `Close` に設定して、ユーザーの応答を想定しないことを Amazon Lex に示しています。
> - `dialogAction.fulfillmentState` – `Fulfilled` に設定されていて、ユーザーに伝える適切なメッセージ (`message`) が含まれています。

d. Amazon Lex は `fulfillmentState` を確認し、以下のレスポンスをクライアントに送信します。



> **Note**
>
> - `dialogState` – Amazon Lex はこの値を `Fulfilled` に設定しています。
> - `message` – Lambda 関数が提供したのと同じメッセージです。

クライアントはそのメッセージを表示します。

データフロー: BookCar インテント

この演習の BookTrip ボットでは 2 つのインテント (BookHotel と BookCar) がサポートされています。ホテルを予約した後に、ユーザーは会話を続行して車を予約できます。セッションがタイムアウトしない限り、それ以降の各リクエストで、クライアントはセッション属性 (この例では、`lastConfirmedReservation`) を送信し続けます。Lambda 関数はこの情報を使用して BookCar インテントのスロットデータを初期化できます。この例では、クロスインテント情報共有でのセッション属性の使用方法を示しています。

具体的には、ユーザーが BookCar インテントを選択すると、Lambda 関数ではセッション属性にある関連情報を使用して、BookCar インテントのスロットデータ (PickUpDate、ReturnDate、および PickUpCity) が事前に設定されます。

> **Note**
>
> Amazon Lex コンソールでは、[Clear] (クリア) リンクを使用して以前のセッション属性をクリアできます。

この手順の以下のステップに従って会話を続行します。

1. ユーザー:「車も予約します」

   a. クライアント は以下の PostText リクエストを Amazon Lex に送信します。

```
POST /bot/BookTrip/alias/$LATEST/user/wch89kjqcpkds8seny7dly5x3otq68j3/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
   "inputText":"also book a car",
   "sessionAttributes":{
       "lastConfirmedReservation":""{\"ReservationType\":\"Hotel\",
                                    \"Location\":\"Chicago\",
                                    \"RoomType\":\"queen\",
                                    \"CheckInDate\":\"2016-12-18\",
                                    \"Nights\":\"5\"}"
   }
}
```

   クライアントは lastConfirmedReservation セッション属性を含めます。

   b. Amazon Lex は inputText からインテント (BookCar) を検出します。このインテントも、Lambda 関数を呼び出して、初期化とユーザーデータの検証を実行するように設定されています。Amazon Lex は以下のイベントを使用して Lambda 関数を呼び出します。

```
{
    "messageVersion": "1.0",
    "invocationSource": "DialogCodeHook",
    "userId": "wch89kjqcpkds8seny7dly5x3otq68j3",
    "sessionAttributes": {
        "lastConfirmedReservation": "{\"ReservationType\":\"Hotel\",\"Location
\":\"Chicago\",\"RoomType\":\"queen\",\"CheckInDate\":\"2016-12-18\",\"Nights
\":\"5\"}"
    },
```

```
    "bot": {
        "name": "BookTrip",
        "alias": null,
        "version": "$LATEST"
    },
    "outputDialogMode": "Text",
    "currentIntent": {
        "name": "BookCar",
        "slots": {
            "PickUpDate": null,
            "ReturnDate": null,
            "DriverAge": null,
            "CarType": null,
            "PickUpCity": null
        },
        "confirmationStatus": "None"
    }
}
```

> **Note**
>
> - `messageVersion` – 現在 Amazon Lex でサポートされているのは 1.0 バージョンだけです。
> - `invocationSource` – 呼び出しの目的が初期化とユーザーデータの検証の実行であることを示しています。
> - `currentIntent` – インテント名とスロットが含まれています。この時点では、すべてのスロット値は null です。

c. Lambda 関数は、すべてのスロット値が null であり、検証する対象がないことに気づきます。ただし、この関数は、セッション属性を使用して一部のスロットデータ (`PickUpDate`、`ReturnDate`、および `PickUpCity`) を初期化し、以下のレスポンスを返します。

```
{
    "sessionAttributes": {
        "lastConfirmedReservation": "{\"ReservationType\":\"Hotel\",\"Location
\":\"Chicago\",\"RoomType\":\"queen\",\"CheckInDate\":\"2016-12-18\",\"Nights
\":\"5\"}",
```

```
        "currentReservation": "{\"ReservationType\":\"Car\",\"PickUpCity
\":null,\"PickUpDate\":null,\"ReturnDate\":null,\"CarType\":null}",
        "confirmationContext": "AutoPopulate"
    },
    "dialogAction": {
        "type": "ConfirmIntent",
        "intentName": "BookCar",
        "slots": {
            "PickUpCity": "Chicago",
            "PickUpDate": "2016-12-18",
            "ReturnDate": "2016-12-22",
            "CarType": null,
            "DriverAge": null
        },
        "message": {
            "contentType": "PlainText",
            "content": "Is this car rental for your 5 night stay in Chicago on
 2016-12-18?"
        }
    }
}
```

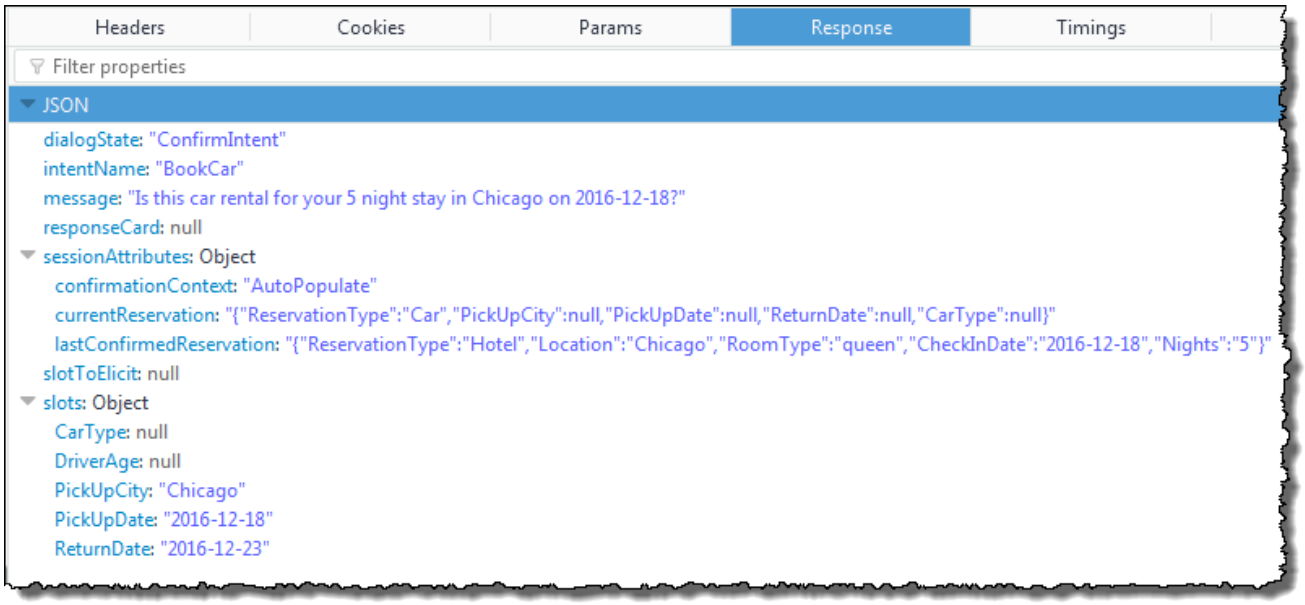
Note

- Lambda 関数は、`lastConfirmedReservation` に加えて、より多くのセッション属性 (`currentReservation` および `confirmationContext`) を含めています。
- ユーザーから「はい/いいえ」の応答が想定されていることを Amazon Lex に通知するために、`dialogAction.type` が `ConfirmIntent` に設定されています。confirmationContext が AutoPopulate に設定されているため、「はい/いいえ」のユーザー応答が、 関数で実行された初期化 (自動入力されたスロットデータ) に対するユーザーの確認を得るためであることを Lambda 関数はわかっています。

  また、Lambda 関数は、Amazon Lex がクライアントに返す `dialogAction.message` の情報メッセージをレスポンスに含めています。

> **Note**
>
> `ConfirmIntent` (`dialogAction.type` の値) という用語は、ボットのインテントとは関係ありません。この例では、ユーザーから「はい/いいえ」の応答を得るように Amazon Lex に指示するために、Lambda 関数でこの用語が使用されています。

d. `dialogAction.type` に従って、Amazon Lex は以下のレスポンスをクライアントに返します。



クライアントは「このレンタカーはシカゴでの 2016 年 12 月 18 日から 5 泊の滞在で使用されますか?」のメッセージを表示します。

2. ユーザー:「はい」

a. クライアント は以下の `PostText` リクエストを Amazon Lex に送信します。

```
POST /bot/BookTrip/alias/$LATEST/user/wch89kjqcpkds8seny7dly5x3otq68j3/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
   "inputText":"yes",
   "sessionAttributes":{
      "confirmationContext":"AutoPopulate",
```

```
      "currentReservation":"{\"ReservationType\":\"Car\",
                             \"PickUpCity\":null,
                             \"PickUpDate\":null,
                             \"ReturnDate\":null,
                             \"CarType\":null}",
      "lastConfirmedReservation":"{\"ReservationType\":\"Hotel\",
                                  \"Location\":\"Chicago\",
                                  \"RoomType\":\"queen\",
                                  \"CheckInDate\":\"2016-12-18\",
                                  \"Nights\":\"5\"}"
   }
}
```

b. Amazon Lex は inputText を読み取り、そのコンテキスト (自動入力の確認をユーザーに求めた) を知っています。Amazon Lex は以下のイベントを送信して Lambda 関数を呼び出します。

```
{
    "messageVersion": "1.0",
    "invocationSource": "DialogCodeHook",
    "userId": "wch89kjqcpkds8seny7dly5x3otq68j3",
    "sessionAttributes": {
        "confirmationContext": "AutoPopulate",
        "currentReservation": "{\"ReservationType\":\"Car\",\"PickUpCity
\":null,\"PickUpDate\":null,\"ReturnDate\":null,\"CarType\":null}",
        "lastConfirmedReservation": "{\"ReservationType\":\"Hotel\",\"Location
\":\"Chicago\",\"RoomType\":\"queen\",\"CheckInDate\":\"2016-12-18\",\"Nights
\":\"5\"}"
    },
    "bot": {
        "name": "BookTrip",
        "alias": null,
        "version": "$LATEST"
    },
    "outputDialogMode": "Text",
    "currentIntent": {
        "name": "BookCar",
        "slots": {
            "PickUpDate": "2016-12-18",
            "ReturnDate": "2016-12-22",
            "DriverAge": null,
            "CarType": null,
            "PickUpCity": "Chicago"
```

```
        },
        "confirmationStatus": "Confirmed"
    }
}
```

ユーザーが「はい」と応答しているため、Amazon Lex は confirmationStatus を Confirmed に設定します。

c. confirmationStatus によって、Lambda 関数は事前入力された値が正しいことを知っています。Lambda 関数は以下を実行します。

- currentReservation セッション属性を、事前入力されていたスロット値に更新します。
- dialogAction.type を ElicitSlot に設定します。
- slotToElicit の値を DriverAge に設定します。

以下の応答が送信されます。

```
{
    "sessionAttributes": {
        "currentReservation": "{\"ReservationType\":\"Car\",\"PickUpCity
\":\"Chicago\",\"PickUpDate\":\"2016-12-18\",\"ReturnDate\":\"2016-12-22\",
\"CarType\":null}",
        "lastConfirmedReservation": "{\"ReservationType\":\"Hotel\",\"Location
\":\"Chicago\",\"RoomType\":\"queen\",\"CheckInDate\":\"2016-12-18\",\"Nights
\":\"5\"}"
    },
    "dialogAction": {
        "type": "ElicitSlot",
        "intentName": "BookCar",
        "slots": {
            "PickUpDate": "2016-12-18",
            "ReturnDate": "2016-12-22",
            "DriverAge": null,
            "CarType": null,
            "PickUpCity": "Chicago"
        },
        "slotToElicit": "DriverAge",
        "message": {
            "contentType": "PlainText",
            "content": "How old is the driver of this car rental?"
```

```
        }
    }
}
```

d. Amazon Lex は以下のレスポンスを返します。

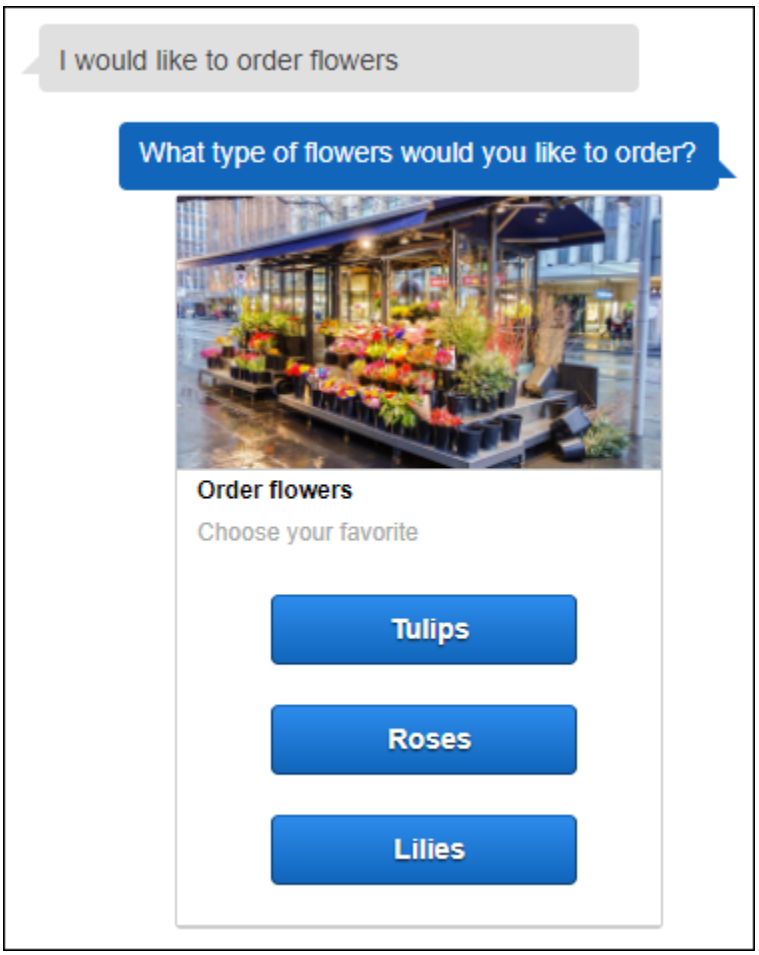
クライアントは「このレンタカーを運転される方の年齢を教えていただけますか?」のメッセージを表示します。会話が続行されます。

レスポンスカードの使用

この演習では、レスポンスカードを追加して「開始方法」の演習 1 を拡張します。OrderFlowers インテントをサポートするボットを作成し、FlowerType スロットのレスポンスカードを追加してそのインテントを更新します。FlowerType スロットの次のプロンプトの他に、ユーザーはレスポンスカードから花の種類を選択できます。

```
What type of flowers would you like to order?
```

以下はレスポンスカードです。



ボットのユーザーは、花の種類をテキスト入力するか、リストから選択できます。このレスポンスカードにはイメージが設定されています。これは次のようにクライアントに表示されます。レスポンスカードの詳細情報については、「レスポンスカード」を参照してください。

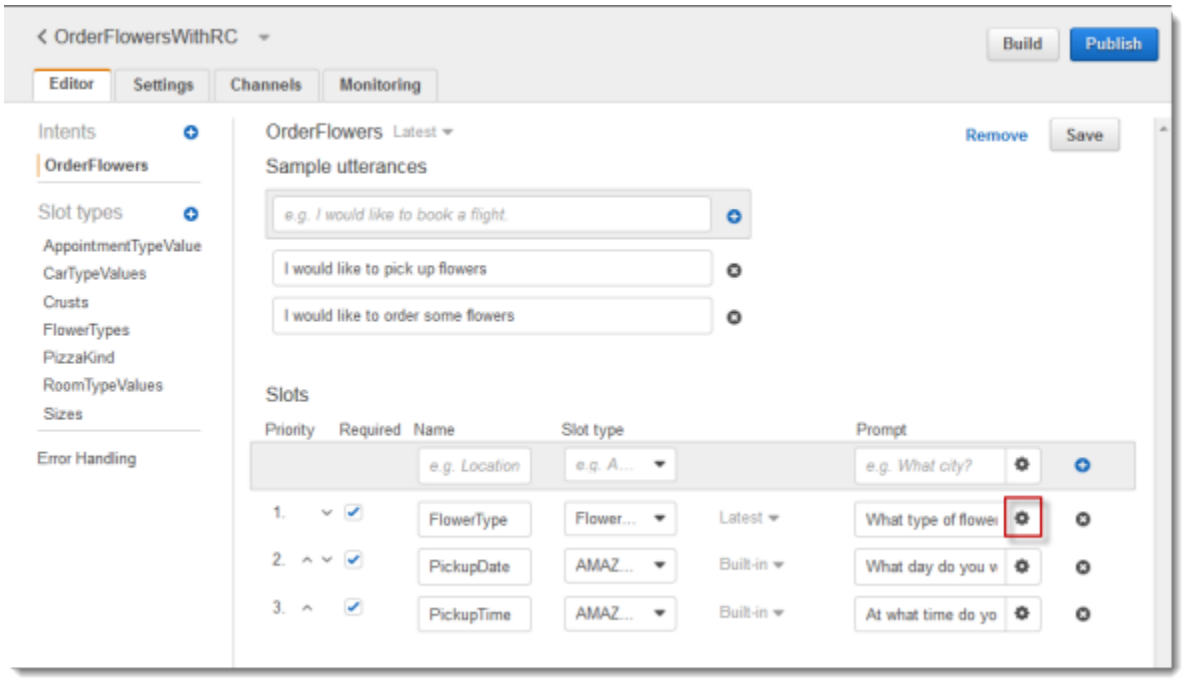
ボットを作成して、レスポンスカードでテストするには

1. 「開始方法」の演習 1 の手順に従って、OrderFlowers ボットを作成しテストします。ステップ 1、2、3 を完了する必要があります。レスポンスカードをテストするために Lambda 関数を追加する必要はありません。手順については、「演習 1: 設計図を使用して Amazon Lex ボットを作成する (コンソール)」を参照してください。
2. レスポンスカードを追加してボットを更新し、バージョンを発行します。バージョンを発行し、そのバージョンを参照するエイリアス (BETA) を指定します。

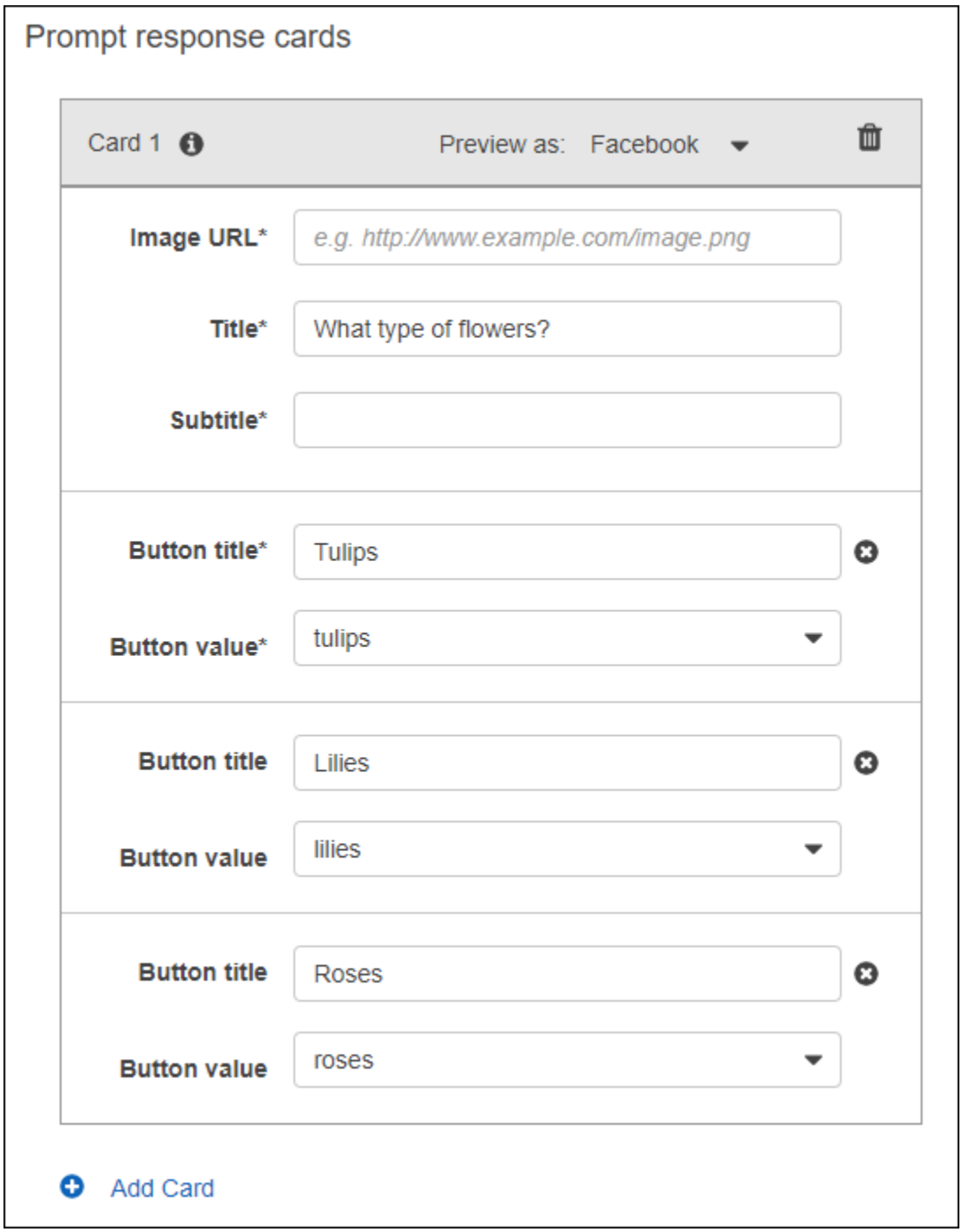
   a. Amazon Lex コンソールでボットを選択します。

   b. [OrderFlowers] インテントを選択します。

c. 次の画像のように、[プロンプト] で「What type of flowers」の横にある設定の歯車アイコンを選択して、`FlowerType` のレスポンスカードを設定します。



d. 次のスクリーンショットに示すように、カードにタイトルを付け、3 つのボタンを設定します。オプションでイメージの URL を指定して、レスポンスカードにイメージを追加することもできます。Twilio SMS を使用してボットをデプロイする場合、イメージの URL を指定する必要があります。



e. [Save] を選択してレスポンスカードを保存します。

f. [Save intent] を選択して、インテントの設定を保存します。

g. ボットを構築するには、[Build] を選択します。

h. ボットのバージョンを発行するには、[Publish] を選択します。ボットのバージョンを参照するエイリアスとして BETA を指定します。バージョニングの詳細については、「バージョニングとエイリアス」を参照してください。

3. メッセージングプラットフォームのボットをデプロイします。

   - Facebook Messenger プラットフォームでボットをデプロイし、統合をテストします。手順については、「Amazon Lex ボットと Facebook Messenger の統合」を参照してください。花を注文する際に、メッセージウィンドウにレスポンスカードが表示され、花の種類が選択できます。
   - Slack プラットフォームでボットをデプロイし、統合をテストします。手順については、「Amazon Lex ボットと Slack との統合 」を参照してください。花を注文する際に、メッセージウィンドウにレスポンスカードが表示され、花の種類が選択できます。
   - Twilio SMS プラットフォームのボットをデプロイします。手順については、「Amazon Lex ボットと Twilio プログラム可能 SMS の統合 」を参照してください。花を注文すると、Twilio のメッセージにレスポンスカードからのイメージが表示されます。Twilio SMS はレスポンスでボタンをサポートしていません。

発話の更新

この演習では、「開始方法」の演習 1 で作成した発話に新しい発話を追加します。Amazon Lex コンソールの [Monitoring] (モニタリング) タブを使用して、ボットで認識されなかった発話を確認します。ユーザーのエクスペリエンスを向上させるために、これらの発話をボットに追加します。

発話の統計は、以下の条件では生成されません。

- ボットが作成されたとき、`childDirected` フィールドが TRUE に設定されます。
- 1 つ以上のスロットでスロットの難読化を実行しています。
- Amazon Lex の改善への参加をオプトアウトしました。

> **Note**
>
> 発話の統計は 1 日に 1 回生成されます。認識されなかった発話、聞いた回数、発話を最後に聞いた日付と時間を確認できます。見逃した発話がコンソールに表示されるまでに最大 24 時間かかる場合があります。

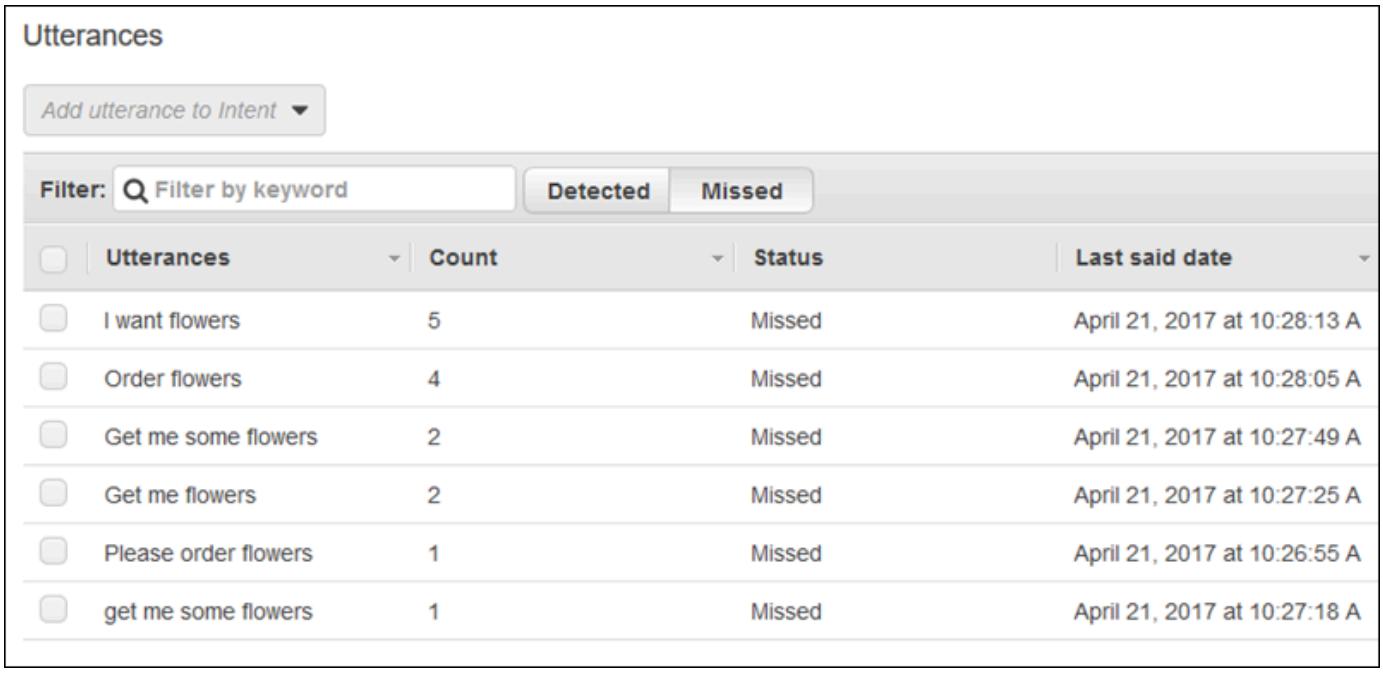
異なるバージョンのボットの発話を確認できます。表示されているボットのバージョンを変更するには、ボット名の横にあるドロップダウンから別のバージョンを選択します。

見逃した発話を確認してボットに追加するには

1. 「開始方法」の演習 1 の最初のステップに従って、`OrderFlowers` ボットを作成しテストします。手順については、「演習 1: 設計図を使用して Amazon Lex ボットを作成する (コンソール)」を参照してください。
2. 次の発話を [Test Bot] ウィンドウに入力してボットをテストします。各発話を数回入力します。サンプルボットで認識されない発話は以下のとおりです。

   - Order flowers
   - Get me flowers
   - Please order flowers
   - Get me some flowers

3. Amazon Lex が見逃した発話に関する使用状況データを収集するまで待機します。発話データは、1 日 1 回、通常、深夜に生成されます。
4. AWS Management Console にサインインし、Amazon Lex コンソール (https://console.aws.amazon.com/lex/) を開きます。
5. [`OrderFlowers`] ボットを選択します。
6. [Monitoring] タブを選択し、左のメニューで [Utterances] を選択します。次に、[Missed] ボタンを選択します。次のペインには、最大 100 個の見逃した発話が表示されます。



7. 見逃した発話を選択してボットに追加するには、発話の横にあるチェックボックスをオンにします。発話をインテントの $LATEST バージョンに追加するには、[Add utterance to intent] ドロップダウンの横にある下向き矢印を選択し、インテントを選択します。
8. ボットを再構築するには、[Build] を選択し、再度 [Build] を選択します。
9. ボットで新しい発話が認識されることを確認するには、[Test Bot] ペインを使用します。

ウェブサイトとの統合

この例では、ボットとウェブサイトを統合し、テキストと音声を使用します。JavaScript と AWS のサービスを使用して、ウェブサイトの訪問者向けにインタラクティブなエクスペリエンスを構築します。AWS AI ブログに示されている例から選択できます。

- chatbot のウェブ UI のデプロイ - Amazon Lex chatbot のウェブクライアントを提供する、豊富な機能を備えたウェブ UI をデモします。これを使用してウェブクライアントについて学習したり、お客様独自のアプリケーションの構成要素としたりできます。

- 「ようこそ、訪問者様」Amazon Lex を使用してウェブユーザーと会話する - AWS SDK for JavaScript in the Browser、および Amazon Cognito を使用してウェブサイトでの会話エクスペリエンスを作成する方法をデモします。

- ブラウザで音声入力をキャプチャして、Amazon Lex に送信する - SDK for JavaScript in the Browser を使用して、ウェブサイトに音声ベースの chatbot を埋め込む方法を示します。アプリケーションは音声を録音し、その音声を Amazon Lex に送信して、レスポンスを再生します。

コールセンターエージェントアシスタント

このチュートリアルでは、Amazon Lex と Amazon Kendra を使用して、カスタマーサポートエージェントを支援するエージェントアシストボットを構築し、ウェブアプリケーションとして公開します。Amazon Kendra は、機械学習を使用してドキュメントを検索して回答を見つけるエンタープライズ検索サービスです。Amazon Kendra の詳細については、「Amazon Kendra Developer Guide」(Amazon Kendra デベロッパーガイド) を参照してください。

Amazon Lex ボットは、顧客にとって最初のお問い合わせ先として、コールセンターで広く使用されています。ボットは、顧客のほとんどの質問を解決できます。ボットが質問に答えられない場合は、カスタマーサポートの従業員に会話を転送します。

このチュートリアルでは、エージェントがリアルタイムで顧客の問い合わせに応答するために使用する Amazon Lex ボットを作成します。ボットが提供する回答を読むことで、エージェント自身が手動で回答を検索する必要が無くなります。

このチュートリアルで作成するボットとウェブアプリケーションは、適切なリソースを迅速に提供することで、エージェントが顧客に対して効率的かつ正確に対応できるようにします。次の図表は、ウェブアプリケーションの動作を示しています。

この図表が示すように、ドキュメントの Amazon Kendra インデックスは Amazon Simple Storage Service (Amazon S3) バケットに保存されます。S3 バケットの設定がお済みでない場合は、Amazon Kendra インデックスを作成するときにバケットをセットアップすることができます。Amazon S3 に加えて、このチュートリアルでは Amazon Cognito を使用します。Amazon Cognito は、ボットをウェブアプリケーションとしてデプロイするためのアクセス許可を管理します。

このチュートリアルでは、顧客の質問に対する回答を提供する Amazon Kendra インデックスを作成し、ボットを作成し、顧客との会話に基づいて回答を提案できるようにするインテントを追加し、アクセス権限を管理する Amazon Cognito を設定し、ボットをウェブアプリケーションとしてデプロイします。

予測時間: 75 分

予測コスト: Amazon Kendra インデックスでは 1 時間あたり 2.50 ドル、Amazon Lex のリクエスト 1000 件に対して 0.75 USD。この演習を終了した後も、Amazon Kendra インデックスは引き続き実行されています。不要なコストを避けるために、必ず削除してください。

注意: このチュートリアルで使用するすべてのサービスに対して、同じ AWS リージョンを選択してください。

トピック

ステップ 1: Amazon Kendra インデックスを作成する

まず、顧客の質問に答えるドキュメントの Amazon Kendra インデックスを作成します。インデックスは、クライアントからの問い合わせに対する検索 API を提供するものです。インデックスは、ソースドキュメントから作成します。Amazon Kendra は、インデックス付きドキュメントで見つかった回答をボットに返し、その回答をエージェントに表示します。

Amazon Kendra が提案する回答の質と正確性は、インデックスを作成するドキュメントによって異なります。ドキュメントには、エージェントが頻繁にアクセスするファイルが含まれて、S3 バケットに保存されている必要があります。非構造化データおよび半構造化データは、.html、Microsoft Office (.doc、.ppt)、PDF、およびテキスト形式でインデックスを作成できます。

Amazon Kendra インデックスを作成するには、「Amazon Kendra Developer Guide」(Amazon Kendra デベロッパーガイド) の「Getting started with an S3 bucket (console)」(S3 バケットの使用を開始する (コンソール)) を参照してください。

顧客からのお問い合わせにお答えするための質問と回答 (よくある質問) を追加するには、「Amazon Kendra Developer Guide」(Amazon Kendra デベロッパーガイド) の「Adding questions and answers」(質問と回答を追加する) を参照してください。このチュートリアルでは、GitHub の ML_FAQ.csv ファイルを使用します。

次のステップ

ステップ 2: Amazon Lex ボットを作成する

ステップ 2: Amazon Lex ボットを作成する

Amazon Lex では、コールセンターエージェントと Amazon Kendra インデックス間のインターフェイスをご用意しています。エージェントと顧客の会話を記録し、顧客の質問に応じて

AMAZON.KendraSearchIntent のインテントを呼び出します。インテントとは、ユーザーが実行を望んでいるアクションです。

Amazon Kendra は、インデックス付きドキュメントを検索し、ボットに表示される Amazon Lex に対する回答を返します。この回答はエージェントにのみ表示されます。

エージェントアシスタントボットを作成するには

1. AWS Management Console にサインインし、Amazon Lex コンソール (https://console.aws.amazon.com/lex/) を開きます。
2. ナビゲーションペインで、[ボット] を選択します。
3. [Create] (作成) を選択します。
4. カスタムボットを選択して、ボットの設定を行います。
    a. [ボット名] - ボットには、**AgentAssistBot** などの目的を示す名前を付けます。
    b. [Output voice] (音声出力) - [None] (なし) を選択します。
    c. [Session timeout] (セッションタイムアウト) -「**5**」と入力します。
    d. [COPPA] - [No] (いいえ) を選択します。
5. [Create] (作成) を選択します。Amazon Lex は、ボットを作成した後、[Editor] (エディター) タブを表示します。

次のステップ

ステップ 3: カスタムインテントと組み込みインテントを追加する

ステップ 3: カスタムインテントと組み込みインテントを追加する

インテントは、コールセンターエージェントがボットに実行させたいアクションを表します。この場合、エージェントは、顧客との会話中にボットから回答や役立つリソースを提案してもらうことを想定しています。

Amazon Lex には、カスタムインテントと組み込みインテントの 2 種類のインテントがあります。AMAZON.KendraSearchIntent は組み込みインテントです。ボットは AMAZON.KendraSearchIntent インデックスをクエリし、Amazon Kendra によって提案されたレスポンスを表示するためのインテントです。

この例のボットはカスタムインテントを必要としません。ボットをビルドするには、1 つ以上のカスタムインテントと 1 つ以上のサンプル発話を作成する必要があります。このインテントは、

エージェントアシスタントボットを構築するためにのみ必要です。これは他の機能を実行しません。インテント用の発話は、顧客が質問する可能性のある質問に答えてはいけません。これにより、顧客からの問い合わせには AMAZON.KendraSearchIntent が呼ばれます。詳細については、「AMAZON.KendraSearchIntent」を参照してください。

必要なカスタムインテントを作成するには

1. [ボットの開始方法] ページで、[インテントの作成] を選択します。
2. [インテントの追加] で、[インテントの作成] を選択します。
3. [Create intent] (インテントの作成) ダイアログボックスで、インテントに「**RequiredIntent**」などの名前を入力します。
4. サンプルの発話には、**Required utterance** のような説明的な発話を入力します。
5. [インテントの保存] を選択します。

AMAZON.KendraSearchIntent インテントとレスポンスメッセージを追加するには

1. ナビゲーションペインで、[Intents] (インテント) の横のプラスサイン (+) を選択します。
2. [Search existing intents] (既存のインテントを検索する) を選択します。
3. [Search intents] (インテントの検索) ボックスに **AMAZON.KendraSearchIntent** と入力し、リストからそのインテントを選択します。
4. インテントに「**AgentAssistSearchIntent**」など機能を説明する名前を付けて、[Add] (追加) を選択します。
5. インテントエディタで、[Amazon Kendra クエリ] を選択してクエリオプションを開きます。
6. インテントで検索するインデックスを選択します。
7. [Response] (レスポンス) セクションで、以下の 3 つのメッセージを、メッセージグループに追加します。

```
I found an answer for the customer query: ((x-amz-lex:kendra-search-response-
question_answer-question-1)) and the answer is ((x-amz-lex:kendra-search-response-
question_answer-answer-1)).
I found an excerpt from a helpful document: ((x-amz-lex:kendra-search-response-
document-1)).
I think this answer will help the customer: ((x-amz-lex:kendra-search-response-
answer-1)).
```

8. [インテントの保存] を選択します。

9. ボットを構築するには、[Build] (ビルド) を選択します。

次のステップ

ステップ 4: Amazon Cognito をセットアップする

ステップ 4: Amazon Cognito をセットアップする

ウェブアプリケーションのアクセス許可とユーザーを管理するには、Amazon Cognito をセットアップする必要があります。Amazon Cognito は、ウェブアプリケーションが安全で、アクセス制御があることを確認します。Amazon Cognito は ID プールを使い、AWS の他のサービスへのアクセス権をユーザーに付与する AWS 認証情報を提供します。このチュートリアルでは、Amazon Lex へのアクセスを提供します。

ID プールを作成する際、Amazon Cognito は認証済みおよび未認証のユーザーのための AWS Identity and Access Management (IAM) ロールを提供します。IAM ロールを変更するには、Amazon Lex へのアクセスを許可するポリシーを追加します。

Amazon Cognito をセットアップするには

1. AWS Management Console にサインインして、Amazon Cognito コンソール (https://console.aws.amazon.com/cognito/) を開きます。
2. [Manage Identity Pools (ID プールの管理)] を選択します。
3. [Create new identity pool] を選択します。
4. ID プールを設定します。
   a. ID プール名 — **`BotPool`** など、プールの目的を示す名前を入力します。
   b. [Unauthenticated identities] (認証されていない ID) セクションで、[Enable access to unauthenticated identities] (認証されていない ID に対してアクセスを有効にする) を選択します。
5. [Create Pool] (プールの作成) を選択します。
6. [Identify the IAM roles to use with your new identity pool] (新しい ID プールで使用する IAM ロールを特定する) ページで、[詳細の表示] (View Details) を選択します。
7. IAM ロール名を記録します。後で変更します。
8. [Allow] (許可) を選択します。
9. [Getting Started with Amazon Cognito] (Amazon Cognito の使用開始方法) ページの [Platform] (プラットフォーム) で、[JavaScript] を選択します。

10. [Get AWS 認証情報の取得] セクションで、[アイデンティティプールの ID] を見つけて記録します。
11. Amazon Lex へのアクセスを許可するには、認証済みおよび認証されていない IAM ロールを変更します。

    a. AWS Management Console にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。

    b. ナビゲーションペインの [Access Management] (アクセス管理) で、[Roles] (ロール) を選択します。

    c. 検索ボックスに、認証された IAM ロールの名前を入力し、その横のチェックボックスを選択します。

        i. [Attach policies] (ポリシーの添付) を選択します。

        ii. 検索ボックスで **AmazonLexRunBotsOnly** を入力し、その横のチェックボックスを選択します。

        iii. [Attach policies] (ポリシーの添付) を選択します。

    d. 検索ボックスに、認証されていないIAM ロールの名前を入力し、その横のチェックボックスをオンにします。

        i. [Attach policies] (ポリシーの添付) を選択します。

        ii. 検索ボックスで **AmazonLexRunBotsOnly** を入力し、その横のチェックボックスを選択します。

        iii. [Attach policies] (ポリシーの添付) を選択します。

次のステップ

[ステップ 5: ボットをウェブアプリケーションとしてデプロイする]

ステップ 5: ボットをウェブアプリケーションとしてデプロイする

ボットをウェブアプリケーションとしてデプロイするには

1. https://github.com/awsdocs/amazon-lex-developer-guide/blob/master/example_apps/agent_assistance_bot/ のリポジトリをコンピュータにダウンロードしてください。
2. ダウンロードしたリポジトリに移動し、エディタで「index.html」ファイルを開きます。
3. 以下の変更を加えます。

a. `AWS.config.credentials` セクションでは、リージョン名と アイデンティティプール ID を入力します。

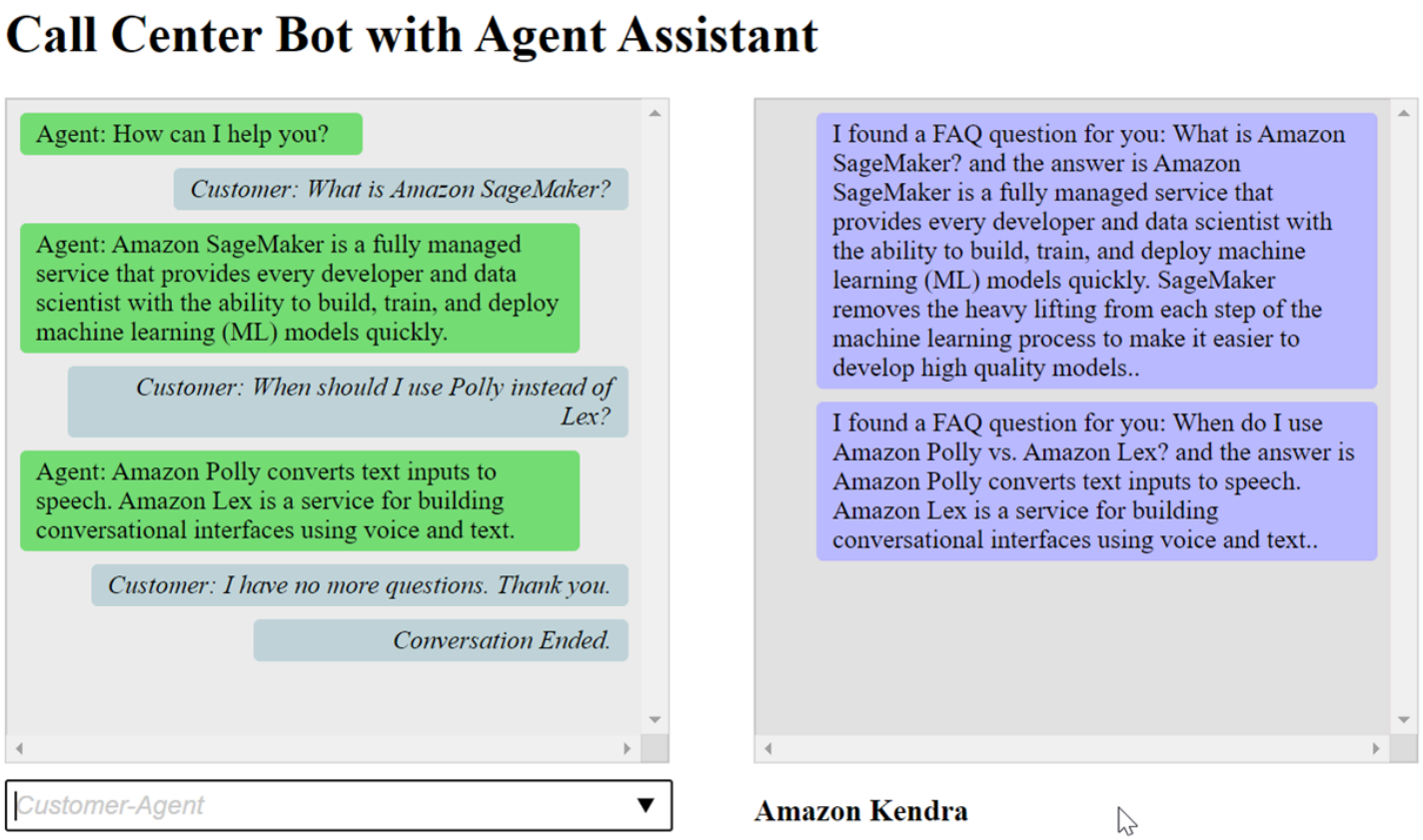
b. `Amazon Lex runtime parameters` セクションで、ボット名を入力します。

c. ファイルを保存します。

ステップ 6: ボットを使用する

デモの目的で、顧客およびエージェントとしてボットにインプットを提供します。両者を区別するために、顧客からの質問には「Customer:」で始まり、エージェントから提供された回答は「Agent:」で始まります。提案された入力候補のメニューから選ぶことができます。

`index.html` を開いてウェブアプリケーションを実行すると、次の画像のようにボットとの対話が行われます。



「index.html 」ファイル内の `pushChat()` 機能について説明します。

```
            var endConversationStatement = "Customer: I have no more questions. Thank
you."
            // If the agent has to send a message, start the message with 'Agent'
            var inputText = document.getElementById('input');
            if (inputText && inputText.value && inputText.value.trim().length > 0 &&
inputText.value[0]=='Agent') {
                showMessage(inputText.value, 'agentRequest','conversation');
                inputText.value = "";
            }
            // If the customer has to send a message, start the message with 'Customer'
            if(inputText && inputText.value && inputText.value.trim().length > 0 &&
inputText.value[0]=='Customer') {
                // disable input to show we're sending it
                var input = inputText.value.trim();
                inputText.value = '...';
                inputText.locked = true;
                customerInput = input.substring(2);

                // Send it to the Lex runtime
                var params = {
                    botAlias: '$LATEST',
                    botName: 'KendraTestBot',
                    inputText: customerInput,
                    userId: lexUserId,
                    sessionAttributes: sessionAttributes
                };

                showMessage(input, 'customerRequest', 'conversation');
                if(input== endConversationStatement){
                    showMessage('Conversation
Ended.','conversationEndRequest','conversation');
                }
                lexruntime.postText(params, function(err, data) {
                    if (err) {
                        console.log(err, err.stack);
                        showMessage('Error:  ' + err.message + ' (see console for
details)', 'lexError', 'conversation1')
                    }

                    if (data &&input!=endConversationStatement) {
                        // capture the sessionAttributes for the next cycle
                        sessionAttributes = data.sessionAttributes;

                            showMessage(data, 'lexResponse', 'conversation1');
```

```
                }
                // re-enable input
                inputText.value = '';
                inputText.locked = false;
            });
        }
        // we always cancel form submission
        return false;
```

顧客として入力すると、Amazon Lex ランタイム API が Amazon Lex に送信します。

showMessage(daText, senderRequest, displayWindow) 機能は、エージェントと顧客との会話をチャットウィンドウに表示します。Amazon Kendra によって提案された回答は、隣接するウィンドウに表示されます。顧客が「**“I have no more questions. Thank you.”**」と言った時点で会話は終了します

注意: Amazon Kendra のインデックスは、ご使用にならないときは削除してください。

ボットの移行

Amazon Lex V2 API は更新された情報アーキテクチャを使用して、リソースのバージョニングを簡素化し、ボットで複数の言語をサポートできるようにします。詳細については、「Amazon Lex V2 デベロッパーガイド」の「Migration guide」(移行ガイド) を参照してください。

これらの新機能を使用するには、ボットを移行する必要があります。ボットを移行すると、Amazon Lex では以下が提供されます。

- 移行は、カスタムインテントとスロットタイプを Amazon Lex V2 ボットにコピーします。
- 同じ Amazon Lex V2 ボットを使用して、複数の言語を追加できます。Amazon Lex V1 では、言語ごとに個別のボットを作成します。それぞれが異なる言語を使用している複数の Amazon Lex V1 ボットを、1 つの Amazon Lex V2 ボットに移行することができます。
- Amazon Lex は、Amazon Lex V1 の組み込みスロットタイプとインテントを Amazon Lex V2 の組み込みスロットタイプとインテントにマッピングします。組み込みが移行できない場合、Amazon Lex は次に何をすべきかを示すメッセージを返します。

移行プロセスでは、次のものは移行されません。

- エイリアス
- Amazon Kendra インデックス
- AWS Lambda 関数
- 会話ログの設定
- Slack などのメッセージングチャンネル
- タグ

ボットを移行するには、ユーザーまたはロールが、Amazon Lex および Amazon Lex V2 の両方の API オペレーションに対する IAM 権限を持っている必要があります。必要なアクセス許可については、「ユーザーが Amazon Lex V2 API にボットを移行できるようにする」を参照してください。

ボットの移行 (コンソール)

Amazon Lex V1 コンソールを使用して、ボットの構造を Amazon Lex V2 ボットに移行します。

コンソールを使用してボットを Amazon Lex V2 API に移行するには

1. AWS Management Console にサインインし、Amazon Lex コンソール (https://console.aws.amazon.com/lex/) を開きます。
2. 左のメニューで、[Migration tool] (移行ツール) を選択します。
3. ボットのリストから、移行するボットを選択し、[Migrate] (移行) を選択します。
4. 移行するボットのバージョンを選択し、移行先のボットの名前を入力します。既存の Amazon Lex V2 ボットの名前を入力すると、Amazon Lex V1 ボットが詳細に表示される言語に移行され、その言語のドラフトバージョンが上書きされます。
5. [Next] (次へ) を選択します。
6. Amazon Lex が Amazon Lex V2 API バージョンのボットの実行に使用する IAM ロールを選択します。ボットの実行に必要な最低限の権限を持つ新しいロールを作成するか、既存の IAM ロールを選択することができます。
7. [Next] (次へ) をクリックします。
8. 移行の設定を確認します。問題ない場合は、[Start migration] (移行開始) をクリックします。

移行処理を開始すると、移行ツールのスタートページに戻ります。移行の進捗状況は、履歴テーブルで確認できます。移行が完了すると、移行状況の列に「Complete」と表示されます。

Amazon Lexは、Amazon Lex V2 API の `StartImport` オペレーションを使用して、移行されたボットをインポートします。Amazon Lex V2 コンソールのインポート履歴テーブルに、移行ごとのエントリが表示されます。

移行中、Amazon Lex はボット内で移行できないリソースを見つける場合があります。移行できないリソースには、エラーまたは警告メッセージが表示されます。各メッセージには、問題を解決する方法を説明したドキュメントへのリンクが含まれています。

Lambda 関数の移行

Amazon Lex V2 は、ボット用に Lambda 関数が定義される方法を変更します。ボット内の言語ごとに、エイリアスで Lambda 関数を 1 つだけ許可します。Lambda 関数の移行の詳細については、「Amazon Lex V1 から Amazon Lex V2 への Lambda 関数の移行」を参照してください。

移行メッセージ

移行中、Amazon Lex は、組み込みスロットタイプなど、同等の Amazon Lex V2 リソースに移行できないリソースを見つけることがあります。この問題が発生すると、Amazon Lex は何が起こったかを説明する移行メッセージを返し、移行問題を解決する方法を示すドキュメントへのリンクを提供します。以下のセクションでは、ボットを移行する際に発生する可能性のある問題と、その問題を解決する方法について説明します。

トピック

組み込みインテント

Amazon Lex V2 でサポートされていない組み込みインテントを使用すると、インテントは Amazon Lex V2 ボットのカスタムインテントにマッピングされます。カスタムインテントには発話は含まれません。インテントを継続して使用するには、サンプルの発話を追加します。

組み込みスロットタイプ

Amazon Lex V2 でサポートされていないスロットタイプを使用している移行済みのスロットには、スロットタイプの値が指定されません。このスロットを使用するには:

- カスタムスロットタイプを作成する
- スロットタイプに想定されるスロットタイプの値を追加する
- 新しいカスタムスロットタイプを使用するようにスロットを更新する

会話ログ

移行では、Amazon Lex V2 ボットの会話ログ設定は更新されません。

会話ログを設定するには

1. https://console.aws.amazon.com/lexv2 から Amazon Lex V2 コンソールを開きます。
2. エイリアスのリストで、会話ログを設定するエイリアスを選択します。
3. 左側のメニューから、[Aliases] (エイリアス) を選択し、リストからエイリアスを選択します。
4. [Conversation logs] (会話ログ) セクションで [Manage conversation logs] (会話ログを管理する) を選択して、ボットエイリアスの会話ログを設定します。

メッセージグループ

Amazon Lex V2 では、メッセージグループごとに 1 つのメッセージと 2 つの代替メッセージのみをサポートしています。Amazon Lex V1 ボットでメッセージグループごとに 3 つ以上のメッセージがある場合は、最初の 3 つのメッセージのみが移行されます。メッセージグループでより多くのメッセージを使用するには、Lambda 関数を使用してさまざまなメッセージを出力します。

プロンプトおよびフレーズ

Amazon Lex V2 は、フォローアップ、説明、ハングアップのプロンプトに異なるメカニズムを使用しています。

フォローアッププロンプトについては、コンテキストキャリーオーバーを使用して、達成後に別のインテントに切り替えることができます。

例えば、レンタカーを予約するインテントがあり book_car_fulfilled という出力コンテキストを返すよう設定されているとします。インテントが達成されると、Amazon Lex は出力コンテキスト変数 book_car_fulfilled を設定します。book_car_fulfilled はアクティブなコンテキストであるため、ユーザーの発話がそのインテントを引き出す試みとして認識される限り、book_car_fulfilled コンテキストを入力コンテキストとして設定したインテントが認識対象として考慮されます。領収書のメール送信や予約内容の変更など、予約後にしか意味をなさないインテントに使用することができます。

Amazon Lex V2 では、明確化プロンプトとハングアップフレーズ (中断ステートメント) はサポートされていません。Amazon Lex V2 ボットには、インテントがマッチしない場合に起動されるデフォルトのフォールバックインテントが含まれています。再試行を含む明確化プロンプトを送信するには、Lambda 関数を設定し、フォールバックインテントでダイアログコードフックを有効にします。Lambda 関数は、応答として明確化プロンプトを出力し、セッション属性の再試行値を出力できます。再試行値が最大再試行回数を超える場合は、ハングアップフレーズを出力してカンバセーションを閉じることができます。

Amazon Lex V1 の他の機能

移行ツールは、Amazon Lex V1 ボットとその基礎となるインテント、スロットタイプ、スロットの移行のみをサポートしています。その他の機能については、「Amazon Lex V2 ドキュメント」の次のトピックを参照してください。

- ボットのエイリアス: エイリアス
- ボットチャンネル: メッセージングプラットフォームでの Amazon Lex V2 ボットのデプロイ
- 会話ログの設定: 会話ログでのモニタリング
- Amazon Kendra インデックス: AMAZON.KendraSearchIntent
- Lambda 関数: AWS Lambda 関数の使用
- タグ: リソースのタグ付け

Amazon Lex V1 から Amazon Lex V2 への Lambda 関数の移行

Amazon Lex V2 では、ボット内の言語ごとに Lambda 関数を 1 つだけ許可します。Lambda 関数とその設定は、実行時に使用するボットエイリアスに対して構成されます。

インテントでダイアログとフルフィルメントコードのフックが有効になっている場合、その言語のすべてのインテントに対して Lambda 関数が呼び出されます。

Amazon Lex V2 Lambda 関数は、Amazon Lex V1 とは異なる入出力メッセージ形式を持っています。Lambda 関数の入力形式の違いは次のとおりです。

- Amazon Lex V2 では、`currentIntent` と `alternativeIntents` の構造が `interpretations` の構造に置き換えられます。各解釈には、インテント、インテントに対する NLU 信頼度スコア、およびオプションのセンチメント分析が含まれます。
- Amazon Lex V2 では、Amazon Lex V1の `activeContexts`、`sessionAttributes` が統一された `sessionState` 構造に移行されます。この構造は、発信元のリクエスト ID など、会話の現在の状態に関する情報を提供します。
- Amazon Lex V2 は `recentIntentSummaryView` を返しません。代わりに `sessionState` 構造の情報を使用してください。
- Amazon Lex V2の入力では、`bot` 属性に `botId` と `localeId` を用意しています。
- 入力構造には、入力のタイプ (テキスト、スピーチ、DTMF) に関する情報を提供する `inputMode` 属性が含まれています。

Lambda 関数の出力形式の違いは次のとおりです。

- Amazon Lex V1 の activeContexts および sessionAttributes 構造は、Amazon Lex V2 の sessionState 構造に置き換えられています。
- recentIntentSummaryView は出力には含まれません。
- Amazon Lex V1 dialogAction 構造は、sessionState 構造の一部である dialogAction と、dialogAction.type が ElicitIntent の時に必要となる messages の 2 つの構造に分かれています。Amazon Lex は、この構造からユーザーに表示するメッセージを選択します。

Amazon Lex V2 API でボットを構築する場合、インテントごとに Lambda 関数を用意するのではなく、言語別にボットのエイリアスごとに 1 つの Lambda 関数しか用意しません。別の機能を引き続き使用する場合は、インテントごとに個別の機能を有効にするルーター機能を作成できます。次に、アプリケーションで使用または変更できるルーター機能を示します。

```
import os
import json
import boto3

# reuse client connection as global
client = boto3.client('lambda')

def router(event):
    intent_name = event['sessionState']['intent']['name']
    fn_name = os.environ.get(intent_name)
    print(f"Intent: {intent_name} -> Lambda: {fn_name}")
    if (fn_name):
        # invoke lambda and return result
        invoke_response = client.invoke(FunctionName=fn_name, Payload =
 json.dumps(event))
        print(invoke_response)
        payload = json.load(invoke_response['Payload'])
        return payload
    raise Exception('No environment variable for intent: ' + intent_name)

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

更新されたフィールドのリスト。

次の表は、Amazon Lex V2 Lambda リクエストとレスポンスの更新されたフィールドに関する詳細情報を示しています。これらのテーブルを使用して、バージョン間のフィールドをマッピングできます。

リクエスト

以下のフィールドが Lambda 関数リクエスト形式で更新されました。

アクティブコンテキスト

これで、`activeContexts` 構造は `sessionState` 構造の一部になりました。

V1 構造	V2 構造
activeContexts	sessionState.activeContexts
activeContexts[*].timeToLive	sessionState.activeContexts[*].timeToLive
activeContexts[*].timeToLive.timeToLiveInSeconds	sessionState.activeContexts[*].timeToLive.timeToLiveInSeconds
activeContexts[*].timeToLive.turnsToLive	sessionState.activeContexts[*].timeToLive.turnsToLive
activeContexts[*].name	sessionState.activeContexts[*].name
activeContexts[*].parameters	sessionState.activeContexts[*].contextAttributes

代替インテント

インデックス 1 から N までの解釈リストには、Amazon Lex V2 によって予測された代替インテントのリストとその信頼スコアが含まれています。`recentIntentSummaryView` は Amazon Lex V2 のリクエスト構造から削除されました。`recentIntentSummaryView` から詳細を確認するには、GetSession オペレーションを使用します。

V1 構造	V2 構造
alternativeIntents	interpretations[1:*]
recentIntentSummaryView	該当なし

ボット

Amazon Lex V2 では、ボットとエイリアスには識別子があります。ボット ID はコードフック入力の一部です。エイリアス ID は含まれますが、エイリアス名は含まれません。Amazon Lex V2 は同じボットに対して複数のロケールをサポートしているため、ロケール ID が含まれます。

V1 構造	V2 構造
ボット	ボット
bot.name	bot.name
該当なし	bot.id
bot.alias	該当なし
該当なし	bot.aliasId
bot.version	bot.version
該当なし	bot.localeId

現在のインテント

`sessionState.intent` 構造には、アクティブインテントの詳細が含まれています。Amazon Lex V2 は、代替インテントを含め、`interpretations` 構造内のすべてのインテントのリストも返します。解釈リストの最初の要素は常に `sessionState.intent` と同じです。

V1 構造	V2 構造
currentIntent	sessionState.intent OR interpretations[0].intent

V1 構造	V2 構造
currentIntent.name	sessionState.intent.name OR interpretations[0].intent.name
currentIntent.nluConfidenceScore	interpretations[0].nluConfidence.score

ダイアログアクション

これで、`confirmationStatus` フィールドは `sessionState` 構造の一部になりました。

V1 構造	V2 構造
currentIntent.confirmationStatus	sessionState.intent.confirmationState OR interpretations[0].intent.confirmationState
該当なし	sessionState.intent.state OR interpretations[*].intent.state

Amazon Kendra

`kendraResponse` フィールドは `sessionState` および `interpretations` 構造の一部になりました。

V1 構造	V2 構造
kendraResponse	sessionState.intent.kendraResponse OR interpretations[0].intent.kendraResponse

感情

`sentimentResponse` 構造は新しい `interpretations` 構造に移動されました。

V1 構造	V2 構造
sentimentResponse	interpretations[0].sentimentResponse

V1 構造	V2 構造
sentimentResponse.sentimentLabel	interpretations[0].sentimentResponse.sentiment
sentimentResponse.sentimentScore	interpretations[0].sentimentResponse.sentimentScore

スロット

Amazon Lex V2 は、解決された値、解釈された値、およびユーザーが発言内容の元の値を含む単一の `slots` オブジェクトを `sessionState.intent` 構造内に提供します。Amazon Lex V2 では、`slotShape`を `List` として設定し、`values` リストを設定することで、複数値のスロットもサポートしています。`value` フィールドでは単一値スロットがサポートされており、その形状は `Scalar` になるとみなされています。

V1 構造	V2 構造
currentIntent.slots	sessionState.intent.slots OR interpretations[0].intent.slots
currentIntent.slots[*].value	sessionState.intent.slots[*].value.interpretedValue OR interpretations[0].intent.slots[*].value.interpretedValue
該当なし	sessionState.intent.slots[*].value.shape OR interpretations[0].intent.slots[*].shape
該当なし	sessionState.intent.slots[*].values OR interpretations[0].intent.slots[*].values
currentIntent.slotDetails	sessionState.intent.slots OR interpretations[0].intent.slots
currentIntent.slotDetails[*].resolutions	sessionState.intent.slots[*].resolvedValues OR interpretations[0].intent.slots[*].resolvedValues
currentIntent.slotDetails[*].originalValue	sessionState.intent.slots[*].originalValue OR interpretations[0].intent.slots[*].originalValue

その他

Amazon Lex V2 `sessionId` フィールドは、Amazon Lex V1 の `userId` フィールドと同じです。Amazon Lex V2 では、発信者の `inputMode` (テキスト、DTMF、または音声) も送信します。

V1 構造	V2 構造
userId	sessionId
inputTranscript	inputTranscript
invocationSource	invocationSource
outputDialogMode	responseContentType
messageVersion	messageVersion
sessionAttributes	sessionState.sessionAttributes
requestAttributes	requestAttributes
該当なし	inputMode
該当なし	originatingRequestId

レスポンス

以下のフィールドが Lambda 関数レスポンスメッセージ形式で更新されました。

アクティブコンテキスト

`sessionState` 構造に移動した `activeContexts` 構造。

V1 構造	V2 構造
activeContexts	sessionState.activeContexts
activeContexts[*].timeToLive	sessionState.activeContexts[*].timeToLive

V1 構造	V2 構造
activeContexts[*].timeToLive.timeToLiveInSeconds	sessionState.activeContexts[*].timeToLive.timeToLiveInSeconds
activeContexts[*].timeToLive.turnsToLive	sessionState.activeContexts[*].timeToLive.turnsToLive
activeContexts[*].name	sessionState.activeContexts[*].name
activeContexts[*].parameters	sessionState.activeContexts[*].contextAttributes

ダイアログアクション

`sessionState` 構造に移動した `dialogAction` 構造。1 つのダイアログアクションで複数のメッセージを指定できるようになり、`genericAttachments` 構造は `imageResponseCard` 構造になりました。

V1 構造	V2 構造
dialogAction	sessionState.dialogAction
dialogAction.type	sessionState.dialogAction.type
dialogAction.slotToElicit	sessionState.intent.dialogAction.slotToElicit
dialogAction.type.fulfillmentState	sessionState.intent.state
dialogAction.message	メッセージ
dialogAction.message.contentType	messages[*].contentType
dialogAction.message.content	messages[*].content
dialogAction.responseCard	messages[*].imageResponseCard
dialogAction.responseCard.version	該当なし
dialogAction.responseCard.contentType	messages[*].contentType

V1 構造	V2 構造
dialogAction.responseCard.genericAttachments	該当なし
dialogAction.responseCard.genericAttachments[*].title	messages[*].imageResponseCard.title
dialogAction.responseCard.genericAttachments[*].subTitle	messages[*].imageResponseCard.subtitle
dialogAction.responseCard.genericAttachments[*].imageUrl	messages[*].imageResponseCard.imageUrl
dialogAction.responseCard.genericAttachments[*].buttons	messages[*].imageResponseCard.buttons
dialogAction.responseCard.genericAttachments[*].buttons[*].value	messages[*].imageResponseCard.buttons[*].value
dialogAction.responseCard.genericAttachments[*].buttons[*].text	messages[*].imageResponseCard.buttons[*].text
dialogAction.kendraQueryRequestPayload	dialogAction.kendraQueryRequestPayload
dialogAction.kendraQueryFilterString	dialogAction.kendraQueryFilterString

インテントとスロット

`dialogAction` 構造の一部であったインテントフィールドとスロットフィールドが `sessionState` 構造の一部になりました。

V1 構造	V2 構造
dialogAction.intentName	sessionState.intent.name
dialogAction.slots	sessionState.intent.slots
dialogAction.slots[*].key	sessionState.intent.slots[*].key

V1 構造	V2 構造
dialogAction.slots[*].value	sessionState.intent.slots[*].value.interpreted Value
該当なし	sessionState.intent.slots[*].value.shape
該当なし	sessionState.intent.slots[*].values

その他

これで、`sessionAttributes` 構造は `sessionState` 構造の一部になりました。`recentIntentSummaryReview` 構造は削除されました。

V1 構造	V2 構造
sessionAttributes	sessionState.sessionAttributes
recentIntentSummaryView	該当なし

Amazon Lex のセキュリティ

のクラウドセキュリティが最優先事項 AWS です。 AWS のお客様は、セキュリティを最も重視する組織の要件を満たすように構築されたデータセンターとネットワークアーキテクチャから利点を得られます。

セキュリティは、 AWS とユーザー間で共有される責任です。責任共有モデルでは、これをクラウドのセキュリティおよびクラウド内のセキュリティとして説明しています。

- クラウドのセキュリティ – AWS は、 AWS クラウドでサービスを実行する AWS インフラストラクチャを保護する責任を担います。また、 は、ユーザーが安全に使用できるサービス AWS も提供します。セキュリティの有効性は、AWS コンプライアンスプログラムの一環として、サードパーティーの審査機関によって定期的にテストおよび検証されています。Amazon Lex に適用されるコンプライアンスプログラムについては、「コンプライアンスプログラムの対象範囲となる AWS サービス」 を参照してください。
- クラウド内のセキュリティ – お客様の責任は、使用する AWS サービスによって決まります。また、お客様は、お客様のデータの機密性、組織の要件、および適用可能な法律および規制などの他の要因についても責任を担います。

このドキュメントは、Amazon Lex を使用する際に適用される責任共有モデルについての理解の助けとなることを目的としています。以下のトピックでは、セキュリティとコンプライアンスの目的を満たすように Amazon Lex を設定する方法について説明します。また、Amazon Lex リソースのモニタリングや保護に役立つ他の AWS サービスの用法についても説明します。

トピック

- Amazon Lex のデータ保護
- Amazon Lex のための Identity and Access Management
- Amazon Lex でのモニタリング
- Amazon Lex のコンプライアンス検証
- Amazon Lex の耐障害性
- Amazon Lex のインフラストラクチャセキュリティ

Amazon Lex のデータ保護

Amazon Lex はトラブルシューティングのためにお客様のコンテンツを収集し、サービスの改善に役立てます。お客様のコンテンツはデフォルトで保護されます。Amazon Lex API を使用すると、個別のお客様のコンテンツを削除することができます。

Amazon Lex は 4 種類のコンテンツを保存します。

- ボットを構築してトレーニングするために使用されるサンプル発話
- ボットとやり取りするユーザーからのカスタマー発話
- ユーザーがボットとやり取りする期間についてのアプリケーション固有の情報を提供するセッション属性
- ボットへの 1 つのリクエストに適用される情報を含むリクエスト属性

児童が使用するために設計されたすべての Amazon Lex ボットは、児童オンラインプライバシー保護法 (COPPA) の適用対象となります。コンソールあるいは Amazon Lex API を使用して `childDirected` フィールドを `true` に設定することで、このボットが COPPA の適用対象であることを Amazon Lex に指示します。`childDirected` フィールドが `true` に設定されている場合、ユーザー発話は一切保存されません。

トピック

- 保管時の暗号化
- 転送時の暗号化
- キーの管理

保管時の暗号化

Amazon Lex は保存するユーザー発話を暗号化します。

トピック

- サンプル発話
- カスタマー発話
- セッション属性
- リクエスト属性

サンプル発話

ボットを開発するときに、インテントおよびスロットごとにサンプル発話を提供できます。また、スロットにカスタムの値およびシノニムを指定することもできます。この情報はボットを構築してユーザーエクスペリエンスを作成するために使用されます。

カスタマー発話

`childDirected` フィールドが `true` に設定されている場合を除き、Amazon Lex はユーザーがボットに送信する発話を暗号化します。

`childDirected` フィールドが `true` に設定されている場合、ユーザー発話は一切保存されません。

`childDirected` フィールドが `false` (デフォルト) に設定されている場合、ユーザー発話は暗号化され、GetUtterancesView オペレーションで使用するために 15 日間保存されます。特定のユーザーの保存された発話を削除するには、DeleteUtterances オペレーションを使用します。

ボットが音声入力を受け付けると、その入力内容は無期限に保存されます。Amazon Lex は、ユーザーの入力に応答するボットの性能を向上させるために使用します。

DeleteUtterances オペレーションを使用して、特定のユーザーの保存された発話を削除します。

セッション属性

セッション属性には、セッション中に Amazon Lex とクライアントアプリケーションの間でやり取りされるアプリケーション固有の情報が含まれます。Amazon Lex は、ボット用に設定されたすべての AWS Lambda 関数にセッション属性を渡します。Lambda 関数でセッション属性が追加または更新されると、Amazon Lex からクライアントアプリケーションに新しい情報が返されます。

セッション属性は、セッションの期間中の暗号化された保存において保持されます。最後のユーザー発話から最低 1 分および最大 24 時間にセッションがアクティブのままになるように設定できます。デフォルトのセッション期間は 5 分間です。

リクエスト属性

リクエスト属性はリクエスト固有の情報を含み、現在のリクエストにのみ適用されます。クライアントアプリケーションはリクエスト属性を使用して、実行時に Amazon Lex に情報を送信します。

セッション全体を通して保持する必要がない情報は、リクエスト属性を使用して渡します。リクエスト属性はリクエスト間で保持されないため、保存されません。

転送時の暗号化

Amazon Lex は HTTPS プロトコルを使用して、クライアントアプリケーションと通信します。HTTPS および AWS 署名を使用して、Amazon Polly やアプリケーションに代わって他の サービスと通信 AWS Lambda します。

キーの管理

Amazon Lex は、内部キーを使用した不正使用からコンテンツを保護します。

Amazon Lex のための Identity and Access Management

AWS Identity and Access Management (IAM) は、管理者が AWS リソースへのアクセスを安全に制御 AWS のサービス するのに役立つ です。IAM 管理者は、誰を認証 (サインイン) し、誰に Amazon Lex リソースの使用を承認する (アクセス許可を付与する) かを制御します。IAM は、追加料金なしで AWS のサービス 使用できる です。

トピック

対象者

AWS Identity and Access Management (IAM) の使用方法は、Amazon Lex で行う作業によって異なります。

サービスユーザー – ジョブを実行するために Amazon Lex サービスを使用する場合は、管理者から必要なアクセス許可と認証情報が与えられます。さらに多くの Amazon Lex 機能を使用して作業を行う場合は、追加のアクセス許可が必要になることがあります。アクセスの管理方法を理解しておく

と、管理者に適切な許可をリクエストするうえで役立ちます。Amazon Lex の機能にアクセスできない場合は、「Amazon Lex アイデンティティとアクセスのトラブルシューティング」を参照してください。

サービス管理者 – 社内の Amazon Lex リソースを担当している場合は、通常、Amazon Lex へのフルアクセスがあります。サービスのユーザーがどの Amazon Lex 機能やリソースにアクセスするかを決めるのは管理者の仕事です。その後、IAM 管理者にリクエストを送信して、サービスユーザーの権限を変更する必要があります。このページの情報を点検して、IAM の基本概念を理解してください。会社で Amazon Lex と IAM を併用する方法の詳細については、「Amazon Lex で IAM を使用する方法」を参照してください。

IAM 管理者 – 管理者は、Amazon Lex へのアクセスを管理するポリシーの作成方法の詳細について確認する場合があります。IAM で使用できる Amazon Lex アイデンティティベースのポリシーの例を表示するには、「Amazon Lex のアイデンティティベースのポリシー例」を参照してください。

アイデンティティを使用した認証

認証とは、ID 認証情報 AWS を使用して にサインインする方法です。として、IAM ユーザーとして AWS アカウントのルートユーザー、または IAM ロールを引き受けて認証 (にサインイン AWS) される必要があります。

ID ソースを介して提供された認証情報を使用して、フェデレーティッド ID AWS として にサインインできます。 AWS IAM Identity Center （ IAM Identity Center) ユーザー、会社のシングルサインオン認証、Google または Facebook の認証情報は、フェデレーティッド ID の例です。フェデレーティッド ID としてサインインする場合、IAM ロールを使用して、前もって管理者により ID フェデレーションが設定されています。フェデレーション AWS を使用して にアクセスすると、間接的にロールを引き受けることになります。

ユーザーのタイプに応じて、 AWS Management Console または AWS アクセスポータルにサインインできます。へのサインインの詳細については AWS、「 ユーザーガイド」の「 へのサインイン AWS アカウント方法AWS サインイン 」を参照してください。

AWS プログラムで にアクセスする場合、 は Software Development Kit (SDK) とコマンドラインインターフェイス (CLI) AWS を提供し、認証情報を使用してリクエストに暗号で署名します。 AWS ツールを使用しない場合は、リクエストに自分で署名する必要があります。推奨される方法を使用してリクエストに自分で署名する方法の詳細については、IAM ユーザーガイドの API AWS リクエストの署名を参照してください。

使用する認証方法を問わず、追加セキュリティ情報の提供をリクエストされる場合もあります。例えば、 AWS では、多要素認証 (MFA) を使用してアカウントのセキュリティを向上させることをお勧

めします。詳細については、「AWS IAM Identity Center ユーザーガイド」の「多要素認証」および「IAM ユーザーガイド」の「AWSでの多要素認証 (MFA) の使用」を参照してください。

AWS アカウント ルートユーザー

を作成するときは AWS アカウント、アカウント内のすべての AWS のサービス およびリソースへの完全なアクセス権を持つ 1 つのサインインアイデンティティから始めます。この ID は AWS アカウント ルートユーザーと呼ばれ、アカウントの作成に使用した E メールアドレスとパスワードでサインインすることでアクセスできます。日常的なタスクには、ルートユーザーを使用しないことを強くお勧めします。ルートユーザーの認証情報は保護し、ルートユーザーでしか実行できないタスクを実行するときに使用します。ルートユーザーとしてサインインする必要があるタスクの完全なリストについては、「IAM ユーザーガイド」の「ルートユーザー認証情報が必要なタスク」を参照してください。

フェデレーティッドアイデンティティ

ベストプラクティスとして、管理者アクセスを必要とするユーザーを含む人間のユーザーに、一時的な認証情報を使用して にアクセスするための ID プロバイダーとのフェデレーションの使用を要求 AWS のサービス します。

フェデレーティッド ID は、エンタープライズユーザーディレクトリ、ウェブ ID プロバイダー、AWS Directory Service、アイデンティティセンターディレクトリのユーザー、または ID ソースを通じて提供された認証情報 AWS のサービス を使用して にアクセスするユーザーです。フェデレーティッド ID が にアクセスすると AWS アカウント、ロールを引き受け、ロールは一時的な認証情報を提供します。

アクセスを一元管理する場合は、 AWS IAM Identity Centerを使用することをお勧めします。IAM Identity Center でユーザーとグループを作成することも、独自の ID ソース内のユーザーとグループのセットに接続して同期して、すべての AWS アカウント とアプリケーションで使用することもできます。IAM Identity Center の詳細については、「AWS IAM Identity Center ユーザーガイド」の「IAM Identity Center とは」を参照してください。

IAM ユーザーとグループ

IAM ユーザーは、単一のユーザーまたはアプリケーションに対して特定のアクセス許可 AWS アカウント を持つ 内のアイデンティティです。可能であれば、パスワードやアクセスキーなどの長期的な認証情報を保有する IAM ユーザーを作成する代わりに、一時認証情報を使用することをお勧めします。ただし、IAM ユーザーでの長期的な認証情報が必要な特定のユースケースがある場合は、アクセスキーをローテーションすることをお勧めします。詳細については、IAM ユーザーガイド の「長

期的な認証情報を必要とするユースケースのためにアクセスキーを定期的にローテーションする」を参照してください。

IAM グループは、IAM ユーザーの集団を指定するアイデンティティです。グループとしてサインインすることはできません。グループを使用して、複数のユーザーに対して一度に権限を指定できます。多数のユーザーグループがある場合、グループを使用することで権限の管理が容易になります。例えば、IAMAdmins という名前のグループを設定して、そのグループに IAM リソースを管理する許可を与えることができます。

ユーザーは、ロールとは異なります。ユーザーは 1 人の人または 1 つのアプリケーションに一意に関連付けられますが、ロールはそれを必要とする任意の人が引き受けるようになっています。ユーザーには永続的な長期の認証情報がありますが、ロールでは一時的な認証情報が提供されます。詳細については、「IAM ユーザーガイド」の「IAM ユーザー (ロールではなく) の作成が適している場合」を参照してください。

IAM ロール

IAM ロールは、特定のアクセス許可 AWS アカウント を持つ 内のアイデンティティです。これは IAM ユーザーに似ていますが、特定のユーザーには関連付けられていません。ロール を切り替える AWS Management Console ことで、 で IAM ロールを一時的に引き受けることができます。ロールを引き受けるには、 または AWS API AWS CLI オペレーションを呼び出すか、カスタム URL を使用します。ロールを使用する方法の詳細については、「IAM ユーザーガイド」の「IAM ロールの使用」を参照してください。

IAM ロールと一時的な認証情報は、次の状況で役立ちます:

- フェデレーションユーザーアクセス – フェデレーティッド ID に許可を割り当てるには、ロールを作成してそのロールの許可を定義します。フェデレーティッド ID が認証されると、その ID はロールに関連付けられ、ロールで定義されている許可が付与されます。フェデレーションの詳細については、「IAM ユーザーガイド」の「Creating a role for a third-party Identity Provider」(サードパーティーアイデンティティプロバイダー向けロールの作成) を参照してください。IAM Identity Center を使用する場合は、許可セットを設定します。アイデンティティが認証後にアクセスできるものを制御するため、IAM Identity Center は、権限セットを IAM のロールに関連付けます。アクセス許可セットの詳細については、「AWS IAM Identity Center ユーザーガイド」の「アクセス許可セット」を参照してください。
- 一時的な IAM ユーザー権限 - IAM ユーザーまたはロールは、特定のタスクに対して複数の異なる権限を一時的に IAM ロールで引き受けることができます。
- クロスアカウントアクセス - IAM ロールを使用して、自分のアカウントのリソースにアクセスすることを、別のアカウントの人物 (信頼済みプリンシパル) に許可できます。クロスアカウントアク

セス権を付与する主な方法は、ロールを使用することです。ただし、一部の では AWS のサービス、 (ロールをプロキシとして使用する代わりに) ポリシーをリソースに直接アタッチできます。クロスアカウントアクセスのロールとリソースベースのポリシーの違いについては、IAM ユーザーガイドの「IAM でのクロスアカウントリソースアクセス」を参照してください。

- クロスサービスアクセス — 一部の は、他の の機能 AWS のサービス を使用します AWS のサービス。例えば、あるサービスで呼び出しを行うと、通常そのサービスによって Amazon EC2 でアプリケーションが実行されたり、Amazon S3 にオブジェクトが保存されたりします。サービスでは、呼び出し元プリンシパルの許可、サービスロール、またはサービスリンクロールを使用してこれを行う場合があります。
  - 転送アクセスセッション (FAS) — IAM ユーザーまたはロールを使用して でアクションを実行する場合 AWS、ユーザーはプリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがあります。FAS は、 を呼び出すプリンシパルのアクセス許可を AWS のサービス、ダウンストリームサービス AWS のサービス へのリクエストリクエストリクエストと組み合わせて使用します。FAS リクエストは、サービスが他の AWS のサービス またはリソースとのやり取りを完了する必要があるリクエストを受け取った場合にのみ行われます。この場合、両方のアクションを実行するためのアクセス許可が必要です。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。
  - サービスロール - サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除できます。詳細については、「IAM ユーザーガイド」の「AWS のサービスにアクセス許可を委任するロールの作成」を参照してください。
  - サービスにリンクされたロール – サービスにリンクされたロールは、 にリンクされたサービスロールの一種です AWS のサービス。サービスは、ユーザーに代わってアクションを実行するロールを引き受けることができます。サービスにリンクされたロールは に表示され AWS アカウント 、サービスによって所有されます。IAM 管理者は、サービスリンクロールの許可を表示できますが、編集することはできません。
- Amazon EC2 で実行されているアプリケーション – IAM ロールを使用して、EC2 インスタンスで実行され、 AWS CLI または AWS API リクエストを行うアプリケーションの一時的な認証情報を管理できます。これは、EC2 インスタンス内でのアクセスキーの保存に推奨されます。 AWS ロールを EC2 インスタンスに割り当て、そのすべてのアプリケーションで使用できるようにするには、インスタンスにアタッチされたインスタンスプロファイルを作成します。インスタンスプロファイルにはロールが含まれ、EC2 インスタンスで実行されるプログラムは一時的な認証情報を取得できます。詳細については、「IAM ユーザーガイド」の「Amazon EC2 インスタンスで実行されるアプリケーションに IAM ロールを使用して許可を付与する」を参照してください。

IAM ロールと IAM ユーザーのどちらを使用するかについては、「IAM ユーザーガイド」の「(IAM ユーザーではなく) IAM ロールをいつ作成したら良いのか?」を参照してください。

ポリシーを使用したアクセスの管理

でアクセスを制御する AWS には、ポリシーを作成し、 AWS ID またはリソースにアタッチします。ポリシーは、アイデンティティまたはリソースに関連付けられているときにアクセス許可を定義する のオブジェクトです。 は、プリンシパル (ユーザー、ルートユーザー、またはロールセッション) AWS がリクエストを行うときに、これらのポリシー AWS を評価します。ポリシーでの権限により、リクエストが許可されるか拒否されるかが決まります。ほとんどのポリシーは JSON ドキュメント AWS として に保存されます。JSON ポリシードキュメントの構造と内容の詳細については、「IAM ユーザーガイド」の「JSON ポリシー概要」を参照してください。

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどんなリソースにどんな条件でアクションを実行できるかということです。

デフォルトでは、ユーザーやロールに権限はありません。IAM 管理者は、リソースで必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。その後、管理者はロールに IAM ポリシーを追加し、ユーザーはロールを引き継ぐことができます。

IAM ポリシーは、オペレーションの実行方法を問わず、アクションの許可を定義します。例えば、`iam:GetRole` アクションを許可するポリシーがあるとします。そのポリシーを持つユーザーは、 AWS Management Console、、 AWS CLIまたは AWS API からロール情報を取得できます。

アイデンティティベースのポリシー

アイデンティティベースポリシーは、IAM ユーザー、ユーザーのグループ、ロールなど、アイデンティティにアタッチできる JSON 権限ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件をコントロールします。アイデンティティベースのポリシーを作成する方法については、「IAM ユーザーガイド」の「IAM ポリシーの作成」を参照してください。

アイデンティティベースのポリシーは、さらにインラインポリシーまたはマネージドポリシーに分類できます。インラインポリシーは、単一のユーザー、グループ、またはロールに直接埋め込まれています。管理ポリシーは、 内の複数のユーザー、グループ、ロールにアタッチできるスタンドアロンポリシーです AWS アカウント。管理ポリシーには、 AWS 管理ポリシーとカスタマー管理ポリシーが含まれます。マネージドポリシーまたはインラインポリシーのいずれかを選択する方法については、「IAM ユーザーガイド」の「マネージドポリシーとインラインポリシーの比較」を参照してください。

リソースベースのポリシー

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。リソースベースのポリシーには例として、IAM ロールの信頼ポリシー や Amazon S3 バケットポリシー があげられます。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスを制御できます。ポリシーがアタッチされているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。プリンシパルには、アカウント、ユーザー、ロール、フェデレーティッドユーザー、または を含めることができます AWS のサービス。

リソースベースのポリシーは、そのサービス内にあるインラインポリシーです。リソースベースのポリシーでは、IAM の AWS マネージドポリシーを使用できません。

アクセスコントロールリスト (ACL)

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするための許可を持つかを制御します。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

Amazon S3、 AWS WAF、および Amazon VPC は、ACLs。ACL の詳細については、『Amazon Simple Storage Service デベロッパーガイド』の「アクセスコントロールリスト (ACL) の概要」を参照してください。

その他のポリシータイプ

AWS は、一般的ではない追加のポリシータイプをサポートします。これらのポリシータイプでは、より一般的なポリシータイプで付与された最大の権限を設定できます。

- アクセス許可の境界 - アクセス許可の境界は、アイデンティティベースのポリシーによって IAM エンティティ (IAM ユーザーまたはロール) に付与できる権限の上限を設定する高度な機能です。エンティティにアクセス許可の境界を設定できます。結果として得られる権限は、エンティティのアイデンティティベースポリシーとそのアクセス許可の境界の共通部分になります。`Principal` フィールドでユーザーまたはロールを指定するリソースベースのポリシーでは、アクセス許可の境界は制限されません。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。アクセス許可の境界の詳細については、「IAM ユーザーガイド」の「IAM エンティティのアクセス許可の境界」を参照してください。
- サービスコントロールポリシー (SCPs) – SCPs は、 の組織または組織単位 (OU) に対する最大アクセス許可を指定する JSON ポリシーです AWS Organizations。 AWS Organizations は、 AWS

アカウント ビジネスが所有する複数の をグループ化して一元管理するためのサービスです。組織内のすべての機能を有効にすると、サービスコントロールポリシー (SCP) を一部またはすべてのアカウントに適用できます。SCP は、各 を含むメンバーアカウントのエンティティのアクセス許可を制限します AWS アカウントのルートユーザー。Organizations と SCP の詳細については、AWS Organizations ユーザーガイドの「SCP の仕組み」を参照してください。

- セッションポリシー - セッションポリシーは、ロールまたはフェデレーションユーザーの一時的なセッションをプログラムで作成する際にパラメータとして渡す高度なポリシーです。結果としてセッションの権限は、ユーザーまたはロールのアイデンティティベースポリシーとセッションポリシーの共通部分になります。また、リソースベースのポリシーから権限が派生する場合もあります。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。詳細については、「IAM ユーザーガイド」の「セッションポリシー」を参照してください。

複数のポリシータイプ

1 つのリクエストに複数のタイプのポリシーが適用されると、結果として作成される権限を理解するのがさらに難しくなります。複数のポリシータイプが関与する場合にリクエストを許可するかどうか AWS を決定する方法については、IAM ユーザーガイドの「ポリシー評価ロジック」を参照してください。

Amazon Lex で IAM を使用する方法

IAM を使用して Amazon Lex へのアクセスを管理する前に、Amazon Lex で利用できる IAM の機能について学習します。

Amazon Lex で使用できる IAM の機能

IAM 機能	Amazon Lex のサポート
アイデンティティベースのポリシー	Yes
リソースベースのポリシー	No
ポリシーアクション	Yes
ポリシーリソース	はい
ポリシー条件キー (サービス固有)	はい

IAM 機能	Amazon Lex のサポート
ACL	No
ABAC (ポリシー内のタグ)	部分的
一時的な認証情報	Yes
プリンシパル権限	Yes
サービスロール	あり
サービスリンクロール	はい

Amazon Lex およびその他の AWS のサービスがほとんどの IAM 機能と連携する方法の概要を把握するには、「IAM ユーザーガイド」のAWS 「IAM と連携する のサービス」を参照してください。

Amazon Lex のアイデンティティベースのポリシー

アイデンティティベースポリシーをサポートする	Yes

アイデンティティベースポリシーは、IAM ユーザー、ユーザーのグループ、ロールなど、アイデンティティにアタッチできる JSON 許可ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件をコントロールします。アイデンティティベースのポリシーを作成する方法については、「IAM ユーザーガイド」の「IAM ポリシーの作成」を参照してください。

IAM アイデンティティベースのポリシーでは、許可または拒否するアクションとリソース、およびアクションを許可または拒否する条件を指定できます。プリンシパルは、それが添付されているユーザーまたはロールに適用されるため、アイデンティティベースのポリシーでは指定できません。JSON ポリシーで使用できるすべての要素については、「IAM ユーザーガイド」の「IAM JSON ポリシーの要素のリファレンス」を参照してください。

Amazon Lex のアイデンティティベースのポリシー例

Amazon Lex のアイデンティティベースポリシーの例を確認するには、「Amazon Lex のアイデンティティベースのポリシー例」を参照してください。

Amazon Lex 内のリソースベースのポリシー

リソースベースのポリシーのサポート	No

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。リソースベースのポリシーには例として、IAM ロールの信頼ポリシー や Amazon S3 バケットポリシー があげられます。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスを制御できます。ポリシーがアタッチされているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。プリンシパルには、アカウント、ユーザー、ロール、フェデレーティッドユーザー、または を含めることができます AWS のサービス。

クロスアカウントアクセスを有効にするには、アカウント全体、または別のアカウントの IAM エンティティをリソースベースのポリシーのプリンシパルとして指定します。リソースベースのポリシーにクロスアカウントのプリンシパルを追加しても、信頼関係は半分しか確立されない点に注意してください。プリンシパルとリソースが異なる にある場合 AWS アカウント、信頼されたアカウントの IAM 管理者は、プリンシパルエンティティ (ユーザーまたはロール) にリソースへのアクセス許可も付与する必要があります。IAM 管理者は、アイデンティティベースのポリシーをエンティティにアタッチすることで権限を付与します。ただし、リソースベースのポリシーで、同じアカウントのプリンシパルへのアクセス権が付与されている場合は、アイデンティティベースのポリシーをさらに付与する必要はありません。詳細については、「IAM ユーザーガイド」の「IAM でのクロスアカウントリソースアクセス」を参照してください。

Amazon Lex のポリシーアクション

ポリシーアクションに対するサポート	はい

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどんなリソースにどんな条件でアクションを実行できるかということです。

JSON ポリシーの`Action`要素には、ポリシー内のアクセスを許可または拒否するために使用できるアクションが記述されます。ポリシーアクションの名前は通常、関連付けられた AWS API オペレーションと同じです。一致する API オペレーションのない許可のみのアクションなど、いくつかの例外があります。また、ポリシーに複数のアクションが必要なオペレーションもあります。これらの追加アクションは、依存アクションと呼ばれます。

このアクションは、関連付けられたオペレーションを実行するための権限を付与するポリシーで使用されます。

Amazon Lex アクションのリストを確認するには、「サービス認証リファレンス」の「Amazon Lex で定義されるアクション」を参照してください。

Amazon Lex のポリシーアクションは、アクションの前にプレフィックス を使用します。

```
lex
```

単一のステートメントで複数のアクションを指定するには、アクションをカンマで区切ります。

```
"Action": [
      "lex:action1",
      "lex:action2"
         ]
```

ワイルドカード (*) を使用して複数アクションを指定できます。例えば、Describe という単語で始まるすべてのアクションを指定するには、次のアクションを含めます。

```
"Action": "lex:Describe*"
```

Amazon Lex のポリシーリソース

ポリシーリソースに対するサポート	はい

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースにどのような条件でアクションを実行できるかということです。

Resource JSON ポリシー要素は、アクションが適用されるオブジェクトを指定します。ステートメントには、Resource または NotResource要素を含める必要があります。ベストプラクティスとして、Amazon リソースネーム (ARN) を使用してリソースを指定します。これは、リソースレベルの許可と呼ばれる特定のリソースタイプをサポートするアクションに対して実行できます。

オペレーションのリスト化など、リソースレベルの権限をサポートしないアクションの場合は、ステートメントがすべてのリソースに適用されることを示すために、ワイルドカード (*) を使用します。

```
"Resource": "*"
```

Amazon Lex ボットのリソースの ARN は、次の形式になります。

```
arn:aws:lex:${Region}:${Account}:bot:${Bot-Name}
```

ARN の形式の詳細については、「Amazon リソースネーム (ARNs AWS 「サービス名前空間」を参照してください。

例えば、ステートメントで OrderFlowers ボットを指定するには、次の ARN を使用します。

```
"Resource": "arn:aws:lex:us-east-2:123456789012:bot:OrderFlowers"
```

特定のアカウントに属するすべてのボットを指定するには、ワイルドカード (*) を使用します。

```
"Resource": "arn:aws:lex:us-east-2:123456789012:bot:*"
```

リソースの作成など、一部の Amazon Lex アクションは、特定のリソースで実行できません。このような場合は、ワイルドカード (*) を使用する必要があります。

```
"Resource": "*"
```

Amazon Lex リソースのタイプとその ARN のリストを確認するには、「サービス認証リファレンス」 の「Amazon Lex で定義されるリソース」を参照してください。どのアクションで各リソースの ARN を指定できるかについては、「Amazon Lex で定義されるアクション」を参照してください。

Amazon Lex のポリシー条件キー

サービス固有のポリシー条件キーのサポート	はい

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどんなリソースにどんな条件でアクションを実行できるかということです。

Condition 要素 (または Condition ブロック) を使用すると、ステートメントが有効な条件を指定できます。Condition 要素はオプションです。イコールや未満などの 条件演算子 を使用して条件式を作成することで、ポリシーの条件とリクエスト内の値を一致させることができます。

1 つのステートメントに複数の Condition 要素を指定するか、1 つの Condition 要素に複数のキーを指定すると、 AWS は AND 論理演算子を使用してそれらを評価します。1 つの条件キーに複数の値を指定すると、 は論理ORオペレーションを使用して条件 AWS を評価します。ステートメントの権限が付与される前にすべての条件が満たされる必要があります。

条件を指定する際にプレースホルダー変数も使用できます。例えば IAM ユーザーに、IAM ユーザー名がタグ付けされている場合のみリソースにアクセスできる権限を付与することができます。詳細については、「IAM ユーザーガイド」の「IAM ポリシーの要素: 変数およびタグ」を参照してください。

AWS は、グローバル条件キーとサービス固有の条件キーをサポートします。すべての AWS グローバル条件キーを確認するには、「IAM ユーザーガイド」のAWS 「 グローバル条件コンテキストキー」を参照してください。

Amazon Lex の条件キーのリストを確認するには、「サービス認証リファレンス」の「Amazon Lex の条件キー」を参照してください。条件キーを使用できるアクションとリソースについては、「Amazon Lex で定義されるアクション」を参照してください。

以下の表では、Amazon Lex リソースに適用される Amazon Lex 条件キーを一覧表示しています。これらのキーは、IAM アクセス権限ポリシーの Condition 要素に含めることができます。

Amazon Lex での ACL

ACL のサポート	No

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするための許可を持つかをコントロールします。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

Amazon Lex での ABAC

ABAC (ポリシー内のタグ) のサポート	部分的

属性ベースのアクセス制御 (ABAC) は、属性に基づいてアクセス許可を定義する認可戦略です。では AWS、これらの属性はタグ と呼ばれます。タグは、IAM エンティティ (ユーザーまたはロール) および多くの AWS リソースにアタッチできます。エンティティとリソースのタグ付けは、ABAC の最初

の手順です。その後、プリンシパルのタグがアクセスしようとしているリソースのタグと一致した場合にオペレーションを許可するように ABAC ポリシーをします。

ABAC は、急成長する環境やポリシー管理が煩雑になる状況で役立ちます。

タグに基づいてアクセスを管理するには、aws:ResourceTag/*key-name*、aws:RequestTag/*key-name*、または aws:TagKeys の条件キーを使用して、ポリシーの条件要素でタグ情報を提供します。

サービスがすべてのリソースタイプに対して 3 つの条件キーすべてをサポートする場合、そのサービスの値ははいです。サービスが一部のリソースタイプに対してのみ 3 つの条件キーのすべてをサポートする場合、値は「部分的」になります。

ABAC の詳細については、IAM ユーザーガイド の「ABAC とは?」を参照してください。ABAC をセットアップするステップを説明するチュートリアルについては、「IAM ユーザーガイド」の「属性ベースのアクセス制御 (ABAC) を使用する」を参照してください。

承認のために、特定のタイプの Amazon Lex リソースにタグを関連付けることができます。タグに基づいてアクセスをコントロールするには、lex:ResourceTag/${TagKey}、aws:RequestTag/${TagKey}、または aws:TagKeys 条件キーを使用して、ポリシーの条件要素でタグ情報を提供します。

Amazon Lex リソースのタグ付けの詳細については、「Amazon Lex リソースのタグ付け」を参照してください。

リソースのタグに基づいてリソースへのアクセスを制限するためのアイデンティティベースポリシーの例を表示するには、「 タグを使用したリソースへのアクセス」を参照してください。

次の表に、タグベースのアクセスコントロールのアクションと対応するリソースタイプを示します。各アクションは、対応するリソースタイプに関連付けられたタグに基づいて許可されます。

アクション	リソースタイプ	条件キー	メモ
CreateBotVersion	ボット	lex:ResourceTag	
DeleteBot	ボット	lex:ResourceTag	
DeleteBotAlias	alias	lex:ResourceTag	
DeleteBotChannelAssociation	チャンネル	lex:ResourceTag	

アクション	リソースタイプ	条件キー	メモ
DeleteBotVersion	ボット	lex:ResourceTag	
DeleteSession	ボットまたはエイリアス	lex:ResourceTag	エイリアスが $LATEST に設定されている場合、ボットに関連付けられたタグを使用します。他のエイリアスと共に使用する場合は、指定したエイリアスに関連付けられたタグを使用します。
DeleteUtterances	ボット	lex:ResourceTag	
GetBot	ボットまたはエイリアス	lex:ResourceTag	versionOrAlias が $LATEST または数値バージョンに設定されている場合、ボットに関連付けられたタグを使用します。エイリアスと共に使用する場合は、指定したエイリアスに関連付けられたタグを使用します。
GetBotAlias	alias	lex:ResourceTag	
GetBotChannelAssociation	チャンネル	lex:ResourceTag	

アクション	リソースタイプ	条件キー	メモ
GetBotChannelAssociations	チャンネル	lex:ResourceTag	エイリアスが "-" に設定されている場合、ボットに関連付けられたタグを使用します。ボットエイリアスが指定されている場合は、指定したエイリアスに関連付けられたタグを使用します。
GetBotVersions	ボット	lex:ResourceTag	
GetExport	ボット	lex:ResourceTag	
GetSession	ボットまたはエイリアス	lex:ResourceTag	エイリアスが $LATEST に設定されている場合、ボットに関連付けられたタグを使用します。他のエイリアスと共に使用する場合は、指定したエイリアスに関連付けられたタグを使用します。
GetUtterancesView	ボット	lex:ResourceTag	
ListTagsForResource	ボット、エイリアス、またはチャンネル	lex:ResourceTag	

アクション	リソースタイプ	条件キー	メモ
PostContent	ボットまたはエイリアス	lex:ResourceTag	エイリアスが $LATEST に設定されている場合、ボットに関連付けられたタグを使用します。他のエイリアスと共に使用する場合は、指定したエイリアスに関連付けられたタグを使用します。
PostText	ボットまたはエイリアス	lex:ResourceTag	エイリアスが $LATEST に設定されている場合、ボットに関連付けられたタグを使用します。他のエイリアスと共に使用する場合は、指定したエイリアスに関連付けられたタグを使用します。
PutBot	ボット	lex:ResourceTag, aws:RequestTag, aws:TagKeys	
PutBotAlias	alias	lex:ResourceTag, aws:RequestTag, aws:TagKeys	

アクション	リソースタイプ	条件キー	メモ
PutSession	ボットまたはエイリアス	lex:ResourceTag	エイリアスが $LATEST に設定されている場合、ボットに関連付けられたタグを使用します。他のエイリアスと共に使用する場合は、指定したエイリアスに関連付けられたタグを使用します。
StartImport	ボット	lex:ResourceTag	PutBot オペレーションのアクセスポリシーに依存します。StartImport オペレーションに固有のタグとアクセス許可は無視されます。
TagResource	ボット、エイリアス、またはチャンネル	lex:ResourceTag, aws:RequestTag, aws:TagKeys	
UntagResource	ボット、エイリアス、またはチャンネル	lex:ResourceTag, aws:RequestTag, aws:TagKeys	

Amazon Lex での一時的な認証情報の使用

一時的な認証情報のサポート	はい

一部の AWS のサービス は、一時的な認証情報を使用してサインインすると機能しません。一時的な認証情報 AWS のサービス を使用する などの詳細については、IAM ユーザーガイドのAWS のサービス「IAM と連携する 」を参照してください。

ユーザー名とパスワード以外の AWS Management Console 方法で にサインインする場合、一時的な認証情報を使用します。例えば、会社の Single Sign-On (SSO) リンク AWS を使用して にアクセスすると、そのプロセスによって一時的な認証情報が自動的に作成されます。また、ユーザーとしてコンソールにサインインしてからロールを切り替える場合も、一時的な認証情報が自動的に作成されます。ロールの切り替えに関する詳細については、「IAM ユーザーガイド」の「ロールへの切り替え (コンソール)」を参照してください。

一時的な認証情報は、 AWS CLI または AWS API を使用して手動で作成できます。その後、これらの一時的な認証情報を使用して . AWS recommends にアクセスできます AWS。これは、長期的なアクセスキーを使用する代わりに、一時的な認証情報を動的に生成することを推奨しています。詳細については、「IAM の一時的セキュリティ認証情報」を参照してください。

一時的な認証情報を使用して、フェデレーションでサインインする、IAM ロールを引き受ける、またはクロスアカウントロールを引き受けることができます。一時的なセキュリティ認証情報を取得するには、 AssumeRoleやGetFederationトークン などの AWS STS API オペレーションを呼び出します。

Amazon Lex のクロスサービスプリンシパルのアクセス許可

フォワードアクセスセッション (FAS) をサポート	はい

IAM ユーザーまたはロールを使用して でアクションを実行すると AWS、プリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがあります。FAS は、 を呼び出すプリンシパルのアクセス許可を AWS のサービス、ダウンストリームサービス AWS のサービス へのリクエストのリクエストと組み合わせて使用します。FAS リクエストは、サービスが他の AWS のサービス またはリソースとのやり取りを完了する必要があるリクエストを受け取った場合にのみ行われます。この場合、両方のアクションを実行するためのアクセス許可が必要です。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。

Amazon Lex のサービスロール

サービスロールに対するサポート	あり

サービスロールとは、サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除できます。詳細については、「IAM ユーザーガイド」の「AWS のサービスにアクセス許可を委任するロールの作成」を参照してください。

> **Warning**
>
> サービスロールのアクセス許可を変更すると、Amazon Lex の機能が破損する可能性があります。Amazon Lex が指示する場合以外は、サービスロールを編集しないでください。

Amazon Lex での IAM ロールの選択

Amazon Lex は、Amazon Comprehend と Amazon Polly を呼び出すために、サービスリンクされたロールを使用しています。 AWS Lambda 関数に対するリソースレベルのアクセス許可を使用して関数を呼び出します。

会話のタグ付けを有効にするには、IAM ロールを指定する必要があります。詳細については、「会話ログ用の IAM ロールとポリシーの作成」を参照してください。

Amazon Lex のサービスリンクロール

サービスリンクロールのサポート	はい

サービスにリンクされたロールは、 にリンクされたサービスロールの一種です AWS のサービス。サービスは、ユーザーに代わってアクションを実行するロールを引き受けることができます。サービスにリンクされたロールは に表示され AWS アカウント 、サービスによって所有されます。IAM 管理者は、サービスリンクロールの許可を表示できますが、編集することはできません。

Amazon Lex でのサービスにリンクされたロールの作成または管理の詳細については、「Amazon Lex のサービスリンクロールの使用」を参照してください。

Amazon Lex のアイデンティティベースのポリシー例

デフォルトでは、ユーザーおよびロールには Amazon Lex リソースを作成または変更するアクセス許可はありません。また、、 AWS Command Line Interface （ AWS CLI ） AWS Management Console、または AWS API を使用してタスクを実行することはできません。IAM 管理者は、リソースで必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。その後、管理者はロールに IAM ポリシーを追加し、ユーザーはロールを引き受けることができます。

これらサンプルの JSON ポリシードキュメントを使用して、IAM アイデンティティベースのポリシーを作成する方法については、「IAM ユーザーガイド」の「IAM ポリシーの作成」を参照してください。

Amazon Lex が定義するアクションとリソースタイプ (リソースタイプごとの ARN の形式を含む) の詳細については、サービス認証リファレンスの「Amazon Lex のアクション、リソース、および条件キー」を参照してください。

トピック

- ポリシーのベストプラクティス
- Amazon Lex コンソールを使用する
- 自分の権限の表示をユーザーに許可する
- すべての Amazon Lex ボットを削除する
- ユーザーが Amazon Lex V2 API にボットを移行できるようにする
- タグを使用したリソースへのアクセス

ポリシーのベストプラクティス

アイデンティティベースのポリシーは、ユーザーのアカウント内で誰かが Amazon Lex リソースを作成、アクセス、または削除できるどうかを決定します。これらのアクションを実行すると、 AWS アカウントに料金が発生する可能性があります。アイデンティティベースポリシーを作成したり編集したりする際には、以下のガイドラインと推奨事項に従ってください:

- AWS 管理ポリシーを開始し、最小特権のアクセス許可に移行する – ユーザーとワークロードにアクセス許可を付与するには、多くの一般的なユースケースにアクセス許可を付与する AWS 管理ポリシーを使用します。これらは で使用できます AWS アカウント。ユースケースに固有の AWS カスタマー管理ポリシーを定義して、アクセス許可をさらに減らすことをお勧めします。詳細につい

ては、「IAM ユーザーガイド」の「AWS マネージドポリシー」または「AWS ジョブ機能の管理ポリシー」を参照してください。

- 最小特権を適用する – IAM ポリシーで許可を設定する場合は、タスクの実行に必要な許可のみを付与します。これを行うには、特定の条件下で特定のリソースに対して実行できるアクションを定義します。これは、最小特権アクセス許可とも呼ばれています。IAM を使用して許可を適用する方法の詳細については、「IAM ユーザーガイド」の「IAM でのポリシーとアクセス許可」を参照してください。
- IAM ポリシーで条件を使用してアクセスをさらに制限する - ポリシーに条件を追加して、アクションやリソースへのアクセスを制限できます。例えば、ポリシー条件を記述して、すべてのリクエストを SSL を使用して送信するように指定できます。条件を使用して、 などの特定の を介してサービスアクションが使用される場合に AWS のサービス、サービスアクションへのアクセスを許可することもできます AWS CloudFormation。詳細については、「IAM ユーザーガイド」の [IAM JSON policy elements: Condition] (IAM JSON ポリシー要素: 条件) を参照してください。
- IAM Access Analyzer を使用して IAM ポリシーを検証し、安全で機能的な権限を確保する - IAM Access Analyzer は、新規および既存のポリシーを検証して、ポリシーが IAM ポリシー言語 (JSON) および IAM のベストプラクティスに準拠するようにします。IAM アクセスアナライザーは 100 を超えるポリシーチェックと実用的な推奨事項を提供し、安全で機能的なポリシーの作成をサポートします。詳細については、「IAM ユーザーガイド」の「IAM Access Analyzer ポリシーの検証」を参照してください。
- 多要素認証 (MFA) を要求する – で IAM ユーザーまたはルートユーザーを必要とするシナリオがある場合は AWS アカウント、セキュリティを強化するために MFA を有効にします。API オペレーションが呼び出されるときに MFA を必須にするには、ポリシーに MFA 条件を追加します。詳細については、「IAM ユーザーガイド」の「MFA 保護 API アクセスの設定」を参照してください。

IAM でのベストプラクティスの詳細については、「IAM ユーザーガイド」の「IAM でのセキュリティのベストプラクティス」を参照してください。

Amazon Lex コンソールを使用する

Amazon Lex コンソールにアクセスするには、アクセス許可の最小限のセットが必要です。これらのアクセス許可により、 の Amazon Lex リソースの詳細を一覧表示および表示できます AWS アカウント。最小限必要な許可よりも制限が厳しいアイデンティティベースのポリシーを作成すると、そのポリシーを持つエンティティ (ユーザーまたはロール) に対してコンソールが意図したとおりに機能しません。

AWS CLI または AWS API のみを呼び出すユーザーには、最小限のコンソールアクセス許可を付与する必要はありません。代わりに、実行しようとしている API オペレーションに一致するアクションのみへのアクセスが許可されます。

AWS は、 によって作成および管理されるスタンドアロン IAM ポリシーを提供することで、多くの一般的なユースケースに対処します AWS。これらのポリシーは AWS 管理ポリシーと呼ばれます。AWS 管理ポリシーでは、ポリシーを自分で記述する場合よりも簡単に、適切なアクセス許可をユーザー、グループ、ロールに割り当てることができます。詳細については、「IAM ユーザーガイド」の「AWS マネージドポリシー」 を参照してください。

アカウントのグループとロールにアタッチできる次の AWS マネージドポリシーは、Amazon Lex に固有です。

- AmazonLexReadOnly — Amazon Lex リソースへの読み取り専用アクセスを許可します。
- AmazonLexRunBotsのみ — Amazon Lex 会話ボットを実行するためのアクセスを許可します。
- AmazonLexFullAccess — すべての Amazon Lex リソースを作成、読み取り、更新、削除、実行するためのフルアクセスを許可します。また、Amazon Lex を使用した `AmazonLex` で始まる名前の Lambda 関数を関連付ける機能を付与します。

> **Note**
>
> これらのアクセス権限ポリシーについては、IAM コンソールにサインインして特定のポリシーを検索することで確認できます。

このAmazonLexFullAccessポリシーは、イン`KendraSearchIntent`テントを使用して Amazon Kendra インデックスをクエリするアクセス許可をユーザーに付与しません。インデックスにクエリを実行するには、ポリシーに追加のアクセス許可を追加する必要があります。必要なアクセス許可については、「Amazon Kendra 検索の IAM ポリシー」を参照してください。

独自のカスタム IAM ポリシーを作成して、Amazon Lex API アクションにアクセス権限を付与することもできます。これらのカスタムポリシーは、それらのアクセス権限が必要な IAM ロールまたはグループにアタッチできます。

Amazon Lex の AWS マネージドポリシーの詳細については、「Amazon Lex の AWS 管理ポリシー」を参照してください。

自分の権限の表示をユーザーに許可する

この例では、ユーザーアイデンティティにアタッチされたインラインおよびマネージドポリシーの表示を IAM ユーザーに許可するポリシーの作成方法を示します。このポリシーには、コンソールで、または AWS CLI または AWS API を使用してプログラムでこのアクションを実行するアクセス許可が含まれています。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

すべての Amazon Lex ボットを削除する

このポリシー例は、AWS アカウントのユーザーにアカウント内の任意のボットを削除するアクセス許可を付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "lex:DeleteBot"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

ユーザーが Amazon Lex V2 API にボットを移行できるようにする

次の IAM 許可ポリシーにより、ユーザーは Amazon Lex から Amazon Lex V2 API へのボットの移行を開始し、移行のリストとその進捗状況を確認することができます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "startMigration",
            "Effect": "Allow",
            "Action": "lex:StartMigration",
            "Resource": "arn:aws:lex:<Region>:<123456789012>:bot:*"
        },
        {
            "Sid": "passRole",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::<123456789012>:role/<v2 bot role>"
        },
        {
            "Sid": "allowOperations",
            "Effect": "Allow",
```

```
            "Action": [
                "lex:CreateBot",
                "lex:CreateIntent",
                "lex:UpdateSlot",
                "lex:DescribeBotLocale",
                "lex:UpdateBotAlias",
                "lex:CreateSlotType",
                "lex:DeleteBotLocale",
                "lex:DescribeBot",
                "lex:UpdateBotLocale",
                "lex:CreateSlot",
                "lex:DeleteSlot",
                "lex:UpdateBot",
                "lex:DeleteSlotType",
                "lex:DescribeBotAlias",
                "lex:CreateBotLocale",
                "lex:DeleteIntent",
                "lex:StartImport",
                "lex:UpdateSlotType",
                "lex:UpdateIntent",
                "lex:DescribeImport",
                "lex:CreateCustomVocabulary",
                "lex:UpdateCustomVocabulary",
                "lex:DeleteCustomVocabulary",
                "lex:DescribeCustomVocabulary",
                "lex:DescribeCustomVocabularyMetadata"
            ],
            "Resource": [
                "arn:aws:lex:<Region>:<123456789012>:bot/*",
                "arn:aws:lex:<Region>:<123456789012>:bot-alias/*/*"
            ]
        },
        {
            "Sid": "showBots",
            "Effect": "Allow",
            "Action": [
                "lex:CreateUploadUrl",
                "lex:ListBots"
            ],
            "Resource": "*"
        },
        {
            "Sid": "showMigrations",
            "Effect": "Allow",
```

```
            "Action": [
                "lex:GetMigration",
                "lex:GetMigrations"
            ],
            "Resource": "*"
        }
    ]
}
```

タグを使用したリソースへのアクセス

このサンプルポリシーでは、 AWS アカウント内のユーザーまたはロールに、キー **Department** と値 **Support** でタグ付けされた任意のリソースで PostText オペレーションを使用するアクセス許可を付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": "lex:PostText",
            "Effect": "Allow",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "lex:ResourceTag/Department": "Support"
                }
            }
        }
    ]
}
```

Amazon Lex の AWS 管理ポリシー

AWS マネージドポリシーは、AWS が作成および管理するスタンドアロンポリシーです。AWS マネージドポリシーは、多くの一般的なユースケースでアクセス許可を提供できるように設計されているため、ユーザー、グループ、ロールへのアクセス許可の割り当てを開始できます。

AWS マネージドポリシーは、ご利用の特定のユースケースに対して最小特権のアクセス許可を付与しない場合があることにご注意ください。AWS のすべてのお客様が使用できるようになるのを避けるためです。ユースケース別にカスタマー管理ポリシーを定義することで、アクセス許可を絞り込むことをお勧めします。

AWS 管理ポリシーで定義したアクセス権限は変更できません。AWS が AWS マネージドポリシーに定義されているアクセス許可を更新すると、更新はポリシーがアタッチされているすべてのプリンシパルアイデンティティ (ユーザー、グループ、ロール) に影響します。新しい AWS のサービスを起動するか、既存のサービスで新しい API オペレーションが使用可能になると、AWS が AWS マネージドポリシーを更新する可能性が最も高くなります。

詳細については、「IAM ユーザーガイド」の「AWS 管理ポリシー」を参照してください。

AWS マネージドポリシー: AmazonLexReadOnly

`AmazonLexReadOnly` ポリシーは IAM ID にアタッチできます。

このポリシーは、ユーザーが Amazon Lex V2 および Amazon Lex Model·Building·Service のすべてのアクションを表示できるようにする読み取り専用権限を付与します。

許可の詳細

このポリシーには、以下の許可が含まれています。

- `lex` — Model·Building·Service の Amazon Lex および Amazon Lex V2 リソースへの読み取り専用アクセス。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "lex:GetBot",
                "lex:GetBotAlias",
                "lex:GetBotAliases",
                "lex:GetBots",
                "lex:GetBotChannelAssociation",
                "lex:GetBotChannelAssociations",
```

```
                "lex:GetBotVersions",
                "lex:GetBuiltinIntent",
                "lex:GetBuiltinIntents",
                "lex:GetBuiltinSlotTypes",
                "lex:GetIntent",
                "lex:GetIntents",
                "lex:GetIntentVersions",
                "lex:GetSlotType",
                "lex:GetSlotTypes",
                "lex:GetSlotTypeVersions",
                "lex:GetUtterancesView",
                "lex:DescribeBot",
                "lex:DescribeBotAlias",
                "lex:DescribeBotChannel",
                "lex:DescribeBotLocale",
                "lex:DescribeBotVersion",
                "lex:DescribeExport",
                "lex:DescribeImport",
                "lex:DescribeIntent",
                "lex:DescribeResourcePolicy",
                "lex:DescribeSlot",
                "lex:DescribeSlotType",
                "lex:ListBots",
                "lex:ListBotLocales",
                "lex:ListBotAliases",
                "lex:ListBotChannels",
                "lex:ListBotVersions",
                "lex:ListBuiltInIntents",
                "lex:ListBuiltInSlotTypes",
                "lex:ListExports",
                "lex:ListImports",
                "lex:ListIntents",
                "lex:ListSlots",
                "lex:ListSlotTypes",
                "lex:ListTagsForResource"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS マネージドポリシー: AmazonLexRunBotsOnly

AmazonLexRunBotsOnly ポリシーは IAM ID にアタッチできます。

このポリシーは、Amazon Lex および Amazon Lex V2 会話ボットを実行するためのアクセスを許可する読み取り専用権限を付与します。

許可の詳細

このポリシーには、以下の許可が含まれています。

- lex — Amazon Lex および Amazon Lex V2 ランタイムのすべてのアクションへの読み取り専用アクセス。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "lex:PostContent",
                "lex:PostText",
                "lex:PutSession",
                "lex:GetSession",
                "lex:DeleteSession",
                "lex:RecognizeText",
                "lex:RecognizeUtterance",
                "lex:StartConversation"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS 管理ポリシー: AmazonLexFullAccess

AmazonLexFullAccess ポリシーは IAM ID にアタッチできます。

このポリシーは、Amazon Lex および Amazon Lex V2 リソースの作成、読み取り、更新、および削除、および Amazon Lex および Amazon Lex V2 会話型ボットの実行を許可する管理者権限を付与します。

許可の詳細

このポリシーには、以下の許可が含まれています。

- `lex` — Amazon Lex および Amazon Lex V2 モデル構築および Runtime·Service のすべてのアクションに対するプリンシパルの読み取りおよび書き込みアクセスを許可します。
- `cloudwatch` — Amazon CloudWatch のメトリクスとアラームをプリンシパルに表示できるようにします。
- `iam` — プリンシパルは、サービスにリンクされたロールの作成と削除、ロールを渡し、ポリシーをロールにアタッチおよびデタッチできるようにします。権限は、Amazon Lex のオペレーションでは「lex.amazonaws.com」、Amazon Lex V2 オペレーションでは「lexv2.amazonaws.com」に制限されています。
- `kendra` — プリンシパルが Amazon Kendra インデックスを一覧表示できるようにします。
- `kms` — プリンシパルが AWS KMS キーとエイリアスを記述できるようにします。
- `lambda` — プリンシパルの一覧表示を許可するAWS Lambda 関数を実行し、Lambda 関数にアタッチされた権限を管理できるようにします。
- `polly` — プリンシパルが Amazon Polly の音声を説明し、音声を合成できるようにします。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:DescribeAlarms",
                "cloudwatch:DescribeAlarmsForMetric",
                "kms:DescribeKey",
                "kms:ListAliases",
                "lambda:GetPolicy",
                "lambda:ListFunctions",
                "lex:*",
                "polly:DescribeVoices",
                "polly:SynthesizeSpeech",
                "kendra:ListIndices",
                "iam:ListRoles",
                "s3:ListAllMyBuckets",
                "logs:DescribeLogGroups",
                "s3:GetBucketLocation"
```

```
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "lambda:AddPermission",
                "lambda:RemovePermission"
            ],
            "Resource": "arn:aws:lambda:*:*:function:AmazonLex*",
            "Condition": {
                "StringEquals": {
                    "lambda:Principal": "lex.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:GetRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/aws-service-role/lex.amazonaws.com/
AWSServiceRoleForLexBots",
                "arn:aws:iam::*:role/aws-service-role/channels.lex.amazonaws.com/
AWSServiceRoleForLexChannels",
                "arn:aws:iam::*:role/aws-service-role/lexv2.amazonaws.com/
AWSServiceRoleForLexV2Bots*",
                "arn:aws:iam::*:role/aws-service-role/channels.lexv2.amazonaws.com/
AWSServiceRoleForLexV2Channels*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateServiceLinkedRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/aws-service-role/lex.amazonaws.com/
AWSServiceRoleForLexBots"
            ],
            "Condition": {
```

```
                "StringEquals": {
                    "iam:AWSServiceName": "lex.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateServiceLinkedRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/aws-service-role/channels.lex.amazonaws.com/
AWSServiceRoleForLexChannels"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:AWSServiceName": "channels.lex.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateServiceLinkedRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/aws-service-role/lexv2.amazonaws.com/
AWSServiceRoleForLexV2Bots*"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:AWSServiceName": "lexv2.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateServiceLinkedRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/aws-service-role/channels.lexv2.amazonaws.com/
AWSServiceRoleForLexV2Channels*"
            ],
```

```
            "Condition": {
                "StringEquals": {
                    "iam:AWSServiceName": "channels.lexv2.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:DeleteServiceLinkedRole",
                "iam:GetServiceLinkedRoleDeletionStatus"
            ],
            "Resource": [
                "arn:aws:iam::*:role/aws-service-role/lex.amazonaws.com/
AWSServiceRoleForLexBots",
                "arn:aws:iam::*:role/aws-service-role/channels.lex.amazonaws.com/
AWSServiceRoleForLexChannels",
                "arn:aws:iam::*:role/aws-service-role/lexv2.amazonaws.com/
AWSServiceRoleForLexV2Bots*",
                "arn:aws:iam::*:role/aws-service-role/channels.lexv2.amazonaws.com/
AWSServiceRoleForLexV2Channels*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/aws-service-role/lex.amazonaws.com/
AWSServiceRoleForLexBots"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "lex.amazonaws.com"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
```

```
            ],
            "Resource": [
                "arn:aws:iam::*:role/aws-service-role/lexv2.amazonaws.com/
AWSServiceRoleForLexV2Bots*"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "lexv2.amazonaws.com"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/aws-service-role/channels.lexv2.amazonaws.com/
AWSServiceRoleForLexV2Channels*"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "channels.lexv2.amazonaws.com"
                    ]
                }
            }
        }
    ]
}
```

Amazon Lex での AWS 管理ポリシーに関する更新

Amazon Lex の AWS 管理ポリシーに対する更新の詳細について、このサービスがこれらの変更の追跡を開始した以降のものを示します。このページでの変更に関する自動通知を受けるには、Amazon Lex Amazon Lex のドキュメント履歴 ページから RSS フィードを購読してください。

変更	説明	日付
AmazonLexFullAccess — 既存のポリシーに対する更新	Amazon Lex では、Amazon Lex V2 Model·Building·Service オペレーションへの読み取り専用アクセスを許可する新しいアクセス許可が追加されました。	2021 年 8 月 18 日
AmazonLexReadOnly — 既存のポリシーに対する更新	Amazon Lex では、Amazon Lex V2 Model Building Service オペレーションへの読み取り専用アクセスを許可する新しいアクセス許可が追加されました。	2021 年 8 月 18 日
AmazonLexRunBotsOnly — 既存のポリシーに対する更新	Amazon Lex では、Amazon Lex V2 Runtime·Service オペレーションへの読み取り専用アクセスを許可する新しいアクセス許可が追加されました。	2021 年 8 月 18 日
Amazon Lex が変更の追跡を開始しました。	Amazon Lex が AWS マネージドポリシーの変更の追跡を開始しました。	2021 年 8 月 18 日

Amazon Lex のサービスリンクロールの使用

Amazon Lex は、AWS Identity and Access Management (IAM) サービスリンクロールを使用しています。サービスリンクロールは、Amazon Lex に直接リンクされた特殊なタイプの IAM ロールです。サービスにリンクされたロールは Amazon Lex によって事前に定義されており、サービスがユーザーに代わって他の AWS のサービスを呼び出すために必要な、すべての許可が含まれています。

必要な許可を手動で追加する必要がないため、サービスリンクロールは Amazon Lex のセットアップを容易にします。サービスリンクロールの許可は Amazon Lex が定義し、別段の定義がない限

り、Amazon Lex のみがその口ールを引き受けることができます。定義される許可には、信頼ポリシーとアクセス許可ポリシーが含まれており、そのアクセス許可ポリシーを他の IAM エンティティに添付することはできません。

サービスリンクロールを削除するには、まずその関連リソースを削除します。これは、リソースにアクセスするための許可を誤って削除できないため、Amazon Lex リソースを保護します。

Amazon Lex でのサービスにリンクされたロールのアクセス許可

Amazon Lex では、サービスにリンクされたロールを 2 つ使用します。

- AWSServiceRoleForLexBots - Amazon Lex は、このサービスにリンクされたロールを使用して、ボットの音声レスポンスの合成のために Amazon Polly を呼び出し、センチメント分析のために Amazon Comprehend を呼び出します。また、オプションで、インデックスの検索のために Amazon Kendra を呼び出します。
- AWSServiceRoleForLexChannels - Amazon Lex は、このサービスにリンクされたロールを使用し、チャンネルの管理時にテキストをボットに投稿します。

サービスにリンクされたロールの作成、編集、削除をIAM エンティティ (ユーザー、グループ、ロールなど) に許可するには、許可を設定する必要があります。詳細については、「IAM ユーザーガイド」の「Service-Linked Role Permissions」(サービスリンクロールの許可) を参照してください。

Amazon Lex のサービスリンクロールの作成

サービスにリンクされたロールを手動で作成する必要はありません。AWS Management Console でボット、ボットチャンネル、または Amazon Kendra 検索インテントを作成すると、Amazon Lex によってサービスにリンクされたロールが作成されます。

このサービスにリンクされたロールを削除した後で再度作成する必要が生じた場合は、同じ方法でアカウントにロールを再作成できます。新しいボット、チャンネルの関連付け、または Amazon Kendra 検索インテント を作成すると、Amazon Lex によって、サービスリンクされたロールが再度作成されます。

AWS CLI を使用して、AWSServiceRoleForLexBots ユースケースで、サービスにリンクされたロールを作成することもできます。AWS CLI で、サービスにリンクされたロールを Amazon Lex サービス名 `lex.amazonaws.com` で作成します。詳細については、「ステップ 1: サービスにリンクされたロールを作成する (AWS CLI)」を参照してください。このサービスリンクロールを削除する場合、この同じプロセスを使用して、もう一度ロールを作成できます。

Amazon Lex のサービスリンクロールの編集

Amazon Lex では、サービスにリンクされたロール Amazon Lex Service-Linked Roles を編集することはできません。サービスにリンクされたロールを作成すると、多くのエンティティによってロールが参照される可能性があるため、ロール名を変更することはできません。ただし、IAM を使用したロールの説明の編集はできます。詳細については、「IAM ユーザーガイド」の「サービスにリンクされたロールの編集」を参照してください。

Amazon Lex のサービスリンクロールの削除

サービスにリンクされたロールが必要な機能またはサービスが不要になった場合には、そのロールを削除することをお勧めします。そうすることで、積極的にモニタリングまたは保守されていない未使用のエンティティを排除できます。ただし、手動で削除する前に、サービスにリンクされたロールのリソースをクリーンアップする必要があります。

> **Note**
>
> リソースを削除しようとしているときに Amazon Lex サービスがロールを使用している場合は、削除が失敗する可能性があります。失敗した場合は、数分待ってから操作を再試行してください。

サービスにリンクされたロールによって使用される Amazon Lex リソースを削除するには:

1. 使用しているボットチャンネルをすべて削除します。
2. アカウント内のボットをすべて削除します。

IAM を使用してサービスリンクロールを手動で削除するには

IAM コンソール、AWS CLI、または AWS API を使用して、Amazon Lex サービスリンクロールを削除します。詳細については、「IAM ユーザーガイド」の「サービスにリンクされたロールの削除」を参照してください。

Amazon Lex のサービスにリンクされたロールがサポートされるリージョン

Amazon Lex は、このサービスを利用できるすべてのリージョンでサービスリンクロールの使用をサポートします。詳細については、「Amazon Lex endpoints and quotas」(Amazon Lex エンドポイントとクォータ) を参照してください。

Amazon Lex アイデンティティとアクセスのトラブルシューティング

次の情報は、Amazon Lex と IAM の使用に伴って発生する可能性がある一般的な問題の診断や修復に役立ちます。

トピック

- Amazon Lex でアクションを実行する権限がない
- iam を実行する権限がありません。PassRole
- 自分の 以外のユーザーに Amazon Lex リソース AWS アカウント へのアクセスを許可したい

Amazon Lex でアクションを実行する権限がない

「I am not authorized to perform an action in Amazon Bedrock」というエラーが表示された場合、そのアクションを実行できるようにポリシーを更新する必要があります。

次のエラー例は、mateojackson IAM ユーザーがコンソールを使用して、ある *my-example-widget* リソースに関する詳細情報を表示しようとしたことを想定して、その際に必要な`lex:`*`GetWidget`* アクセス許可を持っていない場合に発生するものです。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 lex:GetWidget on resource: my-example-widget
```

この場合、`lex:`*`GetWidget`* アクションを使用して *my-example-widget*リソースへのアクセスを許可するように、mateojackson ユーザーのポリシーを更新する必要があります。

サポートが必要な場合は、 AWS 管理者にお問い合わせください。サインイン認証情報を提供した担当者が管理者です。

iam を実行する権限がありません。PassRole

`iam:PassRole` アクションを実行する権限がないというエラーが表示された場合は、ポリシーを更新して Amazon Lex にロールを渡せるようにする必要があります。

一部の AWS のサービス では、新しいサービスロールまたはサービスにリンクされたロールを作成する代わりに、そのサービスに既存のロールを渡すことができます。そのためには、サービスにロールを渡す権限が必要です。

以下の例のエラーは、marymajor という IAM ユーザーがコンソールを使用して Amazon Lex でアクションを実行しようする場合に発生します。ただし、このアクションをサービスが実行するには、

サービスロールから付与された権限が必要です。メアリーには、ロールをサービスに渡す許可がありません。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

この場合、Mary のポリシーを更新してメアリーに `iam:PassRole` アクションの実行を許可する必要があります。

サポートが必要な場合は、 AWS 管理者にお問い合わせください。サインイン認証情報を提供した担当者が管理者です。

自分の 以外のユーザーに Amazon Lex リソース AWS アカウント へのアクセスを許可したい

他のアカウントのユーザーや組織外の人が、リソースにアクセスするために使用できるロールを作成できます。ロールの引き受けを委託するユーザーを指定できます。リソースベースのポリシーまたはアクセスコントロールリスト (ACL) をサポートするサービスの場合、それらのポリシーを使用して、リソースへのアクセスを付与できます。

詳細については、以下を参照してください:

- Amazon Lex がこれらの機能をサポートしているかどうかを確認するには、「Amazon Lex で IAM を使用する方法」を参照してください。
- 所有 AWS アカウント している のリソースへのアクセスを提供する方法については、IAM ユーザーガイドの「所有 AWS アカウント している別の の IAM ユーザーへのアクセスを提供する」を参照してください。
- リソースへのアクセスをサードパーティー に提供する方法については AWS アカウント、IAM ユーザーガイドの「サードパーティー AWS アカウント が所有する へのアクセスを提供する」を参照してください。
- ID フェデレーションを介してアクセスを提供する方法については、IAM ユーザーガイドの外部認証されたユーザーへのアクセスの提供 (ID フェデレーション) を参照してください。
- クロスアカウントアクセスでのロールとリソースベースのポリシーの使用の違いについては、IAM ユーザーガイドの「IAM でのクロスアカウントリソースアクセス」を参照してください。

Amazon Lex でのモニタリング

モニタリングは、Amazon Lex chatbot の信頼性、可用性、パフォーマンスを維持する上で重要です。このトピックでは、Amazon CloudWatch Logs および AWS CloudTrail を使用して Amazon Lex をモニタリングする方法と、Amazon Lex のランタイムおよびチャンネル関連付けのメトリクスについて説明します。

トピック

Amazon CloudWatch による Amazon Lex のモニタリング

Amazon Lex ボットのヘルスを追跡するには、Amazon CloudWatch を使用します。CloudWatch では、アカウントの個別の Amazon Lex オペレーションまたはグローバルな Amazon Lex オペレーションのメトリクスを取得できます。定義したしきい値を 1 つ以上のメトリクスが超えたときに通知するよう CloudWatch アラームを設定することもできます。例えば、指定期間中にボットに送信されたリクエスト数をモニタリングして、成功したリクエストのレイテンシーを確認し、エラー数がしきい値を超えた場合はアラームを生成できます。

Amazon Lex 用の CloudWatch メトリクス

Amazon Lex オペレーションのメトリクスを取得するには、以下の情報を指定する必要があります。

- メトリクスディメンション。ディメンションは、メトリクスを識別するための名前と値のペアのセットです。Amazon Lex には 3 つのディメンションがあります。
  - `BotAlias, BotName, Operation`
  - `BotAlias, BotName, InputMode, Operation`
  - `BotName, BotVersion, InputMode, Operation`
- メトリクス名 (`MissedUtteranceCount`、`RuntimeRequestCount` など)。

Amazon Lexのメトリクスは、AWS Management Console、AWS CLI、または CloudWatch API で取得できます。CloudWatch API は、いずれかの Amazon AWS Software Development Kit (SDK) または Amazon CloudWatch API ツールでも使用できます。Amazon Lex コンソールには、CloudWatch API のraw データに基づいてグラフが表示されます。

CloudWatch で Amazon Lex をモニタリングするには、適切な CloudWatch アクセス権限が必要です。詳細については、「Amazon CloudWatch ユーザーガイド」の「Authentication and Access Control for Amazon CloudWatch」(Amazon CloudWatch に対する認証とアクセスコントロール) を参照してください。

Amazon Lex メトリクスの表示

Amazon Lex コンソールまたは CloudWatch コンソールを使用して、Amazon Lex メトリクスを表示します。

メトリクスを表示するには (Amazon Lex コンソール)

1. AWS Management Console にサインインし、Amazon Lex コンソール (https://console.aws.amazon.com/lex/) を開きます。
2. ボットのリストから、メトリクスを表示する対象のボットを選択します。
3. [モニタリング] を選択します。メトリクスがグラフに表示されます。

メトリクスを表示する方法 (CloudWatch コンソール)

1. AWS Management Console にサインインして、CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. [Metrics] で、[All Metrics]、[AWS/Lex] の順に選択します。
3. ディメンションを選択してメトリクスの名前を選んだら、[Add to graph] (グラフへ追加) を選択します。
4. 日付範囲の値を選択します。選択した日付範囲のメトリクスカウントがグラフに表示されます。

アラームを作成する

CloudWatch アラームは指定期間中に単一のメトリクスを監視し、1 つ以上のアクションを実行して Amazon Simple Notification Service (Amazon SNS) トピックまたは Auto Scaling ポリシーに通知を送信します。アクションは、複数の指定期間にわたって特定のしきい値を基準としたメトリクスの値に応じて実行されます。アラームの状態が変わったときにも、CloudWatch は Amazon SNS メッセージを送信できます。

CloudWatch アラームがアクションを呼び出すのは、状態が変わってから指定期間が経過するまで、その新しい状態が続いた場合に限ります。

アラームを設定するには

1. AWS Management Console にサインインして、CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. [Alarms]、[Create Alarm] の順に選択します。
3. [AWS/Lex Metrics] を選択し、メトリクスを選択します。
4. [Time Range] (時間の範囲) で、モニタリングする期間を選択し、[Next] (次へ) を選択します。
5. [Name] (名前) と [Description] (説明) を入力します。
6. [Whenever] (以下のときは毎回) で [>=] を選択し、最大値を入力します。
7. アラーム状態に達したときに CloudWatch から E メールを送信する場合は、[Actions] (アクション) セクションの [Whenever this alarm] (アラームが次の時) で、[State is ALARM] (状態: 警告) を選択します。[通知の送信先] でメーリングリストを選択するか、[新しいリスト] を選択して新しいメーリングリストを作成します。
8. [Alarm Preview] (アラームの確認) セクションでアラームをプレビューします。アラームに問題がなければ、[Create Alarm] (アラームの作成) を選択します。

Amazon Lex ランタイムの CloudWatch メトリクス

次の表は、Amazon Lex のランタイムメトリクスを示しています。

メトリクス	説明
KendraIndexAccessError	Amazon Lex がお客様の Amazon Kendra インデックスにアクセスできなかった回数。 Text または Speech InputMode での PostContent オペレーションの有効なディメンション: • BotName, BotAlias, Operation, InputMode PostText オペレーションの有効なディメンション: • BotName, BotAlias, Operation 単位: 個

メトリクス	説明
KendraLatency	AMAZON.KendraSearchIntent からのリクエストに Amazon Kendra が応答するまでの時間です。 Text または Speech InputMode での PostContent オペレーションの有効なディメンション: • BotName, BotVersion, Operation, InputMode • BotName, BotAlias, Operation, InputMode PostText オペレーションの有効なディメンション: • BotName, BotVersion, Operation • BotName, BotAlias, Operation 単位: ミリ秒
KendraSuccess	AMAZON.KendraSearchIntent から Amazon Kendra インデックスへの成功したリクエストの数。 Text または Speech InputMode での PostContent オペレーションの有効なディメンション: • BotName, BotVersion, Operation, InputMode • BotName, BotAlias, Operation, InputMode PostText オペレーションの有効なディメンション: • BotName, BotVersion, Operation • BotName, BotAlias, Operation 単位: 個

メトリクス	説明
KendraSystemErrors	Amazon Lex が Amazon Kendra インデックスをクエリできなかった回数。 Text または Speech InputMode での PostContent オペレーションの有効なディメンション: • BotName, BotAlias, Operation, InputMode PostText オペレーションの有効なディメンション: • BotName, BotAlias, Operation 単位: 個
KendraThrottledEvents	Amazon Kendra が AMAZON.KendraSearchIntent からのリクエストをスロットルした回数。 Text または Speech InputMode での PostContent オペレーションの有効なディメンション: • BotName, BotAlias, Operation, InputMode PostText オペレーションの有効なディメンション: • BotName, BotAlias, Operation 単位: 個

メトリクス	説明
MissedUtteranceCount	指定期間中に認識されなかった発話の数。 Text または Speech InputMode での PostContent オペレーションの有効なディメンション: • BotName, BotVersion, Operation, InputMode • BotName, BotAlias, Operation, InputMode PostText オペレーションの有効なディメンション: • BotName, BotVersion, Operation • BotName, BotAlias, Operation
RuntimeConcurrency	指定期間中の同時接続の数。RuntimeConcurrency は StatisticSet として報告されます。 Text または Speech InputMode での PostContent オペレーションの有効なディメンション: • Operation, BotName, BotVersion, InputMode • Operation, BotName, BotAlias, InputMode 他のオペレーションの有効なディメンション: • Operation, BotName, BotVersion • Operation, BotName, BotAlias 単位: 個

メトリクス	説明
RuntimeInvalidLambdaResponses	指定期間中の無効な AWS Lambda (Lambda) レスポンスの数。 Text または Speech InputMode での PostContent オペレーションの有効なディメンション: • BotName, BotAlias, Operation, InputMode PostText オペレーションの有効なディメンション: • BotName, BotAlias, Operation
RuntimeLambdaErrors	指定期間中の Lambda ランタイムエラーの数。 PostContent または Text Speech での InputMode オペレーションの有効なディメンション: • BotName, BotAlias, Operation, InputMode PostText オペレーションの有効なディメンション: • BotName, BotAlias, Operation
RuntimePollyErrors	指定期間中の無効な Amazon Polly レスポンスの数。 Text または Speech InputMode での PostContent オペレーションの有効なディメンション: • BotName, BotAlias, Operation, InputMode PostText オペレーションの有効なディメンション: • BotName, BotAlias, Operation

メトリクス	説明
RuntimeRequestCount	指定期間中のランタイムリクエストの数。 Text または Speech InputMode での PostContent オペレーションの有効なディメンション: • BotName, BotVersion, Operation, InputMode • BotName, BotAlias, Operation, InputMode PostText オペレーションの有効なディメンション: • BotName, BotVersion, Operation • BotName, BotAlias, Operation 単位: 個
RuntimeSucessfulRequestLatency **Important** このメトリクスは RuntimeSuccessfulRequestLatency で、RuntimeSuccessfulRequestLatency ではありません。	リクエストが送信された時間からレスポンスが返された時間までに成功したリクエストのレイテンシー。 Text または Speech InputMode での PostContent オペレーションの有効なディメンション: • BotName, BotVersion, Operation, InputMode • BotName, BotAlias, Operation, InputMode PostText オペレーションの有効なディメンション: • BotName, BotVersion, Operation • BotName, BotAlias, Operation 単位: ミリ秒

メトリクス	説明
RuntimeSystemErrors	指定期間中のシステムエラーの数。システムエラーのレスポンスコード範囲は 500〜599 です。 Text または Speech InputMode での PostContent オペレーションの有効なディメンション: • BotName, BotAlias, Operation, InputMode PostText オペレーションの有効なディメンション: • BotName, BotAlias, Operation 単位: 個
RuntimeThrottledEvents	スロットルされたリクエストの数。Amazon Lex は、1 秒あたりに受け取るトランザクションの数がアカウントに設定された制限数を超えると、リクエストをスロットルします。アカウントに設定された制限を頻繁に超える場合は、制限の引き上げをリクエストできます。引き上げをリクエストするには、「AWS サービス制限」を参照してください。 Text または Speech InputMode での PostContent オペレーションの有効なディメンション: • BotName、BotAlias、Operation、InputMode PostText オペレーションの有効なディメンション: • BotName, BotAlias, Operation 単位: 個

メトリクス	説明
RuntimeUserErrors	指定期間中のユーザーエラーの数。ユーザーエラーのレスポンスコード範囲は 400〜499 です。 Text または Speech InputMode での PostContent オペレーションの有効なディメンション: • BotName, BotAlias, Operation, InputMode PostText オペレーションの有効なディメンション: • BotName, BotAlias, Operation 単位: 個

Amazon Lex ランタイムメトリクスは AWS/Lex 名前空間を使用して、以下のディメンションのメトリクスを提供します。メトリクスは、CloudWatch コンソールのディメンション別にグループ化できます。

ディメンション	説明
BotName, BotAlias, Operation, InputMode	ボットのエイリアス、ボットの名前、オペレーション (PostContent)、および入力がテキストまたは音声であるかに基づいて、メトリクスをグループ化します。
BotName, BotVersion, Operation, InputMode	ボットの名前、ボットのバージョン、オペレーション (PostContent)、および入力がテキストまたは音声であるかに基づいて、メトリクスをグループ化します。
BotName, BotVersion, Operation	ボットの名前、ボットのバージョン、およびオペレーション PostText に基づいて、メトリクスをグループ化します。
BotName, BotAlias, Operation	ボットの名前、ボットのエイリアス、およびオペレーション PostText に基づいて、メトリクスをグループ化します。

Amazon Lex チャンネルアソシエーションの CloudWatch メトリクス

チャンネル関連付けは、Amazon Lex とメッセージングチャンネル (Facebook など) との関連付けです。次の表は、Amazon Lex のチャンネル関連付けの説明です。

メトリクス	説明
BotChannelAuthErrors	指定期間中にメッセージングチャンネルから返された認証エラーの数。認証エラーは、チャンネルの作成時に提供されたシークレットトークンが無効であるか、期限切れになっていることを示します。
BotChannelConfigurationErrors	指定期間中の設定エラーの数。設定エラーは、チャンネルの 1 つ以上の設定エントリが無効であることを示します。
BotChannelInboundThrottledEvents	メッセージングチャンネルから送信されたメッセージが、指定期間中に Amazon Lex でスロットルされた回数。
BotChannelOutboundThrottledEvents	Amazon Lex からメッセージングチャンネルへのアウトバウンドイベントが指定期間中にスロットルされた回数。
BotChannelRequestCount	指定期間中にチャンネルで送信されたリクエストの数。
BotChannelResponseCardErrors	指定期間中に Amazon Lex がレスポンスカードを投稿できなかった回数。
BotChannelSystemErrors	指定期間中にチャンネルに対して Amazon Lex で発生した内部エラーの数。

Amazon Lex のチャンネル関連付けメトリクスでは AWS/Lex 名前空間を使用し、以下のディメンションのメトリクスを提供します。メトリクスは、CloudWatch コンソールのディメンション別にグループ化できます。

ディメンション	説明
BotAlias, BotChannelName, BotName, Source	ボットのエイリアス、チャンネル名、ボットの名前、トラフィックのソースに基づいてメトリクスをグループ化します。

会話ログの CloudWatch メトリクス

Amazon Lex は、会話ログに次のメトリクスを使用します。

メトリクス	説明
ConversationLogsAudioDeliverySuccess	指定した期間中に S3 バケットに正常に配信されたオーディオログの数。 単位はカウント
ConversationLogsAudioDeliveryFailure	指定した期間中に S3 バケットへの配信に失敗したオーディオログの数。配信エラーは、会話ログ用に構成されたリソースのエラーを示します。エラーには、不十分な IAM アクセス許可、アクセスできない AWS KMS キー、アクセスできない S3 バケットなどがあります。 単位はカウント
ConversationLogsTextDeliverySuccess	指定した期間中に CloudWatch Logs に正常に配信されたテキストログの数。 単位はカウント
ConversationLogsTextDeliveryFailure	指定した期間中に CloudWatch Logs への配信に失敗したテキストログの数。配信エラーは、会話ログ用に構成されたリソースのエラーを示します。エラーには、不十分な IAM アクセス許可、アクセスできない AWS KMS キー、アクセスできない CloudWatch Logs ロググループなどがあります。 単位はカウント

Amazon Lex 会話ログメトリクスは AWS/Lex 名前空間を使用し、以下のディメンションのメトリクスを提供します。メトリクスは、CloudWatch コンソールのディメンション別にグループ化できます。

ディメンション	説明
BotAlias	ボットのエイリアスでメトリクスをグループ化する。
BotName	ボットの名前でメトリクスをグループ化する。
BotVersion	ボットのバージョンごとにメトリクスをグループ化する。

AWS CloudTrail による Amazon Lex API コールのモニタリング

Amazon Lex は、AWS CloudTrail と統合されています。これは、Amazon Lex のユーザー、ロール、または AWS のサービスで実行されたアクションを記録するためのサービスです。CloudTrail は、Amazon Lex コンソールからの呼び出しと Amazon Lex API へのコード呼び出しを含む、Amazon Lex の API コールのサブセットをイベントとしてキャプチャします。証跡を作成する場合は、Amazon Lex のイベントなど、Amazon S3 バケットへの CloudTrail イベントの継続的な配信を有効にすることができます。証跡を設定しない場合でも、CloudTrail コンソールの [Event history] (イベント履歴) で最新のイベントを表示できます。CloudTrail で収集された情報を使用して、Amazon Lex に対するリクエスト、リクエスト元の IP アドレス、リクエスト者、リクエスト日時などの詳細を確認できます。

設定や有効化の方法など、CloudTrail の詳細については、「AWS CloudTrail ユーザーガイド」を参照してください。

CloudTrail 内の Amazon Lex 情報

AWS アカウントを作成すると、そのアカウントに対して CloudTrail が有効になります。Amazon Lexでサポートされているイベントアクティビティが発生すると、そのアクティビティは [Event history] (イベント履歴) の他の AWS のサービスのイベントとともに CloudTrail イベントに記録されます。最近のイベントは、AWS アカウントで表示、検索、ダウンロードできます。詳細については、「Viewing Events with CloudTrail Event History」(CloudTrail イベント履歴でのイベントの表示) を参照してください。

Amazon Lex のイベントなど、AWS アカウントのイベントを継続的に記録するには、証跡を作成します。証跡により、CloudTrail はログファイルを Amazon Simple Storage Service (Amazon S3) バケットに配信できます。デフォルトでは、コンソールで作成した証跡がすべての AWS リージョンに

適用されます。証跡では、AWS パーティションのすべてのリージョンからのイベントがログに記録され、指定した S3 バケットにログファイルが配信されます。さらに、CloudTrail ログで収集したイベントデータをより詳細に分析し、それに基づく対応するためにその他の AWS のサービスを設定できます。詳細については、次を参照してください。

- 証跡を作成するための概要
- CloudTrail のサポート対象サービスと統合
- Amazon SNS の CloudTrail の通知の設定
- 「複数のリージョンから CloudTrail ログファイルを受け取る」および「複数のアカウントから CloudTrail ログファイルを受け取る」

Amazon Lex は、CloudTrail ログファイルのイベントとして以下のオペレーションのログ付けをサポートします。

- CreateBotVersion
- CreateIntentVersion
- CreateSlotTypeVersion
- DeleteBot
- DeleteBotAlias
- DeleteBotChannelAssociation
- DeleteBotVersion
- DeleteIntent
- DeleteIntentVersion
- DeleteSlotType
- DeleteSlotTypeVersion
- DeleteUtterances
- GetBot
- GetBotAlias
- GetBotAliases
- GetBotChannelAssociation
- GetBotChannelAssociations

- GetBots
- GetBotVersions
- GetBuiltinIntent
- GetBuiltinIntents
- GetBuiltinSlotTypes
- GetSlotTypeVersions
- GetUtterancesView
- PutBot
- PutBotAlias
- PutIntent
- PutSlotType

各イベントまたはログエントリには、リクエストの生成者に関する情報が含まれます。この情報は以下のことを確認するのに役立ちます:

- リクエストが、ルートと ユーザー認証情報のどちらを使用して送信されたか
- リクエストが、ロールとフェデレーティッドユーザーのどちらの一時的なセキュリティ認証情報を使用して送信されたか
- リクエストが、別の AWS のサービスによって送信されたかどうか

詳細については、CloudTrail userIdentity エレメントを参照してください。

CloudTrail ログに記録される Amazon Lex アクションの詳細については、「Amazon Lex Model·Building·Service」を参照してください。例えば PutBot、GetBot、の各 DeleteBot オペレーションへのコールは、CloudTrail ログファイル内にエントリを生成します。「Amazon Lex Runtime Service」、「PostContent」、「PostText」に記載されているアクションはログに記録されません。

例: Amazon Lex ログファイルのエントリ

証跡は、指定した S3 バケットにイベントをログファイルとして配信するように設定できます。CloudTrail ログファイルには、1 つ以上のログエントリがあります。イベントはあらゆるソースからの単一のリクエストを表し、リクエストされたアクション、アクションの日時、リクエストのパラメータなどの情報が含まれます。CloudTrail ログファイルは、公開 API コールの順序付けられたスタックトレースではないため、特定の順序では表示されません。

次の CloudTrail ログエントリの例は、PutBot オペレーションへの呼び出しの結果を示しています。

```
   {
                "eventVersion": "1.05",
                "userIdentity": {
                    "type": "AssumedRole | FederatedUser | IAMUser | Root | SAMLUser |
WebIdentityUser",
                    "principalId": "principal ID",
                    "arn": "ARN",
                    "accountId": "account ID",
                    "accessKeyId": "access key ID",
                    "userName": "user name"
                },
                "eventTime": "timestamp",
                "eventSource": "lex.amazonaws.com",
                "eventName": "PutBot",
                "awsRegion": "region",
                "sourceIPAddress": "source IP address",
                "userAgent": "user agent",
                "requestParameters": {
                    "name": "CloudTrailBot",
                    "intents": [
                        {
                            "intentVersion": "11",
                            "intentName": "TestCloudTrail"
                        }
                    ],
                    "voiceId": "Salli",
                    "childDirected": false,
                    "locale": "en-US",
                    "idleSessionTTLInSeconds": 500,
                    "processBehavior": "BUILD",
                    "description": "CloudTrail test bot",
                    "clarificationPrompt": {
                        "messages": [
                            {
                                "contentType": "PlainText",
                                "content": "I didn't understand you. What would you
like to do?"
                            }
                        ],
                        "maxAttempts": 2
                    },
                    "abortStatement": {
```

```
                        "messages": [
                            {
                                "contentType": "PlainText",
                                "content": "Sorry. I'm not able to assist at this
time."
                            }
                        ]
                    }
                },
                "responseElements": {
                    "voiceId": "Salli",
                    "locale": "en-US",
                    "childDirected": false,
                    "abortStatement": {
                        "messages": [
                            {
                                "contentType": "PlainText",
                                "content": "Sorry. I'm not able to assist at this
time."
                            }
                        ]
                    },
                    "status": "BUILDING",
                    "createdDate": "timestamp",
                    "lastUpdatedDate": "timestamp",
                    "idleSessionTTLInSeconds": 500,
                    "intents": [
                        {
                            "intentVersion": "11",
                            "intentName": "TestCloudTrail"
                        }
                    ],
                    "clarificationPrompt": {
                        "messages": [
                            {
                                "contentType": "PlainText",
                                "content": "I didn't understand you. What would you
like to do?"
                            }
                        ],
                        "maxAttempts": 2
                    },
                    "version": "$LATEST",
                    "description": "CloudTrail test bot",
```

```
                "checksum": "checksum",
                "name": "CloudTrailBot"
            },
            "requestID": "request ID",
            "eventID": "event ID",
            "eventType": "AwsApiCall",
            "recipientAccountId": "account ID"
        }
    }
```

Amazon Lex のコンプライアンス検証

サードパーティーの監査者は、さまざまな コンプライアンスプログラムの一環として Amazon Lex のセキュリティと AWS コンプライアンスを評価します。Amazon Lex が HIPAA 対応サービスに。これは、PCI、SOC、および ISO に準拠しています。サードパーティーの監査レポートは、 を使用してダウンロードできます AWS Artifact。詳細については、「Downloading Reports in AWS Artifact」(AWS Artifact のレポートのダウンロード) を参照してください。

Amazon Lex を使用する際のお客様のコンプライアンス責任は、組織のデータの機密性や組織のコンプライアンス目的、適用可能な法律、規制によって決定されます。Amazon Lex の使用が PCI などのコンプライアンスに準拠していることを前提としている場合、 AWS は役立つ以下のリソースを提供します。

- セキュリティとコンプライアンスのクイックスタートガイド — アーキテクチャ上の考慮事項について説明し、セキュリティとコンプライアンスに重点を置いたベースライン環境を にデプロイするための手順を説明するデプロイガイド AWS
- HIPAA のセキュリティとコンプライアンスに関するホワイトペーパーを作成する - このホワイトペーパーでは、企業が AWS を使用して HIPAA 準拠のアプリケーションを作成する方法について説明します。
- AWS コンプライアンスのリソース – お客様の業界や場所に適用される可能性があるワークブックとガイドのコレクション。
- AWS Config – 自社プラクティス、業界ガイドライン、および規制に対するリソースの設定の準拠状態を評価するサービス。
- AWS Security Hub — セキュリティ業界標準およびベストプラクティスへの準拠を確認するのに役立つ AWS 内のセキュリティ状態の包括的なビュー

特定のコンプライアンスプログラム AWS の対象となるサービスのリストについては、「コンプライアンスプログラムによる AWS 対象範囲内のサービス」を参照してください。一般的な情報については、「AWS コンプライアンスプログラム」を参照してください。

Amazon Lex の耐障害性

AWS グローバルインフラストラクチャは、 AWS リージョンとアベイラビリティーゾーンを中心に構築されています。 AWS リージョンは、低レイテンシー、高スループット、および高度の冗長ネットワークで接続されている複数の物理的に独立および隔離されたアベイラビリティーゾーンを提供します。アベイラビリティーゾーンでは、アベイラビリティーゾーン間で中断せずに、自動的にフェイルオーバーするアプリケーションとデータベースを設計および運用することができます。アベイラビリティーゾーンは、従来の単一または複数のデータセンターインフラストラクチャよりも可用性、耐障害性、およびスケーラビリティが優れています。

AWS リージョンとアベイラビリティーゾーンの詳細については、AWS 「 グローバルインフラストラクチャ」を参照してください。

Amazon Lex では、 AWS グローバルインフラストラクチャに加えて、データの耐障害性とバックアップのニーズに対応できるように複数の機能を提供しています。

Amazon Lex のインフラストラクチャセキュリティ

マネージドサービスである Amazon Lex は、ホワイトペーパー「アマゾン ウェブ サービス: セキュリティプロセスの概要」に記載されている AWS グローバルネットワークセキュリティの手順で保護されています。

が公開した AWS API コールを使用して、ネットワーク経由で Amazon Lex にアクセスします。クライアントは TLS(Transport Layer Security)1.0 をサポートしている必要があります。TLS 1.2 以降が推奨されています。また、Ephemeral Diffie-Hellman (DHE) や Elliptic Curve Ephemeral Diffie-Hellman (ECDHE) などの Perfect Forward Secrecy (PFS) を使用した暗号スイートもクライアントでサポートされている必要があります。これらのモードは、Java 7 以降など、最近のほとんどのシステムでサポートされています。また、リクエストには、アクセスキー ID と、IAM プリンシパルに関連付けられているシークレットアクセスキーを使用して署名する必要があります。または、AWS Security Token Service (AWS STS) を使用して、一時的なセキュリティ認証情報を生成し、リクエストに署名することもできます。

これらの API オペレーションは任意のネットワークの場所から呼び出すことができますが、Amazon Lex ではリソースベースのアクセスポリシーがサポートされています。これには送信元 IP アドレス

に基づく制限を含めることができます。また、Amazon Lex ポリシーを使用して、特定の Amazon 仮想プライベートクラウド (Amazon VPC) エンドポイントまたは特定の VPC からのアクセスを管理することもできます。これにより、実質的に ネットワーク内の特定の VPC からの特定の Amazon Lex リソースへの AWS ネットワークアクセスが分離されます。

Amazon Lex のガイドラインとクォータ

以下のセクションでは、Amazon Lex を使用する際のガイドラインとクォータについて説明します。

トピック

サポートされるリージョン

Amazon Lex が利用可能な AWS リージョンの一覧については、「Amazon Web Services General Reference」(Amazon ウェブサービス全般のリファレンス)の「AWS Regions and Endpoints」(AWS リージョンとエンドポイント) を参照してください。

一般的なガイドライン

このセクションでは、Amazon Lex を使用する際の一般的なガイドラインについて説明します。

- リクエストの署名 – Amazon Lex のすべてのモデル構築および、「API リファレンス」で説明されているランタイム API オペレーションは、署名 V4 を使用してリクエストを認証します。リクエストの認証の詳細については、『』の「署名バージョン 4 の署名プロセスAmazon Web Services 全般のリファレンス」を参照してください。

  PostContent では、Amazon Lex は「Amazon Simple Storage Service (S3) API リファレンス」の「Signature Calculations for the Authorization Header: Transferring Payload in a Single Chunk」(AWS Signature Version 4) で説明している未署名のペイロードオプションを使用します。

  未署名のペイロードオプションを使用する場合は、ペイロードのハッシュを正規リクエストに含めないでください。代わりに、リテラル文字列の「UNSIGNED-PAYLOAD」をペイロードのハッシュとして使用します。また、ヘッダーを名前 `x-amz-content-sha256` および値 `UNSIGNED-PAYLOAD` で `PostContent` に含めます。

- Amazon Lex でユーザーの発話からスロット値をキャプチャする方法について以下の点に注意してください。

  Amazon Lex では、ユーザーがスロットタイプ定義で指定した列挙値を使用して機械学習モデルをトレーニングします。次のサンプル発話で GetPredictionIntent というインテントを定義したとします。

  ```
  "Tell me the prediction for {Sign}"
  ```

  {Sign} はカスタムタイプ ZodiacSign のスロットです。これには列挙値が 12 個 (Aries〜Pisces) あります。「次の運勢を教えてください: ...」というユーザーの発話を受け取ると、Amazon Lex はこの後に星座が続くと予測します。

  PutSlotType オペレーションを使用して valueSelectionStrategy フィールドを ORIGINAL_VALUE に設定するか、コンソールで [Expand values] (値の拡張) を選択した場合にユーザーが「次の運勢を教えてください: 大地」と言うと、Amazon Lex では「大地」を ZodiacSign と推論し、それをクライアントアプリケーションまたは Lambda 関数に渡します。スロット値をフルフィルメントアクティビティで使用する前に、それが有効な値であることを確認する必要があります。

  PutSlotType オペレーションを使用して valueSelectionStrategy フィールドを TOP_RESOLUTION に設定するか、コンソールで [Restrict to slot values and synonyms] を選択すると、スロットタイプに定義済みの値のみが返されます。例えば、ユーザーが「次の運勢を教えてください: 地球」と言うと、この値はスロットタイプの定義済みの値に含まれていないため、値として認識されません。スロット値のシノニムを定義すると、スロット値と同じように認識されますが、シノニムの代わりにスロット値が返されます。

  Amazon Lex から Lambda 関数を呼び出すか、クライアントアプリケーションとの会話の結果を返す場合、スロット値の大文字と小文字の区別は保証されません。例えば、AMAZON.Movie の組

み込みスロットタイプに対する値を引き出す場合、ユーザーが「Gone with the wind」と言うか入力すると、Amazon Lex が返す値は「Gone with the Wind」、「gone with the wind」、または「Gone With The Wind」になることがあります。テキストでのやり取りの場合、スロット値の大文字と小文字は、`valueResolutionStrategy` フィールドの値に応じて入力したテキストまたはスロット値と一致します。

- 頭字語を含むスロット値を定義する場合は、次のパターンを使用します。
  - ピリオドで区切られた大文字 (D.V.D.)
  - スペースで区切られた大文字 (D V D)

- Amazon Lex では、Alexa Skills Kit でサポートされている組み込みスロットタイプ AMAZON.LITERAL はサポートしていません。ただし、Amazon Lex では、この機能を実装するために使用できるカスタムスロットタイプの作成をサポートしています。前の箇条書きで説明したように、カスタムスロットタイプ定義外の値をキャプチャできます。自動音声認識 (ASR) と自然言語理解 (NLU) の精度を向上させるには、さらに多様な列挙値を追加します。

- 組み込みスロットタイプの AMAZON.DATE と AMAZON.TIME では、絶対的な日時と相対的な日時の両方をキャプチャします。相対的な日時は、Amazon Lex がリクエストを処理しているリージョンで解決されます。

  組み込みスロットタイプ `AMAZON.TIME` で、ユーザーが時間を午前か午後かを指定しない場合、時間があいまいであるため、Amazon Lex はユーザーに再度時間を指定するよう求めます。プロンプトは、絶対時間を引き出すように使うことをお勧めします。例えば、「ピザはいつお届けしますか? 6 PM または夕方 6 時のように指定できます」というプロンプトを使います。

- ボットで紛らわしいトレーニングデータを提供すると、Amazon Lex がユーザー入力を理解する能力が低下します。次の例を検討してください。

  ボットに 2 つのインテント (`OrderPizza` と `OrderDrink`) があり、両方に「私は注文したい」という発話が設定されているとします。この発話は、ボットの言語モデルの構築時に Amazon Lex

が学習できる特定のインテントにはマッピングされません。その結果、この発話をユーザーがランタイムに入力すると、Amazon Lex は高い信頼性におけるインテントを選択できません。

別の例として、ユーザーから確認を得るためのカスタムインテント (`MyCustomConfirmationIntent` など) を定義し、このインテントに「はい」と「いいえ」の発話を設定するとします。Amazon Lex はまた、ユーザーの確認について認識するための言語モデルがあることに注意してください。これにより、競合する状況が生じます。ユーザーが「はい」と答えた場合、これは従来のインテントに対する確認なのか、新しく作成されたカスタムインテントをユーザーがリクエストしているのかが不明です。

通常、サンプルの発話を指定する場合は、特定のインテントや (必要に応じて) 特定のスロット値にマッピングする必要があります。

- ランタイム API オペレーションの PostContent と PostText では、必須パラメーターとしてユーザー ID を使用します。開発者は、これを API で説明されている制約を満たす任意の値に設定できます。このパラメーターを使用してユーザーのログイン情報、メールアドレス、社会保障番号などの機密情報を送信しないでください。この ID は、ボットの会話を一意に識別するために使用します (ピザの注文者は複数いる場合があります)。

- クライアントアプリケーションで Amazon Cognito を認証に使用する場合は、Amazon Cognito のユーザー ID を Amazon Lex のユーザー ID として使用できます。ボットに設定されたすべての Lambda 関数には、Amazon Lex が代わりに Lambda 機能を呼び出すユーザーを識別できる独自の認証方法を保持している必要があります。

- ユーザーの意図をキャプチャして会話を中止するインテントを定義することをお勧めします。例えば、サンプルの発話 (「何も必要ない」、「終了」、「バイバイ」) を使用してインテント (`NothingIntent`) を定義できます。スロットやコードフックとしての Lambda 関数は設定されません。これにより、ユーザーは適切に会話を終了できます。

クォータ

このセクションでは、Amazon Lex の現在のクォータについて説明します。これらのクォータはカテゴリ別にグループ化されています。

Service Quotas は調整または増やすことができます。AWS カスタマーサポートに問い合わせて、クォータの引き上げをリクエストします。サービスクォータの引き上げには、最大数日かかることがあります。大規模なプロジェクトの一部としてクォータの引き上げを行う場合は、必ずこの時間を計画に入れてください。

トピック

ランタイム Service Quotas

API リファレンスで説明しているクォータに加えて、以下の点に注意してください。

API クォータ

- PostContent オペレーションに入力できる音声の長さは最長 15 秒です。

- ランタイム API オペレーションの PostContent と PostText に入力できるテキストのサイズは、最大 1,024 文字 (Unicode) です。

- `PostContent` ヘッダーの最大サイズは 16 KB です。リクエストとセッションヘッダーを組み合わせた最大サイズは 12 KB です。

- テキストモードで `PostContent` または `PostText` オペレーションを使用する場合、ボットとの最大同時会話数は `$LATEST` エイリアスで 2、それ以外のエイリアスで 50 です。クォータは API ごとに個別に適用されます。

- 音声モードで PostContent オペレーションを行う場合、ボットとのテキストモードでの最大同時会話数は、$LATEST エイリアスの場合は 2、その他のエイリアスの場合は 125 となっています。クォータは API ごとに個別に適用されます。

- セッション管理コール (PutSession、GetSession、DeleteSession) の最大同時接続数は、ボットの $LATEST エイリアスが 2、それ以外のエイリアスが 50 です。

- Lambda 関数への最大入力サイズは 12 KB です。最大出力サイズは 25 KB であり、そのうち 12 KB はセッション属性に使用できます。

$LATEST バージョンの使用

- $LATEST バージョンのボットは、手動テストにのみ使用してください。Amazon Lex では、$LATEST バージョンのボットに対するランタイムリクエストの数が制限されています。

- $LATEST バージョンのボットを更新すると、Amazon Lex は $LATEST バージョンのボットを使用しているクライアントアプリケーションで進行中のあらゆる会話を終了します。$LATEST バージョンは更新される場合があるため、通常、本番稼働環境では $LATEST バージョンのボットを使用しないでください。代わりに、別のバージョンを発行して使用します。

- エイリアスを更新すると、Amazon Lex に変更が反映されるまでに数分かかります。ボットの $LATEST バージョンを変更すると、その変更はすぐに反映されます。

セッションタイムアウト

- ボットの作成時に設定したセッションタイムアウトにより、現在のユーザーのインテントやスロットデータなどの会話コンテキストをボットで保持する期間が決まります。

- ユーザーがボットで会話を開始してからセッションが期限切れになるまで、Amazon Lex は同じボットバージョンを使用します (別のバージョンを参照するようにボットのエイリアスを更新した場合でも)。

モデル構築のクォータ

モデル構築とは、ボットの作成と管理のことです。これには、ボット、インテント、スロットタイプ、スロット、ボットチャンネル関連付けの作成と管理などが含まれます。

トピック

ボットのクォータ

- モデル構築 API を使用してプロンプトとステートメントを設定します。各プロンプトやステートメントには最大 5 つのメッセージが設定でき、各メッセージには 1 ~ 1,000 文字 (UTF-8) を含めることができます。

- メッセージグループを使用すると、各メッセージに最大 5 つのメッセージグループを定義できます。各メッセージグループには最大 5 つのメッセージを入れることができます。また、メッセージグループ全体のメッセージ数は合計 15 までに制限されています。

- インテントとスロットのサンプル発話を定義できます。すべての発話を合わせて最大 200,000 文字を使用できます。

- 各カスタムスロットタイプでは、最大 10,000 個の値とシノニムを定義できます。各ボットには、最大 50,000 個のスロットタイプ値とシノニムを含めることができます。

- ボット、エイリアス、およびボットチャンネル関連付けの名前は、作成時に大文字と小文字が区別されません。PizzaBot を作成して pizzaBot を作成しようとすると、エラーが発生します。ただし、リソースにアクセスする場合、リソース名は大文字と小文字が区別されるため、pizzaBot ではなく、PizzaBot と指定する必要があります。これらの名前に使用できる文字数は 2 ~ 50 文字 (ASCII) です。

- すべてのリソースタイプに発行できるバージョンの最大数は 100 です。エイリアスのバージョニングはないことに注意してください。

- ボット内では、インテント名とスロット名は一意である必要があります (インテントとスロットに同じ名前は使用できません)。

- ボットを作成して、複数のインテントをサポートするように設定できます。2 つのインテントが同じ名前のスロットを含む場合、対応するスロットタイプは同じである必要があります。

  例えば、2 つのインテント (OrderPizza と OrderDrink) をサポートするボットを作成したとします。これらのインテントが両方とも size スロットを含む場合、そのスロットタイプは両方のインテントで同じである必要があります。

  さらに、いずれか一方のインテントのスロットにサンプル発話を設定すると、他方のインテントの同じ名前のスロットにもそれが適用されます。

- ボットには最大 250 個のインテントを関連付けることができます。

- ボットの作成時にセッションタイムアウトを指定します。セッションタイムアウトは 1 分 ~ 1 日の間で指定できます。デフォルトは 5 分です。

- ボットには最大 5 つのエイリアスを作成できます。

- AWS アカウントにつき最大 250 個のボットを作成できます。

- 同じ組み込みインテントから拡張されたインテントを複数作成することはできません。

インテントのクォータ

- インテントとスロットの名前は、作成時に大文字と小文字が区別されません。つまり、OrderPizza インテントを作成した後で、さらに別の orderPizza インテントを作成しようとすると、エラーが発生します。ただし、これらのリソースにアクセスする場合、リソース名は大文字と小文字が区別されるため、orderPizza ではなく OrderPizza と指定する必要があります。これらの名前に使用できる文字数は 1 ~ 100 文字 (ASCII) です。

- インテントには最大 1,500 個のサンプル発話を設定できます。最低 1 つのサンプル発話が必要です。各サンプル発話の最大長は 200 UTF-8 文字です。ボット内のすべてのインテントとスロットを合わせた発話には、最大 200,000 文字を使用できます。インテントのサンプル発話では以下のことができます。
  - 0 個以上のスロット名が参照できます。
  - スロット名は 1 回のみ参照できます。

  例:

```
I want a pizza
I want a {pizzaSize} pizza
I want a {pizzaSize} {pizzaTopping} pizza
```

- 各インテントは最大 1,500 の発話をサポートしますが、発話の使用数が少ないほど、指定した分以外の入力を Amazon Lex が認識する能力が高まる場合があります。

- インテントの各メッセージには最大 5 つのメッセージグループを作成することができます。メッセージのメッセージグループ全体で許可されているメッセージの合計数は 15 件です。

- コンソールは `conclusionStatement` と `followUpPrompt` メッセージのグループのみを作成できます。Amazon Lex API を使用して他のメッセージ用にメッセージグループを作成することができます。

- 各スロットには最大 10 個のサンプル発話を設定できます。各サンプル発話は、スロット名を厳密に 1 回だけ参照する必要があります。例:

```
{pizzaSize} please
```

- 各ボットは、インテントとスロットの発話を合わせて最大 200,000 文字を使用できます。

- 組み込みのインテントから拡張されたインテントには発話を設定できません。これ以外のすべてのインテントには、少なくとも 1 つのサンプル発話を設定する必要があります。インテントにはスロットが含まれていますが、スロットレベルのサンプル発話はオプションです。

- 組み込みのインテント
  - 現在、Amazon Lex では組み込みのインテントにスロットを引き出すことはサポートされていません。Lambda 関数を作成し、組み込みのインテントから派生したインテントを使用して、レスポンスで `ElicitSlot` ディレクティブを返すことはできません。詳細については、「レスポンスの形式」を参照してください。
  - このサービスでは、組み込みのインテントにサンプル発話を追加することはサポートされていません。同様に、組み込みのインテントに対してスロットを追加または削除することはできません。

- AWS アカウントにつき最大 1,000 個のインテントを作成できます。1 つのインテントには最大 100 個のスロットを作成できます。

スロットタイプのクォータ

- スロットタイプの名前は、作成時に大文字と小文字が区別されません。PizzaSize スロットタイプを作成した後で、さらに pizzaSize スロットタイプを作成しようとすると、エラーが発生します。ただし、リソースにアクセスするときに、リソース名は大文字と小文字が区別されます (PizzaSize を指定する必要があります。pizzaSize では指定できません)。名前は、1 ~ 100 ASCII 文字にする必要があります。

- カスタムスロットタイプには、最大 10,000 個の列挙値とシノニムを設定できます。各値の最大長は 140 UTF-8 文字です。列挙値とシノニムは重複するものを含めることができません。

- スロットタイプ値については、必要に応じて大文字と小文字の両方を指定します。例えば、Procedure というスロットタイプで、値が MRI の場合、「MRI」と「mri」の両方を値として指定します。

- 組み込みスロットタイプ – 現在、Amazon Lex では、組み込みスロットタイプに列挙値またはシノニムを追加することはサポートされていません。

API リファレンス

このセクションでは、Amazon Lex の API オペレーションについて説明します。Amazon Lex が利用可能な AWS リージョンの一覧については「Amazon Web Services General Reference」(Amazon ウェブサービス全般のリファレンス) の「AWS Regions and Endpoints」(AWS リージョンおよびエンドポイント) を参照してください。

トピック

- アクション
- データ型

アクション

次のアクションは、Amazon Lex Model·Building·Service でサポートされています。

- CreateBotVersion
- CreateIntentVersion
- CreateSlotTypeVersion
- DeleteBot
- DeleteBotAlias
- DeleteBotChannelAssociation
- DeleteBotVersion
- DeleteIntent
- DeleteIntentVersion
- DeleteSlotType
- DeleteSlotTypeVersion
- DeleteUtterances
- GetBot
- GetBotAlias
- GetBotAliases
- GetBotChannelAssociation
- GetBotChannelAssociations

- GetBots
- GetBotVersions
- GetBuiltinIntent
- GetBuiltinIntents
- GetBuiltinSlotTypes
- GetExport
- GetImport
- GetIntent
- GetIntents
- GetIntentVersions
- GetMigration
- GetMigrations
- GetSlotType
- GetSlotTypes
- GetSlotTypeVersions
- GetUtterancesView
- ListTagsForResource
- PutBot
- PutBotAlias
- PutIntent
- PutSlotType
- StartImport
- StartMigration
- TagResource
- UntagResource

次のアクションは、Amazon Lex Runtime Service でサポートされています

- DeleteSession
- GetSession
- PostContent

- PostText
- PutSession

Amazon Lex Model Building Service

次のアクションは、Amazon Lex Model·Building·Service でサポートされています。

- CreateBotVersion
- CreateIntentVersion
- CreateSlotTypeVersion
- DeleteBot
- DeleteBotAlias
- DeleteBotChannelAssociation
- DeleteBotVersion
- DeleteIntent
- DeleteIntentVersion
- DeleteSlotType
- DeleteSlotTypeVersion
- DeleteUtterances
- GetBot
- GetBotAlias
- GetBotAliases
- GetBotChannelAssociation
- GetBotChannelAssociations
- GetBots
- GetBotVersions
- GetBuiltinIntent
- GetBuiltinIntents
- GetBuiltinSlotTypes
- GetExport
- GetImport
- GetIntent

- GetIntents
- GetIntentVersions
- GetMigration
- GetMigrations
- GetSlotType
- GetSlotTypes
- GetSlotTypeVersions
- GetUtterancesView
- ListTagsForResource
- PutBot
- PutBotAlias
- PutIntent
- PutSlotType
- StartImport
- StartMigration
- TagResource
- UntagResource

CreateBotVersion

サービス: Amazon Lex Model Building Service

指定されたボットの $LATEST バージョンに基づいて新しいバージョンを作成します。このリソースの $LATEST バージョンが、最後のバージョンを作成してから変更されていない場合、Amazon Lex は新しいバージョンを作成しません。最後に作成したバージョンを返します。

> **Note**
>
> ボットのバージョンで更新できるのは $LATEST バージョンのみです。CreateBotVersion オペレーションを使用して作成した番号付きのバージョンを更新することはできません。

ボットの最初のバージョンを作成すると、Amazon Lex はバージョンを 1 に設定します。それ以降のバージョンは 1 ずつ増えます。詳細については、「バージョニング」を参照してください。

このオペレーションには lex:CreateBotVersion アクションに対するアクセス許可が必要です。

リクエストの構文

```
POST /bots/name/versions HTTP/1.1
Content-type: application/json

{
   "checksum": "string"
}
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

name

作成する新しいバージョンのボットの名前。名前は、大文字と小文字が区別されます。

長さの制限: 最小長は 2 です。最大長は 50 です。

Pattern: ^([A-Za-z]_?)+$

必須: はい

リクエストボディ

リクエストは以下の JSON 形式のデータを受け入れます。

checksum

ボットの $LATEST バージョンの特定のリビジョンを識別します。チェックサムを指定しても、$LATEST バージョンのボットのチェックサムが異なる場合は、PreconditionFailedException 例外が返され、Amazon Lexは新しいバージョンを公開しません。チェックサムを指定しない場合、Amazon Lex は $LATEST バージョンを公開します。

タイプ: 文字列

必須: いいえ

レスポンスの構文

```
HTTP/1.1 201
Content-type: application/json

{
   "abortStatement": {
      "messages": [
         {
            "content": "string",
            "contentType": "string",
            "groupNumber": number
         }
      ],
      "responseCard": "string"
   },
   "checksum": "string",
   "childDirected": boolean,
   "clarificationPrompt": {
      "maxAttempts": number,
      "messages": [
         {
            "content": "string",
            "contentType": "string",
            "groupNumber": number
         }
      ],
```

```
      "responseCard": "string"
   },
   "createdDate": number,
   "description": "string",
   "detectSentiment": boolean,
   "enableModelImprovements": boolean,
   "failureReason": "string",
   "idleSessionTTLInSeconds": number,
   "intents": [
      {
         "intentName": "string",
         "intentVersion": "string"
      }
   ],
   "lastUpdatedDate": number,
   "locale": "string",
   "name": "string",
   "status": "string",
   "version": "string",
   "voiceId": "string"
}
```

レスポンス要素

アクションが成功すると、HTTP 201 レスポンスが返されます。

サービスから以下のデータが JSON 形式で返されます。

abortStatement

Amazon Lex が会話をキャンセルするために使用するメッセージ。詳細については、「PutBot」を参照してください。

型: Statement オブジェクト

checksum

作成されたボットのバージョンを識別するチェックサム。

型: 文字列

childDirected

Amazon Lex Model·Building·Service で作成された各 Amazon Lex ボットについて、Amazon Lex 全体または一部の使用が、13 歳未満の児童を対象とするウェブサイト、プログラム、その他の

アプリケーションに関連しており、COPPA (Children's Online Privacy Protection Act) の対象となっているかどうかを、childDirected フィールドに true または false を指定する必要があります。childDirected フィールドに true を指定することで、Amazon Lex 全体または一部の使用が、13 歳未満の児童を対象とするウェブサイト、プログラム、その他のアプリケーションに関連しており、COPPA の対象になることに同意します。childDirected フィールドに false を指定することで、Amazon Lex 全体または一部の使用が、13 歳未満の児童を対象とするウェブサイト、プログラム、その他のアプリケーションに関連しておらず、COPPA の対象にならないことに同意します。Amazon Lex の全体または一部の使用が、13歳未満の児童を対象にしており、COPPAの対象となるウェブサイト、プログラム、またはその他のアプリケーションに関連するかどうかを正確に設定するために、childDirected フィールドにデフォルト値を指定することはできません。

Amazon Lex 全体または一部の使用が、13 歳未満の児童を対象とするウェブサイト、プログラム、その他のアプリケーションに関連する場合は、COPPA により必要とされる検証可能な保護者の同意が必要です。13 歳未満の児童を対象とするウェブサイト、プログラム、またはその他のアプリケーションに関連する Amazon Lex の全体または一部の使用についての詳細は、「Amazon Lex FAQ」を参照してください。

型: ブール値

clarificationPrompt

Amazon Lex がユーザーのリクエストを理解できない場合に使用するメッセージ。詳細については、「PutBot」を参照してください。

型: Prompt オブジェクト

createdDate

ボットバージョンの作成日。

型: タイムスタンプ

description

ボットの説明。

型: 文字列

長さの制限: 最小長は 0 です。最大長は 200 です。

detectSentiment

ユーザーが入力した発話を Amazon Comprehend に送信してセンチメント分析を行うかどうかを示します。

型: ブール値

enableModelImprovements

ボットが精度向上を使用しているかどうかを示します。`true` はボットが精度向上を使用していることを示し、それ以外は `false` となります。

型: ブール値

failureReason

`status` が `FAILED` の場合、Amazon Lex はボットの構築に失敗した理由を提示します。

型: 文字列

idleSessionTTLInSeconds

Amazon Lex が会話で収集したデータを保持する最大時間 (秒) です。詳細については、「PutBot」を参照してください。

型: 整数

値の範囲: 最小値 は 60 です。最大値は 86400 です。

intents

`Intent` オブジェクトの配列。詳細については、「PutBot」を参照してください。

型: Intent オブジェクトの配列

lastUpdatedDate

このボットの `$LATEST` バージョンが更新された日。

型: タイムスタンプ

locale

ボットのターゲットロケールを指定します。

型: 文字列

有効な値 : de-DE | en-AU | en-GB | en-IN | en-US | es-419 | es-ES | es-US | fr-FR | fr-CA | it-IT | ja-JP | ko-KR

name

ボットの名前。

型: 文字列

長さの制限: 最小長は 2 です。最大長は 50 です。

Pattern: ^([A-Za-z]_?)+$

status

ボットの作成や更新のリクエストを送信すると、Amazon Lex は status レスポンス要素を BUILDING に設定します。Amazon Lex がボットを構築した後、status を READY に設定します。Amazon Lex がボットをビルドできない場合、status を FAILED に設定します。Amazon Lex は、failureReason レスポンス要素に失敗の理由を返します。

型: 文字列

有効な値 : BUILDING | READY | READY_BASIC_TESTING | FAILED | NOT_BUILT

version

取得するボットのバージョンです。

型: 文字列

長さの制限 : 最小長は 1 です。最大長は 64 文字です。

パターン: \$LATEST|[0-9]+

voiceId

Amazon Lex がユーザーとの音声対話に使用する Amazon Polly 音声 ID です。

型: 文字列

エラー

BadRequestException

リクエストが適切にフォーマットされていません。例えば、値が無効であったり、必須項目が設定されていない場合です。フィールドの値を確認して、再度お試しください。

HTTP ステータスコード : 400

ConflictException

リクエストの処理中に競合が発生しました。リクエストを再試行してください。

HTTP ステータスコード: 409

InternalFailureException

Amazon Lex 内部エラーが発生しました。リクエストを再試行してください。

HTTP ステータスコード : 500

LimitExceededException

リクエストが制限を超えました。リクエストを再試行してください。

HTTP ステータスコード: 429

NotFoundException

リクエストで指定されたリソースは見つかりませんでした。リソースを確認して、もう一度お試しください。

HTTP ステータスコード: 404

PreconditionFailedException

変更しようとしているリソースのチェックサムがリクエストのチェックサムと一致しません。リソースのチェックサムを確認して、もう一度お試しください。

HTTP ステータスコード: 412

その他の参照資料

言語固有の AWS SDK のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go バージョン 2 用 SDK
- AWS Java V2 用 SDK

- AWS V3 用 JavaScript SDK
- AWS PHP V3 用SDK
- AWS Python 用 SDK
- AWS ルビー V3 用 SDK

CreateIntentVersion

サービス: Amazon Lex Model Building Service

$LATEST バージョンのインテントに基づいて、インテントの新しいバージョンを作成します。$LATEST バージョンのインテントの最後の更新から変更がない場合、Amazon Lex は新しいバージョンを作成しません。最後に作成したバージョンを返します。

> **Note**
>
> $LATEST バージョンのインテントのみアップデートすることができます。CreateIntentVersion オペレーションを使用して作成した番号付きのバージョンを更新することはできません。

ボットの最初のバージョンを作成すると、Amazon Lex はバージョンを 1 に設定します。それ以降のバージョンは 1 ずつ増えます。詳細については、「バージョニング」を参照してください。

このオペレーションには lex:CreateIntentVersion アクションを実行するアクセス許可が必要です。

リクエストの構文

```
POST /intents/name/versions HTTP/1.1
Content-type: application/json

{
   "checksum": "string"
}
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

name

新しいバージョンを作成するインテントの名前。名前は、大文字と小文字が区別されます。

長さの制限：最小長は 1 です。最大長は 100 です。

パターン: ^([A-Za-z]_?)+$

必須: はい

リクエストボディ

リクエストは以下の JSON 形式のデータを受け入れます。

checksum

新しいバージョンの作成に使用する $LATEST バージョンのインテントのチェックサム。チェックサムを指定しても、$LATEST バージョンのインテントのチェックサムが異なる場合、Amazon Lex は PreconditionFailedException の例外を返し、新しいバージョンを公開しません。チェックサムを指定しない場合、Amazon Lex は $LATEST バージョンを公開します。

タイプ: 文字列

必須: いいえ

レスポンスの構文

```
HTTP/1.1 201
Content-type: application/json

{
   "checksum": "string",
   "conclusionStatement": {
      "messages": [
         {
            "content": "string",
            "contentType": "string",
            "groupNumber": number
         }
      ],
      "responseCard": "string"
   },
   "confirmationPrompt": {
      "maxAttempts": number,
      "messages": [
         {
            "content": "string",
            "contentType": "string",
            "groupNumber": number
         }
      ],
      "responseCard": "string"
   },
```

```
    "createdDate": number,
    "description": "string",
    "dialogCodeHook": {
       "messageVersion": "string",
       "uri": "string"
    },
    "followUpPrompt": {
       "prompt": {
          "maxAttempts": number,
          "messages": [
             {
                "content": "string",
                "contentType": "string",
                "groupNumber": number
             }
          ],
          "responseCard": "string"
       },
       "rejectionStatement": {
          "messages": [
             {
                "content": "string",
                "contentType": "string",
                "groupNumber": number
             }
          ],
          "responseCard": "string"
       }
    },
    "fulfillmentActivity": {
       "codeHook": {
          "messageVersion": "string",
          "uri": "string"
       },
       "type": "string"
    },
    "inputContexts": [
       {
          "name": "string"
       }
    ],
    "kendraConfiguration": {
       "kendraIndex": "string",
       "queryFilterString": "string",
```

```
      "role": "string"
   },
   "lastUpdatedDate": number,
   "name": "string",
   "outputContexts": [
      {
         "name": "string",
         "timeToLiveInSeconds": number,
         "turnsToLive": number
      }
   ],
   "parentIntentSignature": "string",
   "rejectionStatement": {
      "messages": [
         {
            "content": "string",
            "contentType": "string",
            "groupNumber": number
         }
      ],
      "responseCard": "string"
   },
   "sampleUtterances": [ "string" ],
   "slots": [
      {
         "defaultValueSpec": {
            "defaultValueList": [
               {
                  "defaultValue": "string"
               }
            ]
         },
         "description": "string",
         "name": "string",
         "obfuscationSetting": "string",
         "priority": number,
         "responseCard": "string",
         "sampleUtterances": [ "string" ],
         "slotConstraint": "string",
         "slotType": "string",
         "slotTypeVersion": "string",
         "valueElicitationPrompt": {
            "maxAttempts": number,
            "messages": [
```

```
                {
                   "content": "string",
                   "contentType": "string",
                   "groupNumber": number
                }
             ],
             "responseCard": "string"
          }
       }
    ],
    "version": "string"
}
```

レスポンス要素

アクションが成功すると、HTTP 201 レスポンスが返されます。

サービスから以下のデータが JSON 形式で返されます。

checksum

作成されたインテントバージョンのチェックサム。

型: 文字列

conclusionStatement

`fulfillmentActivity` フィールドで指定された Lambda 関数がインテントを達成すると、Amazon Lex はこのステートメントをユーザーに伝えます。

型: Statement オブジェクト

confirmationPrompt

これが定義されている場合、Amazon Lex がユーザーのインテントを達成する前の確認に使用するプロンプトです。

型: Prompt オブジェクト

createdDate

インテントが作成された日付。

型: タイムスタンプ

description

インテントの説明。

型: 文字列

長さの制限: 最小長は 0 です。最大長は 200 です。

dialogCodeHook

これが定義されている場合、Amazon Lex は各ユーザー入力に対してこの Lambda 関数を呼び出します。

型: CodeHook オブジェクト

followUpPrompt

これが定義されている場合、Amazon Lex はこのプロンプトを使用して、インテントが達成された後に追加のユーザーアクティビティを要求します。

型: FollowUpPrompt オブジェクト

fulfillmentActivity

インテントがどのように達成されるのかを説明します。

型: FulfillmentActivity オブジェクト

inputContexts

Amazon Lex がユーザーとの会話の中でインテントを選択するためのアクティブなコンテキストをリストアップした `InputContext` オブジェクトの配列です。

型: InputContext オブジェクトの配列

配列メンバー：最小数は 0 項目です。最大数は 5 項目です。

kendraConfiguration

Amazon Kendra インデックスと `AMAZON.KendraSearchIntent` インテントを接続するための設定情報 (項目がある場合)。

型: KendraConfiguration オブジェクト

lastUpdatedDate

インテントが更新された日付。

型: タイムスタンプ

name

インテントの名前。

型: 文字列

長さの制限: 最小長は 1 です。最大長は 100 です。

パターン: ^([A-Za-z]_?)+$

outputContexts

OutputContext オブジェクトの配列で、インテントが達成されたときにアクティブになるコンテキストを列挙します。

型: OutputContext オブジェクトの配列

の配列メンバー: 最小数は 0 項目です。最大数は 10 項目です。

parentIntentSignature

組み込みインテントの一意の識別子。

型: 文字列

rejectionStatement

ユーザーが confirmationPrompt で定義された質問に「いいえ」と答えた場合、Amazon Lex はインテントがキャンセルされたことを確認するためにこのステートメントを返します。

型: Statement オブジェクト

sampleUtterances

インテント用に構成されたサンプル発話の配列。

型: 文字列の配列

配列メンバー：最小数は 0 項目です。最大数は 1500 項目です。

長さの制限：最小長は 1 です。最大長は 200 です。

slots

インテントを達成するために必要な情報を定義するスロットタイプの配列。

型: Slot オブジェクトの配列

配列メンバー: 最小数は 0 項目です。最大数は 100 項目です。

version

インテントの新しいバージョンに割り当てられたバージョン番号。

型: 文字列

長さの制限：最小長は 1 です。最大長は 64 文字です。

パターン：\$LATEST|[0-9]+

エラー

BadRequestException

リクエストが適切にフォーマットされていません。例えば、値が無効であったり、必須項目が設定されていない場合です。フィールドの値を確認して、再度お試しください。

HTTP ステータスコード：400

ConflictException

リクエストの処理中に競合が発生しました。リクエストを再試行してください。

HTTP ステータスコード: 409

InternalFailureException

Amazon Lex 内部エラーが発生しました。リクエストを再試行してください。

HTTP ステータスコード：500

LimitExceededException

リクエストが制限を超えました。リクエストを再試行してください。

HTTP ステータスコード: 429

NotFoundException

リクエストで指定されたリソースは見つかりませんでした。リソースを確認して、もう一度お試しください。

HTTP ステータスコード: 404

PreconditionFailedException

変更しようとしているリソースのチェックサムがリクエストのチェックサムと一致しません。リソースのチェックサムを確認して、もう一度お試しください。

HTTP ステータスコード: 412

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go バージョン 2 用 SDK
- AWS Java V2 用 SDK
- AWS V3 用 JavaScript SDK
- AWS PHP V3 用SDK
- AWS Python 用 SDK
- AWS ルビー V3 用 SDK

CreateSlotTypeVersion

サービス: Amazon Lex Model Building Service

指定された $LATEST バージョンのスロットタイプに基づいて新しいバージョンを作成します。このリソースの $LATEST バージョンが最後に作成されたバージョンから変更がない場合、Amazon Lex は新しいバージョンを作成しません。最後に作成したバージョンを返します。

> **Note**
>
> スロットタイプの $LATEST バージョンのみ更新できます。CreateSlotTypeVersion オペレーションを使用して作成した番号付きのバージョンを更新することはできません。

スロットタイプのバージョンを作成すると、Amazon Lex はバージョンを 1 に設定します。それ以降のバージョンは 1 ずつ増えます。詳細については、「バージョニング」を参照してください。

このオペレーションには、lex:CreateSlotTypeVersion アクションに対する許可が必要です。

リクエストの構文

```
POST /slottypes/name/versions HTTP/1.1
Content-type: application/json

{
   "checksum": "string"
}
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

name

新しいバージョンを作成するスロットタイプの名前。名前は、大文字と小文字が区別されます。

長さの制限：最小長は 1 です。最大長は 100 です。

パターン: ^([A-Za-z]_?)+$

必須: はい

リクエストボディ

リクエストは以下の JSON 形式のデータを受け入れます。

checksum

公開したいスロットタイプの $LATEST バージョンのチェックサム。チェックサムを指定しても、$LATEST バージョンのスロットタイプのチェックサムが異なる場合、Amazon Lex は PreconditionFailedException の例外を返し、新しいバージョンを公開しません。チェックサムを指定しない場合、Amazon Lex は $LATEST バージョンを公開します。

タイプ: 文字列

必須: いいえ

レスポンスの構文

```
HTTP/1.1 201
Content-type: application/json

{
   "checksum": "string",
   "createdDate": number,
   "description": "string",
   "enumerationValues": [
      {
         "synonyms": [ "string" ],
         "value": "string"
      }
   ],
   "lastUpdatedDate": number,
   "name": "string",
   "parentSlotTypeSignature": "string",
   "slotTypeConfigurations": [
      {
         "regexConfiguration": {
            "pattern": "string"
         }
      }
   ],
   "valueSelectionStrategy": "string",
   "version": "string"
```

```
}
```

レスポンス要素

アクションが成功すると、HTTP 201 レスポンスが返されます。

サービスから以下のデータが JSON 形式で返されます。

checksum

スロットタイプの $LATEST バージョンのチェックサム。

型: 文字列

createdDate

スロットタイプが作成された日付。

型: タイムスタンプ

description

スロットタイプの説明。

型: 文字列

長さの制限: 最小長は 0 です。最大長は 200 です。

enumerationValues

スロットタイプが取得できる値を定義する EnumerationValue オブジェクトのリスト。

型: EnumerationValue オブジェクトの配列

配列メンバー : 最小数は 0 項目です。最大数は 10000 項目です。

lastUpdatedDate

スロットタイプが更新された日付。リソースを作成する場合、作成日と最終更新日は同じ日付になります。

型: タイムスタンプ

name

スロットタイプの名前。

型: 文字列

長さの制限: 最小長は 1 です。最大長は 100 です。

パターン: ^([A-Za-z]_?)+$

parentSlotTypeSignature

このスロットタイプの親として使用される組み込みスロットタイプです。

型: 文字列

長さの制限: 最小長は 1 です。最大長は 100 です。

パターン: ^((AMAZON\.)_?|[A-Za-z]_?)+

slotTypeConfigurations

親組み込みスロットタイプを拡張する構成情報。

型: SlotTypeConfiguration オブジェクトの配列

の配列メンバー: 最小数は 0 項目です。最大数は 10 項目です。

valueSelectionStrategy

Amazon Lex がスロットの価値を決定するために使用する戦略。詳細については、「PutSlotType」を参照してください。

型: 文字列

有効な値 : ORIGINAL_VALUE | TOP_RESOLUTION

version

新しいスロットタイプバージョンに割り当てられたバージョン。

型: 文字列

長さの制限 : 最小長は 1 です。最大長は 64 文字です。

パターン : \$LATEST|[0-9]+

エラー

BadRequestException

リクエストが適切にフォーマットされていません。例えば、値が無効であったり、必須項目が設定されていない場合です。フィールドの値を確認して、再度お試しください。

HTTP ステータスコード : 400

ConflictException

リクエストの処理中に競合が発生しました。リクエストを再試行してください。

HTTP ステータスコード: 409

InternalFailureException

Amazon Lex 内部エラーが発生しました。リクエストを再試行してください。

HTTP ステータスコード : 500

LimitExceededException

リクエストが制限を超えました。リクエストを再試行してください。

HTTP ステータスコード: 429

NotFoundException

リクエストで指定されたリソースは見つかりませんでした。リソースを確認して、もう一度お試しください。

HTTP ステータスコード: 404

PreconditionFailedException

変更しようとしているリソースのチェックサムがリクエストのチェックサムと一致しません。リソースのチェックサムを確認して、もう一度お試しください。

HTTP ステータスコード: 412

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go バージョン 2 用 SDK
- AWS Java V2 用 SDK
- AWS V3 用 JavaScript SDK

- AWS PHP V3 用SDK
- AWS Python 用 SDK
- AWS ルビー V3 用 SDK

DeleteBot

サービス: Amazon Lex Model Building Service

$LATEST バージョンを含む、すべてのバージョンのボットを削除します。特定のバージョンのボットを削除するには、DeleteBotVersion オペレーションを行います。DeleteBot のオペレーションでは、ボットのスキーマはすぐには削除されません。その代わり、削除マークをつけておき、後で削除します。

Amazon Lex は、ユーザーの入力に対するボットの応答能力を向上させるために、発話を無期限に保存します。これらの発話は、ボットが削除されても削除されません。発話を削除するには、DeleteUtterances オペレーションを使用します。

ボットにエイリアスがある場合、削除することはできません。その代わり、DeleteBot オペレーションは、ボットを参照するエイリアスへの参照を含む ResourceInUseException 例外を返します。ボットへの参照を削除するには、エイリアスを削除します。同じ例外が再び発生した場合は、DeleteBot オペレーションが成功するまで、参照元のエイリアスを削除してください。

このオペレーションには、lex:DeleteBot アクションに対する許可が必要です。

リクエストの構文

```
DELETE /bots/name HTTP/1.1
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

name

ボットの名前。名前は、大文字と小文字が区別されます。

長さの制限: 最小長は 2 です。最大長は 50 です。

Pattern: ^([A-Za-z]_?)+$

必須: はい

リクエストボディ

リクエストにリクエスト本文がありません。

レスポンスの構文

```
HTTP/1.1 204
```

レスポンス要素

アクションが成功した場合、サービスは空の HTTP 本文を持つ HTTP 204 レスポンスを返します。

エラー

BadRequestException

リクエストが適切にフォーマットされていません。例えば、値が無効であったり、必須項目が設定されていない場合です。フィールドの値を確認して、再度お試しください。

HTTP ステータスコード：400

ConflictException

リクエストの処理中に競合が発生しました。リクエストを再試行してください。

HTTP ステータスコード: 409

InternalFailureException

Amazon Lex 内部エラーが発生しました。リクエストを再試行してください。

HTTP ステータスコード：500

LimitExceededException

リクエストが制限を超えました。リクエストを再試行してください。

HTTP ステータスコード: 429

NotFoundException

リクエストで指定されたリソースは見つかりませんでした。リソースを確認して、もう一度お試しください。

HTTP ステータスコード: 404

ResourceInUseException

削除しようとしているリソースは、別のリソースによって参照されています。この情報を使用して、削除するリソースへの参照を解除します。

例外の本文には、リソースを記述する JSON オブジェクトが含まれています。

```
{ "resourceType": BOT | BOTALIAS | BOTCHANNEL | INTENT,

"resourceReference": {

"name": string, "version": string } }
```

HTTP ステータスコード : 400

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go バージョン 2 用 SDK
- AWS Java V2 用 SDK
- AWS V3 用 JavaScript SDK
- AWS PHP V3 用SDK
- AWS Python 用 SDK
- AWS ルビー V3 用 SDK

DeleteBotAlias

サービス: Amazon Lex Model Building Service

指定されたボットのエイリアスを削除します。

ボットとメッセージングチャンネルの関連付けに使用されているエイリアスを削除することはできません。チャンネルアソシエーションにエイリアスが使用されている場合、DeleteBot オペレーションは、ボットを参照しているチャンネルアソシエーションへの参照を含む ResourceInUseException 例外を返します。チャンネルの関連付けを削除することで、エイリアスへの参照を削除することができます。同じ例外が再び発生した場合は、DeleteBotAlias オペレーションが成功するまで参照元のアソシエーションを削除してください。

リクエストの構文

```
DELETE /bots/botName/aliases/name HTTP/1.1
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

botName

エイリアスが指すボットの名前。

長さの制限: 最小長は 2 です。最大長は 50 です。

Pattern: ^([A-Za-z]_?)+$

必須：はい

name

削除するエイリアスの名前。名前は、大文字と小文字が区別されます。

長さの制限：最小長は 1 です。最大長は 100 です。

パターン: ^([A-Za-z]_?)+$

必須: はい

リクエストボディ

リクエストにリクエスト本文がありません。

レスポンスの構文

```
HTTP/1.1 204
```

レスポンス要素

アクションが成功した場合、サービスは空の HTTP 本文を持つ HTTP 204 レスポンスを返します。

エラー

BadRequestException

リクエストが適切にフォーマットされていません。例えば、値が無効であったり、必須項目が設定されていない場合です。フィールドの値を確認して、再度お試しください。

HTTP ステータスコード：400

ConflictException

リクエストの処理中に競合が発生しました。リクエストを再試行してください。

HTTP ステータスコード: 409

InternalFailureException

Amazon Lex 内部エラーが発生しました。リクエストを再試行してください。

HTTP ステータスコード：500

LimitExceededException

リクエストが制限を超えました。リクエストを再試行してください。

HTTP ステータスコード: 429

NotFoundException

リクエストで指定されたリソースは見つかりませんでした。リソースを確認して、もう一度お試しください。

HTTP ステータスコード: 404

ResourceInUseException

削除しようとしているリソースは、別のリソースによって参照されています。この情報を使用して、削除するリソースへの参照を解除します。

例外の本文には、リソースを記述する JSON オブジェクトが含まれています。

```
{ "resourceType": BOT | BOTALIAS | BOTCHANNEL | INTENT,

"resourceReference": {

"name": string, "version": string } }
```

HTTP ステータスコード : 400

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go バージョン 2 用 SDK
- AWS Java V2 用 SDK
- AWS V3 用 JavaScript SDK
- AWS PHP V3 用SDK
- AWS Python 用 SDK
- AWS ルビー V3 用 SDK

DeleteBotChannelAssociation

サービス: Amazon Lex Model Building Service

Amazon Lex ボットとメッセージングプラットフォームの間の関連付けを削除します。

このオペレーションには `lex:DeleteBotChannelAssociation` アクションに対するアクセス許可が必要です。

リクエストの構文

```
DELETE /bots/botName/aliases/aliasName/channels/name HTTP/1.1
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

aliasName

この関連付けが作成されている Amazon Lex ボットの特定のバージョンを指すエイリアス。

長さの制限：最小長は 1 です。最大長は 100 です。

パターン: `^([A-Za-z]_?)+$`

必須：はい

botName

Amazon Lex ボットの名前。

長さの制限: 最小長は 2 です。最大長は 50 です。

Pattern: `^([A-Za-z]_?)+$`

必須：はい

name

関連付けの名前。名前は、大文字と小文字が区別されます。

長さの制限：最小長は 1 です。最大長は 100 です。

パターン: `^([A-Za-z]_?)+$`

必須: はい

リクエストボディ

リクエストにリクエスト本文がありません。

レスポンスの構文

```
HTTP/1.1 204
```

レスポンス要素

アクションが成功した場合、サービスは空の HTTP 本文を持つ HTTP 204 レスポンスを返します。

エラー

BadRequestException

リクエストが適切にフォーマットされていません。例えば、値が無効であったり、必須項目が設定されていない場合です。フィールドの値を確認して、再度お試しください。

HTTP ステータスコード：400

ConflictException

リクエストの処理中に競合が発生しました。リクエストを再試行してください。

HTTP ステータスコード: 409

InternalFailureException

Amazon Lex 内部エラーが発生しました。リクエストを再試行してください。

HTTP ステータスコード：500

LimitExceededException

リクエストが制限を超えました。リクエストを再試行してください。

HTTP ステータスコード: 429

NotFoundException

リクエストで指定されたリソースは見つかりませんでした。リソースを確認して、もう一度お試しください。

HTTP ステータスコード: 404

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go バージョン 2 用 SDK
- AWS Java V2 用 SDK
- AWS V3 用 JavaScript SDK
- AWS PHP V3 用SDK
- AWS Python 用 SDK
- AWS ルビー V3 用 SDK

DeleteBotVersion

サービス: Amazon Lex Model Building Service

指定されたバージョンのボットを削除します。ボットのすべてのバージョンを削除するには、DeleteBot オペレーションを使用します。

このオペレーションには、lex:DeleteBotVersion アクションに対する許可が必要です。

リクエストの構文

```
DELETE /bots/name/versions/version HTTP/1.1
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

name

ボットの名前。

長さの制限: 最小長は 2 です。最大長は 50 です。

Pattern: ^([A-Za-z]_?)+$

必須：はい

version

削除するボットのバージョン。$LATEST バージョンのボットを削除することはできません。$LATEST バージョンを削除するには、DeleteBot オペレーションを行います。

長さの制限：最小長は 1 です。最大長は 64 文字です。

パターン: [0-9]+

必須: はい

リクエストボディ

リクエストにリクエスト本文がありません。

レスポンスの構文

```
HTTP/1.1 204
```

レスポンス要素

アクションが成功した場合、サービスは空の HTTP 本文を持つ HTTP 204 レスポンスを返します。

エラー

BadRequestException

リクエストが適切にフォーマットされていません。例えば、値が無効であったり、必須項目が設定されていない場合です。フィールドの値を確認して、再度お試しください。

HTTP ステータスコード：400

ConflictException

リクエストの処理中に競合が発生しました。リクエストを再試行してください。

HTTP ステータスコード: 409

InternalFailureException

Amazon Lex 内部エラーが発生しました。リクエストを再試行してください。

HTTP ステータスコード：500

LimitExceededException

リクエストが制限を超えました。リクエストを再試行してください。

HTTP ステータスコード: 429

NotFoundException

リクエストで指定されたリソースは見つかりませんでした。リソースを確認して、もう一度お試しください。

HTTP ステータスコード: 404

ResourceInUseException

削除しようとしているリソースは、別のリソースによって参照されています。この情報を使用して、削除するリソースへの参照を解除します。

例外の本文には、リソースを記述する JSON オブジェクトが含まれています。

```
{ "resourceType": BOT | BOTALIAS | BOTCHANNEL | INTENT,

"resourceReference": {

"name": string, "version": string } }
```

HTTP ステータスコード : 400

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go バージョン 2 用 SDK
- AWS Java V2 用 SDK
- AWS V3 用 JavaScript SDK
- AWS PHP V3 用SDK
- AWS Python 用 SDK
- AWS ルビー V3 用 SDK

DeleteIntent

サービス: Amazon Lex Model Building Service

$LATEST バージョンを含む、すべてのバージョンのインテントを削除します。特定のバージョンのインテントを削除するには、DeleteIntentVersion のオペレーションを行います。

インテントのバージョンを削除できるのは、インテントが参照されていない場合のみです。1 つ以上のボット (Amazon Lex: 仕組み を参照してください) で参照されているインテントを削除するには、まずそれらの参照を削除する必要があります。

> **Note**
>
> ResourceInUseException の例外が発生した場合は、インテントがどこで参照されているかを示す参照例を提供します。インテントへの参照を削除するには、ボットを更新するか、ボットを削除します。再度インテントを削除しようとしたときに同じ例外が発生した場合は、インテントの参照先がなくなり、DeleteIntent への呼び出しが成功するまで繰り返します。

このオペレーションには lex:DeleteIntent アクションに対するアクセス許可が必要です。

リクエストの構文

```
DELETE /intents/name HTTP/1.1
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

name

インテントの名前。名前は、大文字と小文字が区別されます。

長さの制限：最小長は 1 です。最大長は 100 です。

パターン: ^([A-Za-z]_?)+$

必須: はい

リクエストボディ

リクエストにリクエスト本文がありません。

レスポンスの構文

```
HTTP/1.1 204
```

レスポンス要素

アクションが成功した場合、サービスは空の HTTP 本文を持つ HTTP 204 レスポンスを返します。

エラー

BadRequestException

リクエストが適切にフォーマットされていません。例えば、値が無効であったり、必須項目が設定されていない場合です。フィールドの値を確認して、再度お試しください。

HTTP ステータスコード：400

ConflictException

リクエストの処理中に競合が発生しました。リクエストを再試行してください。

HTTP ステータスコード: 409

InternalFailureException

Amazon Lex 内部エラーが発生しました。リクエストを再試行してください。

HTTP ステータスコード：500

LimitExceededException

リクエストが制限を超えました。リクエストを再試行してください。

HTTP ステータスコード: 429

NotFoundException

リクエストで指定されたリソースは見つかりませんでした。リソースを確認して、もう一度お試しください。

HTTP ステータスコード: 404

ResourceInUseException

削除しようとしているリソースは、別のリソースによって参照されています。この情報を使用して、削除するリソースへの参照を解除します。

例外の本文には、リソースを記述する JSON オブジェクトが含まれています。

```
{ "resourceType": BOT | BOTALIAS | BOTCHANNEL | INTENT,

"resourceReference": {

"name": string, "version": string } }
```

HTTP ステータスコード : 400

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go バージョン 2 用 SDK
- AWS Java V2 用 SDK
- AWS V3 用 JavaScript SDK
- AWS PHP V3 用SDK
- AWS Python 用 SDK
- AWS ルビー V3 用 SDK

DeleteIntentVersion

サービス: Amazon Lex Model Building Service

指定されたバージョンのインテントを削除します。あるインテントのすべてのバージョンを削除するには、DeleteIntent オペレーションを使用します。

このオペレーションには、lex:DeleteIntentVersion アクションに対する許可が必要です。

リクエストの構文

```
DELETE /intents/name/versions/version HTTP/1.1
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

name

インテントの名前。

長さの制限：最小長は 1 です。最大長は 100 です。

パターン: ^([A-Za-z]_?)+$

必須：はい

version

削除するインテントのバージョン。$LATEST バージョンのインテントを削除することはできません。$LATEST バージョンを削除するには、DeleteIntent オペレーションを行います。

長さの制限：最小長は 1 です。最大長は 64 文字です。

パターン: [0-9]+

必須: はい

リクエストボディ

リクエストにリクエスト本文がありません。

レスポンスの構文

```
HTTP/1.1 204
```

レスポンス要素

アクションが成功した場合、サービスは空の HTTP 本文を持つ HTTP 204 レスポンスを返します。

エラー

BadRequestException

リクエストが適切にフォーマットされていません。例えば、値が無効であったり、必須項目が設定されていない場合です。フィールドの値を確認して、再度お試しください。

HTTP ステータスコード：400

ConflictException

リクエストの処理中に競合が発生しました。リクエストを再試行してください。

HTTP ステータスコード: 409

InternalFailureException

Amazon Lex 内部エラーが発生しました。リクエストを再試行してください。

HTTP ステータスコード：500

LimitExceededException

リクエストが制限を超えました。リクエストを再試行してください。

HTTP ステータスコード: 429

NotFoundException

リクエストで指定されたリソースは見つかりませんでした。リソースを確認して、もう一度お試しください。

HTTP ステータスコード: 404

ResourceInUseException

削除しようとしているリソースは、別のリソースによって参照されています。この情報を使用して、削除するリソースへの参照を解除します。

例外の本文には、リソースを記述する JSON オブジェクトが含まれています。

```
{ "resourceType": BOT | BOTALIAS | BOTCHANNEL | INTENT,

"resourceReference": {

"name": string, "version": string } }
```

HTTP ステータスコード : 400

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go バージョン 2 用 SDK
- AWS Java V2 用 SDK
- AWS V3 用 JavaScript SDK
- AWS PHP V3 用SDK
- AWS Python 用 SDK
- AWS ルビー V3 用 SDK

DeleteSlotType

サービス: Amazon Lex Model Building Service

$LATEST バージョンを含む、スロットタイプのすべてのバージョンを削除します。スロットタイプの特定のバージョンを削除するには、DeleteSlotTypeVersion のオペレーションを行います。

スロットタイプのバージョンは、参照されていない場合にのみ削除できます。1 つ以上のインテントで参照されるスロットタイプを削除するには、まずそれらの参照を解除する必要があります。

> **Note**
>
> ResourceInUseException の例外が発生した場合は、スロットタイプがどこで参照されているかを示す参照例を提供します。スロットタイプへの参照を解除するには、ボットを更新するか、ボットを削除します。もう一度、スロットタイプを削除しようとしたときに同じ例外が発生した場合は、スロットタイプの参照先がすべて解除され、DeleteSlotType への呼び出しが成功するまで繰り返します。

このオペレーションには lex:DeleteSlotType アクションに対するアクセス許可が必要です。

リクエストの構文

```
DELETE /slottypes/name HTTP/1.1
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

name

スロットタイプの名前。名前は、大文字と小文字が区別されます。

長さの制限：最小長は 1 です。最大長は 100 です。

パターン: ^([A-Za-z]_?)+$

必須: はい

リクエストボディ

リクエストにリクエスト本文がありません。

レスポンスの構文

```
HTTP/1.1 204
```

レスポンス要素

アクションが成功した場合、サービスは空の HTTP 本文を持つ HTTP 204 レスポンスを返します。

エラー

BadRequestException

リクエストが適切にフォーマットされていません。例えば、値が無効であったり、必須項目が設定されていない場合です。フィールドの値を確認して、再度お試しください。

HTTP ステータスコード：400

ConflictException

リクエストの処理中に競合が発生しました。リクエストを再試行してください。

HTTP ステータスコード: 409

InternalFailureException

Amazon Lex 内部エラーが発生しました。リクエストを再試行してください。

HTTP ステータスコード：500

LimitExceededException

リクエストが制限を超えました。リクエストを再試行してください。

HTTP ステータスコード: 429

NotFoundException

リクエストで指定されたリソースは見つかりませんでした。リソースを確認して、もう一度お試しください。

HTTP ステータスコード: 404

ResourceInUseException

削除しようとしているリソースは、別のリソースによって参照されています。この情報を使用して、削除するリソースへの参照を解除します。

例外の本文には、リソースを記述する JSON オブジェクトが含まれています。

```
{ "resourceType": BOT | BOTALIAS | BOTCHANNEL | INTENT,

"resourceReference": {

"name": string, "version": string } }
```

HTTP ステータスコード : 400

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go バージョン 2 用 SDK
- AWS Java V2 用 SDK
- AWS V3 用 JavaScript SDK
- AWS PHP V3 用SDK
- AWS Python 用 SDK
- AWS ルビー V3 用 SDK

DeleteSlotTypeVersion

サービス: Amazon Lex Model Building Service

指定されたバージョンのスロットタイプを削除します。あるスロットタイプのすべてのバージョンを削除するには、DeleteSlotType オペレーションを行います。

このオペレーションには、lex:DeleteSlotTypeVersion アクションに対する許可が必要です。

リクエストの構文

```
DELETE /slottypes/name/version/version HTTP/1.1
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

name

スロットタイプの名前。

長さの制限：最小長は 1 です。最大長は 100 です。

パターン: ^([A-Za-z]_?)+$

必須：はい

version

削除するスロットタイプのバージョン。スロットタイプの $LATEST バージョンは削除できません。$LATEST バージョンを削除するには、DeleteSlotType オペレーションを行います。

長さの制限：最小長は 1 です。最大長は 64 文字です。

パターン: [0-9]+

必須: はい

リクエストボディ

リクエストにリクエスト本文がありません。

レスポンスの構文

```
HTTP/1.1 204
```

レスポンス要素

アクションが成功した場合、サービスは空の HTTP 本文を持つ HTTP 204 レスポンスを返します。

エラー

BadRequestException

リクエストが適切にフォーマットされていません。例えば、値が無効であったり、必須項目が設定されていない場合です。フィールドの値を確認して、再度お試しください。

HTTP ステータスコード：400

ConflictException

リクエストの処理中に競合が発生しました。リクエストを再試行してください。

HTTP ステータスコード: 409

InternalFailureException

Amazon Lex 内部エラーが発生しました。リクエストを再試行してください。

HTTP ステータスコード：500

LimitExceededException

リクエストが制限を超えました。リクエストを再試行してください。

HTTP ステータスコード: 429

NotFoundException

リクエストで指定されたリソースは見つかりませんでした。リソースを確認して、もう一度お試しください。

HTTP ステータスコード: 404

ResourceInUseException

削除しようとしているリソースは、別のリソースによって参照されています。この情報を使用して、削除するリソースへの参照を解除します。

例外の本文には、リソースを記述する JSON オブジェクトが含まれています。

```
{ "resourceType": BOT | BOTALIAS | BOTCHANNEL | INTENT,

"resourceReference": {

"name": string, "version": string } }
```

HTTP ステータスコード : 400

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go バージョン 2 用 SDK
- AWS Java V2 用 SDK
- AWS V3 用 JavaScript SDK
- AWS PHP V3 用SDK
- AWS Python 用 SDK
- AWS ルビー V3 用 SDK

DeleteUtterances

サービス: Amazon Lex Model Building Service

保存された発話を削除します。

Amazon Lex は、ユーザーがボットに送信する発話を保存します。発話は、GetUtterancesView のオペレーションを使用するために15日間保存され、その後、ユーザーの入力に反応するボットの能力を向上させるために無期限に保存されます。

DeleteUtterances オペレーションを使用して、特定のユーザーの保存された発話を手動で削除します。DeleteUtterances オペレーションを行うと、ユーザーの入力に対するボットの応答能力を向上させるために保存されていた発話は、直ちに削除されます。GetUtterancesView オペレーションで使用するために保存された発話は 15 日後に削除されます。

このオペレーションには、lex:DeleteUtterances アクションに対する許可が必要です。

リクエストの構文

```
DELETE /bots/botName/utterances/userId HTTP/1.1
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

botName

発話を格納したボットの名前。

長さの制限: 最小長は 2 です。最大長は 50 です。

Pattern: ^([A-Za-z]_?)+$

必須：はい

userId

発話をしたユーザーの一意の識別子。これは、PostContentPostText発話を含むまたは操作リクエストで送信されたユーザーIDです。

長さの制限: 最小長は 2 です。最大長は 100 です。

必須: はい

リクエストボディ

リクエストにリクエスト本文がありません。

レスポンスの構文

```
HTTP/1.1 204
```

レスポンス要素

アクションが成功した場合、サービスは空の HTTP 本文を持つ HTTP 204 レスポンスを返します。

エラー

BadRequestException

リクエストが適切にフォーマットされていません。例えば、値が無効であったり、必須項目が設定されていない場合です。フィールドの値を確認して、再度お試しください。

HTTP ステータスコード：400

InternalFailureException

Amazon Lex 内部エラーが発生しました。リクエストを再試行してください。

HTTP ステータスコード：500

LimitExceededException

リクエストが制限を超えました。リクエストを再試行してください。

HTTP ステータスコード: 429

NotFoundException

リクエストで指定されたリソースは見つかりませんでした。リソースを確認して、もう一度お試しください。

HTTP ステータスコード: 404

その他の参照資料

この API を言語固有の AWS SDK で使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go バージョン 2 用 SDK
- AWS Java V2 用 SDK
- AWS V3 用 JavaScript SDK
- AWS PHP V3 用SDK
- AWS Python 用 SDK
- AWS ルビー V3 用 SDK

GetBot

サービス: Amazon Lex Model Building Service

特定のボットのメタデータ情報を返します。ボットの名前、ボットのバージョンまたはエイリアスを指定する必要があります。

このオペレーションには、lex:GetBot アクションに対する許可が必要です。

リクエストの構文

```
GET /bots/name/versions/versionoralias HTTP/1.1
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

name

ボットの名前。名前は、大文字と小文字が区別されます。

長さの制限: 最小長は 2 です。最大長は 50 です。

Pattern: ^([A-Za-z]_?)+$

必須：はい

versionoralias

ボットのバージョンまたはエイリアス。

必須: はい

リクエストボディ

リクエストにリクエスト本文がありません。

レスポンスの構文

```
HTTP/1.1 200
Content-type: application/json

{
   "abortStatement": {
      "messages": [
```

```
            {
               "content": "string",
               "contentType": "string",
               "groupNumber": number
            }
         ],
         "responseCard": "string"
      },
      "checksum": "string",
      "childDirected": boolean,
      "clarificationPrompt": {
         "maxAttempts": number,
         "messages": [
            {
               "content": "string",
               "contentType": "string",
               "groupNumber": number
            }
         ],
         "responseCard": "string"
      },
      "createdDate": number,
      "description": "string",
      "detectSentiment": boolean,
      "enableModelImprovements": boolean,
      "failureReason": "string",
      "idleSessionTTLInSeconds": number,
      "intents": [
         {
            "intentName": "string",
            "intentVersion": "string"
         }
      ],
      "lastUpdatedDate": number,
      "locale": "string",
      "name": "string",
      "nluIntentConfidenceThreshold": number,
      "status": "string",
      "version": "string",
      "voiceId": "string"
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

abortStatement

ユーザーが会話を完了せずに終了することを選択した場合に Amazon Lex が返すメッセージ。詳細については、「PutBot」を参照してください。

型: Statement オブジェクト

checksum

ボットの $LATEST バージョンの特定のリビジョンを識別するために使用される、ボットのチェックサム。

型: 文字列

childDirected

Amazon Lex Model·Building·Service で作成された各 Amazon Lex ボットについて、Amazon Lex 全体または一部の使用が、13 歳未満の児童を対象とするウェブサイト、プログラム、その他のアプリケーションに関連しており、COPPA (Children's Online Privacy Protection Act) の対象となっているかどうかを、`childDirected` フィールドに `true` または `false` を指定する必要があります。`childDirected` フィールドに `true` を指定することで、Amazon Lex 全体または一部の使用が、13 歳未満の児童を対象とするウェブサイト、プログラム、その他のアプリケーションに関連しており、COPPA の対象になることに同意します。`childDirected` フィールドに `false` を指定することで、Amazon Lex 全体または一部の使用が、13 歳未満の児童を対象とするウェブサイト、プログラム、その他のアプリケーションに関連しておらず、COPPA の対象にならないことに同意します。Amazon Lex の全体または一部の使用が、13歳未満の児童を対象にしており、COPPAの対象となるウェブサイト、プログラム、またはその他のアプリケーションに関連するかどうかを正確に設定するために、`childDirected` フィールドにデフォルト値を指定することはできません。

Amazon Lex 全体または一部の使用が、13 歳未満の児童を対象とするウェブサイト、プログラム、その他のアプリケーションに関連する場合は、COPPA により必要とされる検証可能な保護者の同意が必要です。13 歳未満の児童を対象とするウェブサイト、プログラム、またはその他のアプリケーションに関連する Amazon Lex の全体または一部の使用についての詳細は、「Amazon Lex FAQ」を参照してください。

型: ブール値

clarificationPrompt

Amazon Lex がユーザーのリクエストを理解できない場合に、使用するメッセージ。詳細については、「PutBot」を参照してください。

型: Prompt オブジェクト

createdDate

ボットが作成された日付。

型: タイムスタンプ

description

ボットの説明。

型: 文字列

長さの制限: 最小長は 0 です。最大長は 200 です。

detectSentiment

ユーザーの発話を Amazon Comprehend に送信して、センチメント分析を行うかどうかを示します。

型: ブール値

enableModelImprovements

ボットが精度向上を使用しているかどうかを示します。`true` はボットが精度向上を使用していることを示し、それ以外は `false` となります。

型: ブール値

failureReason

`status` が `FAILED` の場合、Amazon Lex はボットの構築に失敗した理由を提示します。

型: 文字列

idleSessionTTLInSeconds

Amazon Lex が会話で収集したデータを保持する最大時間 (秒) です。詳細については、「PutBot」を参照してください。

型: 整数

値の範囲: 最小値 は 60 です。最大値は 86400 です。

intents

intent オブジェクトの配列。詳細については、「PutBot」を参照してください。

型: Intent オブジェクトの配列

lastUpdatedDate

ボットが更新された日付。リソースを作成すると、作成日と最終更新日は同じ日付になります。

型: タイムスタンプ

locale

ボットのターゲットロケール。

型: 文字列

有効な値 : de-DE | en-AU | en-GB | en-IN | en-US | es-419 | es-ES | es-US | fr-FR | fr-CA | it-IT | ja-JP | ko-KR

name

ボットの名前。

型: 文字列

長さの制限: 最小長は 2 です。最大長は 50 です。

Pattern: ^([A-Za-z]_?)+$

nluIntentConfidenceThreshold

OR レスポンスで代替インテントを返すときに Amazon Lex がAMAZON.FallbackIntentAMAZON.KendraSearchIntent、、または両方を挿入する位置を決定するスコア。PostContentPostText AMAZON.FallbackIntentすべてのインテントの信頼度スコアがこの値を下回る場合に挿入されます。 AMAZON.KendraSearchIntentボット用に設定されている場合にのみ挿入されます。

型: 倍精度

有効な範囲 : 最小値 は 0 です。最大値は 1 です。

status

ボットのステータス。

ステータスが BUILDING の場合、Amazon Lex はテストと使用のためにボットを構築します。

ボットのステータスが READY_BASIC_TESTING の場合、ボットのインテントで指定された正確な発話を使用してボットをテストできます。ボットが完全なテストまたは実行の準備ができたら、ステータスは READY になります。

ボットの構築に問題があった場合、ステータスは FAILED となり、failureReason のフィールドにはボットが構築できなかった理由が記載されます。

ボットが保存され、まだビルドされていない場合、ステータスは NOT_BUILT になります。

型: 文字列

有効な値 : BUILDING | READY | READY_BASIC_TESTING | FAILED | NOT_BUILT

version

ボットのバージョン。新規のボットの場合、バージョンは常に $LATEST です。

型: 文字列

長さの制限 : 最小長は 1 です。最大長は 64 文字です。

パターン: \$LATEST|[0-9]+

voiceId

Amazon Lex がユーザーとの音声対話に使用する Amazon Polly 音声 ID。詳細については、「PutBot」を参照してください。

型: 文字列

エラー

BadRequestException

リクエストが適切にフォーマットされていません。例えば、値が無効であったり、必須項目が設定されていない場合です。フィールドの値を確認して、再度お試しください。

HTTP ステータスコード : 400

InternalFailureException

Amazon Lex 内部エラーが発生しました。リクエストを再試行してください。

HTTP ステータスコード：500

LimitExceededException

リクエストが制限を超えました。リクエストを再試行してください。

HTTP ステータスコード: 429

NotFoundException

リクエストで指定されたリソースは見つかりませんでした。リソースを確認して、もう一度お試しください。

HTTP ステータスコード: 404

その他の参照資料

この API を言語固有の AWS SDK で使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go バージョン 2 用 SDK
- AWS Java V2 用 SDK
- AWS V3 用 JavaScript SDK
- AWS PHP V3 用SDK
- AWS Python 用 SDK
- AWS ルビー V3 用 SDK

GetBotAlias

サービス: Amazon Lex Model Building Service

Amazon Lex ボットエイリアスに関する情報を返します。エイリアスの詳細については、「バージョニングとエイリアス」を参照してください。

このオペレーションには、`lex:GetBotAlias` アクションに対する許可が必要です。

リクエストの構文

```
GET /bots/botName/aliases/name HTTP/1.1
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

botName

ボットの名前。

長さの制限: 最小長は 2 です。最大長は 50 です。

Pattern: `^([A-Za-z]_?)+$`

必須：はい

name

ボットエイリアスの名前。名前は、大文字と小文字が区別されます。

長さの制限：最小長は 1 です。最大長は 100 です。

パターン: `^([A-Za-z]_?)+$`

必須: はい

リクエストボディ

リクエストにリクエスト本文がありません。

レスポンスの構文

```
HTTP/1.1 200
```

```
Content-type: application/json

{
   "botName": "string",
   "botVersion": "string",
   "checksum": "string",
   "conversationLogs": {
      "iamRoleArn": "string",
      "logSettings": [
         {
            "destination": "string",
            "kmsKeyArn": "string",
            "logType": "string",
            "resourceArn": "string",
            "resourcePrefix": "string"
         }
      ]
   },
   "createdDate": number,
   "description": "string",
   "lastUpdatedDate": number,
   "name": "string"
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

botName

エイリアスが指すボットの名前。

型: 文字列

長さの制限: 最小長は 2 です。最大長は 50 です。

Pattern: ^([A-Za-z]_?)+$

botVersion

エイリアスが指すボットのバージョン。

型: 文字列

長さの制限：最小長は 1 です。最大長は 64 文字です。

パターン: \$LATEST|[0-9]+

checksum

ボットエイリアスのチェックサム。

型: 文字列

conversationLogs

Amazon Lex がエイリアスの会話ログをどのように使用するかを決定する設定。

型: ConversationLogsResponse オブジェクト

createdDate

ボットエイリアスが作成された日付。

型: タイムスタンプ

description

ボットエイリアスの説明。

型: 文字列

長さの制限: 最小長は 0 です。最大長は 200 です。

lastUpdatedDate

ボットエイリアスが更新された日付。リソースを作成する場合、作成日と最終更新日は同じ日付になります。

型: タイムスタンプ

name

ボットエイリアスの名前。

型: 文字列

長さの制限: 最小長は 1 です。最大長は 100 です。

パターン：^([A-Za-z]_?)+$

エラー

BadRequestException

リクエストが適切にフォーマットされていません。例えば、値が無効であったり、必須項目が設定されていない場合です。フィールドの値を確認して、再度お試しください。

HTTP ステータスコード：400

InternalFailureException

Amazon Lex 内部エラーが発生しました。リクエストを再試行してください。

HTTP ステータスコード：500

LimitExceededException

リクエストが制限を超えました。リクエストを再試行してください。

HTTP ステータスコード: 429

NotFoundException

リクエストで指定されたリソースは見つかりませんでした。リソースを確認して、もう一度お試しください。

HTTP ステータスコード: 404

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go バージョン 2 用 SDK
- AWS Java V2 用 SDK
- AWS V3 用 JavaScript SDK
- AWS PHP V3 用SDK
- AWS Python 用 SDK
- AWS ルビー V3 用 SDK

GetBotAliases

サービス: Amazon Lex Model Building Service

指定された Amazon Lex ボットのエイリアスのリストを返します。

このオペレーションには、`lex:GetBotAliases` アクションに対する許可が必要です。

リクエストの構文

```
GET /bots/botName/aliases/?
maxResults=maxResults&nameContains=nameContains&nextToken=nextToken HTTP/1.1
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

botName

ボットの名前。

長さの制限: 最小長は 2 です。最大長は 50 です。

Pattern: `^([A-Za-z]_?)+$`

必須：はい

maxResults

レスポンスに返されるエイリアスの最大数。デフォルトは 50 です 。

有効範囲: 最小値 は 1 です。最大値は 50 です。

nameContains

ボットのエイリアス名で一致する部分文字列。名前の一部が部分文字列と一致する場合、エイリアスが返されます。例えば、「xyz」は「xyzabc」と「abcxyz」の両方に一致します。

長さの制限：最小長は 1 です。最大長は 100 です。

パターン: `^([A-Za-z]_?)+$`

nextToken

エイリアスの次のページを取得するためのページ割りトークン。この呼び出しに対する応答が切り捨てられた場合、Amazon Lex は応答の中にページ割りトークンを返します。次のページのエイリアスを取得するには、次のリクエストでページ割りページ割りトークンを指定します。

リクエスト本文

リクエストにリクエスト本文がありません。

レスポンスの構文

```
HTTP/1.1 200
Content-type: application/json

{
   "BotAliases": [
      {
         "botName": "string",
         "botVersion": "string",
         "checksum": "string",
         "conversationLogs": {
            "iamRoleArn": "string",
            "logSettings": [
               {
                  "destination": "string",
                  "kmsKeyArn": "string",
                  "logType": "string",
                  "resourceArn": "string",
                  "resourcePrefix": "string"
               }
            ]
         },
         "createdDate": number,
         "description": "string",
         "lastUpdatedDate": number,
         "name": "string"
      }
   ],
   "nextToken": "string"
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

BotAliases

BotAliasMetadata オブジェクトの配列で、それぞれがボットのエイリアスを表します。

型: BotAliasMetadata オブジェクトの配列

nextToken

エイリアスの次のページを取得するためのページ割りトークン。この呼び出しに対する応答が切り捨てられた場合、Amazon Lex は応答の中にページ割りトークンを返します。次のページのエイリアスを取得するには、次のリクエストでページ割りページ割りトークンを指定します。

型: 文字列

エラー

BadRequestException

リクエストが適切にフォーマットされていません。例えば、値が無効であったり、必須項目が設定されていない場合です。フィールドの値を確認して、再度お試しください。

HTTP ステータスコード：400

InternalFailureException

Amazon Lex 内部エラーが発生しました。リクエストを再試行してください。

HTTP ステータスコード：500

LimitExceededException

リクエストが制限を超えました。リクエストを再試行してください。

HTTP ステータスコード: 429

以下も参照してください。

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++

- AWS Go バージョン 2 用 SDK
- AWS Java V2 用 SDK
- AWS V3 用 JavaScript SDK
- AWS PHP V3 用SDK
- AWS Python 用 SDK
- AWS ルビー V3 用 SDK

GetBotChannelAssociation

サービス: Amazon Lex Model Building Service

Amazon Lex ボットとメッセージングプラットフォームの間の関連付けに関する情報を返します。

このオペレーションには、`lex:GetBotChannelAssociation` アクションに対する許可が必要です。

リクエストの構文

```
GET /bots/botName/aliases/aliasName/channels/name HTTP/1.1
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

aliasName

この関連付けが作成されている Amazon Lex ボットの特定のバージョンを指すエイリアス。

長さの制限：最小長は 1 です。最大長は 100 です。

パターン: `^([A-Za-z]_?)+$`

必須：はい

botName

Amazon Lex ボットの名前。

長さの制限: 最小長は 2 です。最大長は 50 です。

Pattern: `^([A-Za-z]_?)+$`

必須：はい

name

ボットとチャンネル間の関連付けの名前。名前は、大文字と小文字が区別されます。

長さの制限：最小長は 1 です。最大長は 100 です。

パターン: `^([A-Za-z]_?)+$`

必須: はい

リクエストボディ

リクエストにリクエスト本文がありません。

レスポンスの構文

```
HTTP/1.1 200
Content-type: application/json

{
   "botAlias": "string",
   "botConfiguration": {
      "string" : "string"
   },
   "botName": "string",
   "createdDate": number,
   "description": "string",
   "failureReason": "string",
   "name": "string",
   "status": "string",
   "type": "string"
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

botAlias

この関連付けが作成されている Amazon Lex ボットの特定のバージョンを指すエイリアス。

型: 文字列

長さの制限: 最小長は 1 です。最大長は 100 です。

パターン: ^([A-Za-z]_?)+$

botConfiguration

メッセージングプラットフォームが Amazon Lex ボットと通信するために必要な情報を提供します。

型: 文字列間のマッピング

マップエントリ: 10 の項目の最大数。

botName

Amazon Lex ボットの名前。

型: 文字列

長さの制限: 最小長は 2 です。最大長は 50 です。

Pattern: ^([A-Za-z]_?)+$

createdDate

ボットとチャンネルの関連付けが作成された日付。

型: タイムスタンプ

description

ボットとチャンネル間の関連付けの説明。

型: 文字列

長さの制限: 最小長は 0 です。最大長は 200 です。

failureReason

status が FAILED の場合、Amazon Lex は関連性の作成に失敗した理由を提示します。

型: 文字列

name

ボットとチャンネル間の関連付けの名前。

型: 文字列

長さの制限: 最小長は 1 です。最大長は 100 です。

パターン: ^([A-Za-z]_?)+$

status

ボットチャンネルのステータス。

- CREATED - チャンネルが作成され、使用可能な状態です。
- IN_PROGRESS - チャンネルの作成中。
- FAILED - チャンネルの作成中にエラーが発生しました。失敗の原因については、「failureReason」フィールドを参照してください。

型: 文字列

有効な値：IN_PROGRESS | CREATED | FAILED

type

メッセージングプラットフォームのタイプ。

型: 文字列

有効な値：Facebook | Slack | Twilio-Sms | Kik

エラー

BadRequestException

リクエストが適切にフォーマットされていません。例えば、値が無効であったり、必須項目が設定されていない場合です。フィールドの値を確認して、再度お試しください。

HTTP ステータスコード：400

InternalFailureException

Amazon Lex 内部エラーが発生しました。リクエストを再試行してください。

HTTP ステータスコード：500

LimitExceededException

リクエストが制限を超えました。リクエストを再試行してください。

HTTP ステータスコード: 429

NotFoundException

リクエストで指定されたリソースは見つかりませんでした。リソースを確認して、もう一度お試しください。

HTTP ステータスコード: 404

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go バージョン 2 用 SDK
- AWS Java V2 用 SDK
- AWS V3 用 JavaScript SDK
- AWS PHP V3 用SDK
- AWS Python 用 SDK
- AWS ルビー V3 用 SDK

GetBotChannelAssociations

サービス: Amazon Lex Model Building Service

特定のボットに関連付けられているすべてのチャンネルのリストを返します。

この GetBotChannelAssociations オペレーションには、lex:GetBotChannelAssociations アクションに対するアクセス許可が必要です。

リクエストの構文

```
GET /bots/botName/aliases/aliasName/channels/?
maxResults=maxResults&nameContains=nameContains&nextToken=nextToken HTTP/1.1
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

aliasName

この関連付けが作成されている Amazon Lex ボットの特定のバージョンを指すエイリアス。

長さの制限：最小長は 1 です。最大長は 100 です。

パターン: ^(-|^([A-Za-z]_?)+$)$

必須：はい

botName

アソシエーション内の Amazon Lex ボットの名前。

長さの制限: 最小長は 2 です。最大長は 50 です。

Pattern: ^([A-Za-z]_?)+$

必須：はい

maxResults

レスポンスに返されるアソシエーションの最大数。デフォルトは 50 です。

有効範囲: 最小値 は 1 です。最大値は 50 です。

nameContains

チャンネルアソシエーション名で照合する部分文字列。関連付けは、名前の一部が部分文字列と一致する場合に返されます。例えば、「xyz」は「xyzabc」と「abcxyz」の両方に一致します。すべてのボットチャンネルのアソシエーションを返すには、nameContains パラメータにハイフン (「-」) を使用します。

長さの制限：最小長は 1 です。最大長は 100 です。

パターン: ^([A-Za-z]_?)+$

nextToken

関連付けの次のページを取得するためのページ割りトークン。この呼び出しに対する応答が切り捨てられた場合、Amazon Lex は応答の中にページ割りトークンを返します。次のページの関連付けを取得するには、次のリクエストでページ割りトークンを指定します。

リクエスト本文

リクエストにリクエスト本文がありません。

レスポンスの構文

```
HTTP/1.1 200
Content-type: application/json

{
   "botChannelAssociations": [
      {
         "botAlias": "string",
         "botConfiguration": {
            "string" : "string"
         },
         "botName": "string",
         "createdDate": number,
         "description": "string",
         "failureReason": "string",
         "name": "string",
         "status": "string",
         "type": "string"
      }
   ],
   "nextToken": "string"
```

```
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

botChannelAssociations

Amazon Lex ボットとチャンネルとの関連付けに関する情報を提供するオブジェクトの配列 (各関連付けごとに 1 つ)。

型: BotChannelAssociation オブジェクトの配列

nextToken

次のページの関連付けを取得するページ割りトークン。この呼び出しに対する応答が切り捨てられた場合、Amazon Lex は応答の中にページ割りトークンを返します。次のページの関連付けを取得するには、次のリクエストでページ割りトークンを指定します。

型: 文字列

エラー

BadRequestException

リクエストが適切にフォーマットされていません。例えば、値が無効であったり、必須項目が設定されていない場合です。フィールドの値を確認して、再度お試しください。

HTTP ステータスコード：400

InternalFailureException

Amazon Lex 内部エラーが発生しました。リクエストを再試行してください。

HTTP ステータスコード：500

LimitExceededException

リクエストが制限を超えました。リクエストを再試行してください。

HTTP ステータスコード: 429

以下も参照してください。

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go バージョン 2 用 SDK
- AWS Java V2 用 SDK
- AWS V3 用 JavaScript SDK
- AWS PHP V3 用SDK
- AWS Python 用 SDK
- AWS ルビー V3 用 SDK

GetBots

サービス: Amazon Lex Model Building Service

以下のようにボット情報を返します。

- nameContains フィールドを指定すると、名前に指定した文字列が含まれるすべてのボットの $LATEST バージョンの情報が応答に含まれます。
- nameContains フィールドを指定しなかった場合、このオペレーションでは、すべてのボットの $LATEST バージョンの情報が返されます。

このオペレーションには lex:GetBots アクションに対するアクセス許可が必要です。

リクエストの構文

```
GET /bots/?maxResults=maxResults&nameContains=nameContains&nextToken=nextToken HTTP/1.1
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

maxResults

リクエストが返すレスポンスに含まれるボットの最大数。デフォルトは 10 です。

有効範囲: 最小値 は 1 です。最大値は 50 です。

nameContains

ボット名にマッチする部分文字列。ボットの名前の一部が部分文字列と一致する場合、ボットが返されます。例えば、「xyz」は「xyzabc」と「abcxyz」の両方に一致します。

長さの制限: 最小長は 2 です。最大長は 50 です。

Pattern: ^([A-Za-z]_?)+$

nextToken

ボットの次のページを取得するページ割りトークン。この呼び出しに対する応答が切り捨てられた場合、Amazon Lex は応答の中にページ割りトークンを返します。ボットの次のページを取得するには、次のリクエストでページ分割トークンを指定します。

リクエスト本文

リクエストにリクエスト本文がありません。

レスポンスの構文

```
HTTP/1.1 200
Content-type: application/json

{
   "bots": [
      {
         "createdDate": number,
         "description": "string",
         "lastUpdatedDate": number,
         "name": "string",
         "status": "string",
         "version": "string"
      }
   ],
   "nextToken": "string"
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

bots

botMetadata オブジェクトの配列で、各ボットに 1 つのエントリがあります。

型: BotMetadata オブジェクトの配列

nextToken

レスポンスが切り捨てられる場合、次のリクエストでボットの次のページを取得するためのページ割りトークンが含まれます。

型: 文字列

エラー

BadRequestException

リクエストが適切にフォーマットされていません。例えば、値が無効であったり、必須項目が設定されていない場合です。フィールドの値を確認して、再度お試しください。

HTTP ステータスコード：400

InternalFailureException

Amazon Lex 内部エラーが発生しました。リクエストを再試行してください。

HTTP ステータスコード：500

LimitExceededException

リクエストが制限を超えました。リクエストを再試行してください。

HTTP ステータスコード: 429

NotFoundException

リクエストで指定されたリソースは見つかりませんでした。リソースを確認して、もう一度お試しください。

HTTP ステータスコード: 404

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go バージョン 2 用 SDK
- AWS Java V2 用 SDK
- AWS V3 用 JavaScript SDK
- AWS PHP V3 用SDK
- AWS Python 用 SDK
- AWS ルビー V3 用 SDK

GetBotVersions

サービス: Amazon Lex Model Building Service

ボットのすべてのバージョンに関する情報を受け取ります。

GetBotVersions オペレーションは、ボットの各バージョンに対して BotMetadata オブジェクトを返します。例えば、あるボットに 3 つの番号がついたバージョンがある場合、GetBotVersions オペレーションは、各番号のバージョンに 1 つ、$LATEST バージョンに 1 つ、合計 4 つの BotMetadata オブジェクトをレスポンスに返します。

GetBotVersions オペレーションは、少なくとも 1 つの $LATEST バージョンを常に返します。

このオペレーションには、lex:GetBotVersions アクションに対する許可が必要です。

リクエストの構文

```
GET /bots/name/versions/?maxResults=maxResults&nextToken=nextToken HTTP/1.1
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

maxResults

レスポンスに返されるボットバージョンの最大数。デフォルトは 10 です。

有効範囲: 最小値 は 1 です。最大値は 50 です。

name

バージョンを返すボットの名前です。

長さの制限: 最小長は 2 です。最大長は 50 です。

Pattern: ^([A-Za-z]_?)+$

必須：はい

nextToken

ボットバージョンの次のページを取得するためのページ割りトークン。この呼び出しに対する応答が切り捨てられた場合、Amazon Lex は応答の中にページ割りトークンを返します。次のページのバージョンを取得するには、次のリクエストでページ割りトークンを指定します。

リクエスト本文

リクエストにリクエスト本文がありません。

レスポンスの構文

```
HTTP/1.1 200
Content-type: application/json

{
   "bots": [
      {
         "createdDate": number,
         "description": "string",
         "lastUpdatedDate": number,
         "name": "string",
         "status": "string",
         "version": "string"
      }
   ],
   "nextToken": "string"
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

bots

BotMetadata オブジェクトの配列で、ボットの各番号のバージョンに 1 つずつ、さらに $LATEST バージョンに 1 つあります。

型: BotMetadata オブジェクトの配列

nextToken

ボットバージョンの次のページを取得するためのページ割りトークン。この呼び出しに対する応答が切り捨てられた場合、Amazon Lex は応答の中にページ割りトークンを返します。次のページのバージョンを取得するには、次のリクエストでページ割りトークンを指定します。

型: 文字列

エラー

BadRequestException

リクエストが適切にフォーマットされていません。例えば、値が無効であったり、必須項目が設定されていない場合です。フィールドの値を確認して、再度お試しください。

HTTP ステータスコード：400

InternalFailureException

Amazon Lex 内部エラーが発生しました。リクエストを再試行してください。

HTTP ステータスコード：500

LimitExceededException

リクエストが制限を超えました。リクエストを再試行してください。

HTTP ステータスコード: 429

NotFoundException

リクエストで指定されたリソースは見つかりませんでした。リソースを確認して、もう一度お試しください。

HTTP ステータスコード: 404

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go バージョン 2 用 SDK
- AWS Java V2 用 SDK
- AWS V3 用 JavaScript SDK
- AWS PHP V3 用SDK
- AWS Python 用 SDK
- AWS ルビー V3 用 SDK

GetBuiltinIntent

サービス: Amazon Lex Model Building Service

組み込みインテントに関する情報を返します。

このオペレーションには `lex:GetBuiltinIntent` アクションに対するアクセス許可が必要です。

リクエストの構文

```
GET /builtins/intents/signature HTTP/1.1
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

signature

組み込みインテントの一意の識別子。インテントの署名を見つけるには、「Alexa Skills Kit」の「Standard Built-in Intents」(標準の組み込みインテント) を参照してください。

必須: はい

リクエストボディ

リクエストにリクエスト本文がありません。

レスポンスの構文

```
HTTP/1.1 200
Content-type: application/json

{
   "signature": "string",
   "slots": [
      {
         "name": "string"
      }
   ],
   "supportedLocales": [ "string" ]
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

signature

組み込みインテントの一意の識別子。

型: 文字列

slots

BuiltinIntentSlot オブジェクトの配列で、インテントの各スロットタイプごとに 1 つのエントリがあります。

型: BuiltinIntentSlot オブジェクトの配列

supportedLocales

インテントがサポートするロケールのリスト。

タイプ：文字列の配列

有効な値：de-DE | en-AU | en-GB | en-IN | en-US | es-419 | es-ES | es-US | fr-FR | fr-CA | it-IT | ja-JP | ko-KR

エラー

BadRequestException

リクエストが適切にフォーマットされていません。例えば、値が無効であったり、必須項目が設定されていない場合です。フィールドの値を確認して、再度お試しください。

HTTP ステータスコード：400

InternalFailureException

Amazon Lex 内部エラーが発生しました。リクエストを再試行してください。

HTTP ステータスコード：500

LimitExceededException

リクエストが制限を超えました。リクエストを再試行してください。

HTTP ステータスコード: 429

NotFoundException

リクエストで指定されたリソースは見つかりませんでした。リソースを確認して、もう一度お試しください。

HTTP ステータスコード: 404

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go バージョン 2 用 SDK
- AWS Java V2 用 SDK
- AWS V3 用 JavaScript SDK
- AWS PHP V3 用SDK
- AWS Python 用 SDK
- AWS ルビー V3 用 SDK

GetBuiltinIntents

サービス: Amazon Lex Model Building Service

指定された基準を満たす組み込みインテントのリストを取得します。

このオペレーションには `lex:GetBuiltinIntents` アクションに対するアクセス許可が必要です。

リクエストの構文

```
GET /builtins/intents/?
locale=locale&maxResults=maxResults&nextToken=nextToken&signatureContains=signatureContains
 HTTP/1.1
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

[locale](#)

インテントがサポートするロケールのリスト。

有効な値 : `de-DE | en-AU | en-GB | en-IN | en-US | es-419 | es-ES | es-US | fr-FR | fr-CA | it-IT | ja-JP | ko-KR`

[maxResults](#)

レスポンスに返されるインテントの最大数。デフォルトは 10 です。

有効範囲: 最小値 は 1 です。最大値は 50 です。

[nextToken](#)

インテントの次のページを取得するページ割りトークン。この API コールが切り捨てられると、Amazon Lex はレスポンスでページ割りトークンを返します。次のページのエイリアスを取得するには、次のリクエストでページ割りトークンを指定します。

[signatureContains](#)

組み込みインテントシグニチャで照合する部分文字列。インテントは、署名の一部が部分文字列と一致した場合に返されます。例えば、「xyz」は「xyzabc」と「abcxyz」の両方に一致します。インテントの署名を見つけるには、「Alexa Skills Kit」の「Standard Built-in Intents」(標準の組み込みインテント) を参照してください。

リクエスト本文

リクエストにリクエスト本文がありません。

レスポンスの構文

```
HTTP/1.1 200
Content-type: application/json

{
   "intents": [
      {
         "signature": "string",
         "supportedLocales": [ "string" ]
      }
   ],
   "nextToken": "string"
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

intents

builtinIntentMetadata オブジェクトの配列で、レスポンスの各インテントに対して 1 つずつです。

型: BuiltinIntentMetadata オブジェクトの配列

nextToken

インテントの次のページを取得するページ割りトークン。この API コールに対するレスポンスが切り捨てられた場合、Amazon Lex はレスポンスでページ割りトークンを返します。インテントの次のページを取得するには、次のリクエストでページ割りトークンを指定します。

型: 文字列

エラー

BadRequestException

リクエストが適切にフォーマットされていません。例えば、値が無効であったり、必須項目が設定されていない場合です。フィールドの値を確認して、再度お試しください。

HTTP ステータスコード：400

InternalFailureException

Amazon Lex 内部エラーが発生しました。リクエストを再試行してください。

HTTP ステータスコード：500

LimitExceededException

リクエストが制限を超えました。リクエストを再試行してください。

HTTP ステータスコード: 429

以下も参照してください。

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go バージョン 2 用 SDK
- AWS Java V2 用 SDK
- AWS V3 用 JavaScript SDK
- AWS PHP V3 用SDK
- AWS Python 用 SDK
- AWS ルビー V3 用 SDK

GetBuiltinSlotTypes

サービス: Amazon Lex Model Building Service

指定された基準を満たす組み込みスロットタイプのリストを取得します。

組み込みスロットタイプの一覧については、「Alexa Skills Kit」の「Slot Type Reference」を参照してください。

このオペレーションには `lex:GetBuiltInSlotTypes` アクションに対するアクセス許可が必要です。

リクエストの構文

```
GET /builtins/slottypes/?
locale=locale&maxResults=maxResults&nextToken=nextToken&signatureContains=signatureContains
 HTTP/1.1
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

locale

スロットタイプがサポートするロケールのリスト。

有効な値 : `de-DE | en-AU | en-GB | en-IN | en-US | es-419 | es-ES | es-US | fr-FR | fr-CA | it-IT | ja-JP | ko-KR`

maxResults

レスポンスに返されるスロットタイプの最大数。デフォルトは 10 です。

有効範囲: 最小値 は 1 です。最大値は 50 です。

nextToken

スロットタイプの次のページを取得するページ割りトークン。この API コールに対するレスポンスが切り捨てられた場合、Amazon Lex はレスポンスでページ割りトークンを返します。スロットタイプの次のページを取得するには、次のリクエストでページ割りトークンを指定します。

signatureContains

組み込みスロットタイプのシグニチャで一致する部分文字列。スロットタイプは、署名の一部が部分文字列と一致した場合に返されます。例えば、「xyz」は「xyzabc」と「abcxyz」の両方に一致します。

リクエスト本文

リクエストにリクエスト本文がありません。

レスポンスの構文

```
HTTP/1.1 200
Content-type: application/json

{
   "nextToken": "string",
   "slotTypes": [
      {
         "signature": "string",
         "supportedLocales": [ "string" ]
      }
   ]
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

nextToken

レスポンスが切り捨てられた場合、レスポンスにページ割りトークンが含まれており、次のリクエストでスロットタイプの次のページを取得できます。

型: 文字列

slotTypes

BuiltInSlotTypeMetadata オブジェクトの配列で、返された各スロットタイプごとに 1 つのエントリがあります。

型: BuiltinSlotTypeMetadata オブジェクトの配列

エラー

BadRequestException

リクエストが適切にフォーマットされていません。例えば、値が無効であったり、必須項目が設定されていない場合です。フィールドの値を確認して、再度お試しください。

HTTP ステータスコード：400

InternalFailureException

Amazon Lex 内部エラーが発生しました。リクエストを再試行してください。

HTTP ステータスコード：500

LimitExceededException

リクエストが制限を超えました。リクエストを再試行してください。

HTTP ステータスコード: 429

以下も参照してください。

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go バージョン 2 用 SDK
- AWS Java V2 用 SDK
- AWS V3 用 JavaScript SDK
- AWS PHP V3 用SDK
- AWS Python 用 SDK
- AWS ルビー V3 用 SDK

GetExport

サービス: Amazon Lex Model Building Service

Amazon Lex リソースの内容を指定された形式でエクスポートします。

リクエストの構文

```
GET /exports/?exportType=exportType&name=name&resourceType=resourceType&version=version
 HTTP/1.1
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

exportType

エクスポートするデータの形式。

有効な値：ALEXA_SKILLS_KIT | LEX

必須: はい

name

エクスポートするボットの名前。

長さの制限：最小長は 1 です。最大長は 100 です。

パターン: [a-zA-Z_]+

必須：はい

resourceType

エクスポートするリソースのタイプ。

有効な値：BOT | INTENT | SLOT_TYPE

必須: はい

version

エクスポートするボットのバージョン。

長さの制限：最小長は 1 です。最大長は 64 文字です。

パターン: [0-9]+

必須: はい

リクエストボディ

リクエストにリクエスト本文がありません。

レスポンスの構文

```
HTTP/1.1 200
Content-type: application/json

{
   "exportStatus": "string",
   "exportType": "string",
   "failureReason": "string",
   "name": "string",
   "resourceType": "string",
   "url": "string",
   "version": "string"
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

exportStatus

エクスポートのステータス。

- IN_PROGRESS - エクスポートが進行中です。
- READY - エクスポートが完了しました。
- FAILED - エクスポートを完了できませんでした。

型: 文字列

有効な値：IN_PROGRESS | READY | FAILED

exportType

エクスポートするデータの形式

型: 文字列

有効な値：ALEXA_SKILLS_KIT | LEX

failureReason

status が FAILED の場合、Amazon Lex がリソースのエクスポートに失敗した理由を提示します。

型: 文字列

name

エクスポートされるボットの名前。

型: 文字列

長さの制限: 最小長は 1 です。最大長は 100 です。

パターン: [a-zA-Z_]+

resourceType

エクスポートされたリソースのタイプ。

型: 文字列

有効な値：BOT | INTENT | SLOT_TYPE

url

エクスポートされたリソースの場所を提供する S3 の署名付き URL。エクスポートされたリソースは、エクスポートされた JSON 形式のリソースを含む ZIP アーカイブです。アーカイブの構造は変わる可能性があります。コードはアーカイブ構造に依存してはいけません。

型: 文字列

version

エクスポートされるボットのバージョン。

型: 文字列

長さの制限：最小長は 1 です。最大長は 64 文字です。

パターン：`[0-9]+`

エラー

BadRequestException

リクエストが適切にフォーマットされていません。例えば、値が無効であったり、必須項目が設定されていない場合です。フィールドの値を確認して、再度お試しください。

HTTP ステータスコード：400

InternalFailureException

Amazon Lex 内部エラーが発生しました。リクエストを再試行してください。

HTTP ステータスコード：500

LimitExceededException

リクエストが制限を超えました。リクエストを再試行してください。

HTTP ステータスコード: 429

NotFoundException

リクエストで指定されたリソースは見つかりませんでした。リソースを確認して、もう一度お試しください。

HTTP ステータスコード: 404

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go バージョン 2 用 SDK
- AWS Java V2 用 SDK
- AWS V3 用 JavaScript SDK

- AWS PHP V3 用SDK
- AWS Python 用 SDK
- AWS ルビー V3 用 SDK

GetImport

サービス: Amazon Lex Model Building Service

`StartImport` オペレーションで開始したインポートジョブに関する情報を取得します。

リクエストの構文

```
GET /imports/importId HTTP/1.1
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

importId

返すインポートジョブ情報の識別子。

必須: はい

リクエストボディ

リクエストにリクエスト本文がありません。

レスポンスの構文

```
HTTP/1.1 200
Content-type: application/json

{
   "createdDate": number,
   "failureReason": [ "string" ],
   "importId": "string",
   "importStatus": "string",
   "mergeStrategy": "string",
   "name": "string",
   "resourceType": "string"
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

createdDate

インポートジョブが作成された日時のタイムスタンプ。

型: タイムスタンプ

failureReason

インポートジョブの完了に失敗した理由を示す文字列。

型: 文字列の配列

importId

特定のインポートジョブの識別子。

型: 文字列

importStatus

インポートジョブのステータス。ステータスが FAILED の場合、failureReason フィールドから失敗の理由を確認できます。

型: 文字列

有効な値：IN_PROGRESS | COMPLETE | FAILED

mergeStrategy

インポートファイル内の既存のリソースとリソースの間に競合があった場合に実行されるアクション。

型: 文字列

有効な値：OVERWRITE_LATEST | FAIL_ON_CONFLICT

name

インポートジョブに付けられた名前。

型: 文字列

長さの制限: 最小長は 1 です。最大長は 100 です。

パターン: [a-zA-Z_]+

resourceType

インポートされたリソースのタイプ。

型: 文字列

有効な値：BOT | INTENT | SLOT_TYPE

エラー

BadRequestException

リクエストが適切にフォーマットされていません。例えば、値が無効であったり、必須項目が設定されていない場合です。フィールドの値を確認して、再度お試しください。

HTTP ステータスコード：400

InternalFailureException

Amazon Lex 内部エラーが発生しました。リクエストを再試行してください。

HTTP ステータスコード：500

LimitExceededException

リクエストが制限を超えました。リクエストを再試行してください。

HTTP ステータスコード: 429

NotFoundException

リクエストで指定されたリソースは見つかりませんでした。リソースを確認して、もう一度お試しください。

HTTP ステータスコード: 404

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++

- AWS Go バージョン 2 用 SDK
- AWS Java V2 用 SDK
- AWS V3 用 JavaScript SDK
- AWS PHP V3 用SDK
- AWS Python 用 SDK
- AWS ルビー V3 用 SDK

GetIntent

サービス: Amazon Lex Model Building Service

インテントに関する情報を返します。インテント名に加えて、インテントバージョンを指定する必要があります。

このオペレーションには `lex:GetIntent` アクションを実行するアクセス許可が必要です。

リクエストの構文

```
GET /intents/name/versions/version HTTP/1.1
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

name

インテントの名前。名前は、大文字と小文字が区別されます。

長さの制限：最小長は 1 です。最大長は 100 です。

パターン: `^([A-Za-z]_?)+$`

必須：はい

version

インテントのバージョン。

長さの制限：最小長は 1 です。最大長は 64 文字です。

パターン: `\$LATEST|[0-9]+`

必須: はい

リクエストボディ

リクエストにリクエスト本文がありません。

レスポンスの構文

```
HTTP/1.1 200
Content-type: application/json
```

```
{
   "checksum": "string",
   "conclusionStatement": {
      "messages": [
         {
            "content": "string",
            "contentType": "string",
            "groupNumber": number
         }
      ],
      "responseCard": "string"
   },
   "confirmationPrompt": {
      "maxAttempts": number,
      "messages": [
         {
            "content": "string",
            "contentType": "string",
            "groupNumber": number
         }
      ],
      "responseCard": "string"
   },
   "createdDate": number,
   "description": "string",
   "dialogCodeHook": {
      "messageVersion": "string",
      "uri": "string"
   },
   "followUpPrompt": {
      "prompt": {
         "maxAttempts": number,
         "messages": [
            {
               "content": "string",
               "contentType": "string",
               "groupNumber": number
            }
         ],
         "responseCard": "string"
      },
      "rejectionStatement": {
         "messages": [
```

```
            {
               "content": "string",
               "contentType": "string",
               "groupNumber": number
            }
         ],
         "responseCard": "string"
      }
   },
   "fulfillmentActivity": {
      "codeHook": {
         "messageVersion": "string",
         "uri": "string"
      },
      "type": "string"
   },
   "inputContexts": [
      {
         "name": "string"
      }
   ],
   "kendraConfiguration": {
      "kendraIndex": "string",
      "queryFilterString": "string",
      "role": "string"
   },
   "lastUpdatedDate": number,
   "name": "string",
   "outputContexts": [
      {
         "name": "string",
         "timeToLiveInSeconds": number,
         "turnsToLive": number
      }
   ],
   "parentIntentSignature": "string",
   "rejectionStatement": {
      "messages": [
         {
            "content": "string",
            "contentType": "string",
            "groupNumber": number
         }
      ],
```

```
      "responseCard": "string"
   },
   "sampleUtterances": [ "string" ],
   "slots": [
      {
         "defaultValueSpec": {
            "defaultValueList": [
               {
                  "defaultValue": "string"
               }
            ]
         },
         "description": "string",
         "name": "string",
         "obfuscationSetting": "string",
         "priority": number,
         "responseCard": "string",
         "sampleUtterances": [ "string" ],
         "slotConstraint": "string",
         "slotType": "string",
         "slotTypeVersion": "string",
         "valueElicitationPrompt": {
            "maxAttempts": number,
            "messages": [
               {
                  "content": "string",
                  "contentType": "string",
                  "groupNumber": number
               }
            ],
            "responseCard": "string"
         }
      }
   ],
   "version": "string"
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

checksum

インテントのチェックサム。

型: 文字列

conclusionStatement

`fulfillmentActivity` 要素で指定された Lambda 関数がインテントを達成すると、Amazon Lex はこのステートメントをユーザーに伝えます。

型: Statement オブジェクト

confirmationPrompt

これがボットに定義されている場合、Amazon Lex がユーザーのインテントを達成する前の確認に使用するプロンプトです。詳細については、「PutIntent」を参照してください。

型: Prompt オブジェクト

createdDate

インテントが作成された日付。

型: タイムスタンプ

description

インテントの説明。

型: 文字列

長さの制限: 最小長は 0 です。最大長は 200 です。

dialogCodeHook

これがボットに定義されている場合、Amazon Lex は各ユーザー入力に対してこの Lambda 関数を呼び出します。詳細については、「PutIntent」を参照してください。

型: CodeHook オブジェクト

followUpPrompt

これが定義されている場合、Amazon Lex はこのプロンプトを使用して、インテントが達成された後に追加のユーザーアクティビティを要求します。詳細については、「PutIntent」を参照してください。

型: FollowUpPrompt オブジェクト

fulfillmentActivity

インテントがどのように達成されるのかを説明します。詳細については、「PutIntent」を参照してください。

型: FulfillmentActivity オブジェクト

inputContexts

Amazon Lex がユーザーとの会話の中でインテントを選択するためのアクティブなコンテキストをリストアップした InputContext オブジェクトの配列です。

型: InputContext オブジェクトの配列

配列メンバー : 最小数は 0 項目です。最大数は 5 項目です。

kendraConfiguration

AMAZON.KendraSearchIntent インテントで Amazon Kendra インデックスに接続するための設定情報 (項目がある場合)。

型: KendraConfiguration オブジェクト

lastUpdatedDate

インテントが更新された日付。リソースを作成する場合、作成日と最終更新日は同じ日付になります。

型: タイムスタンプ

name

インテントの名前。

型: 文字列

長さの制限: 最小長は 1 です。最大長は 100 です。

パターン: ^([A-Za-z]_?)+$

outputContexts

OutputContext オブジェクトの配列で、インテントが達成されたときにアクティブになるコンテキストを列挙します。

型: OutputContext オブジェクトの配列

の配列メンバー: 最小数は 0 項目です。最大数は 10 項目です。

parentIntentSignature

組み込みインテントの一意の識別子。

型: 文字列

rejectionStatement

ユーザーが `confirmationPrompt` で定義された質問に「いいえ」と答えた場合、Amazon Lex はインテントがキャンセルされたことを確認するためにこのステートメントを返します。

型: Statement オブジェクト

sampleUtterances

インテント用に構成されたサンプル発話の配列。

型: 文字列の配列

配列メンバー：最小数は 0 項目です。最大数は 1500 項目です。

長さの制限：最小長は 1 です。最大長は 200 です。

slots

インテント用に設定されたインテントスロットの配列。

型: Slot オブジェクトの配列

配列メンバー: 最小数は 0 項目です。最大数は 100 項目です。

version

インテントのバージョン。

型: 文字列

長さの制限：最小長は 1 です。最大長は 64 文字です。

パターン：`\$LATEST|[0-9]+`

エラー

BadRequestException

リクエストが適切にフォーマットされていません。例えば、値が無効であったり、必須項目が設定されていない場合です。フィールドの値を確認して、再度お試しください。

HTTP ステータスコード：400

InternalFailureException

Amazon Lex 内部エラーが発生しました。リクエストを再試行してください。

HTTP ステータスコード：500

LimitExceededException

リクエストが制限を超えました。リクエストを再試行してください。

HTTP ステータスコード: 429

NotFoundException

リクエストで指定されたリソースは見つかりませんでした。リソースを確認して、もう一度お試しください。

HTTP ステータスコード: 404

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go バージョン 2 用 SDK
- AWS Java V2 用 SDK
- AWS V3 用 JavaScript SDK
- AWS PHP V3 用SDK
- AWS Python 用 SDK
- AWS ルビー V3 用 SDK

GetIntents

サービス: Amazon Lex Model Building Service

次のようにインテント情報を返します。

- nameContains フィールドを指定した場合は、指定した文字列を含むすべてのインテントの $LATEST バージョン情報を返します。
- nameContains フィールドを指定しない場合は、すべてのインテントの $LATEST バージョンの情報を返します。

オペレーションには lex:GetIntents アクションに対するアクセス許可が必要です。

リクエストの構文

```
GET /intents/?maxResults=maxResults&nameContains=nameContains&nextToken=nextToken
 HTTP/1.1
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

maxResults

レスポンスに返されるインテントの最大数。デフォルトは 10 です。

有効範囲: 最小値 は 1 です。最大値は 50 です。

nameContains

インテント名で一致する部分文字列。インテントの名前の一部が部分文字列と一致する場合、インテントが返されます。例えば、「xyz」は「xyzabc」と「abcxyz」の両方に一致します。

長さの制限：最小長は 1 です。最大長は 100 です。

パターン: ^([A-Za-z]_?)+$

nextToken

インテントの次のページを取得するページ割りトークン。この API コールに対するレスポンスが切り捨てられた場合、Amazon Lex はレスポンスでページ割りトークンを返します。インテントの次のページを取得するには、次のリクエストでページ割りトークンを指定します。

リクエスト本文

リクエストにリクエスト本文がありません。

レスポンスの構文

```
HTTP/1.1 200
Content-type: application/json

{
   "intents": [
      {
         "createdDate": number,
         "description": "string",
         "lastUpdatedDate": number,
         "name": "string",
         "version": "string"
      }
   ],
   "nextToken": "string"
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

intents

Intent オブジェクトの配列。詳細については、「PutBot」を参照してください。

型: IntentMetadata オブジェクトの配列

nextToken

レスポンスが切り捨てられた場合、レスポンスにページ割りトークンが含まれており、次のリクエストで指定してインテントの次のページを取得できます。

型: 文字列

エラー

BadRequestException

リクエストが適切にフォーマットされていません。例えば、値が無効であったり、必須項目が設定されていない場合です。フィールドの値を確認して、再度お試しください。

HTTP ステータスコード：400

InternalFailureException

Amazon Lex 内部エラーが発生しました。リクエストを再試行してください。

HTTP ステータスコード：500

LimitExceededException

リクエストが制限を超えました。リクエストを再試行してください。

HTTP ステータスコード: 429

NotFoundException

リクエストで指定されたリソースは見つかりませんでした。リソースを確認して、もう一度お試しください。

HTTP ステータスコード: 404

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go バージョン 2 用 SDK
- AWS Java V2 用 SDK
- AWS V3 用 JavaScript SDK
- AWS PHP V3 用SDK
- AWS Python 用 SDK
- AWS ルビー V3 用 SDK

GetIntentVersions

サービス: Amazon Lex Model Building Service

インテントのすべてのバージョンに関する情報を取得します。

GetIntentVersions オペレーションは、インテントの各バージョンに IntentMetadata オブジェクトを返します。例えば、あるインテントに 3 つの番号が付いたバージョンがある場合、GetIntentVersions オペレーションは、各番号のバージョンに 1 つ、$LATEST バージョンに 1 つ、合計 4 つの IntentMetadata オブジェクトをレスポンスに返します。

GetIntentVersions オペレーションは、少なくとも 1 つの $LATEST バージョンを常に返します。

このオペレーションには、lex:GetIntentVersions アクションに対する許可が必要です。

リクエストの構文

```
GET /intents/name/versions/?maxResults=maxResults&nextToken=nextToken HTTP/1.1
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

maxResults

レスポンスに返されるインテントバージョンの最大数。デフォルトは 10 です。

有効範囲: 最小値 は 1 です。最大値は 50 です。

name

バージョンを返すインテントの名前。

長さの制限：最小長は 1 です。最大長は 100 です。

パターン: ^([A-Za-z]_?)+$

必須：はい

nextToken

インテントバージョンの次のページを取得するためのページ割りトークン。この呼び出しに対する応答が切り捨てられた場合、Amazon Lex は応答の中にページ割りトークンを返します。次のページのバージョンを取得するには、次のリクエストでページ割りトークンを指定します。

リクエスト本文

リクエストにリクエスト本文がありません。

レスポンスの構文

```
HTTP/1.1 200
Content-type: application/json

{
   "intents": [
      {
         "createdDate": number,
         "description": "string",
         "lastUpdatedDate": number,
         "name": "string",
         "version": "string"
      }
   ],
   "nextToken": "string"
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

intents

IntentMetadata オブジェクトの配列で、インテントの各番号のバージョンに 1 つずつ、さらに $LATEST バージョンに 1 つあります。

型: IntentMetadata オブジェクトの配列

nextToken

インテントバージョンの次のページを取得するためのページ割りトークン。この呼び出しに対する応答が切り捨てられた場合、Amazon Lex は応答の中にページ割りトークンを返します。次のページのバージョンを取得するには、次のリクエストでページ割りトークンを指定します。

型: 文字列

エラー

BadRequestException

リクエストが適切にフォーマットされていません。例えば、値が無効であったり、必須項目が設定されていない場合です。フィールドの値を確認して、再度お試しください。

HTTP ステータスコード：400

InternalFailureException

Amazon Lex 内部エラーが発生しました。リクエストを再試行してください。

HTTP ステータスコード：500

LimitExceededException

リクエストが制限を超えました。リクエストを再試行してください。

HTTP ステータスコード: 429

NotFoundException

リクエストで指定されたリソースは見つかりませんでした。リソースを確認して、もう一度お試しください。

HTTP ステータスコード: 404

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go バージョン 2 用 SDK
- AWS Java V2 用 SDK
- AWS V3 用 JavaScript SDK
- AWS PHP V3 用SDK
- AWS Python 用 SDK
- AWS ルビー V3 用 SDK

GetMigration

サービス: Amazon Lex Model Building Service

Amazon Lex V1 ボットから Amazon Lex V2 ボットへの移行中または完了した移行についての詳細を提供します。このオペレーションを行うと、移行に関連するアラートや警告が表示されます。

リクエストの構文

```
GET /migrations/migrationId HTTP/1.1
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

migrationId

表示する移行の一意の識別子。migrationID は StartMigration のオペレーションで返されます。

長さの制限: 固定長は 10 です。

Pattern: ^[0-9a-zA-Z]+$

必須: はい

リクエストボディ

リクエストにリクエスト本文がありません。

レスポンスの構文

```
HTTP/1.1 200
Content-type: application/json

{
   "alerts": [
      {
         "details": [ "string" ],
         "message": "string",
         "referenceURLs": [ "string" ],
         "type": "string"
      }
```

```
    ],
    "migrationId": "string",
    "migrationStatus": "string",
    "migrationStrategy": "string",
    "migrationTimestamp": number,
    "v1BotLocale": "string",
    "v1BotName": "string",
    "v1BotVersion": "string",
    "v2BotId": "string",
    "v2BotRole": "string"
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

alerts

Amazon Lex V1 ボットから Amazon Lex V2 への移行に関する問題を示すアラートと警告のリスト。Amazon Lex V2 内で Amazon Lex V1 の機能の実装が異なる場合、警告が表示されます。

詳細については、「Amazon Lex V2 developer guide」(Amazon Lex V2 デベロッパーガイド) の「ボットを移行する」(Migrating a bot) を参照してください。

型: MigrationAlert オブジェクトの配列

migrationId

移行の一意の識別子。これは、`GetMigration` オペレーションを呼び出す際に使用される識別子と同じです。

型: 文字列

長さの制限: 固定長は 10 です。

Pattern: `^[0-9a-zA-Z]+$`

migrationStatus

移行のステータスを示します。ステータスが `COMPLETE` になると移行が完了し、Amazon Lex V2 でボットが利用できるようになります。移行を完了するには、アラートや警告を解決する必要がある場合があります。

型: 文字列

有効な値：IN_PROGRESS | COMPLETED | FAILED

migrationStrategy

移行を実行するために使用された戦略。

- CREATE_NEW - 新しい Amazon Lex V2 ボットを作成し、Amazon Lex V1 ボットをこの新しいボットに移行します。
- UPDATE_EXISTING - 既存の Amazon Lex V2 ボットのメタデータおよび移行されるロケールを上書きします。Amazon Lex V2 ボットの他のロケールは変更されません。ロケールが存在しない場合は、Amazon Lex V2 ボットに新しいロケールが作成されます。

型: 文字列

有効な値：CREATE_NEW | UPDATE_EXISTING

migrationTimestamp

移行が開始された日時。

型: タイムスタンプ

v1BotLocale

Amazon Lex V1 ボットのロケールが Amazon Lex V2 に移行しました。

型: 文字列

有効な値：de-DE | en-AU | en-GB | en-IN | en-US | es-419 | es-ES | es-US | fr-FR | fr-CA | it-IT | ja-JP | ko-KR

v1BotName

Amazon Lex V1 ボットの名前が Amazon Lex V2 に移行しました。

型: 文字列

長さの制限: 最小長は 2 です。最大長は 50 です。

Pattern: ^([A-Za-z]_?)+$

v1BotVersion

Amazon Lex V1 ボットのバージョンが Amazon Lex V2 に移行しました。

型: 文字列

長さの制限：最小長は 1 です。最大長は 64 文字です。

パターン: `\$LATEST|[0-9]+`

v2BotId

Amazon Lex V1 の移行先である Amazon Lex V2 ボットの一意の識別子。

型: 文字列

長さの制限: 固定長は 10 です。

Pattern: `^[0-9a-zA-Z]+$`

v2BotRole

Amazon Lex が Amazon Lex V2 ボットの実行に使用する IAM ロール。

型: 文字列

長さの制限: 最小長は 20 です。最大長は 2,048 です。

Pattern: `^arn:[\w\-]+:iam::[\d]{12}:role/.+$`

エラー

BadRequestException

リクエストが適切にフォーマットされていません。例えば、値が無効であったり、必須項目が設定されていない場合です。フィールドの値を確認して、再度お試しください。

HTTP ステータスコード：400

InternalFailureException

Amazon Lex 内部エラーが発生しました。リクエストを再試行してください。

HTTP ステータスコード：500

LimitExceededException

リクエストが制限を超えました。リクエストを再試行してください。

HTTP ステータスコード: 429

NotFoundException

リクエストで指定されたリソースは見つかりませんでした。リソースを確認して、もう一度お試しください。

HTTP ステータスコード: 404

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go バージョン 2 用 SDK
- AWS Java V2 用 SDK
- AWS V3 用 JavaScript SDK
- AWS PHP V3 用SDK
- AWS Python 用 SDK
- AWS ルビー V3 用 SDK

GetMigrations

サービス: Amazon Lex Model Building Service

Amazon Lex V1 と Amazon Lex V2 間の移行の一覧を取得します。

リクエストの構文

```
GET /migrations?
maxResults=maxResults&migrationStatusEquals=migrationStatusEquals&nextToken=nextToken&sortByAtt
 HTTP/1.1
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

maxResults

レスポンスに返される移行の最大数。デフォルトは 10 です。

有効範囲: 最小値 は 1 です。最大値は 50 です。

migrationStatusEquals

指定した状態のマイグレーションのみを含むようにリストをフィルタリングします。

有効な値：`IN_PROGRESS | COMPLETED | FAILED`

nextToken

マイグレーションの次のページを取得するページ割りトークン。このオペレーションに対するレスポンスが切り捨てられた場合、Amazon Lex はレスポンスでページ割りトークンを返します。マイグレーションの次のページを取得するには、リクエストでページ割りトークンを指定します。

sortByAttribute

マイグレーションのリストをソートするフィールド。Amazon Lex V1 のボット名や、移行が開始された日時でソートすることができます。

有効な値：`V1_BOT_NAME | MIGRATION_DATE_TIME`

sortByOrder

順番はリストをソートします。

有効な値：ASCENDING | DESCENDING

v1BotNameContains

リストにフィルタを適用して、指定した文字列を含む名前を持つボットのみを含めます。文字列はボット名の任意の場所で一致します。

長さの制限: 最小長は 2 です。最大長は 50 です。

Pattern: ^([A-Za-z]_?)+$

リクエストボディ

リクエストにリクエスト本文がありません。

レスポンスの構文

```
HTTP/1.1 200
Content-type: application/json

{
   "migrationSummaries": [
      {
         "migrationId": "string",
         "migrationStatus": "string",
         "migrationStrategy": "string",
         "migrationTimestamp": number,
         "v1BotLocale": "string",
         "v1BotName": "string",
         "v1BotVersion": "string",
         "v2BotId": "string",
         "v2BotRole": "string"
      }
   ],
   "nextToken": "string"
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

migrationSummaries

Amazon Lex V1 から Amazon Lex V2 への移行に関する概要の配列。移行の詳細を確認するには、GetMigration のオペレーションの呼び出しでサマリーから `migrationId` を使用します。

型: MigrationSummary オブジェクトの配列

nextToken

レスポンスが切り捨てられる場合は、次のマイグレーションページを取得するために次のリクエストで指定できるページ割りトークンが含まれます。

型: 文字列

エラー

BadRequestException

リクエストが適切にフォーマットされていません。例えば、値が無効であったり、必須項目が設定されていない場合です。フィールドの値を確認して、再度お試しください。

HTTP ステータスコード：400

InternalFailureException

Amazon Lex 内部エラーが発生しました。リクエストを再試行してください。

HTTP ステータスコード：500

LimitExceededException

リクエストが制限を超えました。リクエストを再試行してください。

HTTP ステータスコード: 429

以下も参照してください。

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++

- AWS Go バージョン 2 用 SDK
- AWS Java V2 用 SDK
- AWS V3 用 JavaScript SDK
- AWS PHP V3 用SDK
- AWS Python 用 SDK
- AWS ルビー V3 用 SDK

GetSlotType

サービス: Amazon Lex Model Building Service

スロットタイプの特定のバージョンに関する情報を返します。スロットタイプ名に加えて、スロットタイプバージョンを指定する必要があります。

このオペレーションには、`lex:GetSlotType` アクションに対する許可が必要です。

リクエストの構文

```
GET /slottypes/name/versions/version HTTP/1.1
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

name

スロットタイプの名前。名前は、大文字と小文字が区別されます。

長さの制限：最小長は 1 です。最大長は 100 です。

パターン: `^([A-Za-z]_?)+$`

必須：はい

version

スロットタイプのバージョン。

長さの制限：最小長は 1 です。最大長は 64 文字です。

パターン: `\$LATEST|[0-9]+`

必須: はい

リクエストボディ

リクエストにリクエスト本文がありません。

レスポンスの構文

```
HTTP/1.1 200
Content-type: application/json
```

```
{
   "checksum": "string",
   "createdDate": number,
   "description": "string",
   "enumerationValues": [
      {
         "synonyms": [ "string" ],
         "value": "string"
      }
   ],
   "lastUpdatedDate": number,
   "name": "string",
   "parentSlotTypeSignature": "string",
   "slotTypeConfigurations": [
      {
         "regexConfiguration": {
            "pattern": "string"
         }
      }
   ],
   "valueSelectionStrategy": "string",
   "version": "string"
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

checksum

スロットタイプの $LATEST バージョンのチェックサム。

型: 文字列

createdDate

スロットタイプが作成された日付。

型: タイムスタンプ

description

スロットタイプの説明。

型: 文字列

長さの制限: 最小長は 0 です。最大長は 200 です。

enumerationValues

スロットタイプが取得できる値を定義する EnumerationValue オブジェクトのリスト。

型: EnumerationValue オブジェクトの配列

配列メンバー : 最小数は 0 項目です。最大数は 10000 項目です。

lastUpdatedDate

スロットタイプが更新された日付。リソースを作成する場合、作成日と最終更新日は同じ日付になります。

型: タイムスタンプ

name

スロットタイプの名前。

型: 文字列

長さの制限: 最小長は 1 です。最大長は 100 です。

パターン: ^([A-Za-z]_?)+$

parentSlotTypeSignature

スロットタイプの親として使用される組み込みスロットタイプ。

型: 文字列

長さの制限: 最小長は 1 です。最大長は 100 です。

パターン: ^((AMAZON\.)_?|[A-Za-z]_?)+

slotTypeConfigurations

親組み込みスロットタイプを拡張する構成情報。

型: SlotTypeConfiguration オブジェクトの配列

の配列メンバー: 最小数は 0 項目です。最大数は 10 項目です。

valueSelectionStrategy

Amazon Lex がスロットの価値を決定するために使用する戦略。詳細については、「PutSlotType」を参照してください。

型: 文字列

有効な値：ORIGINAL_VALUE | TOP_RESOLUTION

version

スロットタイプのバージョン。

型: 文字列

長さの制限：最小長は 1 です。最大長は 64 文字です。

パターン：\$LATEST|[0-9]+

エラー

BadRequestException

リクエストが適切にフォーマットされていません。例えば、値が無効であったり、必須項目が設定されていない場合です。フィールドの値を確認して、再度お試しください。

HTTP ステータスコード：400

InternalFailureException

Amazon Lex 内部エラーが発生しました。リクエストを再試行してください。

HTTP ステータスコード：500

LimitExceededException

リクエストが制限を超えました。リクエストを再試行してください。

HTTP ステータスコード: 429

NotFoundException

リクエストで指定されたリソースは見つかりませんでした。リソースを確認して、もう一度お試しください。

HTTP ステータスコード: 404

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go バージョン 2 用 SDK
- AWS Java V2 用 SDK
- AWS V3 用 JavaScript SDK
- AWS PHP V3 用SDK
- AWS Python 用 SDK
- AWS ルビー V3 用 SDK

GetSlotTypes

サービス: Amazon Lex Model Building Service

スロットタイプの情報を次のように返します。

- nameContains フィールドを指定した場合は、指定した文字列を含むすべてのスロットタイプの $LATEST バージョン情報を返します。
- nameContains フィールドを指定しない場合は、すべてのスロットタイプの $LATEST バージョン情報を返します。

オペレーションには lex:GetSlotTypes アクションに対するアクセス許可が必要です。

リクエストの構文

```
GET /slottypes/?maxResults=maxResults&nameContains=nameContains&nextToken=nextToken
 HTTP/1.1
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

maxResults

レスポンスに返されるスロットタイプの最大数。デフォルトは 10 です。

有効範囲: 最小値 は 1 です。最大値は 50 です。

nameContains

スロットタイプ名に一致する部分文字列。スロットタイプの名前の一部が部分文字列と一致する場合、スロットタイプが返されます。例えば、「xyz」は「xyzabc」と「abcxyz」の両方に一致します。

長さの制限：最小長は 1 です。最大長は 100 です。

パターン: ^([A-Za-z]_?)+$

nextToken

スロットタイプの次のページを取得するページ割りトークン。この API コールに対するレスポンスが切り捨てられた場合、Amazon Lex はレスポンスでページ割りトークンを返します。スロットタイプの次のページを取得するには、次のリクエストでページ割りトークンを指定します。

リクエスト本文

リクエストにリクエスト本文がありません。

レスポンスの構文

```
HTTP/1.1 200
Content-type: application/json

{
   "nextToken": "string",
   "slotTypes": [
      {
         "createdDate": number,
         "description": "string",
         "lastUpdatedDate": number,
         "name": "string",
         "version": "string"
      }
   ]
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

nextToken

レスポンスが切り捨てられる場合は、次のリクエストでスロットタイプの次のページを取得するために指定できるページ割りトークンが含まれます。

型: 文字列

slotTypes

スロットタイプごとに 1 つずつ、スロットタイプの名前、バージョン、説明などの情報を提供するオブジェクトの配列。

型: SlotTypeMetadata オブジェクトの配列

エラー

BadRequestException

リクエストが適切にフォーマットされていません。例えば、値が無効であったり、必須項目が設定されていない場合です。フィールドの値を確認して、再度お試しください。

HTTP ステータスコード：400

InternalFailureException

Amazon Lex 内部エラーが発生しました。リクエストを再試行してください。

HTTP ステータスコード：500

LimitExceededException

リクエストが制限を超えました。リクエストを再試行してください。

HTTP ステータスコード: 429

NotFoundException

リクエストで指定されたリソースは見つかりませんでした。リソースを確認して、もう一度お試しください。

HTTP ステータスコード: 404

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go バージョン 2 用 SDK
- AWS Java V2 用 SDK
- AWS V3 用 JavaScript SDK
- AWS PHP V3 用SDK
- AWS Python 用 SDK
- AWS ルビー V3 用 SDK

GetSlotTypeVersions

サービス: Amazon Lex Model Building Service

スロットタイプのすべてのバージョンに関する情報を取得します。

GetSlotTypeVersions オペレーションは、スロットタイプの各バージョンの SlotTypeMetadata オブジェクトを返します。例えば、あるスロットタイプに 3 つの番号が付いたバージョンがある場合、GetSlotTypeVersions オペレーションは、各番号のバージョンに 1 つ、$LATEST バージョンに 1 つ、合計 4 つの SlotTypeMetadata オブジェクトをレスポンスに返します。

GetSlotTypeVersions オペレーションは、少なくとも 1 つの $LATEST バージョンを常に返します。

このオペレーションには、lex:GetSlotTypeVersions アクションに対する許可が必要です。

リクエストの構文

```
GET /slottypes/name/versions/?maxResults=maxResults&nextToken=nextToken HTTP/1.1
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

maxResults

レスポンスに返されるスロットタイプバージョンの最大数。デフォルトは 10 です。

有効範囲: 最小値 は 1 です。最大値は 50 です。

name

バージョンを返すスロットタイプの名前です。

長さの制限：最小長は 1 です。最大長は 100 です。

パターン: ^([A-Za-z]_?)+$

必須：はい

nextToken

スロットタイプのバージョンの次のページを取得するためのページ割りトークン。この呼び出しに対する応答が切り捨てられた場合、Amazon Lex は応答の中にページ割りトークンを返しま

す。次のページのバージョンを取得するには、次のリクエストでページ割りトークンを指定します。

リクエスト本文

リクエストにリクエスト本文がありません。

レスポンスの構文

```
HTTP/1.1 200
Content-type: application/json

{
   "nextToken": "string",
   "slotTypes": [
      {
         "createdDate": number,
         "description": "string",
         "lastUpdatedDate": number,
         "name": "string",
         "version": "string"
      }
   ]
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

nextToken

スロットタイプのバージョンの次のページを取得するためのページ割りトークン。この呼び出しに対する応答が切り捨てられた場合、Amazon Lex は応答の中にページ割りトークンを返します。次のページのバージョンを取得するには、次のリクエストでページ割りトークンを指定します。

型: 文字列

slotTypes

SlotTypeMetadata オブジェクトの配列で、スロットタイプの番号付きバージョンごとに 1 つ、$LATEST バージョンに 1 つです。

型: SlotTypeMetadata オブジェクトの配列

エラー

BadRequestException

リクエストが適切にフォーマットされていません。例えば、値が無効であったり、必須項目が設定されていない場合です。フィールドの値を確認して、再度お試しください。

HTTP ステータスコード : 400

InternalFailureException

Amazon Lex 内部エラーが発生しました。リクエストを再試行してください。

HTTP ステータスコード : 500

LimitExceededException

リクエストが制限を超えました。リクエストを再試行してください。

HTTP ステータスコード: 429

NotFoundException

リクエストで指定されたリソースは見つかりませんでした。リソースを確認して、もう一度お試しください。

HTTP ステータスコード: 404

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++

- AWS Go バージョン 2 用 SDK
- AWS Java V2 用 SDK
- AWS V3 用 JavaScript SDK
- AWS PHP V3 用SDK
- AWS Python 用 SDK
- AWS ルビー V3 用 SDK

GetUtterancesView

サービス: Amazon Lex Model Building Service

`GetUtterancesView` のオペレーションを使用して、ユーザーがボットに対して行った発話に関する情報を取得します。このリストを使用して、ボットが応答する発話を調整できます。

例: 花を注文するボットを作成したとします。ユーザーがボットをしばらく使用した後、`GetUtterancesView` オペレーションを使用して、ユーザーが行ったリクエストとその成功の有無を確認します。「花が欲しい」という発話が認識されていないかもしれません。この発話を `OrderFlowers` インテントに追加することで、ボットがその発話を認識するようになります。

ボットの新しいバージョンを公開すると、古いバージョンと新しいバージョンに関する情報を取得して、2 つのバージョン間でパフォーマンスを比較できます。

発話の統計は 1 日に 1 回生成されます。過去 15 日間のデータを使用できます。1 回のリクエストで最大 5 バージョンのボットの情報をリクエストできます。Amazon Lex は、過去 15 日間にボットが受信した最大頻度の発話を返します。レスポンスには、バージョンごとに最大 100 の発話に関する情報が含まれます。

発話の統計は、以下の条件では生成されません。

- ボットが作成されたとき、`childDirected` フィールドが TRUE に設定されます。
- 1 つ以上のスロットでスロットの難読化を実行しています。
- Amazon Lex の改善への参加をオプトアウトしました。

このオペレーションには、`lex:GetUtterancesView` アクションに対する許可が必要です。

リクエストの構文

```
GET /bots/botname/utterances?
view=aggregation&bot_versions=botVersions&status_type=statusType HTTP/1.1
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

botname

発話情報を返すボットの名前。

長さの制限: 最小長は 2 です。最大長は 50 です。

Pattern: ^([A-Za-z]_?)+$

必須：はい

botVersions

発話情報を返す必要があるボットバージョンの配列。制限は、リクエストごとに 5 バージョンです。

配列メンバー：最小数は 1 項目です。最大数は 5 項目です。

長さの制限：最小長は 1 です。最大長は 64 文字です。

パターン: \$LATEST|[0-9]+

必須：はい

statusType

認識された発話を返すには、Detected を使用します。認識されなかった発話を返すには、Missed を使用します。

有効な値：Detected | Missed

必須: はい

リクエストボディ

リクエストにリクエスト本文がありません。

レスポンスの構文

```
HTTP/1.1 200
Content-type: application/json

{
   "botName": "string",
   "utterances": [
      {
         "botVersion": "string",
         "utterances": [
```

```
            {
               "count": number,
               "distinctUsers": number,
               "firstUtteredDate": number,
               "lastUtteredDate": number,
               "utteranceString": "string"
            }
         ]
      }
   ]
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

botName

利用者情報を返すボットの名前。

型: 文字列

長さの制限: 最小長は 2 です。最大長は 50 です。

Pattern: `^([A-Za-z]_?)+$`

utterances

UtteranceList オブジェクトの配列で、それぞれがボットで処理された発話を記述する UtteranceData オブジェクトのリストを含みます。レスポンスには、各バージョンごとに最大 100 個の `UtteranceData` オブジェクトが含まれます。Amazon Lex は、過去 15 日間にボットが受信した最大頻度の発話を返します。

型: UtteranceList オブジェクトの配列

エラー

BadRequestException

リクエストが適切にフォーマットされていません。例えば、値が無効であったり、必須項目が設定されていない場合です。フィールドの値を確認して、再度お試しください。

HTTP ステータスコード：400

InternalFailureException

Amazon Lex 内部エラーが発生しました。リクエストを再試行してください。

HTTP ステータスコード：500

LimitExceededException

リクエストが制限を超えました。リクエストを再試行してください。

HTTP ステータスコード: 429

以下も参照してください。

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go バージョン 2 用 SDK
- AWS Java V2 用 SDK
- AWS V3 用 JavaScript SDK
- AWS PHP V3 用SDK
- AWS Python 用 SDK
- AWS ルビー V3 用 SDK

ListTagsForResource

サービス: Amazon Lex Model Building Service

指定のリソースグループに関連付けられたタグを取得します。ボット、ボットエイリアス、ボットチャンネルのみに、タグを関連付けることができます。

リクエストの構文

```
GET /tags/resourceArn HTTP/1.1
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

resourceArn

タグのリストを取得するリソースの Amazon リソースネーム (ARN)。

長さの制限：最小長は 1 です。最大長は 1011 です。

必須: はい

リクエストボディ

リクエストにリクエスト本文がありません。

レスポンスの構文

```
HTTP/1.1 200
Content-type: application/json

{
   "tags": [
      {
         "key": "string",
         "value": "string"
      }
   ]
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

tags

リソースに関連付けられたタグ。

型: Tag オブジェクトの配列

配列メンバー: 最小数は 0 項目です。最大数は 200 項目です。

エラー

BadRequestException

リクエストが適切にフォーマットされていません。例えば、値が無効であったり、必須項目が設定されていない場合です。フィールドの値を確認して、再度お試しください。

HTTP ステータスコード : 400

InternalFailureException

Amazon Lex 内部エラーが発生しました。リクエストを再試行してください。

HTTP ステータスコード : 500

LimitExceededException

リクエストが制限を超えました。リクエストを再試行してください。

HTTP ステータスコード: 429

NotFoundException

リクエストで指定されたリソースは見つかりませんでした。リソースを確認して、もう一度お試しください。

HTTP ステータスコード: 404

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET

- AWS SDK for C++
- AWS Go バージョン 2 用 SDK
- AWS Java V2 用 SDK
- AWS V3 用 JavaScript SDK
- AWS PHP V3 用SDK
- AWS Python 用 SDK
- AWS ルビー V3 用 SDK

PutBot

サービス: Amazon Lex Model Building Service

Amazon Lex 会話ボットを作成、または既存のボットを置き換えます。ボットを作成または更新する際に必要なのは、名前、ロケール、およびボットが 13 歳未満の子供を対象としているかどうかを指定することだけです。これにより、後からインテントを追加したり、既存のボットからインテントを削除したりすることができます。最低限の情報でボットを作成した場合、ボットの作成や更新は行われますが、Amazon Lex は の応答 FAILED を返します。1 つ以上のインテントを追加すると、ボットを構築できます。Amazon Lex ボットの詳細については、「Amazon Lex: 仕組み」を参照してください。

既存のボットの名前を指定した場合、リクエストのフィールドは、$LATEST バージョンのボットの既存の値に置き換えられます。 Amazon Lex ではリクエストで値が入力されていないフィールドは削除されますが、idleTTLInSeconds と privacySettings のフィールドはデフォルト値に設定されます。必須フィールドに値を指定しない場合、Amazon Lex は例外をスローします。

このオペレーションには、lex:PutBot アクションに対するアクセス許可が必要です。詳細については、「Amazon Lex のための Identity and Access Management」を参照してください。

リクエストの構文

```
PUT /bots/name/versions/$LATEST HTTP/1.1
Content-type: application/json

{
   "abortStatement": {
      "messages": [
         {
            "content": "string",
            "contentType": "string",
            "groupNumber": number
         }
      ],
      "responseCard": "string"
   },
   "checksum": "string",
   "childDirected": boolean,
   "clarificationPrompt": {
      "maxAttempts": number,
      "messages": [
         {
```

```
            "content": "string",
            "contentType": "string",
            "groupNumber": number
         }
      ],
      "responseCard": "string"
   },
   "createVersion": boolean,
   "description": "string",
   "detectSentiment": boolean,
   "enableModelImprovements": boolean,
   "idleSessionTTLInSeconds": number,
   "intents": [
      {
         "intentName": "string",
         "intentVersion": "string"
      }
   ],
   "locale": "string",
   "nluIntentConfidenceThreshold": number,
   "processBehavior": "string",
   "tags": [
      {
         "key": "string",
         "value": "string"
      }
   ],
   "voiceId": "string"
}
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

name

ボットの名前。キースペース名では、大文字と小文字は区別されません。

長さの制限: 最小長は 2 です。最大長は 50 です。

Pattern: ^([A-Za-z]_?)+$

必須: はい

リクエストボディ

リクエストは以下の JSON 形式のデータを受け入れます。

abortStatement

Amazon Lexは、ユーザーが入力したコンテキストを理解できない場合、何度か情報を聞き出そうとします。Amazon Lex は次に abortStatement で定義されたメッセージをユーザーに送信してから、会話をキャンセルします。リトライの回数を設定するには、スロットタイプの valueElicitationPrompt フィールドを使用します。

例えば、ピザ注文ボットでは、Amazon Lex がユーザーに「どんな種類のクラストが欲しいですか？」と尋ねるかもしれません。ユーザーの回答が想定される回答 (例えば、「薄い生地」、「ディープディッシュ」など) ではない場合、Amazon Lex はさらに数回質問をし、正しい回答を引き出そうとします。

例えば、ピザを注文するアプリケーションでは、OrderPizza がインテントの 1 つになります。このインテントには、CrustType スロットを必要かもしれません。CrustType スロットの作成時に valueElicitationPrompt フィールドを指定します。

フォールバックインテントを定義している場合、キャンセルステートメントはユーザーに送信されず、代わりにフォールバックインテントが使用されます。詳細については、「AMAZON」を参照してください。 FallbackIntent。

タイプ：Statement オブジェクト

必須: いいえ

checksum

$LATEST バージョンの特定のリビジョンを識別します。

新しいボットを作成する場合は、checksum フィールドを空白にします。チェックサムを指定した場合、BadRequestException の例外が発生します。

ボットを更新する場合は、checksum フィールドに $LATEST バージョンの最新リビジョンのチェックサムを設定します。 checksum フィールドを指定しない場合や、チェックサムが $LATEST バージョンと一致しない場合は、PreconditionFailedException 例外が発生します。

タイプ: 文字列

必須: いいえ

childDirected

Amazon Lex Model·Building·Service で作成された各 Amazon Lex ボットについて、Amazon Lex 全体または一部の使用が、13 歳未満の児童を対象とするウェブサイト、プログラム、その他のアプリケーションに関連しており、COPPA (Children's Online Privacy Protection Act) の対象となっているかどうかを、`childDirected` フィールドに `true` または `false` を指定する必要があります。`childDirected` フィールドに `true` を指定することで、Amazon Lex 全体または一部の使用が、13 歳未満の児童を対象とするウェブサイト、プログラム、その他のアプリケーションに関連しており、COPPA の対象になることに同意します。`childDirected` フィールドに `false` を指定することで、Amazon Lex 全体または一部の使用が、13 歳未満の児童を対象とするウェブサイト、プログラム、その他のアプリケーションに関連しておらず、COPPA の対象にならないことに同意します。Amazon Lex の全体または一部の使用が、13歳未満の児童を対象にしており、COPPAの対象となるウェブサイト、プログラム、またはその他のアプリケーションに関連するかどうかを正確に設定するために、`childDirected` フィールドにデフォルト値を指定することはできません。

Amazon Lex 全体または一部の使用が、13 歳未満の児童を対象とするウェブサイト、プログラム、その他のアプリケーションに関連する場合は、COPPA により必要とされる検証可能な保護者の同意が必要です。13 歳未満の児童を対象とするウェブサイト、プログラム、またはその他のアプリケーションに関連する Amazon Lex の全体または一部の使用についての詳細は、「Amazon Lex FAQ」を参照してください。

型: ブール値

必須: はい

clarificationPrompt

Amazon Lex がユーザーのインテントを理解できない場合、このメッセージを使用して明確化します。Amazon Lex が明確化のプロンプトを何回繰り返すかを `maxAttempts` フィールドで指定します。それでも Amazon Lex が理解できない場合は、`abortStatement` フィールドでメッセージを送信します。

明瞭化のためのプロンプトを作成する際には、ユーザーからの正しい回答を示唆するようにしてください。例えば、ピザとドリンクを注文するボットの場合は、このような明確化のためのプロンプトを作成します。「どうなさいますか？「ピザを注文」または「飲み物を注文」と話してください」

フォールバックインテントを定義している場合は、maxAttempts フィールドで定義された回数だけ明確化のプロンプトが繰り返された場合に起動されます。詳細については、AMAZON を参照してください。 FallbackIntent。

明示的なプロンプトを定義しない場合、実行時に Amazon Lex は 3 つのケースで 400 Bad Request の例外を返します。

- フォローアッププロンプト - ユーザーがフォローアッププロンプトに応答しても、インテントが提供されない場合。例えば、「今日は他に何かご希望はありますか？」というフォローアッププロンプトに応答して、ユーザーが「はい」答えた場合。Amazon Lexは、インテントを取得するためにユーザーへ送る明確なプロンプトがないため、400 Bad Request例外を返します。
- Lambda 関数 - Lambda 関数を使用する場合、ElicitIntent のダイアログタイプを返します。Amazon Lex には、ユーザーからインテントを取得するための明確化プロンプトがないため、400 Bad Request 例外が返されます。
- PutSession 操作-PutSession 操作を使用するときは、ElicitIntentダイアログタイプを送信します。Amazon Lex には、ユーザーからインテントを取得するための明確化プロンプトがないため、400 Bad Request 例外が返されます。

型: Prompt オブジェクト

必須: いいえ

createVersion

true に設定すると、新しく採番されたバージョンのボットが作成されます。これは、CreateBotVersion のオペレーションを呼び出すのと同じです。createVersion を指定しなかった場合、デフォルトは false です。

型: ブール値

必須: いいえ

description

ボットの説明。

型: 文字列

長さの制限: 最小長は 0 です。最大長は 200 です。

必須: いいえ

detectSentiment

true に設定すると、ユーザーの発話はセンチメント分析のために Amazon Comprehend へ送られます。detectSentiment を指定しなかった場合、デフォルトは false です。

型: ブール値

必須: いいえ

enableModelImprovements

true に設定すると、自然言語理解の向上にアクセスできるようになります。

enableModelImprovements パラメータを true に設定すると、nluIntentConfidenceThreshold パラメータを使って信頼度スコアを設定できます。詳細については、「Confidence Scores」(信頼スコア) を参照してください。

enableModelImprovements パラメータは、特定のリージョンでしか設定できません。パラメータを true に設定すると、ボットは精度の向上にアクセスできるようになります。

en-US ロケールで enableModelImprovements パラメータを false に設定できるリージョンは、以下の通りです。

- 米国東部 (バージニア北部) (us-east-1)
- 米国西部 (オレゴン) (us-west-2)
- アジアパシフィック (シドニー) (ap-southeast-2)
- 欧州 (アイルランド) (eu-west-1)

他のリージョンとロケールでは、enableModelImprovements パラメータはデフォルトで true に設定されています。これらのリージョンとロケールでは、パラメータを false に設定すると ValidationException の例外が発生します。

型: ブール値

必須: いいえ

idleSessionTTLInSeconds

Amazon Lex が会話で収集したデータを保持する最大時間 (秒) です。

ユーザーインタラクションセッションは、指定された時間の間はアクティブのままです。この間に会話が発生しない場合、セッションは期限切れになり、Amazon Lex はタイムアウト前に提供されたデータをすべて削除します。

たとえば、 OrderPizza ユーザーがインテントを選択したのに、注文の途中で道に迷ったとします。ユーザーは指定した時間内に注文を完了をできなかった場合、Amazon Lex は収集したスロット情報を破棄し、ユーザーは最初からやり直す必要があります。

PutBot オペレーションリクエストに idleSessionTTLInSeconds 要素を含まない場合、Amazon Lex はデフォルト値を使用します。これは、リクエストが既存のボットを置き換える場合も同様です。

デフォルトは 300 秒 (5 分) です。

型: 整数

値の範囲: 最小値 は 60 です。最大値は 86400 です。

必須: いいえ

intents

Intent オブジェクトの配列。各インテントは、ユーザーが表現できるコマンドを表しています。たとえば、ピザを注文するボットがインテントをサポートしているとします。 OrderPizza 詳細については、「Amazon Lex: 仕組み」を参照してください。

型: Intent オブジェクトの配列

必須: いいえ

locale

ボットのターゲットロケールを指定します。ボットで使用するインテントは、ボットのロケールと互換性がある必要があります。

デフォルトは en-US です。

型: 文字列

有効な値 : de-DE | en-AU | en-GB | en-IN | en-US | es-419 | es-ES | es-US | fr-FR | fr-CA | it-IT | ja-JP | ko-KR

必須: はい

nluIntentConfidenceThreshold

Amazon Lex が OR PostTextレスポンスで代替インテントを返すときにAMAZON.FallbackIntentAMAZON.KendraSearchIntent、、、または両

方を挿入するしきい値を決定します。PostContent AMAZON.FallbackIntentまた、AMAZON.KendraSearchIntentボット用に設定されている場合にのみ挿入されます。

以下のリージョンで信頼性スコアを使用するには、enableModelImprovements パラメータを true に設定する必要があります。

- 米国東部 (バージニア北部) (us-east-1)
- 米国西部 (オレゴン) (us-west-2)
- アジアパシフィック (シドニー) (ap-southeast-2)
- 欧州 (アイルランド) (eu-west-1)

他のリージョンでは、enableModelImprovements パラメータはデフォルトで true に設定されています。

例えば、信頼度の閾値が 0.80 で、AMAZON.FallbackIntent を設定したボットがあったとします。Amazon Lex は、以下の 3 つの代替インテントを返します：IntentA (0.70)、IntentB (0.60)、intentC (0.50)。PostText オペレーションからのレスポンスは次のようになります。

- アマゾン。 FallbackIntent
- IntentA
- IntentB
- IntentC

型: 倍精度浮動小数点数

有効な範囲：最小値 は 0 です。最大値は 1 です。

必須: いいえ

processBehavior

processBehavior 要素を BUILD に設定すると、Amazon Lex は実行可能なボットを構築します。要素を SAVE にした場合、Amazon Lex はボットを保存しますが、構築はしません。

この値を指定しない場合、デフォルト値は BUILD です。

型: 文字列

有効な値：SAVE | BUILD

必須：いいえ

tags

ボットに追加するタグのリスト。タグの追加は、ボットの作成時にのみ可能で、PutBot のオペレーションでボットのタグを更新することはできません。タグを更新するには、TagResource のオペレーションを使用します。

型: Tag オブジェクトの配列

配列メンバー: 最小数は 0 項目です。最大数は 200 項目です。

必須: いいえ

voiceId

Amazon Lex がユーザーとの音声対話に使用する Amazon Polly の音声ID です。音声に設定された口ケールは、ボットの口ケールと一致する必要があります。Amazon Polly の詳細については、「Amazon Polly Developer Guide」(Amazon Polly デベロッパーガイド) の「Voices in Amazon Polly」(Amazon Polly における音声) を参照してください。

タイプ: 文字列

必須: いいえ

レスポンスの構文

```
HTTP/1.1 200
Content-type: application/json

{
   "abortStatement": {
      "messages": [
         {
            "content": "string",
            "contentType": "string",
            "groupNumber": number
         }
      ],
      "responseCard": "string"
   },
   "checksum": "string",
   "childDirected": boolean,
```

```
   "clarificationPrompt": {
      "maxAttempts": number,
      "messages": [
         {
            "content": "string",
            "contentType": "string",
            "groupNumber": number
         }
      ],
      "responseCard": "string"
   },
   "createdDate": number,
   "createVersion": boolean,
   "description": "string",
   "detectSentiment": boolean,
   "enableModelImprovements": boolean,
   "failureReason": "string",
   "idleSessionTTLInSeconds": number,
   "intents": [
      {
         "intentName": "string",
         "intentVersion": "string"
      }
   ],
   "lastUpdatedDate": number,
   "locale": "string",
   "name": "string",
   "nluIntentConfidenceThreshold": number,
   "status": "string",
   "tags": [
      {
         "key": "string",
         "value": "string"
      }
   ],
   "version": "string",
   "voiceId": "string"
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

abortStatement

Amazon Lex が会話をキャンセルするために使用するメッセージ。詳細については、「PutBot」を参照してください。

型: Statement オブジェクト

checksum

作成したボットのチェックサム。

型: 文字列

childDirected

Amazon Lex Model·Building·Service で作成された各 Amazon Lex ボットについて、Amazon Lex 全体または一部の使用が、13 歳未満の児童を対象とするウェブサイト、プログラム、その他のアプリケーションに関連しており、COPPA (Children's Online Privacy Protection Act) の対象となっているかどうかを、`childDirected` フィールドに `true` または `false` を指定する必要があります。`childDirected` フィールドに `true` を指定することで、Amazon Lex 全体または一部の使用が、13 歳未満の児童を対象とするウェブサイト、プログラム、その他のアプリケーションに関連しており、COPPA の対象になることに同意します。`childDirected` フィールドに `false` を指定することで、Amazon Lex 全体または一部の使用が、13 歳未満の児童を対象とするウェブサイト、プログラム、その他のアプリケーションに関連しておらず、COPPA の対象にならないことに同意します。Amazon Lex の全体または一部の使用が、13歳未満の児童を対象にしており、COPPAの対象となるウェブサイト、プログラム、またはその他のアプリケーションに関連するかどうかを正確に設定するために、`childDirected` フィールドにデフォルト値を指定することはできません。

Amazon Lex 全体または一部の使用が、13 歳未満の児童を対象とするウェブサイト、プログラム、その他のアプリケーションに関連する場合は、COPPA により必要とされる検証可能な保護者の同意が必要です。13 歳未満の児童を対象とするウェブサイト、プログラム、またはその他のアプリケーションに関連する Amazon Lex の全体または一部の使用についての詳細は、「Amazon Lex FAQ」を参照してください。

型: ブール値

clarificationPrompt

Amazon Lex がユーザーのインテントを理解できない場合に、使用するプロンプト。詳細については、「PutBot」を参照してください。

型: Prompt オブジェクト

createdDate

ボットが作成された日付。

型: タイムスタンプ

createVersion

新しいバージョンのボットが作成された場合の True。リクエストで createVersion フィールドが指定されていない場合は、レスポンスで createVersion フィールドが false に設定されます。

型: ブール値

description

ボットの説明。

型: 文字列

長さの制限: 最小長は 0 です。最大長は 200 です。

detectSentiment

センチメント分析のためにユーザーの発話を Amazon Comprehend に送信するようにボットが設定されている場合の true。リクエストで detectSentiment フィールドが指定されていない場合、detectSentiment フィールドはレスポンスでは false になります。

型: ブール値

enableModelImprovements

ボットが精度向上を使用しているかどうかを示します。true はボットが精度向上を使用していることを示し、それ以外は false となります。

型: ブール値

failureReason

status が FAILED の場合、Amazon Lex はボットの構築に失敗した理由を提示します。

型: 文字列

idleSessionTTLInSeconds

Amazon Lex が会話の中で収集したデータを保持する最大時間。詳細については、「PutBot」を参照してください。

型: 整数

値の範囲: 最小値 は 60 です。最大値は 86400 です。

intents

Intent オブジェクトの配列。詳細については、「PutBot」を参照してください。

型: Intent オブジェクトの配列

lastUpdatedDate

ボットが更新された日付。リソースを作成すると、作成日と最終更新日は同じ日付になります。

型: タイムスタンプ

locale

ボットのターゲットロケール。

型: 文字列

有効な値 : de-DE | en-AU | en-GB | en-IN | en-US | es-419 | es-ES | es-US | fr-FR | fr-CA | it-IT | ja-JP | ko-KR

name

ボットの名前。

型: 文字列

長さの制限: 最小長は 2 です。最大長は 50 です。

Pattern: ^([A-Za-z]_?)+$

nluIntentConfidenceThreshold

OR レスポンスで代替インテントを返すときに Amazon Lex がAMAZON.FallbackIntentAMAZON.KendraSearchIntent、、または両方を挿入する位置を決定するスコア。PostContentPostText AMAZON.FallbackIntentすべてのインテントの信頼度スコアがこの値を下回る場合に挿入されます。 AMAZON.KendraSearchIntentボット用に設定されている場合にのみ挿入されます。

型: 倍精度

有効な範囲 : 最小値 は 0 です。最大値は 1 です。

status

processBehavior を BUILD に設定したボット作成リクエストを送信すると、Amazon Lexは status レスポンス要素を BUILDING に設定します。

READY_BASIC_TESTING の状態では、スロットタイプでボットの意図や値として設定された発話と正確に一致するユーザーの入力でボットをテストすることができます。

Amazon Lex がボットを構築できない場合、Amazon Lex は status を FAILED に設定します。Amazon Lex は、failureReason レスポンス要素に失敗の理由を返します。

processBehavior を SAVE に設定すると、Amazon Lex はステータスコードを NOT BUILT に設定します。

ボットが READY の状態になったら、ボットをテストして公開することができます。

型: 文字列

有効な値 : BUILDING | READY | READY_BASIC_TESTING | FAILED | NOT_BUILT

tags

ボットに関連付けられているタグのリスト。

型: Tag オブジェクトの配列

配列メンバー: 最小数は 0 項目です。最大数は 200 項目です。

version

ボットのバージョン。新規のボットの場合、バージョンは常に $LATEST です。

型: 文字列

長さの制限 : 最小長は 1 です。最大長は 64 文字です。

パターン: \$LATEST|[0-9]+

voiceId

Amazon Lex がユーザーとの音声対話に使用する Amazon Polly 音声 ID。詳細については、「PutBot」を参照してください。

型: 文字列

エラー

BadRequestException

リクエストが適切にフォーマットされていません。例えば、値が無効であったり、必須項目が設定されていない場合です。フィールドの値を確認して、再度お試しください。

HTTP ステータスコード：400

ConflictException

リクエストの処理中に競合が発生しました。リクエストを再試行してください。

HTTP ステータスコード: 409

InternalFailureException

Amazon Lex 内部エラーが発生しました。リクエストを再試行してください。

HTTP ステータスコード：500

LimitExceededException

リクエストが制限を超えました。リクエストを再試行してください。

HTTP ステータスコード: 429

PreconditionFailedException

変更しようとしているリソースのチェックサムがリクエストのチェックサムと一致しません。リソースのチェックサムを確認して、もう一度お試しください。

HTTP ステータスコード: 412

その他の参照資料

この API を言語固有の AWS SDK で使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go バージョン 2 用 SDK
- AWS Java V2 用 SDK

- AWS V3 用 JavaScript SDK
- AWS PHP V3 用SDK
- AWS Python 用 SDK
- AWS ルビー V3 用 SDK

PutBotAlias

サービス: Amazon Lex Model Building Service

指定されたバージョンのボットのエイリアスを作成したり、指定されたボットのエイリアスを置き換えたりします。エイリアスが示すボットのバージョンを変更するには、エイリアスを置き換えます。エイリアスの詳細については、「バージョニングとエイリアス」を参照してください。

このオペレーションには、lex:PutBotAlias アクションに対する許可が必要です。

リクエストの構文

```
PUT /bots/botName/aliases/name HTTP/1.1
Content-type: application/json

{
   "botVersion": "string",
   "checksum": "string",
   "conversationLogs": {
      "iamRoleArn": "string",
      "logSettings": [
         {
            "destination": "string",
            "kmsKeyArn": "string",
            "logType": "string",
            "resourceArn": "string"
         }
      ]
   },
   "description": "string",
   "tags": [
      {
         "key": "string",
         "value": "string"
      }
   ]
}
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

botName

ボットの名前。

長さの制限: 最小長は 2 です。最大長は 50 です。

Pattern: ^([A-Za-z]_?)+$

必須：はい

name

エイリアスの名前。キースペース名では、大文字と小文字は区別されません。

長さの制限：最小長は 1 です。最大長は 100 です。

パターン: ^([A-Za-z]_?)+$

必須: はい

リクエストボディ

リクエストは以下の JSON 形式のデータを受け入れます。

botVersion

ボットのバージョン。

型: 文字列

長さの制限：最小長は 1 です。最大長は 64 文字です。

パターン: \$LATEST|[0-9]+

必須：はい

checksum

$LATEST バージョンの特定のリビジョンを識別します。

新しいボットエイリアスを作成する場合は、checksum フィールドを空白にします。チェックサムを指定した場合、BadRequestException の例外が発生します。

ボットエイリアスを更新する場合は、checksum フィールドに $LATEST バージョンの最新リビジョンのチェックサムを設定します。 checksum フィールドを指定しない場合や、チェックサ

ムが $LATEST バージョンと一致しない場合は、PreconditionFailedException 例外が発生します。

タイプ: 文字列

必須: いいえ

conversationLogs

エイリアスの会話ログの設定。

型: ConversationLogsRequest オブジェクト

必須: いいえ

description

エイリアスの説明。

型: 文字列

長さの制限: 最小長は 0 です。最大長は 200 です。

必須: いいえ

tags

ボットエイリアスに追加するタグのリスト。タグを追加できるのは、エイリアスを作成したときだけで、PutBotAlias オペレーションを使ってボットのエイリアスのタグを更新することはできません。タグを更新するには、TagResource のオペレーションを使用します。

型: Tag オブジェクトの配列

配列メンバー: 最小数は 0 項目です。最大数は 200 項目です。

必須: いいえ

レスポンスの構文

```
HTTP/1.1 200
Content-type: application/json

{
   "botName": "string",
   "botVersion": "string",
```

```
    "checksum": "string",
    "conversationLogs": {
       "iamRoleArn": "string",
       "logSettings": [
          {
             "destination": "string",
             "kmsKeyArn": "string",
             "logType": "string",
             "resourceArn": "string",
             "resourcePrefix": "string"
          }
       ]
    },
    "createdDate": number,
    "description": "string",
    "lastUpdatedDate": number,
    "name": "string",
    "tags": [
       {
          "key": "string",
          "value": "string"
       }
    ]
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

botName

エイリアスが指すボットの名前。

型: 文字列

長さの制限: 最小長は 2 です。最大長は 50 です。

Pattern: ^([A-Za-z]_?)+$

botVersion

エイリアスが指すボットのバージョン。

型: 文字列

長さの制限：最小長は 1 です。最大長は 64 文字です。

パターン: \$LATEST|[0-9]+

checksum

現在のバージョンのエイリアスのチェックサム。

型: 文字列

conversationLogs

Amazon Lex がエイリアスの会話ログをどのように使用するかを決定する設定。

型: ConversationLogsResponse オブジェクト

createdDate

ボットエイリアスが作成された日付。

型: タイムスタンプ

description

エイリアスの説明。

型: 文字列

長さの制限: 最小長は 0 です。最大長は 200 です。

lastUpdatedDate

ボットエイリアスが更新された日付。リソースを作成する場合、作成日と最終更新日は同じ日付になります。

型: タイムスタンプ

name

エイリアスの名前。

型: 文字列

長さの制限: 最小長は 1 です。最大長は 100 です。

パターン: ^([A-Za-z]_?)+$

tags

ボットに関連付けられているタグのリスト。

型: Tag オブジェクトの配列

配列メンバー: 最小数は 0 項目です。最大数は 200 項目です。

エラー

BadRequestException

リクエストが適切にフォーマットされていません。例えば、値が無効であったり、必須項目が設定されていない場合です。フィールドの値を確認して、再度お試しください。

HTTP ステータスコード：400

ConflictException

リクエストの処理中に競合が発生しました。リクエストを再試行してください。

HTTP ステータスコード: 409

InternalFailureException

Amazon Lex 内部エラーが発生しました。リクエストを再試行してください。

HTTP ステータスコード：500

LimitExceededException

リクエストが制限を超えました。リクエストを再試行してください。

HTTP ステータスコード: 429

PreconditionFailedException

変更しようとしているリソースのチェックサムがリクエストのチェックサムと一致しません。リソースのチェックサムを確認して、もう一度お試しください。

HTTP ステータスコード: 412

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go バージョン 2 用 SDK
- AWS Java V2 用 SDK
- AWS V3 用 JavaScript SDK
- AWS PHP V3 用SDK
- AWS Python 用 SDK
- AWS ルビー V3 用 SDK

PutIntent

サービス: Amazon Lex Model Building Service

インテントを作成するか、既存のインテントを置き換えます。

ユーザーとボットのインタラクションを定義するには、1 つ以上のインテントを使用します。例えば、ピザを注文するボットの場合、OrderPizza のインテントを作成します。

インテントを作成するか、既存のインテントを置き換えるには、次の項目を指定する必要があります。

- インテント名。例えば、OrderPizza。
- サンプル発話。例えば、「ピザを注文できますか」「ピザを注文したいです」
- 収集される情報です。ボットがユーザーから要求する情報のスロットタイプを指定します。日付や時刻などの標準的なスロットタイプと、ピザのサイズやクラストなどのカスタムスロットタイプを指定できます。
- インテントはどのように達成されるのか。Lambda 関数を提供するか、クライアントアプリケーションにインテント情報を返すようにインテントを構成します。Lambda 関数を使用した場合、すべてのインテント情報が利用可能になると、Amazon Lex は Lambda 関数を起動します。クライアントアプリケーションにインテント情報を返すようにインテントを設定した場合。

リクエストには、以下のような他のオプション情報を指定することができます。

- ユーザーにインテントを確認してもらうための確認プロンプト。例えば、「ピザを注文しましょうか」
- インテントが達成された後にユーザーに送信する結果ステートメント。例えば、「あなたのピザを注文しました」
- ユーザーに追加のアクティビティを要求するフォローアッププロンプトです。例えば、「ピザと一緒にドリンクを注文しますか」と尋ねます

既存のインテント名を指定してインテントを更新した場合、Amazon Lex は $LATEST バージョンのインテントの値をリクエストの値に置き換えます。Amazon Lex では、リクエストで入力されていないフィールドは削除されます。必須フィールドを指定しない場合、Amazon Lex は例外をスローします。$LATEST バージョンのインテントを更新すると、$LATEST バージョンのインテントを使用しているボットの status フィールドが NOT_BUILT に設定されます。

詳細については、「Amazon Lex: 仕組み」を参照してください。

このオペレーションには、lex:PutIntent アクションに対する許可が必要です。

リクエストの構文

```
PUT /intents/name/versions/$LATEST HTTP/1.1
Content-type: application/json

{
   "checksum": "string",
   "conclusionStatement": {
      "messages": [
         {
            "content": "string",
            "contentType": "string",
            "groupNumber": number
         }
      ],
      "responseCard": "string"
   },
   "confirmationPrompt": {
      "maxAttempts": number,
      "messages": [
         {
            "content": "string",
            "contentType": "string",
            "groupNumber": number
         }
      ],
      "responseCard": "string"
   },
   "createVersion": boolean,
   "description": "string",
   "dialogCodeHook": {
      "messageVersion": "string",
      "uri": "string"
   },
   "followUpPrompt": {
      "prompt": {
         "maxAttempts": number,
         "messages": [
            {
               "content": "string",
               "contentType": "string",
               "groupNumber": number
```

```
            }
         ],
         "responseCard": "string"
      },
      "rejectionStatement": {
         "messages": [
            {
               "content": "string",
               "contentType": "string",
               "groupNumber": number
            }
         ],
         "responseCard": "string"
      }
   },
   "fulfillmentActivity": {
      "codeHook": {
         "messageVersion": "string",
         "uri": "string"
      },
      "type": "string"
   },
   "inputContexts": [
      {
         "name": "string"
      }
   ],
   "kendraConfiguration": {
      "kendraIndex": "string",
      "queryFilterString": "string",
      "role": "string"
   },
   "outputContexts": [
      {
         "name": "string",
         "timeToLiveInSeconds": number,
         "turnsToLive": number
      }
   ],
   "parentIntentSignature": "string",
   "rejectionStatement": {
      "messages": [
         {
            "content": "string",
```

```
            "contentType": "string",
            "groupNumber": number
         }
      ],
      "responseCard": "string"
   },
   "sampleUtterances": [ "string" ],
   "slots": [
      {
         "defaultValueSpec": {
            "defaultValueList": [
               {
                  "defaultValue": "string"
               }
            ]
         },
         "description": "string",
         "name": "string",
         "obfuscationSetting": "string",
         "priority": number,
         "responseCard": "string",
         "sampleUtterances": [ "string" ],
         "slotConstraint": "string",
         "slotType": "string",
         "slotTypeVersion": "string",
         "valueElicitationPrompt": {
            "maxAttempts": number,
            "messages": [
               {
                  "content": "string",
                  "contentType": "string",
                  "groupNumber": number
               }
            ],
            "responseCard": "string"
         }
      }
   ]
}
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

name

インテントの名前。キースペース名では、大文字と小文字は区別されません。

組み込みインテントネームと一致しないか、ビルトインのインテントネームに「AMAZON」が含まれています。削除済み。例えば、AMAZON.HelpIntent という組み込みインテントがあるので、HelpIntent というカスタムインテントを作ることはできません。

組み込みインテントの一覧については、「Alexa Skills Kit」の「Standard Built-in Intents」を参照してください。

長さの制限：最小長は 1 です。最大長は 100 です。

パターン: ^([A-Za-z]_?)+$

必須: はい

リクエストボディ

リクエストは以下の JSON 形式のデータを受け入れます。

checksum

$LATEST バージョンの特定のリビジョンを識別します。

新しいインテントを作成する場合は、checksum フィールドを空白にします。チェックサムを指定した場合、BadRequestException の例外が発生します。

インテントを更新する場合は、checksum フィールドに $LATEST バージョンの最新リビジョンのチェックサムを設定します。 checksum フィールドを指定しない場合や、チェックサムが $LATEST バージョンと一致しない場合は、PreconditionFailedException 例外が発生します。

タイプ: 文字列

必須: いいえ

conclusionStatement

インテントが Lambda 関数によって正常に達成された後に、Amazon Lex がユーザーに伝えるステートメントです。

この要素は、fulfillmentActivity で Lambda 関数を提供する場合にのみ関係します。インテントをクライアントアプリケーションに返す場合は、この要素を指定することはできません。

> **Note**
>
> followUpPrompt および conclusionStatement は相互に排他的です。指定できる値は 1 つだけです。

型: Statement オブジェクト

必須: いいえ

confirmationPrompt

ユーザーにインテントの確認を求めます。この質問には「はい」または「いいえ」の答えがあるはずです。

Amazon Lex は、このプロンプトを使用して、インテントがフルフィルメントの準備ができていることをユーザーが確認できるようにします。例えば、OrderPizza のインテントでは、注文する前に正しいかどうかを確認したいと思うかもしれません。ユーザーの質問に単純に応答するインテントなど、情報提供前のユーザー確認が必要のない場合もあります。

> **Note**
>
> rejectionStatement と confirmationPrompt の両方を提供するか、両方とも提供しません。

型: Prompt オブジェクト

必須: いいえ

createVersion

true に設定すると、新しく採番されたバージョンのインテントが作成されます。これは、CreateIntentVersion のオペレーションを呼び出すのと同じです。createVersion を指定していない場合、デフォルトは false です。

型: ブール値

必須: いいえ

description

インテントの説明。

型: 文字列

長さの制限: 最小長は 0 です。最大長は 200 です。

必須: いいえ

dialogCodeHook

ユーザー入力ごとに呼び出す Lambda 関数を指定します。この Lambda 関数を呼び出して、ユーザーのインタラクションをパーソナライズすることができます。

例えば、ボットがユーザーを John と判断したとします。Lambda 関数は、バックエンドのデータベースから John の情報を取得し、いくつかの値を事前に入力しているかもしれません。例えば、John がグルテンアレルギーであることがわかった場合、対応するインテントスロット `GlutenIntolerant` を true に設定することができます。John の電話番号を見つけて、対応するセッション属性を設定することもあります。

型: CodeHook オブジェクト

必須: いいえ

followUpPrompt

これが定義されている場合、Amazon Lex はこのプロンプトを使用して、インテントが達成された後に追加のユーザーアクティビティを要求します。例えば、`OrderPizza` のインテントが達成された後、ユーザーに飲み物の注文を促すことができます。

Amazon Lex が実行するアクションは、ユーザーのレスポンスによって以下のように異なります。

- ユーザーが 「はい」と答えると、そのボット用に設定された明確なプロンプトが表示されます。
- ユーザーが 「はい」と答えた後、インテントをトリガーする発話を続けると、そのインテントの会話が始まります。
- ユーザーが「いいえ」と答えた場合、フォローアッププロンプトに設定されている拒否ステートメントで応答します。
- 発話が認識されない場合は、フォローアッププロンプトが再び繰り返されます。

`followUpPrompt` フィールドと `conclusionStatement` フィールドは相互に独立しています。指定できる値は 1 つだけです。

型: FollowUpPrompt オブジェクト

必須: いいえ

fulfillmentActivity

必須。インテントがどのように達成されるのかを説明します。例えば、ユーザーがピザを注文するためのすべての情報を提供した後、fulfillmentActivity はボットが地元のピザ店に注文する方法を定義します。

Amazon Lex がすべてのインテント情報をクライアントアプリケーションに返すように設定したり、インテントを処理する Lambda 関数を呼び出すように指示したりすることができます (例えば、ピザ屋に注文を出すなど)。

型: FulfillmentActivity オブジェクト

必須: いいえ

inputContexts

Amazon Lex がユーザーとの会話の中でインテントを選択するためのアクティブなコンテキストをリストアップした InputContext オブジェクトの配列です。

型: InputContext オブジェクトの配列

配列メンバー：最小数は 0 項目です。最大数は 5 項目です。

必須: いいえ

kendraConfiguration

AMAZON.KendraSearchIntent インテントを使って Amazon Kendra のインデックスに接続するために必要な設定情報です。詳細については、「AMAZON」を参照してください。KendraSearchIntent。

タイプ：KendraConfiguration オブジェクト

必須: いいえ

outputContexts

OutputContext オブジェクトの配列で、インテントが達成されたときにアクティブになるコンテキストを列挙します。

型: OutputContext オブジェクトの配列

の配列メンバー: 最小数は 0 項目です。最大数は 10 項目です。

必須: いいえ

parentIntentSignature

このインテントの基礎となる組み込みインテントの一意の識別子です。インテントの署名を見つけるには、「Alexa Skills Kit」の「Standard Built-in Intents」(標準の組み込みインテント) を参照してください。

タイプ: 文字列

必須: いいえ

rejectionStatement

ユーザーが `confirmationPrompt` で定義された質問に「いいえ」と答えた場合、Amazon Lex は、インテントがキャンセルされたことをこのステートメントの応答で確認します。

> **Note**
>
> `rejectionStatement` と `confirmationPrompt` の両方を提供するか、両方とも提供しません。

型: Statement オブジェクト

必須: いいえ

sampleUtterances

インテントを知らせるためにユーザーが言うかもしれない発話 (文字列) の配列。たとえば、「{PizzaSize} ピザが欲しい」、「{数量} {PizzaSize} ピザを注文する」などです。

各発話では、スロット名が中括弧で囲まれます。

型: 文字列の配列

配列メンバー：最小数は 0 項目です。最大数は 1500 項目です。

長さの制限：最小長は 1 です。最大長は 200 です。

必須: いいえ

slots

インテントスロットの配列。実行時に、Amazon Lex は、スロットで定義されているプロンプトを使用して、必要なスロット値をユーザーから引き出します。詳細については、「Amazon Lex: 仕組み」を参照してください。

型: Slot オブジェクトの配列

配列メンバー: 最小数は 0 項目です。最大数は 100 項目です。

必須: いいえ

レスポンスの構文

```
HTTP/1.1 200
Content-type: application/json

{
   "checksum": "string",
   "conclusionStatement": {
      "messages": [
         {
            "content": "string",
            "contentType": "string",
            "groupNumber": number
         }
      ],
      "responseCard": "string"
   },
   "confirmationPrompt": {
      "maxAttempts": number,
      "messages": [
         {
            "content": "string",
            "contentType": "string",
            "groupNumber": number
         }
      ],
      "responseCard": "string"
   },
   "createdDate": number,
   "createVersion": boolean,
   "description": "string",
   "dialogCodeHook": {
      "messageVersion": "string",
      "uri": "string"
   },
   "followUpPrompt": {
      "prompt": {
```

```
        "maxAttempts": number,
        "messages": [
           {
              "content": "string",
              "contentType": "string",
              "groupNumber": number
           }
        ],
        "responseCard": "string"
     },
     "rejectionStatement": {
        "messages": [
           {
              "content": "string",
              "contentType": "string",
              "groupNumber": number
           }
        ],
        "responseCard": "string"
     }
  },
  "fulfillmentActivity": {
     "codeHook": {
        "messageVersion": "string",
        "uri": "string"
     },
     "type": "string"
  },
  "inputContexts": [
     {
        "name": "string"
     }
  ],
  "kendraConfiguration": {
     "kendraIndex": "string",
     "queryFilterString": "string",
     "role": "string"
  },
  "lastUpdatedDate": number,
  "name": "string",
  "outputContexts": [
     {
        "name": "string",
        "timeToLiveInSeconds": number,
```

```
        "turnsToLive": number
      }
   ],
   "parentIntentSignature": "string",
   "rejectionStatement": {
      "messages": [
         {
            "content": "string",
            "contentType": "string",
            "groupNumber": number
         }
      ],
      "responseCard": "string"
   },
   "sampleUtterances": [ "string" ],
   "slots": [
      {
         "defaultValueSpec": {
            "defaultValueList": [
               {
                  "defaultValue": "string"
               }
            ]
         },
         "description": "string",
         "name": "string",
         "obfuscationSetting": "string",
         "priority": number,
         "responseCard": "string",
         "sampleUtterances": [ "string" ],
         "slotConstraint": "string",
         "slotType": "string",
         "slotTypeVersion": "string",
         "valueElicitationPrompt": {
            "maxAttempts": number,
            "messages": [
               {
                  "content": "string",
                  "contentType": "string",
                  "groupNumber": number
               }
            ],
            "responseCard": "string"
         }
```

```
        }
    ],
    "version": "string"
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

checksum

作成または更新されたインテントの $LATEST バージョンのチェックサム。

型: 文字列

conclusionStatement

fulfillmentActivity のインテントで指定された Lambda 関数がインテントを達成すると、Amazon Lex はこのステートメントをユーザーに伝えます。

型: Statement オブジェクト

confirmationPrompt

これがインテントに定義されている場合、Amazon Lex インテントを達成する前に、ユーザーにインテントの確認を促します。

型: Prompt オブジェクト

createdDate

インテントが作成された日付。

型: タイムスタンプ

createVersion

新しいバージョンのインテントが作成された場合の True。リクエストで createVersion フィールドが指定されていない場合は、レスポンスで createVersion フィールドが false に設定されます。

型: ブール値

description

インテントの説明。

型: 文字列

長さの制限: 最小長は 0 です。最大長は 200 です。

dialogCodeHook

これがインテントに定義されている場合、Amazon Lex は各ユーザー入力に対してこの Lambda 関数を呼び出します。

型: CodeHook オブジェクト

followUpPrompt

これがインテントに定義されている場合、Amazon Lex はこのプロンプトを使用して、インテントが達成された後に追加のユーザーアクティビティを要求します。

型: FollowUpPrompt オブジェクト

fulfillmentActivity

これがインテントに定義されている場合、Amazon Lex はこの Lambda 関数を呼び出し、インテントを達成するために、インテントをユーザーがインテントに要求します。

型: FulfillmentActivity オブジェクト

inputContexts

Amazon Lex がユーザーとの会話の中でインテントを選択するためのアクティブなコンテキストをリストアップした `InputContext` オブジェクトの配列です。

型: InputContext オブジェクトの配列

配列メンバー：最小数は 0 項目です。最大数は 5 項目です。

kendraConfiguration

Amazon Kendra インデックスに接続し、`AMAZON.KendraSearchIntent` インテントを使用するために必要な設定情報 (項目がある場合)。

型: KendraConfiguration オブジェクト

lastUpdatedDate

インテントが更新された日付。リソースを作成する場合、作成日と最終更新日は同じ日付になります。

型: タイムスタンプ

name

インテントの名前。

型: 文字列

長さの制限: 最小長は 1 です。最大長は 100 です。

パターン: ^([A-Za-z]_?)+$

outputContexts

OutputContext オブジェクトの配列で、インテントが達成されたときにアクティブになるコンテキストを列挙します。

型: OutputContext オブジェクトの配列

の配列メンバー: 最小数は 0 項目です。最大数は 10 項目です。

parentIntentSignature

このインテントの基礎となる組み込みインテントの一意の識別子です。

型: 文字列

rejectionStatement

confirmationPrompt で定義された質問にユーザーが「いいえ」と答えた場合、Amazon Lex はこのステートメントで、インテントがキャンセルされたことを確認します。

型: Statement オブジェクト

sampleUtterances

インテント用に構成されたサンプル発話の配列。

型: 文字列の配列

配列メンバー : 最小数は 0 項目です。最大数は 1500 項目です。

長さの制限：最小長は 1 です。最大長は 200 です。

slots

インテント用に設定されたインテントスロットの配列。

型: Slot オブジェクトの配列

配列メンバー: 最小数は 0 項目です。最大数は 100 項目です。

version

インテントのバージョン。新しいインテントの場合、バージョンは常に $LATEST です。

型: 文字列

長さの制限：最小長は 1 です。最大長は 64 文字です。

パターン：\$LATEST|[0-9]+

エラー

BadRequestException

リクエストが適切にフォーマットされていません。例えば、値が無効であったり、必須項目が設定されていない場合です。フィールドの値を確認して、再度お試しください。

HTTP ステータスコード：400

ConflictException

リクエストの処理中に競合が発生しました。リクエストを再試行してください。

HTTP ステータスコード: 409

InternalFailureException

Amazon Lex 内部エラーが発生しました。リクエストを再試行してください。

HTTP ステータスコード：500

LimitExceededException

リクエストが制限を超えました。リクエストを再試行してください。

HTTP ステータスコード: 429

PreconditionFailedException

変更しようとしているリソースのチェックサムがリクエストのチェックサムと一致しません。リソースのチェックサムを確認して、もう一度お試しください。

HTTP ステータスコード: 412

その他の参照資料

この API を言語固有の AWS SDK で使用する方法について詳しくは、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go バージョン 2 用 SDK
- AWS Java V2 用 SDK
- AWS V3 用 JavaScript SDK
- AWS PHP V3 用SDK
- AWS Python 用 SDK
- AWS ルビー V3 用 SDK

PutSlotType

サービス: Amazon Lex Model Building Service

カスタムスロットタイプを作成するか、既存のカスタムスロットタイプを置き換えます。

カスタムスロットタイプを作成するには、スロットタイプの名前と、このタイプのスロットが引き受けることができる値の列挙値のセットを指定します。詳細については、「Amazon Lex: 仕組み」を参照してください。

既存のスロットタイプの名前を指定した場合、リクエストのフィールドはそのスロットタイプの $LATEST バージョンの既存の値を置き換えます。Amazon Lex は、リクエストで指定していないフィールドを削除します。必須フィールドに値を指定しない場合、Amazon Lex は例外をスローします。$LATEST バージョンのスロットタイプを更新すると、ボットがそのスロットタイプを含む $LATEST バージョンインテントを使用している場合、ボットの `status` フィールドが `NOT_BUILT` に設定されます。

このオペレーションには、`lex:PutSlotType` アクションに対する許可が必要です。

リクエストの構文

```
PUT /slottypes/name/versions/$LATEST HTTP/1.1
Content-type: application/json

{
   "checksum": "string",
   "createVersion": boolean,
   "description": "string",
   "enumerationValues": [
      {
         "synonyms": [ "string" ],
         "value": "string"
      }
   ],
   "parentSlotTypeSignature": "string",
   "slotTypeConfigurations": [
      {
         "regexConfiguration": {
            "pattern": "string"
         }
      }
   ],
   "valueSelectionStrategy": "string"
```

```
}
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

name

スロットタイプの名前。キースペース名では、大文字と小文字は区別されません。

組み込みインテントネームと一致しないか、組み込みのインテントネームに「AMAZON」が含まれています。削除済み。例えば、AMAZON.DATE というスロットタイプが組み込まれているので、DATE というカスタムスロットタイプを作ることはできません。

組み込みスロットタイプの一覧については、「Alexa Skills Kit」の「Slot Type Reference」(スロットタイプリファレンス) を参照してください。

長さの制限：最小長は 1 です。最大長は 100 です。

パターン: ^([A-Za-z]_?)+$

必須: はい

リクエストボディ

リクエストは以下の JSON 形式のデータを受け入れます。

checksum

$LATEST バージョンの特定のリビジョンを識別します。

新しいスロットタイプを作成する場合は、checksum フィールドを空白にします。チェックサムを指定した場合、BadRequestException の例外が発生します。

スロットタイプを更新する場合は、checksum フィールドに $LATEST バージョンの最新リビジョンのチェックサムを設定します。 checksum フィールドを指定しない場合や、チェックサムが $LATEST バージョンと一致しない場合は、PreconditionFailedException 例外が発生します。

タイプ: 文字列

必須: いいえ

createVersion

true に設定すると、新しく採番されたバージョンのスロットタイプが作成されます。これは、CreateSlotTypeVersion のオペレーションを呼び出すのと同じです。createVersion を指定していない場合、デフォルトは false です。

型: ブール値

必須: いいえ

description

スロットタイプの説明。

型: 文字列

長さの制限: 最小長は 0 です。最大長は 200 です。

必須: いいえ

enumerationValues

スロットタイプが取得できる値を定義する EnumerationValue オブジェクトのリスト。各値は、機械学習モデルがスロットを解決する値を学習するための追加の値である synonyms のリストを持つことができます。

正規表現スロットタイプは、列挙値を必要としません。他のすべてのスロットタイプには、列挙値のリストが必要です。

Amazon Lexは、スロットの値を解決する際に、最大 5 つの可能な値を含む解決リストを生成します。Lambda 関数を使用している場合は、この解決リストが関数に渡されます。Lambda 関数を使用していない場合は、ユーザーが入力した値を返すか、解決リストの最初の値をスロット値として返すかを選択できます。valueSelectionStrategy フィールドには、使用するオプションが表示されます。

型: EnumerationValue オブジェクトの配列

配列メンバー : 最小数は 0 項目です。最大数は 10000 項目です。

必須: いいえ

parentSlotTypeSignature

スロットタイプの親として使用される組み込みスロットタイプです。親スロットタイプを定義すると、新しいスロットタイプには親と同じ設定が適応されます。

AMAZON.AlphaNumeric のみサポートされています。

型: 文字列

長さの制限: 最小長は 1 です。最大長は 100 です。

Pattern: ^((AMAZON\.)_?|[A-Za-z]_?)+

必須: いいえ

slotTypeConfigurations

親組み込みスロットタイプを拡張する構成情報。構成は、親スロットタイプの設定に追加されます。

型: SlotTypeConfiguration オブジェクトの配列

の配列メンバー: 最小数は 0 項目です。最大数は 10 項目です。

必須: いいえ

valueSelectionStrategy

Amazon Lex がスロットタイプの値を返す際に使用するスロット解決方法を決定します。フィールドの値は次のいずれかになります。

- ORIGINAL_VALUE - ユーザー値がスロット値と似ている場合、ユーザーが入力した値を返します。
- TOP_RESOLUTION - スロットの解像度リストがある場合、解像度リストの最初の値をスロットタイプの値として返します。解像度リストがない場合、null が返されます。

valueSelectionStrategy を指定しなかった場合、デフォルトは ORIGINAL_VALUE です。

型: 文字列

有効な値：ORIGINAL_VALUE | TOP_RESOLUTION

必須：いいえ

レスポンスの構文

```
HTTP/1.1 200
```

```
Content-type: application/json

{
   "checksum": "string",
   "createdDate": number,
   "createVersion": boolean,
   "description": "string",
   "enumerationValues": [
      {
         "synonyms": [ "string" ],
         "value": "string"
      }
   ],
   "lastUpdatedDate": number,
   "name": "string",
   "parentSlotTypeSignature": "string",
   "slotTypeConfigurations": [
      {
         "regexConfiguration": {
            "pattern": "string"
         }
      }
   ],
   "valueSelectionStrategy": "string",
   "version": "string"
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

checksum

スロットタイプの $LATEST バージョンのチェックサム。

型: 文字列

createdDate

スロットタイプが作成された日付。

型: タイムスタンプ

createVersion

新しいバージョンのスロットタイプが作成された場合の True。リクエストで createVersion フィールドが指定されていない場合は、レスポンスで createVersion フィールドが false に設定されます。

型: ブール値

description

スロットタイプの説明。

型: 文字列

長さの制限: 最小長は 0 です。最大長は 200 です。

enumerationValues

スロットタイプが取得できる値を定義する EnumerationValue オブジェクトのリスト。

型: EnumerationValue オブジェクトの配列

配列メンバー : 最小数は 0 項目です。最大数は 10000 項目です。

lastUpdatedDate

スロットタイプが更新された日付。スロットタイプを作成する場合、作成日と最終更新日は同じです。

型: タイムスタンプ

name

スロットタイプの名前。

型: 文字列

長さの制限: 最小長は 1 です。最大長は 100 です。

パターン: ^([A-Za-z]_?)+$

parentSlotTypeSignature

スロットタイプの親として使用される組み込みスロットタイプです。

型: 文字列

長さの制限: 最小長は 1 です。最大長は 100 です。

パターン: ^((AMAZON\.)_?|[A-Za-z]_?)+

slotTypeConfigurations

親組み込みスロットタイプを拡張する構成情報。

型: SlotTypeConfiguration オブジェクトの配列

の配列メンバー: 最小数は 0 項目です。最大数は 10 項目です。

valueSelectionStrategy

Amazon Lex がスロットの値を決定するために使用するスロット解決ストラテジー。詳細については、「PutSlotType」を参照してください。

型: 文字列

有効な値 : ORIGINAL_VALUE | TOP_RESOLUTION

version

スロットタイプのバージョン。新しいスロットタイプの場合、バージョンは常に $LATEST となります。

型: 文字列

長さの制限 : 最小長は 1 です。最大長は 64 文字です。

パターン : \$LATEST|[0-9]+

エラー

BadRequestException

リクエストが適切にフォーマットされていません。例えば、値が無効であったり、必須項目が設定されていない場合です。フィールドの値を確認して、再度お試しください。

HTTP ステータスコード : 400

ConflictException

リクエストの処理中に競合が発生しました。リクエストを再試行してください。

HTTP ステータスコード: 409

InternalFailureException

Amazon Lex 内部エラーが発生しました。リクエストを再試行してください。

HTTP ステータスコード : 500

LimitExceededException

リクエストが制限を超えました。リクエストを再試行してください。

HTTP ステータスコード: 429

PreconditionFailedException

変更しようとしているリソースのチェックサムがリクエストのチェックサムと一致しません。リソースのチェックサムを確認して、もう一度お試しください。

HTTP ステータスコード: 412

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go バージョン 2 用 SDK
- AWS Java V2 用 SDK
- AWS V3 用 JavaScript SDK
- AWS PHP V3 用SDK
- AWS Python 用 SDK
- AWS ルビー V3 用 SDK

StartImport

サービス: Amazon Lex Model Building Service

リソースを Amazon Lex にインポートするジョブを開始します。

リクエストの構文

```
POST /imports/ HTTP/1.1
Content-type: application/json

{
   "mergeStrategy": "string",
   "payload": blob,
   "resourceType": "string",
   "tags": [
      {
         "key": "string",
         "value": "string"
      }
   ]
}
```

URI リクエストパラメータ

リクエストでは URI パラメータを使用しません。

リクエストボディ

リクエストは以下の JSON 形式のデータを受け入れます。

mergeStrategy

同名の既存のリソースがある場合に、StartImport オペレーションが取るべきアクションを指定します。

- FAIL_ON_CONFLICT - インポートファイル内のリソースと既存のリソースが競合を起こした時点で、インポート作業を停止します。競合を起こしているリソース名は、GetImport オペレーションへのレスポンスの failureReason フィールドにあります。

  OVERWRITE_LATEST - 既存のリソースとの競合があっても、インポートオペレーションは続行されます。既存のリソースの $LASTEST バージョンは、インポートファイルのデータで上書きされます。

型: 文字列

有効な値 : OVERWRITE_LATEST | FAIL_ON_CONFLICT

必須: はい

payload

バイナリ形式の zip アーカイブ。アーカイブには、インポートするリソースを含む JSON ファイルが 1 つ含まれている必要があります。リソースは、resourceType フィールドで指定されたタイプと一致する必要があります。

型: Base64 でエンコードされたバイナリデータオブジェクト

必須: はい

resourceType

エクスポートするリソースの種類を指定します。また、各リソースは、依存しているリソースもエクスポートします。

- ボットは、依存性を持つインテントをエクスポートします。
- インテントは、依存するスロットタイプをエクスポートします。

型: 文字列

有効な値 : BOT | INTENT | SLOT_TYPE

必須: はい

tags

インポートされたボットに追加するタグのリスト。タグを追加できるのは、ボットをインポートするときだけです。インテントまたはスロットタイプにタグを追加することはできません。

型: Tag オブジェクトの配列

配列メンバー: 最小数は 0 項目です。最大数は 200 項目です。

必須: いいえ

レスポンスの構文

```
HTTP/1.1 201
Content-type: application/json
```

```
{
   "createdDate": number,
   "importId": "string",
   "importStatus": "string",
   "mergeStrategy": "string",
   "name": "string",
   "resourceType": "string",
   "tags": [
      {
         "key": "string",
         "value": "string"
      }
   ]
}
```

レスポンス要素

アクションが成功すると、HTTP 201 レスポンスが返されます。

サービスから以下のデータが JSON 形式で返されます。

createdDate

インポートジョブがリクエストされた日時のタイムスタンプ。

型: タイムスタンプ

importId

特定のインポートジョブの識別子。

型: 文字列

importStatus

インポートジョブのステータス。ステータスが FAILED の場合は、GetImport オペレーションで故障の原因を知ることができます。

型: 文字列

有効な値：IN_PROGRESS | COMPLETE | FAILED

mergeStrategy

マージ競合が発生した場合に実行するアクション。

型: 文字列

有効な値：OVERWRITE_LATEST | FAIL_ON_CONFLICT

name

インポートジョブに付けられた名前。

型: 文字列

長さの制限: 最小長は 1 です。最大長は 100 です。

パターン: [a-zA-Z_]+

resourceType

インポートするリソースのタイプ。

型: 文字列

有効な値：BOT | INTENT | SLOT_TYPE

tags

インポートされたボットに追加されたタグのリスト。

型: Tag オブジェクトの配列

配列メンバー: 最小数は 0 項目です。最大数は 200 項目です。

エラー

BadRequestException

リクエストが適切にフォーマットされていません。例えば、値が無効であったり、必須項目が設定されていない場合です。フィールドの値を確認して、再度お試しください。

HTTP ステータスコード：400

InternalFailureException

Amazon Lex 内部エラーが発生しました。リクエストを再試行してください。

HTTP ステータスコード：500

LimitExceededException

リクエストが制限を超えました。リクエストを再試行してください。

HTTP ステータスコード: 429

以下も参照してください。

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go バージョン 2 用 SDK
- AWS Java V2 用 SDK
- AWS V3 用 JavaScript SDK
- AWS PHP V3 用SDK
- AWS Python 用 SDK
- AWS ルビー V3 用 SDK

StartMigration

サービス: Amazon Lex Model Building Service

Amazon Lex V1 から Amazon Lex V2 へのボットの移行を開始します。Amazon Lex V2 の新機能を利用する場合は、ボットを移行します。

詳細については、「Amazon Lex developer guide」(Amazon Lex デベロッパーガイド) の「Migrating a bot」(ボットの移行) を参照してください。

リクエストの構文

```
POST /migrations HTTP/1.1
Content-type: application/json

{
   "migrationStrategy": "string",
   "v1BotName": "string",
   "v1BotVersion": "string",
   "v2BotName": "string",
   "v2BotRole": "string"
}
```

URI リクエストパラメータ

リクエストでは URI パラメータを使用しません。

リクエストボディ

リクエストは以下の JSON 形式のデータを受け入れます。

migrationStrategy

移行を実行するために使用された戦略。

- CREATE_NEW - 新しい Amazon Lex V2 ボットを作成し、Amazon Lex V1 ボットをこの新しいボットに移行します。
- UPDATE_EXISTING - 既存の Amazon Lex V2 ボットのメタデータおよび移行されるロケールを上書きします。Amazon Lex V2 ボットの他のロケールは変更されません。ロケールが存在しない場合は、Amazon Lex V2 ボットに新しいロケールが作成されます。

型: 文字列

有効な値：CREATE_NEW | UPDATE_EXISTING

必須: はい

v1BotName

Amazon Lex V2 に移行する Amazon Lex V1 ボットの名前。

型: 文字列

長さの制限: 最小長は 2 です。最大長は 50 です。

Pattern: ^([A-Za-z]_?)+$

必須：はい

v1BotVersion

Amazon Lex V2 に移行するボットのバージョン。$LATEST バージョンだけでなく、任意の番号のバージョンを移行することができます。

型: 文字列

長さの制限：最小長は 1 です。最大長は 64 文字です。

パターン: \$LATEST|[0-9]+

必須：はい

v2BotName

Amazon Lex V1 ボットの移行先となる Amazon Lex V2 ボットの名前。

- Amazon Lex V2 のボットが存在しない場合は、CREATE_NEW の移行戦略を使用する必要があります。
- Amazon Lex V2 ボットが存在する場合、UPDATE_EXISTING 移行戦略を使用して Amazon Lex V2 ボットの内容を変更する必要があります。

型: 文字列

長さの制限: 最小長は 1 です。最大長は 100 です。

パターン: ^([0-9a-zA-Z][_-]?)+$

必須：はい

v2BotRole

Amazon Lex が Amazon Lex V2 ボットの実行に使用する IAM ロール。

型: 文字列

長さの制限: 最小長は 20 です。最大長は 2,048 です。

Pattern: `^arn:[\w\-]+:iam::[\d]{12}:role/.+$`

必須：はい

レスポンスの構文

```
HTTP/1.1 202
Content-type: application/json

{
   "migrationId": "string",
   "migrationStrategy": "string",
   "migrationTimestamp": number,
   "v1BotLocale": "string",
   "v1BotName": "string",
   "v1BotVersion": "string",
   "v2BotId": "string",
   "v2BotRole": "string"
}
```

レスポンス要素

アクションが成功すると、HTTP 202 レスポンスが返されます。

サービスから以下のデータが JSON 形式で返されます。

migrationId

Amazon Lex が移行に割り当てた一意の識別子。

型: 文字列

長さの制限: 固定長は 10 です。

Pattern: `^[0-9a-zA-Z]+$`

migrationStrategy

移行を実行するために使用された戦略。

型: 文字列

有効な値：CREATE_NEW | UPDATE_EXISTING

migrationTimestamp

移行が開始された日時。

型: タイムスタンプ

v1BotLocale

Amazon Lex V1 ボットに使用されるロケール。

型: 文字列

有効な値：de-DE | en-AU | en-GB | en-IN | en-US | es-419 | es-ES | es-US | fr-FR | fr-CA | it-IT | ja-JP | ko-KR

v1BotName

Amazon Lex V2 に移行する Amazon Lex V1 ボットの名前。

型: 文字列

長さの制限: 最小長は 2 です。最大長は 50 です。

Pattern: ^([A-Za-z]_?)+$

v1BotVersion

Amazon Lex V2 に移行するボットのバージョン。

型: 文字列

長さの制限：最小長は 1 です。最大長は 64 文字です。

パターン: \$LATEST|[0-9]+

v2BotId

Amazon Lex V2 ボットの一意の識別子。

型: 文字列

長さの制限: 固定長は 10 です。

Pattern: ^[0-9a-zA-Z]+$

v2BotRole

Amazon Lex が Amazon Lex V2 ボットの実行に使用する IAM ロール。

型: 文字列

長さの制限: 最小長は 20 です。最大長は 2,048 です。

Pattern: ^arn:[\w\-]+:iam::[\d]{12}:role/.+$

エラー

AccessDeniedException

お客様の IAM ユーザーまたはロールには、ボットの移行に必要な Amazon Lex V2 API を呼び出す許可がありません。

HTTP ステータスコード: 403

BadRequestException

リクエストが適切にフォーマットされていません。例えば、値が無効であったり、必須項目が設定されていない場合です。フィールドの値を確認して、再度お試しください。

HTTP ステータスコード : 400

InternalFailureException

Amazon Lex 内部エラーが発生しました。リクエストを再試行してください。

HTTP ステータスコード : 500

LimitExceededException

リクエストが制限を超えました。リクエストを再試行してください。

HTTP ステータスコード: 429

NotFoundException

リクエストで指定されたリソースは見つかりませんでした。リソースを確認して、もう一度お試しください。

HTTP ステータスコード: 404

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go バージョン 2 用 SDK
- AWS Java V2 用 SDK
- AWS V3 用 JavaScript SDK
- AWS PHP V3 用SDK
- AWS Python 用 SDK
- AWS ルビー V3 用 SDK

TagResource

サービス: Amazon Lex Model Building Service

指定されたタグを、指定された リソースに追加します。タグキーがすでに存在する場合、既存のタグは新しいタグに置き換えられます。

リクエストの構文

```
POST /tags/resourceArn HTTP/1.1
Content-type: application/json

{
   "tags": [
      {
         "key": "string",
         "value": "string"
      }
   ]
}
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

resourceArn

タグを付けるボット、ボットエイリアス、ボットチャンネルの Amazon リソースネーム (ARN)。

長さの制限：最小長は 1 です。最大長は 1011 です。

必須: はい

リクエストボディ

リクエストは以下の JSON 形式のデータを受け入れます。

tags

リソースに追加するタグキーのリスト。タグキーがすでに存在する場合、既存のタグは新しいタグに置き換えられます。

型: Tag オブジェクトの配列

配列メンバー: 最小数は 0 項目です。最大数は 200 項目です。

必須: はい

レスポンスの構文

```
HTTP/1.1 204
```

レスポンス要素

アクションが成功した場合、サービスは空の HTTP 本文を持つ HTTP 204 レスポンスを返します。

エラー

BadRequestException

リクエストが適切にフォーマットされていません。例えば、値が無効であったり、必須項目が設定されていない場合です。フィールドの値を確認して、再度お試しください。

HTTP ステータスコード：400

ConflictException

リクエストの処理中に競合が発生しました。リクエストを再試行してください。

HTTP ステータスコード: 409

InternalFailureException

Amazon Lex 内部エラーが発生しました。リクエストを再試行してください。

HTTP ステータスコード：500

LimitExceededException

リクエストが制限を超えました。リクエストを再試行してください。

HTTP ステータスコード: 429

NotFoundException

リクエストで指定されたリソースは見つかりませんでした。リソースを確認して、もう一度お試しください。

HTTP ステータスコード: 404

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go バージョン 2 用 SDK
- AWS Java V2 用 SDK
- AWS V3 用 JavaScript SDK
- AWS PHP V3 用SDK
- AWS Python 用 SDK
- AWS ルビー V3 用 SDK

UntagResource

サービス: Amazon Lex Model Building Service

ボット、ボットエイリアス、ボットチャンネルからタグを削除します。

リクエストの構文

```
DELETE /tags/resourceArn?tagKeys=tagKeys HTTP/1.1
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

resourceArn

タグを削除するリソースの Amazon リソースネーム (ARN)。

長さの制限：最小長は 1 です。最大長は 1011 です。

必須: はい

tagKeys

リソースから削除するタグキーのリスト。タグキーがリソースに存在しない場合、タグキーは無視されます。

配列メンバー：最小数は 0 項目です。最大数は 200 項目です。

長さの制限：最小長は 1 です。最大長は 128 です。

必須: はい

リクエストボディ

リクエストにリクエスト本文がありません。

レスポンスの構文

```
HTTP/1.1 204
```

レスポンス要素

アクションが成功した場合、サービスは空の HTTP 本文を持つ HTTP 204 レスポンスを返します。

エラー

BadRequestException

リクエストが適切にフォーマットされていません。例えば、値が無効であったり、必須項目が設定されていない場合です。フィールドの値を確認して、再度お試しください。

HTTP ステータスコード：400

ConflictException

リクエストの処理中に競合が発生しました。リクエストを再試行してください。

HTTP ステータスコード: 409

InternalFailureException

Amazon Lex 内部エラーが発生しました。リクエストを再試行してください。

HTTP ステータスコード：500

LimitExceededException

リクエストが制限を超えました。リクエストを再試行してください。

HTTP ステータスコード: 429

NotFoundException

リクエストで指定されたリソースは見つかりませんでした。リソースを確認して、もう一度お試しください。

HTTP ステータスコード: 404

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go バージョン 2 用 SDK
- AWS Java V2 用 SDK

- AWS V3 用 JavaScript SDK
- AWS PHP V3 用SDK
- AWS Python 用 SDK
- AWS ルビー V3 用 SDK

Amazon Lex Runtime Service

次のアクションは、Amazon Lex Runtime Service でサポートされています

- DeleteSession
- GetSession
- PostContent
- PostText
- PutSession

DeleteSession

サービス: Amazon Lex Runtime Service

指定されたボット、エイリアス、ユーザー ID のセッション情報を削除します。

リクエストの構文

```
DELETE /bot/botName/alias/botAlias/user/userId/session HTTP/1.1
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

botAlias

セッションデータを含むボットに使用されるエイリアス。

必須: はい

botName

セッションデータを含むボットの名前。

必須: はい

userId

セッションデータに関連付けられているユーザーの識別子。

長さの制限: 最小長は 2 です。最大長は 100 です。

パターン: `[0-9a-zA-Z._:-]+`

必須: はい

リクエストボディ

リクエストにリクエスト本文がありません。

レスポンスの構文

```
HTTP/1.1 200
Content-type: application/json

{
```

```
    "botAlias": "string",
    "botName": "string",
    "sessionId": "string",
    "userId": "string"
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

botAlias

セッションデータに関連付けられたボットに使用されているエイリアス。

型: 文字列

botName

セッションデータに関連付けられたボットの名前。

型: 文字列

sessionId

セッションの一意の識別子。

型: 文字列

userId

クライアントアプリケーションユーザーの ID。

型: 文字列

長さの制限: 最小長は 2 です。最大長は 100 です。

パターン：`[0-9a-zA-Z._:-]+`

エラー

BadRequestException

リクエストの検証に失敗したか、コンテキストに使用可能なメッセージがないか、ボットの構築が失敗もしくは進行中であるか、または構築されていない変更が含まれています。

HTTP ステータスコード : 400

ConflictException

2 つのクライアントが同じ AWS アカウント、Amazon Lex bot、ユーザー ID を使用しています。

HTTP ステータスコード: 409

InternalFailureException

内部サービスエラー。呼び出しを再試行します。

HTTP ステータスコード : 500

LimitExceededException

制限を超えました。

HTTP ステータスコード: 429

NotFoundException

参照するリソース (Amazon Lex bot やエイリアスなど) が見つかりません。

HTTP ステータスコード: 404

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go バージョン 2 用 SDK
- AWS Java V2 用 SDK
- AWS V3 用 JavaScript SDK
- AWS PHP V3 用SDK
- AWS Python 用 SDK
- AWS ルビー V3 用 SDK

GetSession

サービス: Amazon Lex Runtime Service

指定されたボット、エイリアス、ユーザー ID のセッション情報を返します。

リクエストの構文

```
GET /bot/botName/alias/botAlias/user/userId/session/?
checkpointLabelFilter=checkpointLabelFilter HTTP/1.1
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

botAlias

セッションデータを含むボットに使用されるエイリアス。

必須: はい

botName

セッションデータを含むボットの名前。

必須: はい

checkpointLabelFilter

`recentIntentSummaryView` 構造体で返されるインテントをフィルタリングするための文字列。

フィルターを指定すると、`checkpointLabel` フィールドにその文字列が設定されているインテントのみが返されます。

長さの制限：最小長は 1 です。最大長は 255 です。

パターン: `[a-zA-Z0-9-]+`

userId

クライアントアプリケーションユーザーの ID。Amazon Lex は、ユーザーとボットとの会話を識別するために使用します。

長さの制限: 最小長は 2 です。最大長は 100 です。

パターン: [0-9a-zA-Z._:-]+

必須: はい

リクエストボディ

リクエストにリクエスト本文がありません。

レスポンスの構文

```
HTTP/1.1 200
Content-type: application/json

{
   "activeContexts": [
      {
         "name": "string",
         "parameters": {
            "string" : "string"
         },
         "timeToLive": {
            "timeToLiveInSeconds": number,
            "turnsToLive": number
         }
      }
   ],
   "dialogAction": {
      "fulfillmentState": "string",
      "intentName": "string",
      "message": "string",
      "messageFormat": "string",
      "slots": {
         "string" : "string"
      },
      "slotToElicit": "string",
      "type": "string"
   },
   "recentIntentSummaryView": [
      {
         "checkpointLabel": "string",
         "confirmationStatus": "string",
         "dialogActionType": "string",
         "fulfillmentState": "string",
```

```
            "intentName": "string",
            "slots": {
               "string" : "string"
            },
            "slotToElicit": "string"
         }
      ],
      "sessionAttributes": {
         "string" : "string"
      },
      "sessionId": "string"
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

activeContexts

セッションのアクティブコンテキストのリスト。コンテキストは、インテントが達成されたとき、または PostContent、PostText、PutSession のオペレーションを呼び出したときに設定できます。

コンテキストを使用して、あるインテントをフォローできるインテントを制御したり、アプリケーションの動作を変更したりすることができます。

型: ActiveContext オブジェクトの配列

配列メンバー : 最小数は 0 項目です。最大数は 20 項目です。

dialogAction

ボットの現在の状態を示します。

型: DialogAction オブジェクト

recentIntentSummaryView

セッションで使用されるインテントに関する情報の配列。配列には最大 3 つのサマリーを含めることができます。セッションで 3 つ以上のインテントが使用されている場合、recentIntentSummaryView オペレーションには最後に使用された 3 つのインテントに関する情報が含まれています。

リクエストに checkpointLabelFilter パラメータを設定した場合、配列には、指定されたラベルを持つインテントのみが含まれます。

型: IntentSummary オブジェクトの配列

配列メンバー：最小数は 0 項目です。最大数は 3 項目です。

sessionAttributes

セッション固有のコンテキスト情報を表すキーバリューのペアのマップ。Amazon Lex とクライアントアプリケーションの間で渡されるアプリケーション情報を含みます。

型: 文字列間のマッピング

sessionId

セッションの一意の識別子。

型: 文字列

エラー

BadRequestException

リクエストの検証に失敗したか、コンテキストに使用可能なメッセージがないか、ボットの構築が失敗もしくは進行中であるか、または構築されていない変更が含まれています。

HTTP ステータスコード：400

InternalFailureException

内部サービスエラー。呼び出しを再試行します。

HTTP ステータスコード：500

LimitExceededException

制限を超えました。

HTTP ステータスコード: 429

NotFoundException

参照するリソース (Amazon Lex bot やエイリアスなど) が見つかりません。

HTTP ステータスコード: 404

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go バージョン 2 用 SDK
- AWS Java V2 用 SDK
- AWS V3 用 JavaScript SDK
- AWS PHP V3 用SDK
- AWS Python 用 SDK
- AWS ルビー V3 用 SDK

PostContent

サービス: Amazon Lex Runtime Service

ユーザー入力 (テキストまたは音声) を Amazon Lex に送信します。クライアントはこの API を使用して、実行時に Amazon Lex にテキストおよびオーディオリクエストを送信します。Amazon Lex は、ボット用に構築された機械学習モデルを使用してユーザー入力を解釈します。

`PostContent` オペレーションは 8kHz と 16kHz のオーディオ入力をサポートしています。8kHz オーディオを使用すると、電話オーディオアプリケーションで高い音声認識精度を実現できます。

レスポンスとして、Amazon Lex はユーザーに伝える次のメッセージを返します。次のメッセージ例を考えます。

- 「ピザが食べたい」というユーザーの入力に対して、Amazon Lex はスロットデータを引き出すメッセージをレスポンスとして返します (例えば、`PizzaSize`): 「どのサイズのピザをご希望ですか？」
- ユーザーがピザの注文情報をすべて入力すると、Amazon Lex はユーザーに確認のメッセージ「ピザを注文しますか？」を含むレスポンスを返すかもしれません。
- ユーザーが確認のプロンプトに「はい」と答えると、Amazon Lex は結果ステートメント「ありがとうございます。チーズピザが注文されました」を返します。

すべての Amazon Lex メッセージがユーザーからの応答を必要とするわけではありません。例えば、結果ステートメントはレスポンスを必要としません。メッセージの中には、「はい」か「いいえ」のどちらかのレスポンスしか必要ないものもあります。Amazon Lex は、`message` に加えて、適切なクライアントのユーザーインターフェースを表示するなど、クライアントの動作を強化するために使用できる、レスポンス内のメッセージに関する追加のコンテキストを提供します。次の例を考えます。

- メッセージがスロットデータを引き出すものであれば、Amazon Lex は次のようなコンテキスト情報を返します。
  - `x-amz-lex-dialog-state` ヘッダーが `ElicitSlot` に設定されました
  - 現在のコンテキストのインテント名に `x-amz-lex-intent-name` ヘッダーが設定されました
  - `message` が情報を取得しているスロット名に `x-amz-lex-slot-to-elicit` ヘッダーが設定されました
  - インテントに設定されたスロットのマップとその現在の値に `x-amz-lex-slots` ヘッダーが設定されました

- メッセージが確認プロンプトの場合、x-amz-lex-dialog-state ヘッダは Confirmation に設定され、x-amz-lex-slot-to-elicit ヘッダは省略されます。
- メッセージが、ユーザーのインテントが理解できないことを示す、インテントに設定された明確化プロンプトである場合、x-amz-dialog-state ヘッダは ElicitIntent に設定され、x-amz-slot-to-elicit ヘッダは省略されます。

さらに、Amazon Lex はまた、お客様の用途に合わせた sessionAttributes も返します。詳細については、「Managing Conversation Context」(会話コンテキストの管理) を参照してください。

リクエストの構文

```
POST /bot/botName/alias/botAlias/user/userId/content HTTP/1.1
x-amz-lex-session-attributes: sessionAttributes
x-amz-lex-request-attributes: requestAttributes
Content-Type: contentType
Accept: accept
x-amz-lex-active-contexts: activeContexts

inputStream
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

accept

これは、Accept HTTP ヘッダーの値として渡します。

Amazon Lex がレスポンスで返すメッセージは、リクエストの Accept HTTP ヘッダー値に基づいたテキストまたは音声のどちらかです。

- 値が text/plain; charset=utf-8 の場合、Amazon Lex はレスポンス内にテキストを返します。
- 値が audio/ で始まっている場合、Amazon Lex はレスポンスで音声を返します。Amazon Lex は Amazon Polly を使ってスピーチを生成します (Accept ヘッダーで指定した設定を使用します)。例えば、値として audio/mpeg を指定した場合、Amazon Lex は MPEG 形式の音声を返します。
- 値が audio/pcm の場合、返される音声は audio/pcm で、16 ット、リトルエンディアン形式です。

- 受け入れ可能な値は以下の通りです。
  - audio/mpeg
  - audio/ogg
  - audio/pcm
  - text/plain; charset=utf-8
  - audio/* (デフォルトは mpeg)

activeContexts

リクエストに対してアクティブなコンテクストのリスト。コンテキストは、以前のインテントが達成されたとき、またはリクエストにコンテキストが含まれているときに有効になります。

コンテキストのリストを指定しない場合、Amazon Lex はセッションの現在のコンテキストリストを使用します。空のリストを指定すると、セッションのすべてのコンテキストがクリアされます。

botAlias

Amazon Lex ボットのエイリアス。

必須: はい

botName

Amazon Lex ボットのエイリアス。

必須: はい

contentType

これは、`Content-Type` HTTP ヘッダーの値として渡します。

オーディオ形式またはテキストを示します。ヘッダーの値は、次のプレフィクスのいずれかで始まる必要があります。

- PCM 形式の場合、オーディオデータはリトルエンディアンのバイトオーダーでなければなりません。
  - audio/l16; rate=16000; channels=1
  - audio/x-l16; sample-rate=16000; channel-count=1
  - オーディオ/LPCM; サンプルレート=8000; =16; チャンネル数=1; =false sample-size-bits is-big-endian
- Opus 形式

- オーディオ/ x-cbr-opus-with-プリアンブル; frame-size-milliseconds プリアンブルサイズ=0; ビットレート =256000; =4
- テキスト形式
  - text/plain; charset=utf-8

必須: はい

requestAttributes

これは、`x-amz-lex-request-attributes` HTTP ヘッダーの値として渡します。

クライアントアプリケーションと Amazon Lex の間で受け渡しされるリクエスト固有の情報。値は、文字列のキーと値を持つ JSON シリアライズされた Base64 エンコードされたマップでなければなりません。`requestAttributes` ヘッダーと `sessionAttributes` ヘッダーの合計サイズは 12KB に制限されています。

名前空間 `x-amz-lex:` は、特別な属性のために予約されています。プレフィックス `x-amz-lex:` を持つリクエスト属性を作成しないでください。

リクエスト属性の詳細については、「Setting Requests Attributes」(リクエスト属性の設定) を参照してください。

sessionAttributes

これは、`x-amz-lex-session-attributes` HTTP ヘッダーの値として渡します。

Amazon Lex とクライアントアプリケーションの間で渡されるアプリケーション固有の情報。値は、文字列のキーと値を持つ JSON シリアライズされた Base64 エンコードされたマップでなければなりません。`sessionAttributes` ヘッダーと `requestAttributes` ヘッダーの合計サイズは 12KB に制限されています。

詳細については、「Setting Session Attributes」 (セッション属性の設定) を参照してください。

userId

クライアントアプリケーションユーザーの ID。Amazon Lex は、ユーザーとボットとの会話を識別するために使用します。実行時には、各リクエストに `userID` フィールドを含める必要があります。

アプリケーションに使用するユーザー ID を決定するには、以下の点を考慮してください。

- `userID` フィールドには、ユーザーの個人識別情報 (氏名、個人識別番号、その他のエンドユーザーの個人情報など) を含めることはできません。

- ユーザーがあるデバイスで会話を始め、別のデバイスで会話を続けたい場合は、ユーザー固有の識別子を使用します。
- 同じユーザーが 2 つの異なるデバイスで別々の会話ができるようにする場合は、デバイス固有の識別子を選択します。
- ユーザーは、同じボットの 2 つの異なるバージョンと、2 つの独立した会話をすることはできません。例えば、ユーザーは同じボットの PROD バージョンとベータ版と会話することはできません。例えば、テスト中に、ユーザーが 2 つの異なるバージョンと会話する必要があることが予想される場合は、ユーザー ID にボットエイリアスを含めて、2 つの会話を区切ります。

長さの制限: 最小長は 2 です。最大長は 100 です。

パターン: `[0-9a-zA-Z._:-]+`

必須: はい

リクエストボディ

リクエストは以下のバイナリデータを受け入れます。

inputStream

`Content-Type` HTTP ヘッダーに記述された PCM または Opus オーディオ形式またはテキスト形式のユーザー入力。

オーディオデータを Amazon Lex にストリーミングすることもできますが、送信前にすべてのオーディオデータをキャプチャするローカルバッファを作成することもできます。一般的には、オーディオデータをローカルにバッファリングするよりも、ストリーミングした方がパフォーマンスが向上します。

必須: はい

レスポンスの構文

```
HTTP/1.1 200
Content-Type: contentType
x-amz-lex-intent-name: intentName
x-amz-lex-nlu-intent-confidence: nluIntentConfidence
x-amz-lex-alternative-intents: alternativeIntents
x-amz-lex-slots: slots
x-amz-lex-session-attributes: sessionAttributes
```

```
x-amz-lex-sentiment: sentimentResponse
x-amz-lex-message: message
x-amz-lex-encoded-message: encodedMessage
x-amz-lex-message-format: messageFormat
x-amz-lex-dialog-state: dialogState
x-amz-lex-slot-to-elicit: slotToElicit
x-amz-lex-input-transcript: inputTranscript
x-amz-lex-encoded-input-transcript: encodedInputTranscript
x-amz-lex-bot-version: botVersion
x-amz-lex-session-id: sessionId
x-amz-lex-active-contexts: activeContexts

audioStream
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

レスポンスでは、以下の HTTP ヘッダーが返されます。

activeContexts

セッションのアクティブコンテキストのリスト。コンテキストは、インテントが達成されたとき、または PostContent、PostText、PutSession のオペレーションを呼び出したときに設定できます。

コンテキストを使用して、あるインテントをフォローできるインテントを制御したり、アプリケーションの動作を変更したりすることができます。

alternativeIntents

ユーザーのインテントに該当する可能性のある 1 ~ 4 つの代替インテント。

各選択肢には、そのインテントがユーザーのインテントがどれだけ一致しているかを示す Amazon Lex のスコアが含まれます。インテントは信頼スコアでソートされます。

botVersion

会話にレスポンスしたボットのバージョンです。この情報を使用して、あるバージョンのボットのパフォーマンスが別のバージョンよりも優れているかどうかを判断できます。

長さの制限：最小長は 1 です。最大長は 64 文字です。

パターン: `[0-9]+|\$LATEST`

contentType

リクエストの Accept HTTP ヘッダーで指定されたコンテンツタイプ。

dialogState

ユーザーインタラクションの現在の状態を示します。Amazon Lex から、dialogState のような以下のいずれかの値が返されます。クライアントはオプションで、この情報を使用してユーザーインターフェイスをカスタマイズできます。

- ElicitIntent - Amazon Lex は、ユーザーのインテントを引き出したいと考えています。次の例を考えます。

  例: ユーザーがインテントを発話します (「ピザを注文します」)。Amazon Lex がこの発話からユーザーのインテントを推測できない場合、このダイアログ状態を返します。

- ConfirmIntent - Amazon Lex は、「はい」か「いいえ」のレスポンスを待機しています。

  例えば、Amazon Lex は、インテントを達成する前にユーザーの確認を求めます。ユーザーは、「はい」または「いいえ」という単純なレスポンスではなく、追加の情報をレスポンスすることがあります。例えば、「はい、でも厚めの生地のピザにしてください」や「いいえ、飲み物を注文したいです」などです。Amazon Lex はこのような追加情報を処理できます (この例では、クラストタイプスロットを更新するか、 OrderPizza OrderDrinkインテントをからに変更します)。

- ElicitSlot - Amazon Lex は、現在のインテントのためのスロット値を待機しています。

  例えば、レスポンスで Amazon Lex が「ピザのサイズを教えてください」というメッセージを送信するとします。ユーザーは、スロットの値 (例:「ミディアム」) と答えるかもしれません。また、ユーザーはレスポンスの中で追加情報を提供することもできます (例:「ミディアムサイズの厚い生地のピザ」)。Amazon Lex は、このような追加情報を適切に処理できます。

- Fulfilled - Lambda 関数がインテントを正常に達成したことを伝えます。
- ReadyForFulfillment - クライアントがリクエストを達成する必要性を伝えます。
- Failed - ユーザーとの会話が失敗したことを伝えます。

  これは、ユーザーがサービスからのプロンプトに適切なレスポンスを提供しない場合 (Amazon Lex がユーザーに特定の情報を促す回数を設定できます) や、Lambda 関数がインテントを達成できない場合など、さまざまな理由で発生します。

有効な値 : ElicitIntent | ConfirmIntent | ElicitSlot | Fulfilled | ReadyForFulfillment | Failed

encodedInputTranscript

リクエストの処理に使用されるテキストです。

入力がオーディオストリームであった場合、encodedInputTranscript フィールドにはオーディオストリームから抽出されたテキストが入ります。これは、インテントとスロット値を認識するために実際に処理されるテキストです。この情報を使用して、Amazon Lex が送信したオーディオを正しく処理しているかどうかを判断できます。

encodedInputTranscript フィールドは base-64 エンコードです。値を使用する前に、フィールドをデコードする必要があります。

encodedMessage

ユーザーに伝えるメッセージ。メッセージは、ボットの設定または Lambda 関数から送信できます。

インテントに Lambda 関数が設定されていない場合、または Lambda 関数が応答で dialogAction.type として Delegate を返した場合、Amazon Lex は次の行動を決定し、現在のインタラクションコンテキストに基づいてボットの設定から適切なメッセージを選択します。例えば、Amazon Lex がユーザー入力を理解できない場合、明確化プロンプトメッセージを使用します。

インテントを作成するときに、グループにメッセージを割り当てることができます。メッセージがグループに割り当てられている場合、Amazon Lex は各グループから 1 つのメッセージをレスポンスに返します。メッセージフィールドは、メッセージを含むエスケープされた JSON 文字列です。返される JSON 文字列の構造の詳細については、「サポートされているメッセージ形式」を参照してください。

Lambda 関数がメッセージを返した場合、Amazon Lex はそれをレスポンスでクライアントに渡します。

encodedMessage フィールドは base-64 エンコードです。値を使用する前に、フィールドをデコードする必要があります。

長さの制限：最小長は 1 です。最大長は 1366 です。

inputTranscript

このヘッダーは廃止されました。

リクエストの処理に使用されるテキストです。

このフィールドは、de-DE、en-AU、en-GB、en-US、es-419、es-ES、es-US、fr-CA、fr-FR、および it-IT ロケールでのみ使用できます。その他すべてのロケールでは、`inputTranscript` フィールドが null です。代わりに `encodedInputTranscript` のフィールドを使用する必要があります。

入力がオーディオストリームであった場合、`inputTranscript` フィールドにはオーディオストリームから抽出されたテキストが入ります。これは、インテントとスロット値を認識するために実際に処理されるテキストです。この情報を使用して、Amazon Lex が送信したオーディオを正しく処理しているかどうかを判断できます。

intentName

Amazon Lex が認識している現在のユーザーのインテント。

message

このヘッダーは廃止されました。

このフィールドは、de-DE、en-AU、en-GB、en-US、es-419、es-ES、es-US、fr-CA、fr-FR、および it-IT ロケールでのみ使用できます。その他すべてのロケールでは、`message` フィールドが null です。代わりに `encodedMessage` のフィールドを使用する必要があります。

ユーザーに伝えるメッセージ。メッセージは、ボットの設定または Lambda 関数から送信できます。

インテントに Lambda 関数が設定されていない場合、または Lambda 関数が応答で `dialogAction.type` として `Delegate` を返した場合、Amazon Lex は次の行動を決定し、現在のインタラクションコンテキストに基づいてボットの設定から適切なメッセージを選択します。例えば、Amazon Lex がユーザー入力を理解できない場合、明確化プロンプトメッセージを使用します。

インテントを作成するときに、グループにメッセージを割り当てることができます。メッセージがグループに割り当てられている場合、Amazon Lex は各グループから 1 つのメッセージをレスポンスに返します。メッセージフィールドは、メッセージを含むエスケープされた JSON 文字列です。返される JSON 文字列の構造の詳細については、「サポートされているメッセージ形式」を参照してください。

Lambda 関数がメッセージを返した場合、Amazon Lex はそれをレスポンスでクライアントに渡します。

長さの制限: 最小長は 1 です。最大長は 1,024 です。

messageFormat

応答メッセージの形式。次のいずれかの値になります。

- PlainText - メッセージには UTF-8 形式テキストが含まれています。
- CustomPayload - メッセージはクライアント向けのカスタム形式です。
- SSML - メッセージには音声出力のテキスト形式が含まれています。
- Composite - メッセージには、インテントの作成時にメッセージが割り当てられたグループからの 1 つ以上のメッセージを含むエスケープされた JSON オブジェクトが含まれています。

有効な値：PlainText | CustomPayload | SSML | Composite

nluIntentConfidence

返されたインテントがユーザーのインテントに合致するものであると Amazon Lex がどれだけ確信しているかを示すスコアを提供します。スコアは 0.0 ~ 1.0 の間です。

スコアは相対的なものであり、絶対的なものではありません。スコアは、Amazon Lex の改善により変更されることがあります。

sentimentResponse

発話で表現されるセンチメント。

ボットがセンチメント分析のために Amazon Comprehend に発話を送信するように設定されている場合、このフィールドには分析の結果が含まれます。

sessionAttributes

セッション固有のコンテキスト情報を表すキーバリューのペアのマップ。

sessionId

セッションの一意の識別子。

slots

会話中にユーザーの入力から検出されたゼロ以上のインテントスロット (名前と値のペア) Amazon Lex のマップ。フィールドは base-64 エンコードです。

Amazon Lex は、あるスロットに対して可能性の高い値を含む解決リストを作成します。返す値は、スロットタイプの作成時や更新時に選択された valueSelectionStrategy によって決まります。valueSelectionStrategy が ORIGINAL_VALUE に設定されている場合、ユーザーの値がスロットの値と類似していれば、ユーザーが提供した値が

返されます。valueSelectionStrategy が TOP_RESOLUTION Amazon Lex に設定されている場合は、解決リストの最初の値を返し、解決リストがない場合は null を返します。valueSelectionStrategy を指定しなかった場合、デフォルトは ORIGINAL_VALUE です。

slotToElicit

dialogState の値が ElicitSlot の場合、Amazon Lex が値を引き出すスロットの名前を返します。

レスポンスは、HTTP 本文として以下を返します。

audioStream

ユーザーに伝えるプロンプトまたはステートメントです。これは、ボットの構成とコンテキストに基づいています。例えば、Amazon Lex がユーザーのインテントを理解できなかった場合、ボット用に設定された clarificationPrompt を送信します。インテントがフルフィルメントアクションを達成する前に確認が必要な場合は、confirmationPrompt を送信します。別の例: Lambda 関数がインテントを正常に達成し、ユーザーに伝えるメッセージを送信したとします。その後、Amazon Lex はそのメッセージをレスポンスに送信します。

エラー

BadGatewayException

Amazon Lex ボットがまだ構築中であるか、依存するサービス (Amazon Polly、AWS Lambda) の 1 つが内部サービスエラーで失敗したかのどちらかです。

HTTP ステータスコード: 502

BadRequestException

リクエストの検証に失敗したか、コンテキストに使用可能なメッセージがないか、ボットの構築が失敗もしくは進行中であるか、または構築されていない変更が含まれています。

HTTP ステータスコード : 400

ConflictException

2 つのクライアントが同じ AWS アカウント、Amazon Lex bot、ユーザー ID を使用しています。

HTTP ステータスコード: 409

DependencyFailedException

AWS Lambda や Amazon Polly などの依存関係にあるものが、例外を発生させました。例、

- Amazon Lex が Lambda 関数を呼び出すのに十分な権限を持っていない場合。
- Lambda 関数の実行に 30 秒以上かかる場合。
- フルフィルメントの Lambda 関数が、スロットの値を削除せずに `Delegate` ダイアログアクションを返す場合。

HTTP ステータスコード: 424

InternalFailureException

内部サービスエラー。呼び出しを再試行します。

HTTP ステータスコード : 500

LimitExceededException

制限を超えました。

HTTP ステータスコード: 429

LoopDetectedException

この例外は使われません。

HTTP ステータスコード: 508

NotAcceptableException

リクエストの accept ヘッダーに有効な値がありません。

HTTP ステータスコード: 406

NotFoundException

参照するリソース (Amazon Lex bot やエイリアスなど) が見つかりません。

HTTP ステータスコード: 404

RequestTimeoutException

入力音声が長すぎます。

HTTP ステータスコード: 408

UnsupportedMediaTypeException

Content-Typeヘッダー (PostContent API) の値が無効です。

HTTP ステータスコード: 415

例

例 1

このリクエストでは、URI がボット (Traffic)、ボットのバージョン ($LATEST)、エンドユーザー名 (someuser) を特定しています。Content-Type ヘッダーは、本文内のオーディオ形式を識別します。Amazon Lex はまた、他の形式をサポートしています。必要に応じて、オーディオをある形式から別の形式に変換するために、オープンソースソフトウェアの SoX を使用することができます。Accept HTTP ヘッダーを追加することで、レスポンスを得るための形式を指定します。

レスポンスでは、x-amz-lex-message ヘッダーがAmazon Lex が返したレスポンスを示しています。そして、クライアントはこのレスポンスをユーザーに送ることができます。同じメッセージを、(リクエストに応じて) チャンクドエンコーディングによりオーディオ/MPEG形式で送信します。

リクエスト例

```
"POST /bot/Traffic/alias/$LATEST/user/someuser/content HTTP/1.1[\r][\n]"
"x-amz-lex-session-attributes: eyJ1c2VyTmFtZSI6IkJvYiJ9[\r][\n]"
"Content-Type: audio/x-l16; channel-count=1; sample-rate=16000f[\r][\n]"
"Accept: audio/mpeg[\r][\n]"
"Host: runtime.lex.us-east-1.amazonaws.com[\r][\n]"
"Authorization: AWS4-HMAC-SHA256 Credential=BLANKED_OUT/20161230/us-east-1/lex/
aws4_request,
SignedHeaders=accept;content-type;host;x-amz-content-sha256;x-amz-date;x-amz-lex-
session-attributes,
 Signature=78ca5b54ea3f64a17ff7522de02cd90a9acd2365b45a9ce9b96ea105bb1c7ec2[\r][\n]"
"X-Amz-Date: 20161230T181426Z[\r][\n]"
"X-Amz-Content-Sha256:
 e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855[\r][\n]"
"Transfer-Encoding: chunked[\r][\n]"
"Connection: Keep-Alive[\r][\n]"
"User-Agent: Apache-HttpClient/4.5.x (Java/1.8.0_112)[\r][\n]"
"Accept-Encoding: gzip,deflate[\r][\n]"
"[\r][\n]"
"1000[\r][\n]"
"[0x7][0x0][0x7][0x0][\n]"
```

```
"[0x0][0x7][0x0][0xfc][0xff][\n]"
"[0x0][\n]"
…
```

レスポンス例

```
"HTTP/1.1 200 OK[\r][\n]"
"x-amzn-RequestId: cc8b34af-cebb-11e6-a35c-55f3a992f28d[\r][\n]"
"x-amz-lex-message: Sorry, can you repeat that?[\r][\n]"
"x-amz-lex-dialog-state: ElicitIntent[\r][\n]"
"x-amz-lex-session-attributes: eyJ1c2VyTmFtZSI6IkJvYiJ9[\r][\n]"
"Content-Type: audio/mpeg[\r][\n]"
"Transfer-Encoding: chunked[\r][\n]"
"Date: Fri, 30 Dec 2016 18:14:28 GMT[\r][\n]"
"[\r][\n]"
"2000[\r][\n]"
"ID3[0x4][0x0][0x0][0x0][0x0][0x0]#TSSE[0x0][0x0][0x0][0xf][0x0][0x0]
[0x3]Lavf57.41.100[0x0][0x0][0x0][0x0][0x0][0x0][0x0][0x0][0x0][0x0][0x0][0xff]
[0xf3]`[0xc4][0x0][0x1b]{[0x8d][0xe8][0x1]C[0x18][0x1][0x0]J[0xe0]`b[0xdd][0xd1]
[0xb][0xfd][0x11][0xdf][0xfe]";[0xbb][0xbb][0x9f][0xee][0xee][0xee][0xee]|DDD/[0xff]
[0xff][0xff][0xff]www?D[0xf7]w^?[0xff][0xfa]h[0x88][0x85][0xfe][0x88][0x88][0x88]
[[0xa2]'[0xff][0xfa]"{[0x9f][0xe8][0x88]]D[0xeb][0xbb][0xbb][0xa2]!u[0xfd][0xdd][0xdf]
[0x88][0x94][0x0]F[0xef][0xa1]8[0x0][0x82]w[0x88]N[0x0][0x0][0x9b][0xbb][0xe8][0xe
…
```

その他の参照資料

この API を言語固有の AWS SDK で使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go バージョン 2 用 SDK
- AWS Java V2 用 SDK
- AWS V3 用 JavaScript SDK
- AWS PHP V3 用SDK
- AWS Python 用 SDK
- AWS ルビー V3 用 SDK

PostText

サービス: Amazon Lex Runtime Service

ユーザー入力 を Amazon Lex に送信します。クライアントアプリケーションは、この API を使用して、実行時に Amazon Lex にリクエストを送信できます。そして、Amazon Lex は、ボットのために構築した機械学習モデルを使って、ユーザーの入力を解釈します。

これに対して Amazon Lex は、次の `message` を返してユーザーに任意の `responseCard` を表示するように伝えます。次のメッセージ例を考えます。

- ユーザーが「ピザが欲しい」と入力すると、Amazon Lex は「どのサイズのピザがいいですか?」というスロットデータを引き出すメッセージを含むレスポンスを返す場合があります (例: PizzaSize)。
- ユーザーがピザの注文情報をすべて提供した後、Amazon Lex はユーザーに確認のメッセージを含むレスポンスを返すかもしれません。「ピザの注文を続けますか？」
- ユーザーが確認のプロンプトに「はい」と答えた後、Amazon Lex は結果ステートメントを返します。「ありがとうございます。チーズピザが注文されました」

すべての Amazon Lex メッセージにユーザーの反応を必要なわけではありません。例えば、結果ステートメントはレスポンスが不要です。メッセージの中には、「はい」か「いいえ」のどちらかのレスポンスしか必要ないものもあります。Amazon Lex は、`message` に加えて、レスポンス内のメッセージに関する追加のコンテキストを提供します。例えば、適切なクライアントのユーザーインターフェースを表示するなど、クライアントの動作を強化するために使用することができます。これがレスポンスの`slotToElicit`、`dialogState`、`intentName`、`slots` の各フィールドです。次の例を考えます。

- メッセージがスロットデータを引き出すものであれば、Amazon Lex は次のようなコンテキスト情報を返します。
  - `dialogState`に設定 ElicitSlot
  - `intentName` が現在のコンテキストのインテント名に設定されました
  - `message` が情報を取得するスロット名に `slotToElicit` が設定されました
  - `slots` がインテントに応じて設定された既知の値を持つスロットのマップに設定されました
- メッセージが確認プロンプトの場合、`dialogState`は null ConfirmIntent に設定されれ`SlotToElicit`、null に設定されます。

- メッセージがユーザーの意図が理解できないことを示す明確化プロンプト (その意図に合わせて設定された) である場合、は null ElicitIntent に設定され、null `slotToElicit` に設定されます。`dialogState`

さらに、Amazon Lex はまた、お客様の用途に合わせた `sessionAttributes` も返します。詳細については、「Managing Conversation Context」(会話コンテキストの管理) を参照してください。

リクエストの構文

```
POST /bot/botName/alias/botAlias/user/userId/text HTTP/1.1
Content-type: application/json

{
   "activeContexts": [
      {
         "name": "string",
         "parameters": {
            "string" : "string"
         },
         "timeToLive": {
            "timeToLiveInSeconds": number,
            "turnsToLive": number
         }
      }
   ],
   "inputText": "string",
   "requestAttributes": {
      "string" : "string"
   },
   "sessionAttributes": {
      "string" : "string"
   }
}
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

botAlias

Amazon Lex ボットのエイリアス。

必須: はい

botName

Amazon Lex ボットの名前。

必須: はい

userId

クライアントアプリケーションユーザーの ID。Amazon Lex は、ユーザーとボットとの会話を識別するために使用します。実行時には、各リクエストに userID フィールドを含める必要があります。

アプリケーションに使用するユーザー ID を決定するには、以下の点を考慮してください。

- userID フィールドには、ユーザーの個人識別情報 (氏名、個人識別番号、その他のエンドユーザーの個人情報など) を含めることはできません。
- ユーザーがあるデバイスで会話を始め、別のデバイスで会話を続けたい場合は、ユーザー固有の識別子を使用します。
- 同じユーザーが 2 つの異なるデバイスで別々の会話ができるようにする場合は、デバイス固有の識別子を選択します。
- ユーザーは、同じボットの 2 つの異なるバージョンと、2 つの独立した会話をすることはできません。例えば、ユーザーは同じボットの PROD バージョンとベータ版と会話することはできません。例えば、テスト中に、ユーザーが 2 つの異なるバージョンと会話する必要があることが予想される場合は、ユーザー ID にボットエイリアスを含めて、2 つの会話を区切ります。

長さの制限: 最小長は 2 です。最大長は 100 です。

パターン: [0-9a-zA-Z._:-]+

必須: はい

リクエストボディ

リクエストは以下の JSON 形式のデータを受け入れます。

activeContexts

リクエストに対してアクティブなコンテキストのリスト。コンテキストは、以前のインテントが達成されたとき、またはリクエストにコンテキストが含まれているときに有効になります。

コンテキストのリストを指定しない場合、Amazon Lex はセッションの現在のコンテキストリストを使用します。空のリストを指定すると、セッションのすべてのコンテキストがクリアされます。

型: ActiveContext オブジェクトの配列

配列メンバー : 最小数は 0 項目です。最大数は 20 項目です。

必須: いいえ

inputText

ユーザーが入力したテキスト (Amazon Lex はこのテキストを解釈します)。

AWS CLI を使用している場合、`--input-text` パラメータに URL を渡すことはできません。代わりに `--cli-input-json` パラメータを使って URL を渡します。

型: 文字列

長さの制限: 最小長は 1 です。最大長は 1,024 です。

必須: はい

requestAttributes

クライアントアプリケーションと Amazon Lex の間で受け渡しされるリクエスト固有の情報。

名前空間 `x-amz-lex:` は、特別な属性のために予約されています。プレフィックス `x-amz-lex:` を持つリクエスト属性を作成しないでください。

リクエスト属性の詳細については、「Setting Requests Attributes」(リクエスト属性の設定) を参照してください。

型: 文字列間のマッピング

必須: いいえ

sessionAttributes

Amazon Lex とクライアントアプリケーションの間で渡されるアプリケーション固有の情報。

詳細については、「Setting Session Attributes」(リクエスト属性の設定) を参照してください。

型: 文字列間のマッピング

必須: いいえ

レスポンスの構文

```
HTTP/1.1 200
Content-type: application/json

{
   "activeContexts": [
      {
         "name": "string",
         "parameters": {
            "string" : "string"
         },
         "timeToLive": {
            "timeToLiveInSeconds": number,
            "turnsToLive": number
         }
      }
   ],
   "alternativeIntents": [
      {
         "intentName": "string",
         "nluIntentConfidence": {
            "score": number
         },
         "slots": {
            "string" : "string"
         }
      }
   ],
   "botVersion": "string",
   "dialogState": "string",
   "intentName": "string",
   "message": "string",
   "messageFormat": "string",
   "nluIntentConfidence": {
      "score": number
   },
   "responseCard": {
      "contentType": "string",
      "genericAttachments": [
         {
            "attachmentLinkUrl": "string",
            "buttons": [
               {
```

```
                "text": "string",
                "value": "string"
            }
        ],
        "imageUrl": "string",
        "subTitle": "string",
        "title": "string"
      }
    ],
    "version": "string"
  },
  "sentimentResponse": {
    "sentimentLabel": "string",
    "sentimentScore": "string"
  },
  "sessionAttributes": {
    "string" : "string"
  },
  "sessionId": "string",
  "slots": {
    "string" : "string"
  },
  "slotToElicit": "string"
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

activeContexts

セッションのアクティブコンテキストのリスト。コンテキストは、インテントが達成されたとき、または PostContent、PostText、PutSession のオペレーションを呼び出したときに設定できます。

コンテキストを使用して、あるインテントをフォローできるインテントを制御したり、アプリケーションの動作を変更したりすることができます。

型: ActiveContext オブジェクトの配列

配列メンバー：最小数は 0 項目です。最大数は 20 項目です。

alternativeIntents

ユーザーのインテントに該当する可能性のある 1～4 つの代替インテント。

各選択肢には、そのインテントがユーザーのインテントがどれだけ一致しているかを示す Amazon Lex のスコアが含まれます。インテントは信頼スコアでソートされます。

型: PredictedIntent オブジェクトの配列

配列メンバー: 最大数は 4 項目です。

botVersion

会話にレスポンスしたボットのバージョンです。この情報を使用して、あるバージョンのボットのパフォーマンスが別のバージョンよりも優れているかどうかを判断できます。

型: 文字列

長さの制限：最小長は 1 です。最大長は 64 文字です。

パターン: `[0-9]+|\$LATEST`

dialogState

ユーザーインタラクションの現在の状態を示します。Amazon Lex から、dialogState のような以下のいずれかの値が返されます。クライアントはオプションで、この情報を使用してユーザーインターフェイスをカスタマイズできます。

- `ElicitIntent` - Amazon Lex は、ユーザーのインテントを引き出したいと考えています。

  例: ユーザーがインテントを発話します (「ピザを注文します」)。Amazon Lex がこの発話からユーザーインテントを推測できない場合、この DialogState を返します。
- `ConfirmIntent` - Amazon Lex は、「はい」か「いいえ」のレスポンスを待機しています。

  例えば、Amazon Lex は、インテントを達成する前にユーザーの確認を求めます。

  単純な「はい」または「いいえ」ではなく、ユーザーは追加情報で応答する場合があります。例えば、「はい、でも厚い生地のピザにします」や「いいえ、飲み物を注文したいです」などです。Amazon Lex はこのような追加情報を処理できます (この例では、クラストタイプのスロット値を更新するか、 OrderPizza OrderDrinkインテントをからに変更します)。
- `ElicitSlot` - Amazon Lex は、現在のインテントのスロット値を待機しています。

  例えば、レスポンスで Amazon Lex が「ピザのサイズを教えてください」というメッセージが送信するとします。ユーザーは、スロットの値 (例:「ミディアム」) と答えるかもしれません。

また、ユーザーはレスポンスの中で追加情報を提供することもできます (例:「ミディアムサイズの厚い生地のピザ」)。Amazon Lex は、このような追加情報を適切に処理できます。

- `Fulfilled` - インテントに設定された Lambda 関数が、正常にインテントを達成したことを伝えます。
- `ReadyForFulfillment` - クライアントがインテントを達成するために必要なことを伝えます。
- `Failed` - ユーザーとの会話が失敗したことを伝えます。

  これは、ユーザーがサービスからのプロンプトに適切なレスポンスを提供しない場合 (Amazon Lex がユーザーに特定の情報を促す回数を設定できます) や、Lambda 関数がインテントを達成できない場合など、さまざまな理由で発生します。

型: 文字列

有効な値 : `ElicitIntent | ConfirmIntent | ElicitSlot | Fulfilled | ReadyForFulfillment | Failed`

intentName

Amazon Lex が認識している現在のユーザーインテントです。

型: 文字列

message

ユーザーに伝えるメッセージ。メッセージは、ボットの設定または Lambda 関数から送信できます。

インテントに Lambda 関数が設定されていない場合、または Lambda 関数が `dialogAction.type` のその応答として `Delegate` を返した場合、Amazon Lex は次の行動を決定し、現在のインタラクションコンテキストに基づいてボットの構成から適切なメッセージを選択します。例えば、Amazon Lex がユーザー入力を理解できない場合、明確化プロンプトメッセージを使用します。

インテントを作成するときに、グループにメッセージを割り当てることができます。メッセージがグループに割り当てられている場合、Amazon Lex は各グループから 1 つのメッセージをレスポンスに返します。メッセージフィールドは、メッセージを含むエスケープされた JSON 文字列です。返される JSON 文字列の構造の詳細については、「サポートされているメッセージ形式」を参照してください。

Lambda 関数がメッセージを返した場合、Amazon Lex はそれをレスポンスでクライアントに渡します。

型: 文字列

長さの制限: 最小長は 1 です。最大長は 1,024 です。

messageFormat

応答メッセージの形式。次のいずれかの値になります。

- PlainText - メッセージには UTF-8 形式テキストが含まれています。
- CustomPayload - メッセージは Lambda 関数で定義されるカスタム形式です。
- SSML - メッセージには音声出力のテキスト形式が含まれています。
- Composite - メッセージには、インテントの作成時にメッセージが割り当てられたグループからの 1 つ以上のメッセージを含むエスケープされた JSON オブジェクトが含まれています。

型: 文字列

有効な値 : PlainText | CustomPayload | SSML | Composite

nluIntentConfidence

返されたインテントがユーザーのインテントに合致するものであると Amazon Lex がどれだけ確信しているかを示すスコアを提供します。スコアは 0.0 ~ 1.0 の間です。詳細については、「Confidence Scores」(信頼スコア) を参照してください。

スコアは相対的なものであり、絶対的なものではありません。スコアは、Amazon Lex の改善により変更されることがあります。

型: IntentConfidence オブジェクト

responseCard

ユーザーが現在のプロンプトに応答するためのオプションを表します。レスポンスカードは、ボットの設定 (Amazon Lex コンソールで、スロットの横にある設定ボタンを選択) からも、コードフック (Lambda 関数) からもアクセスすることができます。

型: ResponseCard オブジェクト

sentimentResponse

ある発話で表現されたセンチメント。

ボットがセンチメント分析のために Amazon Comprehend に発話を送信するように設定されている場合、このフィールドには分析の結果が含まれます。

型: SentimentResponse オブジェクト

sessionAttributes

セッション固有のコンテキスト情報を表すキーバリューのペアのマップ。

型: 文字列間のマッピング

sessionId

セッションの一意の識別子。

型: 文字列

slots

会話中のユーザーの入力内容から Amazon Lex が検出したインテントスロット。

Amazon Lex は、あるスロットに対して可能性の高い値を含む解決リストを作成します。返す値は、スロットタイプの作成時や更新時に選択された `valueSelectionStrategy` によって決まります。`valueSelectionStrategy` が `ORIGINAL_VALUE` に設定されている場合、ユーザーの値がスロットの値と類似していれば、ユーザーが提供した値が返されます。`valueSelectionStrategy` が `TOP_RESOLUTION` Amazon Lex に設定されている場合は、解決リストの最初の値を返し、解決リストがない場合は null を返します。`valueSelectionStrategy` を指定しなかった場合、デフォルトは `ORIGINAL_VALUE` です。

型: 文字列間のマッピング

slotToElicit

`dialogState` の値が `ElicitSlot` の場合、Amazon Lex が値を引き出すスロットの名前を返します。

型: 文字列

エラー

BadGatewayException

Amazon Lex ボットがまだ構築中であるか、依存するサービス (Amazon Polly、AWS Lambda) の 1 つが内部サービスエラーで失敗したかのどちらかです。

HTTP ステータスコード: 502

BadRequestException

リクエストの検証に失敗したか、コンテキストに使用可能なメッセージがないか、ボットの構築が失敗もしくは進行中であるか、または構築されていない変更が含まれています。

HTTP ステータスコード : 400

ConflictException

2 つのクライアントが同じ AWS アカウント、Amazon Lex bot、ユーザー ID を使用しています。

HTTP ステータスコード: 409

DependencyFailedException

AWS Lambda や Amazon Polly などの依存関係にあるものが、例外を発生させました。例、

- Amazon Lex が Lambda 関数を呼び出すのに十分な権限を持っていない場合。
- Lambda 関数の実行に 30 秒以上かかる場合。
- フルフィルメントの Lambda 関数が、スロットの値を削除せずに `Delegate` ダイアログアクションを返す場合。

HTTP ステータスコード: 424

InternalFailureException

内部サービスエラー。呼び出しを再試行します。

HTTP ステータスコード : 500

LimitExceededException

制限を超えました。

HTTP ステータスコード: 429

LoopDetectedException

この例外は使われません。

HTTP ステータスコード: 508

NotFoundException

参照するリソース (Amazon Lex bot やエイリアスなど) が見つかりません。

HTTP ステータスコード: 404

その他の参照資料

言語固有の AWS SDK の 1 つでこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go バージョン 2 用 SDK
- AWS Java V2 用 SDK
- AWS V3 用 JavaScript SDK
- AWS PHP V3 用SDK
- AWS Python 用 SDK
- AWS ルビー V3 用 SDK

PutSession

サービス: Amazon Lex Runtime Service

Amazon Lex ボットで新しいセッションを作成するか、既存のセッションを変更します。このオペレーションにより、アプリケーションでボットの状態を設定できるようになります。

詳細については、「セッションの管理」を参照してください。

リクエストの構文

```
POST /bot/botName/alias/botAlias/user/userId/session HTTP/1.1
Accept: accept
Content-type: application/json

{
   "activeContexts": [
      {
         "name": "string",
         "parameters": {
            "string" : "string"
         },
         "timeToLive": {
            "timeToLiveInSeconds": number,
            "turnsToLive": number
         }
      }
   ],
   "dialogAction": {
      "fulfillmentState": "string",
      "intentName": "string",
      "message": "string",
      "messageFormat": "string",
      "slots": {
         "string" : "string"
      },
      "slotToElicit": "string",
      "type": "string"
   },
   "recentIntentSummaryView": [
      {
         "checkpointLabel": "string",
         "confirmationStatus": "string",
         "dialogActionType": "string",
```

```
        "fulfillmentState": "string",
        "intentName": "string",
        "slots": {
           "string" : "string"
        },
        "slotToElicit": "string"
     }
  ],
  "sessionAttributes": {
     "string" : "string"
  }
}
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

accept

Amazon Lex がレスポンスで返すメッセージは、このフィールドの値に応じて、テキストまたは音声ベースになります。

- 値が `text/plain; charset=utf-8` の場合、Amazon Lex はレスポンス内にテキストを返します。
- 値が `audio/` で始まっている場合、Amazon Lex はレスポンスで音声を返します。Amazon Lex は Amazon Polly を使用して、指定された構成で音声を生成します。例えば、値として `audio/mpeg` を指定した場合、Amazon Lex は MPEG 形式の音声を返します。
- 値が `audio/pcm` の場合、返される音声は `audio/pcm` で、16 ット、リトルエンディアン形式です。
- 受け入れ可能な値は以下の通りです。
  - `audio/mpeg`
  - `audio/ogg`
  - `audio/pcm`
  - `audio/*` (デフォルトは mpeg)
  - `text/plain; charset=utf-8`

botAlias

セッションデータを含むボットに使用されるエイリアス。

必須: はい

botName

セッションデータを含むボットの名前。

必須: はい

userId

クライアントアプリケーションユーザーの ID。Amazon Lex は、ユーザーとボットとの会話を識別するために使用します。

長さの制限: 最小長は 2 です。最大長は 100 です。

パターン: `[0-9a-zA-Z._:-]+`

必須: はい

リクエストボディ

リクエストは以下の JSON 形式のデータを受け入れます。

activeContexts

リクエストに対してアクティブなコンテキストのリスト。コンテキストは、以前のインテントが達成されたとき、またはリクエストにコンテキストが含まれているときに有効になります。

コンテキストのリストを指定しない場合、Amazon Lex はセッションの現在のコンテキストリストを使用します。空のリストを指定すると、セッションのすべてのコンテキストがクリアされます。

型: ActiveContext オブジェクトの配列

配列メンバー：最小数は 0 項目です。最大数は 20 項目です。

必須: いいえ

dialogAction

会話を成立させるために、ボットが取るべき次のアクションを設定します。

型: DialogAction オブジェクト

必須: いいえ

recentIntentSummaryView

ボットの最近のインテントのサマリー。インテントサマリービューを使って、インテントにチェックポイントラベルを設定したり、インテントの属性を変更したりすることができます。また、インテントサマリーオブジェクトをリストから削除したり、追加したりするのにも使用できます。

修正したり、リストに追加するインテントは、ボットにとって意味のあるものでなければなりません。例えば、インテント名はボットが検証できるインテントでなければなりません。有効な値を指定する必要があります:

- intentName
- スロットの名前
- slotToElict

PutSession リクエストで recentIntentSummaryView パラメータを送信した場合、新しいサマリービューの内容が古いサマリービューに置き換わることになります。例えば、GetSession のリクエストがサマリービューで 3 つのインテントを返し、PutSession をサマリービューで 1 つのインテントで呼び出した場合、次に GetSession を呼び出しても 1 つのインテントしか返されません。

型: IntentSummary オブジェクトの配列

配列メンバー : 最小数は 0 項目です。最大数は 3 項目です。

必須: いいえ

sessionAttributes

セッション固有のコンテキスト情報を表すキーバリューのペアのマップ。Amazon Lex とクライアントアプリケーションの間で渡されるアプリケーション情報を含みます。

型: 文字列間のマッピング

必須: いいえ

レスポンスの構文

```
HTTP/1.1 200
Content-Type: contentType
x-amz-lex-intent-name: intentName
```

```
x-amz-lex-slots: slots
x-amz-lex-session-attributes: sessionAttributes
x-amz-lex-message: message
x-amz-lex-encoded-message: encodedMessage
x-amz-lex-message-format: messageFormat
x-amz-lex-dialog-state: dialogState
x-amz-lex-slot-to-elicit: slotToElicit
x-amz-lex-session-id: sessionId
x-amz-lex-active-contexts: activeContexts

audioStream
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

レスポンスでは、以下の HTTP ヘッダーが返されます。

activeContexts

セッションのアクティブコンテキストのリスト。

contentType

リクエストの Accept HTTP ヘッダーで指定されたコンテンツタイプ。

dialogState

- ConfirmIntent - Amazon Lex は、インテントを達成する前に、インテントを確認するために「はい」か「いいえ」のレスポンスを待機しています。
- ElicitIntent - Amazon Lex は、ユーザーのインテントを引き出したいと考えています。
- ElicitSlot - Amazon Lex は、現在のインテントのためのスロット値を想定しています。
- Failed - ユーザーとの会話が失敗したことを伝えます。これは、ユーザーがサービスからのプロンプトに適切なレスポンスを提供しない場合 (Amazon Lex がユーザーに特定の情報を促す回数を設定できます) や、Lambda 関数がインテントを達成できない場合など、さまざまな理由で発生します。
- Fulfilled - Lambda 関数がインテントを正常に達成したことを伝えます。
- ReadyForFulfillment - クライアントがインテントを達成する必要性を伝えます。

有効な値 : ElicitIntent | ConfirmIntent | ElicitSlot | Fulfilled | ReadyForFulfillment | Failed

encodedMessage

ユーザーに提示する次のメッセージ。

encodedMessage フィールドは base-64 エンコードです。値を使用する前に、フィールドをデコードする必要があります。

長さの制限：最小長は 1 です。最大長は 1366 です。

intentName

現在のインテントの名前。

message

このヘッダーは廃止されました。

ユーザーに提示する次のメッセージ。

このフィールドは、de-DE、en-AU、en-GB、en-US、es-419、es-ES、es-US、fr-CA、fr-FR、および it-IT ロケールでのみ使用できます。その他すべてのロケールでは、message フィールドが null です。代わりに encodedMessage のフィールドを使用する必要があります。

長さの制限: 最小長は 1 です。最大長は 1,024 です。

messageFormat

応答メッセージの形式。次のいずれかの値になります。

- PlainText - メッセージには UTF-8 形式テキストが含まれています。
- CustomPayload - メッセージはクライアント向けのカスタム形式です。
- SSML - メッセージには音声出力のテキスト形式が含まれています。
- Composite - メッセージには、インテントの作成時にメッセージが割り当てられたグループからの 1 つ以上のメッセージを含むエスケープされた JSON オブジェクトが含まれています。

有効な値：PlainText | CustomPayload | SSML | Composite

sessionAttributes

セッション固有のコンテキスト情報を表すキーバリューのペアのマップ。

sessionId

セッションの一意の識別子。

slots

会話中にユーザーの入力から検出されたゼロ以上のインテントスロット (名前と値のペア) Amazon Lex のマップ。

Amazon Lex は、あるスロットに対して可能性の高い値を含む解決リストを作成します。返す値は、スロットタイプの作成時や更新時に選択された valueSelectionStrategy によって決まります。valueSelectionStrategy が ORIGINAL_VALUE に設定されている場合、ユーザーの値がスロットの値と類似していれば、ユーザーが提供した値が返されます。valueSelectionStrategy が TOP_RESOLUTION Amazon Lex に設定されている場合は、解決リストの最初の値を返し、解決リストがない場合は null を返します。valueSelectionStrategy を指定しない場合、デフォルトは ORIGINAL_VALUE です。

slotToElicit

dialogState が ElicitSlot の場合、Amazon Lex が値を引き出しているスロットの名前を返します。

レスポンスは、HTTP 本文として以下を返します。

audioStream

ユーザーに伝えるメッセージのオーディオバージョン。

エラー

BadGatewayException

Amazon Lex ボットがまだ構築中であるか、依存するサービス (Amazon Polly、AWS Lambda) の 1 つが内部サービスエラーで失敗したかのどちらかです。

HTTP ステータスコード: 502

BadRequestException

リクエストの検証に失敗したか、コンテキストに使用可能なメッセージがないか、ボットの構築が失敗もしくは進行中であるか、または構築されていない変更が含まれています。

HTTP ステータスコード : 400

ConflictException

2 つのクライアントが同じ AWS アカウント、Amazon Lex bot、ユーザー ID を使用しています。

HTTP ステータスコード: 409

DependencyFailedException

AWS Lambda や Amazon Polly などの依存関係にあるものが、例外を発生させました。例、

- Amazon Lex が Lambda 関数を呼び出すのに十分な権限を持っていない場合。
- Lambda 関数の実行に 30 秒以上かかる場合。
- フルフィルメントの Lambda 関数が、スロットの値を削除せずに `Delegate` ダイアログアクションを返す場合。

HTTP ステータスコード: 424

InternalFailureException

内部サービスエラー。呼び出しを再試行します。

HTTP ステータスコード : 500

LimitExceededException

制限を超えました。

HTTP ステータスコード: 429

NotAcceptableException

リクエストの accept ヘッダーに有効な値がありません。

HTTP ステータスコード: 406

NotFoundException

参照するリソース (Amazon Lex bot やエイリアスなど) が見つかりません。

HTTP ステータスコード: 404

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go バージョン 2 用 SDK

- AWS Java V2 用 SDK
- AWS V3 用 JavaScript SDK
- AWS PHP V3 用SDK
- AWS Python 用 SDK
- AWS ルビー V3 用 SDK

データ型

Amazon Lex モデル構築サービスでは、以下のデータ型がサポートされています。

- BotAliasMetadata
- BotChannelAssociation
- BotMetadata
- BuiltinIntentMetadata
- BuiltinIntentSlot
- BuiltinSlotTypeMetadata
- CodeHook
- ConversationLogsRequest
- ConversationLogsResponse
- EnumerationValue
- FollowUpPrompt
- FulfillmentActivity
- InputContext
- Intent
- IntentMetadata
- KendraConfiguration
- LogSettingsRequest
- LogSettingsResponse
- Message
- MigrationAlert
- MigrationSummary

- OutputContext
- Prompt
- ResourceReference
- Slot
- SlotDefaultValue
- SlotDefaultValueSpec
- SlotTypeConfiguration
- SlotTypeMetadata
- SlotTypeRegexConfiguration
- Statement
- Tag
- UtteranceData
- UtteranceList

以下のデータタイプが Amazon Lex ランタイム でサポートされています。

- ActiveContext
- ActiveContextTimeToLive
- Button
- DialogAction
- GenericAttachment
- IntentConfidence
- IntentSummary
- PredictedIntent
- ResponseCard
- SentimentResponse

Amazon Lex Model Building Service

Amazon Lex モデル構築サービスでは、以下のデータ型がサポートされています。

- BotAliasMetadata

- BotChannelAssociation
- BotMetadata
- BuiltinIntentMetadata
- BuiltinIntentSlot
- BuiltinSlotTypeMetadata
- CodeHook
- ConversationLogsRequest
- ConversationLogsResponse
- EnumerationValue
- FollowUpPrompt
- FulfillmentActivity
- InputContext
- Intent
- IntentMetadata
- KendraConfiguration
- LogSettingsRequest
- LogSettingsResponse
- Message
- MigrationAlert
- MigrationSummary
- OutputContext
- Prompt
- ResourceReference
- Slot
- SlotDefaultValue
- SlotDefaultValueSpec
- SlotTypeConfiguration
- SlotTypeMetadata
- SlotTypeRegexConfiguration
- Statement

- Tag
- UtteranceData
- UtteranceList

BotAliasMetadata

サービス: Amazon Lex Model Building Service

ボットエイリアスに関する情報を提供します。

コンテンツ

botName

エイリアスが指すボットの名前。

型: 文字列

長さの制限: 最小長は 2 です。最大長は 50 です。

Pattern: `^([A-Za-z]_?)+$`

必須: いいえ

botVersion

エイリアスが指す Amazon Lex ボットのバージョン。

型: 文字列

長さの制限：最小長は 1 です。最大長は 64 文字です。

パターン: `\$LATEST|[0-9]+`

必須: いいえ

checksum

ボットエイリアスのチェックサム。

タイプ: 文字列

必須: いいえ

conversationLogs

Amazon Lex がエイリアスの会話ログをどのように使用するかを決定する設定。

型: ConversationLogsResponse オブジェクト

必須: いいえ

createdDate

ボットエイリアスが作成された日付。

型: タイムスタンプ

必須: いいえ

description

ボットエイリアスの説明。

型: 文字列

長さの制限: 最小長は 0 です。最大長は 200 です。

必須: いいえ

lastUpdatedDate

ボットエイリアスが更新された日付。リソースを作成すると、作成日と最終更新日は同じ日付になります。

型: タイムスタンプ

必須: いいえ

name

ボットエイリアスの名前。

型: 文字列

長さの制限: 最小長は 1 です。最大長は 100 です。

Pattern: `^([A-Za-z]_?)+$`

必須: いいえ

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS Java V2 用 SDK

- AWS ルビー V3 用 SDK

BotChannelAssociation

サービス: Amazon Lex Model Building Service

Amazon Lex ボットと外部メッセージングプラットフォーム間の関連付けを表します。

コンテンツ

botAlias

この関連付けが作成されている Amazon Lex ボットの特定のバージョンを指すエイリアス。

型: 文字列

長さの制限: 最小長は 1 です。最大長は 100 です。

Pattern: `^([A-Za-z]_?)+$`

必須: いいえ

botConfiguration

メッセージングプラットフォームとの通信に必要な情報を提供します。

型: 文字列間のマッピング

マップエントリ: 10 の項目の最大数。

必須: いいえ

botName

このアソシエーションが作成されている Amazon Lex ボットの名前。

> **Note**
>
> 現在、Amazon Lex は Facebook と Slack、および Twilio との関連付けをサポートしています。

型: 文字列

長さの制限: 最小長は 2 です。最大長は 50 です。

Pattern: `^([A-Za-z]_?)+$`

必須: いいえ

createdDate

Amazon Lex ボットとチャンネルの関連付けが作成された日付。

型: タイムスタンプ

必須: いいえ

description

作成している関連付けの説明テキスト。

型: 文字列

長さの制限: 最小長は 0 です。最大長は 200 です。

必須: いいえ

failureReason

`status` が `FAILED` の場合、Amazon Lex は関連性の作成に失敗した理由を提示します。

タイプ: 文字列

必須: いいえ

name

ボットとチャンネル間の関連付けの名前。

型: 文字列

長さの制限: 最小長は 1 です。最大長は 100 です。

Pattern: `^([A-Za-z]_?)+$`

必須: いいえ

status

ボットチャンネルのステータス。

- `CREATED` - チャンネルが作成され、使用可能な状態です。
- `IN_PROGRESS` - チャンネルの作成中。

- FAILED - チャンネルの作成中にエラーが発生しました。失敗の原因については、「failureReason」フィールドを参照してください。

型: 文字列

有効な値：IN_PROGRESS | CREATED | FAILED

必須：いいえ

type

Amazon Lex ボットと外部のメッセージングプラットフォームとの間に確立されるチャンネルのタイプにより、関連付けのタイプを指定します。

型: 文字列

有効な値：Facebook | Slack | Twilio-Sms | Kik

必須：いいえ

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS Java V2 用 SDK
- AWS ルビー V3 用 SDK

BotMetadata

サービス: Amazon Lex Model Building Service

ボットに関する情報を提供します。

コンテンツ

createdDate

ボットが作成された日付。

型: タイムスタンプ

必須: いいえ

description

ボットの説明。

型: 文字列

長さの制限: 最小長は 0 です。最大長は 200 です。

必須: いいえ

lastUpdatedDate

ボットが更新された日付。ボットを作成すると、作成日と最終更新日は同じ日付になります。

型: タイムスタンプ

必須: いいえ

name

ボットの名前。

型: 文字列

長さの制限: 最小長は 2 です。最大長は 50 です。

Pattern: `^([A-Za-z]_?)+$`

必須: いいえ

status

ボットのステータス。

型: 文字列

有効な値 : BUILDING | READY | READY_BASIC_TESTING | FAILED | NOT_BUILT

必須 : いいえ

version

ボットのバージョン。新規のボットの場合、バージョンは常に $LATEST です。

型: 文字列

長さの制限 : 最小長は 1 です。最大長は 64 文字です。

パターン: \$LATEST|[0-9]+

必須: いいえ

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS Java V2 用 SDK
- AWS ルビー V3 用 SDK

BuiltinIntentMetadata

サービス: Amazon Lex Model Building Service

組み込みインテント用のメタデータを提供します。

コンテンツ

signature

組み込みインテントの一意の識別子。インテントの署名を見つけるには、「Alexa Skills Kit」の「Standard Built-in Intents」(標準の組み込みインテント) を参照してください。

タイプ: 文字列

必須: いいえ

supportedLocales

インテントがサポートするロケールの識別子のリスト。

タイプ：文字列の配列

有効な値: `de-DE | en-AU | en-GB | en-IN | en-US | es-419 | es-ES | es-US | fr-FR | fr-CA | it-IT | ja-JP | ko-KR`

必須：いいえ

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS Java V2 用 SDK
- AWS ルビー V3 用 SDK

BuiltinIntentSlot

サービス: Amazon Lex Model Building Service

組み込みインテントで使用されるスロットに関する情報を提供します。

コンテンツ

name

インテントに定義されたスロットのリスト。

タイプ: 文字列

必須: いいえ

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS Java V2 用 SDK
- AWS ルビー V3 用 SDK

BuiltinSlotTypeMetadata

サービス: Amazon Lex Model Building Service

組み込みスロットタイプに関する情報を提供します。

コンテンツ

signature

組み込みスロットタイプの一意の識別子。スロットタイプの署名を検索するには、「Alexa Skills Kit」の「Slot Type Reference」(スロットタイプのリファレンス) を参照してください。

タイプ: 文字列

必須: いいえ

supportedLocales

スロットのターゲットロケールのリスト。

タイプ：文字列の配列

有効な値: `de-DE | en-AU | en-GB | en-IN | en-US | es-419 | es-ES | es-US | fr-FR | fr-CA | it-IT | ja-JP | ko-KR`

必須：いいえ

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS Java V2 用 SDK
- AWS ルビー V3 用 SDK

CodeHook

サービス: Amazon Lex Model Building Service

ボットへのリクエストを検証する、またはボットに対するユーザーのリクエストを達成する Lambda 関数を指定します。

コンテンツ

messageVersion

Amazon Lex が Lambda 関数を呼び出すために使用するリクエスト/レスポンスのバージョン。詳細については、「Lambda 関数を使用する」を参照してください。

タイプ: 文字列

長さの制限：最小長は 1 です。最大長は 5 です。

必須: はい

uri

Lambda 関数の Amazon リソースネーム (ARN)。

型: 文字列

長さの制限: 最小長は 20 です。最大長は 2,048 です。

Pattern: `arn:aws[a-zA-Z-]*:lambda:[a-z]+-[a-z]+(-[a-z]+)*-[0-9]:[0-9]{12}:function:[a-zA-Z0-9-_]+(\/[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12})?(:[a-zA-Z0-9-_]+)?`

必須：はい

以下の資料も参照してください。

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS Java V2 用 SDK
- AWS ルビー V3 用 SDK

ConversationLogsRequest

サービス: Amazon Lex Model Building Service

会話ログに必要な設定を提供します。

コンテンツ

iamRoleArn

テキストログの場合はログに、 CloudWatch 音声ログの場合はS3バケットに書き込む権限を持つIAMロールのAmazonリソースネーム (ARN) 。音声暗号化が有効な場合、このロールは、音声ログの暗号化に使用される AWS KMS キーへのアクセス許可も提供します。詳細については、「Creating an IAM Role and Policy for Conversation Logs」(会話ログ用の IAM ロールとポリシーの作成) を参照してください。

型: 文字列

長さの制限: 最小長は 20 です。最大長は 2,048 です。

Pattern: `^arn:[\w\-]+:iam::[\d]{12}:role/.+$`

必須：はい

logSettings

会話ログの設定。会話テキスト、会話オーディオ、またはその両方をログに記録できます。

型: LogSettingsRequest オブジェクトの配列

必須：はい

以下の資料も参照してください。

言語固有の AWS SDK の 1 つでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS Java V2 用 SDK
- AWS ルビー V3 用 SDK

ConversationLogsResponse

サービス: Amazon Lex Model Building Service

会話ログの設定に関する情報が含まれます。

コンテンツ

iamRoleArn

ログまたは S3 バケットにログを書き込むために使用される IAM ロールの Amazon リソースネーム (ARN)。 CloudWatch

型: 文字列

長さの制限: 最小長は 20 です。最大長は 2,048 です。

パターン: `^arn:[\w\-]+:iam::[\d]{12}:role/.+$`

必須: いいえ

logSettings

会話ログの設定。テキスト、オーディオ、またはその両方を口グに記録できます。

型: LogSettingsResponse オブジェクトの配列

必須：いいえ

その他の参照資料

言語固有の AWS SDK の 1 つでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS Java V2 用 SDK
- AWS ルビー V3 用 SDK

EnumerationValue

サービス: Amazon Lex Model Building Service

各スロットタイプは、一連の値を持つことができます。各列挙値は、スロットタイプが取得できる値を表します。

例えば、ピザを注文するボットでは、ピザのクラストの種類を指定するスロットタイプがあります。スロットタイプには、値を含めることができます。

- thick
- thin
- stuffed

コンテンツ

value

スロットタイプの値。

型: 文字列

長さの制限: 最小長は 1 です。最大長は 140 です。

必須: はい

synonyms

スロットタイプの値に関連する追加値。

型: 文字列の配列

長さの制限: 最小長は 1 です。最大長は 140 です。

必須：いいえ

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS Java V2 用 SDK

- AWS ルビー V3 用 SDK

FollowUpPrompt

サービス: Amazon Lex Model Building Service

インテントが達成した後に、追加のアクティビティを求めるプロンプト。例えば、`OrderPizza` のインテントが達成された後、ユーザーが飲み物を注文したいかどうかを確認するためのプロンプトを表示することができます。

コンテンツ

prompt

ユーザーからの情報の入力を促します。

型: Prompt オブジェクト

必須: はい

rejectionStatement

`prompt` フィールドで定義された質問にユーザーが「いいえ」と答えた場合、Amazon Lex はこのステートメントで、インテントがキャンセルされたことを確認します。

型: Statement オブジェクト

必須：はい

以下の資料も参照してください。

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS Java V2 用 SDK
- AWS ルビー V3 用 SDK

FulfillmentActivity

サービス: Amazon Lex Model Building Service

インテントに必要なすべての情報がユーザーから提供された後、インテントがどのように実行されるかを説明します。Lambda 関数を提供するか、インテント情報をクライアントアプリケーションに返すようにインテントを処理できます。関連するロジックがクラウド上に存在するように Lambda 関数を使用し、クライアント側のコードを主にプレゼンテーション層に限定することを推奨します。ロジックを更新する必要がある場合は、Lambda 関数を更新するだけで、クライアントアプリケーションをアップグレードする必要はありません。

次の例を考えます。

- ピザを注文するアプリケーションでは、ユーザーが注文するためのすべての情報を提供した後に、Lambda 関数を使用してピザ屋に注文をします。
- ゲームアプリケーションでは、ユーザーが「石を拾って」と言うと、その情報がクライアントアプリケーションに戻り、オペレーションを実行してグラフィックを更新する必要があります。この場合、Amazon Lex からインテントデータがクライアントに返されます。

コンテンツ

type

インテントをどのように達成するかは、Lambda 関数を実行するか、スロットデータをクライアントアプリケーションに返すかのいずれかです。

型: 文字列

有効な値 : `ReturnIntent | CodeHook`

必須: はい

codeHook

インテントを実行するために使用される Lambda 関数の説明。

型: CodeHook オブジェクト

必須: いいえ

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS Java V2 用 SDK
- AWS ルビー V3 用 SDK

InputContext

サービス: Amazon Lex Model Building Service

Amazon Lex によってインテントを選択するためにアクティブでなければならないコンテキストの名前。

コンテンツ

name

コンテキストの名前。

型: 文字列

長さの制限: 最小長は 1 です。最大長は 100 です。

パターン: `^([A-Za-z]_?)+$`

必須：はい

以下の資料も参照してください。

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS Java V2 用 SDK
- AWS ルビー V3 用 SDK

Intent

サービス: Amazon Lex Model Building Service

インテントの特定バージョンを識別します。

コンテンツ

intentName

インテントの名前。

型: 文字列

長さの制限: 最小長は 1 です。最大長は 100 です。

パターン: `^([A-Za-z]_?)+$`

必須：はい

intentVersion

インテントのバージョン。

型: 文字列

長さの制限：最小長は 1 です。最大長は 64 文字です。

パターン: `\$LATEST|[0-9]+`

必須：はい

以下の資料も参照してください。

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS Java V2 用 SDK
- AWS ルビー V3 用 SDK

IntentMetadata

サービス: Amazon Lex Model Building Service

インテントに関する情報を提供します。

コンテンツ

createdDate

インテントが作成された日付。

型: タイムスタンプ

必須: いいえ

description

インテントの説明。

型: 文字列

長さの制限: 最小長は 0 です。最大長は 200 です。

必須: いいえ

lastUpdatedDate

インテントが更新された日付。インテントを作成すると、作成日と最終更新日は同じ日付になります。

型: タイムスタンプ

必須: いいえ

name

インテントの名前。

型: 文字列

長さの制限: 最小長は 1 です。最大長は 100 です。

Pattern: ^([A-Za-z]_?)+$

必須: いいえ

version

インテントのバージョン。

型: 文字列

長さの制限：最小長は 1 です。最大長は 64 文字です。

パターン: \$LATEST|[0-9]+

必須: いいえ

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS Java V2 用 SDK
- AWS ルビー V3 用 SDK

KendraConfiguration

サービス: Amazon Lex Model Building Service

AMAZON の設定情報を提供します。 KendraSearchIntent意図。このインテントを使用すると、Amazon Lex は指定された Amazon Kendra インデックスを検索し、ユーザーの発話に一致するインデックスからドキュメントを返します。詳細については、AMAZON を参照してください。KendraSearchIntent。

コンテンツ

kendraIndex

Amazon にしたいAmazon Kendra インデックスのアマゾンリソースネーム (ARN)。KendraSearchIntent 検索する意図。バケットは Amazon Lex V2 ボットと同じリージョンにあることが必要です。Amazon Kendra のインデックスが存在しない場合、PutIntent オペレーションを呼び出すと例外が発生します。

型: 文字列

長さの制限: 最小長は 20 です。最大長は 2,048 です。

Pattern: `arn:aws:kendra:[a-z]+-[a-z]+-[0-9]:[0-9]{12}:index\/[a-zA-Z0-9][a-zA-Z0-9_-]*`

必須：はい

role

Amazon Kendra のインデックスを検索する権限を持つ IAM ロールの Amazon リソースネーム (ARN) です。ロールは、Amazon Lex botと同じアカウントとリージョンである必要があります。ロールが存在しない場合、PutIntent オペレーションを呼び出すと例外が発生します。

型: 文字列

長さの制限: 最小長は 20 です。最大長は 2,048 です。

Pattern: `arn:aws:iam::[0-9]{12}:role/.*`

必須：はい

queryFilterString

Amazon Lex がクエリからの応答をフィルタリングするために Amazon Kendra に送信するクエリフィルター。フィルターは Amazon Kendra によって定義された形式です。詳細については、「Filtering queries」(クエリのフィルター) を参照してください。

このフィルター文字列は、実行時に新しいフィルター文字列で上書きできます。

型: 文字列

長さの制限: 最小長は 0 です。

必須：いいえ

その他の参照資料

言語固有の AWS SDK の 1 つでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS Java V2 用 SDK
- AWS ルビー V3 用 SDK

LogSettingsRequest

サービス: Amazon Lex Model Building Service

会話ログの配信モードと送信先を構成するために使用される設定。

コンテンツ

destination

ログが配信される場所。 CloudWatch テキストログはログロググループに配信されます。オーディオログは S3 バケットに配信されます。

型: 文字列

有効な値：`CLOUDWATCH_LOGS | S3`

必須: はい

logType

有効にするログタイプ。 CloudWatchテキストログはログロググループに配信されます。オーディオログは S3 バケットに配信されます。

型: 文字列

有効な値：`AUDIO | TEXT`

必須: はい

resourceArn

CloudWatch ログを配信するロググループまたは S3 バケットの Amazon リソースネーム (ARN)。

型: 文字列

長さの制限：最小長は 1 です。最大長は 2,048 です。

Pattern: `^arn:[\w\-]+:(?:logs:[\w\-]+:[\d]{12}:log-group:[\.\-_/#A-Za-z0-9]{1,512}(?::\*)?|s3:::[a-z0-9][\.\-a-z0-9]{1,61}[a-z0-9])$`

必須：はい

kmsKeyArn

S3 バケットに配信されるオーディオログを暗号化するための AWS KMS カスタマーマネージドキーの Amazon リソースネーム (ARN) です。 CloudWatch このキーはログには適用されず、S3 バケットでは省略できます。

型: 文字列

長さの制限: 最小長は 20 です。最大長は 2,048 です。

パターン: `^arn:[\w\-]+:kms:[\w\-]+:[\d]{12}:(?:key\/[\w\-]+|alias\/[a-zA-Z0-9:\/_\-]{1,256})$`

必須: いいえ

その他の参照資料

言語固有の AWS SDK の 1 つでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS Java V2 用 SDK
- AWS ルビー V3 用 SDK

LogSettingsResponse

サービス: Amazon Lex Model Building Service

会話ログの設定

コンテンツ

destination

ログが配信される送信先。

型: 文字列

有効な値：CLOUDWATCH_LOGS | S3

必須：いいえ

kmsKeyArn

S3 バケット内のオーディオログの暗号化に使用するキーの Amazon リソースネーム (ARN)。

型: 文字列

長さの制限: 最小長は 20 です。最大長は 2,048 です。

パターン: ^arn:[\w\-]+:kms:[\w\-]+:[\d]{12}:(?:key\/[\w\-]+|alias\/[a-zA-Z0-9:\/_\-]{1,256})$

必須: いいえ

logType

有効になっているログのタイプ。

型: 文字列

有効な値：AUDIO | TEXT

必須：いいえ

resourceArn

CloudWatch ログが配信されるロググループまたは S3 バケットの Amazon リソースネーム (ARN)。

型: 文字列

長さの制限：最小長は 1 です。最大長は 2,048 です。

パターン: `^arn:[\w\-]+:(?:logs:[\w\-]+:[\d]{12}:log-group:[\.\-_/#A-Za-z0-9]{1,512}(?::\*)?|s3:::[a-z0-9][\.\-a-z0-9]{1,61}[a-z0-9])$`

必須: いいえ

resourcePrefix

リソースプレフィックスは、オーディオログを格納するために指定した S3 バケット内の S3 オブジェクトキーの最初の部分です。 CloudWatch Logs の場合は、指定したロググループ内のログストリーム名のプレフィックスです。

型: 文字列

長さの制限: 最大長は 1024 です。

必須：いいえ

その他の参照資料

言語固有の AWS SDK の 1 つでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS Java V2 用 SDK
- AWS ルビー V3 用 SDK

Message

サービス: Amazon Lex Model Building Service

メッセージテキストとそのタイプを提供するオブジェクト。

コンテンツ

content

メッセージのテキスト。

型: 文字列

長さの制限：最小長は 1 です。最大長は 1,000 です。

必須: はい

contentType

メッセージ文字列のコンテンツタイプ。

型: 文字列

有効な値：`PlainText | SSML | CustomPayload`

必須: はい

groupNumber

メッセージが属しているメッセージグループを識別します。メッセージがグループに割り当てられている場合、Amazon Lex は各グループから 1 つのメッセージをレスポンスに返します。

型: 整数

有効範囲: 最小値 は 1 です。最大値は 5 です。

必須：いいえ

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS Java V2 用 SDK

- AWS ルビー V3 用 SDK

MigrationAlert

サービス: Amazon Lex Model Building Service

Amazon Lex が移行中に送信するアラートと警告に関する情報を提供します。このアラートには、問題を解決するための情報が含まれています。

コンテンツ

details

アラートに関するその他の詳細

型: 文字列の配列

必須: いいえ

message

アラートが発生された理由を説明するメッセージ。

タイプ: 文字列

必須: いいえ

referenceURLs

アラートを解決方法を説明している Amazon Lex ドキュメントへのリンク。

型: 文字列の配列

必須: いいえ

type

アラートのタイプ。アラートは 2 種類あります。

- ERROR - 移行で解決できない問題がありました。移行が停止します。
- WARN - 移行の際、新しい Amazon Lex V2 ボットに手動で変更する必要があるという問題がありました。移行は継続されます。

型: 文字列

有効な値：ERROR | WARN

必須：いいえ

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS Java V2 用 SDK
- AWS ルビー V3 用 SDK

MigrationSummary

サービス: Amazon Lex Model Building Service

Amazon Lex V1 から Amazon Lex V2 へのボットの移行についての情報を提供します。

コンテンツ

migrationId

Amazon Lex が移行に割り当てた一意の識別子。

型: 文字列

長さの制限: 固定長は 10 です。

Pattern: ^[0-9a-zA-Z]+$

必須: いいえ

migrationStatus

オペレーションのステータス。ステータスが COMPLETE の場合、ボットは Amazon Lex V2 で利用可能です。移行を完了するには、アラートや警告を解決する必要がある場合があります。

型: 文字列

有効な値：IN_PROGRESS | COMPLETED | FAILED

必須：いいえ

migrationStrategy

移行を実行するために使用された戦略。

型: 文字列

有効な値：CREATE_NEW | UPDATE_EXISTING

必須：いいえ

migrationTimestamp

移行が開始された日時。

型: タイムスタンプ

必須: いいえ

v1BotLocale

移行のソースである Amazon Lex V1 ボットの口ケール。

型: 文字列

有効な値 : de-DE | en-AU | en-GB | en-IN | en-US | es-419 | es-ES | es-US | fr-FR | fr-CA | it-IT | ja-JP | ko-KR

必須 : いいえ

v1BotName

移行のソースである Amazon Lex V1 ボットの名前。

型: 文字列

長さの制限: 最小長は 2 です。最大長は 50 です。

Pattern: ^([A-Za-z]_?)+$

必須: いいえ

v1BotVersion

移行のソースである Amazon Lex V1 ボットのバージョン。

型: 文字列

長さの制限 : 最小長は 1 です。最大長は 64 文字です。

パターン: \$LATEST|[0-9]+

必須: いいえ

v2BotId

移行先となる Amazon Lex V2 の一意の識別子。

型: 文字列

長さの制限: 固定長は 10 です。

Pattern: ^[0-9a-zA-Z]+$

必須: いいえ

v2BotRole

Amazon Lex が Amazon Lex V2 ボットの実行に使用する IAM ロール。

型: 文字列

長さの制限: 最小長は 20 です。最大長は 2,048 です。

パターン: `^arn:[\w\-]+:iam::[\d]{12}:role/.+$`

必須: いいえ

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS Java V2 用 SDK
- AWS ルビー V3 用 SDK

OutputContext

サービス: Amazon Lex Model Building Service

インテントが達成されたときに設定される出力コンテキストの指定。

コンテンツ

name

コンテキストの名前。

型: 文字列

長さの制限: 最小長は 1 です。最大長は 100 です。

パターン: ^([A-Za-z]_?)+$

必須：はい

timeToLiveInSeconds

PostContent または PostText レスポンスで最初に送信された後、コンテキストをアクティブにしておく秒数です。5～86,400 秒 (24 時間) の値を設定できます。

型: 整数

値の範囲: 最小値 は 5 です。最大値は 86400 です。

必須: はい

turnsToLive

コンテキストがアクティブになるべき会話のターン数。会話の 1 ターンは、1 つの PostContent または PostText のリクエストと、それに対応する Amazon Lex のレスポンスです。

型: 整数

有効範囲: 最小値 は 1 です。最大値は 20 です。

必須：はい

以下の資料も参照してください。

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS Java V2 用 SDK
- AWS ルビー V3 用 SDK

Prompt

サービス: Amazon Lex Model Building Service

ユーザーから情報を取得します。プロンプトを定義するには、1 つ以上のメッセージを指定し、ユーザーに情報を求める試行回数を指定します。複数のメッセージを指定した場合、Amazon Lex はユーザーへのプロンプトに使用するメッセージの 1 つを選択します。詳細については、「Amazon Lex: 仕組み」を参照してください。

コンテンツ

maxAttempts

ユーザーに情報の入力を求める回数。

型: 整数

有効範囲: 最小値 は 1 です。最大値は 5 です。

必須: はい

messages

オブジェクトの配列です。各オブジェクトはメッセージ文字列とそのタイプがあります。メッセージ文字列は、プレーンテキストまたは音声合成マークアップ言語 (SSML) で指定できます。

型: Message オブジェクトの配列

配列メンバー：最小数は 1 項目です。最大数は 15 項目です。

必須: はい

responseCard

レスポンスカード。Amazon Lex は、実行時に、PostText API レスポンスでこのプロンプトを使用します。レスポンスカードのプレースホルダーの代わりに、セッション属性やスロットの値が使われます。詳細については、「レスポンスカードの使用」を参照してください。

タイプ: 文字列

長さの制限：最小長は 1 です。最大長は 50000 です。

必須：いいえ

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS Java V2 用 SDK
- AWS ルビー V3 用 SDK

ResourceReference

サービス: Amazon Lex Model Building Service

削除するリソースを参照しているリソースについて説明します。このオブジェクトは、ResourceInUseException の例外の一部として返されます。

コンテンツ

name

削除するリソースを使用しているリソースの名前です。

型: 文字列

長さの制限: 最小長は 1 です。最大長は 100 です。

Pattern: [a-zA-Z_]+

必須: いいえ

version

削除するリソースを使用しているリソースのバージョンです。

型: 文字列

長さの制限：最小長は 1 です。最大長は 64 文字です。

パターン: \$LATEST|[0-9]+

必須: いいえ

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS Java V2 用 SDK
- AWS ルビー V3 用 SDK

Slot

サービス: Amazon Lex Model Building Service

特定のスロットのバージョンを示します。

コンテンツツ

name

スロットの名前。

型: 文字列

長さの制限: 最小長は 1 です。最大長は 100 です。

パターン: `^([A-Za-z](-|_|.)?)+$`

必須：はい

slotConstraint

スロットが必須であるかオプションであるかを指定します。

型: 文字列

有効な値：`Required | Optional`

必須: はい

defaultValueSpec

スロットのデフォルト値のリスト。デフォルト値は、Amazon Lex がスロットの値を決定していない場合に使用されます。コンテキスト変数、セッション属性、定義値からデフォルト値を指定できます。

型: SlotDefaultValueSpec オブジェクト

必須: いいえ

description

スロットの説明。

型: 文字列

長さの制限: 最小長は 0 です。最大長は 200 です。

必須: いいえ

obfuscationSetting

会話ログや保存された発話の中で、スロットが難読化されているかどうかを判定します。スロットを難読化すると、その値は中括弧 ({}) で囲まれたスロット名に置き換えられます。例えば、スロット名が「full_name」の場合、難読化された値は「{full_name}」と置き換えられます。詳細については、「Slot Obfuscation」(スロットの難読化) を参照してください。

型: 文字列

有効な値：NONE | DEFAULT_OBFUSCATION

必須：いいえ

priority

Amazon Lex に対して、ユーザーからこのスロット値を引き出す順序を指示します。例えば、インテントに優先度 1 と 2 の 2 つのスロットがある場合、AWS Amazon Lex はまず優先度 1 のスロットの値を引き出します。

複数のスロットが同じ優先順位を持つ場合、Amazon Lex が値を引き出す順序は任意です。

型: 整数

有効な範囲: 最小値 は 0 です。最大値は 100 です。

必須: いいえ

responseCard

テキストベースのクライアントで使用されるスロットタイプの想定されるレスポンスのセット。ユーザーは、テキストを使用して返信するのではなく、レスポンスカードからオプションを選択します。

型: 文字列

長さの制限：最小長は 1 です。最大長は 50000 です。

必須: いいえ

sampleUtterances

スロット値に対する Amazon Lex リクエストにユーザーが応答する可能性のある特定のパターンがわかっている場合は、これらの発話を提供して精度を向上させることができます。これはオプションです。ほとんどの場合、Amazon Lex はユーザーの発話を理解できます。

型: 文字列の配列

の配列メンバー: 最小数は 0 項目です。最大数は 10 項目です。

長さの制限：最小長は 1 です。最大長は 200 です。

必須: いいえ

slotType

スロットのタイプは、定義したカスタムスロットタイプ、または組み込みスロットタイプのいずれかです。

型: 文字列

長さの制限: 最小長は 1 です。最大長は 100 です。

Pattern: `^((AMAZON\.)_?|[A-Za-z]_?)+`

必須: いいえ

slotTypeVersion

スロットタイプのバージョン。

型: 文字列

長さの制限：最小長は 1 です。最大長は 64 文字です。

パターン: `\$LATEST|[0-9]+`

必須: いいえ

valueElicitationPrompt

Amazon Lex がユーザーからスロット値を引き出すために使用するプロンプト。

型: Prompt オブジェクト

必須: いいえ

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS Java V2 用 SDK
- AWS ルビー V3 用 SDK

SlotDefaultValue

サービス: Amazon Lex Model Building Service

スロットのデフォルト値のリスト。

コンテンツ

defaultValue

スロットのデフォルト値。以下のいずれかを指定できます。

- `#context-name.slot-name` - コンテキスト「context-name」内のスロット値「slot-name」です。
- `{attribute}` - セッション属性「attribute」のスロット値です。
- `'value'` - 離散値 「value」。

型: 文字列

長さの制限：最小長は 1 です。最大長は 202 です。

必須：はい

以下の資料も参照してください。

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS Java V2 用 SDK
- AWS ルビー V3 用 SDK

SlotDefaultValueSpec

サービス: Amazon Lex Model Building Service

スロットのデフォルト値が含まれます。デフォルト値は、Amazon Lex がスロットの値を決定していない場合に使用されます。

コンテンツ

defaultValueList

スロットのデフォルト値。複数のデフォルトを指定することができます。例えば、一致するコンテキスト変数、セッション属性、固定値から使用するデフォルト値を指定できます。

選ばれたデフォルト値は、リストで指定した順序に基づいて選択されます。例えば、その注文でコンテキスト変数と固定値を指定した場合、Amazon Lex はコンテキスト変数があればそれを使用し、なければ固定値を使用します。

型: SlotDefaultValue オブジェクトの配列

の配列メンバー: 最小数は 0 項目です。最大数は 10 項目です。

必須：はい

以下の資料も参照してください。

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS Java V2 用 SDK
- AWS ルビー V3 用 SDK

SlotTypeConfiguration

サービス: Amazon Lex Model Building Service

スロットタイプの設定情報を提供します。

コンテンツ

regexConfiguration

スロット値を検証するために使用される正規表現。

型: SlotTypeRegexConfiguration オブジェクト

必須: いいえ

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS Java V2 用 SDK
- AWS ルビー V3 用 SDK

SlotTypeMetadata

サービス: Amazon Lex Model Building Service

スロットタイプの情報を提供します。

コンテンツ

createdDate

スロットタイプが作成された日付。

型: タイムスタンプ

必須: いいえ

description

スロットタイプの説明。

型: 文字列

長さの制限: 最小長は 0 です。最大長は 200 です。

必須: いいえ

lastUpdatedDate

スロットタイプが更新された日付。リソースを作成すると、作成日と最終更新日は同じ日付になります。

型: タイムスタンプ

必須: いいえ

name

スロットタイプの名前。

型: 文字列

長さの制限: 最小長は 1 です。最大長は 100 です。

Pattern: `^([A-Za-z]_?)+$`

必須: いいえ

version

スロットタイプのバージョン。

型: 文字列

長さの制限：最小長は 1 です。最大長は 64 文字です。

パターン: \$LATEST|[0-9]+

必須: いいえ

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS Java V2 用 SDK
- AWS ルビー V3 用 SDK

SlotTypeRegexConfiguration

サービス: Amazon Lex Model Building Service

スロット値を検証するために使用される正規表現を提供します。

コンテンツ

pattern

スロット値を検証するために使用される正規表現。

標準の正規表現を使用します。Amazon Lex は、正規表現で以下の文字をサポートします。

- A ~ Z、a ~ z
- 0-9
- Unicode文字 (「\ u<Unicode>」)

「\u0041」や 「\u005A」など、4 桁の数字で Unicode 文字を表します。

次の正規表現演算子はサポートされていません。

- 無限リピーター: *、+、または 上限のない {x,}
- ワイルドカード (.)

型: 文字列

長さの制限: 最小長は 1 です。最大長は 100 です。

必須：はい

以下の資料も参照してください。

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS Java V2 用 SDK
- AWS ルビー V3 用 SDK

Statement

サービス: Amazon Lex Model Building Service

ユーザーに情報を伝えるためのメッセージのコレクション。実行時に、Amazon Lex は伝えるべきメッセージを選択します。

コンテンツ

messages

メッセージオブジェクトのコレクション。

型: Message オブジェクトの配列

配列メンバー：最小数は 1 項目です。最大数は 15 項目です。

必須: はい

responseCard

実行時に、クライアントが PostTextAPI を使用している場合、Amazon Lex はレスポンスカードをレスポンスに含めます。レスポンスカードのプレースホルダーに、すべてのセッション属性とスロットの値を置き換えます。

型: 文字列

長さの制限：最小長は 1 です。最大長は 50000 です。

必須：いいえ

その他の参照資料

この API を言語固有の AWS SDK で使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS Java V2 用 SDK
- AWS ルビー V3 用 SDK

Tag

サービス: Amazon Lex Model Building Service

ボット、ボットエイリアス、またはボットチャンネルを識別するためのキーバリューのペアのリスト。タグキーと値に使用できる文字は、Unicode 文字、数字、空白、および _ . : / = + - @ の記号です。

コンテンツ

key

タグのキー。キーは大文字と小文字を区別せず、一意でなければなりません。

型: 文字列

長さの制限: 最小長は 1 です。最大長は 128 です。

必須: はい

value

キーに関連付ける値。値は空の文字列でも構いませんが、null にはできません。

型: 文字列

長さの制限: 最小長は 0 です。最大長は 256 です。

必須：はい

以下の資料も参照してください。

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS Java V2 用 SDK
- AWS ルビー V3 用 SDK

UtteranceData

サービス: Amazon Lex Model Building Service

ボットに発せられた単一の発話に関する情報を提供します。

コンテンツ

count

発話が処理された回数。

型: 整数

必須: いいえ

distinctUsers

その発話を使用した個人の数の合計。

型: 整数

必須: いいえ

firstUtteredDate

発話が最初に記録された日付。

型: タイムスタンプ

必須: いいえ

lastUtteredDate

発話が最後に記録された日付。

型: タイムスタンプ

必須: いいえ

utteranceString

ユーザーが入力したテキスト、またはオーディオクリップのテキスト表現。

型: 文字列

長さの制限：最小長は 1 です。最大長は 2000 です。

必須：いいえ

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS Java V2 用 SDK
- AWS ルビー V3 用 SDK

UtteranceList

サービス: Amazon Lex Model Building Service

ボットの特定のバージョンに対して行われた発話のリストを提供します。リストには最大 100 件の発話が含まれます。

コンテンツ

botVersion

リストを処理したボットのバージョン。

型: 文字列

長さの制限：最小長は 1 です。最大長は 64 文字です。

パターン: `\$LATEST|[0-9]+`

必須: いいえ

utterances

ボットへの発話に関する情報を含む 1 つ以上の UtteranceData オブジェクト。オブジェクトの最大数は 100 です。

型: UtteranceData オブジェクトの配列

必須：いいえ

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS Java V2 用 SDK
- AWS ルビー V3 用 SDK

Amazon Lex Runtime Service

以下のデータタイプが Amazon Lex ランタイム でサポートされています。

- ActiveContext
- ActiveContextTimeToLive
- Button
- DialogAction
- GenericAttachment
- IntentConfidence
- IntentSummary
- PredictedIntent
- ResponseCard
- SentimentResponse

ActiveContext

サービス: Amazon Lex Runtime Service

コンテキストとは、ユーザーと Amazon Lex との会話の現在の状態に関する情報を含む変数のことです。コンテキストは、インテントが達成したときに Amazon Lex によって自動的に設定されるか、または PutContent、PutText、PutSession オペレーションを使用してランタイムに設定することができます。

コンテンツ

name

コンテキストの名前。

型: 文字列

長さの制限: 最小長は 1 です。最大長は 100 です。

パターン: ^([A-Za-z]_?)+$

必須：はい

parameters

現在のコンテキストの状態変数。これらの値は、後続のイベントでスロットのデフォルト値として使用できます。

型: 文字列間のマッピング

マップエントリ 0 の項目の最小数。最大数は 10 項目です。

キーの長さ制限: 最小長さは 1 です。最大長は 100 です。

値の長さの制限: 最小長は 1 です。最大長は 1,024 です。

必須: はい

timeToLive

コンテキストがアクティブな時間の長さまたはターン数。

型: ActiveContextTimeToLive オブジェクト

必須：はい

以下の資料も参照してください。

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS Java V2 用 SDK
- AWS ルビー V3 用 SDK

ActiveContextTimeToLive

サービス: Amazon Lex Runtime Service

コンテキストがアクティブな時間の長さまたはターン数。

コンテンツ

timeToLiveInSeconds

PostContent または PostText レスポンスで最初に送信された後、コンテキストをアクティブにしておく秒数です。5 ~ 86,400 秒 (24 時間) の値を設定できます。

型: 整数

値の範囲: 最小値 は 5 です。最大値は 86400 です。

必須: いいえ

turnsToLive

コンテキストがアクティブになるべき会話のターン数。会話の 1 ターンは、1 つの PostContent または PostText のリクエストと、それに対応する Amazon Lex のレスポンスです。

型: 整数

有効範囲: 最小値 は 1 です。最大値は 20 です。

必須：いいえ

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS Java V2 用 SDK
- AWS ルビー V3 用 SDK

Button

サービス: Amazon Lex Runtime Service

クライアントプラットフォーム (Facebook、Slack など) に表示されるオプションを表します。

コンテンツ

text

ボタン上でユーザーに表示されるテキスト。

型: 文字列

長さの制限：最小長は 1 です。最大長は 15 です。

必須: はい

value

ユーザーがボタンを選択したときに Amazon Lex に送信される値。例えば、ボタンテキストを「NYC」としましょう。ユーザーがボタンをクリックすると、送信される値は「New York City」になります。

型: 文字列

長さの制限：最小長は 1 です。最大長は 1,000 です。

必須：はい

以下の資料も参照してください。

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS Java V2 用 SDK
- AWS ルビー V3 用 SDK

DialogAction

サービス: Amazon Lex Runtime Service

ユーザーとのインタラクションにおいてボットが取るべき次のアクションを記述し、そのアクションが行われるコンテキストに関する情報を提供します。インタラクションを特定の状態に設定したり、以前の状態に戻したりするには、DialogAction データタイプを使用します。

コンテンツ

type

ユーザーとのインタラクションにおいて、ボットが取るべき次のアクション。指定できる値は以下のとおりです。

- ConfirmIntent - 次のアクションは、インテントが完了し、履行する準備ができているかどうかをユーザーに尋ねています。これは「注文しますか？」などの「はい」/「いいえ」の質問です。
- Close - ユーザーからのレスポンスが返されないことを示します。例えば、「ご注文を受け付けました」に対するレスポンスは不要です。
- Delegate - 次のアクションは Amazon Lex によって決定されます。
- ElicitIntent - 次のアクションで、ユーザーが達成したいインテントを決定します。
- ElicitSlot - 次のアクションは、ユーザーからスロット値を引き出すことです。

型: 文字列

有効な値 : ElicitIntent | ConfirmIntent | ElicitSlot | Close | Delegate

必須: はい

fulfillmentState

インテントのフルフィルメント状態。指定できる値は以下のとおりです。

- Failed - インテントに関連付けられた Lambda 関数は、インテントを達成できませんでした。
- Fulfilled - インテントに関連付けられた Lambda 関数は、インテントを達成しました。
- ReadyForFulfillment - インテントに必要なすべての情報が揃っており、クライアントアプリケーションがインテントを達成する準備ができています。

型: 文字列

有効な値 : `Fulfilled | Failed | ReadyForFulfillment`

必須 : いいえ

intentName

インテントの名前。

タイプ: 文字列

必須: いいえ

message

ユーザーに表示されるメッセージ。メッセージを指定しない場合、Amazon Lex はインテントに設定されたメッセージを使用します。

型: 文字列

長さの制限: 最小長は 1 です。最大長は 1,024 です。

必須: いいえ

messageFormat

- `PlainText` - メッセージには UTF-8 形式テキストが含まれています。
- `CustomPayload` - メッセージはクライアント向けのカスタム形式です。
- `SSML` - メッセージには音声出力のテキスト形式が含まれています。
- `Composite` - メッセージには、1 つ以上のメッセージを含むエスケープされた JSON オブジェクトが含まれています。詳細については、「Message Groups」(メッセージグループ) を参照してください。

型: 文字列

有効な値 : `PlainText | CustomPayload | SSML | Composite`

必須 : いいえ

slots

収集されたスロットとその値のマップ。

型: 文字列間のマッピング

必須: いいえ

slotToElicit

ユーザーから引き出されるスロットの名前。

タイプ: 文字列

必須: いいえ

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS Java V2 用 SDK
- AWS ルビー V3 用 SDK

GenericAttachment

サービス: Amazon Lex Runtime Service

プロンプトが表示されたときにユーザーに表示されるオプションを表します。イメージ、ボタン、リンク、またはテキストを使用できます。

コンテンツ

attachmentLinkUrl

レスポンスカードの添付ファイルのURL。

型: 文字列

長さの制限：最小長は 1 です。最大長は 2,048 です。

必須: いいえ

buttons

ユーザーに表示するオプションのリスト。

型: Button オブジェクトの配列

配列メンバー：最小数は 0 項目です。最大数は 5 項目です。

必須: いいえ

imageUrl

ユーザーに表示されるイメージの URL。

型: 文字列

長さの制限：最小長は 1 です。最大長は 2,048 です。

必須: いいえ

subTitle

タイトルの下に表示されるサブタイトル。

型: 文字列

長さの制限：最小長は 1 です。最大長は 80 です。

必須: いいえ

title

オプションのタイトル。

型: 文字列

長さの制限：最小長は 1 です。最大長は 80 です。

必須：いいえ

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS Java V2 用 SDK
- AWS ルビー V3 用 SDK

IntentConfidence

サービス: Amazon Lex Runtime Service

Amazon Lex がインテントがユーザーのインテントを満たすものであるという信頼度を示すスコアを提供します。

コンテンツ

score

インテントがユーザーのインテントを満たしているという Amazon Lex の信頼度を示すスコア。0.00 ~ 1.00 までの範囲になります。スコアが高いほど、信頼度が高くなります。

型: 倍精度浮動小数点数

必須：いいえ

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS Java V2 用 SDK
- AWS ルビー V3 用 SDK

IntentSummary

サービス: Amazon Lex Runtime Service

インテントの状態に関する情報を提供します。この情報を使って、インテントの状態を取得してインテントを処理したり、インテントを以前の状態に戻したりすることができます。

コンテンツ

dialogActionType

ユーザーとのインタラクションにおいて、ボットが取るべき次のアクション。指定できる値は以下のとおりです。

- `ConfirmIntent` - 次のアクションは、インテントが完了し、履行する準備ができているかどうかをユーザーに尋ねています。これは「注文しますか？」などの「はい」/「いいえ」の質問です。
- `Close` - ユーザーからのレスポンスが返されないことを示します。例えば、「ご注文を受け付けました」に対するレスポンスは不要です。
- `ElicitIntent` - 次のアクションで、ユーザーが達成したいインテントを決定します。
- `ElicitSlot` - 次のアクションは、ユーザーからスロット値を引き出すことです。

型: 文字列

有効な値 : `ElicitIntent | ConfirmIntent | ElicitSlot | Close | Delegate`

必須: はい

checkpointLabel

特定のインテントを識別するユーザー定義のラベル。このラベルを使用して、前のインテントに戻ることができます。

`GetSessionRequest` オペレーションの `checkpointLabelFilter` パラメータを使用して、オペレーションによって返されるインテントを、指定されたラベルのみを持つものにフィルタリングします。

型: 文字列

長さの制限：最小長は 1 です。最大長は 255 です。

パターン: `[a-zA-Z0-9-]+`

必須: いいえ

confirmationStatus

ユーザーが確認プロンプトにレスポンスした後のインテントの状態。ユーザーがインテントを確認した場合、Amazon Lex はこのフィールドを Confirmed に設定します。ユーザーがインテントを拒否した場合、Amazon Lex はこの値を Denied に設定します。指定できる値は以下のとおりです。

- Confirmed - ユーザーは、インテントが完了し、実行する準備ができていることを確認したら、確認プロンプトに「はい」と答えます。
- Denied - ユーザーは確認プロンプトで「いいえ」とレスポンスしました。
- None - ユーザーに対して、一度も確認を求めていないか、または確認を求めたが肯定も否定もされなかった場合。

型: 文字列

有効な値：None | Confirmed | Denied

必須：いいえ

fulfillmentState

インテントのフルフィルメント状態。指定できる値は以下のとおりです。

- Failed - インテントに関連付けられた Lambda 関数は、インテントを達成できませんでした。
- Fulfilled - インテントに関連付けられた Lambda 関数は、インテントを達成しました。
- ReadyForFulfillment - インテントに必要なすべての情報が揃っており、クライアントアプリケーションがインテントを達成する準備ができています。

型: 文字列

有効な値：Fulfilled | Failed | ReadyForFulfillment

必須：いいえ

intentName

インテントの名前。

タイプ: 文字列

必須: いいえ

slots

収集されたスロットとその値のマップ。

型: 文字列間のマッピング

必須: いいえ

slotToElicit

ユーザーから引き出すための次のスロット。抽出するスロットがない場合、フィールドは空白です。

タイプ: 文字列

必須: いいえ

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS Java V2 用 SDK
- AWS ルビー V3 用 SDK

PredictedIntent

サービス: Amazon Lex Runtime Service

Amazon Lex が提案する、ユーザーのインテントを達成するインテント。インテントの名前、Amazon Lex が持つユーザーのインテント達成の信頼度、およびインテントに定義されたスロットを含みます。

コンテンツ

intentName

Amazon Lex が提案する、ユーザーのインテントを達成するインテントの名前。

タイプ: 文字列

必須: いいえ

nluIntentConfidence

インテントがユーザーのインテントを達成しているという Amazon Lex の信頼度を示します。

型: IntentConfidence オブジェクト

必須: いいえ

slots

予測インテントに関連付けられたスロットとスロットの値。

型: 文字列間のマッピング

必須：いいえ

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS Java V2 用 SDK
- AWS ルビー V3 用 SDK

ResponseCard

サービス: Amazon Lex Runtime Service

ボット作成時にレスポンスカードを設定した場合、Amazon Lex は利用可能なセッション属性とスロットの値を代入し、それを返します。レスポンスカードは、Lambda 関数 (インテントの `dialogCodeHook` と `fulfillmentActivity`) からもアクセスできます。

コンテンツ

contentType

レスポンスのコンテンツタイプ。

型: 文字列

有効な値：`application/vnd.amazonaws.card.generic`

必須：いいえ

genericAttachments

オプションを表すアタッチメントオブジェクトの配列。

型: GenericAttachment オブジェクトの配列

の配列メンバー: 最小数は 0 項目です。最大数は 10 項目です。

必須: いいえ

version

レスポンスカード形式のバージョン。

タイプ: 文字列

必須: いいえ

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS Java V2 用 SDK

- AWS ルビー V3 用 SDK

SentimentResponse

サービス: Amazon Lex Runtime Service

発話で表現されるセンチメント。

ボットがセンチメント分析のために Amazon Comprehend に発話を送信するように設定されている場合、このフィールド構造には分析の結果が含まれます。

コンテンツ

sentimentLabel

Amazon Comprehend が最も信頼度が高いと推測されるセンチメント。

タイプ: 文字列

必須: いいえ

sentimentScore

センチメントが正しく推測された可能性。

タイプ: 文字列

必須: いいえ

その他の参照資料

言語固有の AWS SDK でこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS Java V2 用 SDK
- AWS ルビー V3 用 SDK

Amazon Lex のドキュメント履歴

- 前回のドキュメント更新日: 2021 年 9 月 9 日

次の表に、Amazon Lex の各リリースにおける重要な変更点を示します。このドキュメントの更新に関する通知については、RSS フィードでサブスクライブできます。

変更	説明	日付
新機能	Amazon Lex が韓国語 (ko-KR) で利用可能になりました。詳細については、「Languages supported by Amazon Lex」(Amazon Lex でサポートされている言語) を参照してください。	2021 年 9 月 9 日
新機能	Amazon Lex が英語 (インド) で利用可能になりました。詳細については、「Languages supported by Amazon Lex」(Amazon Lex でサポートされている言語) を参照してください。	2021 年 7 月 15 日
新機能	Amazon Lex では、ボットを Amazon Lex V2 API に移行するためのツールをご用意しています。詳細については、「Migrating a bot」(ボットの移行) を参照してください。	2021 年 7 月 13 日
新機能	Amazon Lex が日本語 (日本) で利用可能になりました。詳細については、「Languages supported by Amazon	2021 年 4 月 1 日

	Lex」(Amazon Lex でサポートされている言語) を参照してください。	
新機能	Amazon Lex が、ドイツ語 (de-DE) およびスペイン語 (ラテンアメリカ) (es-419) で利用可能になりました。詳細については、「Languages supported by Amazon Lex」(Amazon Lex でサポートされている言語) を参照してください。	2020 年 11 月 23 日
新機能	Amazon Lex では、アクティベートインテントの管理にコンテキストの使用がサポートされるようになりました。詳細については、「Setting Intent Context」を参照してください。	2020 年 11 月 19 日
新機能	Amazon Lex が、フランス語 (fr-FR)、カナダフランス語 (fr-CA)、イタリア語 (it-IT)、スペイン語 (es-ES) で利用可能になりました。詳細については、「Languages supported by Amazon Lex」(Amazon Lex でサポートされている言語) を参照してください。	2020 年 11 月 11 日

新機能	Amazon Lex がスペイン語 (米国) (es-US) で利用可能になりました。詳細については、「Languages supported by Amazon Lex」(Amazon Lex でサポートされている言語) を参照してください。	2020 年 9 月 22 日
新機能	Amazon Lex が、英語 (英国) (en-GB) で利用可能になりました。詳細については、「Languages supported by Amazon Lex」(Amazon Lex でサポートされている言語) を参照してください。	2020 年 9 月 15 日
新機能	Amazon Lex が、英語 (オーストラリア) (en-AU) で利用可能になりました。詳細については、「Languages supported by Amazon Lex」(Amazon Lex でサポートされている言語) を参照してください。	2020 年 9 月 8 日
新機能	Amazon Lex に 7 つの新しい組み込みインテントと 9 つの新しいビルトインスロットタイプが追加されました。詳細については、「Built-in Intents and Slot Types」(組み込みのインテントとスロットタイプ) を参照してください。	2020 年 9 月 8 日

新しい例	カスタマーサポートエージェントが Amazon Kendra で回答を検索して顧客の質問に回答するために使用できる Amazon Lex ボットの作成方法について説明します。詳細については、「Example: Call Center Agent Assistant」(例: コールセンターエージェントアシスタント) を参照してください。	2020 年 8 月 10 日
新機能	Amazon Lex では、信頼度スコアに基づいて最大 4 つの代替インテントを返すことができるようになりました。詳細については、「Using Confidence Scores」(信頼スコアの使用) を参照してください。	2020 年 8 月 6 日
リージョンの拡張	Amazon Lex が、アジアパシフィック (東京) (ap-northeast-1) リージョンで利用可能になりました。	2020 年 6 月 30 日
新機能	Amazon Lex がよくある質問への回答を得るための Amazon Kendra インデックスの検索を新たにサポートするようになりました。詳細については、「AMAZON.KendraSearchIntent」を参照してください。	2020 年 6 月 11 日

新機能	Amazon Lex は、会話ログでより多くの情報を返すようになりました。詳細については、「Viewing Text Logs in Amazon CloudWatch Logs」(Amazon CloudWatch Logs でのテキストログの表示) を参照してください。	2020 年 6 月 9 日
リージョンの拡張	Amazon Lex が、アジアパシフィック (シンガポール) (ap-southeast-1)、欧州 (フランクフルト) (eu-central-1)、欧州 (ロンドン) (eu-west-2) で利用可能になりました。	2020 年 4 月 23 日
新機能	Amazon Lex がタグ付けを新たにサポートしました。タグ付けを使用して、リソースの識別、コストの割り当て、アクセスコントロールを行うことができます。詳細については、「Tagging your Amazon Lex Resources」(Amazon Lex リソースにタグを付ける) を参照してください。	2020 年 3 月 12 日
新機能	Amazon Lex が AMAZON.AlphaNumeric 組み込みスロットタイプの正規表現を新たにサポートするようになりました。詳細については、「AMAZON.AlphaNumeric」を参照してください。	2020 年 2 月 6 日

新機能	Amazon Lex は会話情報をログに記録し、それらのログにスロット値を難読化できるようになりました。詳細については、「会話ログの作成」および「スロットの難読化」を参照してください。	2019 年 12 月 19 日
リージョンの拡張	Amazon Lex が、アジアパシフィック (シドニー) (ap-southeast-2) で利用可能になりました。	2019 年 12 月 17 日
新機能	Amazon Lex が HIPAA に準拠しました。詳細については、「Compliance Validation for Amazon Lex」(Amazon Lex のコンプライアンス検証) を参照してください。	2019 年 12 月 10 日
新機能	Amazon Lex がユーザー発話を Amazon Comprehend に送信して、発話のセンチメントを分析できるようになりました。詳細については、「Sentiment Analysis」(センチメント分析) を参照してください。	2019 年 11 月 21 日
新機能	Amazon Lex が SOC に準拠しました。詳細については、「Compliance Validation for Amazon Lex」(Amazon Lex のコンプライアンス検証) を参照してください。	2019 年 11 月 19 日

新機能	Amazon Lex が PCI に準拠しました。詳細については、「Compliance Validation for Amazon Lex」(Amazon Lex のコンプライアンス検証) を参照してください。	2019 年 10 月 17 日
新機能	会話中に簡単にインテントに戻ることができるように、インテントにチェックポイントを追加するためのサポートが追加されました。詳細については、「セッションの管理」を参照してください。	2019 年 10 月 10 日
新機能	ユーザーの入力が想定どおりでない状況をボットで処理できるように、`AMAZON.FallbackIntent` のサポートを追加しました。詳細については、「AMAZON.FallbackIntent」を参照してください。	2019 年 10 月 3 日
新機能	Amazon Lex を使用すると、ボットのセッション情報を管理できます。詳細については、「Managing Sessions With the Amazon Lex API」(Amazon Lex API を使用したセッションの管理) を参照してください。	2019 年 8 月 8 日
リージョンの拡張	Amazon Lex が、米国西部 (オレゴン) (us-west-2) で利用可能になりました。	2018 年 5 月 8 日

新機能	Amazon Lex 形式でのエクスポートおよびインポートのサポートが追加されました。詳細については、「Importing and Exporting Amazon Lex Bots, Intents, and Slot Types」(Amazon Lex ボット、インテント、スロットタイプのインポートとエクスポート) を参照してください。	2018 年 2 月 13 日
新機能	Amazon Lex がボットの追加のレスポンスメッセージを新たにサポートするようになりました。詳細については、「レスポンス」を参照してください。	2018 年 2 月 8 日
リージョンの拡張	Amazon Lex が、欧州 (アイルランド) (eu-west-1) で利用可能になりました。	2017 年 11 月 21 日
新機能	Kik に Amazon Lex ボットをデプロイするためのサポートが追加されました。詳細については、「Integrating an Amazon Lex Bot with Kik」(Amazon Lex ボットと Kik との統合) を参照してください。	2017 年 11 月 20 日

新機能	新しい組み込みスロットタイプとリクエスト属性のサポートを追加しました。詳細については、「組み込みスロット形式」および「リクエスト属性の設定」を参照してください。	2017 年 11 月 3 日
新機能	Alexa Skills Kit にエクスポートする機能を追加しました。詳細については、「Alexa Skill へのエクスポート」を参照してください。	2017 年 9 月 7 日
新機能	スロットタイプの値にシノニムのサポートを追加しました。詳細については、「カスタムスロットタイプ」を参照してください。	2017 年 8 月 31 日
新機能	AWS CloudTrail の統合が追加されました。詳細については、「AWS CloudTrail ログを使用した Amazon Lex API コールのモニタリング」を参照してください。	2017 年 8 月 15 日
ドキュメントの拡張	AWS CLI の開始方法の例が追加されました。詳細については、「https://docs.aws.amazon.com/lex/latest/dg/gs-cli.html」を参照してください。	2017 年 5 月 22 日
新しいガイド	これは「Amazon Lex ユーザーガイド」の最初のリリースです。	2017 年 4 月 19 日

AWS 用語集

AWS の最新の用語については、「AWS の用語集リファレンス」の「AWS 用語集」を参照してください。