aws

V2 デベロッパーガイド

Amazon Lex



Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon Lex: V2 デベロッパーガイド

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon の商標およびトレードドレスは Amazon 以外の製品およびサービスに使用することはできません。また、お客様に誤解を与える可能性がある形式で、または Amazon の信用を損なう形式で使用することもできません。Amazon が所有していないその他のすべての商標は Amazon との提携、関連、支援関係の有無にかかわらず、それら該当する所有者の資産です。

Table of Contents

Amazon Lex V2 とは何でしょう？ 1
Amazon Lex V2 の一般的なユースケース V2 3
Amazon Lex V2 の支払い V2 3
Amazon Lex V2 を初めて利用される方ですか？ 4
AWS SDKs の使用 4
AWS SDK での Amazon Lex V2 の使用 5
最新の機能 8
Bedrock エージェントおよびナレッジベースと接続するための AMAZON.BedrockAgentIntent 8
インテント分類とスロット解決を改善するためのアシスト付き NLU 8
17 の追加言語向けのカスタム語彙のサポート 8
Amazon Connect 統合用の QinConnect 組み込みインテント 8
AMAZON.Currency および AMAZON.Confirmation 組み込みスロットが、すべてのロケールをサポートするようになりました 9
Bedrock ナレッジベースとガードレールを使用した拡張 QnA インテント 9
Anthropic Claude 3 モデルのサポート 9
リージョン間のボットレプリケーション用のグローバルレジリエンシー 9
AWS GovCloud (米国西部) のリージョンサポート 10
Amazon Lex V2 の生成 AI 機能 10
Amazon.Confirmationのビルトインスロットは、「はい」、「いいえ」、「多分」、「わからない」の曖昧さ回避用です。 10
Analytics によるビジネスパフォーマンスの測定 11
Test Workbench によるボットのパフォーマンスの評価 11
業種固有のボットテンプレート 11
ボットのネットワーク 11
ビジュアル会話ビルダー 12
コンポジットスロットタイプ 12
条件付き分岐 12
Automated chatbot designer 12
ランタイムヒント 13
カスタム語彙 13
文法スロットタイプ 13
Amazon Lex V2 のコア概念 14
クイックスタート学習パス 14

詳細な開発プロセス 14
重要な概念と用語 15
Amazon Lex V2 の高度な機能 17
リージョン別の利用可能性 17
サポートされている言語 17
サポートされている言語と口ケール 17
Amazon Lex V2 の機能でサポートされている言語と口ケール 21
リージョン 24
開始方法 25
概要: 25
前提条件チェックリスト 25
Amazon Lex V2 の概要 V2 25
ステップ 1: アカウントを設定する 26
AWS にサインアップする 26
IAM ユーザーの作成 27
プログラマチックアクセス権を付与する 28
次のステップ 28
ステップ 2: 開始方法 (コンソール) 29
コンソールの概要 29
学習パスを選択する 29
基本的な演習 30
クイックスタート: 5 分でチャットボットを作成する 30
演習 1: テンプレートからチャットボットを作成する 32
演習 2: 会話フローを確認する 36
演習 3: 高度なカスタマーサービスチャットボットを構築する 47
開始方法のベストプラクティス 68
ボットの例 73
コールセンターエージェントアシスタント 73
ステップ 1: Amazon Kendra インデックスを作成する 74
ステップ 2: Amazon Lex V2 ボットを作成する V2 75
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 76
ステップ 4: Amazon Cognito をセットアップする 77
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 79
ステップ 6: ボットを使用する 79
Amazon Lex V2 ボットの使用 83
Amazon Lex V2 の会話フローの変更 V2 84

Amazon Lex V2 でボットを作成するさまざまな方法 85
コンソールを使用する 86
ポッドテンプレートの使用 87
自動 Chatbot デザイナーを使用した Amazon Lex V2 ボットの作成 90
Amazon Lex V2 ボットへの新しい言語の追加 99
インテントの追加 99
特定の順序でのプロンプトの設定 102
サンプル発話 102
インテントの構造 104
会話パスの作成 126
ビジュアル会話ビルダーの使用 143
組み込みのインテント 153
スロットタイプの追加 185
組み込みスロットタイプ 186
カスタムスロットタイプ 201
文法スロットタイプ 204
コンポジットスロットタイプ 352
ボットのテスト 31
生成 AI による Lex V2 ボットの最適化 363
説明からボットを構築する記述型ボットビルダー 365
記述型ボットビルダーの例 369
NLD に必要な権限 371
発話生成を使用したインテント認識用のサンプル発話の生成 372
スロット値に対するアシスト付きスロット解決の使用 373
アシスト付きスロット解決の例 374
生成 AI 設定で有効にする 378
スロットに対するアシスト付きスロット解決を有効にする 379
アシスト付きスロット解決の権限 381
BedrockAgentIntent を使用した、Amazon Lex V2 での Bedrock エージェントの使用 381
生成 AI 設定で有効にする 383
ボットの Bedrock エージェントインテントを有効にする 383
Bedrock エージェントインテントのアクセス許可 384
リクエスト例 384
インテント分類とスロット解決を改善する 386
アシスト付き NLU の無効化 388
意図的な曖昧さを解消して、あいまいなユーザー入力を解決する 388

インテントの曖昧さの解消 390
AI を活用した Bot Analyzer を使用してボットを最適化する 391
AMAZON.QnAIntent 396
アクセス許可 398
Lex V2 ボットのネットワークの作成 400
Lex V2 ボットのネットワークを作成する 401
Lex V2 ボットのネットワークを管理する 402
Lex V2 ボットのネットワークのバージョン 403
Lex V2 ボットのネットワークのエイリアス 403
Lex V2 ボットのネットワークのチャネル統合 404
本番環境用の Lex V2 ボットのデプロイ 405
Lex V2 ボットを使用したバージョニングとエイリアス 405
バージョン 405
Lex V2 ボットのエイリアス 406
Java アプリケーションとの統合 408
グローバルレジリエンシーを使用した他のリージョンへのボットのデプロイ 412
アクセス許可 414
Lex V2 ボットを使用したグローバルレジリエンシーのデプロイ 415
メッセージングプラットフォームとの統合 418
Facebook との統合 419
Slack との統合 422
Twilio SMS との統合 427
コンタクトセンターとの統合 429
Amazon Chime SDK 430
Connect のお客様 431
ジェネシスクラウド 432
Amazon Lex V2 ボットの会話について 433
Lex V2 ボットとの会話コンテキスト 434
Lex V2 ボットのインテントコンテキストの設定 435
Lex V2 ボットのインテントでのデフォルトスロット値の使用 437
Lex V2 ボットのセッション属性の設定 438
Lex V2 ボットのリクエスト属性の設定 440
セッションタイムアウトを設定する 443
Lex V2 ボットでのインテント間の情報共有 443
Lex V2 ボットでの複雑な属性の設定 444
ボットセッションについて 445

新しいセッションを開始する 447
インテントの切り替えをする 447
前のインテントを再開する 448
スロット値の検証をする 448
AWS Lambda関数の統合 450
AWS LambdaLex V2 の入力イベント形式 450
AWS LambdaLex V2 のレスポンス形式 458
応答内の必須フィールド 460
共通構造 463
Intent 463
Slots 465
セッション状態 468
Lambda 関数を作成して、ボットエイリアスにアタッチする 472
コンソールを使用して Amazon Lex V2 ボットに AWS Lambda関数をアタッチする 475
API オペレーションを使用して Amazon Lex V2 ボットに AWS Lambda関数をアタッチする 477
Lambda 関数のデバッグ 483
Lex V2 でのユーザーとボットのインタラクションのカスタマイズ 485
会話中のユーザーの感情分析 485
信頼度スコアを使用した会話精度の向上 486
Lex V2 でのインテント信頼スコアを使用したインテント選択の改善 487
音声文字起こしの信頼度スコアを使用した Lex V2 ボットとの会話の改善 490
Lex V2 ボットで使用する音声文字起こしのカスタマイズ 500
音声認識モデルの設定 500
カスタム語彙による音声認識の向上 506
音声アクティビティの検出感度の設定 515
スピーカーダイアライゼーションを使用してプライマリスピーカーに集中する 517
会話内のランタイムヒントによるスロット値の認識の向上 523
会話中のスペルスタイルによるスロット値のキャプチャ 526
オーディオフィラーを使用してボットの応答性を向上させる 534
使用可能なオーディオフィラータイプ 534
タイミングパラメータ 534
オーディオフィラーの設定 535
AI エージェント中間メッセージを含むオーディオフィラー 536
ダイアログおよびフルフィルメントコードフックを備えたオーディオフィラー 537
オーディオフィラーのベストプラクティス 537

Lex V2 のボットパフォーマンスのモニタリング 538
Analytics によるビジネスパフォーマンスの測定 538
主な定義 539
結果のフィルタリング 541
概要 542
会話ダッシュボード 546
パフォーマンスダッシュボード 551
分析に API を使用する 555
分析のアクセス許可の管理 562
Lex V2 ボットの会話ログの有効化 562
Lex V2 の会話ログによる会話のログ記録 563
Lex V2 の会話ログでスロット値を隠す 580
Lex V2 の会話ログの選択的なキャプチャ 582
Lex V2 のエラーログを使用したエラーのログ記録 589
エラーログの IAM ポリシー 590
Lex V2 のエラーログの有効化 590
Lex V2 のエラーログの無効化 591
Lex V2 でのオペレーションメトリクスのモニタリング 591
CloudWatch による Lex V2 のオペレーションメトリクスの測定 592
CloudTrail を使用してイベントを表示する 601
Test Workbench による Lex V2 ボットのパフォーマンス評価 604
Test Workbench のテストセットの生成 605
テストセットを管理する 618
テストを実行する 643
Test Workbench でのテストセットのカバレッジ 645
テスト結果を表示する 646
Test Workbench のテスト結果の詳細 647
Lex V2 ボットへの会話のストリーミング 654
Amazon Lex V2 ボットへの会話ストリーミングの開始 655
Amazon Lex V2 ボットを使用する際の音声会話イベントの時系列 659
ストリーミングの会話をスタートする 661
イベントストリームエンコード 677
ボットの中断を可能にする 679
ユーザーの追加の情報提供を待機する 680
Lex V2 ボットのフルフィルメント進行状況の更新の設定 682
フルフィルメントのアップデート 682

フルフィルメント後のレスポンス 684
ユーザー入力のタイムアウト 686
Lex V2 ボットでの割り込み動作の仕組み 687
音声入力のタイムアウトの設定 687
テキスト入力のタイムアウト 688
DTMF 入力の設定 689
Lex V2 でのボットのインポートとエクスポート 691
Lex V2 からのボットのエクスポート 691
Lex V2 でボットをエクスポートするために必要な IAM 権限 692
Lex V2 ボットのエクスポート (コンソール) 693
Lex V2 でのボットのインポート 695
インポートに必要な IAM 権限 696
Lex V2 ボットのインポート (コンソール) 697
インポートまたはエクスポート時のパスワードの使用 699
Lex V2 でボットをインポートおよびエクスポートするための JSON 形式 699
マニフェストファイル構造 700
ボットファイル構造 701
ボットロケールファイル構造 701
インテントファイル構造 701
スロットファイル構造 703
スロットタイプファイル構造 707
カスタム語彙ファイル構造 709
Lex V2 でのリソースのタグ付け 711
コンソールまたは API を使用したリソースのタグ付け 711
Lex V2 を使用する場合のタグの制限 712
リソースのタグ付け (コンソール) 712
セキュリティ 714
データ保護 715
保管中の暗号化 715
転送中の暗号化 716
ID とアクセス管理 716
オーディエンス 717
アイデンティティを使用した認証 717
ポリシーを使用したアクセスの管理 719
Amazon Lex V2 で IAM が機能する仕組み 721
アイデンティティベースのポリシーの例 730

リソースベースのポリシーの例 740
AWS マネージドポリシー 748
サービスにリンクされたロールの使用 767
サービスロールの使用 772
サービスロールへの移行 773
トラブルシューティング 775
ログ記録とモニタリング 777
コンプライアンス検証 778
耐障害性 779
インフラストラクチャセキュリティ 779
VPC エンドポイント (AWS PrivateLink) 780
Amazon Lex V2 VPC エンドポイントに関する考慮事項 780
Amazon Lex V2 用のインターフェイス VPC エンドポイントの作成 780
Amazon Lex V2 用の VPC エンドポイントポリシーの作成 781
コードの例 782
シナリオ 782
Amazon Lex V2 チャットボットの構築 782
ガイドラインとベストプラクティス 784
クォータ 787
ビルド時のクォータ 787
ランタイムクォータ 790
移行ガイド 794
Amazon Lex V2 の概要 794
ボット内の複数の言語 794
簡略化された情報アーキテクチャ 794
ビルダーの生産性向上 795
AWS CloudFormation リソース 797
Amazon Lex V2 と CloudFormation テンプレート 797
の詳細 CloudFormation 797
ドキュメント履歴 799
API リファレンス 818
AWS 用語集 819
..... dcccxx

Amazon Lex V2 とは何でしょう？

Amazon Lex V2 は、音声とテキストを使用してアプリケーションの会話型インターフェースを構築するための AWS のサービスです。Amazon Lex V2 は、自然言語理解（NLU）と自動音声認識（ASR）の深い機能と柔軟性を提供するので、リアルな会話のようなインタラクションで非常に魅力的なユーザー体験を構築し、新しいカテゴリーの製品を作成することができます。

Amazon Lex V2 を使用すると、開発者は会話型チャットボットをすばやく構築できます。Amazon Lex V2 では、深層学習の専門知識は必要ありません。チャットボットを作成するには、Amazon Lex V2 コンソールで基本的な会話フローを指定します。Amazon Lex V2 がダイアログを管理し、会話中の応答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。その後、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) のチャットボットに会話インターフェイスを追加できます。

Amazon Lex V2 には、ボット構築をさらに強力でアクセス可能なものにする高度な AI を活用した機能が追加されました。アシスト NLU は、大規模言語モデル (LLMs) を使用して、ボットが設定したインテントとスロット内に留まりながら、インテント分類とスロット解決を改善します。つまり、必要なトレーニングデータが少なく、ユーザーリクエストをよりよく理解できます。Amazon Lex V2 は、17 の追加の言語でカスタム語彙もサポートしているため、グローバルなデプロイが可能になり、さまざまな市場で音声認識の精度が向上します。

マルチリージョンレプリケーション (MRR) を使用すると、複数の AWS リージョンにボットをデプロイして可用性とディザスタリカバリを向上させ、世界中のユーザーが会話型インターフェイスにアクセスできるようにできるようになりました。

Amazon Lex V2 は との統合を提供し AWS Lambda、Connect Customer、Amazon Comprehend、Amazon Kendra など、AWS プラットフォーム上の他の多くのサービスと統合できます。Lambda との統合により、ボットは Salesforce などの SaaS アプリケーションのデータにリンクするための、あらかじめ構築されたサーバーレスエンタープライズコネクタにアクセスすることができます。

2022 年 8 月 17 日以降に作成されたボットの場合、条件付き分岐を使用してボットとの会話の流れを制御できます。条件付き分岐を使用すると、Lambda コードを記述しなくても複雑な会話を作成できます。

Amazon Lex V2 は、以下のような利点を提供します。

- シンプルさ - Amazon Lex V2 は、コンソールを使用して独自のボットを数分で作成できるようにガイドします。いくつかのフレーズを入力すると、Amazon Lex V2 は完全な自然言語モデルを構

築し、ボットは音声とテキストを使用して対話し、質問をしたり、回答を得たり、高度なタスクを完了することができます。

- 深層学習技術の一般化 - Amazon Lex V2 は、音声言語理解（SLU）システムを構築するための ASR と NLU の技術を提供します。SLU を通じて、Amazon Lex V2 は自然言語の音声とテキスト入力を受け取り、入力の背後にある意図を理解し、適切なビジネス機能を呼び出すことによってユーザーの意図を満たします。

  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 V2 は、深層学習技術をすべての開発者の手の届くところに提供します。Amazon Lex V2 ボットは、入力された音声をテキストに変換し、ユーザーの意図を理解してインテリジェントな応答を生成するため、顧客にとって付加価値のあるボットの構築に専念でき、会話型インターフェースで実現する全く新しいカテゴリーの製品を定義することが可能です。

- シームレスなデプロイとスケーリング - Amazon Lex V2 では、Amazon Lex V2 コンソールから直接ボットを構築、テスト、デプロイすることができます。Amazon Lex V2 では、モバイルデバイス、ウェブアプリ、チャットサービス（例えば、Facebook Messenger など）で使用する音声またはテキストボットの公開が可能です。Amazon Lex V2 は自動的に拡張します。ボット体験のための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を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 V2 は、 AWS Lambda や Amazon CloudWatch などの他の AWS サービスとネイティブに動作します。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。

- 費用対効果 - Amazon Lex V2 では、初期費用や最低利用料が不要です。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 V2 の無料利用枠を利用すると、初期投資なしで簡単に Amazon Lex V2 を試すことができます。

Amazon Lex V2 の一般的なユースケース V2

Amazon Lex V2 を使用すると、さまざまなビジネスシナリオ向けに高度な会話型インターフェイスを構築できます。一般的なユースケースをいくつか紹介します。

- カスタマーサポート – 一般的な顧客の問い合わせを処理し、問題をトラブルシューティングし、複雑な問題を人間のエージェントにエスカレーションできるインテリジェントなチャットボットを作成します。既存の CRM およびナレッジベースシステムと統合します。
- e コマースと小売 – 顧客が製品を検索し、注文ステータスをチェックし、返品を処理し、購入履歴に基づいてパーソナライズされたレコメンデーションを提供するのに役立つショッピングアシスタントチャットボットを構築します。
- 予約予約 – 可用性の確認、予約の予約、リマインダーの送信、キャンセルの処理が可能な、医療、プロフェッショナルサービス、または接客のためのチャットボットのスケジュールを作成します。
- IT ヘルプデスク – パスワードのリセット、ソフトウェアのインストールガイダンスの提供、IT チケットの追跡、従業員を適切な技術リソースに接続できる内部サポートチャットボットを作成します。
- 金融サービス – 厳格なセキュリティ基準を維持しながら、アカウント残高の確認、資金の移管、取引履歴の提供、財務上のアドバイスを提供できるバンキングチャットボットを構築します。
- 人事および従業員サービス – ポリシーの質問に回答し、福利厚生の登録を支援し、休暇申請を処理し、新入社員のオンボーディングサポートを提供する人事アシスタントチャットボットを開発します。

Amazon Lex V2 は、Slack、Microsoft Teams、WhatsApp、Facebook Messenger、カスタムウェブアプリケーションなどの一般的な通信プラットフォームとシームレスに統合されるため、ユーザーが既に通信しているチャットボットを簡単にデプロイできます。

Amazon Lex V2 の支払い V2

Amazon Lex V2 では、ユーザーが行ったテキストまたは音声リクエストに対してのみ課金されます。このモデルでは、AWS インフラストラクチャのコスト面のメリットを得ながら、ビジネスの成長に応じた可変コストのサービスを利用することができます。詳細については、Amazon Lexの料金 V2」を参照してください。

AWS にサインアップすると、AWS アカウントは Amazon Lex V2 を含む AWS のすべてのサービスに自動的にサインアップされます。ただし、料金が発生するのは実際に使用したサービスの分だけで

す。Amazon Lex V2 を初めて使用する場合は、Amazon Lex V2 の使用を無料で開始できます。 V2 詳細については、「AWS 無料利用枠」を参照してください。

請求を表示するには、AWS Billing and Cost Management コンソールで請求およびコスト管理ダッシュボードに移動します。 AWS アカウント 請求の詳細については、 AWS Billing ユーザーガイドを参照してください。AWS の請求と に関するご質問は AWS アカウント、AWS サポートにお問い合わせください。

Amazon Lex V2 を初めて利用される方ですか？

Amazon Lex V2 を初めてご利用になる方は、以下の項目を順番にお読みいただくことをお勧めします。

1. Amazon Lex V2 のコア概念 - このセクションでは、Amazon Lex V2 と、chatbot を作成するために使用する機能を紹介します。
2. Amazon Lex V2 の開始方法 - このセクションでは、アカウントの設定と Amazon Lex V2 のテストを行います。
3. API リファレンス — このセクションには API オペレーションの詳細が記載されています。

AWS SDK での Amazon Lex V2 の使用

AWS Software Development Kit (SDKsは、多くの一般的なプログラミング言語で利用できます。各 SDK には、デベロッパーが好みの言語でアプリケーションを簡単に構築できるようになる API、コード例、およびドキュメントが提供されています。

SDK ドキュメント	コード例
AWS SDK for C++	AWS SDK for C++ コード例
AWS CLI	AWS CLI コード例
AWS SDK for Go	AWS SDK for Go コード例
AWS SDK for Java	AWS SDK for Java コード例
AWS SDK for JavaScript	AWS SDK for JavaScript コード例
AWS SDK for Kotlin	AWS SDK for Kotlin コード例

SDK ドキュメント	コード例
AWS SDK for .NET	AWS SDK for .NET コード例
AWS SDK for PHP	AWS SDK for PHP コード例
AWS Tools for PowerShell	AWS Tools for PowerShell コード例
AWS SDK for Python v4	AWS SDK for Python v4 コード例
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) コード例
AWS SDK for Ruby	AWS SDK for Ruby コード例
AWS SDK for Rust	AWS SDK for Rust コード例
AWS SDK for SAP ABAP	AWS SDK for SAP ABAP コード例
AWS SDK for Swift	AWS SDK for Swift コード例

> **可用性の例**
>
> 必要なものが見つからなかった場合。このページの下側にある [Provide feedback] リンクから、コードの例をリクエストしてください。

AWS SDK での Amazon Lex V2 の使用

コンソールインターフェイス以外にも、Amazon Lex V2 は AWS SDK を介した包括的なプログラムによるアクセスを提供します。これにより、会話型 AI 機能をアプリケーションに直接統合し、ボット管理タスクを自動化し、スケーラブルなソリューションを構築できます。

Amazon Lex V2 で AWS SDK を使用すると、次のことができます。

- ボットの作成と管理の自動化 – 手動コンソールの操作なしでボットをプログラムで作成、更新、デプロイする
- 既存のアプリケーションとの統合 – ウェブアプリケーション、モバイルアプリケーション、エンタープライズシステムに会話型インターフェイスを追加する

- ボットオペレーションのスケーリング – コードを使用して複数のボット、インテント、スロットタイプを効率的に管理する
- カスタムワークフローの実装 – ビジネスロジックと統合される高度な会話フローを構築する

下の表は、Amazon Lex V2 オペレーションをサポートする AWS SDK を示しています。開発環境に合致する SDK を選択し、提供されているリンクに従ってインストールと実装を開始します。

プログラミング言語	AWS SDK	リソースの利用開始
Java	AWS SDK for Java 2.x	デベロッパーガイド API リファレンス
Python	「AWS SDK for Python (Boto3)」	ご利用開始にあたって API リファレンス
JavaScript/ Node.js	AWS SDK for JavaScript v3	デベロッパーガイド API リファレンス
.NET/C#	AWS SDK for .NET	デベロッパーガイド API リファレンス
Go	AWS SDK for Go v2	デベロッパーガイド API リファレンス
Ruby	AWS SDK for Ruby	デベロッパーガイド API リファレンス
PHP	AWS SDK for PHP	デベロッパーガイド API リファレンス

SDK の使用を開始するには:

1. デベロッパーガイドのインストール手順に従って、希望するプログラミング言語の SDK をインストールする
2. AWS の認証情報とリージョン設定の構成
3. Amazon Lex V2 オペレーションの必要な IAM アクセス許可を設定する
4. 必要な特定のオペレーションについては、API リファレンスドキュメントを参照してください。
5. ボットの作成や既存のリソースの一覧表示などの基本的なオペレーションから始める

プログラムによるボット作成の詳細な例とステップバイステップガイドについては、上記の表に記載されている SDK ドキュメントのリンクを参照してください。

最新の機能

このトピックでは、Amazon Lex V2 が提供する最新の機能について説明します。

Bedrock エージェントおよびナレッジベースと接続するための AMAZON.BedrockAgentIntent

Amazon Lex V2 に AMAZON.BedrockAgentIntent が組み込まれ、ボットを Amazon Bedrock エージェントとナレッジベースにシームレスに接続して会話型 AI 機能を強化できるようになりました。

- ドキュメント

インテント分類とスロット解決を改善するためのアシスト付き NLU

Amazon Lex V2 では、ボットが設定したインテントおよびスロット内に留まりながら、大規模言語モデル (LLM) を使用してインテント分類とスロット解決を改善するアシスト付き NLU が提供されるようになりました。

- ドキュメント

17 の追加言語向けのカスタム語彙のサポート

Amazon Lex V2 は、音声認識の精度を向上させるために、17 の追加言語でカスタム語彙をサポートするようになりました。

- ドキュメント

Amazon Connect 統合用の QinConnect 組み込みインテント

Amazon Lex V2 では、コンタクトセンターのエクスペリエンスを強化するために、ボットを Amazon Connect にシームレスに接続するための QinConnect 組み込みインテントが提供されるようになりました。

- ドキュメント

AMAZON.Currency および AMAZON.Confirmation 組み込みスロットが、すべてのロケールをサポートするようになりました

Amazon Lex V2 は、AMAZON.Currency および AMAZON.Confirmation 組み込みスロットタイプをすべてのロケールでサポートするようになり、以前にサポートされていた英語とスペイン語のロケールを超えて拡張されました。

- ドキュメント

Bedrock ナレッジベースとガードレールを使用した拡張 QnA インテント

生成 AI の QnA 組み込みインテントが、Bedrock ナレッジベースとガードレールをサポートするようになり、より安全で正確なレスポンスが実現されました。

- ドキュメント

Anthropic Claude 3 モデルのサポート

生成 AI 向けの QnA 組み込みインテントが、Anthropic Claude 3 Haiku および Anthropic Claude 3 Sonnet モデルをサポートするようになり、会話機能が拡張されました。

- ドキュメント

リージョン間のボットレプリケーション用のグローバルレジリエンシー

Amazon Lex V2 では、可用性とディザスタリカバリを向上させるために、ボットを 2 番目の AWS リージョンでレプリケートするためのグローバルレジリエンシーが提供されるようになりました。

- ドキュメント

AWS GovCloud (米国西部) のリージョンサポート

Amazon Lex V2 が で利用可能になりました AWS GovCloud (US-West)。

- Amazon Lex エンドポイントとクォータ

Amazon Lex V2 の生成 AI 機能

Amazon Lex V2 では、Amazon Bedrock の生成 AI 機能をボットに活用できるようになりました。

- 記述的ボットビルダー
  - 「新着情報」の投稿
  - ドキュメント
- アシスト付きスロット解決
  - 「新着情報」の投稿
  - ドキュメント
- 発話生成
  - 「新着情報」の投稿
  - ドキュメント
- `AMAZON.QnAIntent` (会話型のよくある質問)
  - 「新着情報」の投稿
  - ドキュメント
- AWS 機械学習ブログ記事

Amazon.Confirmationのビルトインスロットは、「はい」、「いいえ」、「多分」、「わからない」の曖昧さ回避用です。

Amazon Lex V2 では、スロット確認の精度や「はい/いいえ/たぶん/わからない」の各応答の精度を向上させるため、`AMAZON.Confimation` 組み込みスロットを提供するようになりました。

- ドキュメント

Analytics によるビジネスパフォーマンスの測定

Amazon Lex V2 では、ユーザーが Analytics ダッシュボードでインテントとスロットのパフォーマンスを表示できるようになりました。

- 「新着情報」の投稿
- ドキュメント

Test Workbench によるボットのパフォーマンスの評価

Amazon Lex V2 では、ボットのパフォーマンスを測定し、ボットのメトリックスを改善するためのテストセットを作成して実行できるようになりました。

- 「新着情報」の投稿
- ドキュメント
- AWS 機械学習ブログ記事

業種固有のボットテンプレート

Amazon Lex V2 では、音声とチャットの両方の方法で、トレーニングデータとダイアログプロンプトの両方とともに、すぐに使用できる会話フローを備えた事前構築済みのボットテンプレートがユーザーに提供されるようになりました。

- 「新着情報」の投稿
- ドキュメント

ボットのネットワーク

Amazon Lex V2 では、複数のボットを単一のネットワークに結合したり、ユーザーの入力に基づいてリクエストを適切なボットにルーティングしたりできるようになりました。

- 「新着情報」の投稿

- ドキュメント

ビジュアル会話ビルダー

Amazon Lex V2 では、豊富なビジュアル環境内でインテントを使用して会話パスを簡単に設計および視覚化できる、ドラッグアンドドロップの会話ビルダーが提供されるようになりました。

- 「新着情報」の投稿
- ドキュメント
- AWS 機械学習ブログ記事

コンポジットスロットタイプ

Amazon Lex V2 では、ユーザーが論理式を使用して複数のスロットを複合スロットに結合できるようになりました。

- 「新着情報」の投稿
- ドキュメント

条件付き分岐

Amazon Lex V2 では、ユーザーがボットとの会話を通じて顧客がたどるパスをより適切に制御するための条件を記述できるようになりました。

- 「新着情報」の投稿
- ドキュメント

Automated chatbot designer

Amazon Lex V2 では、会話の記録からチャットボットを自動的に設計するオプションがユーザーに提供されるようになりました。使用例については、次をお読みください。

- 「新着情報」の投稿
- ドキュメント

- AWS 機械学習ブログ記事
- Amazon Lex Automated Chatbot Designer ページ

ランタイムヒント

Amazon Lex V2 では、ユーザーがランタイムヒントを設定してフレーズの認識を向上させ、スロット値をより正確に把握できるようになりました。

- 「新着情報」の投稿
- ドキュメント

カスタム語彙

Amazon Lex V2 では、カスタム語彙 (固有名詞やドメイン固有の単語を含むフレーズのリスト) を作成し、Amazon Lex V2 が音声入力で認識できるようにするオプションがユーザーに提供されるようになりました。

- 「新着情報」の投稿
- ドキュメント
- AWS 機械学習ブログ記事

文法スロットタイプ

Amazon Lex V2 では、会話中の情報を収集するために、音声認識文法仕様 (SRGS) に従って XML 形式で文法を作成できるようになりました。

- 「新着情報」の投稿
- ドキュメント
- AWS 機械学習ブログ記事

Amazon Lex V2 のコア概念

Amazon Lex V2 を使用すると、チャットアプリケーション (ボット) を構築して、ユーザーから情報を引き出してタスクを達成できます。たとえば、チャットボットを作成して、カスタマーサポートの提供、よくある質問への回答、予約を行うことができます。Amazon Lex V2 を使用する一般的な手順を以下に示します。

クイックスタート学習パス

Amazon Lex V2 を初めて使用する場合 V2 このプログレッシブラーニングパスに従って、すぐに開始してください。

1. テンプレートから始める (5 分) – カスタマーサポートのよくある質問、予約予約、注文ステータスなどの構築済みのチャットボットテンプレートから選択します。テンプレートには、事前設定されたインテント、スロット、サンプル発話が含まれます。
2. チャットボットをカスタマイズする (15 分) – 特定のユースケースに合わせてテンプレートを変更します。独自のインテントの追加、サンプル発話の更新、ドメインのスロットタイプの設定を行います。
3. テストと改良 (10 分) – 組み込みのテストコンソールを使用して、チャットボットと会話します。アシスト付き NLU を有効にして、最小限のトレーニングデータで理解を深めます。
4. デプロイと統合 (20 分) – チャットボットを公開し、任意のプラットフォーム (Slack、ウェブアプリ、モバイルアプリケーション) と統合します。

チャットボットの使用までの合計時間: ~50 分

より包括的な理解を得るには、以下の詳細な開発プロセスに進みます。

詳細な開発プロセス

より複雑なボットやゼロから構築する場合は、次の包括的な開発プロセスに従います。

1. ボットを作成し、1 つ以上の言語を追加します。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 V2 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、自身のアプリケーションまたはメッセージングプラットフォーム (Facebook Messenger または Slack など) にデプロイできます。

重要な概念と用語

開始する前に、以下の Amazon Lex V2 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 V2 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。

  Amazon Lex V2 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。

- 言語 — Amazon Lex V2 ボットは、1 つ以上の言語で会話できます。各言語は他の言語から独立しており、ネイティブの単語やフレーズを使用してユーザーと会話するように Amazon Lex V2 を設定できます。詳細については、「Amazon Lex V2 でサポートされている言語と口ケール」を参照してください。
- 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`**。
  - 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 V2 からクライアントアプリケーション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 V2 では、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」を参照してください。

Amazon Lex には、ボットごとに常にフォールバックインテントが含まれています。フォールバックインテントは、Amazon Lex がユーザーの意図を推測できない場合に使用されます。詳細については、「AMAZON.FallbackIntent」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 V2 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 V2 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。

  たとえば、`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。スロットごとに、スロットタイプと 1 以上のプロンプトを提供します。Amazon Lex V2 は、ユーザーから値を引き出すためにクライアントにプロンプトを送信されます。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 V2 はまだスロット値を理解しています。

- スロットタイプ – 各スロットにはタイプがあります。自身の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。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 V2 には、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」を参照してください。

- 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。バージョンを作成したら、その作成時点に存在していたとおりのボットを使用できます。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。

- 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。たとえば、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指す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。

Amazon Lex V2 の高度な機能

上記の主要な概念に加えて、Amazon Lex V2 にはボット機能を強化する高度な機能が含まれています。

- アシスト NLU – 大規模言語モデル (LLMs) を使用して、インテント分類とスロット解決を改善します。この機能は、ボットがトレーニング例とは異なるフレーズを使用している場合でも、ユーザーのリクエストをより正確に理解するのに役立ちます。アシスト付き NLU は、設定されたインテントとスロット内で動作するため、広範なトレーニングデータを必要とせずに理解を深めることができます。
- マルチターン会話 – Amazon Lex V2 は、複数の会話ターンにわたってコンテキストを維持できるため、自然なback-and-forthやり取りが可能になります。ユーザーは、会話のコンテキストを失うことなく、情報を徐々に提供したり、考えを変えたり、明確な質問をしたりできます。
- コンテキストの切り替え – 高度なボットは、会話内のトピックの変更を処理できます。たとえば、ユーザーはアカウント情報について質問し始め、注文に切り替えて元のトピックに戻ることができます。Amazon Lex V2 は、これらのコンテキストスイッチを適切に管理できます。
- フォールバック戦略 – Amazon Lex V2 がユーザーのリクエストを理解していない場合は、質問の明確化、提案プロンプト、ヒューマンエージェントへのエスカレーションなど、高度なフォールバック動作を設定できます。これにより、ユーザーは常に会話を進めることができます。
- 会話フロー管理 – 条件分岐と会話フローコントロールを使用して、コードを記述せずに複雑なダイアログパターンを作成します。ユーザーレスポンス、スロット値、または外部データに基づいて会話をルーティングできます。

リージョン別の利用可能性

Amazon Lex V2 が利用可能な AWS リージョンのリストについては、Amazon Web Services 全般のリファレンスの「Amazon Amazon Lex V2 エンドポイントとクォータ」を参照してください。

Amazon Lex V2 でサポートされている言語と口ケール

Amazon Lex V2 は、さまざまな言語と口ケールをサポートしています。このトピックでは、サポートされている言語と、これらの言語をサポートする機能を記載します。

サポートされている言語と口ケール

Amazon Lex V2 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	言語とロケール
af_ZA*	アフリカーンス語 (南アフリカ)
ar_AE	湾岸アラビア語 (アラブ首長国連邦)
ar_SA*	アラビア語 (サウジアラビア)
bg_BG*	ブルガリア語 (ブルガリア)
ca_ES	カタルーニャ語 (スペイン)
cs_CZ*	チェコ語 (チェコ共和国)
cy_GB*	ウェールズ語 (英国)
da_DK*	デンマーク語 (デンマーク)
de_AT	ドイツ語 (オーストリア)
de_CH*	ドイツ語 (スイス)
de_DE	ドイツ語 (ドイツ)
en_AB*	英語 (スコットランド)
en_AU	英語 (オーストラリア)
en_GB	英語 (英国)
en_IE*	英語 (アイルランド)
en_IN	英語 (インド)
en_NZ*	英語 (ニュージーランド)
en_US	英語 (米国)
en_WL*	英語 (ウェールズ)
en_ZA	英語 (南アフリカ)

コード	言語とロケール
es_419	スペイン語 (ラテンアメリカ)
es_ES	スペイン語 (スペイン)
es_MX*	スペイン語 (メキシコ)
Es_US	スペイン語 (米国)
et_ET*	エストニア語 (エストニア)
fa_IR*	ペルシ語 (イラン)
fi_FI	フィンランド語 (フィンランド)
fr_BE*	フランス語 (ベルギー)
fr_CA	フランス語 (カナダ)
fr_FR	フランス語 (フランス)
he_IL*	ヘブライ語 (イスラエル)
hi_IN	ヒンディー語 (インド)
hr_HR*	クロアチア語 (クロアチア)
hu_HU*	ハンガリー語 (ハンガリー)
id_ID*	インドネシア語 (インドネシア)
is_IS*	アイスランド語 (アイスランド)
it_IT	イタリア語 (イタリア)
ja_JP	日本語 (日本)
km_KH*	クメール語 (カンボジア)
ko_KR	韓国語 (韓国)

コード	言語とロケール
lt_LT*	リトアニア語 (リトアニア)
lv_LV*	ラトビア語 (ラトビア)
ms_MY*	マレー語 (マレーシア)
nl_BE*	オランダ語 (ベルギー)
nl_NL	オランダ語 (オランダ)
no_NO	ノルウェー語 (ノルウェー)
pl_PL	ポーランド語 (ポーランド)
pt_BR	ポルトガル語 (ブラジル)
pt_PT	ポルトガル語 (ポルトガル)
ro_RO*	ルーマニア語 (ルーマニア)
ru_RU*	ロシア語 (ロシア)
sk_SK*	スロバキア語 (スロバキア)
sl_SI*	スロベニア語 (スロベニア)
so_SO*	ソマリア語 (ソマリア)
sr_RS*	セルビア語 (セルビア)
su_ID*	スンダ語 (インドネシア)
sv_SE	スウェーデン語 (スウェーデン)
th_TH*	タイ語 (タイ)
tl_PH*	タガログ語/フィリピン語 (フィリピン)
tr_TR*	トルコ語 (トルコ)

コード	言語とロケール
uk_UA*	ウクライナ語 (ウクライナ)
vi_VN*	ベトナム語 (ベトナム)
zh_CN	北京語 (中華人民共和国)
zh_HK	広東語 (香港)
zu_ZA*	ズールー語 (南アフリカ)

機能サポートが制限されたロケール

アスタリスク (*) が付いたロケールでは、生成 AI とサードパーティーの自動音声認識 (ASR) または Text-To-Speech (TTS) による機能サポートが制限されています。サポートされている機能の完全なリストについては、次の表を参照してください。

生成 AI 機能のサポート

生成 AI 機能のサポートには、プライマリモードでのアシスト NLU、アシスト付きスロット解決、および意図的なあいまいさの排除が含まれます。

Amazon Lex V2 の機能でサポートされている言語とロケール

次の表は、特定の言語とロケールに限定される Amazon Lex V2 機能の一覧です。Amazon Lex V2 のその他すべての機能は、すべての言語とロ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ロケール
AMAZON.AlphaNumeric	韓国語 (ko_KR) と制限付き機能サポートロケールを除くすべての言語とロケール
AMAZON.KendraSearchIntent	英語 (米国) (en_US)
カスタム語彙による音声認識の向上	英国英語 (en_GB) 英語 (米国) (en_US)
自動 Chatbot デザイナー	英語 (米国) (en_US)

機能	サポートされている言語と口ケール
リージョンの可用性	次の言語と口ケールは、アジアパシフィック (シンガポール) (ap-southeast-1) およびアフリカ (ケープタウン) (ap-south-1) リージョンでは利用できません。 • 湾岸アラビア語 (アラブ首長国連邦) (ar_AE) • カタルーニャ語 (スペイン) (ca_ES) • フィンランド語 (フィンランド) (fi_FI) • ヒンディー語 (インド) (hi_IN) • オランダ語 (オランダ) (nl_NL) • ノルウェー語 (ノルウェー) (no_NO) • ポーランド語 (pl_PL) • ポルトガル語 (ブラジル) (pt_BR) • ポルトガル語 (ポルトガル) (pt_PT) • スウェーデン語 (sv_SE) • 北京語 (中華人民共和国) (zh_CN) • 広東語 (香港) (zh_HK)
Lex V2 ボットのインテントコンテキストの設定	英語 (米国) (en_US)
文法スロットタイプ	英語 (オーストラリア) (en_AU) 英国英語 (en_GB) 英語 (米国) (en_US)
スロットで複数の値を使用する	英語 (米国) (en_US)
会話内のランタイムヒントによるスロット値の認識の向上	英国英語 (en_GB) 英語 (米国) (en_US)

機能	サポートされている言語と口ケール
会話中のスペルスタイルによるスロット値のキャプチャ	英語 (オーストラリア) (en_AU) 英国英語 (en_GB) 英語 (米国) (en_US)
信頼度スコアを使用した会話精度の向上	英国英語 (en_GB) 英語 (米国) (en_US)
サードパーティー ASR のみ (Deepgram)	オランダ語 (ベルギー) (nl_BE) フランス語 (ベルギー) (fr_BE)

機能	サポートされている言語とロケール
サードパーティー TTS (ElevenLabs) でのみ	アフリカ人 (南アフリカ) (af_ZA) ブルガリア (ブルガリア) (bg_BG) クロアチア語 (クロアチア) (hr_HR) 英語 (スコットランド) (en_AB) エストニア語 (エストニア) (et_ET) Farsi (イラン) (fa_IR) ヘブライ語 (イスラエル) (he_IL) ハンガリー (ハンガリー) (hu_HU) インドネシア語 (インドネシア) (id_ID) ラトビア語 (ラトビア) (lv_LV) リトアニア語 (リトアニア) (lt_LT) マレー語 (マレーシア) (ms_MY) セルビア (セルビア) (sr_RS) スロバキア (スロバキア) (sk_SK) スロベニア語 (スロベニア) (sl_SI)

リージョン

Amazon Lex V2 が利用可能な AWS リージョンのリストについては、の「AWS リージョンとエンドポイント」を参照してください AWS 全般のリファレンス。

Amazon Lex V2 の開始方法

概要:

Amazon Lex V2 を初めて使用する場合は、まず Amazon Lex V2 のコア概念 をお読みいただくことをお勧めします。

前提条件チェックリスト

最初の Amazon Lex V2 ボットの構築を開始する前に、以下があることを確認してください。

- AWS アカウント – アクティブな AWS アカウントが必要です。お持ちでない場合は、無料で作成し、Amazon Lex V2 の 1 か月あたり 10,000 件のテキストリクエストと 5,000 件の音声リクエストを含む AWS 無料利用枠を利用できます。詳細なセットアップ手順については、「」を参照してくださいAWS にサインアップする。
- アクセス許可を持つ IAM ユーザー – 適切な Amazon Lex V2 アクセス許可を持つ IAM ユーザーを作成します。開始するための管理者ユーザーを作成する手順を説明します。IAM アクセス許可とポリシーの包括的な情報については、「」を参照してくださいAmazon Lex V2 のための Identity and Access Management。
- サポートされているブラウザ – 最新のウェブブラウザ (Chrome、Firefox、Safari、または Edge) を使用して Amazon Lex V2 コンソールにアクセスします。
- 時間コミットメント – エクスペリエンスレベルに応じて、開始方法の演習が完了するまでに 30～60 分かかります。

コスト見積もり: このガイドの演習では、AWS 無料利用枠リソースを使用します。無料利用枠の制限内であれば、料金は発生しません。それ以外の場合、通常、すべての演習を完了するためのコストは 1 USD 未満です。

Amazon Lex V2 の概要 V2

Amazon Lex V2 を使用する前に、AWS アカウントを作成して AWS アカウント ID を取得する必要があります。Amazon Lex V2 を使用するために必要なアクセス許可を付与するユーザーアカウントと IAM ロールを作成します。 V2

アカウントを作成したら、AWS コンソールまたは API で Amazon Lex V2 の使用を開始できます。

Amazon Lex V2 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 V2 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。アプリケーションを構築するには、SDK を使用することをお勧めします。
- AWS CLI — AWS CLI を使用して、コードを記述しなくても Amazon Lex V2 の機能にアクセスできます。

ステップ 1: AWS アカウントをセットアップし、管理者ユーザーを作成する

Amazon Lex V2 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. AWS にサインアップする
2. IAM ユーザーの作成

AWS にサインアップする

すでに AWS アカウントをお持ちの場合は、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、Amazon Lex V2 を含む AWS 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。請求されるのは、使用したサービスの料金のみです。

Amazon Lex V2 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 の新規のお客様は、Amazon Lex V2 を無料で使い始めることができます。 V2 詳細については、「AWS 無料利用枠」を参照してください。

すでに AWS アカウントを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、次に説明する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

- AWS アカウントを作成するには、AWS ナレッジセンターの「新しい AWS アカウントを作成してアクティブ化する方法」を参照してください。

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。

IAM ユーザーの作成

Amazon Lex V2 などの AWS のサービスでは、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ます。

Amazon Lex V2 のアカウントにアクセスするための IAM ユーザーアカウントを作成します。 V2 IAM ユーザーの作成に関する包括的なガイダンスについては、IAM ユーザーガイドの「IAM の開始方法」を参照してください。

- AWS Identity and Access Management (IAM) を使用して IAM 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 IAM ユーザーに管理権限を付与する

これで、特別な URL と IAM ユーザーの認証情報を使用して AWS 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 IAM 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 を使用して AWS マネジメントコンソール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、IAM ユーザーガイド を参照してください。

一般的なセットアップの問題

アカウントのセットアップ中に問題が発生した場合の一般的な解決策は次のとおりです。

- アクセス許可拒否エラー – IAM ユーザーに、必要な Amazon Lex V2 アクセス許可があることを確認します。開始するには、管理者権限が推奨されます。Amazon Lex V2 のアクセス許可の詳細については、このガイドの「セキュリティ」セクションを参照してください。

- コンソールアクセスの問題 – ルート AWS コンソールではなく、アカウントの正しい IAM ユーザー URL でサインインしていることを確認してください。IAM のベストプラクティスの詳細については、このガイドの「セキュリティ」セクションを参照してください。
- リージョンの可用性 – Amazon Lex V2 は、一部の AWS リージョンでは使用できません。Amazon Lex エンドポイントとクォータをチェックして、任意のリージョンでの可用性を確認します。サポートされている口ケールの詳細については、「」を参照してくださいAmazon Lex V2 でサポートされている言語と口ケール。
- 無料利用枠の制限 – 使用量をモニタリングして、無料利用枠の制限である 1 か月あたり 10,000 件のテキストリクエストと 5,000 件の音声リクエストの範囲内にとどまります。料金の詳細については、Amazon Lexの料金」を参照してください。
- クォータの制限 – サービスの制限が発生した場合は、現在のクォータを確認し、必要に応じて引き上げをリクエストします。詳細については、「ガイドラインとベストプラクティス」を参照してください。

トラブルシューティングに関するその他のヘルプについては、このガイドの「モニタリング」セクションで、エラーログとデバッグに関する情報を参照してください。

プログラマチックアクセス権を付与する

アクセスキー ID とシークレットアクセスキーを取得する方法については、AWS 全般のリファレンスの「AWS 認証情報の理解と取得」を参照してください。

開発者環境のセットアップ

開発者向け: AWS CLI または SDKs を使用する場合は、開発環境も設定する必要があります。AWS CLI をインストールし、アクセスキーを使用して設定します。詳細な手順については、「AWS CLI のインストール」を参照してください。SDK の例については、「」を参照してくださいボットの例。

次のステップ

ステップ 2: 開始方法 (コンソール)

ステップ 2: 開始方法 (コンソール)

コンソールの概要

Amazon Lex V2 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。ロールとエクスペリエンスレベルに最も適したラーニングパスを選択します。

学習パスを選択する

ロールとエクスペリエンスレベルに最も適したラーニングパスを選択します。

- No-Code Path (ビジネスユーザー) – コンソールインターフェイスのみを使用してボットを構築するビジネスアナリスト、製品マネージャー、非技術ユーザーに最適です。
  - 以下から開始します。 クイックスタート: 5 分でチャットボットを作成する
  - 次に、以下を試してください。 演習 1: テンプレートからチャットボットを作成する
  - 以下について説明します。 開始方法のベストプラクティス
- 開発者パス – Amazon Lex V2 をアプリケーションと統合し、APIs を使用し、カスタムビジネスロジックを実装したいソフトウェア開発者に最適です。
  - 以下から開始します。 演習 1: テンプレートからチャットボットを作成する
  - アドバンスト: 演習 3: 高度なカスタマーサービスチャットボットを構築する
  - 次に、以下について調べます。 AWS Lambda関数を Amazon Lex V2 ボットに統合する
  - 確認: ボットの例
  - アドバンスト: API 統合と SDK の使用
- エンタープライズパス – セキュリティ、コンプライアンス、スケーラビリティの要件を備えた大規模なデプロイを計画するアーキテクトやエンジニア向けに設計されています。
  - 以下から開始します。 演習 1: テンプレートからチャットボットを作成する
  - エンタープライズの例: 演習 3: 高度なカスタマーサービスチャットボットを構築する
  - レビュー: セキュリティと IAM のベストプラクティス
  - 以下について調べます。 グローバルレジリエンシーを使用した他のリージョンへのボットのデプロイ
  - 計画: モニタリング、ログ記録、コスト最適化

基本的な演習

選択したパスに関係なく、以下の基本的な演習から始めることをお勧めします。

- 演習 1 — テンプレートを使用して Amazon Lex V2 ボットを作成します。これは、必要なすべてのボット設定を提供する事前定義されたボットです。最低限の作業を行うだけでエンドツーエンドのセットアップをテストできます。
- 演習 2 — クライアントアプリケーションと Amazon Lex V2 ボットの間で送信される JSON 構造を確認します。
- 演習 3 — 注文管理、インテリジェントなアップセリング、リード生成、パーソナライズされたカスタマーエクスペリエンスのための AI を活用した機能など、エンタープライズレベルの機能を示す高度なカスタマーサービスチャットボットを構築します。

クイックスタート: 5 分でチャットボットを作成する

このクイックスタートガイドは、構築済みのテンプレートを使用して、わずか 5 分で作業中の Amazon Lex V2 チャットボットを作成するのに役立ちます。すぐにテストして二ーズに合わせてカスタマイズできる機能的なチャットボットがあります。

ステップ 1: テンプレートを選択する

Amazon Lex V2 には、一般的なユースケース用に構築済みのテンプレートがいくつか用意されています。

- カスタマーサポートに関するよくある質問 – カスタマーサービスに関する一般的な問い合わせへの対応
- 予約予約 – 予約のスケジュールと管理
- 注文ステータス – 注文情報と配送ステータスを確認する
- IT ヘルプデスク – テクニカルサポートとトラブルシューティングを提供する

テンプレートを選択するには

1. https://console.aws.amazon.com/lexv2/ で Amazon Lex V2 コンソールを開きます。
2. [ボットの作成] を選択します。
3. テンプレートで開始を選択します。
4. このクイックスタートのカスタマーサポートに関するよくある質問テンプレートを選択します。

5. などのボット名を入力しますMyFirstChatbot。
6. [作成] を選択します。

ステップ 2: クイックカスタマイズ (オプション)

テンプレートは事前設定されていますが、特定のニーズに合わせてすばやくカスタマイズできます。

チャットボットをカスタマイズするには

1. ボットの概要で、事前設定されたインテントを確認します。
2. GetAccountInfo など、変更するインテントを選択します。
3. 顧客がリクエストをフレーズする方法に一致する独自のサンプル発話を追加します。
4. ブランドの音声に合わせてレスポンスメッセージを更新します。
5. [インテントの保存] を選択します。

ステップ 3: ボットをテストする

組み込みのテストコンソールを使用して、チャットボットをすぐにテストします。

チャットボットをテストするには

1. ビルド を選択してボットをコンパイルします。
2. ビルドが完了するまで待ちます (通常は 1~2 分) 。
3. 右側のテストコンソールで、次のように入力します。 **I need help with my account**
4. Enter キーを押し、ボットのレスポンスを確認します。
5. 他のフレーズを試して、ボットの理解度をテストします。

ステップ 4: ボットをデプロイする

チャットボットのレスポンスに満足したら、使用できるようにデプロイします。

チャットボットをデプロイするには

1. ボットアクションメニューから発行を選択します。
2. バージョンの作成を選択して新しいバージョンを作成します。
3. バージョンを指すエイリアス (「Production」など) を作成します。

4. 統合方法を選択します。

次のステップ

お疲れ様でした。これで、Amazon Lex V2 チャットボットが動作するようになりました。この後は、以下の内容を実行できます。

- アシスト付き NLU を有効にする – AI を活用した自然言語処理による理解を向上させます。
- 目的の追加 – 追加のユースケースでチャットボットの機能を拡張します。
- Lambda との統合 – ビジネスロジックと外部システム統合を追加します。「AWS Lambda関数を Amazon Lex V2 ボットに統合する」を参照してください。
- モニタリングのセットアップ – 使用状況とパフォーマンスを追跡します。「Amazon CloudWatch によるオペレーションメトリクスの測定」を参照してください。
- Learn Advanced Features – 会話フロー管理、マルチターンダイアログ、コンテキスト切り替えについて説明します。

詳細なチュートリアルについては、「」に進みます演習 1: テンプレートからチャットボットを作成する。

演習 1: テンプレートからチャットボットを作成する

この演習では、最初の Amazon Lex V2 チャットボットを作成し、Amazon Lex V2 コンソールでテストします。 V2 この演習では、E コマースの実用的で実際のユースケースを示す OrderFlowers テンプレートを使用します。

OrderFlowers ボットの例

OrderFlowers テンプレートを使用して、花の注文リクエストを処理できる Amazon Lex V2 チャットボットを作成します。この例では、企業がインテリジェントなチャットボットを使用して注文を自動化する方法を示します。ボットの構造の詳細については、Amazon Lex V2 のコア概念 を参照してください。

- インテント – ボットには 1 つの主要なインテントが含まれています。
  - `OrderFlowers` - 花のタイプ、集荷日、集荷時間を収集して、花の注文リクエストを処理します
- スロットタイプ – ボットは、一般的なデータ形式を自動的に認識して処理する組み込みスロットタイプを使用します。

  - AMAZON.Date - 「明日」、「次の金曜日」、「3 月 15 日」などの日付を認識します
  - AMAZON.Time - 「午後 2 時」、「正午」、「3 か月を過ぎた四半期」などの時間を認識します
  - `FlowerTypes` (カスタム) - 「バラ」、「チューリップ」、「ライリー」などの特定の花の種類
- スロット – OrderFlowers インテントでは、ボットが花の注文を満たす前に次の情報が必要です。
  - `FlowerType` (FlowerTypes カスタムタイプ) - 注文する花のタイプ
  - `PickupDate` (AMAZON.Date タイプ) - 花を拾うタイミング
  - `PickupTime` (AMAZON.Time 型) - 花を拾う時間
- サンプル発話 – 次のサンプル発話は、ユーザーが花の注文をリクエストする自然な方法を示しています。
  - 「花をピックアップしたい」
  - 「花を注文したい」
  - 「花を集荷できますか？」
  - 「花を購入する必要があります」
- 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?」

ボットを作成する

Amazon Lex V2 ボット (コンソール) を作成するには

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/lex/ で Amazon Lex コンソールを開きます。
2. [ボットの作成] を選択します。
3. 作成方法 で、例から始める を選択します。
4. ボットの例 セクションで、リストから OrderFlowers を選択します。
5. [ボットの設定] セクションで、ボットに名前とオプションの説明を与えます。新しい名前は アカウント内で一意である必要があります。

6. アクセス許可 セクションで、基本的な Amazon Lex アクセス許可で新しいロールを作成する を選択します。これにより、Amazon Lex V2 がボットを実行するために必要なアクセス許可を持つ AWS Identity and Access Management (IAM) ロールが作成されます。
7. 児童オンラインプライバシー保護法 (COPPA) セクションで、適切な選択を行います。
8. セッションタイムアウト および アドバンスト設定 セクションで、デフォルトのままにします。
9. [次へ] を選択します。Amazon Lex V2 がボットを作成します。

ボットに言語を追加する

ボットを作成したら、ボットがサポートする言語を 1 つ以上追加する必要があります。言語には、ボットがユーザーとの会話に使用するインテント、スロットタイプ、スロットが含まれます。

ボットに言語を追加するには

1. 言語 セクションで、サポートされている言語を選択し、説明を追加します。
2. デフォルトが設定されている 音声インタラクション および インテント分類信頼スコアのしきい値 フィールドを離れます。
3. [完了] を選択して、ボットに言語を追加します。

ボットをテストする

完了 を選択した後、コンソールがインテント工ディタを開きます。インテント工ディタを使用して、ボットが使用するインテントを調べることができます。ボットの確認が終わったら、ボットをテストできます。

OrderFlowers ボットをテストするには

1. ページの上部で、[構築] を選択します。ボットが構築されるまで待ってください。
2. ビルドが完了したら、[テスト] を選択してテストウィンドウを開きます。
3. ボットをテストします。例えば、「花をピックアップしたい」などのサンプル発話で会話を始めます。

NLU を有効にして理解を高める

作業中のチャットボットができたので、アシスト付き NLU で強化して、インテント認識とスロット解決を改善しましょう。アシスト NLU は、大規模言語モデル (LLMs) を使用して、トレーニング例とは異なるフレーズを使用している場合でも、ユーザーのリクエストをよりよく理解します。

アシスト NLU を有効にするには

1. Amazon Lex V2 コンソールで、ボットの設定に移動します。
2. 左側のナビゲーションペインで、ボット設定を選択します。
3. アシスト NLU で、有効化 を選択します。
4. 保存を選択して変更を適用します。
5. ボットを再度構築して、アシスト付き NLU の機能強化を適用します。

改善をテストする: テストコンソールでこれらのバリエーションを試して、アシスト NLU がさまざまなフレーズをどのように処理するかを確認してください。

- 「バラを購入したい」(OrderFlowers インテントをトリガーして FlowerType をキャプチャする必要があります)
- 「明日花を配達できますか？」 (OrderFlowers インテントをトリガーし、PickupDate をキャプチャする必要があります)
- 「午後 3 時の集荷にチューリップが必要です」(OrderFlowers インテントをトリガーし、FlowerType と PickupTime をキャプチャする必要があります)

チャットボットが、明示的なサンプル発話として追加することなく、これらの自然なバリエーションをどのように理解できるかに注目してください。これは、AI を使用して自然言語の理解を向上させる Assisted NLU を利用しています。

次の手順

テンプレートを使用して最初のボットを作成したので、コンソールを使用して独自のボットを作成できます。カスタムボットの作成手順、およびボットの作成の詳細については、Amazon Lex V2 ボットの使用 を参照してください。

演習 2: 会話フローを確認する

この演習では、クライアントアプリケーションと、演習 1: テンプレートからチャットボットを作成する で作成した Amazon Lex V2 ボットの間で送信される JSON 構造を確認します。会話では、RecognizeText オペレーションを使用して JSON 構造を生成します。RecognizeUtterance は、レスポンスの中の HTTP ヘッダーと同じ情報を返します。

JSON 構造は、会話のターンごとに分割されます。ある ターン はクライアントアプリケーションからのリクエストで、ボットからの応答です。

ターン 1

会話の最初のターンでは、クライアントアプリケーションはボットとの会話を開始します。リクエストの URI と本文の両方でリクエストに関する情報が提供されています。

```
POST /bots/botId/botAliases/botAliasId/botLocales/localeId/sessions/sessionId/text
 HTTP/1.1
Content-type: application/json

{
    "text": "I would like to order flowers"
}
```

- URI は、クライアントアプリケーションが通信しているボットを識別します。また、ユーザーとボットの間の特定の会話を識別するクライアントアプリケーションによって生成されたセッション識別子も含まれます。
- リクエストの本文には、ユーザーがクライアントアプリケーションに入力したテキストが含まれます。この場合、テキストのみが送信されますが、アプリケーションはリクエスト属性やセッション状態などの追加情報を送信できます。詳細については、RecognizeText オペレーションを参照してください。

送信元textでは、Amazon Lex V2 はフラワーを注文するユーザーの意図を検出します。Amazon Lex V2 は、インテントのスロットの 1 つ (FlowerType) とそのスロットのプロンプトの 1 つを選択し、以下のレスポンスをクライアントアプリケーションに送信します。クライアントはユーザーへのレスポンスを表示しています。

```
{
    "interpretations": [
```

```
        {
            "intent": {
                "confirmationState": "None",
                "name": "OrderFlowers",
                "slots": {
                    "FlowerType": null,
                    "PickupDate": null,
                    "PickupTime": null
                },
                "state": "InProgress"
            },
            "nluConfidence": {
                "score": 0.95
            }
        },
        {
            "intent": {
                "name": "FallbackIntent",
                "slots": {}
            }
        }
    ],
    "messages": [
        {
            "content": "What type of flowers would you like to order?",
            "contentType": "PlainText"
        }
    ],
    "sessionId": "bf445a49-7165-4fcd-9a9c-a782493fba5c",
    "sessionState": {
        "dialogAction": {
            "slotToElicit": "FlowerType",
            "type": "ElicitSlot"
        },
        "intent": {
            "confirmationState": "None",
            "name": "OrderFlowers",
            "slots": {
                "FlowerType": null,
                "PickupDate": null,
                "PickupTime": null
            },
            "state": "InProgress"
        },
```

```
        "originatingRequestId": "9e8add70-4106-4a10-93f5-2ce2cb959e5f"
    }
}
```

ターン 2

ターン 2 では、ユーザーは、FlowerType スロットを満たす値を持つ、ターン 1 の Amazon Lex V2 ボットからのプロンプトに応答します。

```
{
    "text": "1 dozen roses"
}
```

ターン 2 の応答は、FlowerType スロットがいっぱいになり、次のスロット値を引き出すプロンプトが表示されます。

```
{
    "interpretations": [
        {
            "intent": {
                "confirmationState": "None",
                "name": "OrderFlowers",
                "slots": {
                    "FlowerType": {
                        "value": {
                            "interpretedValue": "dozen roses",
                            "originalValue": "dozen roses",
                            "resolvedValues": []
                        }
                    },
                    "PickupDate": null,
                    "PickupTime": null
                },
                "state": "InProgress"
            },
            "nluConfidence": {
                "score": 0.98
            }
        },
        {
```

```
            "intent": {
                "name": "FallbackIntent",
                "slots": {}
            }
        }
    ],
    "messages": [
        {
            "content": "What day do you want the dozen roses to be picked up?",
            "contentType": "PlainText"
        }
    ],
    "sessionId": "bf445a49-7165-4fcd-9a9c-a782493fba5c",
    "sessionState": {
        "dialogAction": {
            "slotToElicit": "PickupDate",
            "type": "ElicitSlot"
        },
        "intent": {
            "confirmationState": "None",
            "name": "OrderFlowers",
            "slots": {
                "FlowerType": {
                    "value": {
                        "interpretedValue": "dozen roses",
                        "originalValue": "dozen roses",
                        "resolvedValues": []
                    }
                },
                "PickupDate": null,
                "PickupTime": null
            },
            "state": "InProgress"
        },
        "originatingRequestId": "9e8add70-4106-4a10-93f5-2ce2cb959e5f"
    }
}
```

ターン 3

ターン 3 では、ユーザーは、PickupDate スロットを満たす値を持つ、ターン 2 の Amazon Lex V2 ボットからのプロンプトに応答します。

```
{
    "text": "next monday"
}
```

ターン 3 の応答で FlowerType と PickupDate スロット両方が入力され、次のスロット値を誘発するプロンプトが表示されます。

```
{
    "interpretations": [
        {
            "intent": {
                "confirmationState": "None",
                "name": "OrderFlowers",
                "slots": {
                    "FlowerType": {
                        "value": {
                            "interpretedValue": "dozen roses",
                            "originalValue": "dozen roses",
                            "resolvedValues": []
                        }
                    },
                    "PickupDate": {
                        "value": {
                            "interpretedValue": "2022-12-28",
                            "originalValue": "next monday",
                            "resolvedValues": [
                                "2021-01-04"
                            ]
                        }
                    },
                    "PickupTime": null
                },
                "state": "InProgress"
            },
            "nluConfidence": {
                "score": 1.0
            }
        },
        {
            "intent": {
                "name": "FallbackIntent",
                "slots": {}
```

```
            }
        }
    ],
    "messages": [
        {
            "content": "At what time do you want the 1 dozen roses to be picked up?",
            "contentType": "PlainText"
        }
    ],
    "sessionId": "bf445a49-7165-4fcd-9a9c-a782493fba5c",
    "sessionState": {
        "dialogAction": {
            "slotToElicit": "PickupTime",
            "type": "ElicitSlot"
        },
        "intent": {
            "confirmationState": "None",
            "name": "OrderFlowers",
            "slots": {
                "FlowerType": {
                    "value": {
                        "interpretedValue": "dozen roses",
                        "originalValue": "dozen roses",
                        "resolvedValues": []
                    }
                },
                "PickupDate": {
                    "value": {
                        "interpretedValue": "2021-01-04",
                        "originalValue": "next monday",
                        "resolvedValues": [
                            "2021-01-04"
                        ]
                    }
                },
                "PickupTime": null
            },
            "state": "InProgress"
        },
        "originatingRequestId": "9e8add70-4106-4a10-93f5-2ce2cb959e5f",
        "sessionAttributes": {}
    }
}
```

ターン 4

ターン 4 では、ユーザーはインテントの最終スロット値、つまり花が拾われる時間を提供します。

```
{
    "text": "5 in the evening"
}
```

レスポンスでは、Amazon Lex V2 はオーダーが正しいことを確認する確認プロンプトをユーザーに送信します。dialogAction は ConfirmIntent に設定され、confirmationState は None です。

```
{
    "interpretations": [
        {
            "intent": {
                "confirmationState": "None",
                "name": "OrderFlowers",
                "slots": {
                    "FlowerType": {
                        "value": {
                            "interpretedValue": "dozen roses",
                            "originalValue": "dozen roses",
                            "resolvedValues": []
                        }
                    },
                    "PickupDate": {
                        "value": {
                            "interpretedValue": "2021-01-04",
                            "originalValue": "next monday",
                            "resolvedValues": [
                                "2021-01-04"
                            ]
                        }
                    },
                    "PickupTime": {
                        "value": {
                            "interpretedValue": "17:00",
                            "originalValue": "5 evening",
```

```
                        "resolvedValues": [
                            "17:00"
                        ]
                    }
                }
            },
            "state": "InProgress"
        },
        "nluConfidence": {
            "score": 1.0
        }
    },
    {
        "intent": {
            "name": "FallbackIntent",
            "slots": {}
        }
    }
],
"messages": [
    {
        "content": "Okay, your dozen roses will be ready for pickup by 17:00 on
2021-01-04.  Does this sound okay?",
        "contentType": "PlainText"
    }
],
"sessionId": "bf445a49-7165-4fcd-9a9c-a782493fba5c",
"sessionState": {
    "dialogAction": {
        "type": "ConfirmIntent"
    },
    "intent": {
        "confirmationState": "None",
        "name": "OrderFlowers",
        "slots": {
            "FlowerType": {
                "value": {
                    "interpretedValue": "dozen roses",
                    "originalValue": "dozen roses",
                    "resolvedValues": []
                }
            },
            "PickupDate": {
                "value": {
```

```
                        "interpretedValue": "2021-01-04",
                        "originalValue": "next monday",
                        "resolvedValues": [
                            "2021-01-04"
                        ]
                    }
                },
                "PickupTime": {
                    "value": {
                        "interpretedValue": "17:00",
                        "originalValue": "5 evening",
                        "resolvedValues": [
                            "17:00"
                        ]
                    }
                }
            },
            "state": "InProgress"
        },
        "originatingRequestId": "9e8add70-4106-4a10-93f5-2ce2cb959e5f"
    }
}
```

ターン 5

最終ターンでは、ユーザーは確認プロンプトに応答します。

```
{
    "text": "yes"
}
```

レスポンスで、Amazon Lex V2 の送信は、confirmationState の設定によりインテントが満たされたことを示します。Confirmed と dialogAction を閉じてください。すべてのスロット値は、クライアントアプリケーションで使用できます。

```
{
    "interpretations": [
        {
            "intent": {
                "confirmationState": "Confirmed",
```

```
                "name": "OrderFlowers",
                "slots": {
                    "FlowerType": {
                        "value": {
                            "interpretedValue": "dozen roses",
                            "originalValue": "dozen roses",
                            "resolvedValues": []
                        }
                    },
                    "PickupDate": {
                        "value": {
                            "interpretedValue": "2021-01-04",
                            "originalValue": "next monday",
                            "resolvedValues": [
                                "2021-01-04"
                            ]
                        }
                    },
                    "PickupTime": {
                        "value": {
                            "interpretedValue": "17:00",
                            "originalValue": "5 evening",
                            "resolvedValues": [
                                "17:00"
                            ]
                        }
                    }
                },
                "state": "Fulfilled"
            },
            "nluConfidence": {
                "score": 1.0
            }
        },
        {
            "intent": {
                "name": "FallbackIntent",
                "slots": {}
            }
        }
    ],
    "messages": [
        {
            "content": "Thanks. ",
```

```
            "contentType": "PlainText"
        }
    ],
    "sessionId": "bf445a49-7165-4fcd-9a9c-a782493fba5c",
    "sessionState": {
        "dialogAction": {
            "type": "Close"
        },
        "intent": {
            "confirmationState": "Confirmed",
            "name": "OrderFlowers",
            "slots": {
                "FlowerType": {
                    "value": {
                        "interpretedValue": "dozen roses",
                        "originalValue": "dozen roses",
                        "resolvedValues": []
                    }
                },
                "PickupDate": {
                    "value": {
                        "interpretedValue": "2021-01-04",
                        "originalValue": "next monday",
                        "resolvedValues": [
                            "2021-01-04"
                        ]
                    }
                },
                "PickupTime": {
                    "value": {
                        "interpretedValue": "17:00",
                        "originalValue": "5 evening",
                        "resolvedValues": [
                            "17:00"
                        ]
                    }
                }
            },
            "state": "Fulfilled"
        },
        "originatingRequestId": "9e8add70-4106-4a10-93f5-2ce2cb959e5f"
    }
}
```

演習 3: 高度なカスタマーサービスチャットボットを構築する

この高度な演習では、e コマース会社向けの高度なカスタマーサービスチャットボットを構築します。このボットは、注文管理、インテリジェントアップセリング、リード生成、収益最適化などのエンタープライズレベルの機能を示しています。チャットボットは AI を活用した機能を使用して、パーソナライズされたカスタマーエクスペリエンスを提供し、ビジネスの成長を促進します。

SmartCommerce カスタマーサービスボットの概要

SmartCommerce カスタマーサービスボットは、収益機会を最大化しながら、複雑な顧客とのやりとりを処理するように設計されています。この例では、インテリジェントな会話管理を通じて販売拡大を推進しながら、企業がカスタマーサービスを自動化する方法を示します。

- カスタムインテント – ボットには、包括的なカスタマーサービスのための複数のカスタムインテントが含まれています。
  - `CheckOrderStatus` - 注文ステータス情報を検証して提供します
  - `ProcessReturn` - 戻りリクエストと交換を処理します
  - `UpsellProducts` - 追加の製品やサービスを推奨します
  - `CaptureLeadInfo` - リード生成の顧客情報を収集します
  - `ScheduleCallback` - カスタマーサービスのコールバックを予約します
- 組み込みインテント – 一般的なやり取りに Amazon Lex V2 組み込みインテントを活用します。
  - AMAZON.HelpIntent - ヘルプとガイダンスを提供します
  - AMAZON.CancelIntent - キャンセルリクエストを処理します
  - AMAZON.StopIntent - 会話を正常に終了します
- カスタムスロットタイプ – ビジネス固有のデータ専用のスロットタイプ:
  - `ProductCategories` - 電子製品、衣類、ホーム、書籍、スポーツ
  - `ReturnReasons` - 欠陥、サイズが間違っている、考え方が変わった、説明どおりではない
  - `ContactPreferences` - E メール、SMS、電話、アプリ内通知
  - `CustomerTiers` - 青銅、シルバー、ゴールド、プラチナ
- 組み込みスロットタイプ – 一般的なデータ形式に Amazon Lex V2 組み込みスロットを使用します。
  - AMAZON.Date - 配信日とコールバックスケジューリングの場合

- AMAZON.Time - コールバック時間と配信ウィンドウの場合
- AMAZON.EmailAddress - お客様の E メール収集用
- AMAZON.PhoneNumber - お客様の電話番号の場合
- AMAZON.Number - 注文番号と数量の場合

詳細なインテント設定

CheckOrderStatus のインテント

このインテントは、注文の検証とステータスの問い合わせを処理し、アップセリングの機会を特定しながら、顧客にリアルタイムの注文情報を提供します。

- 必要なスロット:
  - `OrderNumber` (AMAZON.Number) - お客様の注文番号
  - `CustomerEmail` (AMAZON.EmailAddress) - 検証用の E メール
- サンプル発話:
  - 「注文 {OrderNumber} のステータスは何ですか」
  - 「注文番号 {OrderNumber} を確認する必要があります」
  - 「パッケージの場所」
  - 「注文の追跡 {OrderNumber}」
  - 「注文はまだ配送済みです」
- AI を活用した機能:
  - アシスト NLU を使用して、「パッケージ」、「モノ」、「配信」などのバリエーションを理解します
  - 自然な音声パターンから注文番号を自動的に抽出します

UpsellProducts のインテント

このインテントは、顧客履歴と現在のインタラクションコンテキストに基づいて補完的な製品とプレミアムサービスをレコメンデーションすることで、収益機会を積極的に特定します。

- 必要なスロット:
  - `ProductCategory` (ProductCategories) - 目的のカテゴリ
  - `CustomerTier` (CustomerTiers) - 顧客ロイヤルティレベル

  - Budget (AMAZON.Number) - 顧客の予算範囲
- サンプル発話:
  - 「関連製品を表示する」
  - 「これで他に何が起こるか」
  - 「何か取引はありますか」
  - 「{ProductCategory} アイテムに興味がある」
  - 「What's on sale today」
- 収益最適化機能:
  - 顧客層に基づく動的料金
  - 購入履歴を使用してパーソナライズされたレコメンデーション
  - 即時購入を促進する期間限定オファーの作成

CaptureLeadInfo インテント

このインテントは、リード生成のために顧客情報を体系的に収集し、アグリゲータに販売したり、ターゲットを絞ったマーケティングキャンペーンに使用したりできる貴重なデータを作成します。

- 必要なスロット:
  - CustomerFirstName (AMAZON.FirstNameA) - パーソナライゼーションのためのお客様の名
  - CustomerLastName (AMAZON.LastName) - 販売履歴検索のお客様の姓
  - CustomerEmail (AMAZON.EmailAddress) - E メールアドレス
  - CustomerPhone (AMAZON.PhoneNumber) - 電話番号
  - InterestCategory (ProductCategories) - 製品の関心
  - ContactPreference (ContactPreferences) - 希望する問い合わせ方法
- サンプル発話:
  - 「新製品について知りたい」
  - 「更新にサインアップする」
  - 「{InterestCategory} に興味がある」
  - 「売上について知らせてください」
  - 「メーリングリストに参加したい」
  - 「私の名前は {CustomerFirstName} {CustomerLastName}」

  - 「私は {CustomerFirstName} {CustomerLastName} で、パーソナライズされたレコメンデーションが必要です」
- リード生成機能:
  - CRM 統合用の構造化データ収集
  - インタラクション品質に基づくリードスコアリング
  - 自動リード認定とルーティング
  - アグリゲータ販売のデータ形式
- 販売履歴の統合:
  - S3-stored購入履歴テーブルでの顧客名検索
  - パーソナライズされたレコメンデーションの購入パターン分析
  - クロスカテゴリ購入動作の識別
  - ターゲットオファーの季節的な購入傾向分析
  - 階層ベースの料金の顧客有効期間値の計算

マルチターン会話フロー

SmartCommerce ボットは、複数のインテントにまたがる複雑な複数ターンの会話を処理し、インタラクション全体でコンテキストを維持するように設計されています。これにより、収益機会を最大化しながら、自然で人間のようなエクスペリエンスが生まれます。

会話フローの例

ターン 1-3: 注文ステータスチェック

```
Customer: "Hi, I need to check on my order"
Bot: "I'd be happy to help you check your order status. Can you provide your order
 number?"
Customer: "It's order 12345"
Bot: "Thank you. To verify your identity, can you provide the email address associated
 with this order?"
Customer: "john.doe@email.com"
Bot: "Perfect! Order #12345 is currently being prepared for shipment. You should
 receive it by tomorrow.
     Since you're a Gold tier customer, would you like to hear about some exclusive
 deals on related items?"
```

ターン 4～6: アップセリングの機会

```
Customer: "Sure, what do you have?"
Bot: "Based on your Electronics purchase, I can offer you a 20% discount on premium
 accessories.
     What's your budget range for additional items?"
Customer: "Maybe around $50"
Bot: "Excellent! I have a wireless charging pad normally $60, now $48 with your Gold
 discount.
     Plus, if you add it to your current order, shipping is free. Should I add this to
 your cart?"
```

Turn 7-11: 販売履歴ルックアップによるリード生成

```
Customer: "Not right now, but I'd like to hear about future deals"
Bot: "Absolutely! I can set you up to receive personalized offers. What's your first
 name?"
Customer: "John"
Bot: "Thanks John! And your last name so I can look up your purchase history for better
 recommendations?"
Customer: "Smith"
Bot: "Perfect! Let me check your purchase history... I see you've bought electronics
 and home items before.
     Based on your past purchases, I can offer you exclusive deals on smart home
 accessories.
     How would you prefer to be contacted?"
Customer: "Email is fine"
Bot: "Great! I'll make sure john.doe@email.com gets personalized deals based on your
 purchase history.
     You're all set for our premium recommendations program!"
```

コンテキスト管理機能

- セッションの永続性: 複数のインテントにわたって顧客情報を維持します
- インテントチェイニング: 注文チェック、アップセリング、リードキャプチャ間のシームレスな移行
- 動的パーソナライゼーション: 顧客層と購入履歴に基づいてオファーを調整します

- 収益追跡: アップセルの成功率とリード品質スコアをモニタリングします

AI を活用した機能

このチャットボットは、複数の AI 機能を活用して、ビジネスの成長を推進しながら、インテリジェントでパーソナライズされたカスタマーサービスを提供します。

アシスト付き NLU 実装

アシスト NLU は、大規模言語モデルを使用して、非標準フレーズを使用したり、複数のリクエストを 1 回の発話で組み合わせたりしても、顧客の意図を理解します。

- 自然言語の理解:
  - 「私のモノはまだ到着していません」→ CheckOrderStatus インテント
  - 「このジャンクを返したい」→ ProcessReturn インテント
  - 「Show me what else you got」→ UpsellProducts インテント
- マルチインテント認識:
  - 「注文 12345 を確認して契約にサインアップする」→ CheckOrderStatus + CaptureLeadInfo

生成スロットの解像度

AI を使用して、トレーニングデータに完全に一致することなく、複雑な自然言語入力からスロット値を抽出します。

- スマート抽出:
  - 「先週、いくつかの電子機器を購入しました。注文番号は 12345 のようなものだった」→ OrderNumber: 12345、ProductCategory: Electronics
  - 「携帯電話で電話 555-123-4567 または E メール john@company.com」→ 電話: 555-123-4567、E メール: john@company.com

感情分析の統合

会話を通じて顧客の感情をモニタリングし、アプローチを調整し、必要に応じてエスカレーションします。

- 感情ベースのルーティング:
  - ポジティブセンチメント → 積極的なアップセリングアプローチ

- 中立的なセンチメント → 穏やかにアップセリングする標準サービス
- 否定的な感情 → 問題解決に集中し、アップセリングを最小限に抑える

注文の検証および確認システム

ボットには、顧客 ID の検証、注文の詳細の確認、複数のチャネルを介した確認の送信を行う包括的な確認システムが含まれています。

多要素検証プロセス

- プライマリ検証:
  - 注文番号の検証
  - E メールアドレスの確認
  - 支払い方法の最後の 4 桁 (機密性の高いオペレーションの場合)
- セカンダリ検証:
  - 配送先住所の確認
  - 購入日の検証
  - 製品詳細の検証

マルチチャネル確認システム

ボットは、顧客の好みとアクションタイプに基づいて、複数のチャネルを通じて確認を自動的に送信します。

- E メールの確認:
  - 追跡情報を含む注文ステータスの更新
  - 前払い配送ラベルを使用した返品承認
  - 配信の詳細を含む購入確認のアップセル
  - ウェルカムオファーによるリードキャプチャの確認
- SMS 確認:
  - 即時注文ステータスの更新
  - 時間枠を含む配信通知
  - 関心のある顧客向けのフラッシュセールアラート
- アプリ内通知:

- 注文のリアルタイム更新
- パーソナライズされた製品のレコメンデーション
- ロイヤルティプログラムの更新

POS (POS) 統合

チャットボットは POS システムと直接統合して、トランザクションを処理し、割引を適用し、インベントリをリアルタイムで更新します。

リアルタイムトランザクション処理

- アップセルトランザクションフロー:

```
1. Customer accepts upsell offer
2. Bot validates inventory availability
3. Bot applies customer-tier discount
4. Bot processes payment using stored payment method
5. Bot updates order with additional items
6. Bot sends confirmation via preferred channel
7. Bot updates customer profile with purchase data
```

- 収益の追跡:
  - 顧客層別のアップセル変換率
  - インタラクションあたりの平均注文値の増加
  - チャットボットインタラクションへの収益属性
  - お客様のライフタイム価値への影響

動的料金エンジン

- 階層ベースの料金:
  - 青銅: 標準料金
  - シルバー: アップセルの 5% 割引
  - ゴールド: アップセルの 10～20% 割引
  - Platinum: 25% 割引 + 送料無料
- コンテキスト料金:

- 問題解決シナリオ: 満足度を維持するための追加の割引
- 価値の高い顧客: 排他的な料金階層
- 在庫クリアランス: 動きの遅い商品の積極的な割引

高度なリード生成システム

チャットボットには、複数の収益ストリームを通じて顧客データをキャプチャ、認定、収益化する高度なリード生成システムが含まれています。

マルチタッチリードキャプチャ戦略

- オポチュニスティックキャプチャ:
  - 注文ステータスチェック中：「類似製品の更新を希望しますか？」
  - 問題解決後：「製品の改善について通知できますか？」
  - アップセルのやり取り中：「このカテゴリの今後の取引について警告すべきでしょうか？」
- 付加価値キャプチャ:
  - 排他的なメンバー料金アクセス
  - 新製品の発売への早期アクセス
  - パーソナライズされた製品のレコメンデーション
  - 誕生日および記念日の特別オファー

自動リード認定

- スコアリング基準:
  - 購入履歴値: 0～25 ポイント
  - エンゲージメントレベル: 0～20 ポイント
  - 連絡先情報の完全性: 0～15 ポイント
  - 製品カテゴリの対象範囲: 0～15 ポイント
  - アップセルの試行に対するレスポンス: 0～25 ポイント
- リードカテゴリ:
  - ホットリード (80～100 ポイント): セールスチームの即時連絡先
  - ウォームリード (60～79 ポイント): 自動ナーチャシーケンス
  - Cold Leads (40～59 ポイント): 毎月のニュースレター

- 見込み (0～39 ポイント): 四半期ごとのプロモーション E メール

アグリゲータのリードデータ収益化

システム形式とパッケージは、サードパーティーのアグリゲータに販売されるデータをリードし、追加の収益ストリームを作成します。

- データパッケージ:
  - プレミアムパッケージ: 完全な連絡先情報、購入履歴、設定、エンゲージメントスコア
  - スタンダードパッケージ: 連絡先情報、基本設定、カテゴリの関心
  - ベーシックパッケージ: E メールアドレス、主な関心カテゴリ
- コンプライアンス機能:
  - 明示的な同意追跡による GDPR コンプライアンス
  - オプトアウトメカニズムへの CCPA コンプライアンス
  - E メールマーケティングの CAN-SPAM コンプライアンス
  - データ保持および削除ポリシー
- 収益の最適化:
  - リード品質スコアに基づく動的料金
  - プレミアムアグリゲータとの排他的データパートナーシップ
  - 高価値リードのリアルタイム入札

収益最適化戦略

すべてのインタラクションは、インテリジェントなアップセリング、クロスセリング、カスタマーライフタイム価値の最適化を通じて収益を最大化するように設計されています。

インテリジェントアップセリング戦略

- コンテキストアップセリング:
  - 注文ステータスチェック → "注文の準備ができました! 5 USD の速達配送を追加しますか？」
  - 返送リクエスト →「返送する代わりに、アップグレードされたモデルと交換しますか？」
  - 製品のお問い合わせ →「このアイテムは [補完製品] と完全にペアリング - バンドルして 15% を節約」
- 緊急ベースの販売:

- 「このオファーは 24 時間後に期限切れになります」
- 「この価格での在庫は 3 つだけです」
- 「フラッシュセールは午前 0 時に終了します」

クロスセリングの機会

- 製品エコシステムの販売:
  - 電子 → アクセサリ、保証、インストールサービス
  - 服 → アイテムのマッチング、ケア製品、スタイルサービス
  - ホーム商品 → 補完アイテム、メンテナンス製品、設計相談
- サービスアップセリング:
  - 延長保証と保護プラン
  - プレミアムカスタマーサポート階層
  - サブスクリプションサービスと自動配信
  - プロフェッショナルによるインストールとセットアップ

カスタマーライフタイムバリューの最適化

- ロイヤルティプログラムの統合:
  - チャットボットインタラクションに基づく自動階層アップグレード
  - アップセルオファーを利用するためのボーナスポイント
  - チャットボットのみの限定報酬と割引
- 保持戦略:
  - チャーンを防ぐためのプロアクティブ問題解決
  - 非アクティブなお客様向けのパーソナライズされたウィンバックオファー
  - 記念日とマイルストーンの祝賀オファー

実装手順

以下のstep-by-stepの手順に従って、収益最適化機能をすべて備えた高度なカスタマーサービスチャットボットを構築します。

前提条件とセットアップ要件

SmartCommerce ボットを構築する前に、必要な AWS アカウントのセットアップとアクセス許可があることを確認し、サービスに関する考慮事項を理解してください。

AWS アカウントとアクセス要件

- AWS アカウント: 請求が有効になっているアクティブな AWS アカウントが必要です。お持ちでない場合は、aws.amazon.com でサインアップしてください。
- IAM アクセス許可: AWS ユーザーまたはロールには、次のアクセス許可が必要です。
  - `lex:*` - ボットの作成と管理のための完全な Amazon Lex V2 アクセス許可
  - `iam:CreateRole` - ボットのサービスロールを作成するには
  - `iam:AttachRolePolicy` - サービスロールにポリシーをアタッチするには
  - `lambda:CreateFunction` - Lambda 統合の場合 (オプション)
  - `logs:CreateLogGroup` - Amazon CloudWatch Logs ログ記録用
- リージョンの選択: Amazon Lex V2 とターゲットオーディエンスをサポートする AWS リージョンを選択します。 V2 推奨されるリージョンは次のとおりです。
  - 米国東部 (バージニア北部) - us-east-1
  - 米国東部 (オレゴン) - us-west-2
  - ヨーロッパ (アイルランド) eu-west-1
  - アジアパシフィック (シドニー) - ap-southeast-2

サービスの制限とクォータ

Amazon Lex V2 には、ボットの開発とデプロイに影響する可能性のあるいくつかのサービス制限があります。

- ボットの制限:
  - アカウントあたり、リージョンあたり最大 100 個のボット
  - ボットあたり最大 100 インテント
  - ボットあたり最大 100 個のスロットタイプ
  - インテントあたり最大 200 スロット
- ランタイムの制限:
  - 最大 15 分のセッションタイムアウト
  - 1 秒あたり最大 1,000 リクエスト (引き上げ可能)

  - テキスト入力あたり最大 1,500 文字
- トレーニングデータの制限:
  - インテントあたり最大 1,500 サンプル発話
  - 最大 10,000 個のスロットタイプ値
  - サンプル発話あたり最大 140 文字

> **Note**
>
> 本番稼働用により高い制限が必要な場合は、AWS サポートセンターを通じてクォータの引き上げをリクエストできます。

コストについて

Amazon Lex V2 の料金を理解すると、チャットボットデプロイの計画と予算に役立ちます。 V2

- リクエストベースの料金:
  - テキストリクエスト: 1 か月あたり最初の 10,000 件のリクエストの後、リクエストごとに 0.00075 USD
  - スピーチリクエスト: 1 か月あたり最初の 1,000 件のリクエストの後、リクエストごとに 0.004 USD
- 無料利用枠:
  - 最初の 1 年間、1 か月あたり 10,000 件のテキストリクエスト
  - 最初の 1 年間、1 か月あたり 1,000 件の音声リクエスト
- 追加コスト:
  - Lambda 関数 (使用する場合): 100 1Mリクエストあたり 0.20 USD + コンピューティング時間
  - Amazon CloudWatch Logs ログ: 取り込まれた GB あたり 0.50 USD
  - 外部統合のデータ転送コスト
- コスト最適化のヒント:
  - セッション属性を使用して冗長な API コールを減らす
  - 効率的な会話フローを実装してターンを最小限に抑える
  - CloudWatch Logs を使用して使用状況をモニタリングし、最適化の機会を特定する

> **Important**
>
> 現在の料金情報については、料金が変わる可能性があるため、Amazon Lex の料金ページにアクセスしてください。

Lambda 統合の概要

この演習では Lex ボット設定に焦点を当てますが、説明されている高度な機能 (POS 統合、リード生成、収益最適化) では、通常、バックエンド処理に Lambda 関数が必要です。

- Lambda が必要な場合:
  - 外部データベースに対する注文ステータスの検証
  - アップセルのリアルタイムのインベントリチェック
  - 顧客データのストレージと取得
  - 顧客の姓名を使用した S3 テーブルの販売履歴検索
  - パーソナライズされたレコメンデーションの購入パターン分析
  - 支払い処理の統合
  - E メールと SMS の確認の送信
- S3 販売履歴の統合:
  - CustomerFirstName スロットと CustomerLastName スロットを使用して S3 テーブルをクエリする
  - 購入履歴を分析して購入パターンと設定を特定する
  - 過去の購入に基づいてパーソナライズされた製品のレコメンデーションを生成する
  - 動的料金戦略の顧客存続期間価値を計算する
  - 購入相関分析からクロスセルの機会を特定する
- 基本的な Lambda セットアップ要件:
  - 適切なアクセス許可を持つ Lambda 実行ロール
  - プライベートリソースにアクセスする場合の VPC 設定
  - 設定の環境変数
  - エラー処理とログ記録の実装
- 統合ポイント:
  - インテントフルフィルメント - 完了したインテントを処理する

- スロット検証 - ユーザー入力をリアルタイムで検証する
- ダイアログ管理 - 会話フローを制御する

> **Note**
>
> この演習では、Lex ボット設定に焦点を当てます。Lambda 統合は、要件の進化に応じて後で追加できます。ボットはテストとデモンストレーションの目的で静的レスポンスを使用します。

実装前チェックリスト

ボットの作成に進む前に、以下を確認してください。

- ✓ 請求が有効になっているアクティブな AWS アカウント
- ✓ Lex および関連サービスの適切な IAM アクセス許可
- ✓ 選択したターゲット AWS リージョン
- ✓ サービスの制限とクォータの確認
- ✓ コストへの影響を理解している
- ✓ 計画された Lambda 統合アプローチ (必要な場合)
- ✓ AWS マネジメントコンソールへのアクセス

SmartCommerce ボットを作成する

SmartCommerce カスタマーサービスボットを作成するには

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/lex/ で Amazon Lex コンソールを開きます。
2. [ボットの作成] を選択します。
3. 作成方法について、「空白のボットを作成」を選択します。
4. Bot Configuration セクションで、次の操作を行います。
   - ボット名: **`SmartCommerceBot`**
   - 説明: **`Advanced customer service chatbot with upselling, lead generation, and revenue optimization`**

5. アクセス許可 セクションで、基本的な Amazon Lex アクセス許可で新しいロールを作成する を選択します。
6. [Children's Online Privacy Protection Act (COPPA)] セクションで、[いいえ] を選択します。
7. セッションタイムアウトセクションで、複雑なマルチターン会話を許可する**15 minutes**ようにを に設定します。
8. [次へ] を選択します。
9. 言語として英語 (米国) を追加し、完了 を選択します。

カスタムスロットタイプを作成する

カスタムスロットタイプを作成するには

1. 左側のナビゲーションペインで、スロットタイプを選択します。
2. スロットタイプを追加 を選択し、次のスロットタイプを作成します。

    a. ProductCategories:

    - 電化製品
    - 服
    - ホーム
    - 書籍
    - スポーツ
    - ビューティ
    - 自動車

    b. ReturnReasons:

    - 欠陥
    - サイズが正しくない
    - 変更されたマインド
    - 説明どおりではない
    - 配送時に破損
    - より安い料金が見つかりました

    c. ContactPreferences:

    - E メール

- SMS
- 通話
- アプリ内通知
- Mail

d. CustomerTiers:

- 青銅
- シルバー
- ゴールド
- プラチナ

カスタムインテントの作成

CheckOrderStatus インテントを作成するには

1. 左側のナビゲーションペインで、インテントを選択します。
2. インテントを追加 を選択し、 という名前を付けます**CheckOrderStatus**。
3. 次のサンプル発話を追加します。
   - 「注文 {OrderNumber} のステータスは何ですか」
   - 「注文番号 {OrderNumber} を確認する必要があります」
   - 「パッケージの場所」
   - 「注文の追跡 {OrderNumber}」
   - 「注文はまだ配送済みです」
   - 「注文が届くのはいつですか」
   - 「配信について知りたい」
4. 次のスロットを追加します。
   - `OrderNumber` (AMAZON.Number) - 必須

     プロンプト：「注文番号を入力できますか？」
   - `CustomerEmail` (AMAZON.EmailAddress) - 必須

     プロンプト：「ID を確認するには、この注文に関連付けられた E メールアドレスを指定してください。」

5. {CustomerEmail} の「Let me check order #{OrderNumber}」という確認プロンプトを設定します。正しいですか？
6. フルフィルメントメッセージを設定します。「注文を見つけました! 注文 #{OrderNumber} は {OrderStatus} です。{UpsellMessage}"

UpsellProducts インテントを作成するには

1. という名前の新しいインテントを作成します**`UpsellProducts`**。
2. サンプル発話を追加します。
   - 「関連製品を表示する」
   - 「これで他に何が起こるか」
   - 「何か取引はありますか」
   - 「{ProductCategory} アイテムに興味がある」
   - 「What's on sale today」
   - 「他のオプションを表示する」
3. スロットの追加:
   - `ProductCategory` (ProductCategories) - 必須

     プロンプト：「関心のある製品の種類」
   - `Budget` (AMAZON.Number) - オプション

     プロンプト：「追加アイテムの予算範囲は何ですか？」

CaptureLeadInfo インテントを作成するには

1. という名前の新しいインテントを作成します**`CaptureLeadInfo`**。
2. サンプル発話を追加します。
   - 「新製品について知りたい」
   - 「更新にサインアップする」
   - 「{InterestCategory} に興味がある」
   - 「売上について知らせてください」
   - 「メーリングリストに参加したい」

- 「Send me deals and offers」
- 「私の名前は {CustomerFirstName} {CustomerLastName}」
- 「私は {CustomerFirstName} {CustomerLastName} で、パーソナライズされたレコメンデーションが必要です」

3. スロットの追加:

- `CustomerFirstName` (AMAZON.FirstNameA) - 必須

  プロンプト:「パーソナライズされたサービスの名は何ですか？」

- `CustomerLastName` (AMAZON.LastName) - 必須

  プロンプト：「購入履歴を検索できるように姓を入力しますか？」

- `CustomerEmail` (AMAZON.EmailAddress) - 必須
- `CustomerPhone` (AMAZON.PhoneNumber) - オプション
- `InterestCategory` (ProductCategories) - 必須
- `ContactPreference` (ContactPreferences) - 必須

AI を活用した機能を有効にする

アシスト NLU およびその他の AI 機能を有効にするには

1. 左側のナビゲーションペインで、ボット設定を選択します。
2. アシスト NLU で、有効化 を選択します。
3. Generative AI で、以下を有効にします。

- アシスト付きスロット解決
- 記述ボットの構築

4. 「感情分析」で、「有効化」を選択します。
5. [保存] を選択します。

ボットのテストとデプロイ

SmartCommerce ボットをテストするには

1. ビルド を選択してボットをコンパイルします。

2. ビルドが完了したら、テストを選択します。
3. 次の会話フローをテストします。

    a. 注文ステータスフロー:

        - ユーザー : 「注文の確認 12345」
        - ボット: E メール検証をリクエストします
        - ユーザー: E メールを提供する
        - ボット: ステータスとアップセルを提供します

    b. アップセリングフロー:

        - ユーザー: "Show me deals"
        - ボット: 製品カテゴリを尋ねる
        - ユーザー: "Electronics"
        - ボット: パーソナライズされたオファーを提示します

    c. リード生成フロー:

        - ユーザー: 「更新にサインアップする」
        - ボット: 連絡先情報を収集します
        - ユーザー: 詳細を提供します
        - ボット: サブスクリプションを確認し、即時割引を提供します

4. AI 機能が動作していることを確認します。

    - 「私のモノが到着していない」などの自然言語のバリエーションをテストする
    - 感情分析がボットレスポンスを調整することを確認する
    - スロット解決が複雑な入力で動作することを確認する

5. テストが完了したら、パブリッシュを選択してボットをデプロイします。

パフォーマンス分析と最適化

包括的な分析と収益追跡を使用して、SmartCommerce ボットのパフォーマンスをモニタリングおよび最適化します。

主要なパフォーマンスメトリクス

- 収益メトリクス:

  - 顧客層別のアップセル変換率
  - インタラクションあたりの平均注文値の増加
  - 会話あたりの収益
  - お客様のライフタイム価値への影響
- リード生成メトリクス:
  - インタラクションタイプ別のリードキャプチャレート
  - リード品質スコアと変換率
  - アグリゲータ売上からのデータ収益化収益
  - E メールサブスクリプションの増加率
- 運用メトリクス:
  - インテント認識の精度
  - スロットフィルの成功率
  - 会話の完了率
  - 顧客満足度スコア

継続的最適化戦略

- A/B テスト:
  - さまざまなアップセルメッセージングアプローチをテストする
  - 積極的なリードキャプチャ手法と穏やかなリードキャプチャ手法を比較する
  - 最大変換の割引率を最適化する
- Machine Learningの最適化:
  - 成功した会話パターンを分析する
  - アップセルオファーの最適なタイミングを特定する
  - 顧客階層分類アルゴリズムを改良する

演習の結論

お疲れ様でした。収益の最適化とカスタマーエンゲージメントのためのエンタープライズレベルの機能を示す高度なカスタマーサービスチャットボットを正常に構築しました。この SmartCommerce ボットは、企業が Amazon Lex V2 の AI を活用した機能を活用して以下を実現する方法を示してい ます。V2

- 収益を最大化する: インテリジェントなアップセリング、クロスセリング、動的な料金戦略を通じて
- リードの生成: 複数の収益ストリームの顧客情報を体系的にキャプチャして認定する
- カスタマーエクスペリエンスの向上: AI を使用して自然言語を理解し、パーソナライズされたインタラクションを提供する
- オペレーションの最適化: 品質を維持しながら効率的にスケールする自動化されたカスタマーサービスを通じて

この演習で示す手法は、さまざまな業界やユースケースに合わせて適応および拡張でき、ビジネスの成長を促進する高度な会話型 AI ソリューションを構築するための基盤を提供します。

次のステップ

SmartCommerce ボットをさらに強化するには、次の実装を検討してください。

- 高度な統合:
  - 完全な顧客プロファイルのための CRM システム統合
  - リアルタイムの可用性のためのインベントリ管理システム接続
  - シームレスなトランザクションのための支払い処理の統合
- マルチチャネルデプロイ:
  - ウェブサイトチャットウィジェットの統合
  - ソーシャルメディアプラットフォーム接続
  - 音声インターフェイスの実装
- 高度な分析:
  - ビジネスメトリクスのカスタムダッシュボード開発
  - 顧客行動の予測分析
  - ROI の追跡と属性モデリング

開始方法のベストプラクティス

会話設計の原則

これらの会話設計原則に最初から従えば、自然なやり取りを提供する、より効果的で保守可能でユーザーフレンドリーな Amazon Lex V2 チャットボットを構築できます。

コア設計原則

- ユーザー目標から始める - システムができることではなく、ユーザーが達成したいことを中心にボットを設計します。ユーザーのジャーニーと期待される成果に焦点を当てます。
- 自然言語を使用する - プロンプトとレスポンスを会話で書き込みます。技術的な専門用語は避け、役に立つ人間として話してください。
- 明確なオプションを提供する - ユーザーが行き詰まったら、一般的なヘルプテキストではなく、発言できる内容の具体的な例を提供します。
- Keep It Simple - 基本的な機能から始めて、徐々に複雑さを追加します。ユーザーは一般的なタスクをすばやく完了できるはずです。
- エラーを適切に処理する - ボットが理解できない場合は、「理解できない」と言うのではなく、役に立つガイダンスを提供します。
- 重要なアクションの確認 - 注文や情報の削除など、簡単に元に戻すことができないアクションを実行する前に、必ず確認してください。
- エスケープルートを提供する - 常にユーザーに、必要に応じて最初からやり直したり、ヘルプを得たり、人間とつながったりする方法を提供します。

会話フローのベストプラクティス

- 明確な期待を設定する - ボットが会話の早い段階でできることとできないことをユーザーに知らせます。
- プログレッシブ開示を使用する - 複数の質問でユーザーを圧倒するのではなく、一度に 1 つずつ情報を求めます。
- コンテキストを提供する - 既に収集した情報と、まだ必要なものをユーザーに再認識させます。
- 修正を簡単にする - ユーザーが完全にやり直すことなく情報を修正できるようにします。

実際のユースケースと例

これらの実用的な例は、新しい Amazon Lex V2 ユーザーが遭遇する一般的なシナリオに会話設計原則を適用する方法を示しています。

ユースケース 1: 予約予約

シナリオ: メディカルオフィスが予約スケジューリングを自動化したいと考えています。

課題: ユーザーは複数の情報 (サービスタイプ、日付、時刻、連絡先情報) を指定する必要があり、詳細を変更したい場合があります。

ソリューションアプローチ:

- Start Broad: 「どのような予約をしますか？」 (歯、目の検査、診察)
- 絞り込み: 「いつ歯の予約を希望されますか？」 (「次の週」や「金曜日の午後」などの柔軟な入力を受け入れる)
- 変更の確認とオファー: 「3 月 15 日金曜日の午後 2 時に歯のクリーニングを予定しました。これはうまくいきますか？」
- 変更の処理: ユーザーが「代わりに午後 3 時に実行できますか？」と言った場合は、プロセス全体を再起動せずに時間を更新します。

主なテクニック:

- 日付/時刻の柔軟な入力AMAZON.Timeに AMAZON.Dateと を使用する
- 予約タイプのカスタムスロットタイプを作成する
- 準備を確定する前に確認プロンプトを使用する

ユースケース 2: 注文ステータスの照会

シナリオ: e コマース企業は、顧客がサポートを呼び出さずに注文ステータスを確認できるようにしたいと考えています。

課題: ユーザーは注文番号が役に立たないか、さまざまな方法で に尋ねる場合があります。

ソリューションアプローチ:

- 複数のエントリポイント:「注文の場所」、「パッケージの追跡」、または「注文ステータス」を受け入れる
- 柔軟な識別: 「注文の追跡に役立ちます。注文番号をお持ちですか、または E メールアドレスを使用しますか？」
- 役立つガイダンス:「注文番号は通常確認 E メールにあり、「ORD-」で始まります」
- クリア結果: 「注文 #ORD-12345 は昨日発送され、明日午後 8 時までに到着します。詳細を追跡しますか？」

主なテクニック:

- 注文番号に AMAZON.AlphaNumeric などの組み込みスロットタイプを使用する
- 注文を識別する複数の方法 (E メール、電話番号、注文番号) を提供する
- レスポンスで明確で実用的な情報を提供する

ユースケース 3: よくある質問とサポート

シナリオ: ソフトウェア会社が、一般的なサポートの質問を自動的に処理したいと考えています。

課題: ユーザーはさまざまな方法で同じ質問をし、一部の問題には人間の助けが必要です。

ソリューションアプローチ:

- 広範なインテント認識:「ログインできない」、「ログインの問題」、「パスワードが機能しない」などのバリエーションを認識するようにインテントをトレーニングする
- ガイド付きトラブルシューティング:「いくつかの簡単なステップを試してみましょう。まず、パスワードをリセットしようとしましたか？」
- エスカレーションパス:「これらのステップが機能しない場合は、サポートチームに接続できます。これを行いますか？」
- フォローアップ:「これは問題を解決しましたか? 他のサポートが必要な場合は、いつでも「ヘルプ」と言うことができます。」

主なテクニック:

- 多くのサンプル発話を使用して、ユーザーが問題を表現するさまざまな方法をキャプチャする
- 長い説明ではなくstep-by-stepガイダンスを提供する
- 自動ヘルプが不十分な場合は、常にヒューマンサポートへの道を提供する

複雑なシナリオの処理

単純なボットでも複雑な状況に遭遇します。これらを適切に処理する方法は次のとおりです。

- ユーザーが考え方を変更する場合:
  - ユーザー：「実際には、チューリップの代わりにバラが欲しい」
  - ボット: "問題なし! 注文をバラに更新しました。バラは金曜日の午後 2 時に集荷できるようになります。これは正しいですか？」
- 情報が不明確である場合:

  - ユーザー:「来週の花が必要」
  - ボット:「よろしくお願いします。来週のどの曜日が最適ですか?「月曜日」や「次の火曜日」などと言うことができます。
- ボットが理解できない場合:
  - ユーザー:「長い茎を持つ紫が欲しい」
  - ボット:「これを正しく取得したい」。紫色の花をお探しですか? 紫色のバラ、紫色のチューリップ、紫色の虹彩が利用可能です。」
- ユーザーがやり直す必要がある場合:
  - ユーザー:「Cancel this, I want to start over」
  - ボット: "Of course! 現在の注文をクリアしました。今日はどのような種類の花を注文しますか?」

ボットの例

このセクションでは、独自のアプリケーションの開始点として使用できる Amazon Lex ボットの例を示します。

トピック

- コールセンターエージェントアシスタント

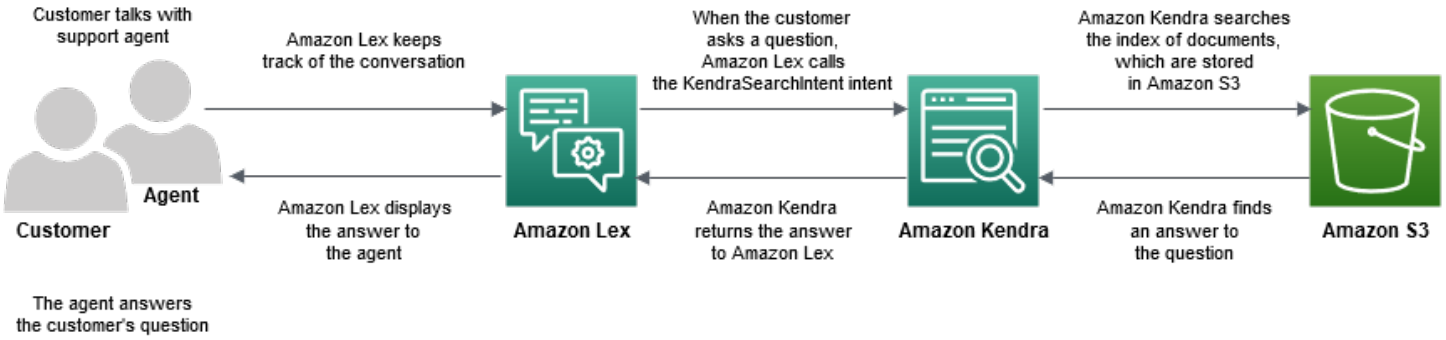
コールセンターエージェントアシスタント

このチュートリアルでは、Amazon Kendra で Amazon Lex V2 を使用して、カスタマーサポートエージェントを支援し、ウェブアプリケーションとして公開するエージェントアシストボットを構築します。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 V2 ボットは、お客様の最初の連絡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、エージェントが顧客のクエリにリアルタイムで回答するために使用する Amazon Lex V2 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

推定コスト: Amazon Kendra インデックスの場合は 1 時間あたり 2.50 USD、Amazon Lex V2 リクエストの場合は 1000 件あたり 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
- ステップ 2: Amazon Lex V2 ボットを作成する V2
- 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 4: Amazon Cognito をセットアップする
- 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
- ステップ 6: ボットを使用する

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 V2 ボットを作成する V2

ステップ 2: Amazon Lex V2 ボットを作成する V2

Amazon Lex V2 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提供します。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回答を Amazon Lex V2 に返します。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 マネジメントコンソールにサインインし、https://console.aws.amazon.com/lex/ で Amazon Lex コンソールを開きます。
2. ナビゲーションペインで、[ボット] を選択します。
3. [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. [作成] を選択します。ボットを作成すると、Amazon Lex V2 にボット工ディタタブが表示されます。

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 V2 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 プールを使用して、ユーザーに他の AWS サービスへのアクセスを許可する AWS 認証情報を提供します。このチュートリアルでは、Amazon Lex V2 へのアクセスを提供します。 V2

ID プールを作成する場合、Amazon Cognito は認証済みおよび未認証ユーザーの AWS Identity and Access Management (IAM) ロールを提供します。Amazon Lex V2 へのアクセスを許可するポリシーを追加して、IAM ロールを変更します。 V2

Amazon Cognito をセットアップするには

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/cognito/ で Amazon Cognito コンソールを開きます。
2. [アイデンティティプールの管理] 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. [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. [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. 「AWS 認証情報の取得」セクションで、ID プール ID を検索して記録します。
11. Amazon Lex V2 へのアクセスを許可するには、認証された IAM ロールと認証されていない IAM ロールを変更します。
    a. AWS マネジメントコンソールにサインインして、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。

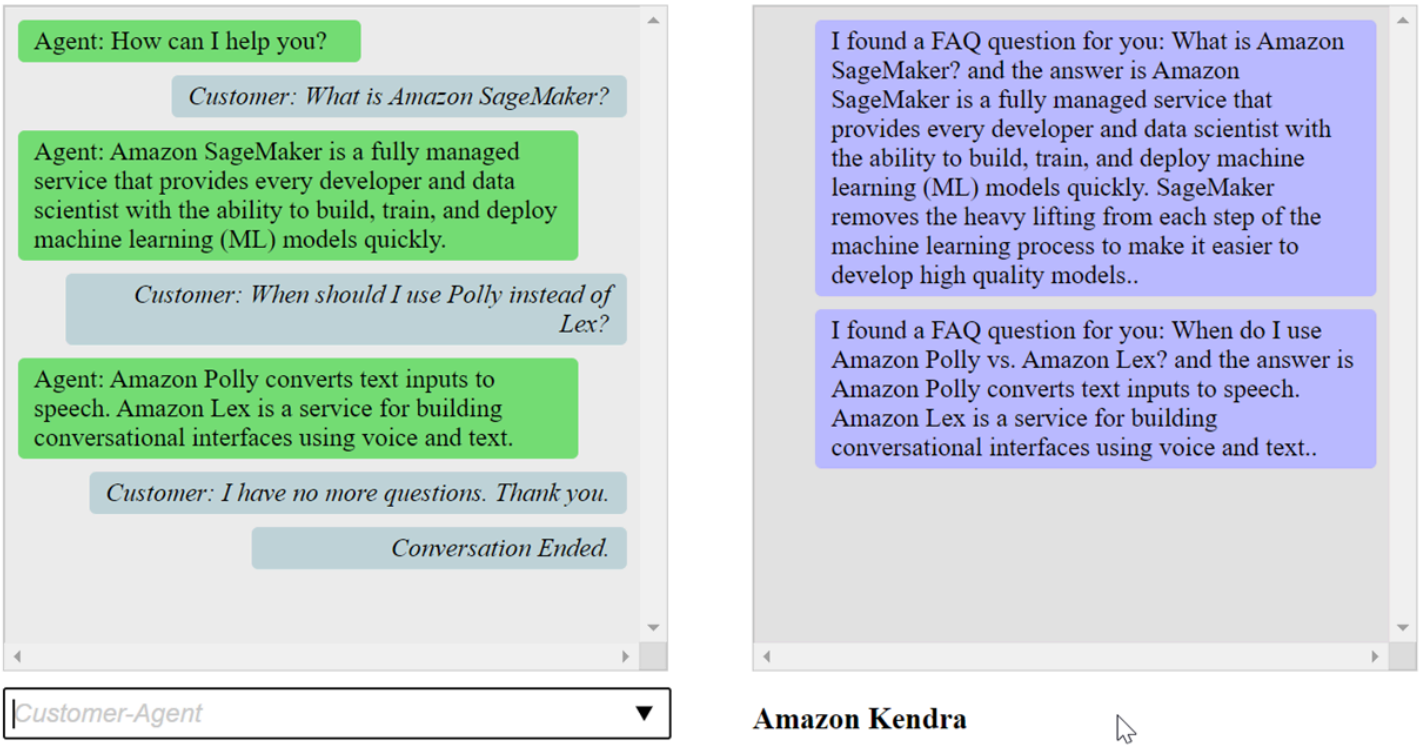
   b. `Amazon Lex V2 runtime parameters` セクションで、ボット名を入力します。

   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。

ウェブアプリケーションを実行するには`index.html`、 を開いて、次のようなボットとの会話を行います。



「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;
```

顧客として入力を指定すると、Amazon Lex V2 ランタイム API はそれを Amazon Lex V2 に送信します。 V2

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。

Amazon Lex V2 ボットの使用

Amazon Lex V2 ボットを作成して、ユーザーと対話して情報を誘発してタスクを達成します。例えば、花束を注文したり、ホテルの部屋を予約したりするために必要な情報を収集するボットを作成できます。

ボットを構築するには、以下の詳細を取得する必要があります。

1. ボットが顧客と対話するために使用する言語。1 つ以上の言語を選択できます。各言語には、独立したインテント、スロット、スロットタイプが含まれています。
2. ボットがユーザーの達成に役立つインテントまたは目標。ボットには、花の注文、ホテルやレンタカーの予約など、1 つ以上のインテントを含めることができます。インテントを開始するためにユーザーが行うステートメントあるいは発話を決定する必要があります。
3. インテントを満たすためにユーザーから収集する必要がある情報またはスロット。例えば、ユーザーから花の種類を取得したり、ホテルの予約の開始日を取得する必要がある場合があります。Amazon Lex V2 がユーザーからスロット値を誘発するために使用される 1 つ以上のプロンプトを定義する必要があります。
4. ユーザーから必要なスロットのタイプ。カスタムスロットタイプ (ユーザーが注文できる花のリストなど) を作成する必要がある場合や、組み込みのスロットタイプ (予約の開始日用の `AMAZON.Date` スロットタイプなど) を使用できる場合があります。
5. インテント間のインタラクションフロー。会話フローを設定して、インテントが呼び出された後のユーザーとボット間のインタラクションを定義できます。Lambda 関数を作成して Lambda 関数を使用して、インテントを検証および応答します。

トピック

- Amazon Lex V2 の会話フローの変更 V2
- Amazon Lex V2 でボットを作成するさまざまな方法
- Amazon Lex V2 ボットへの新しい言語の追加
- インテントの追加
- スロットタイプの追加
- コンソールを使用したボットのテスト

> **Note**
>
> 2022 年 8 月 17 日、Amazon Lex V2 はユーザーとの会話の管理方法の変更をリリースしました。この変更により、ユーザーが会話の中でたどるパスをより細かく制御できるようになりました。詳細については、「Amazon Lex V2 の会話フローの変更 V2」を参照してください。2022 年 8 月 17 日より前に作成されたボットは、ダイアログコードフックメッセージ、値の設定、次のステップの設定、条件の追加をサポートしていません。

Amazon Lex V2 の会話フローの変更 V2

2022 年 8 月 17 日、Amazon Lex V2 はユーザーとの会話の管理方法の変更をリリースしました。この変更により、ユーザーが会話の中でたどるパスをより細かく制御できるようになりました。

変更前は、Amazon Lex V2 はインテントした優先順位に基づいてスロットを誘発することで会話を管理していました。Lambda 関数の DialogAction を使用して、この動作を動的に変更し、ユーザー入力に基づいて会話パスを変更できます。これは、会話の現在の状態を追跡し、セッションの状態に基づいて次に何をすべきかをプログラム的に決定することで実現できます。

この変更により、Lambda 関数を使用せずに Amazon Lex V2 コンソールまたは API を使用して会話パスと条件分岐を作成できます。Amazon Lex V2 は会話の状態を追跡し、ボットの作成時に定義された条件に基づいて次に何をすべきかを制御します。これにより、ボットを設計しながら複雑な会話を簡単に作成できます。

これらの変更により、顧客との会話を完全にコントロールできるようになります。ただし、パスを定義する必要はありません。会話パスを指定しない場合、Amazon Lex V2 はインテント内のスロットの優先順位に基づいてデフォルトパスを作成します。引き続き Lambda 関数を使用して会話パスを動的に定義できます。このようなシナリオでは、Lambda 関数で設定されたセッション状態に基づいて会話が再開されます。

このアップデートでは、以下を提供します:

- 複雑な会話フローのボットを作成するための新しいコンソールエクスペリエンス。
- ボット作成用の既存の API を更新し、新しい会話フローをサポートします。
- インテント呼び出し時にメッセージを送信するための初期応答。
- スロット誘発、ダイアログコードフックとしての Lambda 呼び出し、確認に対する新しい応答。
- 会話のターンごとに次のステップを指定できます。
- 条件を評価して複数の会話パスを設計できます。

- 会話中の任意の時点でのスロット値とセッション属性の設定。

古いボットについては、以下の点に注意してください。

- 2022 年 8 月 17 日より前に作成されたボットは、引き続き古いメカニズムを使用して会話フローを管理します。それ以降に作成されたボットは、新しい会話フロー管理方法を使用します。
- 2022 年 8 月 17 日以降にインポートによって作成された新しいボットは、新しい会話フロー管理を使用します。既存のボットへのインポートでは、引き続き古い会話管理方法が使用されます。
- 2022 年 8 月 17 日より前に作成されたボットの新しい会話フロー管理を有効にするには、ボットをエクスポートし、新しいボット名を使用してボットをインポートします。インポートによって新しく作成されたボットは、新しい会話フロー管理を使用します。

2022 年 8 月 17 日以降に作成された新しいボットについては、次の点に注意してください。

- Amazon Lex V2 は、望ましいエクスペリエンスを提供するために、定義された会話フローを正確に設計どおりに実行します。実行時にデフォルトの会話パスを避けるために、すべてのフローブランチを設定する必要があります。
- コードフックに続く会話ステップは、完全に設定する必要があります。ステップが不完全だと、ボットに障害が発生する可能性があるためです。2022 年 8 月 17 日より前に作成されたボットを検証することをおすすめします。これらのボットには、コードフック後の会話ステップの自動検証機能がないためです。

Amazon Lex V2 でボットを作成するさまざまな方法

Amazon Lex V2 では、以下の方法でボットを作成できます。

1. Amazon Lex V2 コンソールを使用して、ウェブサイトインターフェイスでボットを作成します。詳細については、「Amazon Lex V2 コンソールを使用してボットを作成する」を参照してください。
2. 記述的ボットビルダーを使用して、Amazon Bedrock の生成 AI 機能でボットを作成します。詳細については、「記述的ボットビルダーを使用した、Lex V2 での説明文によるボットの構築」を参照してください。
3. ボットテンプレートを使用して、一般的なビジネスユースケースに適した事前設定済みのボットを作成します。詳細については、「テンプレートを使用した Amazon Lex V2 ボットの作成」を参照してください。

4. AWS SDK を使用して、API オペレーションを使用してボットを作成します。
5. Automated Chatbot Designer を使用して、エージェントと顧客間の既存のチャット文字起こしを使用してボットを作成します。詳細については、「自動 Chatbot デザイナーを使用した Amazon Lex V2 ボットの作成」を参照してください。
6. 既存のボット定義のインポート。詳細については、「Lex V2 でのボットのインポート」を参照してください。
7. CloudFormationを使用してボットを作成します。詳細については、「を使用した Amazon Lex V2 リソースの作成 AWS CloudFormation」を参照してください。

トピック

- Amazon Lex V2 コンソールを使用してボットを作成する
- テンプレートを使用した Amazon Lex V2 ボットの作成
- 自動 Chatbot デザイナーを使用した Amazon Lex V2 ボットの作成

Amazon Lex V2 コンソールを使用してボットを作成する

Amazon Lex V2 ボットを作成して、ユーザーと対話して情報を誘発してタスクを達成します。例えば、花を注文したり、ホテルの部屋を予約したりするために必要な情報を収集するボットを作成できます。AWS コンソールを使用してボットを作成するには、まず名前、説明、およびいくつかの基本情報を定義します。

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/lex/ で Amazon Lex コンソールを開きます。
2. [ボットの作成] を選択します。
3. [作成方法] セクションで、[従来型] を選択し、[空のボットを作成します。] を選択します。
4. Bot configuration セクションで、ボットに名前とオプションの説明を与えます。
5. IAM アクセス許可セクションで、Amazon CloudWatch などの他の AWS サービスにアクセスするためのアクセス許可を Amazon Lex V2 に付与する AWS Identity and Access Management (IAM) ロールを選択します。Amazon Lex V2 でロールを作成させるか、CloudWatch 権限を持つ既存のロールを選択できます。
6. Children's Online Privacy Protection Act (COPPA) セクションで、適切な応答を選択します。
7. Idle session timeout セクションで、Amazon Lex V2 がユーザーとのセッションを開いたままにする期間を選択します。Amazon Lex V2 は、ボットが同じ変数で会話を再開できるように、セッション期間中はセッション変数を保持します。

8. [詳細設定] (オプション) セクションで、ボットを識別し、アクセスを制御し、リソースを監視するために役立つタグを追加します。
9. Next を選択し、ボットを作成して言語の追加に移ります。

テンプレートを使用した Amazon Lex V2 ボットの作成

Amazon Lex V2 は、大規模なエクスペリエンスを作成し、デジタルエンゲージメントを促進するための事前構築済みのソリューションを提供します。あらかじめ用意されたボットテンプレートは、クライアントエクスペリエンスを自動化して標準化します。ボットテンプレートは、音声とチャットの両方の方法で、トレーニングデータとダイアログプロンプトの両方とともに、すぐに使用できる会話フローを提供します。リソースを最適化しながらボットソリューションを迅速に提供できるため、顧客との関係に集中できます。

ビジネスユースケースに基づいてあらかじめ構築されたボットを作成できます。 CloudFormation コンソールを使用して、Amazon S3、Connect Customer、DynamoDB などの関連サービスのビルド済みオプションを選択できます。

現在、Amazon Lex V2 では、次の業種がサポートされています。

- 金融サービス
- 小売注文
- 自動車保険
- 電気通信
- 航空サービス
- 今後、追加のテンプレートが利用可能になります。

提供されているビジネスソリューションテンプレートを使用してボットを構築し、ビジネス要件に合わせてボットをカスタマイズできます。

> **Note**
>
> テンプレートは、 CloudFormation スタックを通じて Amazon Lex V2 の外部にリソースを作成します。スタックは、Lambda や DynamoDB などの他のコンソールで変更する必要がある場合があります。

ボットテンプレートの構築とデプロイの前提条件:

- AWS アカウント
- 次の AWS サービスへのアクセス:
  - ボットを作成する Amazon Lex V2
  - ビジネスロジイン機能用 Lambda
  - テーブルを作成する DynamoDB
  - ロールを作成する IAM アクセス
  - スタックを実行する AWS CloudFormation
- IAM アクセスとシークレットキー認証情報
- Connect Customer インスタンス (オプション)

> **Note**
>
> さまざまな AWS のサービスを使用すると、サービスごとにそれぞれの使用コストが発生します。

Amazon Lex V2 テンプレートからボットを構築するには:

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/lex/ で Amazon Lex コンソールを開きます。
2. 左側のナビゲーションペインから [ボットテンプレート] を選択します。
3. ボットテンプレートに使用する業種を選択します。注: 現在利用できるボットテンプレートは 5 つあります。詳細は近日公開予定です。
4. 使用するテンプレートに対して [作成] を選びます。タブ CloudFormation が で開き、CloudFormation スタックのパラメータを編集できます。選択したテンプレートのオプションはすべて設定済みです。また、[詳細] を選択すると、ボットテンプレートの仕組みについて詳しく知ることができます。
5. CloudFormation コンソールで、選択したテンプレートの値ごとにデフォルト設定 CloudFormation を作成します。独自のスタック名、 CloudFormation パラメータ、Amazon DynamoDB テーブル、および (オプション) Connect Customer パラメータを選択することもできます。
6. ウィンドウの下部にある [スタックの作成] を選択します。
7. CloudFormation はリクエストをバックグラウンドで数分間処理し、新しいボットを設定します。注: このプロセスでは、DynamoDB テーブル、Connect Customer 問い合わせフ

ロー、Connect Customer インスタンスのリソースが自動的に作成されます。 CloudFormation コンソールで進行状況を追跡し、CloudFormation スタックの作成が完了したら Amazon Lex V2 コンソールに戻ります。

8. ビルドに成功すると、メッセージが表示され、[ボットリストに移動] を選択して [ボット] ページに移動できます。ここで、テストや使用の準備ができている新しいボットを見つけることができます。

ボットテンプレートの設定

Lambda 関数 — ボットテンプレートは、デプロイに必要な Lambda 関数を自動的に作成します。複数のボットがテンプレートソリューションの一部である場合、複数の Lambda 関数がパラメータにリストされます。 CloudFormation ボットにデプロイする既存の Lambda 関数がある場合は、カスタム Lambda 関数の名前を入力できます。

Amazon DynamoDB — ボットテンプレートは、サンプルポリシーデータをロードするために必要な DynamoDB テーブルを自動的に作成します。カスタム DynamoDB テーブルの名前を入力することもできます。カスタム DynamoDB テーブルは、ボットテンプレートのデプロイによって作成されたデフォルトテーブルと同じ方法でフォーマットする必要があります。

Connect Customer – ConnectInstanceARN と一意の ContactFlowName を入力して、新しいボットテンプレートを操作するように Connect Customer インスタンスを設定できます。 ContactFlowName Connect Customer を使用すると、IVR システムを使用してボットをend-to-endでテストできます。

ボットテンプレートのトラブルシューティング

- 選択するテンプレートを作成するための適切な権限があることを確認してください。ユーザーにはCloudFormation:CreateStack アクセス許可と、テンプレート内にリストされている AWS リソースのアクセス許可が必要です。ユーザー権限が必要なリソースのリストは、[テンプレートの作成] ページの下部にあります。
- ボットテンプレートの作成に失敗した場合、Amazon Lex V2 コンソール内の赤いバナーには、テンプレートの作成を担当する CloudFormation スタックへのリンクが表示されます。 CloudFormation コンソール内でイベントタブを表示して、テンプレートが失敗する原因となった特定のエラーを確認できます。 CloudFormation エラーを確認したら、「CloudFormation のトラブルシューティング」で詳細を確認してください。
- ボットテンプレートはサンプルデータのみで動作します。テンプレートをカスタムデータで機能させるには、DynamoDB テーブルにデータを入力する必要があります。

自動 Chatbot デザイナーを使用した Amazon Lex V2 ボットの作成

自動 Chatbot デザイナーは、既存の会話トランスクリプトからボットを設計するのに役立ちます。文字起こしを分析し、インテントとスロットタイプの初期設計を提案します。ボットの設計を繰り返し、プロンプトを追加し、ボットのビルド、テスト、デプロイを行うことができます。

Amazon Lex V2 コンソールまたは API を使用して新しいボットを作成するか、ボットに言語を追加した後、2 つのパーティ間で会話の文字起こしをアップロードできます。Automated Chatbot Designer は、文字起こしを分析し、ボットのインテントとスロットタイプを決定します。また、レビュー用の特定のインテントまたはスロットタイプの作成に影響を与えた会話にラベルを付けます。

Amazon Lex V2 コンソールまたは API を使用して、会話の文字起こしを分析し、ボットのインテントとスロットタイプを提案します。

> **Note**
>
> 文字起こしは、英語 (米国) 言語のみで使用できます。

Chatbot デザイナーが分析を完了した後、推奨されるインテントとスロットタイプを確認できます。提案されたインテントまたはスロットタイプを追加したら、コンソールまたは API を使用して、それを変更したり、ボットデザインから削除したりできます。

自動チャットボットデザイナーは、Contact Lens for Connect Customer スキーマを使用して会話トランスクリプトファイルをサポートします。別のコンタクトセンターアプリケーションを使用している場合は、会話の文字起こしを Chatbot Designer が使用する形式に変換する必要があります。詳細については、「文字起こしの入力形式」を参照してください。

Automated Chatbot Designer を使用するには、デザイナーを実行している IAM ロールにアクセス許可を付与する必要があります。特定の IAM ポリシーについては、「自動 Chatbot デザイナーの使用をユーザーに許可する」を参照してください。Amazon Lex V2 がオプションの AWS KMS キーを使用して出力データを暗号化できるようにするには、 に示されているポリシーでキーを更新する必要がありますユーザーが AWS KMS キーを使用してファイルを暗号化および復号できるようにする。

> **Note**
>
> を使用する場合は KMS key、使用する IAM ロールに関係なく、KMS key ポリシーを指定する必要があります。

トピック

- 会話文字起こしのインポート
- インテントとスロットタイプの作成
- 文字起こしの入力形式
- 文字起こしの出力形式

会話文字起こしのインポート

会話文字起こしのインポートは 3 つのステップのプロセスです。

1. 正しい形式に変換して、インポートする転写産物を準備します。Contact Lens for Connect Customer を使用している場合、トランスクリプトはすでに正しい形式です。
2. 文字起こしを Amazon S3 バケットにアップロードします。コンタクトレンズを使用している場合、文字起こしはすでに S3 バケットに入っています。
3. Amazon Lex V2 コンソールまたは API オペレーションを使用して、文字起こしを分析します。トレーニングの完了にかかる時間は、文字起こしの量と会話の複雑さによって異なります。通常、毎分 500 行の文字起こしが分析されます。

各ステップについては、続くセクションで説明します。

Contact Lens for Connect Customer からのトランスクリプトのインポート

Amazon Lex V2 自動 Chatbot デザイナーは、コンタクトレンズの文字起こしファイルと互換性があります。コンタクトレンズ文字起こしファイルを使用するには、コンタクトレンズをオンにし、出力ファイルの場所をメモする必要があります。

コンタクトレンズから転写産物をエクスポートするには

1. Connect Customer インスタンスで Contact Lens をオンにします。手順については、「 Connect Customer administrator guide」の「Enable Contact Lens for Connect Customer」を参照してください。
2. Connect Customer がインスタンスに使用している S3 バケットの場所を書き留めます。場所を確認するには、Connect Customer コンソールでデータストレージページを開きます。手順については、「Connect Customer administrator guide」の「Update instance settings」を参照してください。

コンタクトレンズをオンにし、文字起こしファイルの場所を記録したら、Amazon Lex V2 コンソールを使用して文字起こしの分析文字起こしをインポートして分析する手順を参照してください。

文字起こしの準備

文字起こしファイルを作成して、文字起こしを準備します。

- 会話ごとに 1 つの文字起こしファイルを作成し、当事者間のインタラクションをリストします。会話の各インタラクションは、複数の行にまたがることができます。会話の編集済みバージョンと編集されていないバージョンの両方を提供できます。
- ファイルは、文字起こしの入力形式 で指定された JSON 形式である必要があります。
- 会話ターンは少なくとも 1,000 回指定する必要があります。インテントやスロットタイプを把握しやすくするには、会話のターンを約 10,000 回以上設定する必要があります。Automated Chatbot Designer が処理するのは、最初の 700,000 ターンのみです。
- 使用する文字起こしファイルの数に制限はなく、サイズ制限もありません。

インポートする文字起こしを日付でフィルタリングする場合は、ファイルは次のディレクトリ構造内に存在する必要があります。

```
<path or bucket root>
   --> yyyy
      --> mm
         --> dd
            --> transcript files
```

文字起こしファイルには、ファイル名のどこかに「yyyy-mm-dd」形式の日付を含める必要があります。

他のコンタクトセンターアプリケーションから文字起こしをエクスポートするには

1. コンタクトセンターアプリケーションのツールを使用して、会話をエクスポートします。会話には、少なくとも 文字起こしの入力形式 で指定された情報が含まれている必要があります。
2. コンタクトセンターアプリケーションで作成された文字起こしを 文字起こしの入力形式 で説明されている形式に変換します。トランスフォーメーションを実行する責任はお客様にあります。

文字起こしを準備するためのスクリプトを 3 つ提供しています。具体的には次の 2 つです。

- コンタクトレンズの文字起こしを Amazon Lex V2 会話ログと組み合わせるスクリプトです。Contact Lens のトランスクリプトには、Amazon Lex V2 ボットとやり取りする Connect Customer の会話の一部は含まれません。このスクリプトでは、Amazon Lex V2 の会話ログを有効にし、会話ログ CloudWatch Logs とコンタクトレンズ S3 バケットをクエリするための適切なアクセス許可が必要です。
- Amazon Transcribe コール分析を Amazon Lex V2 入力形式に変換するスクリプト。
- Connect Customer チャットのトランスクリプトを Amazon Lex V2 入力形式に変換するスクリプト。

この GitHub リポジトリからスクリプトをダウンロードできます。https://github.com/aws-samples/amazon-lex-bot-recommendation-integration。

文字起こしを S3 バケットにアップロードします。

コンタクトレンズを使用している場合、文字起こしファイルはすでに S3 バケットに含まれています。トランスクリプトファイルの場所とファイル名については、「Connect Customer administrator guide」の「Example Contact Lens output files」を参照してください。

別のコンタクトセンターアプリケーションを使用していて、文字起こしファイルに S3 バケットを設定していない場合は、次の手順を実行します。それ以外の場合は、既存の S3 バケットがある場合は、Amazon S3 コンソールにログインした後、ステップ 5 から開始してこの手順を実行します。

ファイルを S3 バケットにアップロードするには

1. AWS マネジメントコンソールにサインインして Amazon S3 コンソール (https://console.aws.amazon.com/s3/) を開きます。
2. [バケットを作成] を選択します。
3. [バケット] に名前を付けて、[リージョン] を選択します。リージョンは Amazon Lex V2 で使用するリージョンと同じである必要があります。ユースケースに必要な他のオプションを設定します。
4. [バケットを作成] を選択します。
5. バケットのリストで、既存のバケットか先ほど作成したバケットを選択します。
6. アップロード を選択します。
7. アップロードする文字起こしファイルを追加します。
8. アップロード を選択します。

Amazon Lex V2 コンソールを使用して文字起こしの分析

自動ボット設計を使用できるのは、空の言語のみです。既存のボットに新しい言語を追加することも、新しいボットを作成することもできます。

新しいボットで新しい言語を作成するには

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/lex/ で Amazon Lex コンソールを開きます。
2. [ボットの作成] を選択します。
3. [Automated Chatbot Designer で開始] を選択します。情報を入力して、新しいボットを作成します。
4. [Next (次へ)] を選択します。
5. ボットに言語を追加する で言語の情報を入力します。
6. S3 上の文字起こしファイルの場所 セクションで、必要に応じて文字起こしファイルが含まれている S3 バケットとファイルへのローカルパスを選択します。
7. 以下のオプションを選択できます。
   - 処理中にトランスクリプトデータを暗号化するための AWS KMS キー。キーを選択しない場合、サービス AWS KMS キーが使用されます。
   - 文字起こしを特定の日付範囲にフィルタリングします。文字起こしをフィルタリングする場合は、正しいフォルダ構造内にある必要があります。詳細については、「文字起こしの準備」を参照してください。
8. [Done] (完了) をクリックします。

Amazon Lex V2 が文字起こしを処理するのを待ちます。解析が完了すると、完了メッセージが表示されます。

文字起こしの分析を中止する方法

アップロードした文字起こしの分析を中止する必要がある場合は、`BotRecommendationStatus` ステータスが「処理中」 になっている実行中の `BotRecommendation` ジョブを停止できます。コンソールからジョブを送信した後、または `StopBotRecommendation` API 用の CLI SDK を使用して、バナーに表示される [処理を停止] ボタンをクリックできます。詳細については、「StopBotRecommendation」を参照してください。

StopBotRecommendation を呼び出した後、内部 BotRecommendationStatus 料金が Stopping に設定され、請求されません。ジョブが停止したことを確認するには、DescribeBotRecommendation API を呼び出して、BotRecommendationStatus が Stopped であることを確認します。通常、これには 3～4 分かかります。

StopBotRecommendation API が呼び出された後の処理には課金されません。

インテントとスロットタイプの作成

Chatbot デザイナーがインテントとスロットタイプを作成したら、ボットに追加するインテントとスロットタイプを選択します。各インテントとスロットタイプの詳細を確認して、どのレコメンデーションがユースケースに最も関連しているかを判断できます。

推奨インテントの名前をクリックすると、Chatbot Designer が提案したサンプル発話とスロットが表示されます。[関連する文字起こしを表示] を選択すると、入力した会話をスクロールすることもできます。これらの会話記録は、Chatbot Designer がこのインテントを推奨するかどうかに影響します。サンプル発話をクリックすると、主要な会話と、その特定の発話に影響を与えた関連する会話のターンを確認できます。

特定のスロットタイプの名前をクリックすると、推奨されているスロット値が表示されます。[関連する文字起こしを表示] を選択すると、そのスロットタイプに対して表示されるエージェントプロンプトが強調表示された状態で、そのスロットタイプに影響を与えた会話を確認できます。特定のスロットタイプ値をクリックすると、主要な会話と、その値に影響を与えた関連する会話のターンを確認できます。

インテントとスロットタイプを確認して追加するには

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/lex/ で Amazon Lex コンソールを開きます。
2. ボットのリストから、作業するボットを選択します。
3. 言語の表示 を選択します。
4. 言語のリストから、使用する言語を選択します。
5. 会話の構造 で、確認 を選択します。
6. インテントとスロットタイプのリストで、ボットに追加するインテントとスロットタイプを選択します。インテントまたはスロットタイプを選択して、詳細と関連する文字起こしを表示できます。

インテントは、Amazon Lex V2 がインテントが処理された文字起こしに関連付けられているという信頼度でソートされます。

文字起こしの入力形式

以下は、ボットのインテントとスロットタイプを生成するための入力ファイル形式です。入力ファイルにはこれらのフィールドを含める必要があります。他のフィールドは無視されます。

入力形式は、Contact Lens for Connect Customer からの出力形式と互換性があります。コンタクトレンズを使用している場合は、文字起こしファイルを変更する必要はありません。詳細については、コンタクトレンズ出力ファイルの例 を参照してください。別のコンタクトセンターアプリケーションを使用している場合は、文字起こしファイルをこの形式に変換する必要があります。

```
{
    "Participants": [
        {
            "ParticipantId": "string",
            "ParticipantRole": "AGENT | CUSTOMER"
        }
    ],
    "Version": "1.1.0",
    "ContentMetadata": {
        "RedactionTypes": [
            "PII"
        ],
        "Output": "Raw | Redacted"
    },
    "CustomerMetadata": {
        "ContactId": "string"
    },
    "Transcript": [
        {
            "ParticipantId": "string",
            "Id": "string",
            "Content": "string"
        }
    ]
}
```

入力ファイルに次のフィールドが存在する必要があります。

- 参加者: 会話の参加者と自分が果たす役割を識別します。

- Version: 入力ファイル形式のバージョン。常に「1.1.0」。
- ContentMetadata: 文字起こしから機密情報を削除したかどうかを示します。文字起こしに機密情報が含まれている場合は、Output フィールドを "Raw" に設定します。
- CustomerMetadata: 会話の一意の識別子。
- 文字起こし: 会話中の当事者間の会話のテキスト。会話の各ターンは一意の識別子で識別されます。

文字起こしの出力形式

出力文字起こしの形式は、入力文字起こし形式とほぼ同じです。ただし、一部の顧客メタデータと、インテントとスロットタイプの提案に影響を与えたフィールドリストセグメントも含まれます。出力文字起こしは、コンソール の確認 から、または Amazon Lex V2 API を使用してページをダウンロードできます。詳細については、「文字起こしの入力形式」を参照してください。

```
{
    "Participants": [
        {
            "ParticipantId": "string",
            "ParticipantRole": "AGENT | CUSTOMER"
        }
    ],
    "Version": "1.1.0",
    "ContentMetadata": {

        "RedactionTypes": [
            "PII"
        ],
        "Output": "Raw | Redacted"
    },
    "CustomerMetadata": {
        "ContactId": "string",
        "FileName": "string",
        "InputFormat": "Lex"
    },
    "InfluencingSegments": [
        {
            "Id": "string",
            "StartTurnIndex": number,
            "EndTurnIndex": number,
            "Intents": [
```

```
                {
                    "Id": "string",
                    "Name": "string",
                    "SampleUtteranceIndex": [
                        {
                            "Index": number,
                            "Content": "String"
                        }
                    ]
                }
            ],
            "SlotTypes": [
                {
                    "Id": "string",
                    "Name": "string",
                    "SlotValueIndex": [
                        {
                            "Index": number,
                            "Content": "String"
                        }
                    ]
                }
            ]
        }
    ],
    "Transcript": [
        {

            "ParticipantId": "string",
            "Id": "string",
            "Content": "string"
        }
    ]
}
```

- CustomerMetadata – CustomerMetadataフィールドには、2 つのフィールドが追加されています。会話を含む入力ファイルの名前と、常に「Lex」という入力形式が表示されます。
- InfluencingSegments: インテントまたはスロットタイプの提案に影響を与えた会話のセグメントを識別します。インテントまたはスロットタイプの ID は、会話の影響を受ける特定のものを識別します。

Amazon Lex V2 ボットへの新しい言語の追加

ボットに 1 つ以上の言語とロケールを追加して、ボットがユーザーの言語でコミュニケーションできるようにします。発話、プロンプト、スロット値が言語に固有になるように、各言語にインテント、スロット、スロットタイプを個別に定義します。

ボットには、少なくとも 1 つの言語を含める必要があります。

ボットに言語を追加するには

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/lex/ で Amazon Lex コンソールを開きます。
2. [ボット] セクションで、言語を追加するボットを選択します。
3. [言語を追加] セクションで、[言語を表示] をクリックします。
4. [すべての言語] セクションで、[言語を追加] をクリックします。
5. [新しい言語を追加] セクションで、[言語を最初から追加する] を選択します。
6. [言語の詳細] セクションで、追加したい語を選択します。
7. ボットが音声インタラクションをサポートしている場合、[音声] セクションで、Amazon Lex V2 がユーザーとの通信に使用する Amazon Polly 音声を選択します。ボットが音声をサポートしていない場合は、None を選択します。
8. [分類信頼スコアのしきい値] セクションで、インテントが正しいインテントかどうかを判断するために Amazon Lex V2 が使用する値を設定します。この値は、ボットのテスト後に調整できます。
9. [Add] (追加) を選択します。

インテントの追加

インテントとは、花の注文やホテルの予約など、ユーザーが達成を希望する目標です。ボットには少なくとも 1 つのインテントが必要です。

デフォルトでは、すべてのボットに単一の組み込みインテント、フォールバックインテントが含まれています。このインテントは、Amazon Lex V2 が他のインテントを認識しない場合に使用されます。例えば、ユーザーがホテルの予約インテントに「花を注文したい」と言うと、フォールバックインテントがトリガーされます。

インテントを追加するには

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/lex/ で Amazon Lex コンソールを開きます。
2. ボットのリストで、インテントを追加するボットを選択し、Add languages から View languages を選択します。
3. インテントを追加する言語を選択し、次に Intents を選択します。
4. Add intent を選択し、インテントに名前を付けて、Add を選択します。
5. インテントエディタで、インテントの詳細を追加します。

   - Conversation flow — 会話フロー図表を使用して、ボットとのダイアログがどのように表示されるかを確認します。会話のさまざまなセクションを選択して、インテントエディタのそのセクションにジャンプできます。
   - Intent details — インテントの目的を識別するのに役立つ名前と説明をインテントに与えます。Amazon Lex V2 がインテントに割り当てた一意の識別子も確認できます。
   - Contexts — インテントの入力コンテキストと出力コンテキストを設定します。コンテキストは、インテントに関連付けられた状態変数です。インテントが満たされると、出力コンテキストが設定されます。入力コンテキストを持つインテントは、コンテキストがアクティブである場合にのみ認識されます。入力コンテキストのないインテントは、常に認識されます。
   - Sample utterances — ユーザーがインテントを開始するために使用すると思われるフレーズを 10 個以上指定する必要があります。Amazon Lex V2 は、ユーザーがインテントを開始したいことを認識するために、これらのフレーズから一般化します。
   - 初期応答 — インテントが呼び出された後にユーザーに送信される最初のメッセージ。レスポンスを提供し、値を初期化し、インテントの開始時に Amazon Lex V2 がユーザーに応答するために実行する次のステップを定義できます。
   - Slots — インテントを達成するために必要なスロットまたはパラメータを定義します。各スロットには、スロットに入力できる値を定義するタイプがあります。カスタムスロットタイプから選択するか、ビルトインスロットタイプを選択できます。
   - 確認 — これらの応答は、ユーザーとの会話を終了し、インテントのフルフィルメントを確認または拒否するために使用されます。確認プロンプトは、スロット値を確認するようにユーザに要求します。例えば、「金曜日のホテルの部屋を予約しました。正しいですか？ 拒否応答は、彼らが確認を拒否したときにユーザーに送信されます。応答を提供し、値を設定し、ユーザーからの確認または拒否の応答に対応して Amazon Lex V2 が実行する次のステップを定義できます。

- フルフィルメント — フルフィルメントの過程でユーザーに送信される応答。フルフィルメント開始時およびフルフィルメントの進行中に定期的にフルフィルメントの進捗状況を更新するように設定できます。例えば、「パスワードを変更しています。これには数分かかる場合があります」や「まだリクエストを処理しています」などです。フルフィルメントの更新は、ストリーミング会話にのみ使用されます。フルフィルメント後の成功メッセージ、失敗メッセージ、およびタイムアウトメッセージも設定できます。ストリーミングと通常の会話の両方で、フルフィルメント後のメッセージを送信できます。例えば、フルフィルメントが成功した場合は、「パスワードを変更しました」を送信できます。フルフィルメントが成功しない場合は、「パスワードを変更できませんでした。ヘルプデスクにお問い合わせください」などの詳細情報を含む回答を送信できます。フルフィルメントに時間がかかりすぎて、設定されたタイムアウト期間を超える場合は、「当社のサーバーは非常にビジー状態です」など、のメッセージをユーザーに送ることができます。後でリクエストを再試行してください。応答を提供し、値を設定し、Amazon Lex V2 がユーザーに応答するために実行する次のステップを定義できます。
- 終了応答 — インテントが満たされ、他のすべてのメッセージが再生された後にユーザーに送信される応答。例えば、ホテルの部屋を予約してくれてありがとうございます または、「部屋を予約してくれてありがとうございます、レンタカーを予約しますか？」など、別のインテントを始めるためにユーザーに促すこともできます。インテントを達成して終了応答で応答した後に、応答を提供し、フォローアップの次のアクションを設定できます。
- コードフック — インテントを初期化し、ユーザー入力を検証するために AWS Lambda の関数を使用しているかどうかを示します。ボットの実行に使用するエイリアスで Lambda 関数を指定します。

6. Save intent を選択して、インテントを保存します。

> **Note**
>
> 2022 年 8 月 17 日、Amazon Lex V2 はユーザーとの会話の管理方法の変更をリリースしました。この変更により、ユーザーが会話の中でたどるパスをより細かく制御できるようになりました。詳細については、「Amazon Lex V2 の会話フローの変更 V2」を参照してください。2022 年 8 月 17 日より前に作成されたボットは、ダイアログコードフックメッセージ、値の設定、次のステップの設定、条件の追加をサポートしていません。

特定の順序でのプロンプトの設定

[メッセージを順番に再生] のチェックボックスをオンにすると、あらかじめ定義された順序でメッセージを再生するようにボットを設定できます。それ以外の場合、ボットはメッセージとバリエーションをランダムな順序で再生します。

順序付けられたプロンプトでは、メッセージグループのメッセージとバリエーションを再試行するたびに順番に再生できます。ユーザーがプロンプトに対して無効な応答をした場合や、インテントを確認するために、メッセージを別の言い回しにすることができます。各スロットには、元のメッセージのバリエーションを最大 2 つ設定できます。メッセージを順番に再生するか、ランダムに再生するかを選択できます。

順序付きプロンプトは、テキスト、カスタムペイロード応答、SSML、カードグループの 4 種類のメッセージすべてをサポートします。応答は同じメッセージグループ内で順序付けられます。メッセージグループはそれぞれ独立しています。

トピック

- サンプル発話
- インテントの構造
- 会話パスの作成
- ビジュアル会話ビルダーの使用
- 組み込みのインテント

サンプル発話

ユーザーがインテントを開始するために使用するであろうフレーズのバリエーションであるサンプル発話を作成します。たとえば、**BookFlight** インテントには次のような発話を含めることができます。

1. 飛行機を予約したい
2. 飛行機の予約を取って
3. 飛行機のチケットをお願い!
4. {*DepartureCity*} から {*DestinationCity*} の飛行機

10 個以上のサンプル発話を提供する必要があります。ユーザーが発するであろう幅広い文の構造や単語を表すサンプルを提供してください。上記の例 3 と 4 のように、不完全な文も考慮してく

ださい。例 4 の {*DepartureCity*} のように、スロット名を中括弧で囲むことで、サンプル発話でインテント用に定義したスロットを使用することもできます。サンプル発話にスロット名を含めると、Amazon Lex V2 はユーザーが発話で指定した値をインテントのスロットに入力します。

Amazon Lex V2 は、ユーザーがインテントを開始したいことを効果的に認識するために、さまざまなサンプル発話から Amazon Lex V2 が一般化します。

サンプル発話は、インテントエディター、ビジュアル会話ビルダー、または CreateIntent または UpdateIntent API オペレーションを使用して追加できます。Amazon Bedrock の生成 AI 機能を活用して、サンプル発話を自動的に生成することもできます。詳細については、「発話生成を使用したインテント認識用のサンプル発話の生成」を参照してください。

インテントエディターまたはビジュアル会話ビルダーを使用します。

1. インテントエディターで、[サンプル発話] セクションに移動します。ビジュアル会話ビルダーの開始ブロックにある [サンプル発話] セクションを探してください。
2. 透明なテキスト **I want to book a flight** が表示されたボックスに、サンプル発話を入力します。[発話を追加] を選択して発話を追加します。
3. 追加したサンプル発話を [プレビュー] または [プレーンテキスト] モードで表示します。プレーンテキストでは、各行が個別の発話です。プレビューモードでは、発話にカーソルを合わせると以下のオプションが表示されます。
   - テキストボックスを選択して発話を編集します。
   - テキストボックスの右側にある [x] ボタンを選択して、発話を削除します。
   - テキストボックスの左側にあるボタンをドラッグして、サンプル発話の順序を変更します。
4. 上部の検索バーを使用してサンプル発話を検索し、その横にあるドロップダウンメニューを使用して発話を追加した順序またはアルファベット順に並べ替えます。

API オペレーションを使用する

1. CreateIntent オペレーションで新しいインテントを作成するか、UpdateIntent オペレーションで既存のインテントを更新します。
2. API リクエストには、SampleUtterance オブジェクトの配列にマップされる `sampleUtterances` フィールドが含まれています。
3. 追加するサンプル発話ごとに、配列に `SampleUtterance` オブジェクトを追加します。サンプル発話を `utterance` フィールドの値として追加します。

4. サンプル発話を編集、削除するには、UpdateIntent リクエストを送信します。sampleUtterances フィールドに入力した発話のリストが、既存の発話と置き換わります。

> **Important**
>
> UpdateIntent リクエストでフィールドを空白のままにすると、インテント内の既存の設定が削除されます。DescribeIntent オペレーションを使用してボットの設定を返し、削除したくない設定を UpdateIntent リクエストにコピーします。

インテントの構造

インテントとは、花の注文やホテルの予約など、ユーザーが達成を希望する目標です。ボットには少なくとも 1 つのインテントが必要です。インテントは以下のコンポーネントで構成されます。

- 初期応答 — インテントが呼び出された後にユーザーに送信される最初のメッセージ。レスポンスを設定し、値を初期化し、インテントの開始時にボットがユーザーに応答するために実行する次のステップを定義できます。
- スロット – インテントを満たすために必要なパラメータ。各スロットには、スロットに入力できる値を定義するタイプがあります。カスタムスロットタイプから選択するか、ビルトインスロットタイプを選択できます。
- 確認 – ユーザーとの会話が完了し、インテントのスロット値が入力されたら、スロット値が正しいかどうかをユーザーに尋ねる確認プロンプトを設定できます。
- フルフィルメント — フルフィルメントの過程でユーザーに送信される応答。フルフィルメント開始時にフルフィルメントの進捗状況を更新するように設定し、フルフィルメントの進行中に定期的に更新情報を送信できます。フルフィルメント後の成功メッセージ、失敗メッセージ、およびタイムアウトメッセージも設定できます。
- 終了応答 – ユーザーのインテントが達成した後に送信される終了応答。終了応答を設定して会話を終了したり、別のインテントで続行できることをユーザーに知らせたりできます。

トピック

- 初期応答
- Slots
- 確認

- フルフィルメント
- 終了応答

初期応答

Amazon Lex V2 がインテントを判断し、スロット値を取得し始める前に、初期応答がユーザーに送信されます。この応答を使用して、認識されたインテントをユーザーに伝え、そのインテントを実現するために収集した情報をユーザーに理解してもらうことができます。

たとえば、自動車のサービス予約を予定している場合、最初の応答は次のようになります。

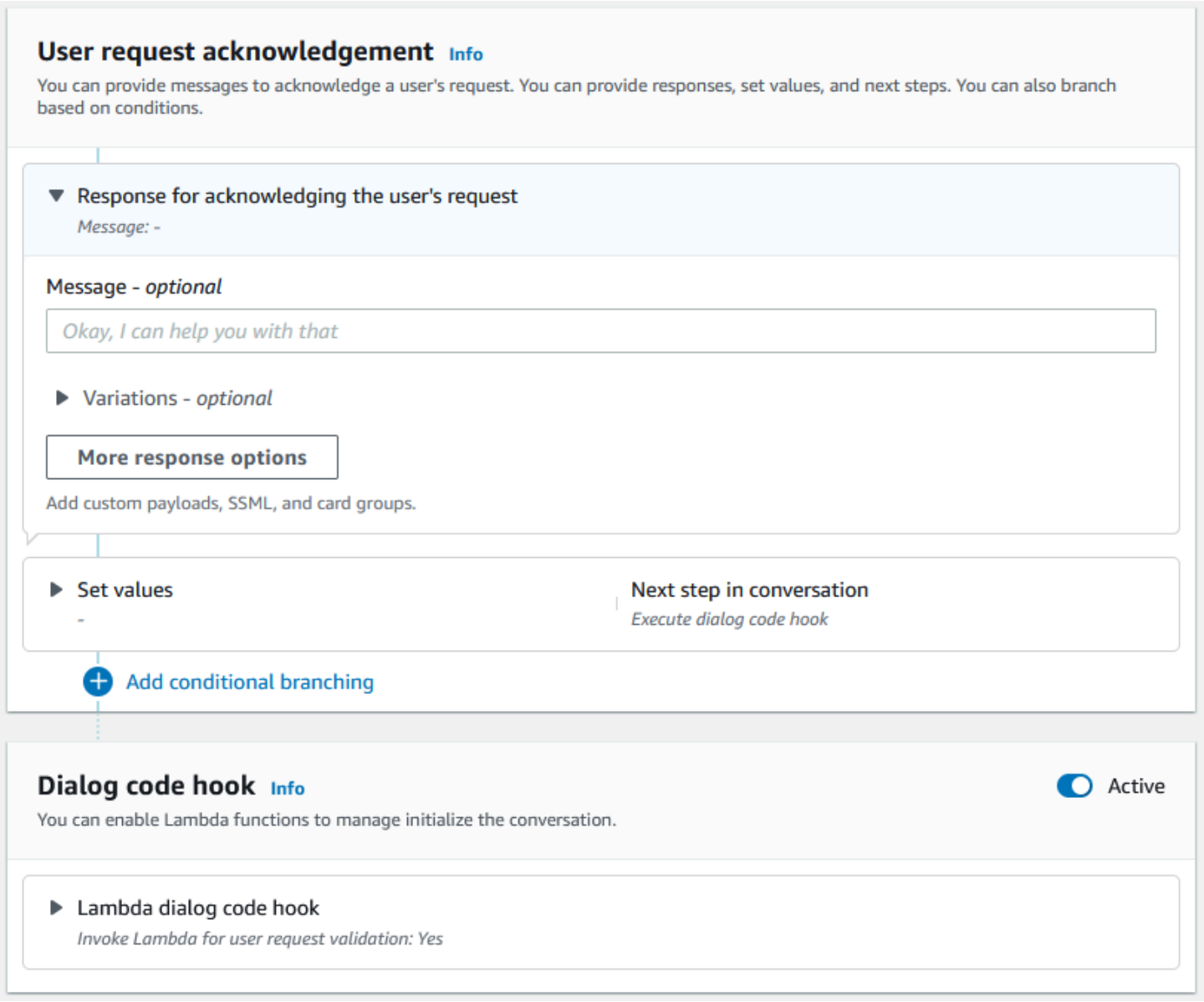
> 予約を取るお手伝いをします。車のメーカー、モデル、年式を提供する必要があります。

最初の応答メッセージは必須ではありません。指定しない場合、Amazon Lex V2 は引き続き初期応答の次のステップに従います。

初期応答で次のオプションを設定できます。

- 次のステップの設定 — 特定のダイアログアクションへのジャンプ、特定のスロットの誘発、別のインテントへのジャンプなど、会話の次のステップを指定できます。詳細については、「会話の次のステップを設定します。」を参照してください。
- 値の設定 — スロットとセッション属性の値を設定できます。詳細については、会話中に値を設定するを参照してください。
- 条件分岐の追加 — 最初のレスポンスを再生した後に条件を適用できます。条件が true に評価された場合、定義したアクションが実行されます。詳細については、「ブランチ会話への条件の追加」を参照してください。
- ダイアログコードフックを実行 — データを初期化し、ビジネスロジックを実行する Lambda コードフックを定義できます。詳細については、「ダイアログコードフックを呼び出す」を参照してください。Lambda 関数を実行するオプションがインテントで有効になっている場合、ダイアログコードフックはデフォルトで実行されます。[アクティブ] ボタンを切り替えることでダイアログコードフックを無効にできます。

条件がない場合や次のステップが明示的に指定されていない場合、Amazon Lex V2 は優先順位に従って次のスロットに移動します。



> **Note**
>
> 2022 年 8 月 17 日、Amazon Lex V2 はユーザーとの会話の管理方法の変更をリリースしました。この変更により、ユーザーが会話の中でたどるパスをより細かく制御できるようになりました。詳細については、「Amazon Lex V2 の会話フローの変更 V2」を参照してください。2022 年 8 月 17 日より前に作成されたボットは、ダイアログコードフックメッセージ、値の設定、次のステップの設定、条件の追加をサポートしていません。

Slots

スロットは、インテントを満たすためにユーザーが提供する値です。スロットには、次の 2 種類があります。

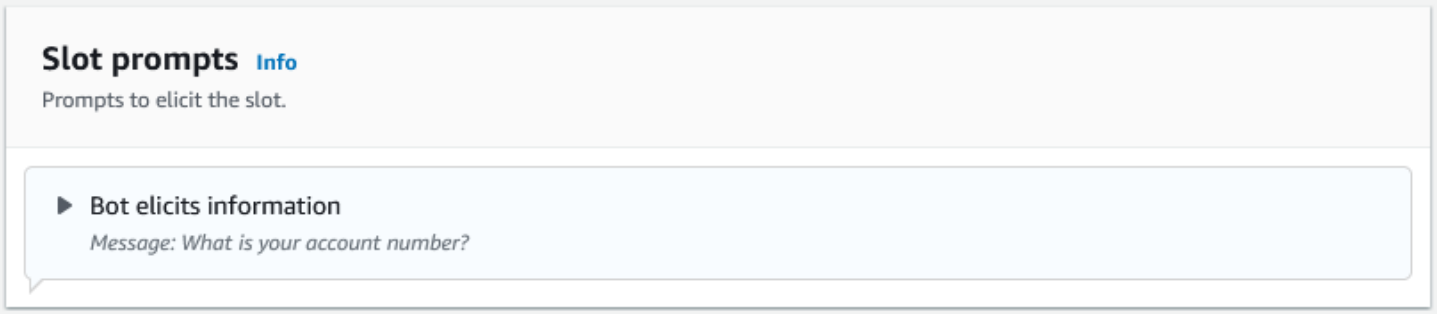
- 組み込みスロットタイプ — 組み込みスロットタイプを使用して、番号、名前、都市などの標準値を取得できます。サポートされている組み込みスロットタイプについては、「組み込みスロットタイプ」を参照してください。
- カスタムスロットタイプ — カスタムスロットタイプを使用して、インテント固有のカスタム値をキャプチャできます。たとえば、カスタムスロットタイプを使用して、口座タイプを「当座預金」または「貯蓄」としてキャプチャできます。詳細については、「カスタムスロットタイプ」を参照してください。

インテントにスロットを定義するには、以下を設定する必要があります。

- スロット情報 — このフィールドにはスロットの名前と説明 (任意) が含まれます。たとえば、スロット名を「AccountNumber」として指定すると、アカウント番号を取得できます。インテントを満たすために会話フローの一部としてそのスロットが必要な場合は、必須とマークする必要があります。
- スロットタイプ — スロットタイプは、スロットが受け付けることができる値のリストを定義します。カスタムスロットタイプを作成することも、定義済みのスロットタイプを使用することもできます。
- スロットプロンプト — スロットプロンプトは、情報を収集するためにユーザーに提示される質問です。情報を収集するために使用する再試行の回数と、各再試行に使用するプロンプトのバリエーションを設定できます。また、キャプチャした入力を処理して有効な入力への解決を試みるたびに Lambda 関数呼び出しを有効にすることもできます。
- 待機と続行 (オプション) - この動作を有効にすると、ユーザーは「ちょっと待って」などのフレーズを言うことで、ボットに情報の発見と提供を待機させることができます。ストリーミング会話でのみ有効です。詳細については、「一時停止中に、Amazon Lex V2 ボットによるユーザーの追加の情報提供の待機を可能にする」を参照してください。
- スロットキャプチャ応答 — ユーザー入力からスロット値を取得した結果に基づいて、成功の応答と失敗の応答を設定できます。
- 条件分岐 — 最初のレスポンスを再生した後に条件を適用できます。条件が true に評価された場合、定義したアクションが実行されます。詳細については、「ブランチ会話への条件の追加」を参照してください。

- ダイアログコードフック — Lambda コードフックを使用してスロット値を検証し、ビジネスロジックを実行することもできます。詳細については、「ダイアログコードフックを呼び出す」を参照してください。
- ユーザー入力タイプ — ボットが特定のモダリティを受け入れるように入力タイプを設定できます。デフォルトでは、音声と DTMF の両方のモダリティが受け入れられます。音声のみ、または DTMF のみに設定できます。
- 音声入力のタイムアウトと長さ — ボイスタイムアウト、サイレンスタイムアウトなどの音声タイムアウトを設定できます。また、音声の最大長も設定できます。
- DTMF 入力タイムアウト、文字数、長さ — DTMF タイムアウトは、削除文字と終了文字とともに設定できます。また、DTMF の最大長も設定できます。
- テキスト長 — テキストモダリティの最大長を設定できます。

スロットプロンプトが再生されたら、ユーザーはスロット値を入力として入力します。Amazon Lex V2 がユーザーから提供されたスロット値を認識しない場合、値を認識するまで、またはスロットに設定された最大再試行回数を超えるまで、スロットの取得を再試行します。再試行の詳細設定を使用すると、タイムアウトを設定したり、入力タイプを制限したり、最初のプロンプトや再試行時の中断を有効または無効にしたりできます。入力のキャプチャを試みるたびに、Amazon Lex V2 は再試行用に用意された呼び出しラベルを使用して、ボット用に設定された Lambda 関数を呼び出すことができます。たとえば、Lambda 関数を使用して、ビジネスロジックを適用して有効な値への解決を試みることができます。この Lambda 関数は、スロットプロンプトの詳細オプションで有効にできます。



スロット値が入力されたとき、または最大リトライ回数を超えたときに、ボットがユーザーに送信するレスポンスを定義できます。たとえば、自動車のサービスをスケジュールするボットの場合、車両識別番号 (VIN) が入力されるとユーザーにメッセージを送信できます。

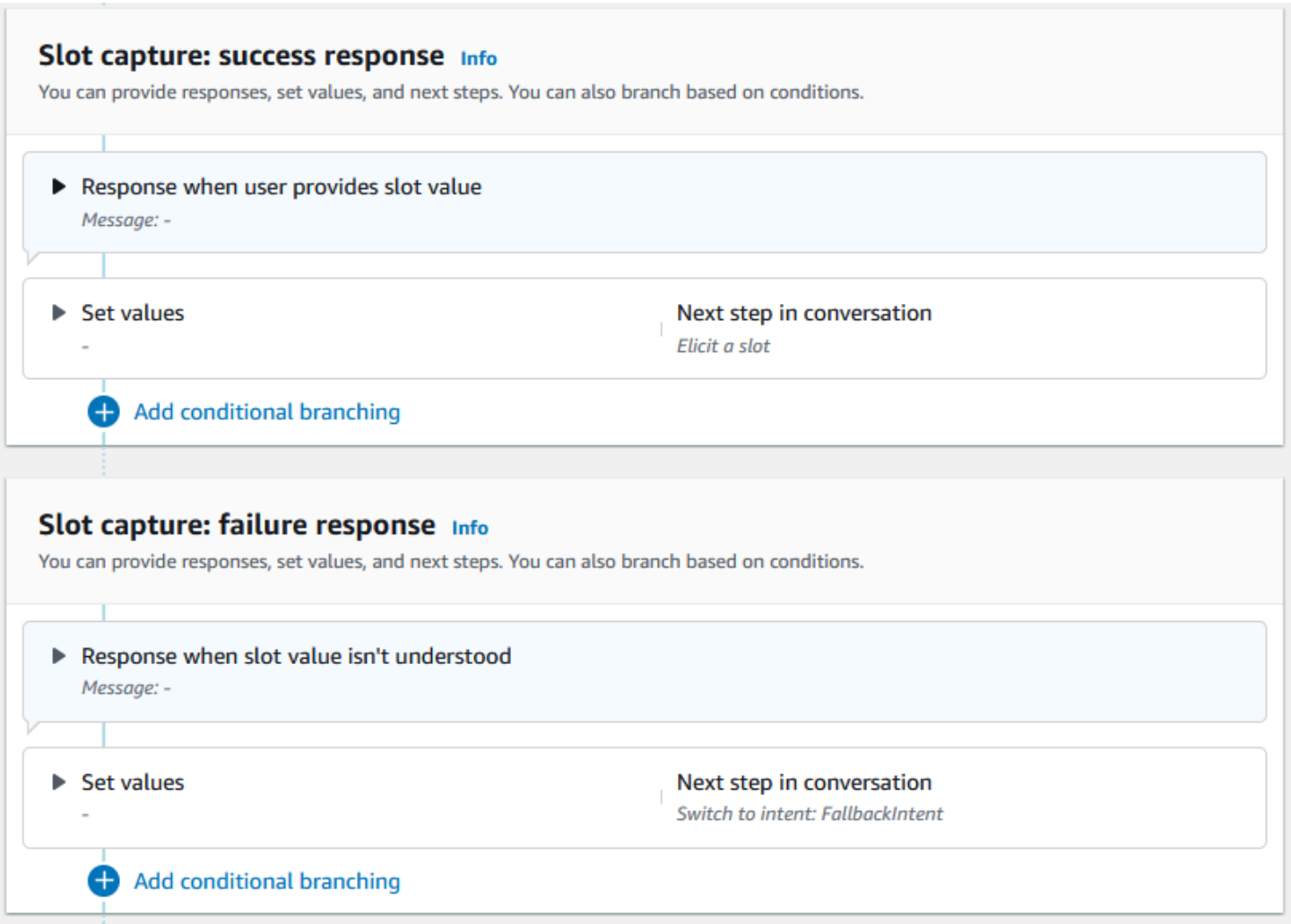
お車の VIN 番号を教えていただきありがとうございます。それでは、予約を入れておきます。

2 つの応答を作成できます。

- 成功の応答 — Amazon Lex V2 がスロット値を認識したときに送信されます。
- 失敗の応答 — Amazon Lex V2 が最大回数再試行してもユーザーからのスロット値を認識できない場合に送信されます。

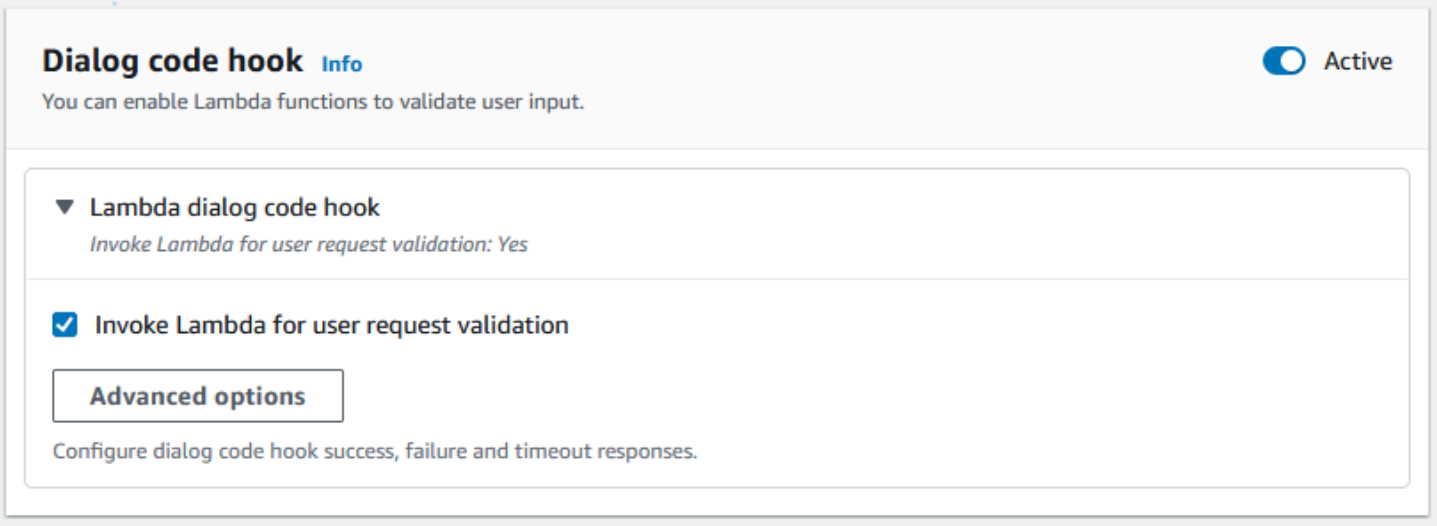
値を設定し、次のステップを設定し、各応答に対応する条件を適用して会話フローを設計できます。

条件がない場合や次のステップが明示的に指定されていない場合、Amazon Lex V2 は優先順位に従って次のスロットに移動します。



Lambda 関数を使用して、ユーザーが入力したスロット値を検証し、次のアクションを決定できます。たとえば、検証関数を使用して、入力された値が正しい範囲内にあることや、正しい形式になっていることを確認できます。Lambda 関数を有効にするには、[Lambda 関数を呼び出す] チェックボックスと [ダイアログコードフック] セクションの [アクティブ] ボタンを選択します。ダイアログ

コードフックの呼び出しラベルを指定できます。この呼び出しラベルを Lambda 関数で使用して、スロット誘発に対応するビジネスロジックを記述できます。



インテントに必要のないスロットは、メインの会話フローには含まれません。ただし、ユーザーの発話に、ボットがオプションスロットに対応すると識別する値が含まれている場合、ボットはその値をスロットに入力できます。たとえば、ビジネスインテリジェンスボットにオプションの `City` スロットとユーザー発話 **`What is the sales for April in San Diego?`** を持たせるように設定した場合、ボットは **`San Diego`** でオプションのスロットを埋めます。オプションのスロット値がある場合は、その値を使用するようにビジネスロジックを設定できます。

インテントに必要のないスロットは、次のステップでは誘発できません。これらのステップは、(前の例のように) インテントの誘発中にのみ入力することも、Lambda 関数内でダイアログの状態を設定することによって生成することもできます。Lambda 関数を使用してスロットを誘発する場合は、スロットの誘発が完了した後に Lambda 関数を使用して会話の次のステップを決定する必要があります。ボットの構築中に次のステップのサポートを有効にするには、スロットをインテントに必要なものとしてマークする必要があります。

> **Note**
>
> 2022 年 8 月 17 日、Amazon Lex V2 はユーザーとの会話の管理方法の変更をリリースしました。この変更により、ユーザーが会話の中でたどるパスをより細かく制御できるようになりました。詳細については、「Amazon Lex V2 の会話フローの変更 V2」を参照してください。2022 年 8 月 17 日より前に作成されたボットは、ダイアログコードフックメッセージ、値の設定、次のステップの設定、条件の追加をサポートしていません。

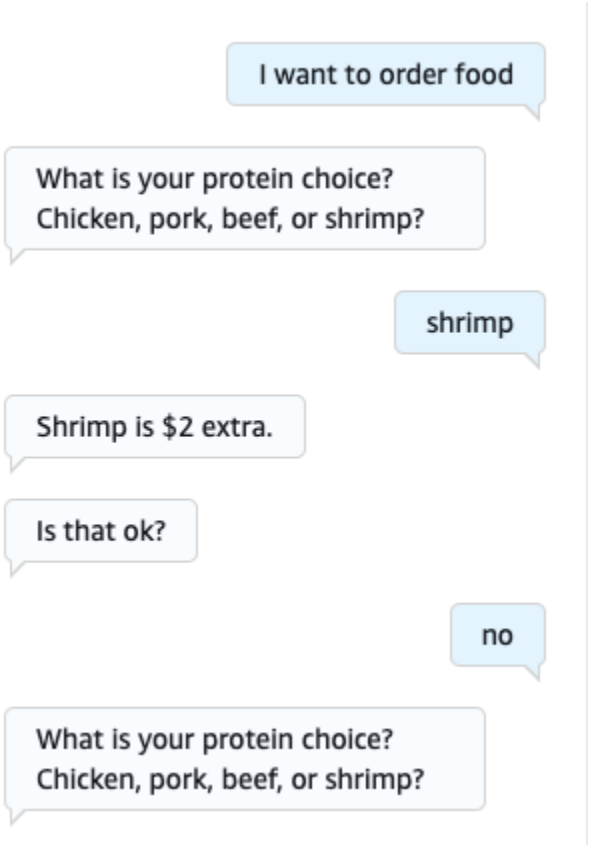
以下のトピックでは、すでに入力されているスロット値を再取得するようにボットを設定する方法と、複数の値で構成されるスロットを作成する方法について説明します。

トピック

- 再誘発スロット
- スロットで複数の値を使用する

再誘発スロット

スロット値を **null** に設定し、会話の次のステップでそのスロットを誘発するようにループバックするように設定することで、すでに埋まっているスロットを再利用するようにボットを設定できます。たとえば、次の会話のように、顧客が追加情報に基づいてスロット誘発の確認を拒否した後に、スロットを再誘発したい場合があります。



インテントエディターまたは ビジュアル会話ビルダーの使用 を使用して、確認応答から戻ってスロットを再表示するループを設定できます。

> **Note**
>
> スロット値を事前に **null** に設定しておけば、会話のどの時点でもループバックしてスロットを再利用することができます。

上記の例をインテントエディターで再現します。

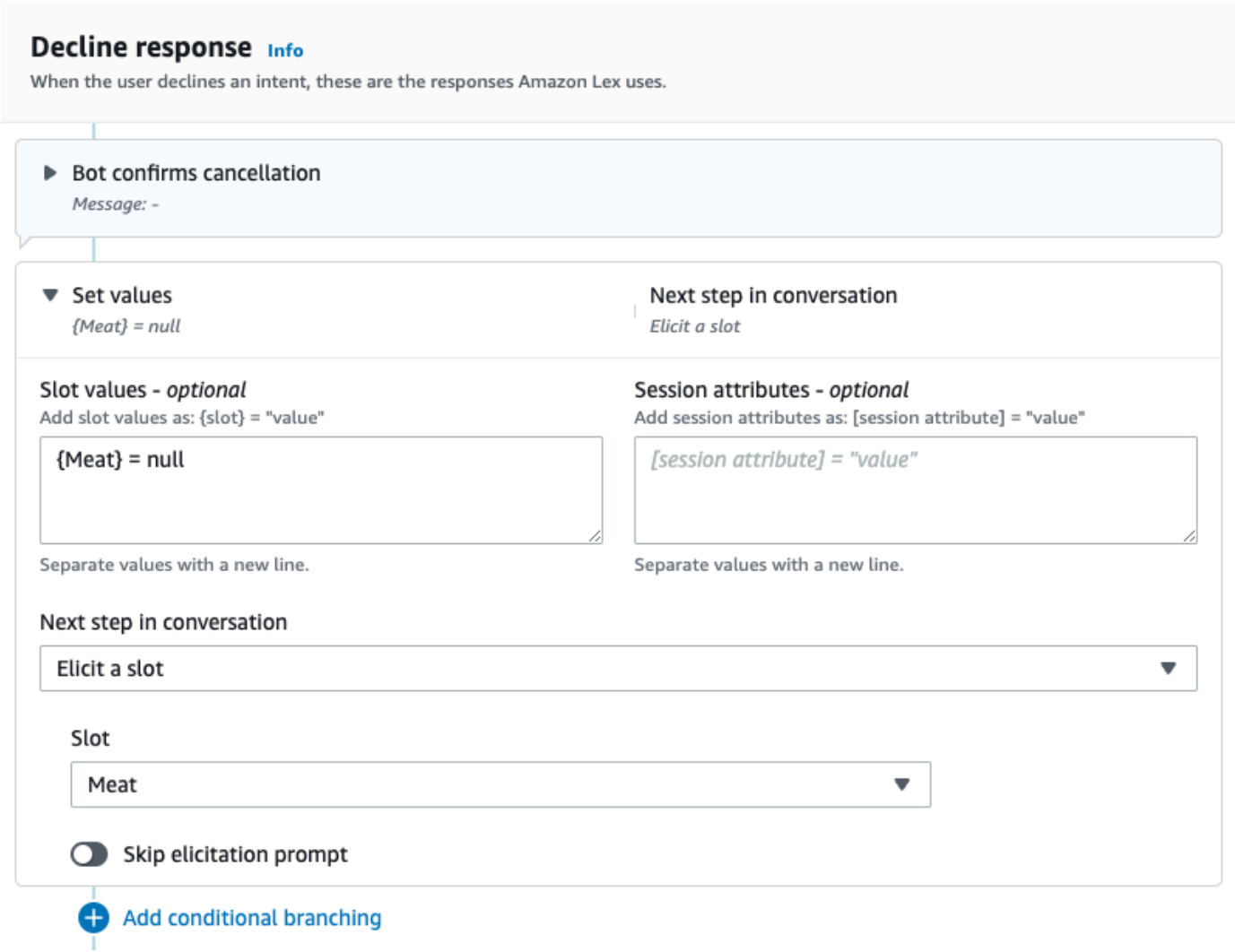
1. インテントエディターの [確認] セクションで、[インテントを確認するプロンプトを表示] の横にある右矢印を選択して、セクションを拡張します。
2. 下部にある [詳細オプション] を選択します。
3. [回答を拒否] セクションで、[値を設定] の横にある右矢印を選択してセクションを展開します。以下の画像のように、このセクションに次の手順を入力します。

    a. **null** に再誘発するスロット値を設定します。この例では、`Meat` スロットを再誘発したいので、[スロット値] セクションに **`{Meat} = null`** を入力します。

    b. [会話の次のステップ] の下のドロップダウンメニューで、[スロットを誘発] を選択します。

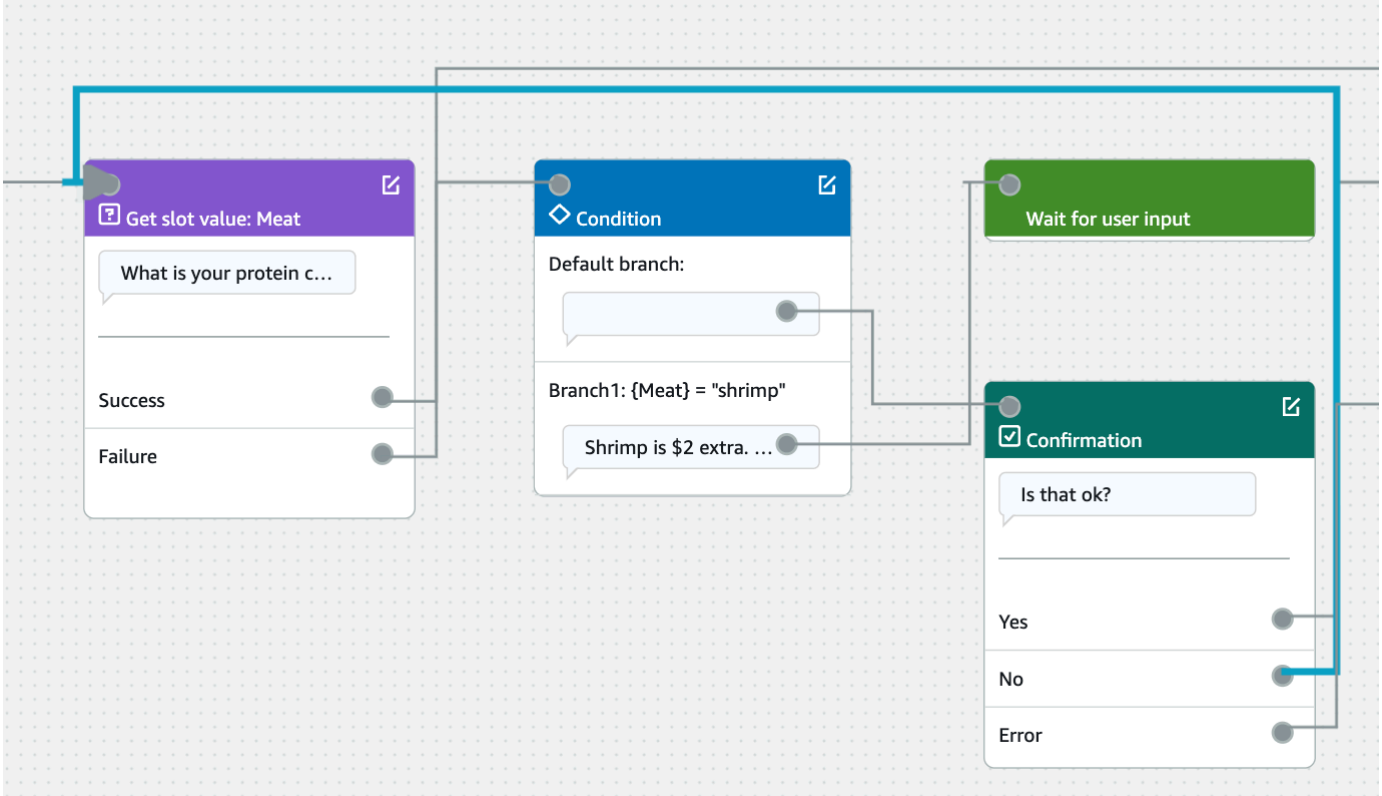
    c. [スロット] セクションが表示されます。その下のドロップダウンメニューで、再誘発したいスロットを選択します。

    d. [更新オプション] を選択して変更を確定します。

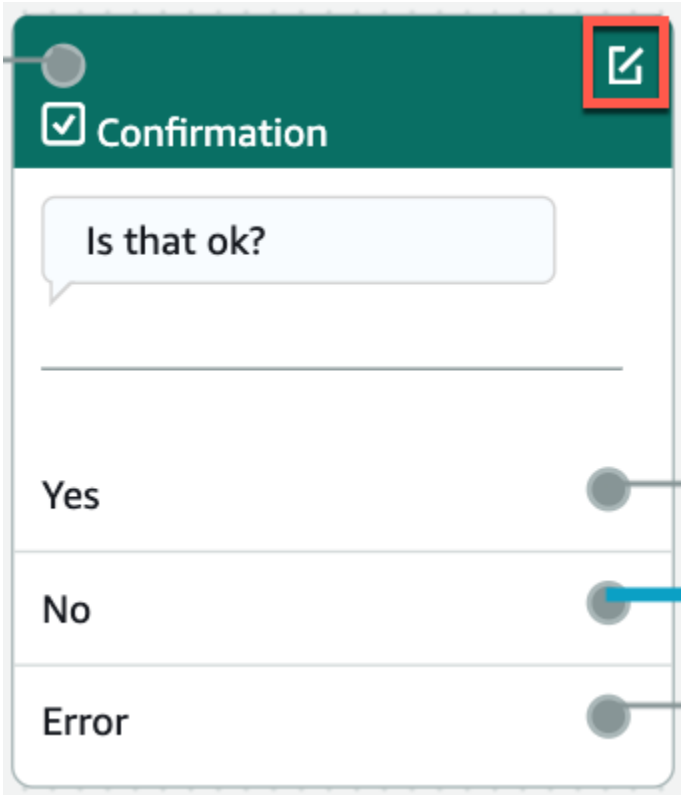


上の例をビジュアル会話ビルダーで再現しています。

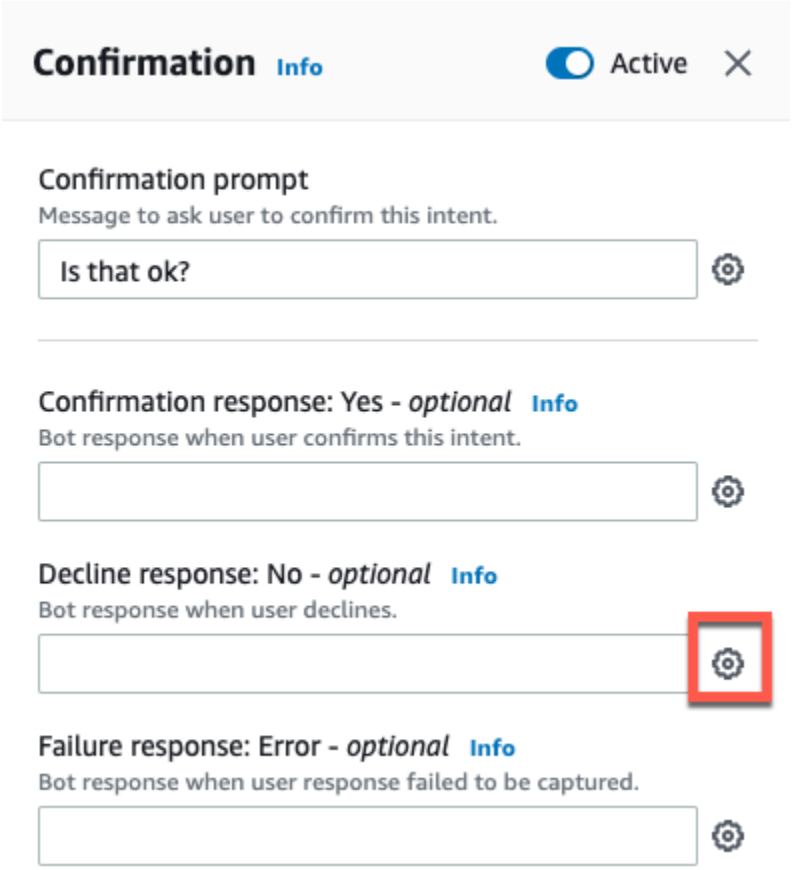
1. [確認] ブロックの [いいえ] ポートから [スロット値の取得: 肉] ブロックの受信ポートへの接続を作成します。



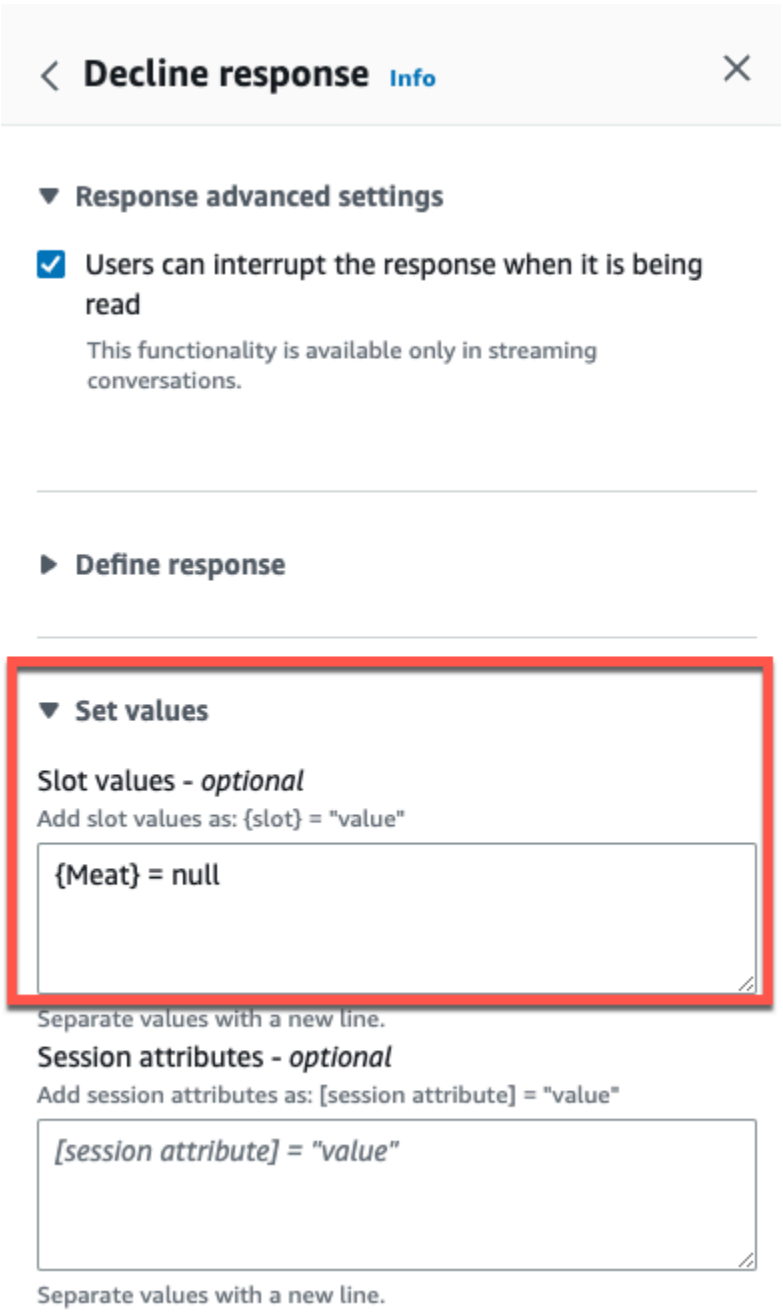
2. [確認] ブロックの右上隅にある [編集] アイコンを選択します。



3. [応答拒否] セクションのボット応答の横にある歯車アイコンを選択します。



4. [値を設定] セクションの [スロット値] ボックスに "{Meat} = null" と入力します。



5. [インテントの保存] を選択します。

スロットで複数の値を使用する

> **Note**
>
> 複数の値スロットは、英語 (米国語) でのみサポートされています。

インテントによっては、1 つのスロットに対して複数の値をキャプチャしたい場合があります。たとえば、ピザ注文ボットは次のような発話のインテントを持っていることがあります。

```
I want a pizza with {toppings}
```

このインテントは、{toppings} スロットに、カスタマーが自分のピザに欲しいトッピング、例えば「ペパロニとパイナップル」のリストが含まれていることを想定しています。

複数の値をキャプチャするようにスロットを設定するには、スロットの allowMultipleValues フィールドを true に設定します。フィールドの設定は、コンソールを使って行うか、CreateSlot または UpdateSlot のオペレーションで行うことができます。

カスタムスロットタイプのスロットのみを複数値スロットとしてマークできます。

複数値スロットの場合、Amazon Lex V2 は RecognizeText または RecognizeUtterance オペレーションへの応答でスロット値のリストを返します。以下は、ピザ注文ボットからの「ペパロニとパイナップルのピザが欲しい」という発話で返されるスロット情報です。

```
    "slots": {
        "toppings": {
            "shape": "List",
            "value": {
                "interpretedValue": "pepperoni and pineapple",
                "originalValue": "pepperoni and pineapple",
                "resolvedValues": [
                    "pepperoni and pineapple"
                ]
            },
            "values": [
                {
                    "shape": "Scalar",
                    "value": {
                        "interpretedValue": "pepperoni",
                        "originalValue": "pepperoni",
                        "resolvedValues": [
                            "pepperoni"
                        ]
                    }
                },
                {
                    "shape": "Scalar",
                    "value:": {
```

```
                    "interpretedValue": "pineapple",
                    "originalValue": "pineapple",
                    "resolvedValues": [
                        "pineapple"
                    ]
                }
            }
        ]
    }
}
```

複数値のスロットは、常に値のリストを返します。発話に含まれる値が 1 つだけの場合、返される値のリストにはレスポンスが 1 つだけ含まれます。

Amazon Lex V2 では、スペース、カンマ (,)、および「and」で区切られた複数の値を認識します。複数値スロットは、テキスト入力と音声入力の両方で動作します。

プロンプトでは複数値スロットを使用できます。たとえば、インテントの確認プロンプトを次のように設定し

```
Would you like me to order your {toppings} pizza?
```

Amazon Lex V2 がユーザーにプロンプトを送信すると、「ペパロニとパイナップルのピザを注文しましょうか？」 と送信されます。

複数値スロットは単一のデフォルト値をサポートします。複数のデフォルト値が指定されている場合、Amazon Lex V2 は最初に使用可能な値のみをスロットに入力します。詳細については、「Lex V2 ボットのインテントでのデフォルトスロット値の使用」を参照してください。

スロット難読化を使用すると、会話ログの複数値スロットの値をマスクすることができます。スロット値を難読化すると、各スロット値の値がスロットの名前に置き換えられます。詳細については、「Lex V2 の会話ログでスロット値を隠す」を参照してください。

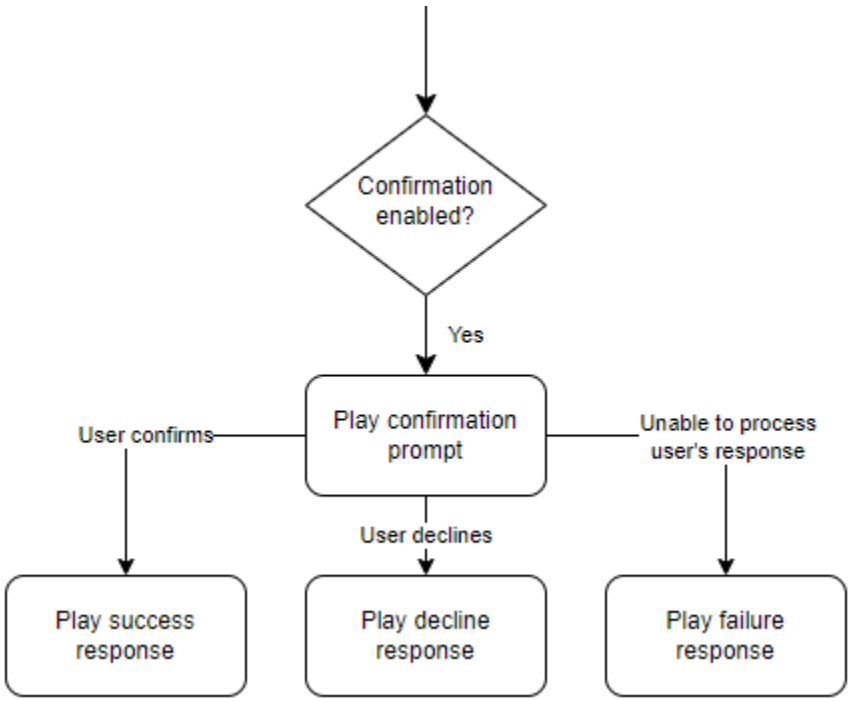
確認

ユーザーとの会話が完了し、インテントのスロット値が入力されたら、スロット値が正しいかどうかをユーザーに尋ねる確認プロンプトを設定できます。たとえば、自動車のサービスを予約するボットは、ユーザーに次のようなプロンプトを出すことがあります。

2017 年のホンダシビックのサービスを 3 月 25 日午後 3 時に予定しています。よろしいですか?

確認プロンプトには次の 3 種類の応答を定義できます。

- 確認の応答 — この応答は、ユーザーがインテントを確認したときにユーザーに送信されます。たとえば、ユーザーが「注文しますか?」というプロンプトに「はい」と答えた後などです。
- 拒否の応答 — この応答は、ユーザーがインテントを拒否したときにユーザーに送信されます。たとえば、ユーザーが「注文しますか?」というプロンプトに「いいえ」と答えた後などです。
- 失敗の応答 — この応答は、確認プロンプトを処理できないときにユーザーに送信されます。たとえば、ユーザーの応答が理解できなかったり、「はい」または「いいえ」という応答ができなかった場合などです。

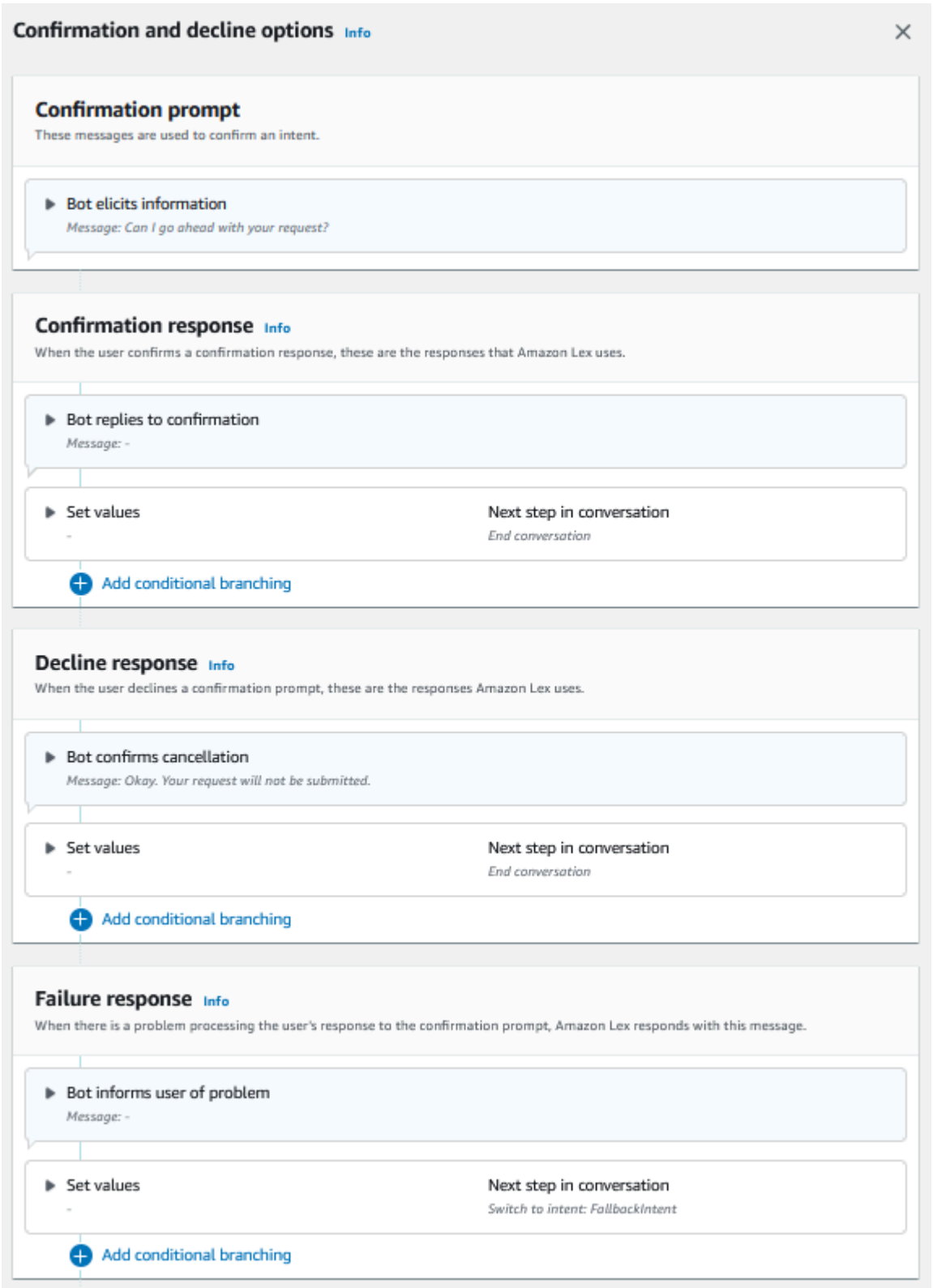


確認プロンプトを指定しない場合、Amazon Lex V2 は達成ステップまたは終了応答に移動します。

値を設定し、次のステップを設定し、各応答に対応する条件を適用して会話フローを設計できます。条件がない場合や次のステップが明示的に指定されていない場合、Amazon Lex V2 はフルフィルメントステップに移動します。

また、ダイアログコードフックを有効にして、インテントにキャプチャされた情報を検証してからフルフィルメントに送ることもできます。コードフックを使用するには、確認プロンプトの詳細オプションでダイアログコードフックを有効にします。さらに、前の状態の次のステップでダイアログコードフックを実行するように設定します。詳細については、「ダイアログコードフックを呼び出す」を参照してください。

> **Note**
>
> コードフックを使用して実行時に確認ステップをトリガーする場合は、構築時に確認ステップを [アクティブ] とマークする必要があります。


Confirmation and decline options Info
Confirmation prompt
These messages are used to confirm an intent.
Bot elicits information
Message: Can I go ahead with your request?
Confirmation response Info
When the user confirms a confirmation response, these are the responses that Amazon Lex uses.
Bot replies to confirmation
Message: -
Set values
-
Next step in conversation
End conversation
Add conditional branching
Decline response Info
When the user declines a confirmation prompt, these are the responses Amazon Lex uses.
Bot confirms cancellation
Message: Okay. Your request will not be submitted.
Set values
-
Next step in conversation
End conversation
Add conditional branching
Failure response Info
When there is a problem processing the user's response to the confirmation prompt, Amazon Lex responds with this message.
Bot informs user of problem
Message: -
Set values
-
Next step in conversation
Switch to intent: FallbackIntent
Add conditional branching

> **Note**
>
> 2022 年 8 月 17 日、Amazon Lex V2 はユーザーとの会話の管理方法の変更をリリースしました。この変更により、ユーザーが会話の中でたどるパスをより細かく制御できるようになりました。詳細については、「Amazon Lex V2 の会話フローの変更 V2」を参照してください。2022 年 8 月 17 日より前に作成されたボットは、ダイアログコードフックメッセージ、値の設定、次のステップの設定、条件の追加をサポートしていません。

Lambda 関数を使用してインテントを検証します。

Lambda コードフックを定義して、フルフィルメントのために送信する前にインテントを検証できます。コードフックを使用するには、確認プロンプトの詳細オプションでダイアログコードフックを有効にします。

コードフックを使用すると、コードフックの実行後に Amazon Lex V2 が実行するアクションを定義できます。次の 3 種類の応答を作成できます。

- 成功の応答 — コードフックが正常に完了するとユーザーに送信されます。
- 失敗の応答 — コードフックが正常に実行されなかった場合、またはコードフックが応答で `Failure` を返した場合にユーザーに送信されます。
- タイムアウト応答 — 設定したタイムアウト期間内にコードフックが完了しなかった場合にユーザーに送信されます。

フルフィルメント

インテントのすべてのスロット値がユーザーによって提供されると、Amazon Lex V2 はユーザーのリクエストを処理します。フルフィルメントについては、次のオプションを設定できます。

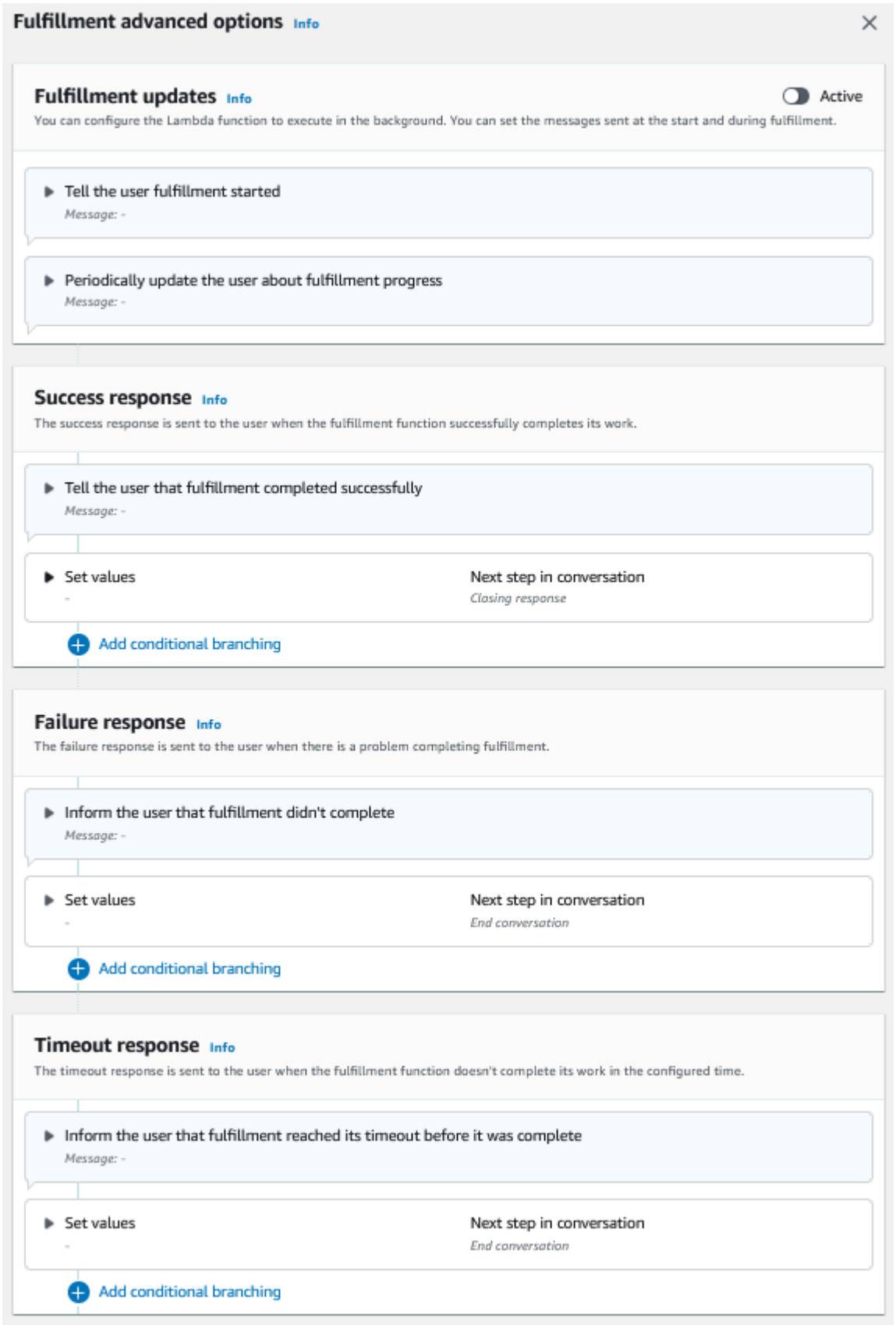
- フルフィルメントコードフック — このオプションを使用して、フルフィルメント Lambda 呼び出しを制御できます。オプションが無効になっている場合、Lambda 関数を呼び出さなくてもフルフィルメントの処理は成功します。
- フルフィルメントアップデート — 完了までに数秒以上かかる Lambda 関数のフルフィルメントアップデートを有効にして、プロセスが進行中であることをユーザーに知らせることができます。詳細については、「Lex V2 ボットのフルフィルメント進行状況の更新の設定」を参照してください。この機能は、ストリーミング会話でのみ使用できます。

- フルフィルメント応答 — 成功の応答、失敗の応答、タイムアウトの応答を設定できます。フルフィルメント Lambda 呼び出しのステータスに基づいて、適切な応答がユーザーに返されます。

フルフィルメント後の応答には、次の 3 種類があります。

- 成功の応答 — Lambda のフルフィルメントが正常に完了したときに送信されるメッセージ。
- 失敗の応答 — フルフィルメントが失敗した場合や、何らかの理由で Lambda を完了できなかった場合に送信されるメッセージ。
- タイムアウト応答 — フルフィルメント Lambda 関数が設定されたタイムアウト内に終了しない場合に送信されるメッセージ。

値を設定し、次のステップを設定し、各応答に対応する条件を適用して会話フローを設計できます。条件がない場合や次のステップが明示的に指定されていない場合、Amazon Lex V2 は終了応答に移動します。


Fulfillment advanced options Info
Fulfillment updates Info
Active
You can configure the Lambda function to execute in the background. You can set the messages sent at the start and during fulfillment.
Tell the user fulfillment started
Message: -
Periodically update the user about fulfillment progress
Message: -
Success response Info
The success response is sent to the user when the fulfillment function successfully completes its work.
Tell the user that fulfillment completed successfully
Message: -
Set values
-
Next step in conversation
Closing response
Add conditional branching
Failure response Info
The failure response is sent to the user when there is a problem completing fulfillment.
Inform the user that fulfillment didn't complete
Message: -
Set values
-
Next step in conversation
End conversation
Add conditional branching
Timeout response Info
The timeout response is sent to the user when the fulfillment function doesn't complete its work in the configured time.
Inform the user that fulfillment reached its timeout before it was complete
Message: -
Set values
-
Next step in conversation
End conversation
Add conditional branching

> **Note**
>
> 2022 年 8 月 17 日、Amazon Lex V2 はユーザーとの会話の管理方法の変更をリリースしました。この変更により、ユーザーが会話の中でたどるパスをより細かく制御できるようになりました。詳細については、「Amazon Lex V2 の会話フローの変更 V2」を参照してください。2022 年 8 月 17 日より前に作成されたボットは、ダイアログコードフックメッセージ、値の設定、次のステップの設定、条件の追加をサポートしていません。

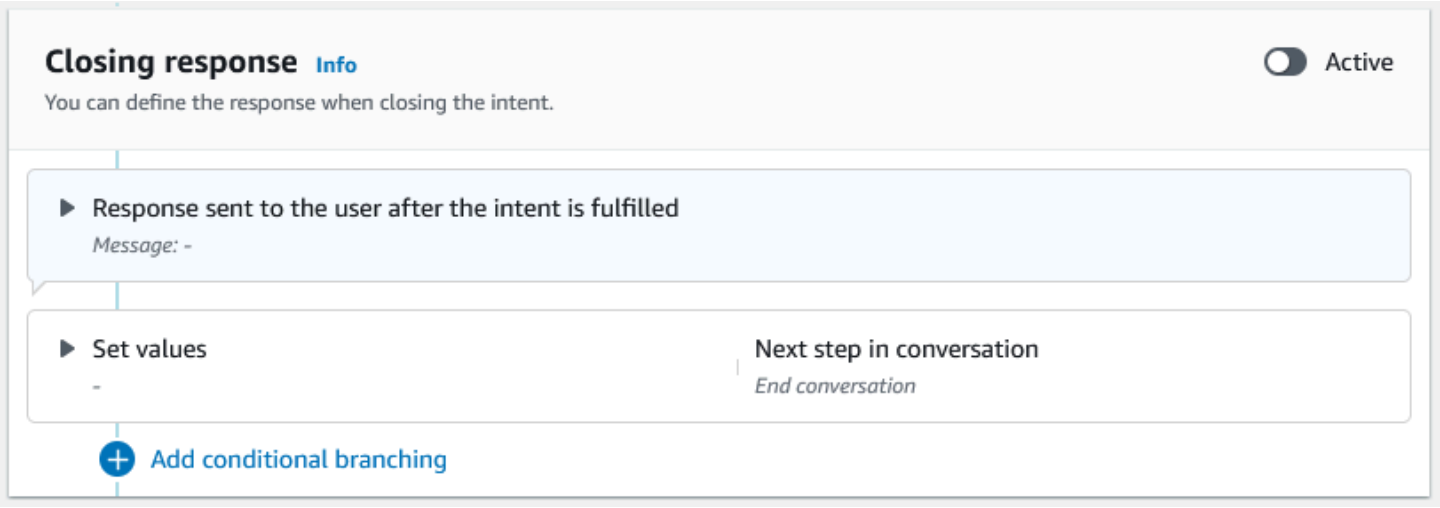
終了応答

終了応答は、ユーザーのインテントが達成した後に送信されます。終了応答を使用して会話を終了したり、別のインテントで続行できることをユーザーに知らせたりできます。たとえば、旅行予約ボットの場合、ホテルの予約インテントの終了答を次のように設定できます。

```
これで、ホテルの部屋が予約されました。他に何か手伝えることはありますか?
```

値を設定し、次のステップを設定し、終了応答後の条件を会話パスの設計に適用できます。条件がない場合や次のステップが明示的に指定されていない場合、Amazon Lex V2 は会話を終了します。

終了応答を提供しない場合、またはどの条件も true と評価されない場合、Amazon Lex V2 はボットとの会話を終了します。

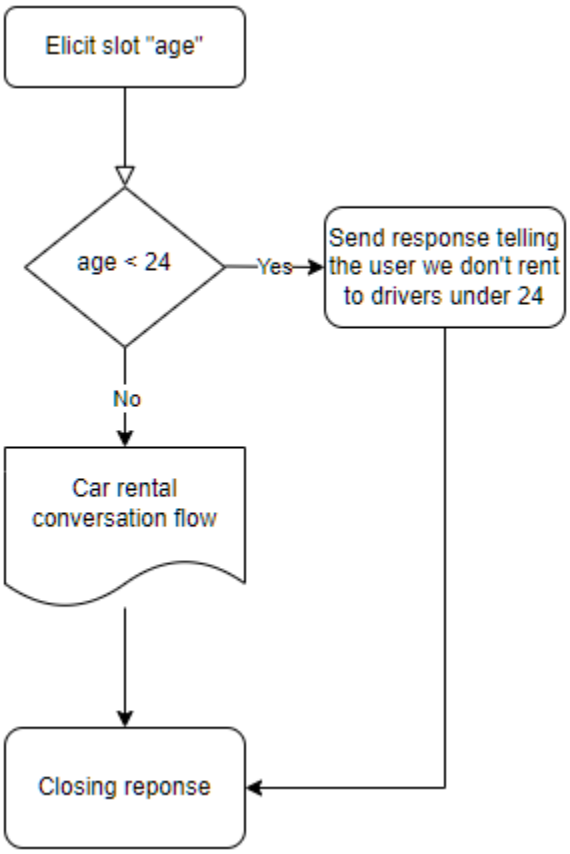


> **Note**
>
> 2022 年 8 月 17 日、Amazon Lex V2 はユーザーとの会話の管理方法の変更をリリースしました。この変更により、ユーザーが会話の中でたどるパスをより細かく制御できるようになりました。詳細については、「Amazon Lex V2 の会話フローの変更 V2」を参照してください。2022 年 8 月 17 日より前に作成されたボットは、ダイアログコードフックメッセージ、値の設定、次のステップの設定、条件の追加をサポートしていません。

会話パスの作成

通常、Amazon Lex V2 は、ユーザーとの会話の流れを管理します。シンプルなボットの場合は、デフォルトのフローで十分なユーザーエクスペリエンスを提供できます。ただし、より複雑なボットの場合は、会話を制御して、フローをより複雑なパスに誘導したい場合があります。

たとえば、レンタカーを予約するボットでは、若いドライバーにはレンタルしないかもしれません。この場合、ドライバーが特定の年齢未満かどうかをチェックする条件を作成し、年齢が満たされていない場合は終了応答にジャンプできます。



このようなやりとりを設計するには、会話の各段階で次のステップを設定し、条件を評価し、値を設定し、コードフックを呼び出します。

条件付き分岐は、複雑なやり取りを通じてユーザーへのパスを作成するのに役立ちます。条件付きブランチはいつでも使用でき、会話の制御をボットに渡すことができます。たとえば、ボットが最初のスロット値を誘発される前に条件を作成したり、各スロット値を誘発するまでの間に条件を作成したり、ボットが会話を終了する前に条件を作成したりできます。条件を追加できる場所のリストについては、「インテントの追加」を参照してください。

ボットを作成すると、Amazon Lex V2 は、スロットの優先順位に基づいて会話のデフォルトパスを作成します。会話のパスをカスタマイズするには、会話のどの時点でも次のステップを変更できます。詳細については、「会話の次のステップを設定します。」を参照してください。

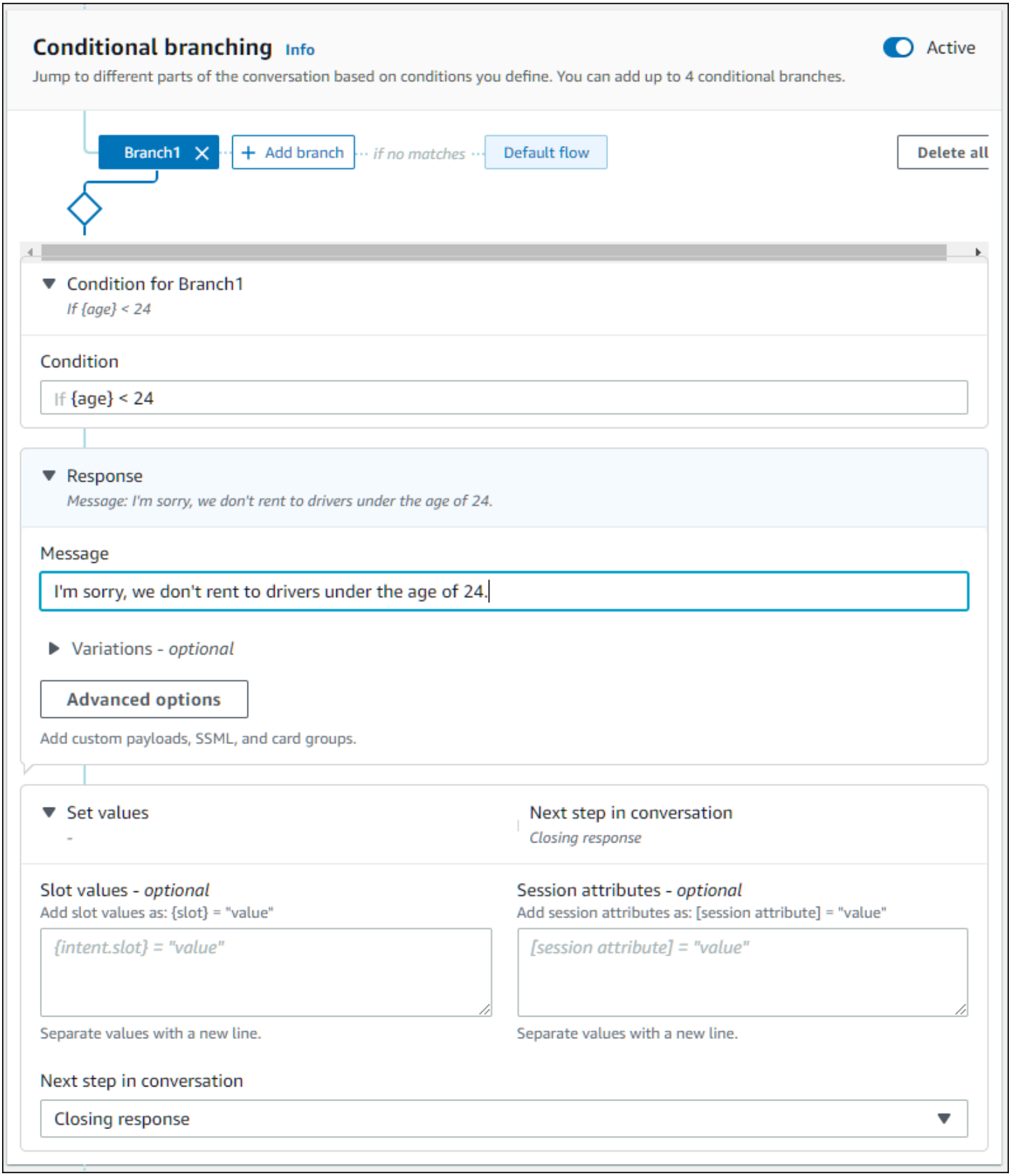
条件に基づいて代替パスを作成するには、会話のどの時点でも条件分岐を使用できます。たとえば、ボットが最初のスロット値を誘発する前に条件を作成できます。各スロット値を誘発するまでの間に条件を作成することも、ボットが会話を終了する前に条件を作成することもできます。条件を追加できる場所のリストについては、「ブランチ会話への条件の追加」を参照してください。

スロット値、セッション属性、入力モードと入力文字起こし、または Amazon Kendra からのレスポンスに基づいて条件を設定できます。

会話中の各時点でのスロット値とセッション属性を設定します。詳細については、「会話中に値を設定する」を参照してください。

次のアクションをダイアログコードフックに設定して Lambda 関数を実行することもできます。詳細については、「ダイアログコードフックを呼び出す」を参照してください。

次の図は、コンソール内のスロットへのパスの作成を示しています。この例では、Amazon Lex V2 はスロット「年齢」を引き出します。スロットの値が 24 未満の場合、Amazon Lex V2 は終了レスポンスにジャンプします。それ以外の場合、Amazon Lex V2 はデフォルトのパスに従います。


Conditional branching Info
Active
Jump to different parts of the conversation based on conditions you define. You can add up to 4 conditional branches.
Branch1
Add branch
if no matches
Default flow
Delete all
Condition for Branch1
If {age} < 24
Condition
If {age} < 24
Response
Message: I'm sorry, we don't rent to drivers under the age of 24.
Message
I'm sorry, we don't rent to drivers under the age of 24.
Variations - optional
Advanced options
Add custom payloads, SSML, and card groups.
Set values
-
Next step in conversation
Closing response
Slot values - optional
Add slot values as: {slot} = "value"
{intent.slot} = "value"
Separate values with a new line.
Session attributes - optional
Add session attributes as: [session attribute] = "value"
[session attribute] = "value"
Separate values with a new line.
Next step in conversation
Closing response

> **Note**
>
> 2022 年 8 月 17 日、Amazon Lex V2 はユーザーとの会話の管理方法の変更をリリースしました。この変更により、ユーザーが会話の中でたどるパスをより細かく制御できるようになりました。詳細については、「Amazon Lex V2 の会話フローの変更 V2」を参照してください。2022 年 8 月 17 日より前に作成されたボットは、ダイアログコードフックメッセージ、値の設定、次のステップの設定、条件の追加をサポートしていません。

会話の次のステップを設定します。

会話の各段階で次のステップを設定して、会話を設計できます。通常、Amazon Lex V2 は、次の順序に従って、会話の各段階のデフォルトの次のステップを自動的に設定します。

初期応答 → スロット誘発 → 確認 (アクティブな場合) → フルフィルメント (アクティブな場合) → 終了応答 (アクティブな場合) → 会話の終了

デフォルトの次のステップを変更し、想定されるユーザーエクスペリエンスに基づいて会話を設計できます。下記の次のステップは、会話の各段階で設定できます。

ジャンプ先

- 初回応答 — 会話はインテントの最初から再開されます。この次のステップを設定するときに、最初の応答をスキップすることもできます。
- スロットを引き出す — インテント内の任意のスロットを誘発することができます。
- 条件の評価 — 会話のどの段階でも条件や分岐会話を評価できます。
- ダイアログコードフックを呼び出す — ビジネスロジックはいつでも呼び出すことができます。
- インテントを確認 — ユーザーはインテントを確認するよう求められます。
- インテントを果たす — 次のステップとしてインテントのフルフィルメントが開始されます。
- 終了応答 — 終了応答がユーザーに返されます。

切り替え先

- インテント — 別のインテントに切り替えて、そのインテントに合わせて会話を続けることができます。移行中は、インテントの初期応答をスキップすることもできます。
- インテント: 特定のスロット — 現在のインテントですでにいくつかのスロット値をキャプチャしている場合は、別のインテントの特定のスロットを直接誘発することができます。

ユーザーの入力を待つ — ボットは、新しいインテントを認識するための入力をユーザーが入力するのを待ちます。次のステップを設定する前に「他に何かお手伝いできることはありますか?」 などのプロンプトを設定できます。ボットは ElicitIntent ダイアログ状態になります。

会話を終了 — ボットとの会話は終了します。

> **Note**
>
> 2022 年 8 月 17 日、Amazon Lex V2 はユーザーとの会話の管理方法の変更をリリースしました。この変更により、ユーザーが会話の中でたどるパスをより細かく制御できるようになりました。詳細については、「Amazon Lex V2 の会話フローの変更 V2」を参照してください。2022 年 8 月 17 日より前に作成されたボットは、ダイアログコードフックメッセージ、値の設定、次のステップの設定、条件の追加をサポートしていません。

会話中に値を設定する

Amazon Lex V2 では、会話のすべてのステップでスロット値とセッション属性値を設定できます。その後、これらの値を会話中に使用して条件を評価したり、インテントフルフィルメントに使用したりできます。

現在のインテントのスロット値を設定できます。会話の次のステップが別のインテントを呼び出すことである場合は、新しいインテントのスロット値を設定できます。

割り当てられたスロットが埋まっていない場合、または JSON パスを解析できない場合、属性は null に設定されます。

スロット値とセッション属性を使用するときは、次の構文を使用してください。

- スロット値 — スロット名を中括弧 (「{}」) で囲みます。現在のインテントのスロット値には、スロット名のみを使用する必要があります。例えば、{slot}。次のインテントで値を設定する場合は、インテント名とスロット名の両方を使用してスロットを識別する必要があります。例えば、{intent.slot}。

  例:

  - {PhoneNumber} = "1234567890"
  - {CheckBalance.AccountNumber} = "99999999"
  - {BookingID} = "ABC123"

- {FirstName} = "John"

スロットの値には次のいずれかを指定できます。

- 定数文字列
- Amazon Lex レスポンス内の文字起こしブロックを参照する JSON パス (en-US および en-GB)
- セッション属性

例:

- {username} = "john.doe"
- {username_confidence} = $.transcriptions[0].transcriptionConfidence
- {username_slot_value} = [username]

> **Note**
>
> スロット値を null に設定することもできます。埋まったスロット値を再誘発する必要がある場合は、顧客にスロット値の入力を再促進する前に、値を null に設定する必要があります。割り当てられたスロットが埋まっていない場合、または JSON パスを解析できない場合、属性は null に設定されます。

- セッション属性 — 属性名を角括弧 ("[ ]") で囲みます。例えば、[sessionAttribute]。

例:

- [username] = "john.doe"
- [username_confidence] = $.transcriptions[0].transcriptionConfidence
- [username_slot_value] = {username}

セッション属性の値は、次のいずれかを指定できます。

- 定数文字列
- Amazon Lex レスポンス内の文字起こしブロックを参照する JSON パス (en-US および en-GB)
- スロット値リファレンス

> **Note**
>
> 割り当てられたスロットが埋まっていない場合、または JSON パスを解析できない場合、属性は null に設定されます。

> **Note**
>
> 2022 年 8 月 17 日、Amazon Lex V2 はユーザーとの会話の管理方法の変更をリリースしました。この変更により、ユーザーが会話の中でたどるパスをより細かく制御できるようになりました。詳細については、「Amazon Lex V2 の会話フローの変更 V2」を参照してください。2022 年 8 月 17 日より前に作成されたボットは、ダイアログコードフックメッセージ、値の設定、次のステップの設定、条件の追加をサポートしていません。

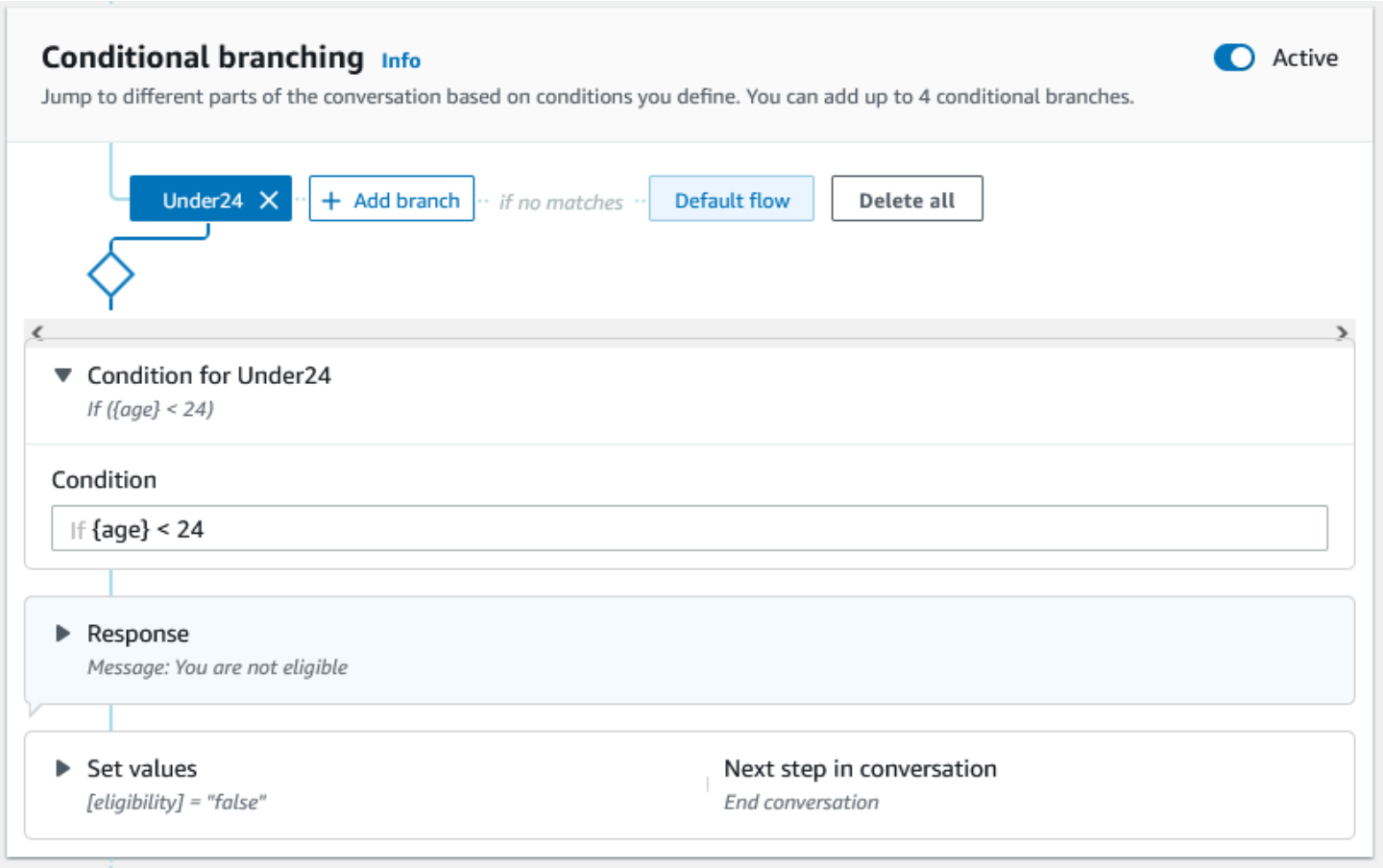
ブランチ会話への条件の追加

条件分岐を使用すると、顧客がボットとの会話を進めるパスを制御できます。スロット値、セッション属性、入力モードの内容、入力文字起こしフィールド、または Amazon Kendra からの応答に基づいて会話を分岐できます。

最大で 4 つのブランチを定義できます。各ブランチには、Amazon Lex V2 がそのブランチに従うために満たさなければならない条件があります。どのブランチも条件が満たされない場合、デフォルトブランチが実行されます。

ブランチを定義するときは、そのブランチに対応する条件が true と評価された場合に Amazon Lex V2 が実行するアクションを定義します。次のいずれかを使用してアクションに定義できます。

- ユーザーに送信された応答。
- スロットに適用するスロット値。
- 現在のセッションのセッション属性値。
- 会話の次のステップ。詳細については、「会話パスの作成」を参照してください。



各条件付きブランチには、Amazon Lex V2 がそのブランチに従うために満たさなければならないブール式があります。条件に使用できる比較演算子、ブール演算子、関数、および量指定演算子があります。たとえば、{age} スロットが 24 未満の場合、次の条件は true を返します。

```
{age} < 24
```

{toppings} 複数値スロットに「パイナップル」という単語が含まれている場合、次の条件は true を返します。

```
{toppings} CONTAINS "pineapple"
```

複数の比較演算子を 1 つのブール演算子と組み合わせると、より複雑な条件を作成できます。たとえば、{make} スロット値が「ホンダ」で {model} スロット値が「シビック」の場合、次の条件は true を返します。括弧を使用して評価順序を設定します。

```
({make} = "Honda") AND ({model} = "Civic")
```

次のトピックでは、条件分岐演算子と関数について詳しく説明します。

> **Note**
>
> 2022 年 8 月 17 日、Amazon Lex V2 はユーザーとの会話の管理方法の変更をリリースしました。この変更により、ユーザーが会話の中でたどるパスをより細かく制御できるようになりました。詳細については、「Amazon Lex V2 の会話フローの変更 V2」を参照してください。2022 年 8 月 17 日より前に作成されたボットは、ダイアログコードフックメッセージ、値の設定、次のステップの設定、条件の追加をサポートしていません。

トピック

- 比較演算子
- ブール演算子
- 数量演算子
- 関数
- サンプル条件式

比較演算子

Amazon Lex V2 では、条件について次の比較演算子がサポートされています。

- 等しい (=)
- 等しくない (!=)
- 未満 (<)
- 以下 (<=)
- Greater than (>)
- 以上 (>=)

比較演算子を使用するときには、以下のルールが使用されます。

- 左側はリファレンスである必要があります。たとえば、スロット値を参照するには、`{slotName}` を使用します。セッション属性値を参照するには、`[attribute]` を使用します。入力モードと入力文字起こしには、`$.inputMode` と `$.inputTranscript` を使用します。
- 右側は定数で、左側と同じ型でなければなりません。

- 設定されていない属性を参照する式はすべて無効として扱われ、評価されません。
- 複数の値を持つスロットを比較する場合、使用される値は、解釈されるすべての値をカンマで区切ったリストです。

比較は参照のスロットタイプに基づいて行われます。それらは以下のように解決されます。

- 文字列 — 文字列は ASCII 表現に基づいて比較されます。比較では、大文字小文字を区別しません。
- 数値 — 数値ベースのスロットは、文字列表現から数値に変換されて比較されます。
- 日付/時刻 — 時間ベースのスロットは時系列に基づいて比較されます。日付または時刻が早いほど小さいとみなされます。期間については、期間が短いほど短いとみなされます。

ブール演算子

Amazon Lex V2 は、比較演算子を組み合わせるブール演算子をサポートしています。以下のようなステートメントを作成できます。

```
({number} >= 5) AND ({number} <= 10)
```

以下のブール演算子を使用することができます。

- AND (&&)
- OR (||)
- NOT (!)

数量演算子

数量演算子はシーケンスの要素を評価し、1 つ以上の要素が条件を満たすかどうかを判断します。

- CONTAINS — 指定された値が複数の値を持つスロットに含まれているかどうかを判断し、含まれている場合は true を返します。たとえば、ユーザーがピザにパイナップルのトッピングを注文した場合、`{toppings} CONTAINS "pineapple"` は true を返します。

関数

関数には文字列 `fn.` をプレフィックスとして付ける必要があります。関数への引数は、スロット、セッション属性、またはリクエスト属性への参照です。Amazon Lex V2 は、スロットの値から情報を取得するための 2 つの関数、sessionAttribute と requestAttribute を提供します。

- fn.Count() — 複数値を持つスロット内の値の数をカウントします。

  たとえば、スロット `{toppings}` に「ペパロニ、パイナップル」という値が含まれている場合:

  ```
  fn.COUNT({toppings}) = 2
  ```

- fn.IS_SET() — 現在のセッションでスロット、セッション属性、またはリクエスト属性が設定されている場合、値は true です。

  前の例に基づく:

  ```
  fn.IS_SET({toppings})
  ```

- fn.LENGTH() – 値は、現在のセッションで設定されたセッション属性、スロット値、またはスロット属性の値の長さです。この関数は、複数値スロットまたは複合スロットをサポートしていません。

  例:

  スロット `{credit-card-number}` に「123456781234」値が含まれている場合:

  ```
  fn.LENGTH({credit-card-number}) = 12
  ```

サンプル条件式

条件式の例をいくつか示します。注: は Amazon Lex V2 JSON レスポンスへのエントリポイント`$.`を表します。次の値は、値を取得するために Amazon Lex V2 レスポンス内で解析`$.`されます。Amazon Lex V2 レスポンスの文字起こしブロックへの JSON パス参照を使用する条件式は、ASR 文字起こしスコアをサポートする同じロケールでのみサポートされます。

値の型	ユースケース	条件式
カスタムスロット	`pizzaSize` スロット値が「L サイズ」と等しい	`{pizzaSize} = "large"`

値の型	ユースケース	条件式
カスタムスロット	pizzaSize が「L サイズ」または「M サイズ」と等しい	{pizzaSize} = "large" または {pizzaSize} = "medium"
カスタムスロット	( ) と AND/OR を使った式	{pizzaType} = "pepperoni" または {pizzaSize} = "medium" または {pizzaSize} = "small"
カスタムスロット (複数値スロット)	トッピングにオニオンが入っているか確認する	{toppings} CONTAINS "Onion"
カスタムスロット (複数値スロット)	トッピングの数が3 種 類以上	fn.COUNT({topping}) > 2
AMAZON.AlphaNumeric	bookingID は ABC123	{bookingID} = "ABC123"
AMAZON.Number	年齢スロット値が 30 を超過	{age} > 30
AMAZON.Number	年齢スロット値が 10 と等しい	{age} = 10
AMAZON.Date	dateOfBirth スロット値が 1990 年より前	{dateOfBirth} < "1990-10-01"
AMAZON.State	destinationState スロット値がワシントンと等しい	{destinationState} = "washington"
AMAZON.Country	destinationCountry スロット値が米国ではない	{destinationCountry} != "united states"
AMAZON.FirstName	firstName スロット値が John	{firstName} = "John"
AMAZON.PhoneNumber	phoneNumber スロット値が 716767891932	{phoneNumer} = 716767891932

値の型	ユースケース	条件式
AMAZON.Percentage	スロットのパーセンテージ値が 78 以上かどうかをチェックしてください。	{percentage} >= 78
AMAZON.EmailAddress	emailAddress スロット値が userA@hmail.com	{emailAddress} = "userA@hmail.com"
AMAZON.LastName	lastName スロット値が Doe	{lastName} = "Doe"
AMAZON.City	市町村スロット値がシアトルと等しい	{city} = "Seattle"
AMAZON.Time	時刻は午後 8 時より後	{time} > "20:00"
AMAZON.StreetName	streetName スロット値がボレン通り	{streetName} = "boren avenue"
AMAZON.Duration	travelDuration スロット値が 2 時間未満	{travelDuration} < P2H
入力モード	入力モードは音声	$.inputMode = "Speech"
文字起こしを入力する	入力文字起こしは「L サイズのピザが欲しい」と等しい	$.inputTranscript = "I want a large pizza"
セッション属性	check customer_subscription_type attribute	[customer_subcription_type] = "yearly"
リクエスト属性	check retry_enabled flag	((retry_enabled)) = "TRUE"
Kendra レスポンス	Kendra レスポンス「よくある質問」が含まれています	fn.IS_SET(((x-amz-lex:kendra-search-response-question_answer-question-1))

値の型	ユースケース	条件式
文字起こしによる条件式	文字起こし (JSON パス) を使用した条件式	`$.transcriptions[0].transcriptionConfidence < 0.8 AND $.transcriptions[1].transcriptionConfidence > 0.5`
セッション属性を設定する	文字起こし JSON パスおよびスロット値を使用してセッション属性を設定する	`[sessionAttribute] = "$.transcriptions...." AND [sessionAttribute] = "{<slotName>}"`
スロット値の設定	セッション属性および文字起こし JSON パスを使用してスロット属性を設定する	`{slotName} = [<sessionAttribute>] AND {slotName} = "$.transcriptions...."`

> **Note**
>
> `slotName` は、Amazon Lex V2 ボット内のスロットの名前を指します。スロットが解決されない場合 (null)、またはスロットが存在しない場合、割り当ては実行時に無視されます。`sessionAttribute` は、構築時に顧客が設定したセッション属性の名前を指します。

ダイアログコードフックを呼び出す

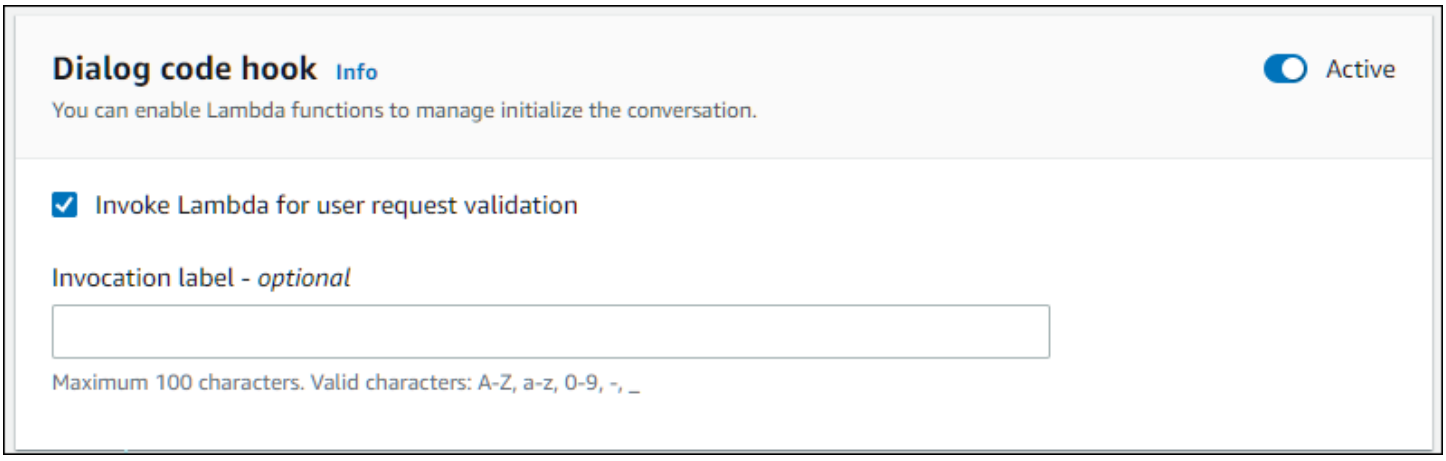
Amazon Lex V2 がユーザーにメッセージを送信するときの会話の各ステップで、会話の次のステップとして Lambda 関数を使用できます。この関数を使用して、会話の現在の状態に基づいてビジネスロジックを実装できます。

実行される Lambda 関数は、使用しているボットエイリアスに関連付けられています。インテント内のすべてのダイアログコードフックで Lambda 関数を呼び出すには、インテントの初期化と検証

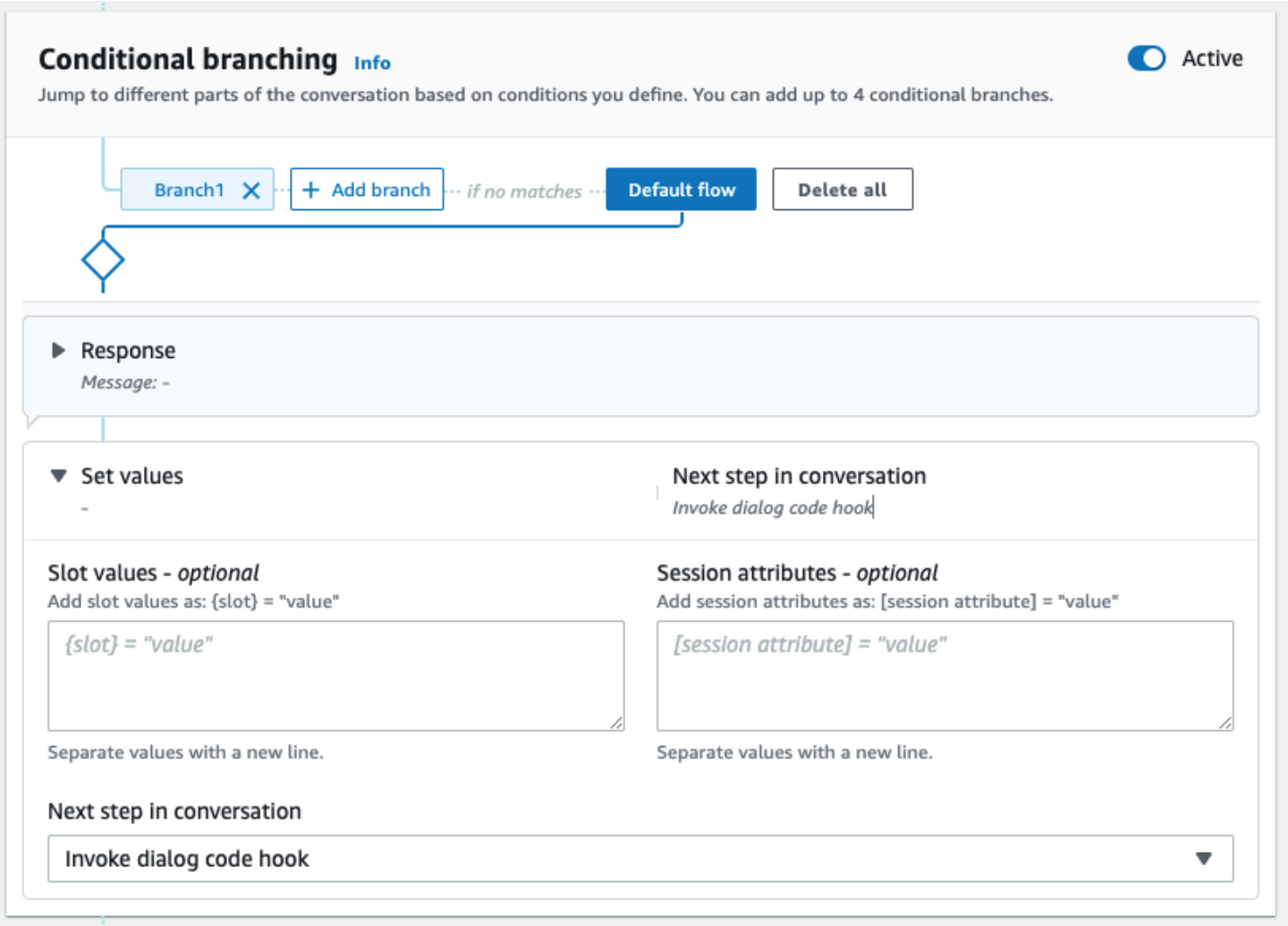
に [Lambda 関数を使用する] を選択する必要があります。Lambda 関数の選択の詳細については、「Amazon Lex V2 ボットの AWS Lambda関数の作成」を参照してください。

Lambda 関数を使用するには 2 つのステップがあります。まず、会話のどの時点でもダイアログコードフックを有効にする必要があります。次に、会話の次のステップでダイアログコードフックを使用するように設定する必要があります。

次の図は、ダイアログコードフックがアクティブになっていることを示しています。

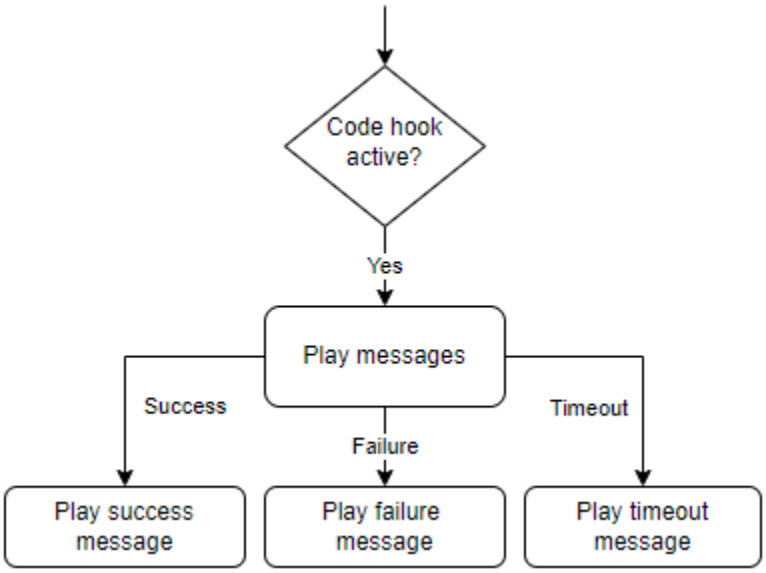


次に、コードフックを会話ステップの次のアクションとして設定します。そのためには、会話の次のステップを「ダイアログコードフックを呼び出す」に設定してください。以下の画像は、会話のデフォルトパスの次のステップとしてダイアログコードフックを呼び出す条件分岐を示しています。



コードフックがアクティブになると、ユーザーに返すレスポンスを 3 つ設定できます。

- 成功 — Lambda 関数が正常に完了したときに送信されます。
- 失敗 — Lambda 関数の実行で問題が発生した場合、または Lambda 関数が `Failed` の `intent.state` 値を返した場合に送信されます。
- タイムアウト — Lambda 関数が設定されたタイムアウト期間内に完了しなかった場合に送信されます。



[Lambda ダイアログコードフック] を選択し、[詳細オプション] を選択すると、Lambda 関数の呼び出しに対応するレスポンスの 3 つのオプションが表示されます。値を設定し、次のステップを設定し、各レスポンスに対応する条件を適用して会話フローを設計できます。条件がない場合や次のステップが明記されていない場合、Amazon Lex V2 は会話の現在の状態に基づいて次のステップを決定します。

[詳細オプション] ページでは、Lambda 関数の呼び出しを有効または無効にすることもできます。関数が有効になると、Lambda 呼び出しでダイアログコードフックが呼び出され、続いて Lambda 呼び出しの結果に基づいて成功、失敗、またはタイムアウトのメッセージが続きます。この関数を無効にすると、Amazon Lex V2 は Lambda 関数を実行せず、ダイアログコードフックが成功したかのように処理を進めます。

Lambda 関数をこのメッセージで呼び出したときにその関数に渡す呼び出しラベルを設定することもできます。これを使用すると、実行する Lambda 関数のセクションを特定しやすくなります。

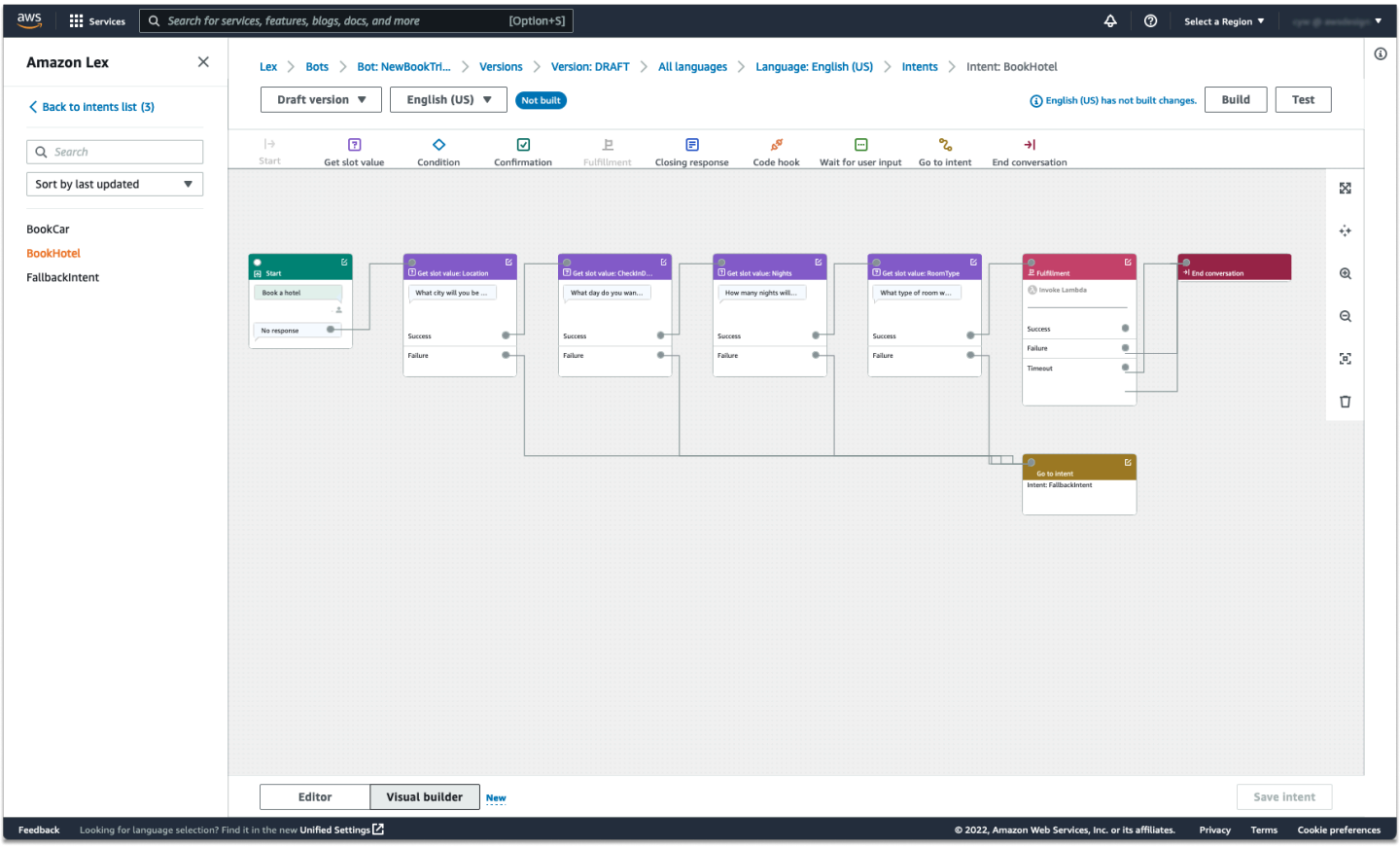
> **Note**
>
> 2022 年 8 月 17 日、Amazon Lex V2 はユーザーとの会話の管理方法の変更をリリースしました。この変更により、ユーザーが会話の中でたどるパスをより細かく制御できるようになりました。詳細については、「Amazon Lex V2 の会話フローの変更 V2」を参照してください。2022 年 8 月 17 日より前に作成されたボットは、ダイアログコードフックメッセージ、値の設定、次のステップの設定、条件の追加をサポートしていません。

ビジュアル会話ビルダーの使用

ビジュアル会話ビルダーは、豊富なビジュアル環境内でインテントを使用して会話パスを簡単に設計および視覚化できる、ドラッグアンドドロップの会話ビルダーです。

ビジュアル会話ビルダーにアクセスするには

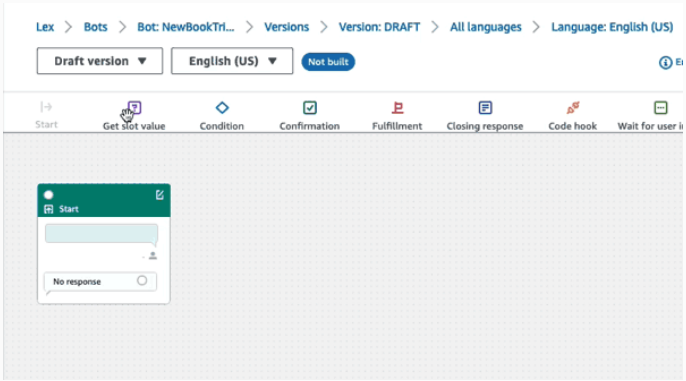
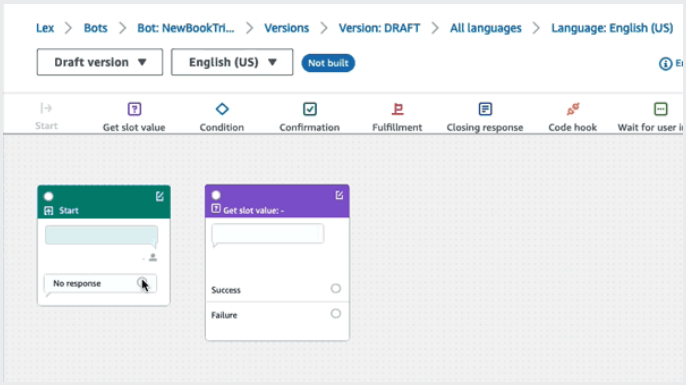
1. Amazon Lex V2 コンソールで、ボットを選択し、左側のナビゲーションペインから [インテント] を選択します。
2. 以下のいずれかの方法で、インテントエディタを開きます。
   - [インテント] セクションの右上隅にある [インテントを追加] を選択し、空のインテントを追加するか、ビルトインインテントを追加するかを選択します。
   - [インテント] セクションからインテントの名前を選択します。
3. インテントエディターの画面下部のペインで [ビジュアルビルダー] を選択し、ビジュアル会話ビルダーにアクセスします。
4. メニューインテントエディターのインターフェースに戻るには、[エディター] を選択します。

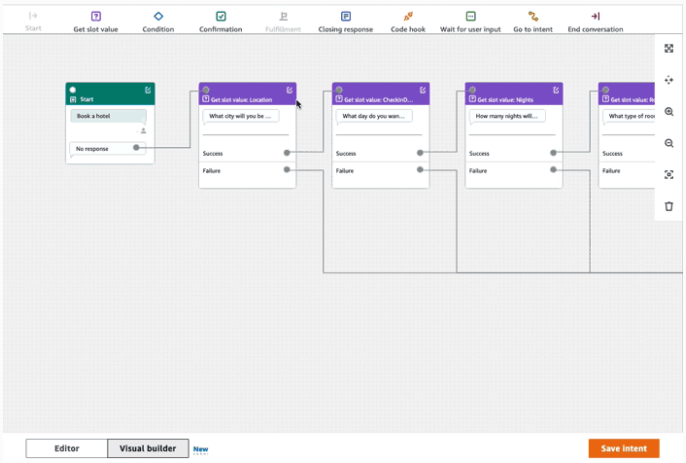
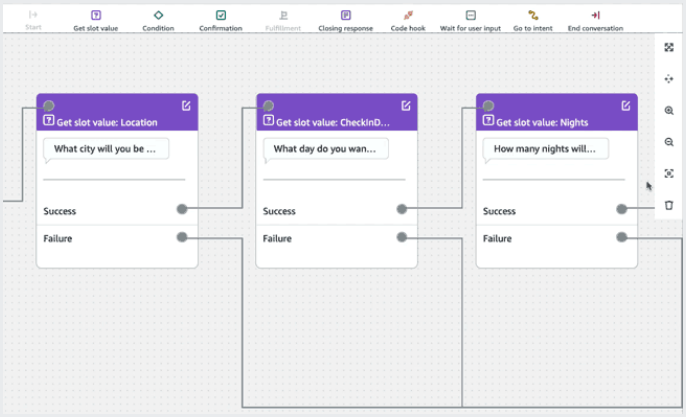
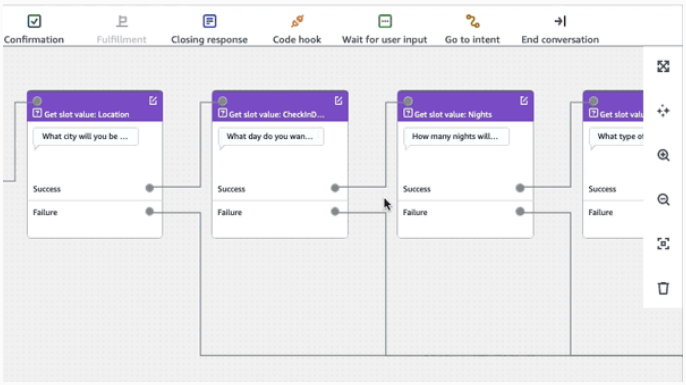


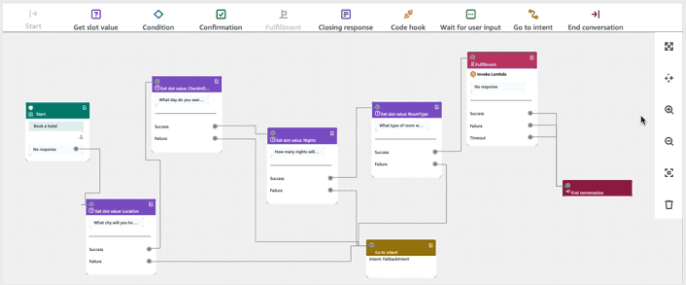
ビジュアル会話ビルダーは、会話フローを視覚化して変更できる、より直感的なユーザーインターフェースを備えています。ブロックをドラッグアンドドロップすることで、既存のフローを拡張したり、会話のステップを並べ替えたりできます。Lambda コードを一切記述しなくても、複雑な分岐を伴う会話フローを開発できます。

この変更は、会話フロー設計を Lambda の他のビジネスロジックから切り離すのに役立ちます。ビジュアル会話ビルダーは既存のインテントエディターと組み合わせて使用でき、会話フローの構築にも使用できます。ただし、より複雑な会話フローにはビジュアルエディタービューを使用することをおすすめします。

インテントを保存すると、Amazon Lex V2 は接続に失敗したと判断したときにインテントを自動接続したり、Amazon Lex V2 が接続を提案したり、ブロック用に独自の接続を選択したりできます。

アクション	例
ブロックをワークスペースに追加する	
ブロック間の接続	

アクション	例
ブロックで設定パネルを開く	
大きさを合わせてズーム	
会話フローからブロックを削除する	

アクション	例
ワークスペースを自動クリーンアップする	

用語:

ブロック — 会話フローの基本構成単位。各ブロックには、会話のさまざまなユースケースに対応するための特定の機能があります。

ポート — 各ブロックには、あるブロックを別のブロックに接続するために使用できるポートが含まれています。ブロックには入力ポートと出力ポートを含めることができます。各出力端子は、ブロックの特定の機能バリエーション (エラー、タイムアウト、成功など) を表します。

エッジ — エッジは、あるブロックの出力ポートと別のブロックの入力ポートの間の接続です。会話フローの分岐の一部です。

会話フロー — 顧客とのインテントレベルのやりとりを説明する、エッジでつながった一連のブロック。

ブロック

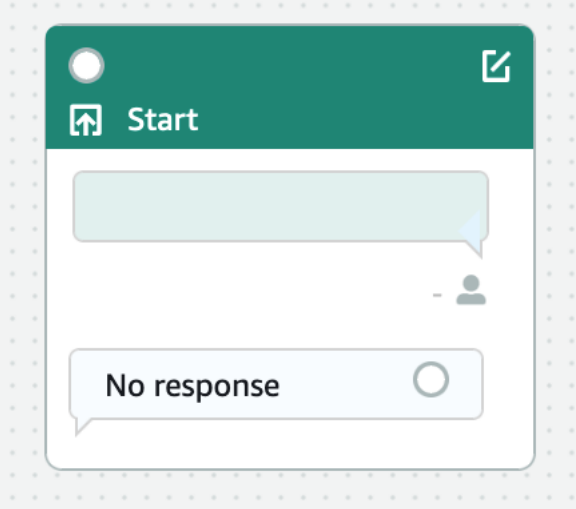
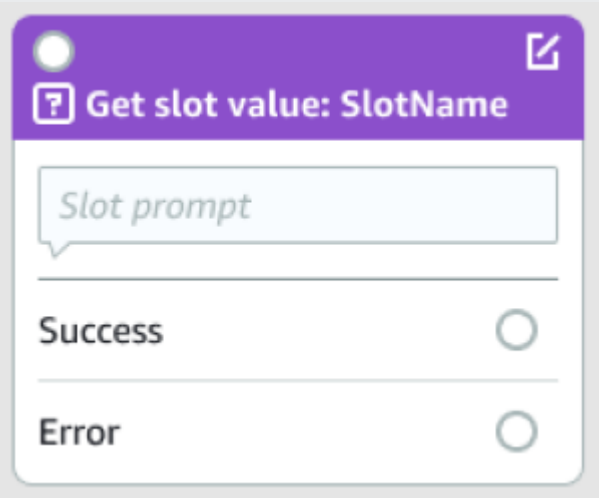
ブロックは、会話フロー設計の構成要素です。インテントの開始からユーザー入力、終了までのインテント内のさまざまな状態を表します。

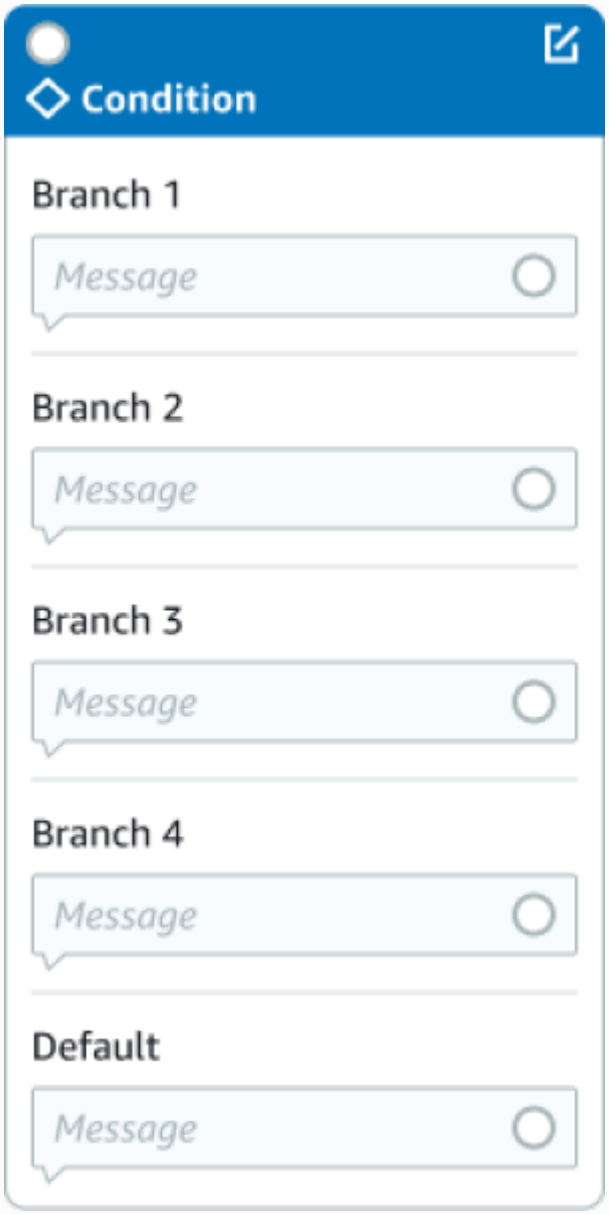
各ブロックには、ブロックタイプに応じて 1 つのエントリポイントと 1 つまたは複数の終了ポイントがあります。各終了ポイントには、会話が終了ポイントを通過するにつれて、対応するメッセージを設定できます。複数の終了ポイントあるブロックの場合、終了ポイントはノードに対応するステータスに関係します。条件ノードの場合、終了ポイントは異なる条件を表します。

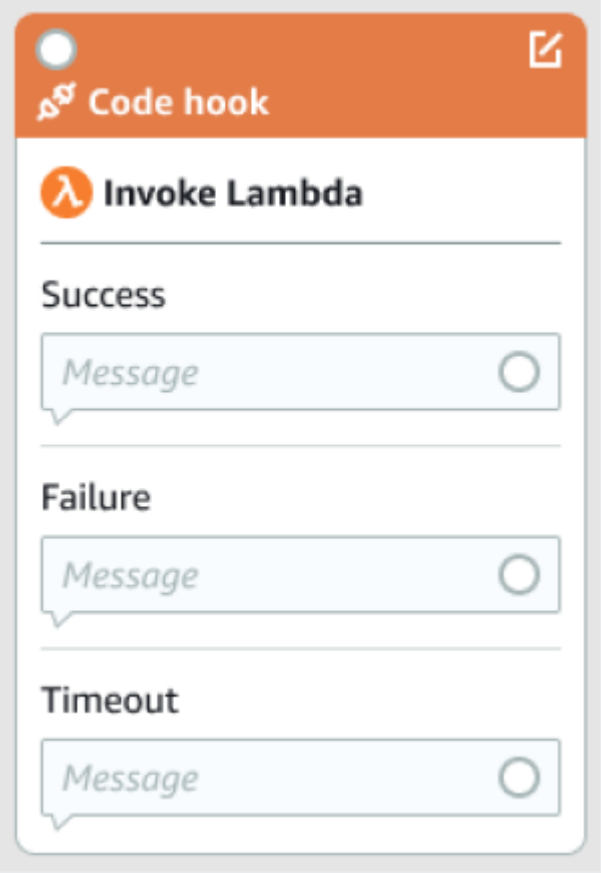
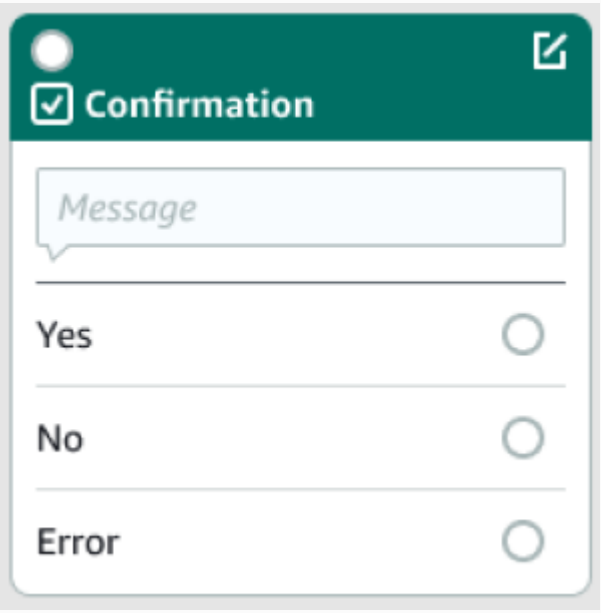
各ブロックには設定パネルがあり、ブロックの右上隅にある [編集] アイコンをクリックすると開きます。設定パネルには、各ブロックに対応するように設定できる詳細フィールドがあります。

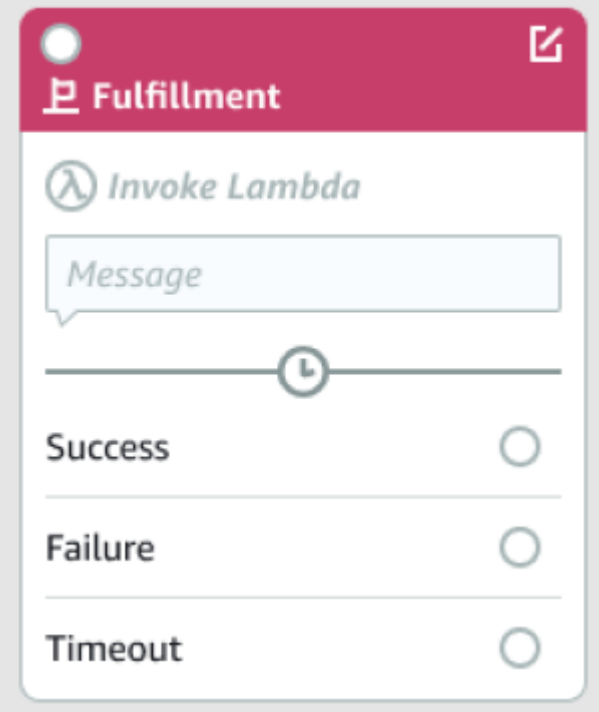
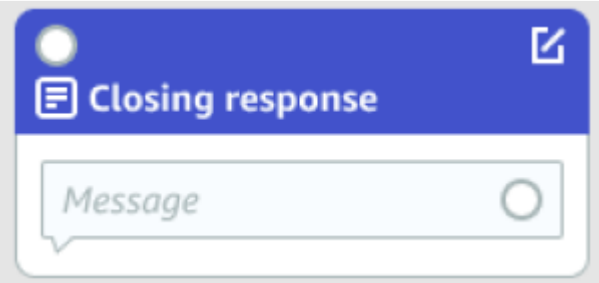
ボットのプロンプトとメッセージは、新しいブロックをドラッグしてノード上で直接設定することも、ブロックの他の属性とともに右側のパネル内で変更することもできます。

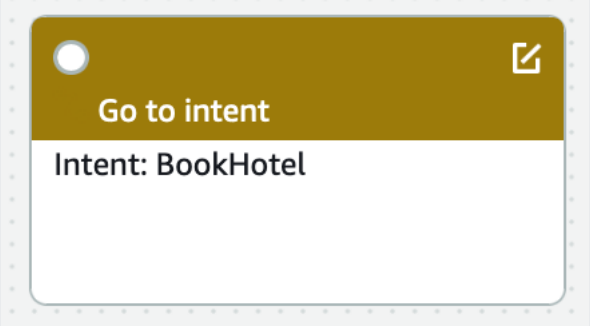
ブロックタイプ — ビジュアル会話ビルダーで使用できるブロックタイプは次のとおりです。

ブロックタイプ	ブロック
開始 — 会話フローのルートまたは最初のブロックです。このブロックは、ボットが初期応答 (意図を認識したというメッセージ) を送信できるように設定することもできます。詳細については、「初期応答」を参照してください。	
スロット値取得 — このブロックは 1 つのスロットの価値を誘発しようとします。このブロックには、スロット誘発プロンプトに対する顧客の応答を待つ設定があります。詳細については、「Slots」を参照してください。	

ブロックタイプ	ブロック
条件 — このブロックには条件文が含まれます。最大 4 つのカスタムブランチ (条件付き) と 1 つのデフォルトブランチが含まれます。詳細については、「ブランチ会話への条件の追加」を参照してください。	

ブロックタイプ	ブロック
ダイアログコードフック — このブロックはダイアログ Lambda 関数の呼び出しを処理します。このブロックには、ダイアログの Lambda 関数の成功、失敗、またはタイムアウトに基づくボット応答が含まれます。詳細については、「ダイアログコードフックを呼び出す」を参照してください。	
確認 — このブロックは、インテントがフルフィルメントされる前に顧客にクエリを実行します。確認プロンプトに対して顧客が「はい」または「いいえ」と言ったことに基づいてボット応答が含まれます。詳細については、「確認」を参照してください。	

ブロックタイプ	ブロック
フルフィルメント — このブロックは、通常はスロット誘発後のインテントのフルフィルメントを処理します。フルフィルメントが成功または失敗した場合に、Lambda 関数を呼び出したり、メッセージで応答したりするように設定できます。詳細については、「フルフィルメント」を参照してください。	
応答を閉じる — このブロックにより、ボットは会話を終了する前にメッセージを返信できます。詳細については、「終了応答」を参照してください。	
会話を終了する — このブロックは会話フローの終了を示します。	
ユーザーの入力を待つ — このブロックを使用すると、顧客からの入力を取り込み、その発話に基づいて別のインテントに切り替えることができます。	

ブロックタイプ	ブロック
インテントに移動 — このブロックを使用して、新しいインテントに移動したり、そのインテントの特定のスロットを直接誘発したりできます。	

ポートタイプ

すべてのブロックには、親ブロックを接続するための入力ポートが 1 つあります。会話は、親ブロックの出力ポートから特定のブロックの入力ポートにのみ流すことができます。ただし、ブロックに含めることができるのは、0、1、または多数の出力ポートです。出力ポートのないブロックは、現在のインテント (GoToIntent、EndConversation、WaitForUserInput) における会話フローの終了を示します。

インテントデザインのルール:

- インテント内のすべてのフローは開始ブロックから始まります。
- 各終了ポイントに対応するメッセージは省略可能です。
- 設定パネルで各終了ポイントに対応する値を設定するようにブロックを設定できます。
- インテント内の 1 つのフローに存在できるのは、開始、確認、フルフィルメント、終了の各ブロック 1 つのみです。複数の条件、ダイアログコードフック、スロット値の取得、会話の終了、転送、ユーザー入力待ちの各ブロックが存在する可能性があります。
- 条件ブロックを条件ブロックに直接接続することはできません。同じことがダイアログコードフックにも当てはまります。
- 循環フローは 3 ブロック許可されていますが、Start Intent への受信コネクタは許可されていません。
- オプションスロットには受信コネクタも送信接続もなく、主にインテントの誘発中に存在するデータをキャプチャするために使用されます。会話パスの一部であるその他すべてのスロットは必須スロットでなければなりません。

ブロック:

- 開始ブロックには出力エッジが必要です。
- スロットが必要な場合は、すべての get slot value ブロックに成功ポートからの出力エッジが必要です。
- ブロックがアクティブであれば、すべての条件ブロックには各ブランチからの出力エッジが必要です。
- 1 つの条件ブロックに複数の親を設定することはできません。
- アクティブな条件ブロックには入力エッジが必要です。
- すべてのアクティブなコードフックブロックには、成功、失敗、タイムアウトの各ポートからの出力エッジが必要です。
- アクティブなコードフックブロックには入力エッジが必要です。
- アクティブな確認ブロックには入力エッジが必要です。
- アクティブなフルフィルメントブロックには入力エッジが必要です。
- アクティブな終了ブロックには入力エッジが必要です。
- 条件ブロックには、デフォルト以外のブランチを 1 つ以上含める必要があります。
- インテントに移動ブロックにはインテントが指定されている必要があります。

エッジ:

- 条件ブロックを別の条件ブロックに接続することはできません。
- コードフックブロックを別のコードフックブロックに接続することはできません。
- 条件ブロックは、0 個または 1 個のコードフックブロックにのみ接続できます。
- 接続 (コードフック-> 条件-> コードフック) は無効です。
- フルフィルメントブロックが子としてコードフックブロックを持つことはできません。
- フルフィルメントブロックの子である条件ブロックは、コードフックブロックの子を持つことはできません。
- 終了ブロックが子としてコードフックブロックを持つことはできません。
- 終了ブロックの子である条件ブロックは、コードフックブロックの子を持つことはできません。
- 開始ブロック、確認ブロック、またはスロット値取得ブロックでは、依存関係チェーンに複数のコードフックブロックを含めることはできません。

> **Note**
>
> 2022 年 8 月 17 日、Amazon Lex V2 はユーザーとの会話の管理方法の変更をリリースしました。この変更により、ユーザーが会話の中でたどるパスをより細かく制御できるようになりました。詳細については、「Amazon Lex V2 の会話フローの変更 V2」を参照してください。2022 年 8 月 17 日より前に作成されたボットは、ダイアログコードフックメッセージ、値の設定、次のステップの設定、条件の追加をサポートしていません。

組み込みのインテント

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 マネジメントコンソールにサインインし、https://console.aws.amazon.com/lex/ で Amazon Lex コンソールを開きます。
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. 左側のメニューで、言語を選択してから、[インテント] を選択します。
4. [インテントの追加] を選択してから、[組み込みインテントを使用する] を選択します。
5. [組み込みインテント] で、使用するインテントを選択します。
6. インテントに名前を付けて、[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.BedrockAgentIntent
- AMAZON.CancelIntent
- AMAZON.FallbackIntent

- AMAZON.HelpIntent
- AMAZON.KendraSearchIntent
- AMAZON.PauseIntent
- AMAZON.QnAIntent
- AMAZON.QnAIntent (multiple use support)
- AMAZON.QinConnectIntent
- AMAZON.RepeatIntent
- AMAZON.ResumeIntent
- AMAZON.StartOverIntent
- AMAZON.StopIntent

AMAZON.BedrockAgentIntent

> **Note**
>
> 生成 AI 機能を活用するには、最初に以下の前提条件を満たす必要があります。
>
> 1. Amazon Bedrock の使用料金については、「Amazon Bedrock の料金」を参照してください。
> 2. ボットロケールで生成 AI 機能を有効にします。これを行うには、「生成 AI を使用した Lex V2 ボットの作成とパフォーマンスの最適化」の手順に従います。

顧客のリクエストに応答し、エージェントワークフローをアクティブ化して定義されたタスクを達成するインテントで定義された Amazon Bedrock エージェントをアクティブ化します。この機能は、Amazon Lex V2 がサポートするすべてのロケールと Amazon Lex V2 と Amazon Bedrock エージェントの両方が存在するすべての商用リージョンで使用できます。

このインテントが `FallbackIntent` を上書きしている場合、インテントは、発話がボットに存在する他のインテントのいずれにも分類されていない場合にアクティブ化されます。それ以外の場合は、発話がこのインテントに分類されたときにのみアクティブ化されます。スロット値を引き出す際に発話見逃しがあった場合、このインテントは有効にならないので注意してください。

Amazon Lex V2 ボットによって認識されると、`AMAZON.BedrockAgentIntent` は定義された `BedrockAgent` または `BedrockKnowledgeBase` をアクティブ化して顧客に応答します。Amazon Bedrock エージェントを使用している場合、会話は `BedrockAgentIntent` にと

どまり、Amazon Bedrock エージェントが会話が FINISH とマークされていると判断するまで、ユーザーリクエストはエージェントに中継されます。その後、Amazon Lex V2 は会話の制御を担い、AMAZON.BedrockAgentIntent で定義されている次のステップに従います。

Amazon Bedrock エージェントおよびナレッジベースを使用して顧客の質問に回答し、詳細なレスポンスを提供します。

> **Warning**
>
> サンプル発話のない AMAZON.BedrockAgentIntent、サンプル発話のない AMAZON.QnAIntent、および AMAZON.KendraSearchIntent を同じボットロケールで使用することはできません。

このインテントを選択した場合は、以下のフィールドを設定し、[追加] を選択してインテントを追加します。

- Amazon Bedrock エージェント ID - Amazon Bedrock エージェントの識別子。使用する Bedrock エージェントを選択します。
- Amazon Bedrock エージェントエイリアス ID - Amazon Bedrock エージェントのエイリアス識別子。

> **Important**
>
> Amazon Lex V2 で使用する Amazon Bedrock エージェントを作成するときは、[追加設定] の [ユーザー入力] が ENABLED であることを確認します。この設定は、エージェントが明確化またはフォローアップの質問をできるようにするために重要です。これにより、Amazon Lex V2 は対応するタスクを完了するためにエージェントに委任し直すことができます。

(オプション) 以下のオプションで BedrockAgentIntent を追加することもできます。

- Amazon Bedrock モデル — このインテントに使用するプロバイダーと基盤モデルを選択します。現在、一部の Anthropic Claude モデルがサポートされています。
- Amazon Bedrock ナレッジベース — このオプションを選択する場合は、Amazon Bedrock ナレッジベースの ID を指定します。ID は、コンソールの Amazon Bedrock ナレッジベースの詳細ページで確認できます。GetKnowledgeBase リクエストを送信して確認することもできます。

BedrockAgentIntent からのレスポンスは、次に示すようにセッションおよびリクエスト属性に保存されます。

- `x-amz-lex:bedrock-agent-search-response` – Amazon Bedrock エージェントから質問または発話へのレスポンス。
- `x-amz-lex:bedrock-knowledge-base-search-response-source` – Amazon Bedrock ナレッジベース設定を使用している場合に、レスポンスを生成するために使用されるドキュメントまたはドキュメントのリストを指します。
- `x-amz-lex:bedrock-agent-action-group-invocation-input` - エージェントアクショングループによって収集された入力値を含むオブジェクト。エージェントアクショングループの詳細については、「ActionGroupInvocationInput」を参照してください。
- `x-amz-lex:bedrock-agent-knowledge-base-lookup-input` - Amazon Bedrock ナレッジベースのルックアップに関連する詳細を含むオブジェクト。
- `x-amz-lex:bedrock-agent-agent-collaborator-details` – マルチエージェントコラボレーション呼び出しの一部として呼び出されたサブエージェントからの入出力の詳細を含むオブジェクト。

詳細については、BedrockAgentIntent を使用した Amazon Lex での Bedrock エージェントの使用」を参照してください。

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

ユーザーのインテントへの入力がボットの想定と異なる場合、Amazon Lex V2 を設定して [フォールバックインテント] を呼び出すことができます。例えば、ユーザー入力「キャンディーを注文したい

です」が OrderFlowers ボットのインテントと一致しない場合、Amazon Lex V2 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。

コンソールを使用してボットを作成するか、CreateBotLocale オペレーションを使用してボットにロケールを追加すると、組み込み AMAZON.FallbackIntent インテントタイプが自動的にボットに追加されます。

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。

Amazon Lex V2 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 V2 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。

- 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に以下を追加することはできません。

- 発話
- Slots
- 確認プロンプト

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 CreateIntent オペレーションに対する fulfillmentCodeHook 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。

- エイリアスの検証とフルフィルメント 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、CreateIntent オペレーションの `fulfillmentCodeHook` 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 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 V2 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、「Lex V2 ボットの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
- 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 V2 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)ロケール、および米国東部 (バージニア北部)、米国西部 (オレゴン)、欧州 (アイルランド) のリージョンでのみ使用できます。

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 V2 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。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。

> **Note**
>
> Amazon Lex V2 は現在、スロット誘発中の AMAZON.KendraSearchIntent をサポートしていません。Amazon Lex V2 がスロットのユーザー発話を判別できない場合、AMAZON.FallbackIntent を呼び出します。

同じボットで AMAZON.KendraSearchIntent と AMAZON.FallbackIntent の両方を使用する場合、Amazon Lex V2 は以下のようにインテントを使用します。

1. Amazon Lex V2 が 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 V2 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 V2 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 V2 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 V2 が認識する発話が含まれていない場合、Amazon Lex V2 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 V2 は `AMAZON.FallbackIntent` を呼び出し、ユーザーとの会話を終了します。

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

- 検索インテントにリクエストを作成させます。Amazon Lex V2 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ー文字列を定義できます。Amazon Lex V2 は、クエリリクエストでフィルターを使用します。
- Lambda 関数を使用して検索結果を絞り込むために、リクエストにクエリパラメータを追加します。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。
- Lambda 関数を使用して、新しいクエリを作成します。Amazon Lex V2 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。属性フィルターの詳細については、[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クエリをフィルタリングする] を参照してください。

Amazon Lex V2 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 V2 は `InvalidLambdaResponseException` 例外を返します。詳細については、[Amazon Kendra デベロッパーガイド] の [クエリ操作] 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、Amazon Lex V2 が Amazon Kendra インQuery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引き受けることができる AWS Identity and Access Management (IAM) ポリシーを提供するロールを使用する必要があります。V2 使用する IAM 設定は、Amazon Lex V2 コンソールAMAZON.KendraSearchIntentを使用するか、AWS SDK または AWS Command Line Interface (AWS CLI) を使用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 V2 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場合は、 Queryオペレーションの呼び出し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 V2 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 V2 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。

JSON

```
{
"Version":"2012-10-17",
"Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "iam:AttachRolePolicy",
            "iam:PutRolePolicy",
            "iam:GetRolePolicy"
        ],
        "Resource": "arn:aws:iam::*:role/aws-service-role/lexv2.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 Identity and Access Management ユーザーガイド AWS STS の「IAM および条件コンテキストキー」を参照してください。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::111122223333:role/role"
        }
    ]
}
```

Amazon Lex V2 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。
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 V2 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 マネジメントコンソールにサインインし、https://console.aws.amazon.com/lex/ で Amazon Lex コンソールを開きます。
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 V2 サービスにリンクされたロールを更新するには、[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、[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、「Lex V2 ボットの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、「Lex V2 ボットの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、文字列を使用して 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 IntentClosingSetting ステートメントから、検索に対するレスポンスを返します。Lambda 関数が終了応答メッセージを生成しない限り、インテントには closingResponse ステートメントが必要です。

Amazon Kendra には 5 タイプのレスポンスがあります。

- 次の 2 つの応答では、Amazon Kendra インデックスに関するよくある質問を設定する必要があります。詳細については、「質問と回答をインデックスに直接追加する」を参照してください。
  - 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。

- 次の 3 つの応答では、Amazon Kendra インデックスに対してデータソースを設定する必要があります。詳細については、「データソースの作成」を参照してください。
  - 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。

closingResponse 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 V2 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 デベロッパーガイド」の「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 V2 にレスポンスを送信します。Amazon Lex V2 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

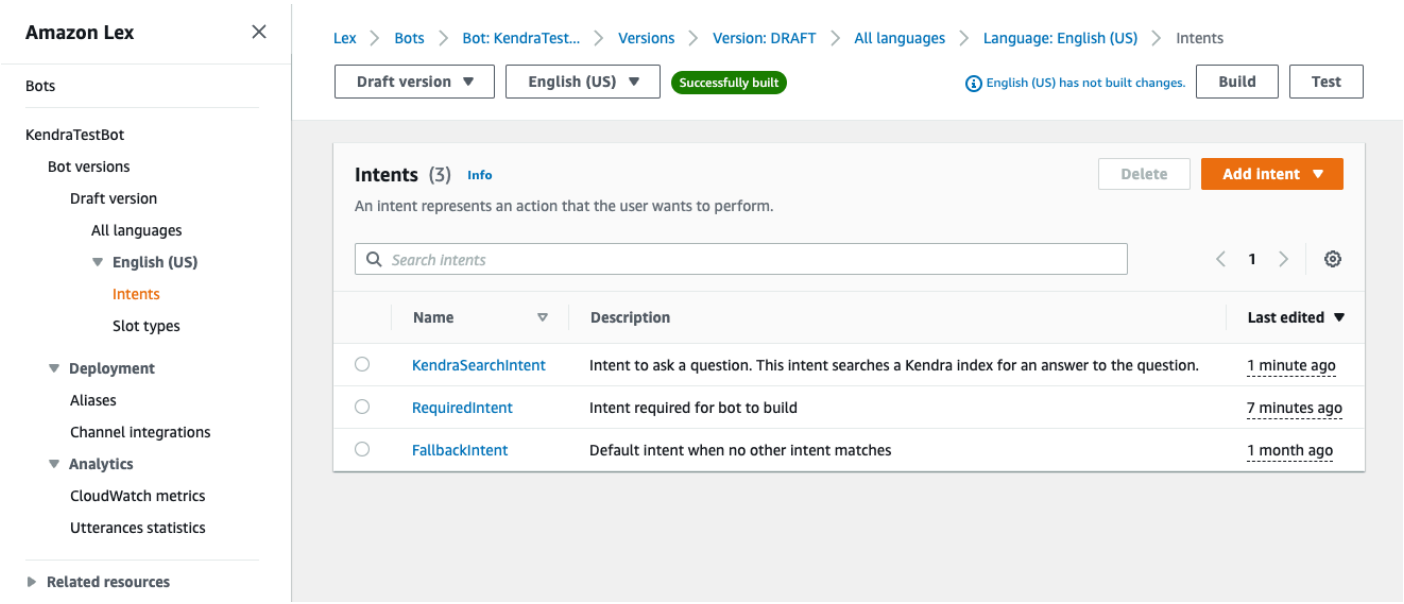
Amazon Lex V2 がクエリを Amazon Kendra に送信した後、クエリレスポンスが `AMAZON.KendraSearchIntent` フルフィルメント Lambda 関数に返されます。コードフックへの入力イベントには、Amazon Kendra からの完全なレスポンスが含まれます。クエリデータは、Amazon Kendra `Query` オペレーションによって返されるものと同じ構造になります。詳細については、[Amazon Kendra デベロッパーガイド] の [クエリレスポンス構文] を参照してください。

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 V2 はレスポンスに `closingResponse` 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 V2 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。Amazon Kendra インデックスを使用して FAQ ボットを作成する方法の概要は次のとおりです。

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。`AMAZON.KendraSearchIntent` および `AMAZON.FallbackIntent` とはバックアップインテントなので、ボットには少なくとも 1 つの発話を含むインテントがもう 1 つ必要です。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。そのため、以下の画像のように、FAQ ボットには少なくとも 3 つのインテントが含まれます。



3. ボットに `AMAZON.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. クエリを実行し、Amazon Kendra インデックスの結果がクエリに答えるドキュメントであることを確認して、ボットをテストします。

前提条件

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 デベロッパーガイド」の「Amazon Kendra コンソールの使用を開始する」を

参照してください。この例では、データソースとしてサンプルデータセット (サンプル AWS ドキュメント) を選択します。

FAQ ボットを作成するには:

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/lex/ で Amazon Lex コンソールを開きます。
2. ナビゲーションペインで、[ボット] を選択します。
3. [ボットの作成] を選択します。
   a. 作成方法について、「空白のボットを作成」を選択します。
   b. [ボット設定] セクションで、ボットに **KendraTestBot** など、目的を示す名前とオプションの説明を与えます。新しい名前は アカウント内で一意である必要があります。
   c. [IAM アクセス許可] セクションで、[基本的な Amazon Lex アクセス許可でロールを作成する] を選択します。これにより、Amazon Lex V2 がボットを実行するために必要なアクセス許可を持つ AWS Identity and Access Management (IAM) ロールが作成されます。
   d. [Children's Online Privacy Protection Act (COPPA)] セクションで、[いいえ] を選択します。
   e. [アイドルセッションタイムアウト] および [詳細設定] セクションで、デフォルト設定のままにして、[次へ] を選択します。
   f. これで [ボットに言語を追加] セクションに移動しました。[音声インタラクション] の下のメニューで、[なし] を選択します。これは単なるテキストベースのアプリケーションです。残りのフィールドをデフォルト値のままにします。
   g. [Done] (完了) をクリックします。Amazon Lex V2 はボットと NewIntent というデフォルトインテントを作成し、このインテントを設定するページに移動します。

ボットを正常にビルドするには、`AMAZON.FallbackIntent` および `AMAZON.KendraSearchIntent` とは別のインテントを少なくとも 1 つ以上の作成します。このインテントは Amazon Lex V2 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。このインテントには少なくとも 1 つのサンプル発話が含まれている必要があり、その発話は顧客が尋ねるどの質問にも当てはまらないようにする必要があります。

必要なインテントを作成するには:

1. [インテント詳細] セクションで、インテントに名前 (**RequiredIntent** など) を付けます。
2. [サンプル発話] セクションで、[発話を追加] の横にあるボックスに、**Required utterance** などの発話を入力します。次に [発話を追加] を選択します。

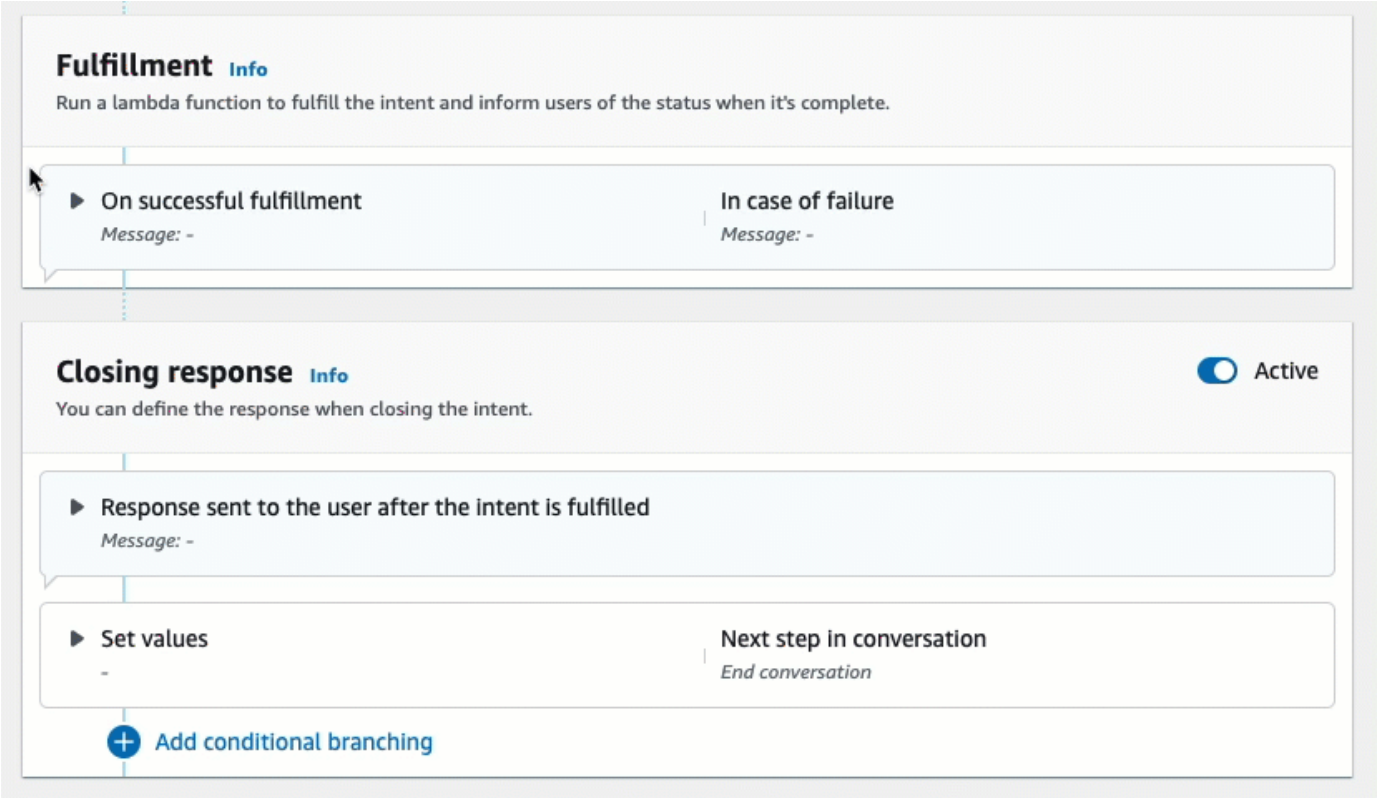
3. [インテントの保存] を選択します。

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の [インテントリストに戻る] を選択すると、ボットの [インテント] ページに戻ります。[インテントを追加] を選択し、ドロップダウンメニューから [組み込みインテントを使用する] を選択します。
2. 表示されるボックスで、[組み込みインテント] の下にあるメニューを選択します。検索バーに「**AMAZON.KendraSearchIntent**」と入力し、リストからそれを選択します。
3. インテントに **KendraSearchIntent** のような名前を付けます。
4. [Amazon Kendra インデックス] ドロップダウンメニューから、検索するインデックスを選択します。[前提条件] セクションで作成した索引が使用可能になっているはずです。
5. [追加] を選択します。
6. インテントエディターで [フルフィルメント] セクションまでスクロールし、右矢印を選択してセクションを展開し、[フルフィルメントが成功した場合] の下のボックスに次のメッセージを追加します。

```
I found a link to a document that could help you: ((x-amz-lex:kendra-search-
response-document-link-1)).
```



Amazon Kendra 検索レスポンスの詳細については、[検索レスポンスの使用] を参照してください。

7. [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。ボットの準備が整うと、画面上部のバナーが緑色に変わり、成功メッセージが表示されます。

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。

FAQ ボットをテストするには:

1. ボットが正常に構築されたら、[テスト] を選択します。
2. コンソールのテストウィンドウに「**`What is Amazon Kendra?`**」と入力します。ボットがリンクを返すことを確認します。
3. `AMAZON.KendraSearchIntent` の設定の詳細については、「AMAZON.KendraSearchIntent」と「KendraConfiguration」を参照してください。

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.QnAIntent

> **Note**
>
> 生成 AI 機能を活用するには、最初に以下の前提条件を満たす必要があります。
>
> 1. Amazon Bedrock の使用料金については、「Amazon Bedrock の料金」を参照してください。
> 2. ボットロケールで生成 AI 機能を有効にします。これを行うには、「生成 AI を使用した Lex V2 ボットの作成とパフォーマンスの最適化」の手順に従います。

Amazon Bedrock FM を使用してよくある質問の回答を検索して要約することにより、お客様の質問に回答します。このインテントが有効になるのは、発話がボット内に存在する他のインテントのいずれにも分類されない場合です。スロット値を引き出す際に発話見逃しがあった場合、このインテントは有効にならないので注意してください。認識された場合、AMAZON.QnAIntent は指定された Amazon Bedrock モデルを使用して設定済みの Amazon Bedrock ナレッジベースを検索し、顧客の質問に回答します。

> **Warning**
>
> AMAZON.QnAIntent と AMAZON.KendraSearchIntent を同じボットロケールで使用することはできません。

以下のナレッジストアオプションが利用できます。ナレッジストアが作成済みで、その中のドキュメントがインデックス化されている必要があります。

- OpenSearch サービスドメイン — インデックス化されたドキュメントが含まれています。ドメインを作成するには、「Amazon OpenSearch Service ドメインの作成と管理」の手順に従ってください。
- Amazon Kendra インデックス — インデックス化されたよくある質問ドキュメントが含まれています。Amazon Kendra インデックスを作成するには、「インデックスの作成」の手順に従ってください。
- Amazon Bedrock ナレッジベース — インデックス付けされたデータソースが含まれています。Amazon Bedrock ナレッジベースを設定するには、「ナレッジベースの構築」の手順に従ってください。

このインテントを選択した場合は、以下のフィールドを設定し、[追加] を選択してインテントを追加します。

- Bedrock モデル — このインテントで使用するプロバイダーと基盤モデルを選択します。利用可能な最新のモデルと廃止スケジュールを確認し、それに応じて移行を計画してください。詳細については、「モデルのライフサイクル」を参照してください。
- ナレッジストア — お客様の質問に回答するためにモデルで情報を取得するソースを選択します。以下のステータスがあります。
  - OpenSearch — 以下のフィールドを設定します。
    - ドメインエンドポイント — ドメイン用に作成したドメインエンドポイント、またはドメイン作成後に提供されたドメインエンドポイントを指定します。
    - インデックス名 – 検索するインデックスを指定します。詳細については、「Amazon OpenSearch Service でのデータのインデックス作成」を参照してください。
    - お客様に応答を返す方法を選択します。
      - 正確な応答 — このオプションを有効にすると、[回答] フィールドの値がそのままボットの応答に使用されます。設定済みの Amazon Bedrock 基盤モデルを使用して、内容の合成や要約を行わずに、正確な回答内容をそのまま選択します。OpenSearch データベースで設定した質問フィールドと回答フィールドの名前を指定します。
      - フィールドを含める — 指定したフィールドを使用してモデルが生成した回答を返します。OpenSearch データベースで設定したフィールドの名前を最大 5 つまで指定します。セミコロン (;) を使用してフィールド間を区切ります。
  - Amazon Kendra — 以下のフィールドを設定します。

- Amazon Kendra インデックス – ボットを使用して検索する Amazon Kendra インデックスを選択します。
- Amazon Kendra フィルター — フィルターを作成するには、このチェックボックスをオンにします。Amazon Kendra 検索フィルター JSON 形式の詳細については、「ドキュメント属性を使用した検索結果のフィルタリング」を参照してください。
- 正確な応答 — Amazon Kendra から返された正確な応答をボットが返すようにするには、このチェックボックスをオンにします。それ以外の場合は、選択した Amazon Bedrock モデルが、結果に基づいて応答を生成します。

> **Note**
>
> この機能を使用するには、まず「よくある質問 (FAQ) のインデックスへの追加」の手順に従って、FAQ の質問をインデックスに追加する必要があります。

- Amazon Bedrock ナレッジベース — このオプションを選択する場合は、Amazon Bedrock ナレッジベースの ID を指定します。ID は、コンソールの Amazon Bedrock ナレッジベースの詳細ページで確認できます。GetKnowledgeBase リクエストを送信して確認することもできます。
  - 正確な応答 — このオプションを有効にすると、[回答] フィールドの値がそのままボットの応答に使用されます。設定済みの Amazon Bedrock 基盤モデルを使用して、内容の合成や要約を行わずに、正確な回答内容をそのまま選択します。Amazon Bedrock ナレッジベースの正確な応答を使用するには、次の手順を実行する必要があります。
    - 各ファイルにエンドユーザーに返す必要がある正確な応答を含む回答フィールドを含めた、個別の JSON ファイルを作成します。
    - Bedrock ナレッジベースでこれらのドキュメントのインデックスを作成するときは、[チャンキング戦略] に [チャンキングなし] を選択します。
    - Amazon Lex V2 の回答フィールドを Bedrock ナレッジベースの回答フィールドとして定義します。

QnAIntent からの応答は、次に示すようにリクエスト属性に保存されます。

- `x-amz-lex:qnA-search-response` — 質問または発話に対する QnAIntent からの応答。
- `x-amz-lex:qnA-search-response-source` — 応答の生成に使用されたドキュメントまたはドキュメントのリストを指します。
- `x-amz-lex:qna-additional-context` – レスポンスを生成するために QnAIntent で使用される追加のコンテキスト。

追加のモデル設定

AMAZON.QnAIntent が呼び出されると、デフォルトのプロンプトテンプレートを使用し、指示とコンテキストをユーザーのクエリと組み合わせて、応答生成のためにモデルに送信されるプロンプトを作成します。カスタムプロンプトを指定したり、要件に合わせてデフォルトのプロンプトを更新したりすることもできます。

プロンプトテンプレートは、次のツールを使用して作成できます。

プロンプトプレースホルダー – AMAZON.QnAIntent for Amazon Bedrock で事前に定義された変数。これは、Bedrock 呼び出し中のランタイムに動的に入力されます。システムプロンプトでは、これらのプレースホルダーが $ 記号で囲まれていることを確認できます。次のリストは、使用できるプレースホルダーを示しています。

変数	置換後の値	モデル	必須?
$query_results$	ナレッジストアから取得したユーザークエリの結果	選択した Bedrock モデル	はい
$output_instruction$	応答の生成と引用をフォーマット化するための基本手順。モデルによって異なります。独自のフォーマット指示を定義する場合は、このプレースホルダーを削除することをお勧めします。	選択した Bedrock モデル	いいえ
$additional_context$	QnAIntent がレスポンスを生成するために使用する追加のコンテキスト	選択した Bedrock モデル	いいえ
$locale$	ボットが顧客のクエリに応答する言語	選択した Bedrock モデル	いいえ

使用されているデフォルトのプロンプトは次のとおりです。

```
$query_results$

$additional_context$

Please only follow the instructions in <instruction> tags below.
<instruction>
Given the conversation history, <additional_context> and <Context>:
(1) first, identify the user query intent and classify it as one of the categories:
 FAQ_QUERY, OTHER_QUERY, GIBBERISH, GREETINGS, AFFIRMATION, CHITCHAT, or MISC;
(2) second, if the intent is FAQ_QUERY, predict the most relevant grounding passage(s)
 by providing the passage id(s) or output CANNOTANSWER;
(3) then, generate a concise, to-the-point FAQ-style response in $locale$ locale
 ONLY USING the grounding content in <Context> and <additional_context>; or output
 CANNOTANSWER if the user query/request cannot be directly answered with the grounding
 content. DO NOT mention about the grounding passages such as ids or other meta data;
 do not create new content not presented in <Context>. Do NOT respond to query that is
 ill-intented or off-topic;
(4) lastly, provide the confidence level of the above prediction as LOW, MID or HIGH.
</instruction>

$output_instruction$
```

$output_instruction$ は次のように置き換えられます。

```
Give your final response in the following form:
<answer>
<intent>FAQ_QUERY or OTHER_QUERY or GIBBERISH or GREETINGS or AFFIRMATION or CHITCHAT
 or MISC</intent>
<text>a concise FAQ-style response or CANNOTANSWER</text>
<passage_id>passage_id or CANNOTANSWER</passage_id>
<confidence>LOW or MID or HIGH</confidence>
</answer>
```

> **Note**
>
> デフォルトの手順を使用しない場合、LLM が提供する出力はそのまま エンドユーザーに返されます。

出力指示には <text></text> タグと <passageId></passageId> タグを含める必要があり、LLM が応答とソースの帰属を提供するために passageId を返す指示を含める必要があります。

セッション属性による追加のコンテキストサポート

セッション属性 を使用して、実行AMAZON.QnAIntent時に追加のコンテキストを に渡すことができますx-amz-lex:qna-additional-context。これにより、レスポンスの生成時にモデルがナレッジストアの結果とともに使用できる補足情報を提供できます。追加のコンテキストは、$additional_context$プレースホルダーを介してプロンプトテンプレートに挿入されます。

例:

```
{"sessionAttributes": {"x-amz-lex:qna-additional-context":"Our support hours are Monday
 through Friday, 8AM-8PM EST"}}
```

セッション属性による Amazon Bedrock ナレッジベースのメタデータフィルタリングのサポート

Amazon Bedrock ナレッジベースのメタデータフィルターは、セッション属性 `x-amz-lex:bkb-retrieval-filter` の一部として渡すことができます。

```
            {"sessionAttributes":{"x-amz-lex:bkb-retrieval-filter":"{\"equals\":{\"key
\":\"insurancetype\",\"value\":\"farmers\"}}
```

> **Note**
>
> このフィルターを使用するには、QnAIntent のデータストアとして Amazon Bedrock ナレッジベースを使用する必要があります。詳細については、Metadata filteringを参照してください。

推論の設定

セッション属性を使用して LLM を呼び出すときに使用する推論の設定を定義できます。

- 温度: 整数型
- topP
- maxTokens

例:

```
        {"sessionAttributes":{"x-amz-lex:llm-text-inference-config":"{\"temperature
\":0,\"topP\":1,\"maxTokens\":200}"}}
```

Bedrock のガードレール機能は、ビルドタイム属性とセッション属性によってサポートします。

- ビルド時にコンソールを使用 – GuardrailsIdentifier と GuardrailsVersion を指定します。詳細については、「追加のモデル設定」セクションを参照してください。
- セッション属性を使用 – セッション属性 `x-amz-lex:bedrock-guardrails-identifier` および `x-amz-lex:bedrock-guardrails-version` を使用してガードレール機能設定を定義することもできます。

Bedrock のガードレール機能使用の詳細については、「ガードレール」を参照してください。

AMAZON.QnAIntent (multiple use support)

ロケール内に複数の AMAZON.QnAIntents を含めることを選択できます。Amazon Lex V2 は、ボットロケール内に最大 5 つの AMAZON.QnAIntents をサポートします。

AMAZON.QnAIntent は、次のいずれかのケースが true の場合にトリガーできます。

- ボットロケールに含まれる AMAZON.QnAIntent が 1 つだけで、そのインテントにサンプル発話が含まれていない場合、発話がボットに存在する他のインテントのいずれにも分類されていない場合にアクティブ化されます。このインテントが有効になるのは、発話がボット内に存在する他のインテントのいずれにも分類されない場合です。スロット値を引き出す際に発話見逃しがあった場合、このインテントは有効にならないので注意してください。

> **Note**
>
> FM からの応答が不十分だったり、FM への呼び出しが失敗したりすると、Amazon Lex V2 は `AMAZON.FallbackIntent` を呼び出します。

- AMAZON.QnAIntent にサンプル発話が含まれている場合、ユーザー入力に基づいて、ユーザーがそのインテントを開始したいと考えていると Amazon Lex V2 が認識したときにのみアクティブ化されます。

  > **Note**
  >
  > FM からの応答が不十分だったり、FM への呼び出しが失敗したりすると、Amazon Lex V2 はフルフィルメントブロックで定義された失敗時の次のステップを呼び出します。

> **Note**
>
> `botLocale` に複数の AMAZON.QnAIntent がある場合、各 AMAZON.QnAIntent には少なくとも 1 つのサンプル発話が必要です。

AMAZON.QinConnectIntent

> **Note**
>
> Amazon Q In Connect を使用して生成 AI 機能を使用するには、次の前提条件を満たす必要があります。
>
> 1. Amazon Connect コンソールに移動してインスタンスを作成します。まだインスタンスがない場合は、「Amazon Connect の使用を開始する」を参照してください。
> 2. インスタンスの Amazon Q in Connect を有効にします。「インスタンスで Amazon Q in Connect を有効にする」を参照してください。

AMAZON.QinConnectIntent は、LLM が強化した Amazon Connect Wisdom の進化形を使用して顧客の質問に応答します。この進化形は、コンタクトセンターの顧客とエージェントが顧客の問題を迅速かつ正確に解決できるようにリアルタイムの推奨事項を提供します。このインテントが有効になるのは、発話がボット内に存在する他のインテントのいずれにも分類されない場合です。スロット値を引き出す際に発話見逃しがあった場合、このインテントは有効にならないので注意してください。認識された場合、AMAZON.QinConnectIntent は指定された Q in Connect ドメインを使用して、設定済みの Amazon Bedrock ナレッジベースを検索し、顧客の質問に回答します。

> **Note**
>
> - 同じボットロケールで、AMAZON.QinConnectIntent を AMAZON.QnAIntent と共に使用することはできません。
> - 米国英語以外の言語を選択した場合は、指定された言語で応答するようにセルフサービスプロンプト (SELF_SERVICE_PRE_PROCESSING および SELF_SERVICE_ANSWER_GENERATION) をカスタマイズする必要があります。プロンプトをカスタマイズする方法の詳細については、「Amazon Q in Connect をカスタマイズする」を参照してください。

このインテントを選択した場合は、以下のフィールドを設定し、[インテントを保存] を選択してインテントをボットに追加します。

- Amazon Q In Connect の設定 - Amazon Q in Connect アシスタントの Amazon リソースネーム (ARN) を指定します。アシスタントの ARN パターン: `^arn:[a-z-]*?:wisdom:[a-z0-9-]*?:[0-9]{12}:[a-z-]*?/[a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}(?:/[a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}){0,2}$`>。

QinConnectIntent からのレスポンスは、以下に示すようにリクエスト属性に保存されます。

- `x-amz-lex:q-in-connect-response` – 質問または発話に対する QinConnectIntent からのレスポンス。

QinConnectIntent から返されるセッション属性

QinConnect インテントとのやり取りは、セッション属性を介した会話に関する追加データを提供します。

1. `x-amz-lex:q-in-connect:session-arn` – 会話中に Amazon Q In Connect で作成されたセッションの一意の識別子。
2. `x-amz-lex:q-in-connect:conversation-status` – QinConnect アシスタントまたはドメインとの会話の現在のステータス。このステータスに可能な値は以下の 3 つです。
   - `CLOSED`
   - `READY`

- PROCESSING

3. x-amz-lex:q-in-connect:conversation-status-reason – 上記の属性で報告された現在のステータスの理由を提供します。考えられる理由は次のとおりです。

   - SUCCESS – 顧客がもう質問することがなく、質問への回答が成功したことを示します。
   - FAILED – 顧客の質問に回答する際の失敗を示します。これらは主に、顧客の質問を理解できないことが原因です。
   - REJECTED – アシスタントが顧客の質問に答えることを拒否し、ユーザーやエージェントと話すなど、ボットとのやり取りの外で質問を処理して詳細情報を取得することを推奨していることを示します。

> **Note**
>
> Amazon Connect インスタンスによって生じた顧客とのやり取り中に QinConnectIntent を持つボットが呼び出された場合、セッション ARN を作成して Amazon Connect インスタンスから渡す必要があります。セッションを作成するには、Amazon Connect フローを Amazon Q in Connect ステップで設定できます。

制限事項

- 同じボットロケールで、AMAZON.QinConnectIntent を特定の発話のないインテント (AMAZON.QnAIntent、AMAZON.BedrockAgentIntent など) と共に使用することはできません。
- Amazon Connect インスタンスによって生じた顧客とのやり取り中に QinConnectIntent を持つボットが呼び出された場合、セッション ARN を作成して Amazon Connect インスタンスから渡す必要があります。セッションを作成するには、Amazon Connect フローを Amazon Q In Connect ステップで設定できます。
- ボットロケールごとに 2 つ以上の AMAZON.QinConnectIntent を含めることはできません。
- AMAZON.QinConnectIntent で使用する Amazon Q in Connect ドメインは、Amazon Lex V2 ボットと同じ AWS リージョンに存在する必要があります。

アクセス許可

QinConnect Intent が Amazon Lex V2 ボットで使用され、ボットがサービスリンクロール (SLR) を使用している場合、Amazon Lex V2 サービスには、ロールの適切なポリシーを更新して Q in Connect アシスタントと統合するためのアクセス許可があります。ボットがカスタム IAM ロールを

使用している場合、ユーザーはこれらのアクセス許可を IAM ロールに手動で追加する必要があります。

QinConnect インテントが追加された場合、サービスリンクロールは次のアクセス許可で更新されます。QinConnect アクセス用に新しいポリシーが追加されます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Sid": "QInConnectAssistantPolicy",
            "Action": [
                "wisdom:CreateSession",
                "wisdom:GetAssistant"
            ],
            "Resource": [
                "arn:aws:wisdom:*:accountId:assistant/assistantId",
                "arn:aws:wisdom:*:accountId:assistant/assistantId/*"
            ]
        },
        {
            "Effect": "Allow",
            "Sid": "QInConnectSessionsPolicy",
            "Action": [
                "wisdom:SendMessage",
                "wisdom:GetNextMessage"
            ],
            "Resource": [
                "arn:aws:wisdom:*:accountId:session/assistantId/*"
            ]
        },
        {
            "Sid": "QInConnectKmsCMKPolicy",
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:GenerateDataKey"
            ],
            "Resource": [
                "arn:aws:kms:region:accountId:key/keyId"
            ],
            "Condition": {
```

```
                "StringEquals": {
                    "aws:ResourceAccount": "accountId",
                    "kms:ViaService": "wisdom.region.amazonaws.com",
                    "kms:EncryptionContext:aws:wisdom:assistant:arn":
 ["arn:aws:wisdom:region:accountId:assistant/assistantId"]
                }
            }
        }
    ]
}
```

> **Note**
>
> QInConnectKmsCMKPolicy ステートメントは、Amazon Q in Connect アシスタントでカスタマーマネージド KMS キーを使用している場合にのみ必要です。

信頼ポリシー

```
{
    "Effect": "Allow",
    "Sid": "LexV2InternalTrustPolicy",
    "Principal": {
        "Service": "lexv2.aws.internal"
    },
    "Action": "sts:AssumeRole",
    "Condition": {
        "StringEquals": {
            "aws:SourceAccount": "accountId"
        },
        "ArnLike": {
            "aws:SourceArn": "arn:aws:lex:*:accountId:bot-alias/botId/*"
        }
    }
}
```

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:

- 停止
- オフ
- 黙って

スロットタイプの追加

スロットタイプは、ユーザーがインテント変数のために指定できる値を定義します。各言語のスロットタイプを定義して、値がその言語に固有になるようにします。例えば、ペイントカラーをリストするスロットタイプについては、英語で「red」の値を、フランス語で「"rouge」の値を、スペイン語で「rojo」の値を含めることができました。

このトピックでは、インテントのスロットのために値を提供するカスタムスロットタイプの作成方法について説明します。標準値のために組み込みスロットタイプを使用することもできます。例えば、世界各国のリストには、組み込みスロットタイプ `AMAZON.Country` を使用できます。

スロットタイプを作成するには

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/lex/ で Amazon Lex コンソールを開きます。
2. ボットのリストで、言語を追加するボットを選択し、Conversation structure を選択し、次に All languages を選択します。
3. スロットタイプを追加する言語を選択し、Slot types を選択します。
4. Add slot type を選択し、スロットタイプに名前を付けて、Add を選択します。
5. スロットタイプエディタで、スロットタイプの詳細を追加します。

    - Slot value resolution — スロット値の解決方法を指定します。Expand values を選択した場合、Amazon Lex V2 はトレーニングの代表的な値として、値を使用します。Restrict to slot values を使った場合、スロットの許容値は指定した値に制限されます。
    - Slot type values — スロットの値 Restrict to slot values を選択した場合、値の同義語を追加できます。例えば、値 "football"の場合、同義語の "soccer." を追加できます。ユーザーがボットとの会話で "soccer" を入力した場合、スロットの実際の値は "football" になります。
    - スロット値をカスタム語彙として使用 — このオプションを有効にすると、音声会話のスロット値やシノニムの認識が向上します。スロット値が「はい」、「いいえ」、「1」、「2、3」などの一般的な用語である場合は、このオプションを有効にしないでください。
6. Save slot type を選択します。

Amazon Lex V2 は、以下のようなスロットタイプを提供します。

トピック

- 組み込みスロットタイプ
- カスタムスロットタイプ
- 文法スロットタイプ
- コンポジットスロットタイプ

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.AlphaNumeric	文字と数字で構成される単語を認識します。		
AMAZON.City	都市を表す単語を認識します。	すべてのロケール	
Amazon 確認	「はい」、「いいえ」、「多分」、「わからない」を意味する単語を認識し、それらを標準 (はい/いいえ/多分/知らない) 形式に変換します。	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	

スロットタイプ	短い説明	サポート対象ロケール	
AMAZON.Currency	通過の単位を表す単語を認識して、標準の通貨の略語に変換	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	
AMAZON.FirstNameA	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	

スロットタイプ	短い説明	サポート対象ロケール	
AMAZON.State	州を表す単語を認識します。	すべてのロケール	
AMAZON.StreetName	通りの名前を表す単語を認識します。	すべてのロケール	
AMAZON.Time	時間を示す単語を認識し、時間形式に変換します。	すべてのロケール	
AMAZON.UK 郵便番号	英国の郵便番号を表す単語を認識し、標準形式に変換します。	英語 (英国) (en-GB) のみ	
AMAZON.FreeFormInput	任意の単語または文字で構成される文字列を認識します。	すべてのロケール	

AMAZON.AlphaNumeric

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

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)

AMAZON.AlphaNumeric スロットタイプはスペルスタイルを使った入力をサポートします。スペルバイレターおよびスペルバイワードのスタイルを使用して、カスタマーの文字入力を支援することができます。詳細については、「会話中のスペルスタイルによるスロット値のキャプチャ」を参照してください。

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。

- 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 V2 では、正規表現で次の文字がサポートされています。

- A ~ Z、a ~ z
- 0-9

Amazon Lex V2 では、正規表現で Unicode 文字もサポートされています。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 CreateSlotType 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 V2 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 V2 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 V2 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 V2 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 確認

このスロットタイプは、Amazon Lex V2 の「はい」、「いいえ」、「多分」、「わからない」のフレーズや単語に対応する入力フレーズを認識し、4 つの値のいずれかに変換します。これを使用して、ユーザーからの確認や承認をキャプチャすることができます。最終的に解決された値に基づいて、複数の会話パスを設計するための条件を作成できます。

例えば、次のようになります。

if {confirmation} = "Yes", fulfill the intent

else, elicit another slot

例:

- はい: うん、うん、わかった、わかった、持ってるよ、同意するよ...
- いいえ: いや、否定、いや、もういい、断る、まさか...
- 多分：可能性はある、たぶん、ときどき、そうかもしれない、そうかもしれない...
- わからない：知らない、不明、さっぱり、はっきりしない、誰も知らない...

2023 年 8 月 17 日現在、「確認」という名前のカスタムスロットタイプが既に存在する場合は、組み込みスロットの「確認」との競合を避けるために名前を変更する必要があります。Lex コンソー

ルの左側のナビゲーションで、スロットタイプ (「確認」という名前の既存のカスタムスロットタイプの場合) に移動し、スロットタイプ名を更新します。新しいスロットタイプ名に、組み込み確認スロットタイプの予約キーワードである「確認」は使用できません。

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.Currency

通貨を表す単語を、標準の ISO 4217 アルファベット通貨コードおよび数値に変換します。Amazon Lex V2 は通貨を認識しますが、ある通貨から別の通貨への変換はしません。

詳細については、国際標準化機構 (ISO) ウェブサイトの「通貨コード - ISO 4217」を参照してください。

表示される通貨は、次のように構成されています: `{Unit} {Amount}`

- {Unit} は、特定の通貨単位 (USD など) を指します。
- {Amount} は、小数点以下 2 桁 (300.00 など) の形式の金銭的価値を示します。

例 (以下のすべての例では en-US ロケールを使用しています。ロケールによって結果が異なる場合があります)。

- 「3USD」: USD 3.00
- 「USD300」: USD 300.00
- 「3 dimes」: USD 0.30
- 「$1.56」: USD 1.56
- 「5c」: USD 0.05
- 「1 dollar」: USD 1.00

- 「five fifteen」: USD 515.00
- 「five dollars fifteen cents」: USD 5.15
- 「5 usd and 1/2」: USD 5.50

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。
- 「次の週」など、将来の週にマップされる発話は、現在の週の最終日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 2020-11-29 に変換されます。現在週または前の週にマップされる日付は、週の最初の日に変換されます。例えば、今日が 2020-11-25 の場合、「先週」は 2020-11-16 に変換されます。
- 「来月」などの特定の日付ではなく、将来の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。現在月または前の月にマップされる日付は、月の最初の日に変換されます。例えば、今日が 2020-11-25 の場合、「今月」は 2020-11-01 に変換されます。
- 「来年」などの特定の月や日付ではなく、将来の年にマップされる発話は、その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。現在年または前年にマップされる日付については、その年の 1 日目に変換します。例えば、今日が 2020-11-25 の場合、「昨年」は 2019-01-01 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.EmailAddress スロットタイプはスペルスタイルを使った入力をサポートします。スペルバイレターおよびスペルバイワードのスタイルを使用して、カスタマーの E メールアドレス入力を支援することができます。詳細については、「会話中のスペルスタイルによるスロット値のキャプチャ」を参照してください。

AMAZON.FirstNameA

よく使われる名前。このスロットタイプは、正式な名前、非公式のニックネーム、複数の単語からなる名前を認識します。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 V2 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 V2 からインテントに共通フォームが送信されます。

AMAZON.FirstName スロットタイプはスペルスタイルを使った入力をサポートします。スペルバイレターおよびスペルバイワードのスタイルを使用して、カスタマーの名前入力を支援することができます。詳細については、「会話中のスペルスタイルによるスロット値のキャプチャ」を参照してください。

例:

- エミリー
- ジョン

- ソフィー
- Anil Kumar

Amazon.FirstName では、元の値に基づいて密接に関連する名前のリストも返されます。解決された値のリストを使用して、タイプミスを修正したり、ユーザーに名前を確認したり、ユーザーディレクトリ内の有効な名前をデータベースで検索したりできます。

たとえば、「John」と入力すると、「John J」や「John-Paul」などの関連する名前がさらに返される場合があります。

AMAZON.FirstName 組み込みスロットタイプの応答形式は次のとおりです。

```
"value": {
    "originalValue": "John",
    "interpretedValue": "John",
    "resolvedValues": [
        "John",
        "John J.",
        "John-Paul"
    ]
}
```

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 V2 からインテントに共通フォームが送信されます。

AMAZON.LastName スロットタイプはスペルスタイルを使った入力をサポートします。スペルバイレターおよびスペルバイワードのスタイルを使用して、カスタマーの名前入力を支援することができます。詳細については、「会話中のスペルスタイルによるスロット値のキャプチャ」を参照してください。

例:

- ブロスキー
- ダッシャー
- エバース
- パレス

- ウェルト

AMAZON.LastName では、元の値に基づいて密接に関連する名前のリストも返されます。解決された値のリストを使用して、タイプミスを修正したり、ユーザーに名前を確認したり、ユーザーディレクトリ内の有効な名前をデータベースで検索したりできます。

たとえば、「Smith」と入力すると、「Smyth」や「Smithe」などの関連する名前がさらに返される場合があります。

AMAZON.LastName 組み込みスロットタイプの応答形式は次のとおりです。

```
"value": {
    "originalValue": "Smith",
    "interpretedValue": "Smith",
    "resolvedValues": [
        "Smith",
        "Smyth",
        "Smithe"
    ]
}
```

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。

Input	[応答]
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

Input	[応答]
-15	-15
マイナス 15	-15
マイナス 15 ポイント 2 4 5	-15.245

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。

Input	[応答]
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	Input	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`

型	説明	Input	結果
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電話番号	555-1212	5551212

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。

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

時間を表す単語を時間値に変換します。AMAZON.Time で、正確な時刻、あいまいな値、時間範囲を解決できます。スロット値は次の時間範囲で解決できます。

- AM

- PM
- MO (午前)
- AF (午後)
- EV (夕方)
- NI (夜間)

ユーザーがあいまいな時刻を入力すると、Amazon Lex V2 は Lambda イベントの `slots` 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、`interpretedValue` フィールド値は `null` となり、`resolvedValues` フィールドに 2 つの可能な時刻の解決が含まれます。Amazon Lex V2 は Lambda 関数に次のように入力します。

```
"slots": {
    "deliveryTime": {
    "value": {
        "originalValue": "10 o'clock",
        "interpretedValue": null,
        "resolvedValues": [
            "10:00", "22:00"
        ]
    }
}
```

ユーザーが明確な時刻で応答すると、Amazon Lex V2 はLambdaイベントの `interpretedValue` 属性の `slots` フィールドで Lambda関数に時刻を送信します。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 AM」と返答すると、Amazon Lex V2 は Lambda 関数に次のように入力します。

```
"slots": {
    "deliveryTime": {
        "value": {
        "originalValue": "10 AM",
        "interpretedValue": 10:00,
        "resolvedValues": [
            "10:00"
        ]
    }
```

```
}
```

ユーザーが配達時間を求めるプロンプトに対して「午前」と応答すると、Amazon Lex V2 は Lambda 関数に次のように入力します。

```
"slots": {
    "deliveryTime": {
    "value": {
        "originalValue": "morning",
        "interpretedValue": "MO",
        "resolvedValues": [
            "MO"
        ]
    }
}
```

Amazon Lex V2 から Lambda 関数に送信されるデータの詳細については、AWS LambdaLex V2 の入力イベント形式 を参照してください。

AMAZON.UK 郵便番号

英国の郵便番号を表す単語を、英国の郵便番号の標準形式に変換します。AMAZON.UKPostalCode スロットタイプは、郵便コードを一連の標準化されたフォーマットに検証して解決しますが、郵便番号が有効かどうかはチェックされません。アプリケーションで、郵便コードを検証する必要があります。

AMAZON.UKPostalCode 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。

AMAZON.UKPostalCode スロットタイプはスペルスタイルを使った入力をサポートします。スペルバイレターおよびスペルバイワードのスタイルを使用して、カスタマーの文字入力を支援することができます。詳細については、「会話中のスペルスタイルによるスロット値のキャプチャ」を参照してください。

スロットタイプは、英国で使用される以下の有効な郵便番号形式のみを認識します。有効な形式は次のとおりです (「A」は文字、「9」は数字を表します)。

- AA9A 9AA
- A9A 9AA
- A9 9AA

- A99 9AA
- AA9 9AA
- AA99 9AA

テキスト入力の場合、ユーザは大文字と小文字を任意に組み合わせて入力できます。ユーザーは、郵便番号内のスペースを使用または省略できます。解決された値には、郵便番号の適切な場所にスペースが常に含まれます。

音声入力の場合、ユーザーは個々の文字を話すことができ、「ダブルA」や「ダブル9」などの二重文字の発音を使用できます。また、99 を九十九のように、二桁の発音を使用することもできます。

> **Note**
>
> すべての英国の郵便番号が認識されるわけではありません。上記の形式のみがサポートされています。

AMAZON.FreeFormInput

`AMAZON.FreeFormInput` は、エンドユーザーからの自由形式入力をキャプチャするために使用できます。単語または文字で構成される文字列を認識します。解決される値は入力された発話全体です。

例:

ボット: 通話体験から得たフィードバックをお寄せください。

ユーザー: すべての質問に対する回答が得られ、取引を完了することができました。

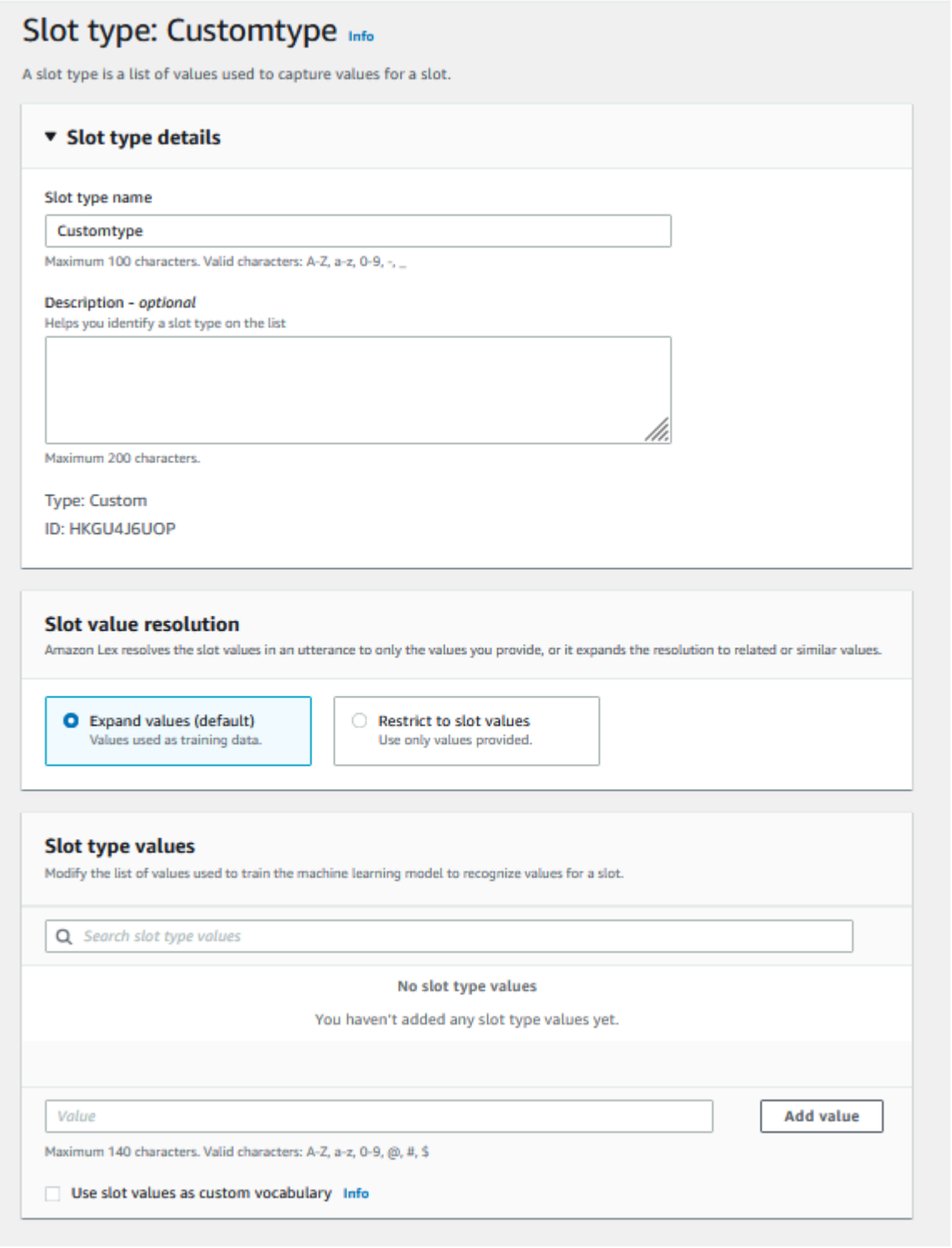
注記:

- `AMAZON.FreeFormInput` は、エンドユーザーからの自由形式入力をそのままキャプチャするために使用できます。
- `AMAZON.FreeFormInput` は、インテントのサンプル発話には使用できません。
- `AMAZON.FreeFormInput` に、スロットサンプル発話は使用できません。
- `AMAZON.FreeFormInput` は、誘発された時のみ認識されます。
- `AMAZON.FreeFormInput` は、待機して続行をサポートしていません。

- AMAZON.FreeFormInput は現在、Connect Customer Chat チャネルではサポートされていません。
- AMAZON.FreeFormInput スロットが誘発されても、FallbackIntent はトリガーされません。
- AMAZON.FreeFormInput スロットが誘発されても、インテントスイッチはありません。

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 V2 で使用する値のリストです。例えば、「comedy」、「adventure」、「documentary」などの値を指定して Genres というスロットタイプを定義できます。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。



または、スロット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、ユーザーが提供した値がスロット値やシノニムに類似している場合に、スロットがその値に解決されます。これがデフォルトの動作です。Amazon Lex V2 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 V2 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。

または、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。この場合、モデルによってユーザーが入力したスロット値が既存のスロット値に変換されるのは、そのスロット値がそのスロット値と同じか、シノニムである場合のみです。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。

ユーザーが入力した値がスロットタイプ値のシノニムである場合、モデルが `resolvedValues` のリストの最初のエントリとしてスロットタイプ値を返します。たとえば、ユーザーが「funny」と入力すると、`originalValue` フィールドに「funny」と自動入力され、[resolvedValues] フィールドの最初のエントリが「comedy」になります。CreateSlotType 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。

カスタムスロットタイプはスペルスタイルを使った入力をサポートします。スペルバイレターおよびスペルバイワードのスタイルを使用して、カスタマーの文字入力を支援することができます。詳細については、「会話中のスペルスタイルによるスロット値のキャプチャ」を参照してください。

Lambda 関数を使用している場合は、関数への入力イベントに `resolvedValues` 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セクションを示しています。

```
"slots": {
   "MovieGenre": {
      "value": {
         "originalValue": "funny",
         "interpretedValue": "comedy",
         "resolvedValues": [
            "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。

カスタムスロットタイプは、組み込みスロットタイプと同じ名前にできません。たとえば、カスタムスロットタイプに「日付」、「番号」、または「確認」という予約キーワードを使用して名前を付けるべきではありません。これらのキーワードは、組み込みスロットタイプ専用です。すべての組み込みスロットタイプについては、「組み込みスロットタイプ」を参照してください。

文法スロットタイプ

> **Important**
>
> Amazon Lex V2 は、各 AWS アカウントの文法スロットタイプへのアクセスを許可します。アカウントにアクセスできない場合、Amazon Lex V2 はボットへの文法スロットタイプの追加を拒否します。Amazon Lex V2 は、文法スロットタイプを含むロケールのビルドも拒否します。アカウントのアクセスをリクエストするには、AWS サポートにお問い合わせください。独自の文法を作成する必要がない場合は、代わりにカスタムスロットタイプを使用できます。

文法スロットタイプを使用すると、SRGS 仕様に従って XML 形式で独自の文法を作成して、会話の中で情報を収集できます。Amazon Lex V2 は、文法で指定されたルールに一致する発話を認識します。文法ファイル内の ECMAScript タグを使用してセマンティック解釈ルールを指定することもできます。Amazon Lex は、一致が発生すると、タグに設定されたプロパティを解決済みの値として返します。

文法スロットタイプは、英語 (オーストラリア)、英語 (英国)、および英語 (米国) のロケールでのみ作成できます。

文法スロットタイプには 2 つの部分があります。1 つ目は、SRGS 仕様フォーマットを使用して記述された文法自体です。文法はユーザーからの発話を解釈します。発話が文法に受け入れられた場合は一致し、そうでない場合は拒否されます。発話が一致すると、スクリプトに渡されます (ある場合)。

もう 1 つは文法スロットタイプの一部です。これは ECMAScript で記述されたオプションのスクリプトで、入力をスロットタイプによって返される解決済みの値に変換します。たとえば、スクリプトを使用して音声番号を数字に変換できます。ECMAScript ステートメントは <tag> 要素で囲まれています。

以下の例は、Amazon Lex V2 で受け入れられている有効な文法を示す、SRGS 仕様に従った XML 形式です。カード番号を受け付ける文法スロットタイプを定義し、カード番号が通常アカウント用か

プレミアムアカウント用かを判断します。許容される構文の詳細については、「文法の定義」および「文字起こし形式」トピックを参照してください。

```
<grammar version="1.0" xmlns="http://www.w3.org/2001/06/grammar"
         xml:lang="en-US" tag-format="semantics/1.0" root="card_number">

   <rule id="card_number" scope="public">
        <item repeat="0-1">
            card number
        </item>
        <item>
            seven
            <tag>out.value = "7";</tag>
         </item>
         <item>
            <one-of>
               <item>
                  two four one
                  <tag> out.value = out.value + "241"; out.card_type = "premium"; </
tag>
               </item>
               <item>
                  zero zero one
                  <tag> out.value = out.value + "001"; out.card_type = "regular";</tag>
               </item>
            </one-of>
         </item>
   </rule>
</grammar>
```

上記の文法では、7241 と 7001 の 2 種類のカード番号しか使用できません。どちらの場合も、オプションで「カード番号」のプレフィックスを付けることができます。また、セマンティックな解釈に使用できる ECMAScript タグも含まれています。セマンティックな解釈では、「カード番号 7 2 4 1」という発話は次のオブジェクトを返します。

```
{
    "value": "7241",
    "card_type": "premium"
}
```

このオブジェクトは、RecognizeText、RecognizeUtterance、および StartConversation オペレーションによって返される `resolvedValues` オブジェクトに、 JSON シリアル化された文字列として返されます。

文法スロットタイプの追加

文法スロットタイプを追加するには

1. スロットタイプの XML 定義を S3 バケットにアップロードします。バケット名、ファイルへのパスを書き留めておきます。

   > **Note**
   >
   > ファイルの最大サイズは 100 KB です。

2. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/lex/ で Amazon Lex コンソールを開きます。
3. 左側のメニューから [ボット] を選択し、文法スロットタイプを追加するボットを選択します。
4. [言語を表示] を選択し、文法スロットタイプを追加する言語を選択します。
5. [スロットタイプを表示] を選択します。
6. [スロットタイプを追加] を選択し、スロットタイプに名前を付けて、[追加] を選択します。
7. インテントに名前を付けて、[追加] を選択します。
8. 定義ファイルが含まれている S3 バケットを選択し、ファイルへのパスを入力します。Save slot type を選択します。

文法の定義

このトピックでは、Amazon Lex V2 がサポートする SRGS 仕様の一部を示します。すべてのルールは SRGS 仕様で定義されています。詳細については、音声認識文法仕様バージョン 1.0 W3C レコメンデーションを参照してください。

トピック

- ヘッダー宣言
- サポートされている XML 要素
- トークン
- ルール参照

- シーケンスとカプセル化
- 反復
- 言語
- タグ
- 重み

この文書には、W3C 音声認識文法仕様バージョン 1.0 (https://www.w3.org/TR/speech-grammar/ で入手可能) からコピーおよび派生した内容が含まれています。引用情報は次のとおりです。

Copyright © 2,004 W3C® (MIT, ERCIM, Keio, All Rights Reserved. W3C の責任、商標、文書の使用およびソフトウェアライセンスに関する規則が適用されます。

W3C レコメンデーションである SRGS 仕様書は、以下のライセンスに基づいて W3C から入手できます。

ライセンステキスト

ライセンス

本書、または本ステートメントがリンクされている W3C ドキュメントを使用またはコピーすることにより、お客様 (ライセンシー) は、以下の条件を読み、理解し、遵守することに同意したものとみなされます。

本書または本ステートメントのリンク元である W3C ドキュメントの内容を、目的を問わず、費用や使用料なしで、あらゆる媒体で複製および配布することをここに許可します。ただし、使用する文書のすべてのコピーまたはその一部に以下を含めることを条件とします。

- 元の W3C ドキュメントへのリンクまたは URL。
- 原作者の既存の著作権表示、または存在しない場合は、フォーム「Copyright © [$date-of-document] World Wide Web Consortium (MIT、ERCIM、Keio、Beihang)。http://www.w3.org/Consortium/Legal/2015/doc-license」の通知 (ハイパーテキストが望ましいが、テキスト表現も可)。
- 存在する場合は、W3C ドキュメントのステータス。

スペースに余裕がある場合は、本通知の全文を記載してください。本書の内容、あるいはその一部の実施に従ってお客様が作成したソフトウェア、文書、またはその他のアイテムまたは製品には、著者名義の帰属を記載するようお願いします。

以下の場合を除き、W3C ドキュメントの修正または派生版を作成する権利は、本ライセンスで付与されません。本書に定める技術仕様の実装を容易にするために、誰でもソフトウェア、ソフトウェアに付属する補足資料、およびソフトウェアのドキュメンテーションにおいて、二次的著作物および本書の一部を作成して配布することができます。ただし、かかるすべての著作物には、以下の注意事項が含まれていることが条件となります。ただし、本書の派生著作物を技術仕様として使用するために公開することは明示的に禁止されています。

さらに、「コードコンポーネント」(Web IDL と明記されたセクション内の Web IDL、コード例として明示されている W3C 定義のマークアップ (HTML、CSS など)、およびコンピュータプログラミング言語コード) は、W3C ソフトウェアライセンスに基づいてライセンスされています。

通知は以下のとおりです。

「Copyright © 2,015 W3C® (MIT, ERCIM, Keio, Beihang). このソフトウェアまたはドキュメントには、[W3C ドキュメントのタイトルと URI] からコピーまたは派生した内容が含まれています。」

免責事項

本書は「現状のまま」提供され、著作権者は、明示または黙示を問わず、商品性、特定目的への適合性、非侵害、または所有権の保証を含むがこれらに限定されない、ドキュメントの内容があらゆる目的に適していること、およびそのような内容の実装が第三者の特許、著作権、商標、またはその他の権利を侵害しないことについての表明または保証を行いません。

著作権者は、ドキュメントの使用またはその内容の実行または実装から生じる直接的、間接的、特別または結果的な損害について一切の責任を負わないものとします。

書面による事前の許可がない限り、著作権者の名前および商標を本書またはその内容に関する広告または宣伝に使用することはできません。本書の著作権の所有権は、常に著作権者に帰属します。

ヘッダー宣言

次の表は、文法スロットタイプがサポートするヘッダー宣言を示しています。詳細については、音声認識文法仕様バージョン 1 W3C レコメンデーションの「文法ヘッダー宣言」を参照してください。

宣言	仕様要件	XML フォーム	Amazon Lex のサポート	の仕様
文法バージョン	必須	4.3: `grammar` 要素の `version` 属性	必須	SRGS

宣言	仕様要件	XML フォーム	Amazon Lex のサポート	の仕様
XML 名前空間	必須 (XML のみ)	4.3: grammar 要素の xmlns 属性	必須	SRGS
[Document type (ドキュメントタイプ)]	必須 (XML のみ)	4.3: XML DOCTYPE	推奨	SRGS
文字エンコーディング	推奨	4.4: XML 宣言に含まれる encoding 属性	推奨	SRGS
言語	音声モードでは必須 DTMF モードでは無視	4.5: grammar 要素の xml:lang 属性	音声モードでは必須 DTMF モードでは無視	SRGS
モード	オプションです。	4.6: grammar 要素の mode 属性	オプションです。	SRGS
ルートルール	オプションです。	4.7: grammar 要素の root 属性	必須	SRGS
タグ形式	オプションです。	4.8: grammar 要素の tag-format 属性	文字列リテラルと ECMAScript がサポートされています	SRGS、SISR
基本 URI	オプションです。	4.9: grammar 要素の xml:base 属性	オプションです。	SRGS
発音レキシコン	オプション、複数可	4.10: lexicon 属性	サポートされません	SRGS, PLS

宣言	仕様要件	XML フォーム	Amazon Lex のサポート	の仕様
メタデータ	オプション、複数可	4.11.1: meta 属性	必須	SRGS
XML メタデータ	オプション、XML のみ	4.11.2: metadata 属性	オプションです。	SRGS
タグ	オプション、複数可	4.12: tag 属性	グローバルタグはサポートされていません	SRGS

例

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE grammar PUBLIC "-//W3C//DTD GRAMMAR 1.0//EN"
                  "http://www.w3.org/TR/speech-grammar/grammar.dtd">

<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xml:base="http://www.example.com/base-file-path"
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                             http://www.w3.org/TR/speech-grammar/grammar.xsd"
         xml:lang="en-US"
         version="1.0"
         mode="voice"
         root="city"
         tag-format="semantics/1.0">
```

サポートされている XML 要素

Amazon Lex V2 では、カスタム文法用に次の XML 要素がサポートされています。

- <item>
- <token>
- <tag>
- <one-of>

- `<rule-ref>`

トークン

次の表は、文法スロットタイプがサポートするトークン仕様を示しています。詳細については、音声認識文法仕様バージョン 1 W3C レコメンデーションの「トークン」を参照してください。

トークンタイプ	例	サポート対象?
引用符で囲まれていない 1 つのトークン	hello	はい
引用符で囲まれていない 1 つのトークン: アルファベット以外	2	はい
一重引用符で囲まれたトークン、スペースなし	"hello"	はい、トークンが 1 つしか含まれていない場合は、二重引用符を削除する
スペースで区切られた 2 つのトークン	bon voyage	はい
スペースで区切られた 4 つのトークン	this is a test	はい
一重引用符で囲まれたトークン、スペースを含む	"San Francisco	いいえ
<token> タグ内の 1 つの XML トークン	<token>サンフランシスコ</token>	いいえ (一重引用符で囲まれたトークン、スペースを含むと同じ)

Notes (メモ)

- 一重引用符付きトークン (空白を含む) — この仕様では、二重引用符で囲まれた単語を単一のトークンとして扱う必要があります。Amazon Lex V2 はそれらをスペースで区切られたトークンとして扱います。

- <token> の単一の XML トークン — 仕様では <token> で区切られた単語を 1 つのトークンとして示す必要があります。Amazon Lex V2 はそれらをスペースで区切られたトークンとして扱います。
- Amazon Lex V2 では、文法にどちらかの使用が見つかると、検証エラーが発生します。

例

```
<rule id="state" scope="public">
    <one-of>
        <item>FL</item>
        <item>MA</item>
        <item>NY</item>
    </one-of>
</rule>
```

ルール参照

次の表は、文法ドキュメント内で考えられるさまざまな形式のルール参照をまとめたものです。詳細については、音声認識文法仕様バージョン 1 W3C レコメンデーションの「ルール参照」を参照してください。

参照タイプ	XML フォーム	サポート
2.2.1 明示的なローカルルール参照	`<ruleref uri="#rulename"/>`	はい
2.2.2 URI で識別される文法の名前付きルールへの明示的な参照	`<ruleref uri="grammarURI#rulename"/>`	いいえ
2.2.2 URI で識別される文法のルートルールへの暗示的な参照	`<ruleref uri="grammarURI"/>`	いいえ
2.2.2 メディアタイプを持つ URI で識別される文法の名前付きルールへの明示的な参照	`<ruleref uri="grammarURI#rulename" type="media-type"/>`	いいえ

参照タイプ	XML フォーム	サポート
2.2.2 メディアタイプを持つ URI で識別される文法のルートルールへの暗示的な参照	`<ruleref uri="grammarURI" type="media-type"/>`	いいえ

2.2.3 特殊ルールの定義	`<ruleref special="NULL"/>`
	`<ruleref special="VOID"/>`
	`<ruleref special="GARBAGE"/>`
		いいえ

Notes (メモ)

1. 文法 URI は外部 URI です。例えば、`http://grammar.example.com/world-cities.grxml`。
2. メディアタイプには以下のものがあります。
   - `application/srgs+xml`
   - `text/plain`

例

```
<rule id="city" scope="public">
    <one-of>
        <item>Boston</item>
        <item>Philadelphia</item>
        <item>Fargo</item>
    </one-of>
</rule>

<rule id="state" scope="public">
    <one-of>
        <item>FL</item>
        <item>MA</item>
        <item>NY</item>
```

```
    </one-of>
</rule>

<!-- "Boston MA" -> city = Boston, state = MA -->
<rule id="city_state" scope="public">
    <ruleref uri="#city"/> <ruleref uri="#state"/>
</rule>
```

シーケンスとカプセル化

以下の例は、サポートされているシーケンスを示しています。詳細については、音声認識文法仕様バージョン 1 W3C レコメンデーションの「シーケンスとカプセル化」を参照してください。

例

```
<!-- sequence of tokens -->
this is a test

<!--sequence of rule references-->
<ruleref uri="#action"/> <ruleref uri="#object"/>

<!--sequence of tokens and rule references-->
the <ruleref uri="#object"/> is <ruleref uri="#color"/>

<!-- sequence container -->
<item>fly to <ruleref uri="#city"/> </item>
```

反復

次の表では、サポートされるルールの反復拡張が示されます。詳細については、音声認識文法仕様バージョン 1 W3C レコメンデーションの「反復」を参照してください。

XML フォーム 例	動作	サポート対象?
repeat="n" repeat="6"	含まれている式はちょうど「n」回繰り返されます。「n」は「0」または正の整数でなければなりません。	はい

XML フォーム 例	動作	サポート対象?
repeat="m-n" repeat="4-6"	含まれている拡張が「m」から「n」回 (両端を含む) 繰り返されます。「m」と「n」はどちらも「0」または正の整数で、「m」は「n」以下でなければなりません。	はい
repeat="m-" repeat="3-"	含まれている展開は「m」回以上 (含めて) 繰り返されます。「m」は「0」または正の整数でなければなりません。たとえば、「3-」は、格納されている拡張が 3 回、4 回、5 回以上発生する可能性があることを宣言します。	はい
repeat="0-1"	含まれる拡張はオプションです。	はい
<item repeat="2-4" repeat-prob="0.8">		いいえ

言語

以下の説明は、文法に適用される言語識別子が対象です。詳細については、音声認識文法仕様バージョン 1 W3C レコメンデーションの「言語」を参照してください。

デフォルトでは、文法は文法ヘッダーの言語宣言に言語識別子が与えられた単一言語文書です。その文法内のすべてのトークンは、特に宣言されていない限り、文法の言語に従って処理されます。文法レベルの言語宣言はサポートされていません。

以下の例で、次の操作を行います。

1. 「en-US」言語の文法ヘッダー宣言は、Amazon Lex V2 でサポートされています。

2. アイテムレベルの言語アタッチメント (#で強調表示) はサポートされていません。言語アタッチメントがヘッダー宣言と異なる場合、Amazon Lex V2 では検証エラーが発生します。

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE grammar PUBLIC "-//W3C//DTD GRAMMAR 1.0//EN"
                  "http://www.w3.org/TR/speech-grammar/grammar.dtd">

<!-- the default grammar language is US English -->
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                             http://www.w3.org/TR/speech-grammar/grammar.xsd"
         xml:lang="en-US" version="1.0">

  <!--
     single language attachment to tokens
     "yes" inherits US English language
     "oui" is Canadian French language
  -->
  <rule id="yes">
    <one-of>
      <item>yes</item>
      <item xml:lang="fr-CA">oui</item>
    </one-of>
  </rule>

  <!-- Single language attachment to an expansion -->
  <rule id="people1">
    <one-of xml:lang="fr-CA">
      <item>Michel Tremblay</item>
      <item>André Roy</item>
    </one-of>
  </rule>
</grammar>
```

タグ

以下の説明は、文法用に定義されたタグが対象です。詳細については、音声認識文法仕様バージョン 1 W3C レコメンデーションの「タグ」を参照してください。

SRGS 仕様に基づいて、タグは次のように定義できます。

1. 「ヘッダー宣言」で説明されている通り、ヘッダー宣言の一部として。
2. <rule> 定義の一部として。

以下のタグ形式がサポートされています。

- `semantics/1.0` (SISR, ECMAScript)
- `semantics/1.0-literals` (SISR 文字列リテラル)

以下のタグ形式がサポートされています。

- `swi-semantics/1.0` (ニュアンス独自)

例

```
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xml:base="http://www.example.com/base-file-path"
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                             http://www.w3.org/TR/speech-grammar/grammar.xsd"
         xml:lang="en-US"
         version="1.0"
         mode="voice"
         root="city"
         tag-format="semantics/1.0-literals">
    <rule id="no">
        <one-of>
            <item>no</item>
            <item>nope</item>
            <item>no way</item>
        </one-of>
        <tag>no</tag>
    </rule>
</grammar>
```

重み

要素には重み属性を追加できます。重みは正の浮動小数点値で、音声認識中に項目内のフレーズがどの程度ブーストされるかを表します。詳細については、音声認識文法仕様バージョン 1 W3C レコメンデーションの「重み」を参照してください。

重みは、0 より大きく 10 以下で、小数点以下桁数は 1 つのみである必要があります。重みが 0 より大きく 1 未満の場合、フレーズはマイナスにブーストされます。重みが 1 より大きく 10 未満の場合、フレーズはプラスにブーストされます。重み 1 は重みが皆無であると同じことであり、フレーズのブーストは行われません。

音声認識のパフォーマンスを向上させるために項目に適切な重みを付けるのは難しい作業です。重みを割り当てる際のヒントをいくつか紹介します。

- 項目に重みを割り当てていない文法から始めましょう。
- スピーチのどのパターンがよく誤認されているかを判断します。
- 音声認識のパフォーマンスが向上し、リグレッションがなくなるまで、重みに異なる値を適用します。

例 1

たとえば、空港に関する文法があって、ニューヨークがニューアークと誤認されることが多い場合は、重み 5 を割り当てることでニューヨークをポジティブに評価できます。

```
<rule> id="airport">
    <one-of>
        <item>
            Boston
            <tag>out="Boston"</tag>
        </item>
        <item weight="5">
            New York
            <tag>out="New York"</tag>
        </item>
        <item>
            Newark
            <tag>out="Newark"</tag>
        </item>
    </one-of>
</rule>
```

例 2

たとえば、航空会社の予約コードに、英語のアルファベットで始まり、その後に 3 桁が続く文法があるとします。予約コードはほとんどの場合 B または D で始まりますが、B は P、D は T と誤認されることが多いことに気付きます。B と D をプラスにブーストできます。

```
<rule> id="alphabet">
    <one-of>
        <item>A<tag>out.letters+='A';</tag></item>
        <item weight="3.5">B<tag>out.letters+='B';</tag></item>
        <item>C<tag>out.letters+='C';</tag></item>
        <item weight="2.9">D<tag>out.letters+='D';</tag></item>
        <item>E<tag>out.letters+='E';</tag></item>
        <item>F<tag>out.letters+='F';</tag></item>
        <item>G<tag>out.letters+='G';</tag></item>
        <item>H<tag>out.letters+='H';</tag></item>
        <item>I<tag>out.letters+='I';</tag></item>
        <item>J<tag>out.letters+='J';</tag></item>
        <item>K<tag>out.letters+='K';</tag></item>
        <item>L<tag>out.letters+='L';</tag></item>
        <item>M<tag>out.letters+='M';</tag></item>
        <item>N<tag>out.letters+='N';</tag></item>
        <item>O<tag>out.letters+='O';</tag></item>
        <item>P<tag>out.letters+='P';</tag></item>
        <item>Q<tag>out.letters+='Q';</tag></item>
        <item>R<tag>out.letters+='R';</tag></item>
        <item>S<tag>out.letters+='S';</tag></item>
        <item>T<tag>out.letters+='T';</tag></item>
        <item>U<tag>out.letters+='U';</tag></item>
        <item>V<tag>out.letters+='V';</tag></item>
        <item>W<tag>out.letters+='W';</tag></item>
        <item>X<tag>out.letters+='X';</tag></item>
        <item>Y<tag>out.letters+='Y';</tag></item>
        <item>Z<tag>out.letters+='Z';</tag></item>
    </one-of>
</rule>
```

文字起こし形式

Amazon Lex V2 では、次の文法定義の ECMAScript 機能がサポートされています。

Amazon Lex V2 では、文法でタグを指定する際に次の ECMAScript 機能がサポートされています。ECMAScript タグを使用する場合、`tag-format` を `semantics/1.0` に送信する必要があります。詳細については、「ECMA-262 ECMAScript 2021 言語仕様」を参照してください。

```
<grammar version="1.0"
xmlns="http://www.w3.org/2001/06/grammar"
xml:lang="en-US"
tag-format="semantics/1.0"
root="card_number">
```

トピック

- 変数ステートメント
- 表現
- If ステートメント
- 切り替えステートメント
- 関数宣言
- 反復ステートメント
- ブロックステートメント
- コメント
- サポートされていないステートメント

このドキュメントには ECMAScript 標準 (https://www.ecma-international.org/publications-and-standards/standards/ecma-262/ で入手可能) の資料が含まれています。ECMAScript 言語仕様書は、以下のライセンスに基づいて Ecma International から入手できます。

ライセンステキスト

© 2,020 Ecma International

本文書は複製、出版、他者への配布が可能であり、その一部または一部の派生著作物は、その全部または一部を作成、複製、出版、配布することができます。ただし、上記の著作権表示および本著作権使用許諾書および免責事項が、かかる複製物および派生著作物のすべてに含まれている場合に限ります。本著作権ライセンスおよび免責事項のもとで許容される二次的著作物は以下に限定されます。

(i) 解説または説明を提供する目的で本書の全部または一部を組み込んだ作品 (文書の注釈付きバージョンなど)

(ii) アクセシビリティを提供する機能を組み込む目的で本書の全部または一部を組み込んだ作品

(iii) 本書の英語以外の言語および異なる形式への翻訳、および

(iv) 本仕様書に含まれる機能を (全部または一部をコピーアンドペーストなどで) 実装することにより、この仕様書を標準適合製品で使用することで機能します。

ただし、英語以外の言語または異なる形式への翻訳に必要な場合を除き、著作権表示や Ecma International への参照を削除するなど、本書の内容自体を変更することはいかなる方法でもできません。

ECMA International ドキュメントの公式バージョンは、Ecma Internationalのウェブサイトにある英語版です。翻訳版と公式版の間に相違がある場合は、公式版が優先されるものとします。

上記で付与された限定的な許可は永続的であり、Ecma International またはその後継者または譲受人によって取り消されることはありません。本書および本書に含まれる情報は「現状のまま」提供され、ECMA International は、明示または黙示を問わず、一切の保証を否認します。これには、本書に記載されている情報の使用が、所有権、または商品性または特定目的への適合性に関する黙示的保証を侵害しないという保証が含まれますが、これらに限定されません。」

変数ステートメント

変数ステートメントは 1 つ以上の変数を定義します。

```
var x = 10;
var x = 10, var y = <expression>;
```

表現

式文字列を追加して、Amazon Lex V2 で関数を実行できます。この表は、SRGS 式で使用できる構文と例を示しています。

式タイプ	Syntax	例	サポート対象?
正規表現リテラル	有効な正規表現特殊文字を含む文字列リテラル	`"^\d\.$"`	いいえ
関数	`function functionN`	`var x = function calc() { return 10;`	いいえ

式タイプ	Syntax	例	サポート対象?
	ame(param eters) { functionBody}	}	
Delete	delete expressio n	delete obj.prope rty;	いいえ
Void	void expression	void (2 == '2');	いいえ
Typeof	typeof expressio n	typeof 42;	いいえ
メンバーインデックス	expression [expressions]	var fruits = ["apple"]; fruits[0];	はい
メンバードット	expression . identifier	out.value	はい
引数	expression (arguments)	new Date('199 4-10-11')	はい
インクリメント後	expression++	var x=10; x++;	はい
デクリメント後	expression--	var x=10; x--;	はい
インクリメント前	++expression	var x=10; ++x;	はい
デクリメント後	--expression	var x=10; --x;	はい
単項プラス/単項マイナス	+expression / - expression	+x / -x;	はい

式タイプ	Syntax	例	サポート対象?
ビット否定	`~ expression`	`const a = 5; consol e.log( ~a );`	はい
論理否定	`! expression`	`!(a > 0 || b > 0)`	はい
乗算	`expression ('*' | '/' | '%') expression`	`(x + y) * (a / b)`	はい
追加	`expression ('+' | '-') expressio n`	`(a + b) - (a - (a + b))`	はい
ビットシフト	`expression ('<<' | '>>' | '>>>') expressio n`	`(a >> b) >>> c`	はい
相対	`expression ('<' | '>' | '<=' | '>=') expressio n`	`if (a > b) { ... }`	はい
In (中にある)	`expression in expression`	`fruits[0] in otherFruits;`	はい
等価	`expression ('==' | '!=' | '===' | '!==') expression`	`if (a == b) { ... }`	はい
ビット and / xor / or	`expression ('&' | '^' | '|') expression`	`a & b / a ^ b / a | b`	はい

式タイプ	Syntax	例	サポート対象?
論理的 and / or	`expression ('&&' | '||') expression`	`if (a && (b ||c)) { ...}`	はい
三項	`expression ? expression : expression`	`a > b ? obj.prop : 0`	はい
代入	`expression = expression`	`out.value = "string";`	はい
代入演算子	`expression ('*=' | '/=' | '+=' | '-=' | '%=') expression`	`a *= 10;`	はい
代入ビット演算子	`expression ('<<=' | '>>=' | '>>>=' | '&=' | '^=' | '|=') expression`	`a <<= 10;`	はい
識別子	`identifierSequence` 、ここでは識別子/シーケンスが有効な文字のシーケンスです。	`fruits=[10, 20, 30];`	はい
Null リテラル	`null`	`x = null;`	はい
ブールリテラル	`true | false`	`x = true;`	はい

式タイプ	Syntax	例	サポート対象?
文字列リテラル	`'string' / "string"`	`a = 'hello', b = "world";`	はい
10 進リテラル	`integer [.] digits [exponent]`	`111.11 e+12`	はい
16 進リテラル	`0 (x | X)[0-9a-fA-F]`	`0x123ABC`	はい
8 進リテラル	`0 [0-7]`	`"051"`	はい
配列リテラル	`[ expression, ... ]`	`v = [a, b, c];`	はい
オブジェクトリテラル	`{property: value, ...}`	`out = {value: 1, flag: false};`	はい
括弧付き	`( expressions )`	`x + (x + y)`	はい

If ステートメント

Amazon Lex V2 で関数を実行するための if ステートメントを追加できます。この例では、SRGS 式で使用できる構文を示しています。

```
if (expressions) {
    statements;
} else {
    statements;
}
```

注: 前述の例では、`expressions` と `statements` が、このドキュメントでサポートされているもののいずれかである必要があります。

切り替えステートメント

Amazon Lex V2 で関数を実行するスイッチステートメントを追加できます。この例では、SRGS 式で使用できる構文を示しています。

```
switch (expression) {
    case (expression):
     statements
     .
     .
     .
   default:
      statements
}
```

注: 前述の例では、expressions と statements が、このドキュメントでサポートされているもののいずれかである必要があります。

関数宣言

関数宣言を追加して、Amazon Lex V2 で関数を実行できます。この例では、SRGS 式で使用できる構文を示しています。

```
function functionIdentifier([parameterList, ...]) {
     <function body>
}
```

反復ステートメント

反復ステートメントは、次のいずれかを指定できます。

```
// Do..While statement
do {
   statements
} while (expressions)


// While Loop
while (expressions) {
   statements
}
```

```
// For Loop
for ([initialization]; [condition]; [final-expression])
   statement

// For..In
for (variable in object) {
  statement
}
```

ブロックステートメント

Amazon Lex V2 で関数を実行するためのステートメントブロックを追加できます。この例では、SRGS 式で使用できる構文を示しています。

```
{
   statements
}

// Example
{
    x = 10;
   if (x > 10) {
     console.log("greater than 10");
   }
}
```

注: 前述の例では、ブロックに提供された `statements` が、このドキュメントでサポートされているもののいずれかである必要があります。

コメント

Amazon Lex V2 でコメントを追加できます。この例では、SRGS 式で使用できる構文を示しています。

```
// Single Line Comments
"// <comment>"

// Multineline comments
/**
<comment>
**/
```

サポートされていないステートメント

Amazon Lex V2 は次の ECMAScript 機能をサポートしていません。

トピック

- 空のステートメント
- Continue ステートメント
- Break ステートメント
- Return ステートメント
- Throw ステートメント
- Try ステートメント
- Debugger ステートメント
- ラベル付きステートメント
- クラス宣言

空のステートメント

空のステートメントはステートメントを指定しない場合に使用します。空のステートメントの構文は次のとおりです。

```
;
```

Continue ステートメント

ラベルのない continue ステートメントは、反復ステートメント でサポートされています。ラベル付きの continue ステートメントは、サポートされていません。

```
// continue with label
// this allows the program to jump to a
// labelled statement (see labelled statement below)
continue <label>;
```

Break ステートメント

ラベルのない break ステートメントは、反復ステートメント でサポートされています。ラベル付きの break ステートメントは、サポートされていません。

```
// break with label
```

```
// this allows the program to break out of a
// labelled statement (see labelled statement below)
break <label>;
```

Return ステートメント

```
return expression;
```

Throw ステートメント

throw ステートメントは、ユーザー定義の例外を投げるために使用されます。

```
throw expression;
```

Try ステートメント

```
try {
  statements
}
catch (expression) {
  statements
}
finally {
  statements
}
```

Debugger ステートメント

debugger ステートメントは、環境によって提供されるデバッグ機能を呼び出すために使用されます。

```
debugger;
```

ラベル付きステートメント

ラベル付きステートメントは `break` または `continue` ステートメントと併用できます。

```
label:
   statements


// Example
```

```
let str = '';

loop1:
for (let i = 0; i < 5; i++) {
  if (i === 1) {
    continue loop1;
  }
  str = str + i;
}

console.log(str);
```

クラス宣言

```
class Rectangle {
  constructor(height, width) {
    this.height = height;
    this.width = width;
  }
}
```

業界文法

業界文法は、文法スロットタイプで使用する XML ファイルのセットです。これらを使用すると、インタラクティブな音声応答ワークフローを Amazon Lex V2 に移行する際に、一貫したエンドユーザーエクスペリエンスをすばやく提供できます。金融サービス、保険、通信の 3 つの分野にわたって、あらかじめ用意されているさまざまな文法から選択できます。独自の文法の開始点として使用できる文法セットもあります。

文法には、情報を収集するための規則と、セマンティック解釈のための ECMAScript タグが含まれています。

金融サービスの文法 (ダウンロード)

金融サービスでは、口座番号、ルーティング番号、クレジットカード番号、ローン番号、クレジットスコア、口座開設日と決済日、社会保障番号などの文法がサポートされています。

アカウント番号

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                             http://www.w3.org/TR/speech-grammar/grammar.xsd"
         xml:lang="en-US" version="1.0"
         root="main"
         mode="voice"
         tag-format="semantics/1.0">


        <!-- Test Cases

        Grammar will support the following inputs:

            Scenario 1:
                Input: My account number is A B C 1 2 3 4
                Output: ABC1234

            Scenario 2:
                Input: My account number is 1 2 3 4 A B C
                Output: 1234ABC

            Scenario 3:
                Input: Hmm My account number is 1 2 3 4 A B C 1
                Output: 123ABC1
        -->

        <rule id="main" scope="public">
            <tag>out=""</tag>
            <item><ruleref uri="#alphanumeric"/><tag>out +=
rules.alphanumeric.alphanum;</tag></item>
            <item repeat="0-1"><ruleref uri="#alphabets"/><tag>out +=
rules.alphabets.letters;</tag></item>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out +=
rules.digits.numbers</tag></item>
        </rule>

        <rule id="text">
            <item repeat="0-1"><ruleref uri="#hesitation"/></item>
            <one-of>
                <item repeat="0-1">account number is</item>
                <item repeat="0-1">Account Number</item>
                <item repeat="0-1">Here is my Account Number </item>
                <item repeat="0-1">Yes, It is</item>
                <item repeat="0-1">Yes It is</item>
                <item repeat="0-1">Yes It's</item>
```

```
                <item repeat="0-1">My account Id is</item>
                <item repeat="0-1">This is the account Id</item>
                <item repeat="0-1">account Id</item>
            </one-of>
        </rule>

        <rule id="hesitation">
          <one-of>
             <item>Hmm</item>
             <item>Mmm</item>
             <item>My</item>
          </one-of>
        </rule>

        <rule id="alphanumeric" scope="public">
            <tag>out.alphanum=""</tag>
            <item><ruleref uri="#alphabets"/><tag>out.alphanum +=
rules.alphabets.letters;</tag></item>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out.alphanum +=
rules.digits.numbers</tag></item>
        </rule>

        <rule id="alphabets">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <tag>out.letters=""</tag>
            <tag>out.firstOccurence=""</tag>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out.firstOccurence +=
rules.digits.numbers; out.letters += out.firstOccurence;</tag></item>
            <item repeat="1-">
                <one-of>
                    <item>A<tag>out.letters+='A';</tag></item>
                    <item>B<tag>out.letters+='B';</tag></item>
                    <item>C<tag>out.letters+='C';</tag></item>
                    <item>D<tag>out.letters+='D';</tag></item>
                    <item>E<tag>out.letters+='E';</tag></item>
                    <item>F<tag>out.letters+='F';</tag></item>
                    <item>G<tag>out.letters+='G';</tag></item>
                    <item>H<tag>out.letters+='H';</tag></item>
                    <item>I<tag>out.letters+='I';</tag></item>
                    <item>J<tag>out.letters+='J';</tag></item>
                    <item>K<tag>out.letters+='K';</tag></item>
                    <item>L<tag>out.letters+='L';</tag></item>
                    <item>M<tag>out.letters+='M';</tag></item>
                    <item>N<tag>out.letters+='N';</tag></item>
```

```
                    <item>O<tag>out.letters+='O';</tag></item>
                    <item>P<tag>out.letters+='P';</tag></item>
                    <item>Q<tag>out.letters+='Q';</tag></item>
                    <item>R<tag>out.letters+='R';</tag></item>
                    <item>S<tag>out.letters+='S';</tag></item>
                    <item>T<tag>out.letters+='T';</tag></item>
                    <item>U<tag>out.letters+='U';</tag></item>
                    <item>V<tag>out.letters+='V';</tag></item>
                    <item>W<tag>out.letters+='W';</tag></item>
                    <item>X<tag>out.letters+='X';</tag></item>
                    <item>Y<tag>out.letters+='Y';</tag></item>
                    <item>Z<tag>out.letters+='Z';</tag></item>
                </one-of>
            </item>
        </rule>

        <rule id="digits">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <tag>out.numbers=""</tag>
            <item repeat="1-10">
                <one-of>
                    <item>0<tag>out.numbers+=0;</tag></item>
                    <item>1<tag>out.numbers+=1;</tag></item>
                    <item>2<tag>out.numbers+=2;</tag></item>
                    <item>3<tag>out.numbers+=3;</tag></item>
                    <item>4<tag>out.numbers+=4;</tag></item>
                    <item>5<tag>out.numbers+=5;</tag></item>
                    <item>6<tag>out.numbers+=6;</tag></item>
                    <item>7<tag>out.numbers+=7;</tag></item>
                    <item>8<tag>out.numbers+=8;</tag></item>
                    <item>9<tag>out.numbers+=9;</tag></item>
                </one-of>
            </item>
        </rule>
</grammar>
```

支店コード

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                             http://www.w3.org/TR/speech-grammar/grammar.xsd"
```

```
         xml:lang="en-US" version="1.0"
         root="digits"
         mode="voice"
         tag-format="semantics/1.0">

         <!-- Test Cases

         Grammar will support the following inputs:

             Scenario 1:
                 Input: My routing number is 1 2 3 4 5 6 7 8 9
                 Output: 123456789

             Scenario 2:
                 Input: routing number 1 2 3 4 5 6 7 8 9
                 Output: 123456789

         -->

        <rule id="digits">
            <tag>out=""</tag>
            <item><ruleref uri="#singleDigit"/><tag>out += rules.singleDigit.digit;</
tag></item>
        </rule>

        <rule id="text">
            <item repeat="0-1"><ruleref uri="#hesitation"/></item>
           <one-of>
              <item repeat="0-1">My routing number</item>
              <item repeat="0-1">Routing number of</item>
              <item repeat="0-1">The routing number is</item>
           </one-of>
        </rule>

        <rule id="hesitation">
            <one-of>
               <item>Hmm</item>
               <item>Mmm</item>
               <item>My</item>
            </one-of>
          </rule>

        <rule id="singleDigit">
            <item repeat="0-1"><ruleref uri="#text"/></item>
```

```
            <tag>out.digit=""</tag>
            <item repeat="16">
                <one-of>
                    <item>0<tag>out.digit+=0;</tag></item>
                    <item>zero<tag>out.digit+=0;</tag></item>
                    <item>1<tag>out.digit+=1;</tag></item>
                    <item>one<tag>out.digit+=1;</tag></item>
                    <item>2<tag>out.digit+=2;</tag></item>
                    <item>two<tag>out.digit+=2;</tag></item>
                    <item>3<tag>out.digit+=3;</tag></item>
                    <item>three<tag>out.digit+=3;</tag></item>
                    <item>4<tag>out.digit+=4;</tag></item>
                    <item>four<tag>out.digit+=4;</tag></item>
                    <item>5<tag>out.digit+=5;</tag></item>
                    <item>five<tag>out.digit+=5;</tag></item>
                    <item>6<tag>out.digit+=6;</tag></item>
                    <item>six<tag>out.digit+=5;</tag></item>
                    <item>7<tag>out.digit+=7;</tag></item>
                    <item>seven<tag>out.digit+=7;</tag></item>
                    <item>8<tag>out.digit+=8;</tag></item>
                    <item>eight<tag>out.digit+=8;</tag></item>
                    <item>9<tag>out.digit+=9;</tag></item>
                    <item>nine<tag>out.digit+=9;</tag></item>
                </one-of>
            </item>
        </rule>
</grammar>
```

クレジットカード番号

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                             http://www.w3.org/TR/speech-grammar/grammar.xsd"
         xml:lang="en-US" version="1.0"
         root="digits"
         mode="voice"
         tag-format="semantics/1.0">

         <!-- Test Cases

         Grammar will support the following inputs:
```

```
             Scenario 1:
                 Input: My credit card number is 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7
                 Output: 1234567891234567

             Scenario 2:
                 Input: card number 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7
                 Output: 1234567891234567

          -->

        <rule id="digits">
            <tag>out=""</tag>
            <item><ruleref uri="#singleDigit"/><tag>out += rules.singleDigit.digit;</
tag></item>
        </rule>

        <rule id="text">
            <item repeat="0-1"><ruleref uri="#hesitation"/></item>
           <one-of>
              <item repeat="0-1">My credit card number is</item>
              <item repeat="0-1">card number</item>
           </one-of>
        </rule>

        <rule id="hesitation">
            <one-of>
               <item>Hmm</item>
               <item>Mmm</item>
               <item>My</item>
            </one-of>
          </rule>

        <rule id="singleDigit">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <tag>out.digit=""</tag>
            <item repeat="16">
                <one-of>
                    <item>0<tag>out.digit+=0;</tag></item>
                    <item>zero<tag>out.digit+=0;</tag></item>
                    <item>1<tag>out.digit+=1;</tag></item>
                    <item>one<tag>out.digit+=1;</tag></item>
                    <item>2<tag>out.digit+=2;</tag></item>
                    <item>two<tag>out.digit+=2;</tag></item>
```

```
                    <item>3<tag>out.digit+=3;</tag></item>
                    <item>three<tag>out.digit+=3;</tag></item>
                    <item>4<tag>out.digit+=4;</tag></item>
                    <item>four<tag>out.digit+=4;</tag></item>
                    <item>5<tag>out.digit+=5;</tag></item>
                    <item>five<tag>out.digit+=5;</tag></item>
                    <item>6<tag>out.digit+=6;</tag></item>
                    <item>six<tag>out.digit+=5;</tag></item>
                    <item>7<tag>out.digit+=7;</tag></item>
                    <item>seven<tag>out.digit+=7;</tag></item>
                    <item>8<tag>out.digit+=8;</tag></item>
                    <item>eight<tag>out.digit+=8;</tag></item>
                    <item>9<tag>out.digit+=9;</tag></item>
                    <item>nine<tag>out.digit+=9;</tag></item>
                </one-of>
            </item>
        </rule>
</grammar>
```

ローン ID

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                             http://www.w3.org/TR/speech-grammar/grammar.xsd"
         xml:lang="en-US" version="1.0"
         root="main"
         mode="voice"
         tag-format="semantics/1.0">


        <!-- Test Cases

        Grammar will support the following inputs:

            Scenario 1:
                Input: My loan Id is A B C 1 2 3 4
                Output: ABC1234
        -->

        <rule id="main" scope="public">
            <tag>out=""</tag>
```

```
            <item><ruleref uri="#alphanumeric"/><tag>out +=
rules.alphanumeric.alphanum;</tag></item>
            <item repeat="0-1"><ruleref uri="#alphabets"/><tag>out +=
rules.alphabets.letters;</tag></item>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out +=
rules.digits.numbers</tag></item>
        </rule>

        <rule id="text">
            <item repeat="0-1"><ruleref uri="#hesitation"/></item>
            <one-of>
                <item repeat="0-1">my loan number is</item>
                <item repeat="0-1">The loan number</item>
                <item repeat="0-1">The loan is </item>
                <item repeat="0-1">The number is</item>
                <item repeat="0-1">loan number</item>
                <item repeat="0-1">loan number of</item>
                <item repeat="0-1">loan Id is</item>
                <item repeat="0-1">My loan Id is</item>
            </one-of>
        </rule>

        <rule id="hesitation">
          <one-of>
             <item>Hmm</item>
             <item>Mmm</item>
             <item>My</item>
          </one-of>
        </rule>

        <rule id="alphanumeric" scope="public">
            <tag>out.alphanum=""</tag>
            <item><ruleref uri="#alphabets"/><tag>out.alphanum +=
rules.alphabets.letters;</tag></item>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out.alphanum +=
rules.digits.numbers</tag></item>
        </rule>

        <rule id="alphabets">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <tag>out.letters=""</tag>
            <tag>out.firstOccurence=""</tag>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out.firstOccurence +=
rules.digits.numbers; out.letters += out.firstOccurence;</tag></item>
```

```
            <item repeat="1-">
                <one-of>
                    <item>A<tag>out.letters+='A';</tag></item>
                    <item>B<tag>out.letters+='B';</tag></item>
                    <item>C<tag>out.letters+='C';</tag></item>
                    <item>D<tag>out.letters+='D';</tag></item>
                    <item>E<tag>out.letters+='E';</tag></item>
                    <item>F<tag>out.letters+='F';</tag></item>
                    <item>G<tag>out.letters+='G';</tag></item>
                    <item>H<tag>out.letters+='H';</tag></item>
                    <item>I<tag>out.letters+='I';</tag></item>
                    <item>J<tag>out.letters+='J';</tag></item>
                    <item>K<tag>out.letters+='K';</tag></item>
                    <item>L<tag>out.letters+='L';</tag></item>
                    <item>M<tag>out.letters+='M';</tag></item>
                    <item>N<tag>out.letters+='N';</tag></item>
                    <item>O<tag>out.letters+='O';</tag></item>
                    <item>P<tag>out.letters+='P';</tag></item>
                    <item>Q<tag>out.letters+='Q';</tag></item>
                    <item>R<tag>out.letters+='R';</tag></item>
                    <item>S<tag>out.letters+='S';</tag></item>
                    <item>T<tag>out.letters+='T';</tag></item>
                    <item>U<tag>out.letters+='U';</tag></item>
                    <item>V<tag>out.letters+='V';</tag></item>
                    <item>W<tag>out.letters+='W';</tag></item>
                    <item>X<tag>out.letters+='X';</tag></item>
                    <item>Y<tag>out.letters+='Y';</tag></item>
                    <item>Z<tag>out.letters+='Z';</tag></item>
                </one-of>
            </item>
        </rule>

        <rule id="digits">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <tag>out.numbers=""</tag>
            <item repeat="1-10">
                <one-of>
                    <item>0<tag>out.numbers+=0;</tag></item>
                    <item>1<tag>out.numbers+=1;</tag></item>
                    <item>2<tag>out.numbers+=2;</tag></item>
                    <item>3<tag>out.numbers+=3;</tag></item>
                    <item>4<tag>out.numbers+=4;</tag></item>
                    <item>5<tag>out.numbers+=5;</tag></item>
                    <item>6<tag>out.numbers+=6;</tag></item>
```

```
                    <item>7<tag>out.numbers+=7;</tag></item>
                    <item>8<tag>out.numbers+=8;</tag></item>
                    <item>9<tag>out.numbers+=9;</tag></item>
                </one-of>
            </item>
        </rule>
</grammar>
```

クレジットスコア

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                             http://www.w3.org/TR/speech-grammar/grammar.xsd"
         xml:lang="en-US" version="1.0"
         root="main"
         mode="voice"
         tag-format="semantics/1.0">

         <!-- Test Cases

         Grammar will support the following inputs:

             Scenario 1:
                 Input: The number is fifteen
                 Output: 15

             Scenario 2:
                 Input: My credit score is fifteen
                 Output: 15
         -->

        <rule id="main" scope="public">
            <tag>out=""</tag>
            <one-of>
              <item repeat="1"><ruleref uri="#digits"/><tag>out+= rules.digits;</tag></
item>
              <item repeat="1"><ruleref uri="#teens"/><tag>out+= rules.teens;</tag></
item>
              <item repeat="1"><ruleref uri="#above_twenty"/><tag>out+=
 rules.above_twenty;</tag></item>
            </one-of>
```

```
        </rule>

        <rule id="text">
           <one-of>
              <item repeat="0-1">Credit score is</item>
              <item repeat="0-1">Last digits are</item>
              <item repeat="0-1">The number is</item>
              <item repeat="0-1">That's</item>
              <item repeat="0-1">It is</item>
              <item repeat="0-1">My credit score is</item>
           </one-of>
        </rule>

        <rule id="digits">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <one-of>
                <item>0<tag>out=0;</tag></item>
                <item>1<tag>out=1;</tag></item>
                <item>2<tag>out=2;</tag></item>
                <item>3<tag>out=3;</tag></item>
                <item>4<tag>out=4;</tag></item>
                <item>5<tag>out=5;</tag></item>
                <item>6<tag>out=6;</tag></item>
                <item>7<tag>out=7;</tag></item>
                <item>8<tag>out=8;</tag></item>
                <item>9<tag>out=9;</tag></item>
                <item>one<tag>out=1;</tag></item>
                <item>two<tag>out=2;</tag></item>
                <item>three<tag>out=3;</tag></item>
                <item>four<tag>out=4;</tag></item>
                <item>five<tag>out=5;</tag></item>
                <item>six<tag>out=6;</tag></item>
                <item>seven<tag>out=7;</tag></item>
                <item>eight<tag>out=8;</tag></item>
                <item>nine<tag>out=9;</tag></item>
            </one-of>
        </rule>

        <rule id="teens">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <one-of>
                <item>ten<tag>out=10;</tag></item>
                <item>eleven<tag>out=11;</tag></item>
                <item>twelve<tag>out=12;</tag></item>
```

```
                <item>thirteen<tag>out=13;</tag></item>
                <item>fourteen<tag>out=14;</tag></item>
                <item>fifteen<tag>out=15;</tag></item>
                <item>sixteen<tag>out=16;</tag></item>
                <item>seventeen<tag>out=17;</tag></item>
                <item>eighteen<tag>out=18;</tag></item>
                <item>nineteen<tag>out=19;</tag></item>
                <item>10<tag>out=10;</tag></item>
                <item>11<tag>out=11;</tag></item>
                <item>12<tag>out=12;</tag></item>
                <item>13<tag>out=13;</tag></item>
                <item>14<tag>out=14;</tag></item>
                <item>15<tag>out=15;</tag></item>
                <item>16<tag>out=16;</tag></item>
                <item>17<tag>out=17;</tag></item>
                <item>18<tag>out=18;</tag></item>
                <item>19<tag>out=19;</tag></item>
            </one-of>
        </rule>

        <rule id="above_twenty">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <one-of>
                <item>twenty<tag>out=20;</tag></item>
                <item>thirty<tag>out=30;</tag></item>
                <item>forty<tag>out=40;</tag></item>
                <item>fifty<tag>out=50;</tag></item>
                <item>sixty<tag>out=60;</tag></item>
                <item>seventy<tag>out=70;</tag></item>
                <item>eighty<tag>out=80;</tag></item>
                <item>ninety<tag>out=90;</tag></item>
                <item>20<tag>out=20;</tag></item>
                <item>30<tag>out=30;</tag></item>
                <item>40<tag>out=40;</tag></item>
                <item>50<tag>out=50;</tag></item>
                <item>60<tag>out=60;</tag></item>
                <item>70<tag>out=70;</tag></item>
                <item>80<tag>out=80;</tag></item>
                <item>90<tag>out=90;</tag></item>
            </one-of>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out += rules.digits;</
tag></item>
        </rule>
```

```
</grammar>
```

口座開設日

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                             http://www.w3.org/TR/speech-grammar/grammar.xsd"
         xml:lang="en-US" version="1.0"
         root="main"
         mode="voice"
         tag-format="semantics/1.0">

         <!-- Test Cases

         Grammar will support the following inputs:

             Scenario 1:
                 Input: I opened account on July Two Thousand and Eleven
                 Output: 07/11

             Scenario 2:
                 Input: I need account number opened on July Two Thousand and Eleven
                 Output: 07/11

         -->

        <rule id="main" scope="public">
            <tag>out=""</tag>
            <item repeat="1-10">
                <item repeat="1"><ruleref uri="#months"/><tag>out = out +
 rules.months.mon + "/";</tag></item>
                <one-of>
                    <item><ruleref uri="#thousands"/><tag>out += rules.thousands;</
tag></item>
                    <item repeat="0-1"><ruleref uri="#digits"/><tag>out +=
 rules.digits;</tag></item>
                    <item repeat="0-1"><ruleref uri="#teens"/><tag>out +=
 rules.teens;</tag></item>
                    <item repeat="0-1"><ruleref uri="#above_twenty"/><tag>out +=
 rules.above_twenty;</tag></item>
                </one-of>
```

```
            </item>
        </rule>

        <rule id="text">
            <item repeat="0-1"><ruleref uri="#hesitation"/></item>
           <one-of>
              <item repeat="0-1">I opened account on </item>
              <item repeat="0-1">I need account number opened on </item>
           </one-of>
        </rule>

        <rule id="hesitation">
            <one-of>
               <item>Hmm</item>
               <item>Mmm</item>
               <item>My</item>
            </one-of>
          </rule>
        <rule id="months">
           <item repeat="0-1"><ruleref uri="#text"/></item>
           <tag>out.mon=""</tag>
           <one-of>
               <item>january<tag>out.mon+="01";</tag></item>
               <item>february<tag>out.mon+="02";</tag></item>
               <item>march<tag>out.mon+="03";</tag></item>
               <item>april<tag>out.mon+="04";</tag></item>
               <item>may<tag>out.mon+="05";</tag></item>
               <item>june<tag>out.mon+="06";</tag></item>
               <item>july<tag>out.mon+="07";</tag></item>
               <item>august<tag>out.mon+="08";</tag></item>
               <item>september<tag>out.mon+="09";</tag></item>
               <item>october<tag>out.mon+="10";</tag></item>
               <item>november<tag>out.mon+="11";</tag></item>
               <item>december<tag>out.mon+="12";</tag></item>
               <item>jan<tag>out.mon+="01";</tag></item>
               <item>feb<tag>out.mon+="02";</tag></item>
               <item>aug<tag>out.mon+="08";</tag></item>
               <item>sept<tag>out.mon+="09";</tag></item>
               <item>oct<tag>out.mon+="10";</tag></item>
               <item>nov<tag>out.mon+="11";</tag></item>
               <item>dec<tag>out.mon+="12";</tag></item>
           </one-of>
       </rule>
```

```
        <rule id="digits">
            <one-of>
                <item>zero<tag>out=0;</tag></item>
                <item>one<tag>out=1;</tag></item>
                <item>two<tag>out=2;</tag></item>
                <item>three<tag>out=3;</tag></item>
                <item>four<tag>out=4;</tag></item>
                <item>five<tag>out=5;</tag></item>
                <item>six<tag>out=6;</tag></item>
                <item>seven<tag>out=7;</tag></item>
                <item>eight<tag>out=8;</tag></item>
                <item>nine<tag>out=9;</tag></item>
            </one-of>
        </rule>

        <rule id="teens">
            <one-of>
                <item>ten<tag>out=10;</tag></item>
                <item>eleven<tag>out=11;</tag></item>
                <item>twelve<tag>out=12;</tag></item>
                <item>thirteen<tag>out=13;</tag></item>
                <item>fourteen<tag>out=14;</tag></item>
                <item>fifteen<tag>out=15;</tag></item>
                <item>sixteen<tag>out=16;</tag></item>
                <item>seventeen<tag>out=17;</tag></item>
                <item>eighteen<tag>out=18;</tag></item>
                <item>nineteen<tag>out=19;</tag></item>
            </one-of>
        </rule>

        <rule id="thousands">
            <item>two thousand<!--<tag>out=2000;</tag>--></item>
            <item repeat="0-1">and</item>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out = rules.digits;</tag></
item>
            <item repeat="0-1"><ruleref uri="#teens"/><tag>out = rules.teens;</tag></
item>
            <item repeat="0-1"><ruleref uri="#above_twenty"/><tag>out =
 rules.above_twenty;</tag></item>
        </rule>

        <rule id="above_twenty">
            <one-of>
                <item>twenty<tag>out=20;</tag></item>
```

```
                <item>thirty<tag>out=30;</tag></item>
                <item>forty<tag>out=40;</tag></item>
                <item>fifty<tag>out=50;</tag></item>
                <item>sixty<tag>out=60;</tag></item>
                <item>seventy<tag>out=70;</tag></item>
                <item>eighty<tag>out=80;</tag></item>
                <item>ninety<tag>out=90;</tag></item>
            </one-of>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out += rules.digits;</
tag></item>
        </rule>
</grammar>
```

自動支払い日

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                             http://www.w3.org/TR/speech-grammar/grammar.xsd"
         xml:lang="en-US" version="1.0"
         root="main"
         mode="voice"
         tag-format="semantics/1.0">

         <!-- Test Cases

         Grammar will support the following inputs:

             Scenario 1:
                 Input: I want to schedule auto pay for twenty five Dollar
                 Output: $25

             Scenario 2:
                 Input: Setup automatic payments for twenty five dollars
                 Output: $25

         -->

        <rule id="main" scope="public">
            <tag>out="$"</tag>
            <one-of>
```

```
                <item><ruleref uri="#sub_hundred"/><tag>out += rules.sub_hundred.sh;</
tag></item>
                <item><ruleref uri="#subThousands"/><tag>out += rules.subThousands;</
tag></item>
            </one-of>
        </rule>

        <rule id="text">
            <item repeat="0-1"><ruleref uri="#hesitation"/></item>
           <one-of>
              <item repeat="0-1">I want to schedule auto pay for</item>
              <item repeat="0-1">Setup automatic payments for twenty five dollars</
item>
              <item repeat="0-1">Auto pay amount of</item>
              <item repeat="0-1">Set it up for</item>
           </one-of>
        </rule>

        <rule id="hesitation">
            <one-of>
               <item>Hmm</item>
               <item>Mmm</item>
               <item>My</item>
            </one-of>
          </rule>

        <rule id="digits">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <tag>out.num = 0;</tag>
            <one-of>
                <item>0<tag>out.num+=0;</tag></item>
                <item>1<tag>out.num+=1;</tag></item>
                <item>2<tag>out.num+=2;</tag></item>
                <item>3<tag>out.num+=3;</tag></item>
                <item>4<tag>out.num+=4;</tag></item>
                <item>5<tag>out.num+=5;</tag></item>
                <item>6<tag>out.num+=6;</tag></item>
                <item>7<tag>out.num+=7;</tag></item>
                <item>8<tag>out.num+=8;</tag></item>
                <item>9<tag>out.num+=9;</tag></item>
                <item>one<tag>out.num+=1;</tag></item>
                <item>two<tag>out.num+=2;</tag></item>
                <item>three<tag>out.num+=3;</tag></item>
                <item>four<tag>out.num+=4;</tag></item>
```

```
                <item>five<tag>out.num+=5;</tag></item>
                <item>six<tag>out.num+=6;</tag></item>
                <item>seven<tag>out.num+=7;</tag></item>
                <item>eight<tag>out.num+=8;</tag></item>
                <item>nine<tag>out.num+=9;</tag></item>
            </one-of>
            <item repeat="0-1"><ruleref uri="#currency"/></item>
        </rule>

        <rule id="teens">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <tag>out.teen = 0;</tag>
            <one-of>
                <item>ten<tag>out.teen+=10;</tag></item>
                <item>eleven<tag>out.teen+=11;</tag></item>
                <item>twelve<tag>out.teen+=12;</tag></item>
                <item>thirteen<tag>out.teen+=13;</tag></item>
                <item>fourteen<tag>out.teen+=14;</tag></item>
                <item>fifteen<tag>out.teen+=15;</tag></item>
                <item>sixteen<tag>out.teen+=16;</tag></item>
                <item>seventeen<tag>out.teen+=17;</tag></item>
                <item>eighteen<tag>out.teen+=18;</tag></item>
                <item>nineteen<tag>out.teen+=19;</tag></item>
            </one-of>
            <item repeat="0-1"><ruleref uri="#currency"/></item>
        </rule>

        <rule id="above_twenty">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <tag>out.tens = 0;</tag>
            <one-of>
                <item>twenty<tag>out.tens+=20;</tag></item>
                <item>thirty<tag>out.tens+=30;</tag></item>
                <item>forty<tag>out.tens+=40;</tag></item>
                <item>fifty<tag>out.tens+=50;</tag></item>
                <item>sixty<tag>out.tens+=60;</tag></item>
                <item>seventy<tag>out.tens+=70;</tag></item>
                <item>eighty<tag>out.tens+=80;</tag></item>
                <item>ninety<tag>out.tens+=90;</tag></item>
                <item>hundred<tag>out.tens+=100;</tag></item>
            </one-of>
            <item repeat="0-1"><ruleref uri="#currency"/></item>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out.tens +=
  rules.digits.num;</tag></item>
```

```
        </rule>

        <rule id="currency">
            <one-of>
                <item repeat="0-1">dollars</item>
                <item repeat="0-1">Dollars</item>
                <item repeat="0-1">dollar</item>
                <item repeat="0-1">Dollar</item>
            </one-of>
        </rule>


        <rule id="sub_hundred">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <tag>out.sh = 0;</tag>
            <one-of>
                <item><ruleref uri="#teens"/><tag>out.sh += rules.teens.teen;</tag></
item>
                <item>
                    <ruleref uri="#above_twenty"/><tag>out.sh +=
 rules.above_twenty.tens;</tag>
                </item>
                <item><ruleref uri="#digits"/><tag>out.sh += rules.digits.num;</tag></
item>
            </one-of>
         </rule>

        <rule id="subThousands">
            <ruleref uri="#sub_hundred"/><tag>out = (100 * rules.sub_hundred.sh);</tag>
            hundred
            <item repeat="0-1"><ruleref uri="#above_twenty"/><tag>out +=
 rules.above_twenty.tens;</tag></item>
            <item repeat="0-1"><ruleref uri="#teens"/><tag>out += rules.teens.teen;</
tag></item>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out += rules.digits.num;</
tag></item>
            <item repeat="0-1"><ruleref uri="#currency"/></item>
        </rule>
</grammar>
```

クレジットカードの有効期限

```
<?xml version="1.0" encoding="UTF-8" ?>
```

```
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                             http://www.w3.org/TR/speech-grammar/grammar.xsd"
         xml:lang="en-US" version="1.0"
         root="dateCardExpiration"
         mode="voice"
         tag-format="semantics/1.0">

        <rule id="dateCardExpiration" scope="public">
            <tag>out=""</tag>
            <item repeat="1"><ruleref uri="#months"/><tag>out = out + rules.months;</
tag></item>
            <item repeat="1"><ruleref uri="#year"/><tag>out += " " + rules.year.yr;</
tag></item>
        </rule>

        <!-- Test Cases

        Grammar will support the following inputs:

            Scenario 1:
                Input: My card expiration date is july eleven
                Output: 07 2011

            Scenario 2:
                Input: My card expiration date is may twenty six
                Output: 05 2026

        -->

        <rule id="text">
           <item repeat="0-1"><ruleref uri="#hesitation"/></item>
           <one-of>
              <item repeat="0-1">My card expiration date is </item>
              <item repeat="0-1">Expiration date is </item>
           </one-of>
        </rule>

        <rule id="hesitation">
          <one-of>
             <item>Hmm</item>
             <item>Mmm</item>
             <item>My</item>
```

```
          </one-of>
        </rule>

        <rule id="months">
           <item repeat="0-1"><ruleref uri="#text"/></item>
           <one-of>
               <item>january<tag>out="01";</tag></item>
               <item>february<tag>out="02";</tag></item>
               <item>march<tag>out="03";</tag></item>
               <item>april<tag>out="04";</tag></item>
               <item>may<tag>out="05";</tag></item>
               <item>june<tag>out="06";</tag></item>
               <item>july<tag>out="07";</tag></item>
               <item>august<tag>out="08";</tag></item>
               <item>september<tag>out="09";</tag></item>
               <item>october<tag>out="10";</tag></item>
               <item>november<tag>out="11";</tag></item>
               <item>december<tag>out="12";</tag></item>
               <item>jan<tag>out="01";</tag></item>
               <item>feb<tag>out="02";</tag></item>
               <item>aug<tag>out="08";</tag></item>
               <item>sept<tag>out="09";</tag></item>
               <item>oct<tag>out="10";</tag></item>
               <item>nov<tag>out="11";</tag></item>
               <item>dec<tag>out="12";</tag></item>
               <item>1<tag>out="01";</tag></item>
               <item>2<tag>out="02";</tag></item>
               <item>3<tag>out="03";</tag></item>
               <item>4<tag>out="04";</tag></item>
               <item>5<tag>out="05";</tag></item>
               <item>6<tag>out="06";</tag></item>
               <item>7<tag>out="07";</tag></item>
               <item>8<tag>out="08";</tag></item>
               <item>9<tag>out="09";</tag></item>
               <item>ten<tag>out="10";</tag></item>
               <item>eleven<tag>out="11";</tag></item>
               <item>twelve<tag>out="12";</tag></item>
           </one-of>
       </rule>

        <rule id="digits">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <one-of>
                <item>0<tag>out=0;</tag></item>
```

```
                <item>1<tag>out=1;</tag></item>
                <item>2<tag>out=2;</tag></item>
                <item>3<tag>out=3;</tag></item>
                <item>4<tag>out=4;</tag></item>
                <item>5<tag>out=5;</tag></item>
                <item>6<tag>out=6;</tag></item>
                <item>7<tag>out=7;</tag></item>
                <item>8<tag>out=8;</tag></item>
                <item>9<tag>out=9;</tag></item>
                <item>one<tag>out=1;</tag></item>
                <item>two<tag>out=2;</tag></item>
                <item>three<tag>out=3;</tag></item>
                <item>four<tag>out=4;</tag></item>
                <item>five<tag>out=5;</tag></item>
                <item>six<tag>out=6;</tag></item>
                <item>seven<tag>out=7;</tag></item>
                <item>eight<tag>out=8;</tag></item>
                <item>nine<tag>out=9;</tag></item>
            </one-of>
        </rule>

        <rule id="year">
          <tag>out.yr="20"</tag>
          <one-of>
              <item><ruleref uri="#teens"/><tag>out.yr += rules.teens;</tag></item>
              <item><ruleref uri="#above_twenty"/><tag>out.yr += rules.above_twenty;</
tag></item>
          </one-of>
        </rule>

        <rule id="teens">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <one-of>
                <item>ten<tag>out=10;</tag></item>
                <item>eleven<tag>out=11;</tag></item>
                <item>twelve<tag>out=12;</tag></item>
                <item>thirteen<tag>out=13;</tag></item>
                <item>fourteen<tag>out=14;</tag></item>
                <item>fifteen<tag>out=15;</tag></item>
                <item>sixteen<tag>out=16;</tag></item>
                <item>seventeen<tag>out=17;</tag></item>
                <item>eighteen<tag>out=18;</tag></item>
                <item>nineteen<tag>out=19;</tag></item>
                <item>10<tag>out=10;</tag></item>
```

```
                <item>11<tag>out=11;</tag></item>
                <item>12<tag>out=12;</tag></item>
                <item>13<tag>out=13;</tag></item>
                <item>14<tag>out=14;</tag></item>
                <item>15<tag>out=15;</tag></item>
                <item>16<tag>out=16;</tag></item>
                <item>17<tag>out=17;</tag></item>
                <item>18<tag>out=18;</tag></item>
                <item>19<tag>out=19;</tag></item>
            </one-of>
        </rule>

        <rule id="above_twenty">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <one-of>
                <item>twenty<tag>out=20;</tag></item>
                <item>thirty<tag>out=30;</tag></item>
                <item>forty<tag>out=40;</tag></item>
                <item>fifty<tag>out=50;</tag></item>
                <item>sixty<tag>out=60;</tag></item>
                <item>seventy<tag>out=70;</tag></item>
                <item>eighty<tag>out=80;</tag></item>
                <item>ninety<tag>out=90;</tag></item>
                <item>20<tag>out=20;</tag></item>
                <item>30<tag>out=30;</tag></item>
                <item>40<tag>out=40;</tag></item>
                <item>50<tag>out=50;</tag></item>
                <item>60<tag>out=60;</tag></item>
                <item>70<tag>out=70;</tag></item>
                <item>80<tag>out=80;</tag></item>
                <item>90<tag>out=90;</tag></item>
            </one-of>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out += rules.digits;</
tag></item>
        </rule>
</grammar>
```

明細書の日付

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
```

```
                              http://www.w3.org/TR/speech-grammar/grammar.xsd"
         xml:lang="en-US" version="1.0"
         root="main"
         mode="voice"
         tag-format="semantics/1.0">

         <!-- Test Cases

         Grammar will support the following inputs:

             Scenario 1:
                 Input: Show me statements from July Five Two Thousand and Eleven
                 Output: 07/5/11

             Scenario 2:
                 Input: Show me statements from July Sixteen Two Thousand and Eleven
                 Output: 07/16/11

             Scenario 3:
                 Input: Show me statements from July Thirty Two Thousand and Eleven
                 Output: 07/30/11
         -->

         <rule id="main" scope="public">
             <tag>out=""</tag>
             <item>
                 <item repeat="1"><ruleref uri="#months"/><tag>out = out +
 rules.months.mon + "/";</tag></item>
                 <one-of>
                   <item><ruleref uri="#digits"/><tag>out += rules.digits + "/";</
tag></item>
                   <item><ruleref uri="#teens"/><tag>out += rules.teens+ "/";</tag></
item>
                   <item><ruleref uri="#above_twenty"/><tag>out += rules.above_twenty+
 "/";</tag></item>
                 </one-of>
                 <one-of>
                     <item><ruleref uri="#thousands"/><tag>out += rules.thousands;</
tag></item>
                     <item repeat="0-1"><ruleref uri="#digits"/><tag>out +=
 rules.digits;</tag></item>
                     <item repeat="0-1"><ruleref uri="#teens"/><tag>out +=
 rules.teens;</tag></item>
```

```
                    <item repeat="0-1"><ruleref uri="#above_twenty"/><tag>out +=
rules.above_twenty;</tag></item>
                </one-of>
            </item>
        </rule>

        <rule id="text">
          <item repeat="0-1"><ruleref uri="#hesitation"/></item>
           <one-of>
              <item repeat="0-1">I want to see bank statements from </item>
              <item repeat="0-1">Show me statements from</item>
           </one-of>
        </rule>

        <rule id="hesitation">
           <one-of>
              <item>Hmm</item>
              <item>Mmm</item>
              <item>My</item>
           </one-of>
         </rule>

        <rule id="months">
           <tag>out.mon=""</tag>
        <item repeat="0-1"><ruleref uri="#text"/></item>
           <one-of>
             <item>january<tag>out.mon+="01";</tag></item>
             <item>february<tag>out.mon+="02";</tag></item>
             <item>march<tag>out.mon+="03";</tag></item>
             <item>april<tag>out.mon+="04";</tag></item>
             <item>may<tag>out.mon+="05";</tag></item>
             <item>june<tag>out.mon+="06";</tag></item>
             <item>july<tag>out.mon+="07";</tag></item>
             <item>august<tag>out.mon+="08";</tag></item>
             <item>september<tag>out.mon+="09";</tag></item>
             <item>october<tag>out.mon+="10";</tag></item>
             <item>november<tag>out.mon+="11";</tag></item>
             <item>december<tag>out.mon+="12";</tag></item>
             <item>jan<tag>out.mon+="01";</tag></item>
             <item>feb<tag>out.mon+="02";</tag></item>
             <item>aug<tag>out.mon+="08";</tag></item>
             <item>sept<tag>out.mon+="09";</tag></item>
             <item>oct<tag>out.mon+="10";</tag></item>
             <item>nov<tag>out.mon+="11";</tag></item>
```

```
             <item>dec<tag>out.mon+="12";</tag></item>
           </one-of>
        </rule>

         <rule id="digits">
             <one-of>
                 <item>zero<tag>out=0;</tag></item>
                 <item>one<tag>out=1;</tag></item>
                 <item>two<tag>out=2;</tag></item>
                 <item>three<tag>out=3;</tag></item>
                 <item>four<tag>out=4;</tag></item>
                 <item>five<tag>out=5;</tag></item>
                 <item>six<tag>out=6;</tag></item>
                 <item>seven<tag>out=7;</tag></item>
                 <item>eight<tag>out=8;</tag></item>
                 <item>nine<tag>out=9;</tag></item>
             </one-of>
         </rule>

         <rule id="teens">
             <one-of>
                 <item>ten<tag>out=10;</tag></item>
                 <item>eleven<tag>out=11;</tag></item>
                 <item>twelve<tag>out=12;</tag></item>
                 <item>thirteen<tag>out=13;</tag></item>
                 <item>fourteen<tag>out=14;</tag></item>
                 <item>fifteen<tag>out=15;</tag></item>
                 <item>sixteen<tag>out=16;</tag></item>
                 <item>seventeen<tag>out=17;</tag></item>
                 <item>eighteen<tag>out=18;</tag></item>
                 <item>nineteen<tag>out=19;</tag></item>
             </one-of>
         </rule>

         <rule id="thousands">
             <item>two thousand</item>
             <item repeat="0-1">and</item>
             <item repeat="0-1"><ruleref uri="#digits"/><tag>out = rules.digits;</
tag></item>
             <item repeat="0-1"><ruleref uri="#teens"/><tag>out = rules.teens;</tag></
item>
             <item repeat="0-1"><ruleref uri="#above_twenty"/><tag>out =
 rules.above_twenty;</tag></item>
         </rule>
```

```
         <rule id="above_twenty">
             <one-of>
                 <item>twenty<tag>out=20;</tag></item>
                 <item>thirty<tag>out=30;</tag></item>
             </one-of>
             <item repeat="0-1"><ruleref uri="#digits"/><tag>out += rules.digits;</
tag></item>
         </rule>
 </grammar>
```

取引日

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                             http://www.w3.org/TR/speech-grammar/grammar.xsd"
         xml:lang="en-US" version="1.0"
         root="main"
         mode="voice"
         tag-format="semantics/1.0">

         <!-- Test Cases

         Grammar will support the following inputs:

             Scenario 1:
                 Input: My last incorrect transaction date is july twenty three
                 Output: 07/23

             Scenario 2:
                 Input: My last incorrect transaction date is july fifteen
                 Output: 07/15

         -->

        <rule id="main" scope="public">
            <tag>out=""</tag>
            <item repeat="1-10">
                <item><ruleref uri="#months"/><tag>out= rules.months.mon + "/";</tag></
item>
                <one-of>
```

```
                    <item><ruleref uri="#digits"/><tag>out+= rules.digits;</tag></item>
                    <item><ruleref uri="#teens"/><tag>out+= rules.teens;</tag></item>
                    <item><ruleref uri="#above_twenty"/><tag>out+=
rules.above_twenty;</tag></item>
                </one-of>
            </item>
        </rule>

        <rule id="text">
            <item repeat="0-1"><ruleref uri="#hesitation"/></item>
           <one-of>
              <item repeat="0-1">My last incorrect transaction date is</item>
              <item repeat="0-1">It is</item>
           </one-of>
        </rule>
        <rule id="hesitation">
            <one-of>
               <item>Hmm</item>
               <item>Mmm</item>
               <item>My</item>
            </one-of>
          </rule>

        <rule id="months">
           <item repeat="0-1"><ruleref uri="#text"/></item>
           <tag>out.mon=""</tag>
           <one-of>
               <item>january<tag>out.mon+="01";</tag></item>
               <item>february<tag>out.mon+="02";</tag></item>
               <item>march<tag>out.mon+="03";</tag></item>
               <item>april<tag>out.mon+="04";</tag></item>
               <item>may<tag>out.mon+="05";</tag></item>
               <item>june<tag>out.mon+="06";</tag></item>
               <item>july<tag>out.mon+="07";</tag></item>
               <item>august<tag>out.mon+="08";</tag></item>
               <item>september<tag>out.mon+="09";</tag></item>
               <item>october<tag>out.mon+="10";</tag></item>
               <item>november<tag>out.mon+="11";</tag></item>
               <item>december<tag>out.mon+="12";</tag></item>
               <item>jan<tag>out.mon+="01";</tag></item>
               <item>feb<tag>out.mon+="02";</tag></item>
               <item>aug<tag>out.mon+="08";</tag></item>
               <item>sept<tag>out.mon+="09";</tag></item>
               <item>oct<tag>out.mon+="10";</tag></item>
```

```
            <item>nov<tag>out.mon+="11";</tag></item>
            <item>dec<tag>out.mon+="12";</tag></item>
        </one-of>
    </rule>

    <rule id="digits">
        <item repeat="0-1"><ruleref uri="#text"/></item>
        <one-of>
            <item>0<tag>out=0;</tag></item>
            <item>1<tag>out=1;</tag></item>
            <item>2<tag>out=2;</tag></item>
            <item>3<tag>out=3;</tag></item>
            <item>4<tag>out=4;</tag></item>
            <item>5<tag>out=5;</tag></item>
            <item>6<tag>out=6;</tag></item>
            <item>7<tag>out=7;</tag></item>
            <item>8<tag>out=8;</tag></item>
            <item>9<tag>out=9;</tag></item>
            <item>first<tag>out=01;</tag></item>
            <item>second<tag>out=02;</tag></item>
            <item>third<tag>out=03;</tag></item>
            <item>fourth<tag>out=04;</tag></item>
            <item>fifth<tag>out=05;</tag></item>
            <item>sixth<tag>out=06;</tag></item>
            <item>seventh<tag>out=07;</tag></item>
            <item>eighth<tag>out=08;</tag></item>
            <item>ninth<tag>out=09;</tag></item>
            <item>one<tag>out=1;</tag></item>
            <item>two<tag>out=2;</tag></item>
            <item>three<tag>out=3;</tag></item>
            <item>four<tag>out=4;</tag></item>
            <item>five<tag>out=5;</tag></item>
            <item>six<tag>out=6;</tag></item>
            <item>seven<tag>out=7;</tag></item>
            <item>eight<tag>out=8;</tag></item>
            <item>nine<tag>out=9;</tag></item>
        </one-of>
    </rule>

    <rule id="teens">
        <item repeat="0-1"><ruleref uri="#text"/></item>
        <one-of>
            <item>ten<tag>out=10;</tag></item>
            <item>tenth<tag>out=10;</tag></item>
```

```
                <item>eleven<tag>out=11;</tag></item>
                <item>twelve<tag>out=12;</tag></item>
                <item>thirteen<tag>out=13;</tag></item>
                <item>fourteen<tag>out=14;</tag></item>
                <item>fifteen<tag>out=15;</tag></item>
                <item>sixteen<tag>out=16;</tag></item>
                <item>seventeen<tag>out=17;</tag></item>
                <item>eighteen<tag>out=18;</tag></item>
                <item>nineteen<tag>out=19;</tag></item>
                <item>tenth<tag>out=10;</tag></item>
                <item>eleventh<tag>out=11;</tag></item>
                <item>twelveth<tag>out=12;</tag></item>
                <item>thirteenth<tag>out=13;</tag></item>
                <item>fourteenth<tag>out=14;</tag></item>
                <item>fifteenth<tag>out=15;</tag></item>
                <item>sixteenth<tag>out=16;</tag></item>
                <item>seventeenth<tag>out=17;</tag></item>
                <item>eighteenth<tag>out=18;</tag></item>
                <item>nineteenth<tag>out=19;</tag></item>
            </one-of>
        </rule>

        <rule id="above_twenty">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <one-of>
                <item>twenty<tag>out=20;</tag></item>
                <item>thirty<tag>out=30;</tag></item>
            </one-of>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out += rules.digits;</
tag></item>
        </rule>
</grammar>
```

振込金額

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                             http://www.w3.org/TR/speech-grammar/grammar.xsd"
         xml:lang="en-US" version="1.0"
         root="main"
         mode="voice"
```

```
         tag-format="semantics/1.0">

         <!-- Test Cases

         Grammar will support the following inputs:

             Scenario 1:
                 Input: I want to transfer twenty five Dollar
                 Output: $25

             Scenario 2:
                 Input: transfer twenty five dollars
                 Output: $25

         -->

        <rule id="main" scope="public">
            <tag>out="$"</tag>
            <one-of>
                <item><ruleref uri="#sub_hundred"/><tag>out += rules.sub_hundred.sh;</
tag></item>
                <item><ruleref uri="#subThousands"/><tag>out += rules.subThousands;</
tag></item>
            </one-of>
        </rule>

        <rule id="text">
            <item repeat="0-1"><ruleref uri="#hesitation"/></item>
           <one-of>
              <item repeat="0-1">I want to transfer</item>
              <item repeat="0-1">transfer</item>
              <item repeat="0-1">make a transfer for</item>
           </one-of>
        </rule>

        <rule id="hesitation">
            <one-of>
               <item>Hmm</item>
               <item>Mmm</item>
               <item>My</item>
            </one-of>
          </rule>

        <rule id="digits">
```

```
        <item repeat="0-1"><ruleref uri="#text"/></item>
        <tag>out.num = 0;</tag>
        <one-of>
            <item>0<tag>out.num+=0;</tag></item>
            <item>1<tag>out.num+=1;</tag></item>
            <item>2<tag>out.num+=2;</tag></item>
            <item>3<tag>out.num+=3;</tag></item>
            <item>4<tag>out.num+=4;</tag></item>
            <item>5<tag>out.num+=5;</tag></item>
            <item>6<tag>out.num+=6;</tag></item>
            <item>7<tag>out.num+=7;</tag></item>
            <item>8<tag>out.num+=8;</tag></item>
            <item>9<tag>out.num+=9;</tag></item>
            <item>one<tag>out.num+=1;</tag></item>
            <item>two<tag>out.num+=2;</tag></item>
            <item>three<tag>out.num+=3;</tag></item>
            <item>four<tag>out.num+=4;</tag></item>
            <item>five<tag>out.num+=5;</tag></item>
            <item>six<tag>out.num+=6;</tag></item>
            <item>seven<tag>out.num+=7;</tag></item>
            <item>eight<tag>out.num+=8;</tag></item>
            <item>nine<tag>out.num+=9;</tag></item>
        </one-of>
        <item repeat="0-1"><ruleref uri="#currency"/></item>
    </rule>

    <rule id="teens">
        <item repeat="0-1"><ruleref uri="#text"/></item>
        <tag>out.teen = 0;</tag>
        <one-of>
            <item>ten<tag>out.teen+=10;</tag></item>
            <item>eleven<tag>out.teen+=11;</tag></item>
            <item>twelve<tag>out.teen+=12;</tag></item>
            <item>thirteen<tag>out.teen+=13;</tag></item>
            <item>fourteen<tag>out.teen+=14;</tag></item>
            <item>fifteen<tag>out.teen+=15;</tag></item>
            <item>sixteen<tag>out.teen+=16;</tag></item>
            <item>seventeen<tag>out.teen+=17;</tag></item>
            <item>eighteen<tag>out.teen+=18;</tag></item>
            <item>nineteen<tag>out.teen+=19;</tag></item>
        </one-of>
        <item repeat="0-1"><ruleref uri="#currency"/></item>
    </rule>
```

```
        <rule id="above_twenty">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <tag>out.tens = 0;</tag>
            <one-of>
                <item>twenty<tag>out.tens+=20;</tag></item>
                <item>thirty<tag>out.tens+=30;</tag></item>
                <item>forty<tag>out.tens+=40;</tag></item>
                <item>fifty<tag>out.tens+=50;</tag></item>
                <item>sixty<tag>out.tens+=60;</tag></item>
                <item>seventy<tag>out.tens+=70;</tag></item>
                <item>eighty<tag>out.tens+=80;</tag></item>
                <item>ninety<tag>out.tens+=90;</tag></item>
                <item>hundred<tag>out.tens+=100;</tag></item>
            </one-of>
            <item repeat="0-1"><ruleref uri="#currency"/></item>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out.tens +=
 rules.digits.num;</tag></item>
        </rule>

        <rule id="currency">
            <one-of>
                <item repeat="0-1">dollars</item>
                <item repeat="0-1">Dollars</item>
                <item repeat="0-1">dollar</item>
                <item repeat="0-1">Dollar</item>
            </one-of>
        </rule>


        <rule id="sub_hundred">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <tag>out.sh = 0;</tag>
            <one-of>
                <item><ruleref uri="#teens"/><tag>out.sh += rules.teens.teen;</tag></
item>
                <item>
                    <ruleref uri="#above_twenty"/><tag>out.sh +=
 rules.above_twenty.tens;</tag>
                </item>
                <item><ruleref uri="#digits"/><tag>out.sh += rules.digits.num;</tag></
item>
            </one-of>
         </rule>
```

```
        <rule id="subThousands">
            <ruleref uri="#sub_hundred"/><tag>out = (100 * rules.sub_hundred.sh);</tag>
            hundred
            <item repeat="0-1"><ruleref uri="#above_twenty"/><tag>out +=
 rules.above_twenty.tens;</tag></item>
            <item repeat="0-1"><ruleref uri="#teens"/><tag>out += rules.teens.teen;</
tag></item>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out += rules.digits.num;</
tag></item>
            <item repeat="0-1"><ruleref uri="#currency"/></item>
        </rule>
</grammar>
```

社会保障番号

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                             http://www.w3.org/TR/speech-grammar/grammar.xsd"
         xml:lang="en-US" version="1.0"
         root="main"
         mode="voice"
         tag-format="semantics/1.0">

         <rule id="main" scope="public">
             <tag>out=""</tag>
             <ruleref uri="#digits"/><tag>out += rules.digits.numbers;</tag>
         </rule>

         <rule id="digits">
             <tag>out.numbers=""</tag>
             <item repeat="1-12">
                 <one-of>
                     <item>0<tag>out.numbers+=0;</tag></item>
                     <item>1<tag>out.numbers+=1;</tag></item>
                     <item>2<tag>out.numbers+=2;</tag></item>
                     <item>3<tag>out.numbers+=3;</tag></item>
                     <item>4<tag>out.numbers+=4;</tag></item>
                     <item>5<tag>out.numbers+=5;</tag></item>
                     <item>6<tag>out.numbers+=6;</tag></item>
                     <item>7<tag>out.numbers+=7;</tag></item>
                     <item>8<tag>out.numbers+=8;</tag></item>
```

```
                    <item>9<tag>out.numbers+=9;</tag></item>
                    <item>zero<tag>out.numbers+=0;</tag></item>
                    <item>one<tag>out.numbers+=1;</tag></item>
                    <item>two<tag>out.numbers+=2;</tag></item>
                    <item>three<tag>out.numbers+=3;</tag></item>
                    <item>four<tag>out.numbers+=4;</tag></item>
                    <item>five<tag>out.numbers+=5;</tag></item>
                    <item>six<tag>out.numbers+=6;</tag></item>
                    <item>seven<tag>out.numbers+=7;</tag></item>
                    <item>eight<tag>out.numbers+=8;</tag></item>
                    <item>nine<tag>out.numbers+=9;</tag></item>
                    <item>dash</item>
                </one-of>
            </item>
        </rule>
</grammar>
```

保険の文法 (ダウンロード)

保険分野では、請求番号と保険契約番号、運転免許証とナンバープレート番号、有効期限、開始日と更新日、請求額と保険金額という文法がサポートされています。

請求 ID

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                             http://www.w3.org/TR/speech-grammar/grammar.xsd"
         xml:lang="en-US" version="1.0"
         root="digits"
         mode="voice"
         tag-format="semantics/1.0">

         <!-- Test Cases

         Grammar will support the following inputs:

             Scenario 1:
                 Input: My claim number is One Five Four Two
                 Output: 1542

             Scenario 2:
                 Input: Claim number One Five Four Four
```

```
                Output: 1544

         -->

        <rule id="digits">
            <tag>out=""</tag>
            <item><ruleref uri="#singleDigit"/><tag>out += rules.singleDigit.digit;</
tag></item>
        </rule>

        <rule id="text">
            <item repeat="0-1"><ruleref uri="#hesitation"/></item>
            <one-of>
                <item repeat="0-1">My claim number is</item>
                <item repeat="0-1">Claim number</item>
                <item repeat="0-1">This is for claim</item>
            </one-of>
        </rule>

        <rule id="hesitation">
            <one-of>
               <item>Hmm</item>
               <item>Mmm</item>
               <item>My</item>
            </one-of>
          </rule>

        <rule id="singleDigit">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <tag>out.digit=""</tag>
            <item repeat="1-10">
                <one-of>
                    <item>0<tag>out.digit+=0;</tag></item>
                    <item>zero<tag>out.digit+=0;</tag></item>
                    <item>1<tag>out.digit+=1;</tag></item>
                    <item>one<tag>out.digit+=1;</tag></item>
                    <item>2<tag>out.digit+=2;</tag></item>
                    <item>two<tag>out.digit+=2;</tag></item>
                    <item>3<tag>out.digit+=3;</tag></item>
                    <item>three<tag>out.digit+=3;</tag></item>
                    <item>4<tag>out.digit+=4;</tag></item>
                    <item>four<tag>out.digit+=4;</tag></item>
                    <item>5<tag>out.digit+=5;</tag></item>
                    <item>five<tag>out.digit+=5;</tag></item>
```

```
                    <item>6<tag>out.digit+=6;</tag></item>
                    <item>six<tag>out.digit+=5;</tag></item>
                    <item>7<tag>out.digit+=7;</tag></item>
                    <item>seven<tag>out.digit+=7;</tag></item>
                    <item>8<tag>out.digit+=8;</tag></item>
                    <item>eight<tag>out.digit+=8;</tag></item>
                    <item>9<tag>out.digit+=9;</tag></item>
                    <item>nine<tag>out.digit+=9;</tag></item>
                </one-of>
            </item>
        </rule>
</grammar>
```

ポリシー ID

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                             http://www.w3.org/TR/speech-grammar/grammar.xsd"
         xml:lang="en-US" version="1.0"
         root="main"
         mode="voice"
         tag-format="semantics/1.0">


        <!-- Test Cases

        Grammar will support the following inputs:

            Scenario 1:
                Input: My policy number is A B C 1 2 3 4
                Output: ABC1234

            Scenario 2:
                Input: This is the policy number 1 2 3 4 A B C
                Output: 1234ABC

            Scenario 3:
                Input: Hmm My policy number is 1 2 3 4 A B C 1
                Output: 123ABC1
        -->
```

```
        <rule id="main" scope="public">
            <tag>out=""</tag>
            <item><ruleref uri="#alphanumeric"/><tag>out +=
rules.alphanumeric.alphanum;</tag></item>
            <item repeat="0-1"><ruleref uri="#alphabets"/><tag>out +=
rules.alphabets.letters;</tag></item>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out +=
rules.digits.numbers</tag></item>
            <item repeat="0-1"><ruleref uri="#thanks"/></item>
        </rule>

        <rule id="text">
            <item repeat="0-1"><ruleref uri="#hesitation"/></item>
            <one-of>
                <item repeat="0-1">My policy number is</item>
                <item repeat="0-1">This is the policy number</item>
                <item repeat="0-1">Policy number</item>
                <item repeat="0-1">Yes, It is</item>
                <item repeat="0-1">Yes It is</item>
                <item repeat="0-1">Yes It's</item>
                <item repeat="0-1">My policy Id is</item>
                <item repeat="0-1">This is the policy Id</item>
                <item repeat="0-1">Policy Id</item>
            </one-of>
        </rule>

        <rule id="hesitation">
          <one-of>
             <item>Hmm</item>
             <item>Mmm</item>
             <item>My</item>
          </one-of>
        </rule>

        <rule id="thanks">
            <one-of>
               <item>Thanks</item>
               <item>I think</item>
            </one-of>
          </rule>

        <rule id="alphanumeric" scope="public">
            <tag>out.alphanum=""</tag>
```

```
            <item><ruleref uri="#alphabets"/><tag>out.alphanum +=
rules.alphabets.letters;</tag></item>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out.alphanum +=
rules.digits.numbers</tag></item>
        </rule>

        <rule id="alphabets">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <tag>out.letters=""</tag>
            <tag>out.firstOccurence=""</tag>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out.firstOccurence +=
rules.digits.numbers; out.letters += out.firstOccurence;</tag></item>
            <item repeat="1-">
                <one-of>
                    <item>A<tag>out.letters+='A';</tag></item>
                    <item>B<tag>out.letters+='B';</tag></item>
                    <item>C<tag>out.letters+='C';</tag></item>
                    <item>D<tag>out.letters+='D';</tag></item>
                    <item>E<tag>out.letters+='E';</tag></item>
                    <item>F<tag>out.letters+='F';</tag></item>
                    <item>G<tag>out.letters+='G';</tag></item>
                    <item>H<tag>out.letters+='H';</tag></item>
                    <item>I<tag>out.letters+='I';</tag></item>
                    <item>J<tag>out.letters+='J';</tag></item>
                    <item>K<tag>out.letters+='K';</tag></item>
                    <item>L<tag>out.letters+='L';</tag></item>
                    <item>M<tag>out.letters+='M';</tag></item>
                    <item>N<tag>out.letters+='N';</tag></item>
                    <item>O<tag>out.letters+='O';</tag></item>
                    <item>P<tag>out.letters+='P';</tag></item>
                    <item>Q<tag>out.letters+='Q';</tag></item>
                    <item>R<tag>out.letters+='R';</tag></item>
                    <item>S<tag>out.letters+='S';</tag></item>
                    <item>T<tag>out.letters+='T';</tag></item>
                    <item>U<tag>out.letters+='U';</tag></item>
                    <item>V<tag>out.letters+='V';</tag></item>
                    <item>W<tag>out.letters+='W';</tag></item>
                    <item>X<tag>out.letters+='X';</tag></item>
                    <item>Y<tag>out.letters+='Y';</tag></item>
                    <item>Z<tag>out.letters+='Z';</tag></item>
                </one-of>
            </item>
        </rule>
```

```
        <rule id="digits">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <tag>out.numbers=""</tag>
            <item repeat="1-10">
                <one-of>
                    <item>0<tag>out.numbers+=0;</tag></item>
                    <item>1<tag>out.numbers+=1;</tag></item>
                    <item>2<tag>out.numbers+=2;</tag></item>
                    <item>3<tag>out.numbers+=3;</tag></item>
                    <item>4<tag>out.numbers+=4;</tag></item>
                    <item>5<tag>out.numbers+=5;</tag></item>
                    <item>6<tag>out.numbers+=6;</tag></item>
                    <item>7<tag>out.numbers+=7;</tag></item>
                    <item>8<tag>out.numbers+=8;</tag></item>
                    <item>9<tag>out.numbers+=9;</tag></item>
                </one-of>
            </item>
        </rule>
</grammar>
```

運転免許証番号

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                             http://www.w3.org/TR/speech-grammar/grammar.xsd"
         xml:lang="en-US" version="1.0"
         root="digits"
         mode="voice"
         tag-format="semantics/1.0">

         <!-- Test Cases

         Grammar will support the following inputs:

             Scenario 1:
                 Input: My drivers license number is One Five Four Two
                 Output: 1542

             Scenario 2:
                 Input: driver license number One Five Four Four
                 Output: 1544
```

```
         -->

        <rule id="digits">
            <tag>out=""</tag>
            <item><ruleref uri="#singleDigit"/><tag>out += rules.singleDigit.digit;</
tag></item>
        </rule>

        <rule id="text">
            <item repeat="0-1"><ruleref uri="#hesitation"/></item>
            <one-of>
                <item repeat="0-1">My drivers license number is</item>
                <item repeat="0-1">My drivers license id is</item>
                <item repeat="0-1">Driver license number</item>
            </one-of>
        </rule>

        <rule id="hesitation">
            <one-of>
               <item>Hmm</item>
               <item>Mmm</item>
               <item>My</item>
            </one-of>
          </rule>

        <rule id="singleDigit">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <tag>out.digit=""</tag>
            <item repeat="1-10">
                <one-of>
                    <item>0<tag>out.digit+=0;</tag></item>
                    <item>zero<tag>out.digit+=0;</tag></item>
                    <item>1<tag>out.digit+=1;</tag></item>
                    <item>one<tag>out.digit+=1;</tag></item>
                    <item>2<tag>out.digit+=2;</tag></item>
                    <item>two<tag>out.digit+=2;</tag></item>
                    <item>3<tag>out.digit+=3;</tag></item>
                    <item>three<tag>out.digit+=3;</tag></item>
                    <item>4<tag>out.digit+=4;</tag></item>
                    <item>four<tag>out.digit+=4;</tag></item>
                    <item>5<tag>out.digit+=5;</tag></item>
                    <item>five<tag>out.digit+=5;</tag></item>
                    <item>6<tag>out.digit+=6;</tag></item>
```

```
                    <item>six<tag>out.digit+=5;</tag></item>
                    <item>7<tag>out.digit+=7;</tag></item>
                    <item>seven<tag>out.digit+=7;</tag></item>
                    <item>8<tag>out.digit+=8;</tag></item>
                    <item>eight<tag>out.digit+=8;</tag></item>
                    <item>9<tag>out.digit+=9;</tag></item>
                    <item>nine<tag>out.digit+=9;</tag></item>
                </one-of>
            </item>
        </rule>
</grammar>
```

ナンバープレート番号

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                             http://www.w3.org/TR/speech-grammar/grammar.xsd"
         xml:lang="en-US" version="1.0"
         root="main"
         mode="voice"
         tag-format="semantics/1.0">


        <!-- Test Cases

        Grammar will support the following inputs:

            Scenario 1:
                Input: my license plate is A B C D 1 2
                Output: ABCD12

            Scenario 2:
                Input: license plate number A B C 1 2 3 4
                Output: ABC1234

            Scenario 3:
                Input: my plates say A F G K 9 8 7 6 Thanks
                Output: AFGK9876
        -->

        <rule id="main" scope="public">
```

```
            <tag>out.licenseNum=""</tag>
            <item><ruleref uri="#alphabets"/><tag>out.licenseNum +=
rules.alphabets.letters;</tag></item>
            <item repeat="0-1"><ruleref uri="#thanks"/></item>
        </rule>

        <rule id="text">
            <item repeat="0-1"><ruleref uri="#hesitation"/></item>
            <one-of>
                <item repeat="0-1">my license plate is</item>
                <item repeat="0-1">license plate number</item>
                <item repeat="0-1">my plates say</item>
            </one-of>
        </rule>

        <rule id="hesitation">
          <one-of>
             <item>Hmm</item>
             <item>Mmm</item>
             <item>My</item>
          </one-of>
        </rule>

        <rule id="thanks">
            <one-of>
               <item>Thanks</item>
               <item>I think</item>
            </one-of>
          </rule>

        <rule id="alphabets">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <tag>out.letters=""</tag>
            <tag>out.firstOccurence=""</tag>
            <item repeat="3-4">
                <one-of>
                    <item>A<tag>out.letters+='A';</tag></item>
                    <item>B<tag>out.letters+='B';</tag></item>
                    <item>C<tag>out.letters+='C';</tag></item>
                    <item>D<tag>out.letters+='D';</tag></item>
                    <item>E<tag>out.letters+='E';</tag></item>
                    <item>F<tag>out.letters+='F';</tag></item>
                    <item>G<tag>out.letters+='G';</tag></item>
                    <item>H<tag>out.letters+='H';</tag></item>
```

```
                    <item>I<tag>out.letters+='I';</tag></item>
                    <item>J<tag>out.letters+='J';</tag></item>
                    <item>K<tag>out.letters+='K';</tag></item>
                    <item>L<tag>out.letters+='L';</tag></item>
                    <item>M<tag>out.letters+='M';</tag></item>
                    <item>N<tag>out.letters+='N';</tag></item>
                    <item>O<tag>out.letters+='O';</tag></item>
                    <item>P<tag>out.letters+='P';</tag></item>
                    <item>Q<tag>out.letters+='Q';</tag></item>
                    <item>R<tag>out.letters+='R';</tag></item>
                    <item>S<tag>out.letters+='S';</tag></item>
                    <item>T<tag>out.letters+='T';</tag></item>
                    <item>U<tag>out.letters+='U';</tag></item>
                    <item>V<tag>out.letters+='V';</tag></item>
                    <item>W<tag>out.letters+='W';</tag></item>
                    <item>X<tag>out.letters+='X';</tag></item>
                    <item>Y<tag>out.letters+='Y';</tag></item>
                    <item>Z<tag>out.letters+='Z';</tag></item>
                </one-of>
            </item>
        <item repeat="0-1"><ruleref uri="#digits"/><tag>out.firstOccurence +=
 rules.digits.numbers; out.letters += out.firstOccurence;</tag></item>
        </rule>

        <rule id="digits">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <tag>out.numbers=""</tag>
            <item repeat="2-4">
                <one-of>
                    <item>0<tag>out.numbers+=0;</tag></item>
                    <item>1<tag>out.numbers+=1;</tag></item>
                    <item>2<tag>out.numbers+=2;</tag></item>
                    <item>3<tag>out.numbers+=3;</tag></item>
                    <item>4<tag>out.numbers+=4;</tag></item>
                    <item>5<tag>out.numbers+=5;</tag></item>
                    <item>6<tag>out.numbers+=6;</tag></item>
                    <item>7<tag>out.numbers+=7;</tag></item>
                    <item>8<tag>out.numbers+=8;</tag></item>
                    <item>9<tag>out.numbers+=9;</tag></item>
                </one-of>
            </item>
        </rule>
</grammar>
```

クレジットカードの有効期限

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                             http://www.w3.org/TR/speech-grammar/grammar.xsd"
         xml:lang="en-US" version="1.0"
         root="dateCardExpiration"
         mode="voice"
         tag-format="semantics/1.0">

        <rule id="dateCardExpiration" scope="public">
            <tag>out="";</tag>
            <item repeat="1"><ruleref uri="#months"/><tag>out = out + rules.months;</
tag></item>
            <item repeat="1"><ruleref uri="#year"/><tag>out += " " + rules.year.yr;</
tag></item>
            <item repeat="0-1"><ruleref uri="#thanks"/></item>
        </rule>

        <!-- Test Cases

        Grammar will support the following inputs:

            Scenario 1:
                Input: My card expiration date is july eleven
                Output: 07 2011

            Scenario 2:
                Input: My card expiration date is may twenty six
                Output: 05 2026

        -->

        <rule id="text">
           <item repeat="0-1"><ruleref uri="#hesitation"/></item>
           <one-of>
              <item repeat="0-1">My card expiration date is </item>
           </one-of>
        </rule>

        <rule id="hesitation">
          <one-of>
```

```
            <item>Hmm</item>
            <item>Mmm</item>
            <item>My</item>
        </one-of>
    </rule>

    <rule id="thanks">
        <one-of>
            <item>Thanks</item>
            <item>I think</item>
        </one-of>
    </rule>

    <rule id="months">
        <item repeat="0-1"><ruleref uri="#text"/></item>
        <one-of>
            <item>january<tag>out="01";</tag></item>
            <item>february<tag>out="02";</tag></item>
            <item>march<tag>out="03";</tag></item>
            <item>april<tag>out="04";</tag></item>
            <item>may<tag>out="05";</tag></item>
            <item>june<tag>out="06";</tag></item>
            <item>july<tag>out="07";</tag></item>
            <item>august<tag>out="08";</tag></item>
            <item>september<tag>out="09";</tag></item>
            <item>october<tag>out="10";</tag></item>
            <item>november<tag>out="11";</tag></item>
            <item>december<tag>out="12";</tag></item>
            <item>jan<tag>out="01";</tag></item>
            <item>feb<tag>out="02";</tag></item>
            <item>aug<tag>out="08";</tag></item>
            <item>sept<tag>out="09";</tag></item>
            <item>oct<tag>out="10";</tag></item>
            <item>nov<tag>out="11";</tag></item>
            <item>dec<tag>out="12";</tag></item>
            <item>1<tag>out="01";</tag></item>
            <item>2<tag>out="02";</tag></item>
            <item>3<tag>out="03";</tag></item>
            <item>4<tag>out="04";</tag></item>
            <item>5<tag>out="05";</tag></item>
            <item>6<tag>out="06";</tag></item>
            <item>7<tag>out="07";</tag></item>
            <item>8<tag>out="08";</tag></item>
            <item>9<tag>out="09";</tag></item>
```

```
                <item>ten<tag>out="10";</tag></item>
                <item>eleven<tag>out="11";</tag></item>
                <item>twelve<tag>out="12";</tag></item>
            </one-of>
        </rule>

        <rule id="digits">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <one-of>
                <item>0<tag>out=0;</tag></item>
                <item>1<tag>out=1;</tag></item>
                <item>2<tag>out=2;</tag></item>
                <item>3<tag>out=3;</tag></item>
                <item>4<tag>out=4;</tag></item>
                <item>5<tag>out=5;</tag></item>
                <item>6<tag>out=6;</tag></item>
                <item>7<tag>out=7;</tag></item>
                <item>8<tag>out=8;</tag></item>
                <item>9<tag>out=9;</tag></item>
                <item>one<tag>out=1;</tag></item>
                <item>two<tag>out=2;</tag></item>
                <item>three<tag>out=3;</tag></item>
                <item>four<tag>out=4;</tag></item>
                <item>five<tag>out=5;</tag></item>
                <item>six<tag>out=6;</tag></item>
                <item>seven<tag>out=7;</tag></item>
                <item>eight<tag>out=8;</tag></item>
                <item>nine<tag>out=9;</tag></item>
            </one-of>
        </rule>

        <rule id="year">
          <tag>out.yr="20"</tag>
          <one-of>
              <item><ruleref uri="#teens"/><tag>out.yr += rules.teens;</tag></item>
              <item><ruleref uri="#above_twenty"/><tag>out.yr += rules.above_twenty;</
tag></item>
          </one-of>
        </rule>

        <rule id="teens">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <one-of>
                <item>ten<tag>out=10;</tag></item>
```

```
                <item>eleven<tag>out=11;</tag></item>
                <item>twelve<tag>out=12;</tag></item>
                <item>thirteen<tag>out=13;</tag></item>
                <item>fourteen<tag>out=14;</tag></item>
                <item>fifteen<tag>out=15;</tag></item>
                <item>sixteen<tag>out=16;</tag></item>
                <item>seventeen<tag>out=17;</tag></item>
                <item>eighteen<tag>out=18;</tag></item>
                <item>nineteen<tag>out=19;</tag></item>
                <item>10<tag>out=10;</tag></item>
                <item>11<tag>out=11;</tag></item>
                <item>12<tag>out=12;</tag></item>
                <item>13<tag>out=13;</tag></item>
                <item>14<tag>out=14;</tag></item>
                <item>15<tag>out=15;</tag></item>
                <item>16<tag>out=16;</tag></item>
                <item>17<tag>out=17;</tag></item>
                <item>18<tag>out=18;</tag></item>
                <item>19<tag>out=19;</tag></item>
            </one-of>
        </rule>

        <rule id="above_twenty">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <one-of>
                <item>twenty<tag>out=20;</tag></item>
                <item>thirty<tag>out=30;</tag></item>
                <item>forty<tag>out=40;</tag></item>
                <item>fifty<tag>out=50;</tag></item>
                <item>sixty<tag>out=60;</tag></item>
                <item>seventy<tag>out=70;</tag></item>
                <item>eighty<tag>out=80;</tag></item>
                <item>ninety<tag>out=90;</tag></item>
                <item>20<tag>out=20;</tag></item>
                <item>30<tag>out=30;</tag></item>
                <item>40<tag>out=40;</tag></item>
                <item>50<tag>out=50;</tag></item>
                <item>60<tag>out=60;</tag></item>
                <item>70<tag>out=70;</tag></item>
                <item>80<tag>out=80;</tag></item>
                <item>90<tag>out=90;</tag></item>
            </one-of>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out += rules.digits;</
tag></item>
```

```
        </rule>
</grammar>
```

契約有効期限、日/月/年

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                             http://www.w3.org/TR/speech-grammar/grammar.xsd"
         xml:lang="en-US" version="1.0"
         root="main"
         mode="voice"
         tag-format="semantics/1.0">

         <!-- Test Cases

         Grammar will support the following inputs:

             Scenario 1:
                 Input: My policy expired on July Five Two Thousand and Eleven
                 Output: 07/5/11

             Scenario 2:
                 Input: My policy will expire on July Sixteen Two Thousand and Eleven
                 Output: 07/16/11

             Scenario 3:
                 Input: My policy expired on July Thirty Two Thousand and Eleven
                 Output: 07/30/11
         -->

         <rule id="main" scope="public">
             <tag>out=""</tag>
             <item>
                 <item repeat="1"><ruleref uri="#months"/><tag>out = out +
 rules.months.mon + "/";</tag></item>
                 <one-of>
                   <item><ruleref uri="#digits"/><tag>out += rules.digits + "/";</
tag></item>
                   <item><ruleref uri="#teens"/><tag>out += rules.teens+ "/";</tag></
item>
```

```
                    <item><ruleref uri="#above_twenty"/><tag>out += rules.above_twenty+
 "/";</tag></item>
                </one-of>
                <one-of>
                    <item><ruleref uri="#thousands"/><tag>out += rules.thousands;</
tag></item>
                    <item repeat="0-1"><ruleref uri="#digits"/><tag>out +=
 rules.digits;</tag></item>
                    <item repeat="0-1"><ruleref uri="#teens"/><tag>out +=
 rules.teens;</tag></item>
                    <item repeat="0-1"><ruleref uri="#above_twenty"/><tag>out +=
 rules.above_twenty;</tag></item>
                </one-of>
             </item>
        </rule>

        <rule id="text">
          <item repeat="0-1"><ruleref uri="#hesitation"/></item>
           <one-of>
              <item repeat="0-1">My policy expired on</item>
              <item repeat="0-1">My policy will expire on</item>
           </one-of>
        </rule>

        <rule id="hesitation">
           <one-of>
              <item>Hmm</item>
              <item>Mmm</item>
              <item>My</item>
           </one-of>
         </rule>

        <rule id="months">
           <tag>out.mon=""</tag>
    <item repeat="0-1"><ruleref uri="#text"/></item>
           <one-of>
             <item>january<tag>out.mon+="01";</tag></item>
             <item>february<tag>out.mon+="02";</tag></item>
             <item>march<tag>out.mon+="03";</tag></item>
             <item>april<tag>out.mon+="04";</tag></item>
             <item>may<tag>out.mon+="05";</tag></item>
             <item>june<tag>out.mon+="06";</tag></item>
             <item>july<tag>out.mon+="07";</tag></item>
             <item>august<tag>out.mon+="08";</tag></item>
```

```
            <item>september<tag>out.mon+="09";</tag></item>
            <item>october<tag>out.mon+="10";</tag></item>
            <item>november<tag>out.mon+="11";</tag></item>
            <item>december<tag>out.mon+="12";</tag></item>
            <item>jan<tag>out.mon+="01";</tag></item>
            <item>feb<tag>out.mon+="02";</tag></item>
            <item>aug<tag>out.mon+="08";</tag></item>
            <item>sept<tag>out.mon+="09";</tag></item>
            <item>oct<tag>out.mon+="10";</tag></item>
            <item>nov<tag>out.mon+="11";</tag></item>
            <item>dec<tag>out.mon+="12";</tag></item>
        </one-of>
    </rule>

    <rule id="digits">
        <one-of>
            <item>zero<tag>out=0;</tag></item>
            <item>one<tag>out=1;</tag></item>
            <item>two<tag>out=2;</tag></item>
            <item>three<tag>out=3;</tag></item>
            <item>four<tag>out=4;</tag></item>
            <item>five<tag>out=5;</tag></item>
            <item>six<tag>out=6;</tag></item>
            <item>seven<tag>out=7;</tag></item>
            <item>eight<tag>out=8;</tag></item>
            <item>nine<tag>out=9;</tag></item>
        </one-of>
    </rule>

    <rule id="teens">
        <one-of>
            <item>ten<tag>out=10;</tag></item>
            <item>eleven<tag>out=11;</tag></item>
            <item>twelve<tag>out=12;</tag></item>
            <item>thirteen<tag>out=13;</tag></item>
            <item>fourteen<tag>out=14;</tag></item>
            <item>fifteen<tag>out=15;</tag></item>
            <item>sixteen<tag>out=16;</tag></item>
            <item>seventeen<tag>out=17;</tag></item>
            <item>eighteen<tag>out=18;</tag></item>
            <item>nineteen<tag>out=19;</tag></item>
        </one-of>
    </rule>
```

```
        <rule id="thousands">
            <item>two thousand</item>
            <item repeat="0-1">and</item>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out = rules.digits;</
tag></item>
            <item repeat="0-1"><ruleref uri="#teens"/><tag>out = rules.teens;</tag></
item>
            <item repeat="0-1"><ruleref uri="#above_twenty"/><tag>out =
 rules.above_twenty;</tag></item>
        </rule>

        <rule id="above_twenty">
            <one-of>
                <item>twenty<tag>out=20;</tag></item>
                <item>thirty<tag>out=30;</tag></item>
            </one-of>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out += rules.digits;</
tag></item>
        </rule>
 </grammar>
```

契約更新日、月/年

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                             http://www.w3.org/TR/speech-grammar/grammar.xsd"
         xml:lang="en-US" version="1.0"
         root="main"
         mode="voice"
         tag-format="semantics/1.0">

         <!-- Test Cases

         Grammar will support the following inputs:

             Scenario 1:
                 Input: I renewed my policy on July Two Thousand and Eleven
                 Output: 07/11

             Scenario 2:
                 Input: My policy will renew on July Two Thousand and Eleven
```

```
                 Output: 07/11

         -->

        <rule id="main" scope="public">
            <tag>out=""</tag>
            <item repeat="1-10">
                <item repeat="1"><ruleref uri="#months"/><tag>out = out +
 rules.months.mon + "/";</tag></item>
                <one-of>
                    <item><ruleref uri="#thousands"/><tag>out += rules.thousands;</
tag></item>
                    <item repeat="0-1"><ruleref uri="#digits"/><tag>out +=
 rules.digits;</tag></item>
                    <item repeat="0-1"><ruleref uri="#teens"/><tag>out +=
 rules.teens;</tag></item>
                    <item repeat="0-1"><ruleref uri="#above_twenty"/><tag>out +=
 rules.above_twenty;</tag></item>
                </one-of>
            </item>
        </rule>

        <rule id="text">
            <item repeat="0-1"><ruleref uri="#hesitation"/></item>
            <one-of>
                <item repeat="0-1">My policy will renew on</item>
                <item repeat="0-1">My policy was renewed on</item>
                <item repeat="0-1">Renew policy on</item>
                <item repeat="0-1">I renewed my policy on</item>
            </one-of>
        </rule>

        <rule id="hesitation">
            <one-of>
               <item>Hmm</item>
               <item>Mmm</item>
               <item>My</item>
            </one-of>
          </rule>

        <rule id="months">
           <item repeat="0-1"><ruleref uri="#text"/></item>
           <tag>out.mon=""</tag>
           <one-of>
```

```
            <item>january<tag>out.mon+="01";</tag></item>
            <item>february<tag>out.mon+="02";</tag></item>
            <item>march<tag>out.mon+="03";</tag></item>
            <item>april<tag>out.mon+="04";</tag></item>
            <item>may<tag>out.mon+="05";</tag></item>
            <item>june<tag>out.mon+="06";</tag></item>
            <item>july<tag>out.mon+="07";</tag></item>
            <item>august<tag>out.mon+="08";</tag></item>
            <item>september<tag>out.mon+="09";</tag></item>
            <item>october<tag>out.mon+="10";</tag></item>
            <item>november<tag>out.mon+="11";</tag></item>
            <item>december<tag>out.mon+="12";</tag></item>
            <item>jan<tag>out.mon+="01";</tag></item>
            <item>feb<tag>out.mon+="02";</tag></item>
            <item>aug<tag>out.mon+="08";</tag></item>
            <item>sept<tag>out.mon+="09";</tag></item>
            <item>oct<tag>out.mon+="10";</tag></item>
            <item>nov<tag>out.mon+="11";</tag></item>
            <item>dec<tag>out.mon+="12";</tag></item>
        </one-of>
    </rule>

    <rule id="digits">
        <one-of>
            <item>zero<tag>out=0;</tag></item>
            <item>one<tag>out=1;</tag></item>
            <item>two<tag>out=2;</tag></item>
            <item>three<tag>out=3;</tag></item>
            <item>four<tag>out=4;</tag></item>
            <item>five<tag>out=5;</tag></item>
            <item>six<tag>out=6;</tag></item>
            <item>seven<tag>out=7;</tag></item>
            <item>eight<tag>out=8;</tag></item>
            <item>nine<tag>out=9;</tag></item>
        </one-of>
    </rule>

    <rule id="teens">
        <one-of>
            <item>ten<tag>out=10;</tag></item>
            <item>eleven<tag>out=11;</tag></item>
            <item>twelve<tag>out=12;</tag></item>
            <item>thirteen<tag>out=13;</tag></item>
            <item>fourteen<tag>out=14;</tag></item>
```

```
                <item>fifteen<tag>out=15;</tag></item>
                <item>sixteen<tag>out=16;</tag></item>
                <item>seventeen<tag>out=17;</tag></item>
                <item>eighteen<tag>out=18;</tag></item>
                <item>nineteen<tag>out=19;</tag></item>
            </one-of>
        </rule>

        <rule id="thousands">
            <item>two thousand<!--<tag>out=2000;</tag>--></item>
            <item repeat="0-1">and</item>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out = rules.digits;</tag></
item>
            <item repeat="0-1"><ruleref uri="#teens"/><tag>out = rules.teens;</tag></
item>
            <item repeat="0-1"><ruleref uri="#above_twenty"/><tag>out =
 rules.above_twenty;</tag></item>
        </rule>

        <rule id="above_twenty">
            <one-of>
                <item>twenty<tag>out=20;</tag></item>
                <item>thirty<tag>out=30;</tag></item>
                <item>forty<tag>out=40;</tag></item>
                <item>fifty<tag>out=50;</tag></item>
                <item>sixty<tag>out=60;</tag></item>
                <item>seventy<tag>out=70;</tag></item>
                <item>eighty<tag>out=80;</tag></item>
                <item>ninety<tag>out=90;</tag></item>
            </one-of>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out += rules.digits;</
tag></item>
        </rule>
</grammar>
```

契約開始日

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                             http://www.w3.org/TR/speech-grammar/grammar.xsd"
         xml:lang="en-US" version="1.0"
```

```
         root="main"
         mode="voice"
         tag-format="semantics/1.0">

         <!-- Test Cases

         Grammar will support the following inputs:

             Scenario 1:
                 Input: I bought my policy on july twenty three
                 Output: 07/23

             Scenario 2:
                 Input: My policy started on july fifteen
                 Output: 07/15

         -->

        <rule id="main" scope="public">
            <tag>out=""</tag>
            <item repeat="1-10">
                <item><ruleref uri="#months"/><tag>out= rules.months.mon + "/";</tag></
item>
                <one-of>
                    <item><ruleref uri="#digits"/><tag>out+= rules.digits;</tag></item>
                    <item><ruleref uri="#teens"/><tag>out+= rules.teens;</tag></item>
                    <item><ruleref uri="#above_twenty"/><tag>out+=
 rules.above_twenty;</tag></item>
                </one-of>
            </item>
        </rule>

        <rule id="text">
            <item repeat="0-1"><ruleref uri="#hesitation"/></item>
             <one-of>
                <item repeat="0-1">I bought my policy on</item>
                <item repeat="0-1">I bought policy on</item>
                <item repeat="0-1">My policy started on</item>
             </one-of>
          </rule>

          <rule id="hesitation">
             <one-of>
                <item>Hmm</item>
```

```
            <item>Mmm</item>
            <item>My</item>
          </one-of>
       </rule>

    <rule id="months">
       <item repeat="0-1"><ruleref uri="#text"/></item>
       <tag>out.mon=""</tag>
       <one-of>
           <item>january<tag>out.mon+="01";</tag></item>
           <item>february<tag>out.mon+="02";</tag></item>
           <item>march<tag>out.mon+="03";</tag></item>
           <item>april<tag>out.mon+="04";</tag></item>
           <item>may<tag>out.mon+="05";</tag></item>
           <item>june<tag>out.mon+="06";</tag></item>
           <item>july<tag>out.mon+="07";</tag></item>
           <item>august<tag>out.mon+="08";</tag></item>
           <item>september<tag>out.mon+="09";</tag></item>
           <item>october<tag>out.mon+="10";</tag></item>
           <item>november<tag>out.mon+="11";</tag></item>
           <item>december<tag>out.mon+="12";</tag></item>
           <item>jan<tag>out.mon+="01";</tag></item>
           <item>feb<tag>out.mon+="02";</tag></item>
           <item>aug<tag>out.mon+="08";</tag></item>
           <item>sept<tag>out.mon+="09";</tag></item>
           <item>oct<tag>out.mon+="10";</tag></item>
           <item>nov<tag>out.mon+="11";</tag></item>
           <item>dec<tag>out.mon+="12";</tag></item>
       </one-of>
    </rule>

    <rule id="digits">
        <item repeat="0-1"><ruleref uri="#text"/></item>
        <one-of>
            <item>0<tag>out=0;</tag></item>
            <item>1<tag>out=1;</tag></item>
            <item>2<tag>out=2;</tag></item>
            <item>3<tag>out=3;</tag></item>
            <item>4<tag>out=4;</tag></item>
            <item>5<tag>out=5;</tag></item>
            <item>6<tag>out=6;</tag></item>
            <item>7<tag>out=7;</tag></item>
            <item>8<tag>out=8;</tag></item>
            <item>9<tag>out=9;</tag></item>
```

```
            <item>first<tag>out=01;</tag></item>
            <item>second<tag>out=02;</tag></item>
            <item>third<tag>out=03;</tag></item>
            <item>fourth<tag>out=04;</tag></item>
            <item>fifth<tag>out=05;</tag></item>
            <item>sixth<tag>out=06;</tag></item>
            <item>seventh<tag>out=07;</tag></item>
            <item>eighth<tag>out=08;</tag></item>
            <item>ninth<tag>out=09;</tag></item>
            <item>one<tag>out=1;</tag></item>
            <item>two<tag>out=2;</tag></item>
            <item>three<tag>out=3;</tag></item>
            <item>four<tag>out=4;</tag></item>
            <item>five<tag>out=5;</tag></item>
            <item>six<tag>out=6;</tag></item>
            <item>seven<tag>out=7;</tag></item>
            <item>eight<tag>out=8;</tag></item>
            <item>nine<tag>out=9;</tag></item>
        </one-of>
    </rule>

    <rule id="teens">
        <item repeat="0-1"><ruleref uri="#text"/></item>
        <one-of>
            <item>ten<tag>out=10;</tag></item>
            <item>tenth<tag>out=10;</tag></item>
            <item>eleven<tag>out=11;</tag></item>
            <item>twelve<tag>out=12;</tag></item>
            <item>thirteen<tag>out=13;</tag></item>
            <item>fourteen<tag>out=14;</tag></item>
            <item>fifteen<tag>out=15;</tag></item>
            <item>sixteen<tag>out=16;</tag></item>
            <item>seventeen<tag>out=17;</tag></item>
            <item>eighteen<tag>out=18;</tag></item>
            <item>nineteen<tag>out=19;</tag></item>
            <item>tenth<tag>out=10;</tag></item>
            <item>eleventh<tag>out=11;</tag></item>
            <item>twelveth<tag>out=12;</tag></item>
            <item>thirteenth<tag>out=13;</tag></item>
            <item>fourteenth<tag>out=14;</tag></item>
            <item>fifteenth<tag>out=15;</tag></item>
            <item>sixteenth<tag>out=16;</tag></item>
            <item>seventeenth<tag>out=17;</tag></item>
            <item>eighteenth<tag>out=18;</tag></item>
```

```
                <item>nineteenth<tag>out=19;</tag></item>
            </one-of>
        </rule>

        <rule id="above_twenty">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <one-of>
                <item>twenty<tag>out=20;</tag></item>
                <item>thirty<tag>out=30;</tag></item>
            </one-of>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out += rules.digits;</
tag></item>
        </rule>
</grammar>
```

請求金額

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                             http://www.w3.org/TR/speech-grammar/grammar.xsd"
         xml:lang="en-US" version="1.0"
         root="main"
         mode="voice"
         tag-format="semantics/1.0">

         <!-- Test Cases

         Grammar will support the following inputs:

             Scenario 1:
                 Input: I want to make a claim of one hundre ten dollars
                 Output: $110

             Scenario 2:
                 Input: Requesting claim of Two hundred dollars
                 Output: $200

         -->

        <rule id="main" scope="public">
            <tag>out="$"</tag>
```

```
            <one-of>
                <item><ruleref uri="#sub_hundred"/><tag>out += rules.sub_hundred.sh;</
tag></item>
                <item><ruleref uri="#subThousands"/><tag>out += rules.subThousands;</
tag></item>
            </one-of>
            <item repeat="0-1"><ruleref uri="#thanks"/></item>
        </rule>

        <rule id="text">
           <item repeat="0-1"><ruleref uri="#hesitation"/></item>
           <one-of>
              <item repeat="0-1">I want to place a claim for</item>
              <item repeat="0-1">I want to make a claim of</item>
              <item repeat="0-1">I assess damage of</item>
              <item repeat="0-1">Requesting claim of</item>
           </one-of>
        </rule>

        <rule id="hesitation">
            <one-of>
               <item>Hmm</item>
               <item>Mmm</item>
               <item>My</item>
            </one-of>
          </rule>

          <rule id="thanks">
              <one-of>
                 <item>Thanks</item>
                 <item>I think</item>
              </one-of>
           </rule>

        <rule id="digits">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <tag>out.num = 0;</tag>
            <one-of>
                <item>0<tag>out.num+=0;</tag></item>
                <item>1<tag>out.num+=1;</tag></item>
                <item>2<tag>out.num+=2;</tag></item>
                <item>3<tag>out.num+=3;</tag></item>
                <item>4<tag>out.num+=4;</tag></item>
                <item>5<tag>out.num+=5;</tag></item>
```

```
            <item>6<tag>out.num+=6;</tag></item>
            <item>7<tag>out.num+=7;</tag></item>
            <item>8<tag>out.num+=8;</tag></item>
            <item>9<tag>out.num+=9;</tag></item>
            <item>one<tag>out.num+=1;</tag></item>
            <item>two<tag>out.num+=2;</tag></item>
            <item>three<tag>out.num+=3;</tag></item>
            <item>four<tag>out.num+=4;</tag></item>
            <item>five<tag>out.num+=5;</tag></item>
            <item>six<tag>out.num+=6;</tag></item>
            <item>seven<tag>out.num+=7;</tag></item>
            <item>eight<tag>out.num+=8;</tag></item>
            <item>nine<tag>out.num+=9;</tag></item>
        </one-of>
        <item repeat="0-1"><ruleref uri="#currency"/></item>
    </rule>

    <rule id="teens">
        <item repeat="0-1"><ruleref uri="#text"/></item>
        <tag>out.teen = 0;</tag>
        <one-of>
            <item>ten<tag>out.teen+=10;</tag></item>
            <item>eleven<tag>out.teen+=11;</tag></item>
            <item>twelve<tag>out.teen+=12;</tag></item>
            <item>thirteen<tag>out.teen+=13;</tag></item>
            <item>fourteen<tag>out.teen+=14;</tag></item>
            <item>fifteen<tag>out.teen+=15;</tag></item>
            <item>sixteen<tag>out.teen+=16;</tag></item>
            <item>seventeen<tag>out.teen+=17;</tag></item>
            <item>eighteen<tag>out.teen+=18;</tag></item>
            <item>nineteen<tag>out.teen+=19;</tag></item>
        </one-of>
        <item repeat="0-1"><ruleref uri="#currency"/></item>
    </rule>

    <rule id="above_twenty">
        <item repeat="0-1"><ruleref uri="#text"/></item>
        <tag>out.tens = 0;</tag>
        <one-of>
            <item>twenty<tag>out.tens+=20;</tag></item>
            <item>thirty<tag>out.tens+=30;</tag></item>
            <item>forty<tag>out.tens+=40;</tag></item>
            <item>fifty<tag>out.tens+=50;</tag></item>
            <item>sixty<tag>out.tens+=60;</tag></item>
```

```
                <item>seventy<tag>out.tens+=70;</tag></item>
                <item>eighty<tag>out.tens+=80;</tag></item>
                <item>ninety<tag>out.tens+=90;</tag></item>
                <item>hundred<tag>out.tens+=100;</tag></item>
            </one-of>
            <item repeat="0-1"><ruleref uri="#currency"/></item>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out.tens +=
 rules.digits.num;</tag></item>
        </rule>

        <rule id="currency">
            <one-of>
                <item repeat="0-1">dollars</item>
                <item repeat="0-1">Dollars</item>
                <item repeat="0-1">dollar</item>
                <item repeat="0-1">Dollar</item>
            </one-of>
        </rule>


        <rule id="sub_hundred">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <tag>out.sh = 0;</tag>
            <one-of>
                <item><ruleref uri="#teens"/><tag>out.sh += rules.teens.teen;</tag></
item>
                <item>
                    <ruleref uri="#above_twenty"/><tag>out.sh +=
 rules.above_twenty.tens;</tag>
                </item>
                <item><ruleref uri="#digits"/><tag>out.sh += rules.digits.num;</tag></
item>
            </one-of>
         </rule>

        <rule id="subThousands">
            <ruleref uri="#sub_hundred"/><tag>out = (100 * rules.sub_hundred.sh);</tag>
            hundred
            <item repeat="0-1"><ruleref uri="#above_twenty"/><tag>out +=
 rules.above_twenty.tens;</tag></item>
            <item repeat="0-1"><ruleref uri="#teens"/><tag>out += rules.teens.teen;</
tag></item>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out += rules.digits.num;</
tag></item>
```

```
            <item repeat="0-1"><ruleref uri="#currency"/></item>
        </rule>
</grammar>
```

保険料

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                             http://www.w3.org/TR/speech-grammar/grammar.xsd"
         xml:lang="en-US" version="1.0"
         root="main"
         mode="voice"
         tag-format="semantics/1.0">

         <!-- Test Cases

         Grammar will support the following inputs:

             Premium amounts
             Scenario 1:
                 Input: The premium for one hundre ten dollars
                 Output: $110

             Scenario 2:
                 Input: RPremium amount of Two hundred dollars
                 Output: $200

         -->

        <rule id="main" scope="public">
            <tag>out="$"</tag>
            <one-of>
                <item><ruleref uri="#sub_hundred"/><tag>out += rules.sub_hundred.sh;</
tag></item>
                <item><ruleref uri="#subThousands"/><tag>out += rules.subThousands;</
tag></item>
            </one-of>
            <item repeat="0-1"><ruleref uri="#thanks"/></item>
        </rule>

        <rule id="text">
```

```
            <item repeat="0-1"><ruleref uri="#hesitation"/></item>
            <one-of>
               <item repeat="0-1">A premium of</item>
               <item repeat="0-1">Premium amount of</item>
               <item repeat="0-1">The premium for</item>
               <item repeat="0-1">Insurance premium for</item>
            </one-of>
        </rule>

        <rule id="hesitation">
            <one-of>
               <item>Hmm</item>
               <item>Mmm</item>
               <item>My</item>
            </one-of>
          </rule>

          <rule id="thanks">
              <one-of>
                 <item>Thanks</item>
                 <item>I think</item>
              </one-of>
            </rule>

        <rule id="digits">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <tag>out.num = 0;</tag>
            <one-of>
                <item>0<tag>out.num+=0;</tag></item>
                <item>1<tag>out.num+=1;</tag></item>
                <item>2<tag>out.num+=2;</tag></item>
                <item>3<tag>out.num+=3;</tag></item>
                <item>4<tag>out.num+=4;</tag></item>
                <item>5<tag>out.num+=5;</tag></item>
                <item>6<tag>out.num+=6;</tag></item>
                <item>7<tag>out.num+=7;</tag></item>
                <item>8<tag>out.num+=8;</tag></item>
                <item>9<tag>out.num+=9;</tag></item>
                <item>one<tag>out.num+=1;</tag></item>
                <item>two<tag>out.num+=2;</tag></item>
                <item>three<tag>out.num+=3;</tag></item>
                <item>four<tag>out.num+=4;</tag></item>
                <item>five<tag>out.num+=5;</tag></item>
                <item>six<tag>out.num+=6;</tag></item>
```

```
                <item>seven<tag>out.num+=7;</tag></item>
                <item>eight<tag>out.num+=8;</tag></item>
                <item>nine<tag>out.num+=9;</tag></item>
            </one-of>
            <item repeat="0-1"><ruleref uri="#currency"/></item>
        </rule>

        <rule id="teens">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <tag>out.teen = 0;</tag>
            <one-of>
                <item>ten<tag>out.teen+=10;</tag></item>
                <item>eleven<tag>out.teen+=11;</tag></item>
                <item>twelve<tag>out.teen+=12;</tag></item>
                <item>thirteen<tag>out.teen+=13;</tag></item>
                <item>fourteen<tag>out.teen+=14;</tag></item>
                <item>fifteen<tag>out.teen+=15;</tag></item>
                <item>sixteen<tag>out.teen+=16;</tag></item>
                <item>seventeen<tag>out.teen+=17;</tag></item>
                <item>eighteen<tag>out.teen+=18;</tag></item>
                <item>nineteen<tag>out.teen+=19;</tag></item>
            </one-of>
            <item repeat="0-1"><ruleref uri="#currency"/></item>
        </rule>

        <rule id="above_twenty">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <tag>out.tens = 0;</tag>
            <one-of>
                <item>twenty<tag>out.tens+=20;</tag></item>
                <item>thirty<tag>out.tens+=30;</tag></item>
                <item>forty<tag>out.tens+=40;</tag></item>
                <item>fifty<tag>out.tens+=50;</tag></item>
                <item>sixty<tag>out.tens+=60;</tag></item>
                <item>seventy<tag>out.tens+=70;</tag></item>
                <item>eighty<tag>out.tens+=80;</tag></item>
                <item>ninety<tag>out.tens+=90;</tag></item>
                <item>hundred<tag>out.tens+=100;</tag></item>
            </one-of>
            <item repeat="0-1"><ruleref uri="#currency"/></item>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out.tens +=
rules.digits.num;</tag></item>
        </rule>
```

```
        <rule id="currency">
            <one-of>
                <item repeat="0-1">dollars</item>
                <item repeat="0-1">Dollars</item>
                <item repeat="0-1">dollar</item>
                <item repeat="0-1">Dollar</item>
            </one-of>
        </rule>


        <rule id="sub_hundred">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <tag>out.sh = 0;</tag>
            <one-of>
                <item><ruleref uri="#teens"/><tag>out.sh += rules.teens.teen;</tag></
item>
                <item>
                    <ruleref uri="#above_twenty"/><tag>out.sh +=
 rules.above_twenty.tens;</tag>
                </item>
                <item><ruleref uri="#digits"/><tag>out.sh += rules.digits.num;</tag></
item>
            </one-of>
         </rule>

        <rule id="subThousands">
            <ruleref uri="#sub_hundred"/><tag>out = (100 * rules.sub_hundred.sh);</tag>
            hundred
            <item repeat="0-1"><ruleref uri="#above_twenty"/><tag>out +=
 rules.above_twenty.tens;</tag></item>
            <item repeat="0-1"><ruleref uri="#teens"/><tag>out += rules.teens.teen;</
tag></item>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out += rules.digits.num;</
tag></item>
            <item repeat="0-1"><ruleref uri="#currency"/></item>
        </rule>
</grammar>
```

契約量

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                             http://www.w3.org/TR/speech-grammar/grammar.xsd"
         xml:lang="en-US" version="1.0"
         root="main"
         mode="voice"
         tag-format="semantics/1.0">

         <!-- Test Cases

         Grammar will support the following inputs:

             Scenario 1:
                 Input: The number is one
                 Output: 1

             Scenario 2:
                 Input: I want policy for ten
                 Output: 10

         -->

        <rule id="main" scope="public">
            <tag>out=""</tag>
            <one-of>
              <item repeat="1"><ruleref uri="#digits"/><tag>out+= rules.digits;</tag></
item>
              <item repeat="1"><ruleref uri="#teens"/><tag>out+= rules.teens;</tag></
item>
              <item repeat="1"><ruleref uri="#above_twenty"/><tag>out+=
 rules.above_twenty;</tag></item>
            </one-of>
            <item repeat="0-1"><ruleref uri="#thanks"/></item>
        </rule>

        <rule id="text">
           <one-of>
              <item repeat="0-1">I want policy for</item>
              <item repeat="0-1">I want to order policy for</item>
              <item repeat="0-1">The number is</item>
           </one-of>
        </rule>

        <rule id="thanks">
            <one-of>
```

```
            <item>Thanks</item>
            <item>I think</item>
        </one-of>
      </rule>

    <rule id="digits">
        <item repeat="0-1"><ruleref uri="#text"/></item>
        <one-of>
            <item>0<tag>out=0;</tag></item>
            <item>1<tag>out=1;</tag></item>
            <item>2<tag>out=2;</tag></item>
            <item>3<tag>out=3;</tag></item>
            <item>4<tag>out=4;</tag></item>
            <item>5<tag>out=5;</tag></item>
            <item>6<tag>out=6;</tag></item>
            <item>7<tag>out=7;</tag></item>
            <item>8<tag>out=8;</tag></item>
            <item>9<tag>out=9;</tag></item>
            <item>one<tag>out=1;</tag></item>
            <item>two<tag>out=2;</tag></item>
            <item>three<tag>out=3;</tag></item>
            <item>four<tag>out=4;</tag></item>
            <item>five<tag>out=5;</tag></item>
            <item>six<tag>out=6;</tag></item>
            <item>seven<tag>out=7;</tag></item>
            <item>eight<tag>out=8;</tag></item>
            <item>nine<tag>out=9;</tag></item>
        </one-of>
    </rule>

    <rule id="teens">
        <item repeat="0-1"><ruleref uri="#text"/></item>
        <one-of>
            <item>ten<tag>out=10;</tag></item>
            <item>eleven<tag>out=11;</tag></item>
            <item>twelve<tag>out=12;</tag></item>
            <item>thirteen<tag>out=13;</tag></item>
            <item>fourteen<tag>out=14;</tag></item>
            <item>fifteen<tag>out=15;</tag></item>
            <item>sixteen<tag>out=16;</tag></item>
            <item>seventeen<tag>out=17;</tag></item>
            <item>eighteen<tag>out=18;</tag></item>
            <item>nineteen<tag>out=19;</tag></item>
            <item>10<tag>out=10;</tag></item>
```

```
                <item>11<tag>out=11;</tag></item>
                <item>12<tag>out=12;</tag></item>
                <item>13<tag>out=13;</tag></item>
                <item>14<tag>out=14;</tag></item>
                <item>15<tag>out=15;</tag></item>
                <item>16<tag>out=16;</tag></item>
                <item>17<tag>out=17;</tag></item>
                <item>18<tag>out=18;</tag></item>
                <item>19<tag>out=19;</tag></item>
            </one-of>
        </rule>

        <rule id="above_twenty">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <one-of>
                <item>twenty<tag>out=20;</tag></item>
                <item>thirty<tag>out=30;</tag></item>
                <item>forty<tag>out=40;</tag></item>
                <item>fifty<tag>out=50;</tag></item>
                <item>sixty<tag>out=60;</tag></item>
                <item>seventy<tag>out=70;</tag></item>
                <item>eighty<tag>out=80;</tag></item>
                <item>ninety<tag>out=90;</tag></item>
                <item>20<tag>out=20;</tag></item>
                <item>30<tag>out=30;</tag></item>
                <item>40<tag>out=40;</tag></item>
                <item>50<tag>out=50;</tag></item>
                <item>60<tag>out=60;</tag></item>
                <item>70<tag>out=70;</tag></item>
                <item>80<tag>out=80;</tag></item>
                <item>90<tag>out=90;</tag></item>
            </one-of>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out += rules.digits;</
tag></item>
        </rule>

</grammar>
```

通信用文法 (ダウンロード)

通信業界では、電話番号、シリアル番号、SIM 番号、米国郵便番号、クレジットカード有効期限、プラン開始、更新日、有効期限、サービス開始日、機器数量、請求金額などの文法がサポートされています。

Phone number (電話番号)

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                             http://www.w3.org/TR/speech-grammar/grammar.xsd"
         xml:lang="en-US" version="1.0"
         root="digits"
         mode="voice"
         tag-format="semantics/1.0">

         <!-- Test Cases

         Grammar will support 10-12 digits number and here are couple of examples of
 valid inputs:

             Scenario 1:
                 Input: Mmm My phone number is two zero one two five two six seven
 eight five
                 Output: 2012526785

             Scenario 2:
                 Input: My phone number is two zero one two five two six seven eight
 five
                 Output: 2012526785

          -->

        <rule id="digits">
            <tag>out="";</tag>
            <item><ruleref uri="#singleDigit"/><tag>out += rules.singleDigit.digit;</
tag></item>
        </rule>

        <rule id="text">
            <item repeat="0-1"><ruleref uri="#hesitation"/></item>
           <one-of>
              <item repeat="0-1">My phone number is</item>
              <item repeat="0-1">Phone number is</item>
              <item repeat="0-1">It is</item>
              <item repeat="0-1">Yes, it's</item>
              <item repeat="0-1">Yes, it is</item>
              <item repeat="0-1">Yes it is</item>
```

```
           </one-of>
        </rule>

        <rule id="hesitation">
            <one-of>
               <item>Hmm</item>
               <item>Mmm</item>
               <item>My</item>
            </one-of>
          </rule>

        <rule id="singleDigit">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <tag>out.digit=""</tag>
            <item repeat="10-12">
                <one-of>
                    <item>0<tag>out.digit+=0;</tag></item>
                    <item>zero<tag>out.digit+=0;</tag></item>
                    <item>1<tag>out.digit+=1;</tag></item>
                    <item>one<tag>out.digit+=1;</tag></item>
                    <item>2<tag>out.digit+=2;</tag></item>
                    <item>two<tag>out.digit+=2;</tag></item>
                    <item>3<tag>out.digit+=3;</tag></item>
                    <item>three<tag>out.digit+=3;</tag></item>
                    <item>4<tag>out.digit+=4;</tag></item>
                    <item>four<tag>out.digit+=4;</tag></item>
                    <item>5<tag>out.digit+=5;</tag></item>
                    <item>five<tag>out.digit+=5;</tag></item>
                    <item>6<tag>out.digit+=6;</tag></item>
                    <item>six<tag>out.digit+=5;</tag></item>
                    <item>7<tag>out.digit+=7;</tag></item>
                    <item>seven<tag>out.digit+=7;</tag></item>
                    <item>8<tag>out.digit+=8;</tag></item>
                    <item>eight<tag>out.digit+=8;</tag></item>
                    <item>9<tag>out.digit+=9;</tag></item>
                    <item>nine<tag>out.digit+=9;</tag></item>
                </one-of>
            </item>
        </rule>
</grammar>
```

シリアル番号

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                             http://www.w3.org/TR/speech-grammar/grammar.xsd"
         xml:lang="en-US" version="1.0"
         root="digits"
         mode="voice"
         tag-format="semantics/1.0">

         <!-- Test Cases

         Grammar will support the following inputs:

             Scenario 1:
                 Input: My serial number is 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6
                 Output: 123456789123456

             Scenario 2:
                 Input: Device Serial number 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6
                 Output: 123456789123456

         -->

        <rule id="digits">
            <tag>out="";</tag>
            <item><ruleref uri="#singleDigit"/><tag>out += rules.singleDigit.digit;</
tag></item>
        </rule>

        <rule id="text">
            <item repeat="0-1"><ruleref uri="#hesitation"/></item>
            <one-of>
                <item repeat="0-1">My serial number is</item>
                <item repeat="0-1">Device Serial number</item>
                <item repeat="0-1">The number is</item>
                <item repeat="0-1">The IMEI number is</item>
            </one-of>
        </rule>

        <rule id="hesitation">
            <one-of>
```

```
            <item>Hmm</item>
            <item>Mmm</item>
            <item>My</item>
          </one-of>
        </rule>

      <rule id="singleDigit">
          <item repeat="0-1"><ruleref uri="#text"/></item>
          <tag>out.digit=""</tag>
          <item repeat="15">
              <one-of>
                  <item>0<tag>out.digit+=0;</tag></item>
                  <item>zero<tag>out.digit+=0;</tag></item>
                  <item>1<tag>out.digit+=1;</tag></item>
                  <item>one<tag>out.digit+=1;</tag></item>
                  <item>2<tag>out.digit+=2;</tag></item>
                  <item>two<tag>out.digit+=2;</tag></item>
                  <item>3<tag>out.digit+=3;</tag></item>
                  <item>three<tag>out.digit+=3;</tag></item>
                  <item>4<tag>out.digit+=4;</tag></item>
                  <item>four<tag>out.digit+=4;</tag></item>
                  <item>5<tag>out.digit+=5;</tag></item>
                  <item>five<tag>out.digit+=5;</tag></item>
                  <item>6<tag>out.digit+=6;</tag></item>
                  <item>six<tag>out.digit+=5;</tag></item>
                  <item>7<tag>out.digit+=7;</tag></item>
                  <item>seven<tag>out.digit+=7;</tag></item>
                  <item>8<tag>out.digit+=8;</tag></item>
                  <item>eight<tag>out.digit+=8;</tag></item>
                  <item>9<tag>out.digit+=9;</tag></item>
                  <item>nine<tag>out.digit+=9;</tag></item>
              </one-of>
          </item>
      </rule>
</grammar>
```

SIM 番号

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                             http://www.w3.org/TR/speech-grammar/grammar.xsd"
```

```
         xml:lang="en-US" version="1.0"
         root="main"
         mode="voice"
         tag-format="semantics/1.0">


        <!-- Test Cases

        Grammar will support the following inputs:

            Scenario 1:
                Input: My SIM number is A B C 1 2 3 4
                Output: ABC1234

            Scenario 2:
                Input: My SIM number is 1 2 3 4 A B C
                Output: 1234ABC

            Scenario 3:
                Input: My SIM number is 1 2 3 4 A B C 1
                Output: 123ABC1
        -->

        <rule id="main" scope="public">
            <tag>out=""</tag>
            <item><ruleref uri="#alphanumeric"/><tag>out +=
rules.alphanumeric.alphanum;</tag></item>
            <item repeat="0-1"><ruleref uri="#alphabets"/><tag>out +=
rules.alphabets.letters;</tag></item>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out +=
rules.digits.numbers</tag></item>
        </rule>

        <rule id="text">
            <item repeat="0-1"><ruleref uri="#hesitation"/></item>
            <one-of>
                <item repeat="0-1">My SIM number is</item>
                <item repeat="0-1">SIM number is</item>
            </one-of>
        </rule>

        <rule id="hesitation">
          <one-of>
             <item>Hmm</item>
```

```
            <item>Mmm</item>
            <item>My</item>
          </one-of>
        </rule>

        <rule id="alphanumeric" scope="public">
            <tag>out.alphanum=""</tag>
            <item><ruleref uri="#alphabets"/><tag>out.alphanum +=
rules.alphabets.letters;</tag></item>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out.alphanum +=
rules.digits.numbers</tag></item>
        </rule>

        <rule id="alphabets">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <tag>out.letters=""</tag>
            <tag>out.firstOccurence=""</tag>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out.firstOccurence +=
rules.digits.numbers; out.letters += out.firstOccurence;</tag></item>
            <item repeat="1-">
                <one-of>
                    <item>A<tag>out.letters+='A';</tag></item>
                    <item>B<tag>out.letters+='B';</tag></item>
                    <item>C<tag>out.letters+='C';</tag></item>
                    <item>D<tag>out.letters+='D';</tag></item>
                    <item>E<tag>out.letters+='E';</tag></item>
                    <item>F<tag>out.letters+='F';</tag></item>
                    <item>G<tag>out.letters+='G';</tag></item>
                    <item>H<tag>out.letters+='H';</tag></item>
                    <item>I<tag>out.letters+='I';</tag></item>
                    <item>J<tag>out.letters+='J';</tag></item>
                    <item>K<tag>out.letters+='K';</tag></item>
                    <item>L<tag>out.letters+='L';</tag></item>
                    <item>M<tag>out.letters+='M';</tag></item>
                    <item>N<tag>out.letters+='N';</tag></item>
                    <item>O<tag>out.letters+='O';</tag></item>
                    <item>P<tag>out.letters+='P';</tag></item>
                    <item>Q<tag>out.letters+='Q';</tag></item>
                    <item>R<tag>out.letters+='R';</tag></item>
                    <item>S<tag>out.letters+='S';</tag></item>
                    <item>T<tag>out.letters+='T';</tag></item>
                    <item>U<tag>out.letters+='U';</tag></item>
                    <item>V<tag>out.letters+='V';</tag></item>
                    <item>W<tag>out.letters+='W';</tag></item>
```

```
                    <item>X<tag>out.letters+='X';</tag></item>
                    <item>Y<tag>out.letters+='Y';</tag></item>
                    <item>Z<tag>out.letters+='Z';</tag></item>
                </one-of>
            </item>
        </rule>

        <rule id="digits">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <tag>out.numbers=""</tag>
            <item repeat="1-10">
                <one-of>
                    <item>0<tag>out.numbers+=0;</tag></item>
                    <item>1<tag>out.numbers+=1;</tag></item>
                    <item>2<tag>out.numbers+=2;</tag></item>
                    <item>3<tag>out.numbers+=3;</tag></item>
                    <item>4<tag>out.numbers+=4;</tag></item>
                    <item>5<tag>out.numbers+=5;</tag></item>
                    <item>6<tag>out.numbers+=6;</tag></item>
                    <item>7<tag>out.numbers+=7;</tag></item>
                    <item>8<tag>out.numbers+=8;</tag></item>
                    <item>9<tag>out.numbers+=9;</tag></item>
                </one-of>
            </item>
        </rule>
</grammar>
```

米国郵便番号

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                             http://www.w3.org/TR/speech-grammar/grammar.xsd"
         xml:lang="en-US" version="1.0"
         root="digits"
         mode="voice"
         tag-format="semantics/1.0">

         <!-- Test Cases

         Grammar will support 5 digits code and here are couple of examples of valid
 inputs:
```

```
             Scenario 1:
                 Input: Mmmm My zipcode is umm One Oh Nine Eight Seven
                 Output: 10987

             Scenario 2:
                 Input: My zipcode is One Oh Nine Eight Seven
                 Output: 10987

         -->

        <rule id="digits">
            <tag>out=""</tag>
            <item><ruleref uri="#singleDigit"/><tag>out += rules.singleDigit.digit;</
tag></item>
        </rule>

        <rule id="text">
            <item repeat="0-1"><ruleref uri="#hesitation"/></item>
           <one-of>
              <item repeat="0-1">My zipcode is</item>
              <item repeat="0-1">Zipcode is</item>
              <item repeat="0-1">It is</item>
           </one-of>
        </rule>

        <rule id="hesitation">
            <one-of>
               <item>Hmm</item>
               <item>Mmm</item>
               <item>My</item>
            </one-of>
          </rule>

        <rule id="singleDigit">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <tag>out.digit=""</tag>
            <item repeat="5">
                <one-of>
                    <item>0<tag>out.digit+=0;</tag></item>
                    <item>zero<tag>out.digit+=0;</tag></item>
                    <item>Oh<tag>out.digit+=0;</tag></item>
                    <item>1<tag>out.digit+=1;</tag></item>
                    <item>one<tag>out.digit+=1;</tag></item>
```

```
                    <item>2<tag>out.digit+=2;</tag></item>
                    <item>two<tag>out.digit+=2;</tag></item>
                    <item>3<tag>out.digit+=3;</tag></item>
                    <item>three<tag>out.digit+=3;</tag></item>
                    <item>4<tag>out.digit+=4;</tag></item>
                    <item>four<tag>out.digit+=4;</tag></item>
                    <item>5<tag>out.digit+=5;</tag></item>
                    <item>five<tag>out.digit+=5;</tag></item>
                    <item>6<tag>out.digit+=6;</tag></item>
                    <item>six<tag>out.digit+=5;</tag></item>
                    <item>7<tag>out.digit+=7;</tag></item>
                    <item>seven<tag>out.digit+=7;</tag></item>
                    <item>8<tag>out.digit+=8;</tag></item>
                    <item>eight<tag>out.digit+=8;</tag></item>
                    <item>9<tag>out.digit+=9;</tag></item>
                    <item>nine<tag>out.digit+=9;</tag></item>
                </one-of>
            </item>
        </rule>
</grammar>
```

クレジットカードの有効期限

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                             http://www.w3.org/TR/speech-grammar/grammar.xsd"
         xml:lang="en-US" version="1.0"
         root="dateCardExpiration"
         mode="voice"
         tag-format="semantics/1.0">

        <rule id="dateCardExpiration" scope="public">
            <tag>out="";</tag>
            <item repeat="1"><ruleref uri="#months"/><tag>out = out + rules.months;</
tag></item>
            <item repeat="1"><ruleref uri="#year"/><tag>out += " " + rules.year.yr;</
tag></item>
        </rule>

        <!-- Test Cases
```

```
        Grammar will support the following inputs:

            Scenario 1:
                Input: My card expiration date is july eleven
                Output: 07 2011

            Scenario 2:
                Input: My card expiration date is may twenty six
                Output: 05 2026

        -->

        <rule id="text">
           <item repeat="0-1"><ruleref uri="#hesitation"/></item>
           <one-of>
              <item repeat="0-1">My card expiration date is </item>
           </one-of>
        </rule>

        <rule id="hesitation">
          <one-of>
             <item>Hmm</item>
             <item>Mmm</item>
             <item>My</item>
          </one-of>
        </rule>

        <rule id="months">
           <item repeat="0-1"><ruleref uri="#text"/></item>
           <one-of>
               <item>january<tag>out="01";</tag></item>
               <item>february<tag>out="02";</tag></item>
               <item>march<tag>out="03";</tag></item>
               <item>april<tag>out="04";</tag></item>
               <item>may<tag>out="05";</tag></item>
               <item>june<tag>out="06";</tag></item>
               <item>july<tag>out="07";</tag></item>
               <item>august<tag>out="08";</tag></item>
               <item>september<tag>out="09";</tag></item>
               <item>october<tag>out="10";</tag></item>
               <item>november<tag>out="11";</tag></item>
               <item>december<tag>out="12";</tag></item>
               <item>jan<tag>out="01";</tag></item>
               <item>feb<tag>out="02";</tag></item>
```

```
            <item>aug<tag>out="08";</tag></item>
            <item>sept<tag>out="09";</tag></item>
            <item>oct<tag>out="10";</tag></item>
            <item>nov<tag>out="11";</tag></item>
            <item>dec<tag>out="12";</tag></item>
            <item>1<tag>out="01";</tag></item>
            <item>2<tag>out="02";</tag></item>
            <item>3<tag>out="03";</tag></item>
            <item>4<tag>out="04";</tag></item>
            <item>5<tag>out="05";</tag></item>
            <item>6<tag>out="06";</tag></item>
            <item>7<tag>out="07";</tag></item>
            <item>8<tag>out="08";</tag></item>
            <item>9<tag>out="09";</tag></item>
            <item>ten<tag>out="10";</tag></item>
            <item>eleven<tag>out="11";</tag></item>
            <item>twelve<tag>out="12";</tag></item>
        </one-of>
    </rule>

    <rule id="digits">
        <item repeat="0-1"><ruleref uri="#text"/></item>
        <one-of>
            <item>0<tag>out=0;</tag></item>
            <item>1<tag>out=1;</tag></item>
            <item>2<tag>out=2;</tag></item>
            <item>3<tag>out=3;</tag></item>
            <item>4<tag>out=4;</tag></item>
            <item>5<tag>out=5;</tag></item>
            <item>6<tag>out=6;</tag></item>
            <item>7<tag>out=7;</tag></item>
            <item>8<tag>out=8;</tag></item>
            <item>9<tag>out=9;</tag></item>
            <item>one<tag>out=1;</tag></item>
            <item>two<tag>out=2;</tag></item>
            <item>three<tag>out=3;</tag></item>
            <item>four<tag>out=4;</tag></item>
            <item>five<tag>out=5;</tag></item>
            <item>six<tag>out=6;</tag></item>
            <item>seven<tag>out=7;</tag></item>
            <item>eight<tag>out=8;</tag></item>
            <item>nine<tag>out=9;</tag></item>
        </one-of>
    </rule>
```

```
        <rule id="year">
          <tag>out.yr="20"</tag>
          <one-of>
              <item><ruleref uri="#teens"/><tag>out.yr += rules.teens;</tag></item>
              <item><ruleref uri="#above_twenty"/><tag>out.yr += rules.above_twenty;</
tag></item>
          </one-of>
        </rule>

        <rule id="teens">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <one-of>
                <item>ten<tag>out=10;</tag></item>
                <item>eleven<tag>out=11;</tag></item>
                <item>twelve<tag>out=12;</tag></item>
                <item>thirteen<tag>out=13;</tag></item>
                <item>fourteen<tag>out=14;</tag></item>
                <item>fifteen<tag>out=15;</tag></item>
                <item>sixteen<tag>out=16;</tag></item>
                <item>seventeen<tag>out=17;</tag></item>
                <item>eighteen<tag>out=18;</tag></item>
                <item>nineteen<tag>out=19;</tag></item>
                <item>10<tag>out=10;</tag></item>
                <item>11<tag>out=11;</tag></item>
                <item>12<tag>out=12;</tag></item>
                <item>13<tag>out=13;</tag></item>
                <item>14<tag>out=14;</tag></item>
                <item>15<tag>out=15;</tag></item>
                <item>16<tag>out=16;</tag></item>
                <item>17<tag>out=17;</tag></item>
                <item>18<tag>out=18;</tag></item>
                <item>19<tag>out=19;</tag></item>
            </one-of>
        </rule>

        <rule id="above_twenty">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <one-of>
                <item>twenty<tag>out=20;</tag></item>
                <item>thirty<tag>out=30;</tag></item>
                <item>forty<tag>out=40;</tag></item>
                <item>fifty<tag>out=50;</tag></item>
                <item>sixty<tag>out=60;</tag></item>
```

```
                <item>seventy<tag>out=70;</tag></item>
                <item>eighty<tag>out=80;</tag></item>
                <item>ninety<tag>out=90;</tag></item>
                <item>20<tag>out=20;</tag></item>
                <item>30<tag>out=30;</tag></item>
                <item>40<tag>out=40;</tag></item>
                <item>50<tag>out=50;</tag></item>
                <item>60<tag>out=60;</tag></item>
                <item>70<tag>out=70;</tag></item>
                <item>80<tag>out=80;</tag></item>
                <item>90<tag>out=90;</tag></item>
            </one-of>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out += rules.digits;</
tag></item>
        </rule>
</grammar>
```

プラン有効期限、日/月/年

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                             http://www.w3.org/TR/speech-grammar/grammar.xsd"
         xml:lang="en-US" version="1.0"
         root="main"
         mode="voice"
         tag-format="semantics/1.0">

         <!-- Test Cases

         Grammar will support the following inputs:

             Scenario 1:
                 Input: My plan expires on July Five Two Thousand and Eleven
                 Output: 07/5/11

             Scenario 2:
                 Input: My plan will expire on July Sixteen Two Thousand and Eleven
                 Output: 07/16/11

             Scenario 3:
                 Input: My plan will expire on July Thirty Two Thousand and Eleven
```

```
                Output: 07/30/11
        -->

        <rule id="main" scope="public">
            <tag>out=""</tag>
            <item>
                <item repeat="1"><ruleref uri="#months"/><tag>out = out +
 rules.months.mon + "/";</tag></item>
                <one-of>
                  <item><ruleref uri="#digits"/><tag>out += rules.digits + "/";</
tag></item>
                  <item><ruleref uri="#teens"/><tag>out += rules.teens+ "/";</tag></
item>
                  <item><ruleref uri="#above_twenty"/><tag>out += rules.above_twenty+
 "/";</tag></item>
                </one-of>
                <one-of>
                    <item><ruleref uri="#thousands"/><tag>out += rules.thousands;</
tag></item>
                    <item repeat="0-1"><ruleref uri="#digits"/><tag>out +=
 rules.digits;</tag></item>
                    <item repeat="0-1"><ruleref uri="#teens"/><tag>out +=
 rules.teens;</tag></item>
                    <item repeat="0-1"><ruleref uri="#above_twenty"/><tag>out +=
 rules.above_twenty;</tag></item>
                </one-of>
            </item>
        </rule>

        <rule id="text">
          <item repeat="0-1"><ruleref uri="#hesitation"/></item>
          <one-of>
               <item repeat="0-1">My plan expires on</item>
               <item repeat="0-1">My plan expired on</item>
               <item repeat="0-1">My plan will expire on</item>
           </one-of>
        </rule>

        <rule id="hesitation">
           <one-of>
              <item>Hmm</item>
              <item>Mmm</item>
              <item>My</item>
           </one-of>
```

```
 </rule>

<rule id="months">
   <tag>out.mon=""</tag>
<item repeat="0-1"><ruleref uri="#text"/></item>

   <one-of>
     <item>january<tag>out.mon+="01";</tag></item>
     <item>february<tag>out.mon+="02";</tag></item>
     <item>march<tag>out.mon+="03";</tag></item>
     <item>april<tag>out.mon+="04";</tag></item>
     <item>may<tag>out.mon+="05";</tag></item>
     <item>june<tag>out.mon+="06";</tag></item>
     <item>july<tag>out.mon+="07";</tag></item>
     <item>august<tag>out.mon+="08";</tag></item>
     <item>september<tag>out.mon+="09";</tag></item>
     <item>october<tag>out.mon+="10";</tag></item>
     <item>november<tag>out.mon+="11";</tag></item>
     <item>december<tag>out.mon+="12";</tag></item>
     <item>jan<tag>out.mon+="01";</tag></item>
     <item>feb<tag>out.mon+="02";</tag></item>
     <item>aug<tag>out.mon+="08";</tag></item>
     <item>sept<tag>out.mon+="09";</tag></item>
     <item>oct<tag>out.mon+="10";</tag></item>
     <item>nov<tag>out.mon+="11";</tag></item>
     <item>dec<tag>out.mon+="12";</tag></item>
   </one-of>
</rule>

<rule id="digits">
    <one-of>
        <item>zero<tag>out=0;</tag></item>
        <item>one<tag>out=1;</tag></item>
        <item>two<tag>out=2;</tag></item>
        <item>three<tag>out=3;</tag></item>
        <item>four<tag>out=4;</tag></item>
        <item>five<tag>out=5;</tag></item>
        <item>six<tag>out=6;</tag></item>
        <item>seven<tag>out=7;</tag></item>
        <item>eight<tag>out=8;</tag></item>
        <item>nine<tag>out=9;</tag></item>
    </one-of>
</rule>
```

```
        <rule id="teens">
            <one-of>
                <item>ten<tag>out=10;</tag></item>
                <item>eleven<tag>out=11;</tag></item>
                <item>twelve<tag>out=12;</tag></item>
                <item>thirteen<tag>out=13;</tag></item>
                <item>fourteen<tag>out=14;</tag></item>
                <item>fifteen<tag>out=15;</tag></item>
                <item>sixteen<tag>out=16;</tag></item>
                <item>seventeen<tag>out=17;</tag></item>
                <item>eighteen<tag>out=18;</tag></item>
                <item>nineteen<tag>out=19;</tag></item>
            </one-of>
        </rule>

        <rule id="thousands">
            <item>two thousand</item>
            <item repeat="0-1">and</item>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out = rules.digits;</
tag></item>
            <item repeat="0-1"><ruleref uri="#teens"/><tag>out = rules.teens;</tag></
item>
            <item repeat="0-1"><ruleref uri="#above_twenty"/><tag>out =
 rules.above_twenty;</tag></item>
        </rule>

        <rule id="above_twenty">
            <one-of>
                <item>twenty<tag>out=20;</tag></item>
                <item>thirty<tag>out=30;</tag></item>
            </one-of>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out += rules.digits;</
tag></item>
        </rule>
 </grammar>
```

プラン更新日、月/年

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                            http://www.w3.org/TR/speech-grammar/grammar.xsd"
```

```
         xml:lang="en-US" version="1.0"
         root="main"
         mode="voice"
         tag-format="semantics/1.0">

         <!-- Test Cases

         Grammar will support the following inputs:

             Scenario 1:
                 Input: My plan will renew on July Two Thousand and Eleven
                 Output: 07/11

             Scenario 2:
                 Input: Renew plan on July Two Thousand and Eleven
                 Output: 07/11

         -->

        <rule id="main" scope="public">
            <tag>out=""</tag>
            <item repeat="1-10">
                <item repeat="1"><ruleref uri="#months"/><tag>out = out +
 rules.months.mon + "/";</tag></item>
                <one-of>
                    <item><ruleref uri="#thousands"/><tag>out += rules.thousands;</
tag></item>
                    <item repeat="0-1"><ruleref uri="#digits"/><tag>out +=
 rules.digits;</tag></item>
                    <item repeat="0-1"><ruleref uri="#teens"/><tag>out +=
 rules.teens;</tag></item>
                    <item repeat="0-1"><ruleref uri="#above_twenty"/><tag>out +=
 rules.above_twenty;</tag></item>
                </one-of>
            </item>
        </rule>

        <rule id="text">
            <item repeat="0-1"><ruleref uri="#hesitation"/></item>
            <one-of>
                <item repeat="0-1">My plan will renew on</item>
                <item repeat="0-1">My plan was renewed on</item>
                <item repeat="0-1">Renew plan on</item>
            </one-of>
```

```
        </rule>

        <rule id="hesitation">
            <one-of>
               <item>Hmm</item>
               <item>Mmm</item>
               <item>My</item>
            </one-of>
          </rule>

        <rule id="months">
           <item repeat="0-1"><ruleref uri="#text"/></item>
           <tag>out.mon=""</tag>
           <one-of>
               <item>january<tag>out.mon+="01";</tag></item>
               <item>february<tag>out.mon+="02";</tag></item>
               <item>march<tag>out.mon+="03";</tag></item>
               <item>april<tag>out.mon+="04";</tag></item>
               <item>may<tag>out.mon+="05";</tag></item>
               <item>june<tag>out.mon+="06";</tag></item>
               <item>july<tag>out.mon+="07";</tag></item>
               <item>august<tag>out.mon+="08";</tag></item>
               <item>september<tag>out.mon+="09";</tag></item>
               <item>october<tag>out.mon+="10";</tag></item>
               <item>november<tag>out.mon+="11";</tag></item>
               <item>december<tag>out.mon+="12";</tag></item>
               <item>jan<tag>out.mon+="01";</tag></item>
               <item>feb<tag>out.mon+="02";</tag></item>
               <item>aug<tag>out.mon+="08";</tag></item>
               <item>sept<tag>out.mon+="09";</tag></item>
               <item>oct<tag>out.mon+="10";</tag></item>
               <item>nov<tag>out.mon+="11";</tag></item>
               <item>dec<tag>out.mon+="12";</tag></item>
           </one-of>
        </rule>

        <rule id="digits">
            <one-of>
                <item>zero<tag>out=0;</tag></item>
                <item>one<tag>out=1;</tag></item>
                <item>two<tag>out=2;</tag></item>
                <item>three<tag>out=3;</tag></item>
                <item>four<tag>out=4;</tag></item>
                <item>five<tag>out=5;</tag></item>
```

```
                <item>six<tag>out=6;</tag></item>
                <item>seven<tag>out=7;</tag></item>
                <item>eight<tag>out=8;</tag></item>
                <item>nine<tag>out=9;</tag></item>
            </one-of>
        </rule>

        <rule id="teens">
            <one-of>
                <item>ten<tag>out=10;</tag></item>
                <item>eleven<tag>out=11;</tag></item>
                <item>twelve<tag>out=12;</tag></item>
                <item>thirteen<tag>out=13;</tag></item>
                <item>fourteen<tag>out=14;</tag></item>
                <item>fifteen<tag>out=15;</tag></item>
                <item>sixteen<tag>out=16;</tag></item>
                <item>seventeen<tag>out=17;</tag></item>
                <item>eighteen<tag>out=18;</tag></item>
                <item>nineteen<tag>out=19;</tag></item>
            </one-of>
        </rule>

        <rule id="thousands">
            <item>two thousand</item>
            <item repeat="0-1">and</item>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out = rules.digits;</tag></
item>
            <item repeat="0-1"><ruleref uri="#teens"/><tag>out = rules.teens;</tag></
item>
            <item repeat="0-1"><ruleref uri="#above_twenty"/><tag>out =
 rules.above_twenty;</tag></item>
        </rule>

        <rule id="above_twenty">
            <one-of>
                <item>twenty<tag>out=20;</tag></item>
                <item>thirty<tag>out=30;</tag></item>
                <item>forty<tag>out=40;</tag></item>
                <item>fifty<tag>out=50;</tag></item>
                <item>sixty<tag>out=60;</tag></item>
                <item>seventy<tag>out=70;</tag></item>
                <item>eighty<tag>out=80;</tag></item>
                <item>ninety<tag>out=90;</tag></item>
            </one-of>
```

```
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out += rules.digits;</
tag></item>
        </rule>
</grammar>
```

プラン開始日、月/日

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                             http://www.w3.org/TR/speech-grammar/grammar.xsd"
         xml:lang="en-US" version="1.0"
         root="main"
         mode="voice"
         tag-format="semantics/1.0">

         <!-- Test Cases

         Grammar will support the following inputs:

             Scenario 1:
                 Input: My plan will start on july twenty three
                 Output: 07/23

             Scenario 2:
                 Input: My plan will start on july fifteen
                 Output: 07/15

         -->

        <rule id="main" scope="public">
            <tag>out=""</tag>
            <item repeat="1-10">
                <item><ruleref uri="#months"/><tag>out= rules.months.mon + "/";</tag></
item>
                <one-of>
                    <item><ruleref uri="#digits"/><tag>out+= rules.digits;</tag></item>
                    <item><ruleref uri="#teens"/><tag>out+= rules.teens;</tag></item>
                    <item><ruleref uri="#above_twenty"/><tag>out+=
 rules.above_twenty;</tag></item>
                </one-of>
            </item>
```

```
        </rule>

        <rule id="text">
            <item repeat="0-1"><ruleref uri="#hesitation"/></item>
             <one-of>
                <item repeat="0-1">My plan started on</item>
                <item repeat="0-1">My plan will start on</item>
                <item repeat="0-1">I paid it on</item>
                <item repeat="0-1">I paid bill for</item>
             </one-of>
          </rule>

          <rule id="hesitation">
             <one-of>
                <item>Hmm</item>
                <item>Mmm</item>
                <item>My</item>
             </one-of>
           </rule>

        <rule id="months">
           <item repeat="0-1"><ruleref uri="#text"/></item>
           <tag>out.mon=""</tag>
           <one-of>
               <item>january<tag>out.mon+="01";</tag></item>
               <item>february<tag>out.mon+="02";</tag></item>
               <item>march<tag>out.mon+="03";</tag></item>
               <item>april<tag>out.mon+="04";</tag></item>
               <item>may<tag>out.mon+="05";</tag></item>
               <item>june<tag>out.mon+="06";</tag></item>
               <item>july<tag>out.mon+="07";</tag></item>
               <item>august<tag>out.mon+="08";</tag></item>
               <item>september<tag>out.mon+="09";</tag></item>
               <item>october<tag>out.mon+="10";</tag></item>
               <item>november<tag>out.mon+="11";</tag></item>
               <item>december<tag>out.mon+="12";</tag></item>
               <item>jan<tag>out.mon+="01";</tag></item>
               <item>feb<tag>out.mon+="02";</tag></item>
               <item>aug<tag>out.mon+="08";</tag></item>
               <item>sept<tag>out.mon+="09";</tag></item>
               <item>oct<tag>out.mon+="10";</tag></item>
               <item>nov<tag>out.mon+="11";</tag></item>
               <item>dec<tag>out.mon+="12";</tag></item>
           </one-of>
```

```
        </rule>

        <rule id="digits">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <one-of>
                <item>0<tag>out=0;</tag></item>
                <item>1<tag>out=1;</tag></item>
                <item>2<tag>out=2;</tag></item>
                <item>3<tag>out=3;</tag></item>
                <item>4<tag>out=4;</tag></item>
                <item>5<tag>out=5;</tag></item>
                <item>6<tag>out=6;</tag></item>
                <item>7<tag>out=7;</tag></item>
                <item>8<tag>out=8;</tag></item>
                <item>9<tag>out=9;</tag></item>
                <item>first<tag>out=01;</tag></item>
                <item>second<tag>out=02;</tag></item>
                <item>third<tag>out=03;</tag></item>
                <item>fourth<tag>out=04;</tag></item>
                <item>fifth<tag>out=05;</tag></item>
                <item>sixth<tag>out=06;</tag></item>
                <item>seventh<tag>out=07;</tag></item>
                <item>eighth<tag>out=08;</tag></item>
                <item>ninth<tag>out=09;</tag></item>
                <item>one<tag>out=1;</tag></item>
                <item>two<tag>out=2;</tag></item>
                <item>three<tag>out=3;</tag></item>
                <item>four<tag>out=4;</tag></item>
                <item>five<tag>out=5;</tag></item>
                <item>six<tag>out=6;</tag></item>
                <item>seven<tag>out=7;</tag></item>
                <item>eight<tag>out=8;</tag></item>
                <item>nine<tag>out=9;</tag></item>
            </one-of>
        </rule>

        <rule id="teens">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <one-of>
                <item>ten<tag>out=10;</tag></item>
                <item>tenth<tag>out=10;</tag></item>
                <item>eleven<tag>out=11;</tag></item>
                <item>twelve<tag>out=12;</tag></item>
                <item>thirteen<tag>out=13;</tag></item>
```

```
                <item>fourteen<tag>out=14;</tag></item>
                <item>fifteen<tag>out=15;</tag></item>
                <item>sixteen<tag>out=16;</tag></item>
                <item>seventeen<tag>out=17;</tag></item>
                <item>eighteen<tag>out=18;</tag></item>
                <item>nineteen<tag>out=19;</tag></item>
                <item>tenth<tag>out=10;</tag></item>
                <item>eleventh<tag>out=11;</tag></item>
                <item>twelveth<tag>out=12;</tag></item>
                <item>thirteenth<tag>out=13;</tag></item>
                <item>fourteenth<tag>out=14;</tag></item>
                <item>fifteenth<tag>out=15;</tag></item>
                <item>sixteenth<tag>out=16;</tag></item>
                <item>seventeenth<tag>out=17;</tag></item>
                <item>eighteenth<tag>out=18;</tag></item>
                <item>nineteenth<tag>out=19;</tag></item>
            </one-of>
        </rule>

        <rule id="above_twenty">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <one-of>
                <item>twenty<tag>out=20;</tag></item>
                <item>thirty<tag>out=30;</tag></item>
            </one-of>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out += rules.digits;</
tag></item>
        </rule>
</grammar>
```

サービス開始日、月/日

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                             http://www.w3.org/TR/speech-grammar/grammar.xsd"
         xml:lang="en-US" version="1.0"
         root="main"
         mode="voice"
         tag-format="semantics/1.0">

         <!-- Test Cases
```

```
         Grammar will support the following inputs:

             Scenario 1:
                 Input: My plan starts on july twenty three
                 Output: 07/23

             Scenario 2:
                 Input: I want to activate on july fifteen
                 Output: 07/15

         -->

        <rule id="main" scope="public">
            <tag>out=""</tag>
            <item repeat="1-10">
                <item><ruleref uri="#months"/><tag>out= rules.months.mon + "/";</tag></
item>
                <one-of>
                    <item><ruleref uri="#digits"/><tag>out+= rules.digits;</tag></item>
                    <item><ruleref uri="#teens"/><tag>out+= rules.teens;</tag></item>
                    <item><ruleref uri="#above_twenty"/><tag>out+=
 rules.above_twenty;</tag></item>
                </one-of>
            </item>
        </rule>

        <rule id="text">
            <item repeat="0-1"><ruleref uri="#hesitation"/></item>
             <one-of>
                <item repeat="0-1">My plan starts on</item>
                <item repeat="0-1">I want to start my plan on</item>
                <item repeat="0-1">Activation date of</item>
                <item repeat="0-1">Start activation on</item>
                <item repeat="0-1">I want to activate on</item>
                <item repeat="0-1">Activate plan starting</item>
                <item repeat="0-1">Starting</item>
                <item repeat="0-1">Start on</item>
             </one-of>
          </rule>

          <rule id="hesitation">
             <one-of>
                <item>Hmm</item>
```

```
            <item>Mmm</item>
            <item>My</item>
          </one-of>
       </rule>

    <rule id="months">
       <item repeat="0-1"><ruleref uri="#text"/></item>
       <tag>out.mon=""</tag>
       <one-of>
           <item>january<tag>out.mon+="01";</tag></item>
           <item>february<tag>out.mon+="02";</tag></item>
           <item>march<tag>out.mon+="03";</tag></item>
           <item>april<tag>out.mon+="04";</tag></item>
           <item>may<tag>out.mon+="05";</tag></item>
           <item>june<tag>out.mon+="06";</tag></item>
           <item>july<tag>out.mon+="07";</tag></item>
           <item>august<tag>out.mon+="08";</tag></item>
           <item>september<tag>out.mon+="09";</tag></item>
           <item>october<tag>out.mon+="10";</tag></item>
           <item>november<tag>out.mon+="11";</tag></item>
           <item>december<tag>out.mon+="12";</tag></item>
           <item>jan<tag>out.mon+="01";</tag></item>
           <item>feb<tag>out.mon+="02";</tag></item>
           <item>aug<tag>out.mon+="08";</tag></item>
           <item>sept<tag>out.mon+="09";</tag></item>
           <item>oct<tag>out.mon+="10";</tag></item>
           <item>nov<tag>out.mon+="11";</tag></item>
           <item>dec<tag>out.mon+="12";</tag></item>
       </one-of>
    </rule>

    <rule id="digits">
        <item repeat="0-1"><ruleref uri="#text"/></item>
        <one-of>
            <item>0<tag>out=0;</tag></item>
            <item>1<tag>out=1;</tag></item>
            <item>2<tag>out=2;</tag></item>
            <item>3<tag>out=3;</tag></item>
            <item>4<tag>out=4;</tag></item>
            <item>5<tag>out=5;</tag></item>
            <item>6<tag>out=6;</tag></item>
            <item>7<tag>out=7;</tag></item>
            <item>8<tag>out=8;</tag></item>
            <item>9<tag>out=9;</tag></item>
```

```
            <item>first<tag>out=01;</tag></item>
            <item>second<tag>out=02;</tag></item>
            <item>third<tag>out=03;</tag></item>
            <item>fourth<tag>out=04;</tag></item>
            <item>fifth<tag>out=05;</tag></item>
            <item>sixth<tag>out=06;</tag></item>
            <item>seventh<tag>out=07;</tag></item>
            <item>eighth<tag>out=08;</tag></item>
            <item>ninth<tag>out=09;</tag></item>
            <item>one<tag>out=1;</tag></item>
            <item>two<tag>out=2;</tag></item>
            <item>three<tag>out=3;</tag></item>
            <item>four<tag>out=4;</tag></item>
            <item>five<tag>out=5;</tag></item>
            <item>six<tag>out=6;</tag></item>
            <item>seven<tag>out=7;</tag></item>
            <item>eight<tag>out=8;</tag></item>
            <item>nine<tag>out=9;</tag></item>
        </one-of>
    </rule>

    <rule id="teens">
        <item repeat="0-1"><ruleref uri="#text"/></item>
        <one-of>
            <item>ten<tag>out=10;</tag></item>
            <item>tenth<tag>out=10;</tag></item>
            <item>eleven<tag>out=11;</tag></item>
            <item>twelve<tag>out=12;</tag></item>
            <item>thirteen<tag>out=13;</tag></item>
            <item>fourteen<tag>out=14;</tag></item>
            <item>fifteen<tag>out=15;</tag></item>
            <item>sixteen<tag>out=16;</tag></item>
            <item>seventeen<tag>out=17;</tag></item>
            <item>eighteen<tag>out=18;</tag></item>
            <item>nineteen<tag>out=19;</tag></item>
            <item>tenth<tag>out=10;</tag></item>
            <item>eleventh<tag>out=11;</tag></item>
            <item>twelveth<tag>out=12;</tag></item>
            <item>thirteenth<tag>out=13;</tag></item>
            <item>fourteenth<tag>out=14;</tag></item>
            <item>fifteenth<tag>out=15;</tag></item>
            <item>sixteenth<tag>out=16;</tag></item>
            <item>seventeenth<tag>out=17;</tag></item>
            <item>eighteenth<tag>out=18;</tag></item>
```

```
                <item>nineteenth<tag>out=19;</tag></item>
            </one-of>
        </rule>

        <rule id="above_twenty">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <one-of>
                <item>twenty<tag>out=20;</tag></item>
                <item>thirty<tag>out=30;</tag></item>
            </one-of>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out += rules.digits;</
tag></item>
        </rule>
</grammar>
```

機器の数量

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                             http://www.w3.org/TR/speech-grammar/grammar.xsd"
         xml:lang="en-US" version="1.0"
         root="main"
         mode="voice"
         tag-format="semantics/1.0">

         <!-- Test Cases

         Grammar will support the following inputs:

             Scenario 1:
                 Input: The number is one
                 Output: 1

             Scenario 2:
                 Input: It is ten
                 Output: 10

         -->

        <rule id="main" scope="public">
            <tag>out="";</tag>
```

```
            <one-of>
              <item repeat="1"><ruleref uri="#digits"/><tag>out+= rules.digits;</tag></
item>
              <item repeat="1"><ruleref uri="#teens"/><tag>out+= rules.teens;</tag></
item>
              <item repeat="1"><ruleref uri="#above_twenty"/><tag>out+=
 rules.above_twenty;</tag></item>
            </one-of>
            <item repeat="0-1"><ruleref uri="#thanks"/></item>
        </rule>

        <rule id="text">
            <item repeat="0-1"><ruleref uri="#hesitation"/></item>
           <one-of>
              <item repeat="0-1">It is</item>
              <item repeat="0-1">The number is</item>
              <item repeat="0-1">Order</item>
              <item repeat="0-1">I want to order</item>
              <item repeat="0-1">Total equipment</item>
           </one-of>
        </rule>

        <rule id="hesitation">
            <one-of>
               <item>Hmm</item>
               <item>Mmm</item>
               <item>My</item>
            </one-of>
          </rule>

        <rule id="thanks">
            <one-of>
               <item>Thanks</item>
               <item>I think</item>
            </one-of>
          </rule>

        <rule id="digits">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <one-of>
                <item>0<tag>out=0;</tag></item>
                <item>1<tag>out=1;</tag></item>
                <item>2<tag>out=2;</tag></item>
                <item>3<tag>out=3;</tag></item>
```

```
            <item>4<tag>out=4;</tag></item>
            <item>5<tag>out=5;</tag></item>
            <item>6<tag>out=6;</tag></item>
            <item>7<tag>out=7;</tag></item>
            <item>8<tag>out=8;</tag></item>
            <item>9<tag>out=9;</tag></item>
            <item>one<tag>out=1;</tag></item>
            <item>two<tag>out=2;</tag></item>
            <item>three<tag>out=3;</tag></item>
            <item>four<tag>out=4;</tag></item>
            <item>five<tag>out=5;</tag></item>
            <item>six<tag>out=6;</tag></item>
            <item>seven<tag>out=7;</tag></item>
            <item>eight<tag>out=8;</tag></item>
            <item>nine<tag>out=9;</tag></item>
        </one-of>
    </rule>

    <rule id="teens">
        <item repeat="0-1"><ruleref uri="#text"/></item>
        <one-of>
            <item>ten<tag>out=10;</tag></item>
            <item>eleven<tag>out=11;</tag></item>
            <item>twelve<tag>out=12;</tag></item>
            <item>thirteen<tag>out=13;</tag></item>
            <item>fourteen<tag>out=14;</tag></item>
            <item>fifteen<tag>out=15;</tag></item>
            <item>sixteen<tag>out=16;</tag></item>
            <item>seventeen<tag>out=17;</tag></item>
            <item>eighteen<tag>out=18;</tag></item>
            <item>nineteen<tag>out=19;</tag></item>
            <item>10<tag>out=10;</tag></item>
            <item>11<tag>out=11;</tag></item>
            <item>12<tag>out=12;</tag></item>
            <item>13<tag>out=13;</tag></item>
            <item>14<tag>out=14;</tag></item>
            <item>15<tag>out=15;</tag></item>
            <item>16<tag>out=16;</tag></item>
            <item>17<tag>out=17;</tag></item>
            <item>18<tag>out=18;</tag></item>
            <item>19<tag>out=19;</tag></item>
        </one-of>
    </rule>
```

```
        <rule id="above_twenty">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <one-of>
                <item>twenty<tag>out=20;</tag></item>
                <item>thirty<tag>out=30;</tag></item>
                <item>forty<tag>out=40;</tag></item>
                <item>fifty<tag>out=50;</tag></item>
                <item>sixty<tag>out=60;</tag></item>
                <item>seventy<tag>out=70;</tag></item>
                <item>eighty<tag>out=80;</tag></item>
                <item>ninety<tag>out=90;</tag></item>
                <item>20<tag>out=20;</tag></item>
                <item>30<tag>out=30;</tag></item>
                <item>40<tag>out=40;</tag></item>
                <item>50<tag>out=50;</tag></item>
                <item>60<tag>out=60;</tag></item>
                <item>70<tag>out=70;</tag></item>
                <item>80<tag>out=80;</tag></item>
                <item>90<tag>out=90;</tag></item>
            </one-of>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out += rules.digits;</
tag></item>
        </rule>

</grammar>
```

請求額

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                             http://www.w3.org/TR/speech-grammar/grammar.xsd"
         xml:lang="en-US" version="1.0"
         root="main"
         mode="voice"
         tag-format="semantics/1.0">

         <!-- Test Cases

         Grammar will support the following inputs:

                 Input: I want to make a payment of one hundred ten dollars
```

```
                Output: $110

         -->

        <rule id="main" scope="public">
            <tag>out="$"</tag>
            <one-of>
                <item><ruleref uri="#sub_hundred"/><tag>out += rules.sub_hundred.sh;</
tag></item>
                <item><ruleref uri="#subThousands"/><tag>out += rules.subThousands;</
tag></item>
            </one-of>
            <item repeat="0-1"><ruleref uri="#thanks"/></item>
        </rule>

        <rule id="text">
           <item repeat="0-1"><ruleref uri="#hesitation"/></item>
           <one-of>
              <item repeat="0-1">I want to make a payment for</item>
              <item repeat="0-1">I want to make a payment of</item>
              <item repeat="0-1">Pay a total of</item>
              <item repeat="0-1">Paying</item>
              <item repeat="0-1">Pay bill for </item>
           </one-of>
        </rule>

        <rule id="hesitation">
            <one-of>
               <item>Hmm</item>
               <item>Mmm</item>
               <item>My</item>
            </one-of>
          </rule>

          <rule id="thanks">
              <one-of>
                 <item>Thanks</item>
                 <item>I think</item>
              </one-of>
            </rule>

        <rule id="digits">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <tag>out.num = 0;</tag>
```

```
        <one-of>
            <item>0<tag>out.num+=0;</tag></item>
            <item>1<tag>out.num+=1;</tag></item>
            <item>2<tag>out.num+=2;</tag></item>
            <item>3<tag>out.num+=3;</tag></item>
            <item>4<tag>out.num+=4;</tag></item>
            <item>5<tag>out.num+=5;</tag></item>
            <item>6<tag>out.num+=6;</tag></item>
            <item>7<tag>out.num+=7;</tag></item>
            <item>8<tag>out.num+=8;</tag></item>
            <item>9<tag>out.num+=9;</tag></item>
            <item>one<tag>out.num+=1;</tag></item>
            <item>two<tag>out.num+=2;</tag></item>
            <item>three<tag>out.num+=3;</tag></item>
            <item>four<tag>out.num+=4;</tag></item>
            <item>five<tag>out.num+=5;</tag></item>
            <item>six<tag>out.num+=6;</tag></item>
            <item>seven<tag>out.num+=7;</tag></item>
            <item>eight<tag>out.num+=8;</tag></item>
            <item>nine<tag>out.num+=9;</tag></item>
        </one-of>
        <item repeat="0-1"><ruleref uri="#currency"/></item>
    </rule>

    <rule id="teens">
        <item repeat="0-1"><ruleref uri="#text"/></item>
        <tag>out.teen = 0;</tag>
        <one-of>
            <item>ten<tag>out.teen+=10;</tag></item>
            <item>eleven<tag>out.teen+=11;</tag></item>
            <item>twelve<tag>out.teen+=12;</tag></item>
            <item>thirteen<tag>out.teen+=13;</tag></item>
            <item>fourteen<tag>out.teen+=14;</tag></item>
            <item>fifteen<tag>out.teen+=15;</tag></item>
            <item>sixteen<tag>out.teen+=16;</tag></item>
            <item>seventeen<tag>out.teen+=17;</tag></item>
            <item>eighteen<tag>out.teen+=18;</tag></item>
            <item>nineteen<tag>out.teen+=19;</tag></item>
        </one-of>
        <item repeat="0-1"><ruleref uri="#currency"/></item>
    </rule>

    <rule id="above_twenty">
        <item repeat="0-1"><ruleref uri="#text"/></item>
```

```
            <tag>out.tens = 0;</tag>
            <one-of>
                <item>twenty<tag>out.tens+=20;</tag></item>
                <item>thirty<tag>out.tens+=30;</tag></item>
                <item>forty<tag>out.tens+=40;</tag></item>
                <item>fifty<tag>out.tens+=50;</tag></item>
                <item>sixty<tag>out.tens+=60;</tag></item>
                <item>seventy<tag>out.tens+=70;</tag></item>
                <item>eighty<tag>out.tens+=80;</tag></item>
                <item>ninety<tag>out.tens+=90;</tag></item>
                <item>hundred<tag>out.tens+=100;</tag></item>
            </one-of>
            <item repeat="0-1"><ruleref uri="#currency"/></item>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out.tens +=
 rules.digits.num;</tag></item>
        </rule>

        <rule id="currency">
            <one-of>
                <item repeat="0-1">dollars</item>
                <item repeat="0-1">Dollars</item>
                <item repeat="0-1">dollar</item>
                <item repeat="0-1">Dollar</item>
            </one-of>
        </rule>


        <rule id="sub_hundred">
            <item repeat="0-1"><ruleref uri="#text"/></item>
            <tag>out.sh = 0;</tag>
            <one-of>
                <item><ruleref uri="#teens"/><tag>out.sh += rules.teens.teen;</tag></
item>
                <item>
                    <ruleref uri="#above_twenty"/><tag>out.sh +=
 rules.above_twenty.tens;</tag>
                </item>
                <item><ruleref uri="#digits"/><tag>out.sh += rules.digits.num;</tag></
item>
            </one-of>
         </rule>

        <rule id="subThousands">
            <ruleref uri="#sub_hundred"/><tag>out = (100 * rules.sub_hundred.sh);</tag>
```

```
            hundred
            <item repeat="0-1"><ruleref uri="#above_twenty"/><tag>out +=
 rules.above_twenty.tens;</tag></item>
            <item repeat="0-1"><ruleref uri="#teens"/><tag>out += rules.teens.teen;</
tag></item>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out += rules.digits.num;</
tag></item>
            <item repeat="0-1"><ruleref uri="#currency"/></item>
        </rule>
</grammar>
```

一般的文法 (ダウンロード)

英数字、通貨、日付 (mm/dd/yy)、数字、あいさつ、ためらい、エージェントといった一般的な文法を用意しています。

英数字

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                             http://www.w3.org/TR/speech-grammar/grammar.xsd"
         xml:lang="en-US" version="1.0"
         root="main"
         mode="voice"
         tag-format="semantics/1.0">


        <!-- Test Cases

            Scenario 1:
                Input: A B C 1 2 3 4
                Output: ABC1234

            Scenario 2:
                Input: 1 2 3 4 A B C
                Output: 1234ABC

            Scenario 3:
                Input: 1 2 3 4 A B C 1
                Output: 123ABC1
        -->
```

```
        <rule id="main" scope="public">
            <tag>out=""</tag>
            <item><ruleref uri="#alphanumeric"/><tag>out +=
rules.alphanumeric.alphanum;</tag></item>
            <item repeat="0-1"><ruleref uri="#alphabets"/><tag>out +=
rules.alphabets.letters;</tag></item>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out +=
rules.digits.numbers</tag></item>
        </rule>

        <rule id="alphanumeric" scope="public">
            <tag>out.alphanum=""</tag>
            <item><ruleref uri="#alphabets"/><tag>out.alphanum +=
rules.alphabets.letters;</tag></item>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out.alphanum +=
rules.digits.numbers</tag></item>
        </rule>

        <rule id="alphabets">
            <tag>out.letters=""</tag>
            <tag>out.firstOccurence=""</tag>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out.firstOccurence +=
rules.digits.numbers; out.letters += out.firstOccurence;</tag></item>
            <item repeat="1-">
                <one-of>
                    <item>A<tag>out.letters+='A';</tag></item>
                    <item>B<tag>out.letters+='B';</tag></item>
                    <item>C<tag>out.letters+='C';</tag></item>
                    <item>D<tag>out.letters+='D';</tag></item>
                    <item>E<tag>out.letters+='E';</tag></item>
                    <item>F<tag>out.letters+='F';</tag></item>
                    <item>G<tag>out.letters+='G';</tag></item>
                    <item>H<tag>out.letters+='H';</tag></item>
                    <item>I<tag>out.letters+='I';</tag></item>
                    <item>J<tag>out.letters+='J';</tag></item>
                    <item>K<tag>out.letters+='K';</tag></item>
                    <item>L<tag>out.letters+='L';</tag></item>
                    <item>M<tag>out.letters+='M';</tag></item>
                    <item>N<tag>out.letters+='N';</tag></item>
                    <item>O<tag>out.letters+='O';</tag></item>
                    <item>P<tag>out.letters+='P';</tag></item>
                    <item>Q<tag>out.letters+='Q';</tag></item>
                    <item>R<tag>out.letters+='R';</tag></item>
                    <item>S<tag>out.letters+='S';</tag></item>
```

```
                    <item>T<tag>out.letters+='T';</tag></item>
                    <item>U<tag>out.letters+='U';</tag></item>
                    <item>V<tag>out.letters+='V';</tag></item>
                    <item>W<tag>out.letters+='W';</tag></item>
                    <item>X<tag>out.letters+='X';</tag></item>
                    <item>Y<tag>out.letters+='Y';</tag></item>
                    <item>Z<tag>out.letters+='Z';</tag></item>
                </one-of>
            </item>
        </rule>

        <rule id="digits">
            <tag>out.numbers="";</tag>
            <item repeat="1-10">
                <one-of>
                    <item>0<tag>out.numbers+=0;</tag></item>
                    <item>1<tag>out.numbers+=1;</tag></item>
                    <item>2<tag>out.numbers+=2;</tag></item>
                    <item>3<tag>out.numbers+=3;</tag></item>
                    <item>4<tag>out.numbers+=4;</tag></item>
                    <item>5<tag>out.numbers+=5;</tag></item>
                    <item>6<tag>out.numbers+=6;</tag></item>
                    <item>7<tag>out.numbers+=7;</tag></item>
                    <item>8<tag>out.numbers+=8;</tag></item>
                    <item>9<tag>out.numbers+=9;</tag></item>
                </one-of>
            </item>
        </rule>
</grammar>
```

通貨

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                             http://www.w3.org/TR/speech-grammar/grammar.xsd"
         xml:lang="en-US" version="1.0"
         root="main"
         mode="voice"
         tag-format="semantics/1.0">

        <rule id="main" scope="public">
```

```
            <tag>out="$"</tag>
            <one-of>
                <item><ruleref uri="#sub_hundred"/><tag>out += rules.sub_hundred.sh;</
tag></item>
                <item><ruleref uri="#subThousands"/><tag>out += rules.subThousands;</
tag></item>
            </one-of>
        </rule>

        <rule id="digits">
            <tag>out.num = 0;</tag>
            <one-of>
                <item>0<tag>out.num+=0;</tag></item>
                <item>1<tag>out.num+=1;</tag></item>
                <item>2<tag>out.num+=2;</tag></item>
                <item>3<tag>out.num+=3;</tag></item>
                <item>4<tag>out.num+=4;</tag></item>
                <item>5<tag>out.num+=5;</tag></item>
                <item>6<tag>out.num+=6;</tag></item>
                <item>7<tag>out.num+=7;</tag></item>
                <item>8<tag>out.num+=8;</tag></item>
                <item>9<tag>out.num+=9;</tag></item>
                <item>one<tag>out.num+=1;</tag></item>
                <item>two<tag>out.num+=2;</tag></item>
                <item>three<tag>out.num+=3;</tag></item>
                <item>four<tag>out.num+=4;</tag></item>
                <item>five<tag>out.num+=5;</tag></item>
                <item>six<tag>out.num+=6;</tag></item>
                <item>seven<tag>out.num+=7;</tag></item>
                <item>eight<tag>out.num+=8;</tag></item>
                <item>nine<tag>out.num+=9;</tag></item>
            </one-of>
            <item repeat="0-1"><ruleref uri="#currency"/></item>
        </rule>

        <rule id="teens">
            <tag>out.teen = 0;</tag>
            <one-of>
                <item>ten<tag>out.teen+=10;</tag></item>
                <item>eleven<tag>out.teen+=11;</tag></item>
                <item>twelve<tag>out.teen+=12;</tag></item>
                <item>thirteen<tag>out.teen+=13;</tag></item>
                <item>fourteen<tag>out.teen+=14;</tag></item>
                <item>fifteen<tag>out.teen+=15;</tag></item>
```

```
                <item>sixteen<tag>out.teen+=16;</tag></item>
                <item>seventeen<tag>out.teen+=17;</tag></item>
                <item>eighteen<tag>out.teen+=18;</tag></item>
                <item>nineteen<tag>out.teen+=19;</tag></item>
            </one-of>
            <item repeat="0-1"><ruleref uri="#currency"/></item>
        </rule>

        <rule id="above_twenty">
            <tag>out.tens = 0;</tag>
            <one-of>
                <item>twenty<tag>out.tens+=20;</tag></item>
                <item>thirty<tag>out.tens+=30;</tag></item>
                <item>forty<tag>out.tens+=40;</tag></item>
                <item>fifty<tag>out.tens+=50;</tag></item>
                <item>sixty<tag>out.tens+=60;</tag></item>
                <item>seventy<tag>out.tens+=70;</tag></item>
                <item>eighty<tag>out.tens+=80;</tag></item>
                <item>ninety<tag>out.tens+=90;</tag></item>
            </one-of>
            <item repeat="0-1"><ruleref uri="#currency"/></item>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out.tens +=
 rules.digits.num;</tag></item>
        </rule>

        <rule id="currency">
            <one-of>
                <item repeat="0-1">dollars</item>
                <item repeat="0-1">Dollars</item>
                <item repeat="0-1">dollar</item>
                <item repeat="0-1">Dollar</item>
            </one-of>
        </rule>


        <rule id="sub_hundred">
            <tag>out.sh = 0;</tag>
            <one-of>
                <item><ruleref uri="#teens"/><tag>out.sh += rules.teens.teen;</tag></
item>
                <item>
                    <ruleref uri="#above_twenty"/><tag>out.sh +=
 rules.above_twenty.tens;</tag>
                </item>
```

```
                <item><ruleref uri="#digits"/><tag>out.sh += rules.digits.num;</tag></
item>
            </one-of>
         </rule>

        <rule id="subThousands">
            <ruleref uri="#sub_hundred"/><tag>out = (100 * rules.sub_hundred.sh);</tag>
            hundred
            <item repeat="0-1"><ruleref uri="#above_twenty"/><tag>out +=
 rules.above_twenty.tens;</tag></item>
            <item repeat="0-1"><ruleref uri="#teens"/><tag>out += rules.teens.teen;</
tag></item>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out += rules.digits.num;</
tag></item>
        </rule>
</grammar>
```

日付、dd/mm

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                             http://www.w3.org/TR/speech-grammar/grammar.xsd"
         xml:lang="en-US" version="1.0"
         root="main"
         mode="voice"
         tag-format="semantics/1.0">

        <rule id="main" scope="public">
            <tag>out=""</tag>
            <item repeat="1-10">
                <one-of>
                    <item><ruleref uri="#digits"/><tag>out += rules.digits + " ";</
tag></item>
                    <item><ruleref uri="#teens"/><tag>out += rules.teens+ " ";</tag></
item>
                    <item><ruleref uri="#above_twenty"/><tag>out += rules.above_twenty+
 " ";</tag></item>
                </one-of>
                <item><ruleref uri="#months"/><tag>out = out + rules.months;</tag></
item>
            </item>
```

```
        </rule>

        <rule id="months">
           <one-of>
             <item>january<tag>out="january";</tag></item>
             <item>february<tag>out="february";</tag></item>
             <item>march<tag>out="march";</tag></item>
             <item>april<tag>out="april";</tag></item>
             <item>may<tag>out="may";</tag></item>
             <item>june<tag>out="june";</tag></item>
             <item>july<tag>out="july";</tag></item>
             <item>august<tag>out="august";</tag></item>
             <item>september<tag>out="september";</tag></item>
             <item>october<tag>out="october";</tag></item>
             <item>november<tag>out="november";</tag></item>
             <item>december<tag>out="december";</tag></item>
             <item>jan<tag>out="january";</tag></item>
             <item>feb<tag>out="february";</tag></item>
             <item>aug<tag>out="august";</tag></item>
             <item>sept<tag>out="september";</tag></item>
             <item>oct<tag>out="october";</tag></item>
             <item>nov<tag>out="november";</tag></item>
             <item>dec<tag>out="december";</tag></item>
           </one-of>
       </rule>

        <rule id="digits">
            <one-of>
                <item>0<tag>out=0;</tag></item>
                <item>1<tag>out=1;</tag></item>
                <item>2<tag>out=2;</tag></item>
                <item>3<tag>out=3;</tag></item>
                <item>4<tag>out=4;</tag></item>
                <item>5<tag>out=5;</tag></item>
                <item>6<tag>out=6;</tag></item>
                <item>7<tag>out=7;</tag></item>
                <item>8<tag>out=8;</tag></item>
                <item>9<tag>out=9;</tag></item>
                <item>first<tag>out=1;</tag></item>
                <item>second<tag>out=2;</tag></item>
                <item>third<tag>out=3;</tag></item>
                <item>fourth<tag>out=4;</tag></item>
                <item>fifth<tag>out=5;</tag></item>
                <item>sixth<tag>out=6;</tag></item>
```

```
            <item>seventh<tag>out=7;</tag></item>
            <item>eighth<tag>out=8;</tag></item>
            <item>ninth<tag>out=9;</tag></item>
            <item>one<tag>out=1;</tag></item>
            <item>two<tag>out=2;</tag></item>
            <item>three<tag>out=3;</tag></item>
            <item>four<tag>out=4;</tag></item>
            <item>five<tag>out=5;</tag></item>
            <item>six<tag>out=6;</tag></item>
            <item>seven<tag>out=7;</tag></item>
            <item>eight<tag>out=8;</tag></item>
            <item>nine<tag>out=9;</tag></item>
        </one-of>
    </rule>

    <rule id="teens">
        <one-of>
            <item>ten<tag>out=10;</tag></item>
            <item>tenth<tag>out=10;</tag></item>
            <item>eleven<tag>out=11;</tag></item>
            <item>twelve<tag>out=12;</tag></item>
            <item>thirteen<tag>out=13;</tag></item>
            <item>fourteen<tag>out=14;</tag></item>
            <item>fifteen<tag>out=15;</tag></item>
            <item>sixteen<tag>out=16;</tag></item>
            <item>seventeen<tag>out=17;</tag></item>
            <item>eighteen<tag>out=18;</tag></item>
            <item>nineteen<tag>out=19;</tag></item>
            <item>tenth<tag>out=10;</tag></item>
            <item>eleventh<tag>out=11;</tag></item>
            <item>twelveth<tag>out=12;</tag></item>
            <item>thirteenth<tag>out=13;</tag></item>
            <item>fourteenth<tag>out=14;</tag></item>
            <item>fifteenth<tag>out=15;</tag></item>
            <item>sixteenth<tag>out=16;</tag></item>
            <item>seventeenth<tag>out=17;</tag></item>
            <item>eighteenth<tag>out=18;</tag></item>
            <item>nineteenth<tag>out=19;</tag></item>
        </one-of>
    </rule>

    <rule id="above_twenty">
        <one-of>
            <item>twenty<tag>out=20;</tag></item>
```

```
                <item>thirty<tag>out=30;</tag></item>
            </one-of>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out += rules.digits;</
tag></item>
        </rule>
</grammar>
```

日付、mm/yy

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                             http://www.w3.org/TR/speech-grammar/grammar.xsd"
         xml:lang="en-US" version="1.0"
         root="main"
         mode="voice"
         tag-format="semantics/1.0">

        <rule id="main" scope="public">
            <tag>out="";</tag>
            <item repeat="1-10">
                <item repeat="1"><ruleref uri="#months"/><tag>out = out +
 rules.months.mon + " ";</tag></item>
                <one-of>
                    <item><ruleref uri="#thousands"/><tag>out += rules.thousands;</
tag></item>
                    <item repeat="0-1"><ruleref uri="#digits"/><tag>out +=
 rules.digits;</tag></item>
                    <item repeat="0-1"><ruleref uri="#teens"/><tag>out +=
 rules.teens;</tag></item>
                    <item repeat="0-1"><ruleref uri="#above_twenty"/><tag>out +=
 rules.above_twenty;</tag></item>
                </one-of>
            </item>
        </rule>

        <rule id="months">
           <tag>out.mon="";</tag>
           <one-of>
               <item>january<tag>out.mon+="january";</tag></item>
               <item>february<tag>out.mon+="february";</tag></item>
               <item>march<tag>out.mon+="march";</tag></item>
```

```
            <item>april<tag>out.mon+="april";</tag></item>
            <item>may<tag>out.mon+="may";</tag></item>
            <item>june<tag>out.mon+="june";</tag></item>
            <item>july<tag>out.mon+="july";</tag></item>
            <item>august<tag>out.mon+="august";</tag></item>
            <item>september<tag>out.mon+="september";</tag></item>
            <item>october<tag>out.mon+="october";</tag></item>
            <item>november<tag>out.mon+="november";</tag></item>
            <item>december<tag>out.mon+="december";</tag></item>
            <item>jan<tag>out.mon+="january";</tag></item>
            <item>feb<tag>out.mon+="february";</tag></item>
            <item>aug<tag>out.mon+="august";</tag></item>
            <item>sept<tag>out.mon+="september";</tag></item>
            <item>oct<tag>out.mon+="october";</tag></item>
            <item>nov<tag>out.mon+="november";</tag></item>
            <item>dec<tag>out.mon+="december";</tag></item>
        </one-of>
    </rule>

    <rule id="digits">
        <one-of>
            <item>zero<tag>out=0;</tag></item>
            <item>one<tag>out=1;</tag></item>
            <item>two<tag>out=2;</tag></item>
            <item>three<tag>out=3;</tag></item>
            <item>four<tag>out=4;</tag></item>
            <item>five<tag>out=5;</tag></item>
            <item>six<tag>out=6;</tag></item>
            <item>seven<tag>out=7;</tag></item>
            <item>eight<tag>out=8;</tag></item>
            <item>nine<tag>out=9;</tag></item>
        </one-of>
    </rule>

    <rule id="teens">
        <one-of>
            <item>ten<tag>out=10;</tag></item>
            <item>eleven<tag>out=11;</tag></item>
            <item>twelve<tag>out=12;</tag></item>
            <item>thirteen<tag>out=13;</tag></item>
            <item>fourteen<tag>out=14;</tag></item>
            <item>fifteen<tag>out=15;</tag></item>
            <item>sixteen<tag>out=16;</tag></item>
            <item>seventeen<tag>out=17;</tag></item>
```

```
                <item>eighteen<tag>out=18;</tag></item>
                <item>nineteen<tag>out=19;</tag></item>
            </one-of>
        </rule>

       <!--  <rule id="singleDigit">
            <item><ruleref uri="#digits"/><tag>out += rules.digits;</tag></item>
        </rule> -->

        <rule id="thousands">
            <!-- <item>
                <ruleref uri="#digits"/>
                <tag>out = (1000 * rules.digits);</tag>
                thousand
            </item> -->
            <item>two thousand<tag>out=2000;</tag></item>
            <item repeat="0-1">and</item>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out += rules.digits;</
tag></item>
            <item repeat="0-1"><ruleref uri="#teens"/><tag>out += rules.teens;</tag></
item>
            <item repeat="0-1"><ruleref uri="#above_twenty"/><tag>out +=
 rules.above_twenty;</tag></item>
        </rule>

        <rule id="above_twenty">
            <one-of>
                <item>twenty<tag>out=20;</tag></item>
                <item>thirty<tag>out=30;</tag></item>
                <item>forty<tag>out=40;</tag></item>
                <item>fifty<tag>out=50;</tag></item>
                <item>sixty<tag>out=60;</tag></item>
                <item>seventy<tag>out=70;</tag></item>
                <item>eighty<tag>out=80;</tag></item>
                <item>ninety<tag>out=90;</tag></item>
            </one-of>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out += rules.digits;</
tag></item>
        </rule>

</grammar>
```

日付、dd/mm/yyyy

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                             http://www.w3.org/TR/speech-grammar/grammar.xsd"
         xml:lang="en-US" version="1.0"
         root="main"
         mode="voice"
         tag-format="semantics/1.0">

         <rule id="main" scope="public">
             <tag>out=""</tag>
             <item repeat="1-10">
                 <one-of>
                   <item><ruleref uri="#digits"/><tag>out += rules.digits + " ";</
tag></item>
                   <item><ruleref uri="#teens"/><tag>out += rules.teens+ " ";</tag></
item>
                   <item><ruleref uri="#above_twenty"/><tag>out += rules.above_twenty+
 " ";</tag></item>
                 </one-of>
                 <item repeat="1"><ruleref uri="#months"/><tag>out = out +
 rules.months.mon + " ";</tag></item>
                 <one-of>
                     <item><ruleref uri="#thousands"/><tag>out += rules.thousands;</
tag></item>
                     <item repeat="0-1"><ruleref uri="#digits"/><tag>out +=
 rules.digits;</tag></item>
                     <item repeat="0-1"><ruleref uri="#teens"/><tag>out +=
 rules.teens;</tag></item>
                     <item repeat="0-1"><ruleref uri="#above_twenty"/><tag>out +=
 rules.above_twenty;</tag></item>
                 </one-of>
             </item>
         </rule>

         <rule id="months">
            <tag>out.mon=""</tag>
            <one-of>
                <item>january<tag>out.mon+="january";</tag></item>
                <item>february<tag>out.mon+="february";</tag></item>
                <item>march<tag>out.mon+="march";</tag></item>
```

```
            <item>april<tag>out.mon+="april";</tag></item>
            <item>may<tag>out.mon+="may";</tag></item>
            <item>june<tag>out.mon+="june";</tag></item>
            <item>july<tag>out.mon+="july";</tag></item>
            <item>august<tag>out.mon+="august";</tag></item>
            <item>september<tag>out.mon+="september";</tag></item>
            <item>october<tag>out.mon+="october";</tag></item>
            <item>november<tag>out.mon+="november";</tag></item>
            <item>december<tag>out.mon+="december";</tag></item>
            <item>jan<tag>out.mon+="january";</tag></item>
            <item>feb<tag>out.mon+="february";</tag></item>
            <item>aug<tag>out.mon+="august";</tag></item>
            <item>sept<tag>out.mon+="september";</tag></item>
            <item>oct<tag>out.mon+="october";</tag></item>
            <item>nov<tag>out.mon+="november";</tag></item>
            <item>dec<tag>out.mon+="december";</tag></item>
        </one-of>
    </rule>

    <rule id="digits">
        <one-of>
            <item>zero<tag>out=0;</tag></item>
            <item>one<tag>out=1;</tag></item>
            <item>two<tag>out=2;</tag></item>
            <item>three<tag>out=3;</tag></item>
            <item>four<tag>out=4;</tag></item>
            <item>five<tag>out=5;</tag></item>
            <item>six<tag>out=6;</tag></item>
            <item>seven<tag>out=7;</tag></item>
            <item>eight<tag>out=8;</tag></item>
            <item>nine<tag>out=9;</tag></item>
        </one-of>
    </rule>

    <rule id="teens">
        <one-of>
            <item>ten<tag>out=10;</tag></item>
            <item>eleven<tag>out=11;</tag></item>
            <item>twelve<tag>out=12;</tag></item>
            <item>thirteen<tag>out=13;</tag></item>
            <item>fourteen<tag>out=14;</tag></item>
            <item>fifteen<tag>out=15;</tag></item>
            <item>sixteen<tag>out=16;</tag></item>
            <item>seventeen<tag>out=17;</tag></item>
```

```
                <item>eighteen<tag>out=18;</tag></item>
                <item>nineteen<tag>out=19;</tag></item>
            </one-of>
        </rule>

        <rule id="thousands">
            <item>two thousand<tag>out=2000;</tag></item>
            <item repeat="0-1">and</item>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out += rules.digits;</
tag></item>
            <item repeat="0-1"><ruleref uri="#teens"/><tag>out += rules.teens;</tag></
item>
            <item repeat="0-1"><ruleref uri="#above_twenty"/><tag>out +=
 rules.above_twenty;</tag></item>
        </rule>

        <rule id="above_twenty">
            <one-of>
                <item>twenty<tag>out=20;</tag></item>
                <item>thirty<tag>out=30;</tag></item>
                <item>forty<tag>out=40;</tag></item>
                <item>fifty<tag>out=50;</tag></item>
                <item>sixty<tag>out=60;</tag></item>
                <item>seventy<tag>out=70;</tag></item>
                <item>eighty<tag>out=80;</tag></item>
                <item>ninety<tag>out=90;</tag></item>
            </one-of>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out += rules.digits;</
tag></item>
        </rule>

 </grammar>
```

数字、桁数

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                            http://www.w3.org/TR/speech-grammar/grammar.xsd"
        xml:lang="en-US" version="1.0"
        root="digits"
        mode="voice"
```

```
         tag-format="semantics/1.0">

        <rule id="digits">
            <tag>out=""</tag>
            <item><ruleref uri="#singleDigit"/><tag>out += rules.singleDigit.digit;</
tag></item>
        </rule>

        <rule id="singleDigit">
            <tag>out.digit=""</tag>
            <item repeat="1-10">
                <one-of>
                    <item>0<tag>out.digit+=0;</tag></item>
                    <item>zero<tag>out.digit+=0;</tag></item>
                    <item>1<tag>out.digit+=1;</tag></item>
                    <item>one<tag>out.digit+=1;</tag></item>
                    <item>2<tag>out.digit+=2;</tag></item>
                    <item>two<tag>out.digit+=2;</tag></item>
                    <item>3<tag>out.digit+=3;</tag></item>
                    <item>three<tag>out.digit+=3;</tag></item>
                    <item>4<tag>out.digit+=4;</tag></item>
                    <item>four<tag>out.digit+=4;</tag></item>
                    <item>5<tag>out.digit+=5;</tag></item>
                    <item>five<tag>out.digit+=5;</tag></item>
                    <item>6<tag>out.digit+=6;</tag></item>
                    <item>six<tag>out.digit+=6;</tag></item>
                    <item>7<tag>out.digit+=7;</tag></item>
                    <item>seven<tag>out.digit+=7;</tag></item>
                    <item>8<tag>out.digit+=8;</tag></item>
                    <item>eight<tag>out.digit+=8;</tag></item>
                    <item>9<tag>out.digit+=9;</tag></item>
                    <item>nine<tag>out.digit+=9;</tag></item>
                </one-of>
            </item>
        </rule>
</grammar>
```

数字、序数

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
```

```
                              http://www.w3.org/TR/speech-grammar/grammar.xsd"
         xml:lang="en-US" version="1.0"
         root="main"
         mode="voice"
         tag-format="semantics/1.0">

        <rule id="main" scope="public">
            <tag>out=""</tag>
            <one-of>
              <item repeat="1"><ruleref uri="#digits"/><tag>out+= rules.digits;</tag></
item>
              <item repeat="1"><ruleref uri="#teens"/><tag>out+= rules.teens;</tag></
item>
              <item repeat="1"><ruleref uri="#above_twenty"/><tag>out+=
 rules.above_twenty;</tag></item>
            </one-of>
        </rule>

        <rule id="digits">
            <one-of>
                <item>0<tag>out=0;</tag></item>
                <item>1<tag>out=1;</tag></item>
                <item>2<tag>out=2;</tag></item>
                <item>3<tag>out=3;</tag></item>
                <item>4<tag>out=4;</tag></item>
                <item>5<tag>out=5;</tag></item>
                <item>6<tag>out=6;</tag></item>
                <item>7<tag>out=7;</tag></item>
                <item>8<tag>out=8;</tag></item>
                <item>9<tag>out=9;</tag></item>
                <item>one<tag>out=1;</tag></item>
                <item>two<tag>out=2;</tag></item>
                <item>three<tag>out=3;</tag></item>
                <item>four<tag>out=4;</tag></item>
                <item>five<tag>out=5;</tag></item>
                <item>six<tag>out=6;</tag></item>
                <item>seven<tag>out=7;</tag></item>
                <item>eight<tag>out=8;</tag></item>
                <item>nine<tag>out=9;</tag></item>
            </one-of>
        </rule>

        <rule id="teens">
            <one-of>
```

```
                <item>ten<tag>out=10;</tag></item>
                <item>eleven<tag>out=11;</tag></item>
                <item>twelve<tag>out=12;</tag></item>
                <item>thirteen<tag>out=13;</tag></item>
                <item>fourteen<tag>out=14;</tag></item>
                <item>fifteen<tag>out=15;</tag></item>
                <item>sixteen<tag>out=16;</tag></item>
                <item>seventeen<tag>out=17;</tag></item>
                <item>eighteen<tag>out=18;</tag></item>
                <item>nineteen<tag>out=19;</tag></item>
                <item>10<tag>out=10;</tag></item>
                <item>11<tag>out=11;</tag></item>
                <item>12<tag>out=12;</tag></item>
                <item>13<tag>out=13;</tag></item>
                <item>14<tag>out=14;</tag></item>
                <item>15<tag>out=15;</tag></item>
                <item>16<tag>out=16;</tag></item>
                <item>17<tag>out=17;</tag></item>
                <item>18<tag>out=18;</tag></item>
                <item>19<tag>out=19;</tag></item>
            </one-of>
        </rule>

        <rule id="above_twenty">
            <one-of>
                <item>twenty<tag>out=20;</tag></item>
                <item>thirty<tag>out=30;</tag></item>
                <item>forty<tag>out=40;</tag></item>
                <item>fifty<tag>out=50;</tag></item>
                <item>sixty<tag>out=60;</tag></item>
                <item>seventy<tag>out=70;</tag></item>
                <item>eighty<tag>out=80;</tag></item>
                <item>ninety<tag>out=90;</tag></item>
                <item>20<tag>out=20;</tag></item>
                <item>30<tag>out=30;</tag></item>
                <item>40<tag>out=40;</tag></item>
                <item>50<tag>out=50;</tag></item>
                <item>60<tag>out=60;</tag></item>
                <item>70<tag>out=70;</tag></item>
                <item>80<tag>out=80;</tag></item>
                <item>90<tag>out=90;</tag></item>
            </one-of>
            <item repeat="0-1"><ruleref uri="#digits"/><tag>out += rules.digits;</
tag></item>
```

```
        </rule>

</grammar>
```

[エージェント]

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                             http://www.w3.org/TR/speech-grammar/grammar.xsd"
         xml:lang="en-US" version="1.0"
         root="main"
         mode="voice"
         tag-format="semantics/1.0">

         <rule id="main" scope="public">
             <tag>out=""</tag>
             <ruleref uri="#text"/><tag>out = rules.text</tag>
         </rule>

         <rule id="text">
           <one-of>
              <item>Can I talk to the agent<tag>out="You will be trasnfered to the
 agent in a while"</tag></item>
              <item>talk to an agent<tag>out="You will be trasnfered to the agent in a
 while"</tag></item>
           </one-of>
         </rule>
</grammar>
```

あいさつ

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                             http://www.w3.org/TR/speech-grammar/grammar.xsd"
         xml:lang="en-US" version="1.0"
         root="main"
         mode="voice"
         tag-format="semantics/1.0">
```

```
         <rule id="main" scope="public">
             <tag>out=""</tag>
             <ruleref uri="#text"/><tag>out = rules.text</tag>
         </rule>

         <rule id="text">
           <one-of>
              <item>hey<tag>out="Greeting"</tag></item>
              <item>hi<tag>out="Greeting"</tag></item>
              <item>Hi<tag>out="Greeting"</tag></item>
              <item>Hey<tag>out="Greeting"</tag></item>
              <item>Hello<tag>out="Greeting"</tag></item>
              <item>hello<tag>out="Greeting"</tag></item>
           </one-of>
         </rule>
</grammar>
```

ためらい

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                             http://www.w3.org/TR/speech-grammar/grammar.xsd"
         xml:lang="en-US" version="1.0"
         root="main"
         mode="voice"
         tag-format="semantics/1.0">

         <rule id="main" scope="public">
             <tag>out=""</tag>
             <ruleref uri="#text"/><tag>out = rules.text</tag>
         </rule>

         <rule id="text">
           <one-of>
              <item>Hmm<tag>out="Waiting for your input"</tag></item>
              <item>Mmm<tag>out="Waiting for your input"</tag></item>
              <item>Can you please wait<tag>out="Waiting for your input"</tag></item>
           </one-of>
         </rule>
</grammar>
```

コンポジットスロットタイプ

複合スロットは、2 つ以上のスロットを組み合わせたもので、1 回のユーザー入力で複数の情報をキャプチャします。たとえば、「都市と都道府県または郵便番号」をリクエストすることで位置情報を誘発するようにボットを設定できます。これとは対照的に、会話が別々のスロットタイプを使用するように設定されていると、会話がぎこちなくなります (「都市は何ですか?」の次に「郵便番号は何ですか?」)。複合スロットでは、1 つのスロットからすべての情報を取り込むことができます。複合スロットは、都市、州、郵便番号などのサブスロットと呼ばれるスロットを組み合わせたものです。

使用可能な Amazon Lex スロットタイプ (組み込み) と独自のスロット (カスタムスロット) を組み合わせて使用できます。必要なサブスロット内の情報をキャプチャする論理式を設計できます。例: 市区町村と都道府県、郵便番号。

複合スロットタイプは、Generative AI を介した機能サポートが制限された en-US およびロケールでのみ使用できます。

複合スロットタイプの作成

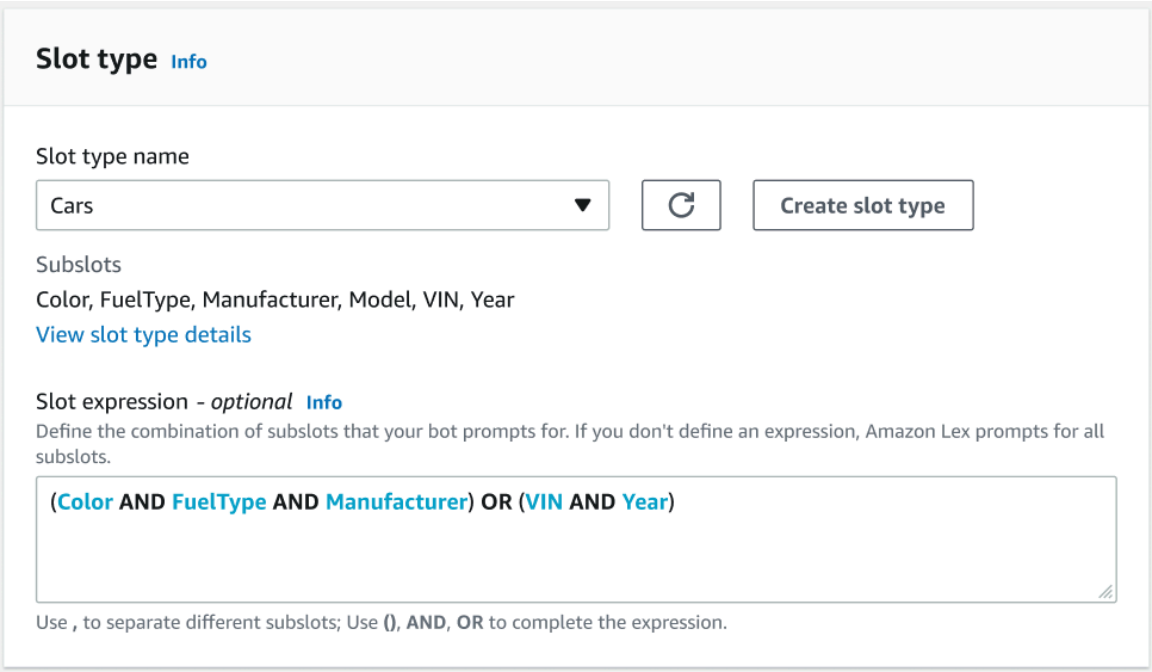
複合スロット内のサブスロットを使用するには、まず複合スロットタイプを設定する必要があります。そのためには、スロットタイプの追加コンソールステップまたは API オペレーションを使用します。複合スロットタイプの名前と説明を選択したら、サブスロットの情報を入力する必要があります。スロットタイプの追加についての詳細は、「スロットタイプの追加」を参照してください。

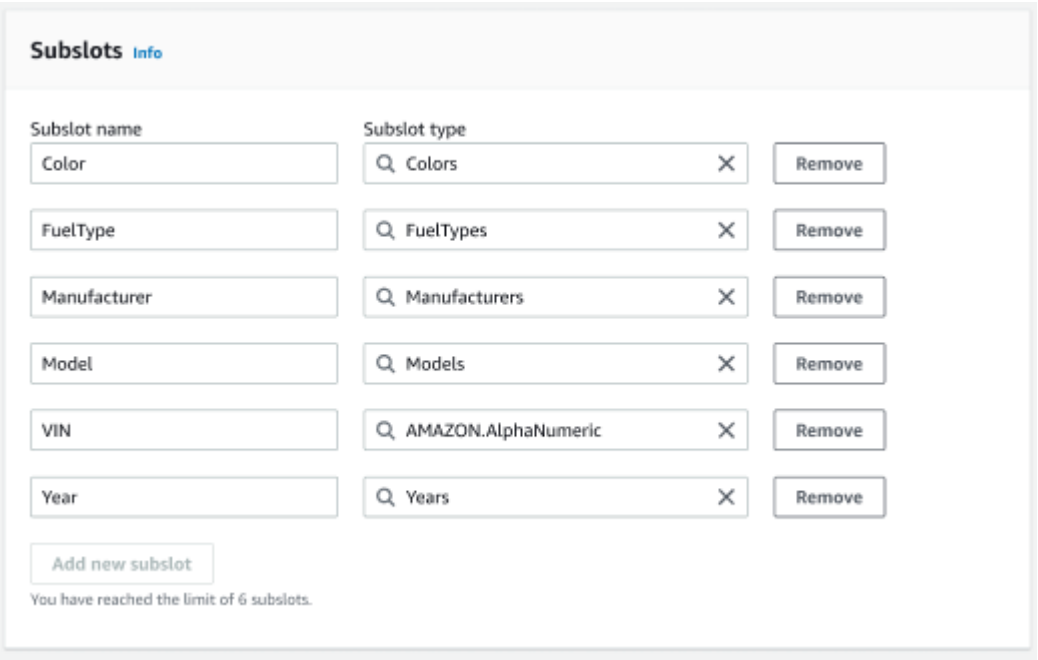
サブスロット

複合スロットタイプでは、サブスロットと呼ばれる基礎となるスロットの設定が必要です。1 回のリクエストで顧客から複数の情報を誘発したい場合は、サブスロットを組み合わせて設定してください。例: 市区町村と都道府県、郵便番号。コンポジットスロットには最大 6 個のサブスロットを追加できます。

単一スロットタイプのスロットを使用して、複合スロットタイプにサブスロットを追加できます。ただし、複合スロットタイプをサブスロットのスロットタイプとして使用することはできません。

以下の画像は、次のサブスロットを組み合わせた複合スロット「Car」を示しています: Color、FuelType、Manufacturer、Model、VIN、および Year。





式ビルダー

コンポジットスロットの処理を促進するには、オプションで式ビルダーを使用できます。式ビルダーを使用すると、必要なサブスロット値を目的の順序でキャプチャする論理スロット式を設計できます。ブール式の一部として、AND や OR などの演算子を使用できます。設計された式に基づいて、必要なサブスロットが満たされると、複合スロットは満たされたと見なされます。

複合スロットタイプの使用

インテントによっては、1 つのスロットの一部としてキャプチャしたい場合があります。たとえば、車両メンテナンススケジュールボットなら、次のような発話のインテントを持っていることがあります。

```
My car is a {car}
```

このインテントは、{car} 複合スロットに車の詳細を含むスロットのリストが含まれていることを想定しています。たとえば、「2021 年式ホワイトトヨタカムリ」などです。

複合スロットは複数値スロットとは異なります。複合スロットは複数のスロットで構成され、それぞれに独自の値があります。一方、複数値スロットは値のリストを格納できる単一のスロットです。複数値スロットの詳細については、「スロットで複数の値を使用する」を参照してください。

複合スロットの場合、Amazon Lex は、RecognizeText または RecognizeUtterance オペレーションへの応答で各サブスロットの値を返します。以下は、発話で返されるスロット情報です。「『2021 年式ホワイトトヨタカムリ』のサービス予約を CarService ボットから入れたいと思っています。」

```
"slots": {
    "CarType": {
        "value": {
            "originalValue": "White Toyota Camry 2021",
            "interpretedValue": "White Toyota Camry 2021",
            "resolvedValues": [
                "white Toyota Camry 2021"
            ]
        },
        "subSlots": {
            "Color": {
                "value": {
                    "originalValue": "White",
                    "interpretedValue": "White",
                    "resolvedValues": [
                        "white"
                    ]
                },
                "shape": "Scalar"
            },
            "Manufacturer": {
                "value": {
                    "originalValue": "Toyota",
                    "interpretedValue": "Toyota",
                    "resolvedValues": [
```

```
                        "Toyota"
                    ]
                },
                "shape": "Scalar"
            },
            "Model": {
                "value": {
                    "originalValue": "Camry",
                    "interpretedValue": "Camry",
                    "resolvedValues": [
                       "Camry"
                    ]
                },
                "shape": "Scalar"
            },
            "Year": {
                "value": {
                    "originalValue": "2021",
                    "interpretedValue": "2021",
                    "resolvedValues": [
                        "2021"
                    ]
                },
                "shape": "Scalar"
            }
        }
    },
    ...
}
```

複合スロットは、会話の最初のターン、または N 番目のターンで誘発します。入力された値に基づいて、複合スロットは残りの必要なサブスロットを誘発することができます。

複合スロットは常に各サブスロットの値を返します。発話に特定のサブスロットの認識可能な値が含まれていない場合、その特定のサブスロットに対する応答は返されません。

複合スロットは、テキスト入力と音声入力の両方で動作します。

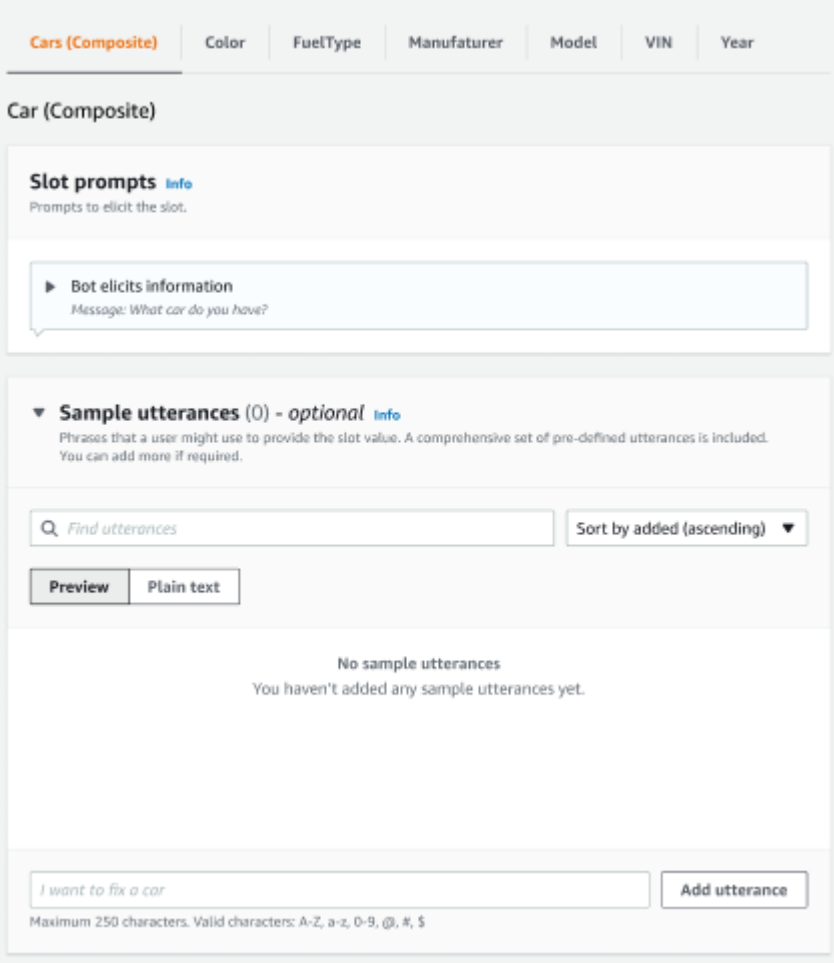
インテントにスロットを追加する場合、複合スロットはカスタムスロットタイプとしてのみ使用できます。

プロンプトでは複合スロットを使用できます。たとえば、インテントの確認プロンプトを次のように設定します。

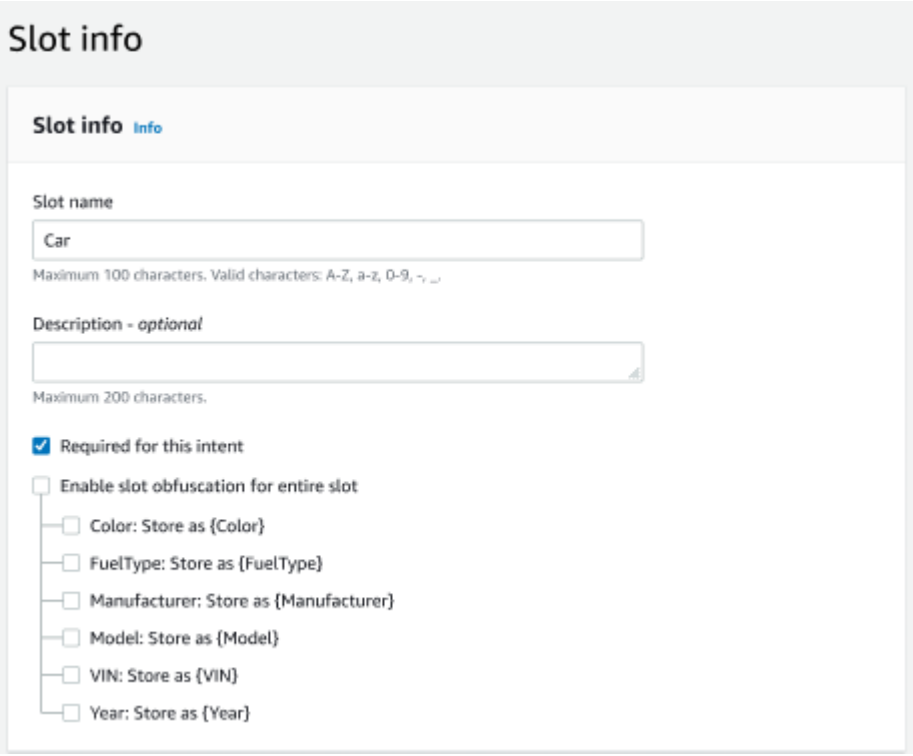
```
Would you like me to schedule service for your 2021 White Toyota Camry?
```

Amazon Lex がユーザーにプロンプトを送信すると、「2021 年式ホワイトトヨタカムリのサービス予約を入れますか?」と送信されます。

各サブスロットはスロットとして設定されます。スロットプロンプトを追加して、サブスロットとサンプル発話を誘発することができます。デフォルト値だけでなく、サブスロットの待機と続行も有効にできます。詳細については、Lex V2 ボットのインテントでのデフォルトスロット値の使用を参照してください。



スロット難読化を使用すると、会話ログの複合スロット全体をマスクすることができます。スロットの難読化は複合スロットレベルで適用され、有効にすると複合スロットに属するサブスロットの値が難読化されることに注意してください。スロット値を難読化すると、各スロット値の値がスロットの名前に置き換えられます。詳細については、「Lex V2 の会話ログでスロット値を隠す」を参照してください。



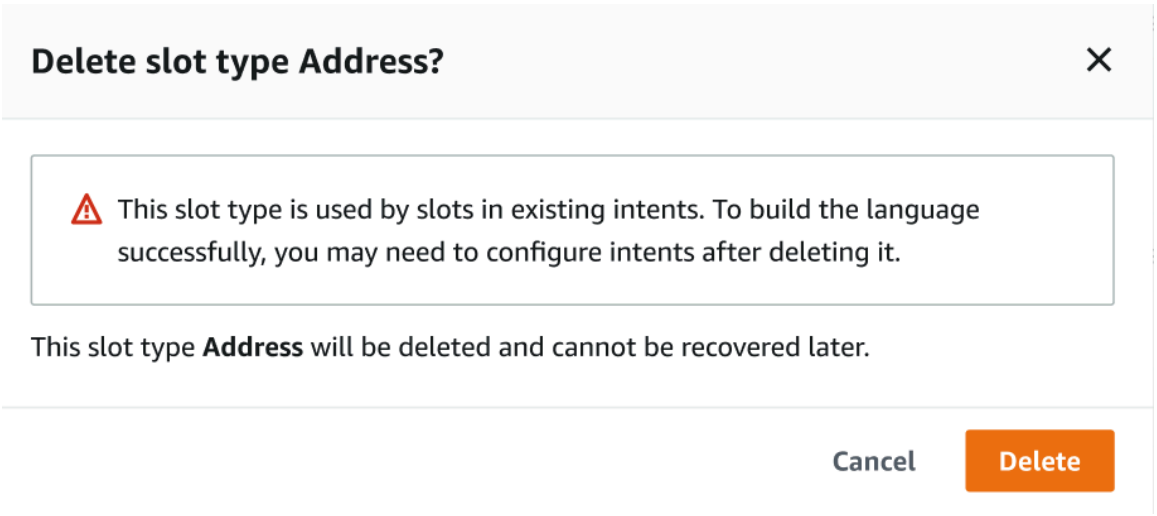
複合スロットタイプの編集

複合スロット構成内からサブスロットを編集して、サブスロット名とスロットタイプを変更できます。ただし、インテントが複合スロットを使用している場合は、サブスロットを変更する前にインテントを編集する必要があります。

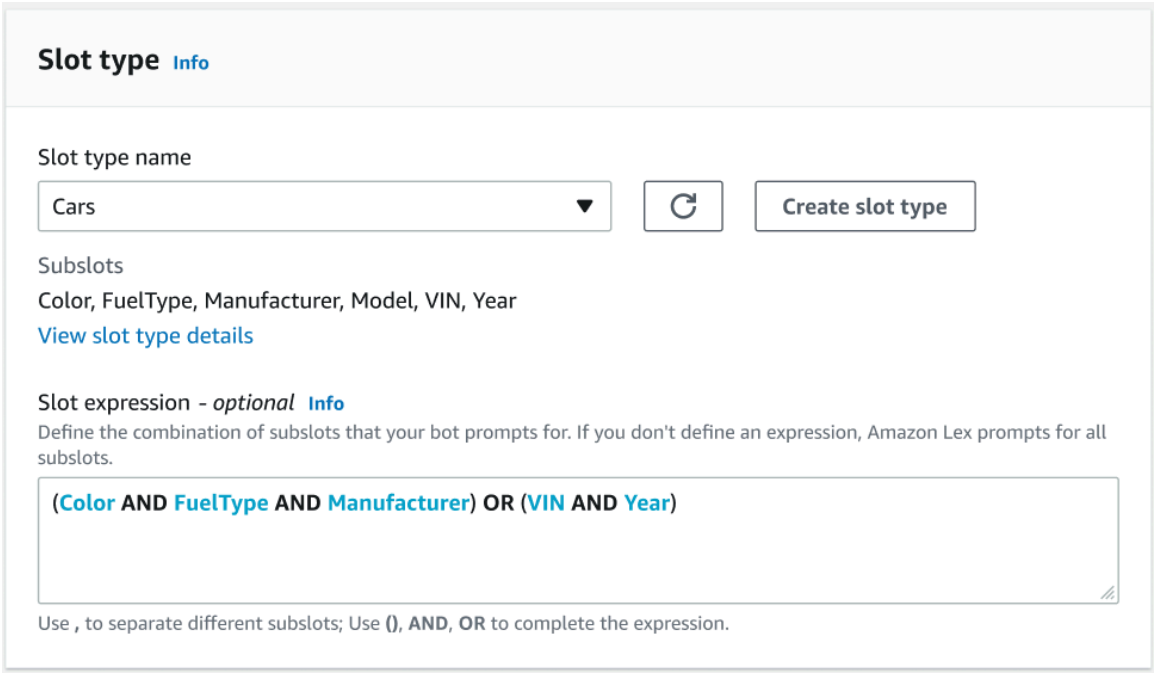
> Existing intents use this slot type. To build the language successfully, you may need to configure those intents after editing sub slots.

複合スロットタイプの削除

複合スロット構成内からサブスロットを削除できます。サブスロットがインテント内で使用されている場合でも、そのサブスロットはそのインテントから削除されることに注意してください。



式ビルダーのスロット式は、削除されたサブスロットについて通知するアラートを表示します。



コンソールを使用したボットのテスト

Amazon Lex V2 コンソールには、ボットとのインタラクションをテストするために使用できるテストウィンドウが含まれています。テストウィンドウを使用して、ボットとテスト会話を行い、アプリケーションがボットから受け取る応答を確認します。

ボットを用いて実行できるテストには 2 つのタイプがあります。最初のエクスプレステストでは、ボットの作成に使用した正確なフレーズでボットをテストできます。例えば、インテントに「花を拾いたい」という発話を追加した場合、その正確なフレーズを使用してボットをテストできます。

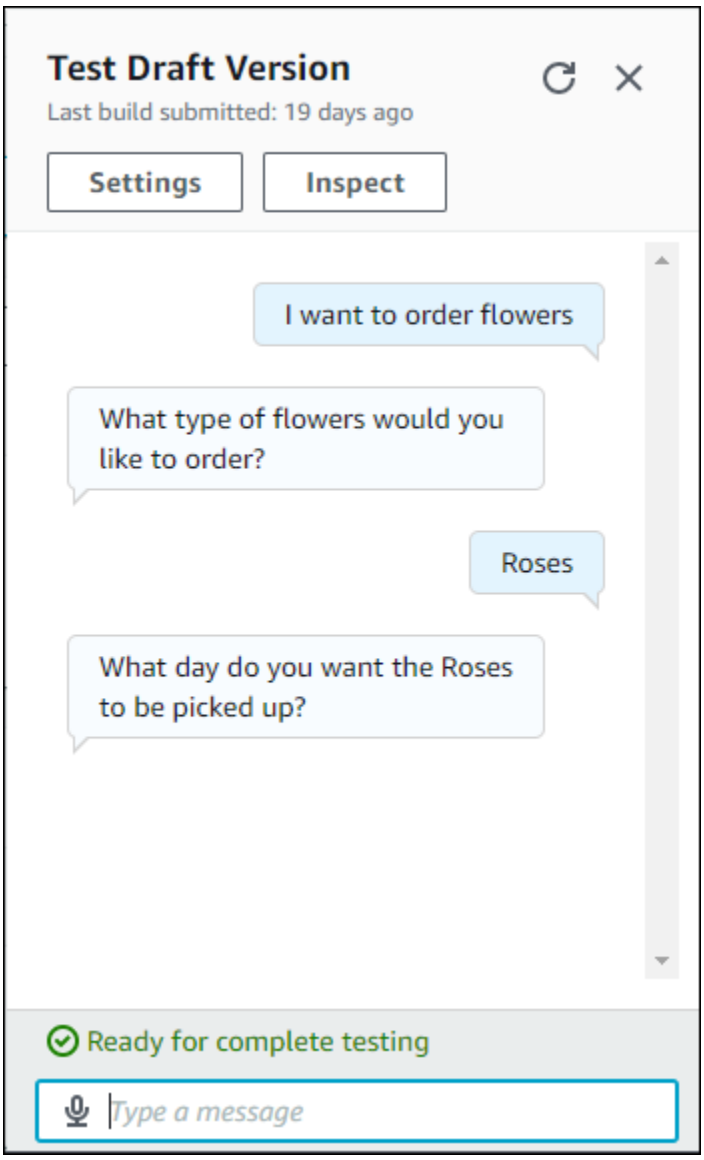
2 番目のタイプである [Complete testing] では、設定した発話に関連するフレーズを使用してボットをテストできます。例えば、「花を注文できますか」というフレーズを使用して、ボットとの会話を始めることができます。

ボットをテストするには、特定のエイリアスと言語を使用します。ボットの開発バージョンをテストする場合は、`TestBotAlias` のテスト用のエイリアスを使います。

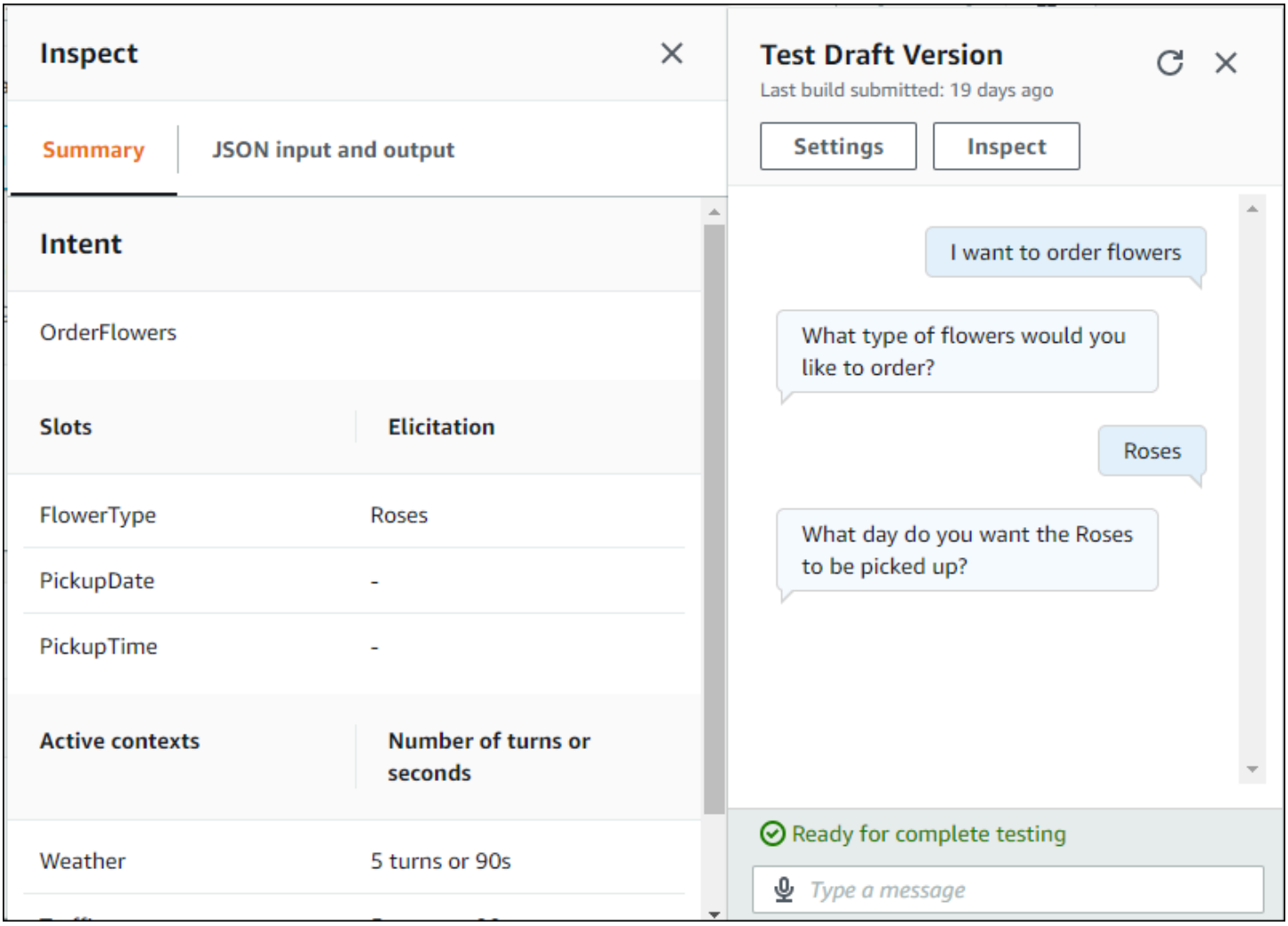
[Test] ウィンドウを開くには

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/lex/ で Amazon Lex コンソールを開きます。
2. ボットのリストからテストするボットを選択します。
3. 左側のメニューから、Aliases を選択します。
4. エイリアスのリストから、テストするエイリアスを選択します。
5. Languages から、テストする言語のラジオボタンを選択し、Test を選択します。

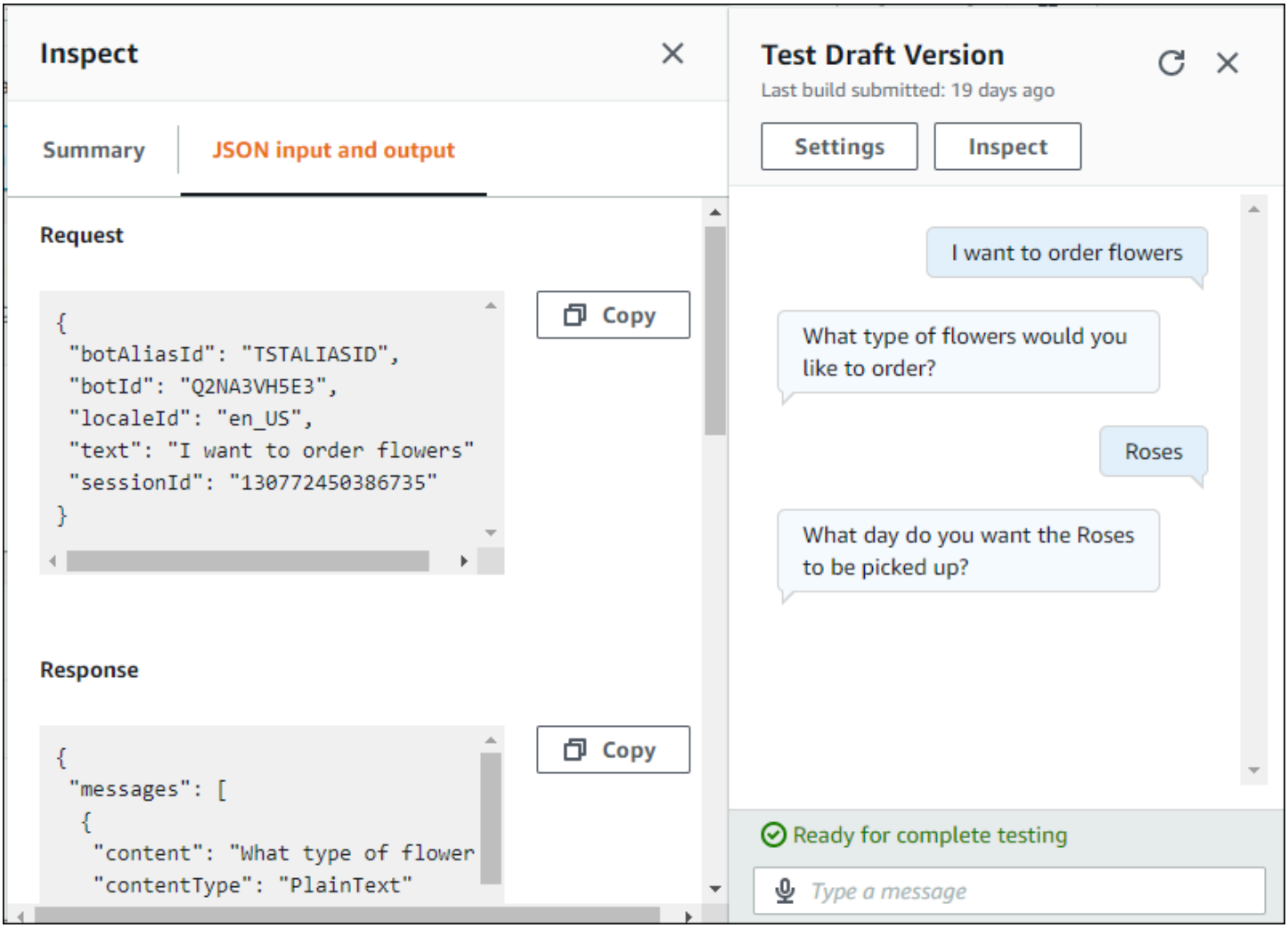
Test を選択したあとに、コンソールにテストウィンドウが開きます。次の図に示すように、テストウィンドウを使用してボットを操作できます。



会話に加えて、テストウィンドウで Inspect を選択し、ボットから返された応答を確認することもできます。最初のビューには、ボットからテストウィンドウに返された情報の概要が表示されます。



テスト検査ウィンドウを使用して、ボットとテストウィンドウの間で送信される JSON 構造を確認することもできます。テストウィンドウからのリクエストと Amazon Lex V2 からのレスポンスの両方を確認できます。


Inspect
Summary
JSON input and output
Request
{
"botAliasId": "TSTALIASID",
"botId": "Q2NA3VH5E3",
"localeId": "en_US",
"text": "I want to order flowers"
"sessionId": "130772450386735"
}
Copy
Response
{
"messages": [
{
"content": "What type of flower
"contentType": "PlainText"
Copy
Test Draft Version
Last build submitted: 19 days ago
Settings
Inspect
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?
Ready for complete testing
Type a message

生成 AI を使用した Lex V2 ボットの作成とパフォーマンスの最適化

Amazon Bedrock の生成 AI 機能を活用して、Amazon Lex V2 ボットの構築プロセスを自動化し、スピードアップします。Amazon Bedrock を使用して以下のプロセスを実行できます。

> **Note**
>
> これらの機能では、生成 AI を使用します。このサービスを利用する際には、不正確または不適切な応答が返される場合があることを覚えておいてください。詳細については、「AWS の責任ある AI に関するポリシー」を参照してください。
> Amazon Bedrock を利用: AWS は自動不正検出を実装しています。Amazon Lex V2 の生成 AI 機能は Amazon Bedrock 上に構築されているため、ユーザーは Amazon Bedrock に実装されているコントロールを引き継ぎ、AI の安全性、セキュリティ、責任ある使用を実現できます。

- 自然言語の説明を使用して、新しいボットを作成し、関連するインテントとスロットタイプを効率的に追加します。
- ボットのインテントに合ったサンプル発話を自動的に生成します。
- ボットのスロット解決パフォーマンスを向上させます。
- 顧客の質問に答えるのに役立つインテントを作成します。
- Amazon Bedrock エージェントと Amazon Bedrock ナレッジベースを使用して、顧客の質問に答えます。

Amazon Lex V2 の生成 AI 機能は、コンソールまたは API を使用して有効にすることができます。

> **Note**
>
> 生成 AI 機能を活用するには、最初に以下の前提条件を満たす必要があります。
>
> 1. Amazon Bedrock の使用料金については、「Amazon Bedrock の料金」を参照してください。
> 2. ボットロケールで生成 AI 機能を有効にします。これを行うには、「生成 AI を使用した Lex V2 ボットの作成とパフォーマンスの最適化」の手順に従います。

Using the console

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/lexv2/home で Amazon Lex V2 コンソールを開きます。
2. 生成 AI 機能を有効にするボットとボットロケールを選択します。
3. [生成 AI 設定] セクションで、[設定] を選択します。
4. 有効にする機能ごとに [有効] ボタンを切り替えます。この機能で使用するモデルとバージョンを選択します。機能を有効にすると、追加の料金が発生する場合があります。Amazon Bedrock の使用料金については、「Amazon Bedrock の料金」を参照してください。機能の詳細については、以下のリストから対応するトピックを選択してください。有効にする機能をオンにしたら、[保存] を選択します。機能がオンになったことを確認する緑色の成功バナーが表示されます。

Using the API

1. 新しいボットで生成 AI 機能を有効にするには、CreateBot オペレーションを使用して新しいボットを作成します。
2. CreateBotLocale リクエストを送信します (必要に応じて `generativeAISettings` オブジェクトを変更してください)。既存のボットの機能を有効にする場合は、代わりに UpdateBotLocale リクエストを送信します。

   a. 記述的ボットビルダーを使用できるようにするには、`descriptiveBotBuilder` オブジェクトを変更します。使用する基盤モデルを `modelArn` フィールドで指定し、`enabled` の値を `True` に設定します。

   b. スロット解決の向上を有効にするには、`slotResolutionImprovement` オブジェクトを変更します。使用する基盤モデルを `modelArn` フィールドで指定し、`enabled` の値を `True` に設定します。

   c. サンプル発話の生成を有効にするには、`sampleUtteranceGeneration` オブジェクトを変更します。使用する基盤モデルを `modelArn` フィールドで指定し、`enabled` の値を `True` に設定します。

トピック

- 記述的ボットビルダーを使用した、Lex V2 での説明文によるボットの構築
- 発話生成を使用したインテント認識用のサンプル発話の生成
- アシスト付きスロット解決を使用した Amazon Lex V2 のスロット値の明確化

- BedrockAgentIntent を使用した、Amazon Lex V2 での Amazon Bedrock エージェントの使用
- アシスト付き NLU を使用して Lex V2 のインテント分類とスロット解決を改善する
- 意図的な曖昧さを解消して、あいまいなユーザー入力を解決する
- AI を活用した Bot Analyzer を使用してボットを最適化する
- AMAZON.QnAIntent

記述的ボットビルダーを使用した、Lex V2 での説明文によるボットの構築

> **Note**
>
> 生成 AI 機能を活用するには、最初に以下の前提条件を満たす必要があります。
>
> 1. Amazon Bedrock の使用料金については、「Amazon Bedrock の料金」を参照してください。
> 2. ボットロケールで生成 AI 機能を有効にします。これを行うには、「生成 AI を使用した Lex V2 ボットの作成とパフォーマンスの最適化」の手順に従います。

記述型ボットビルダーでは、Amazon Bedrock の大規模言語モデルへのアクセスを活用して、ボット作成プロセスの効率を高めることができます。ボットの目的および実行すべきアクションを含むプロンプトを自然言語で入力します。Amazon Lex V2 は Amazon Bedrock の機能を利用して、ユーザーの説明に基づいてボットに関連するインテントとスロットタイプを生成します。保持したいインテントとスロットタイプを選択したら、ボットを反復処理して特定のユースケースに合わせて変更できます。記述的ボットビルダーを使用すると、ボットのインテントとスロットタイプを手動で作成する必要がなくなるため、時間を節約できます。

記述的ボットビルダーは英語ロケールで使用できます (「Amazon Lex V2 でサポートされている言語と ロケール」の表で en_ で始まるロケールを参照してください)。

ボットを作成する前に、次のことを行います。

1. 「Lex V2 で自然言語による説明でボットを作成するために必要な権限」の手順を参照して、ロールに正しい権限があることを確認します。
2. 使用する説明を決定します。ボットの説明のサンプルについては、「記述型ボットビルダーのボットの説明例」を参照してください。

自然言語を使用してボットができることを説明し、ボットを作成します。Amazon Lex V2 は Amazon Bedrock モデルを呼び出し、ボットのユースケースに合ったインテントとスロットタイプを生成します。ボットは、コンソールまたは API を使用して作成できます。

Console

記述的ボットビルダーを使用してボットを作成する

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/lexv2/home で Amazon Lex V2 コンソールを開きます。
2. [ボット] ページで、[ボットを作成] を選択します。
3. [作成方法] で、[記述的ボットビルダー - 生成 AI] を選択します。
4. ボットの名前とオプションの説明を入力し、IAM 権限を設定して、ボットが COPPA の対象となるかどうかを選択します。[次へ] を選択します。
5. ボットを作成する言語、ボットの音声、インテント分類の信頼度しきい値を選択します (詳細については、「Lex V2 でのインテント信頼スコアを使用したインテント選択の改善」を参照してください)。
6. [記述的ボットビルダー - 生成 AI] で、作成するボットの説明を入力します。説明は、詳細かつ正確な内容とし、ボットのインテントを適切かつ十分に生成できるようにします。インテント作成プロセスを改善するためのアクションのリストを含めます。
7. [モデルを選択] で、モデルプロバイダーとモデルを選択します。
8. 別の口ケールでボットを作成するには、[別の言語を追加] を選択します。言語の追加が完了したら、[完了] を選択します。Amazon Lex V2 でボットを作成し、記述的ボットビルダーでボットのインテントとスロットを生成します。口ケールが生成されると、バナーは青から緑に変わります。[レビュー] を選択すると、生成されたインテントとスロットタイプが表示されます。

> **Note**
>
> 現在、記述的ボットビルダーは、英語口ケールでのみ利用可能です。ただし、ボットを作成した後で英語以外の口ケールにコピーすることはできます。

生成されたインテントとスロットタイプを確認してボットに追加する

1. ボットのユースケースに適用できるインテントとスロットタイプが十分にある場合は、生成されたインテントを確認できます。

a. 生成されたインテントを確認します。

   i. インテントの横のチェックボックスをオフにすると、ボットに追加するインテントのリストからインテントが削除されます。

   ii. インテント名を選択すると、そのインテント用に生成されたサンプル発話とスロットが表示されます。

   iii. デフォルトでは、すべての発話とスロットが選択されます。チェックボックスをオフにすると、その項目がインテントから削除されます。[選択内容に追加] を選択すると、チェックした項目がインテントに残ります。

b. 生成されたスロットタイプを確認します。

   i. スロットタイプの横にあるチェックボックスをオフにすると、ボットに追加するインテントのリストからスロットタイプが削除されます。

   ii. スロットタイプをボットに追加した後に、そのスロットタイプに値を追加できます。

2. インテントとスロットタイプを確認したら、ページ上部の [インテントとスロットタイプを追加] を選択して、インテントとスロットタイプをボットに追加します。

3. リソースの追加が完了すると、緑色の成功バナーが表示されます。生成されたインテントやスロットタイプを編集したり、値を追加したりするには、[インテント] と [スロットタイプ] に移動します。

4. 生成されたインテントと生成されたスロットタイプが、作成するボットにほとんど適用できない場合は、次の手順を実行します。

   a. [記述的ボットビルダーの詳細] セクションで、[新しい生成] を選択します。

   b. プロンプトを書き直して [再生成] を選択し、新しいインテントとスロットタイプを生成します。別のモデルを使用すると、結果は異なります。

> **Important**
>
> 同じインテントやスロットが生成される保証はありません。インテントやスロットタイプを再生成するたびに課金されます。

API

自然言語の説明を使用してボットを作成する

API を介して記述的ボットビルダーを使用すると、Amazon S3 バケットにボット定義が .zip ファイルとして作成されます。このファイルをダウンロードし、ボット定義を Amazon Lex V2 にインポートしてボットを作成します。

1. CreateBot リクエストを送信して、新しいボットを作成します。次に、CreateBotLocale リクエストを送信して、ボットロケールを作成します。
2. ボットの ID、バージョン、ロケールを指定して、StartBotResourceGeneration リクエストを送信します。ボットバージョンとして DRAFT を使用できます。generationInputPrompt フィールドにプロンプトを入力します。説明は、詳細かつ正確な内容とし、ボットのインテントを適切かつ十分に生成できるようにします。インテント作成プロセスを改善するためのアクションのリストを含めます。
3. レスポンス内の generationId を記録しておきます。
4. StartBotResourceGeneration レスポンスで受け取った generationId を使用して DescribeBotResourceGeneration リクエストを送信します。ボット ID、バージョン、ロケールを含めます。
5. DescribeBotResourceGeneration レスポンス内の generationStatus が Complete である場合、generatedBotLocaleUrl フィールドも入力されます。この Amazon S3 URI を使用し、「オブジェクトのダウンロード」の手順に従ってボット定義をダウンロードします。

生成されたボット定義を確認してインポートする

1. DescribeBotResourceGeneration レスポンス内の generationStatus から Amazon S3 URI を使用し、「オブジェクトのダウンロード」の手順に従ってボット定義をダウンロードします。
2. ファイルを編集することで、生成されたコンテンツをボットの特定のユースケースに合わせて直接変更できます。別の StartBotResourceGeneration リクエストを送信してインテントとスロットを再生成することもできます。

> **Important**
>
> 同じインテントやスロットが生成される保証はありません。インテントやスロットタイプを再生成するたびに課金されます。

3. ボット定義をインポートするには、「Lex V2 でのボットのインポート」の手順に従います。

4. インポート後に、UpdateIntent、UpdateSlot、および UpdateSlotType の各オペレーションを使用して、生成されたインテントとスロットを変更できます。

ボットロケールで生成されたすべての項目に関するメタデータを一覧表示するには、ListBotResourceGenerations オペレーションを使用します。DescribeBotResourcGeneration リクエストで返された generationId 値のいずれかを使用して、生成されたボット定義の Amazon S3 URI を取得します。

トピック

- 記述型ボットビルダーのボットの説明例
- Lex V2 で自然言語による説明でボットを作成するために必要な権限

記述型ボットビルダーのボットの説明例

Amazon Lex V2 の記述型ボットビルダーで使用できるボットの説明の例をいくつか示します。

業界	プロンプトの例
金融サービス	当社は、ユーザーが新しいカードを受け取ったときに、カードの有効化、暗証番号の Eメール送信または郵送、新しいカードの確認 (郵便番号を使用) などのタスクをユーザーが実行できるようにする金融カードサービスです。また、クレジットカードの特典に関する問い合わせ、紛失したカードの報告、新しいカードのリクエスト、カード暗証番号のリセット、カード請求書の支払いなど、既存のカードに関連するタスクを支援します。
フードサービス	お客様による食品の注文 (商品 ID、数量、サイズを使用)、注文状況の確認、注文のキャンセルを支援するボットが必要です。注文 ID は注文のインデックス作成に使用します。
航空会社	当社は、ユーザーが航空券の予約、予約の詳細の確認、予約したフライトの領収書の取得、フ

業界	プロンプトの例
	ライト状況の照会、予約したフライトのスケジュール変更、フライトの詳細の取得、予約したフライトのキャンセルを支援する航空会社ドメインです。ドメインの説明で機能のサポートに役立つ場合は、追加のインテントを生成することもできます。
保険	目的: 当社は自動車保険、住宅保険、年金保険を販売する保険会社です。請求状況の確認、請求の提出、保険金の支払い、保険契約のキャンセルができるボットが必要です。アカウントの識別と検証には policy_id と SSN の最後 4 桁を使用します。ボットには少なくとも次のインテントとスロットが必要です: 認証 - policy_id、SSN の最後の 4 桁。ポリシータイプ: 車、住宅、年金。ポリシーステータス: 残高確認、期日確認、補償範囲確認。支払い: 一括払い、分割払い、金額
車両管理	当社は、都市のドライバーが車の牽引先を見つけるのを支援する牽引車検索ボットを構築しています。このボットは、自動車がレッカー移動された元の場所の住所または場所、車のナンバープレートとメーカー、モデル、年式などの自動車の詳細を尋ねる必要があります。ボットは、牽引された車の駐車場の場所と営業時間を返信する必要があります。

業界	プロンプトの例
旅行	当社は旅行代理店で、お客様のディズニーへの旅行予約を支援するボットが必要です。ディズニーは世界中に複数のパークを展開しており、予約可能なホテル、レストラン、特別なエンターテイメントもあります。ボットのユーザーは、予約を変更またはキャンセルできる必要があります。予約には、少なくともパーク、日程、ホテルを含める必要があります。食事やエンターテイメントの追加はオプションで、後で追加または変更できます。

Lex V2 で自然言語による説明でボットを作成するために必要な権限

- この機能に Amazon Lex V2 コンソールからアクセスするには、コンソールのロールに `bedrock:ListFoundationModels` 権限と `bedrock:ListInferenceProfiles` 権限があることを確認してください。
- ボットに関連付けられた IAM ロールには、`bedrock:InvokeModel` 権限が必要です。この機能を Amazon Lex コンソールで有効にすると、Amazon Lex によって生成されたサービスリンクロールをボットで使用している場合、ポリシーがボットロールに自動的に追加されます。

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "bedrock:InvokeModel"
            ],
            "Resource": [
                "arn:aws:bedrock:us-east-1::foundation-model/model-id"
            ]
        }
    ]
}
```

発話生成を使用したインテント認識用のサンプル発話の生成

> **Note**
>
> 生成 AI 機能を利用するには、最初に以下の前提条件を満たす必要があります。
>
> 1. Amazon Bedrock の使用料金については、「Amazon Bedrock の料金」を参照してください。
> 2. ボットロケールで生成 AI 機能を有効にします。これを行うには、「生成 AI を使用した Lex V2 ボットの作成とパフォーマンスの最適化」の手順に従います。

発話の生成を使用して、インテントのサンプル発話の作成を自動化します。Amazon Lex V2 では、サンプル発話を手動で入力する代わりに、インテント名、説明、既存のサンプル発話に基づいてサンプル発話が生成されるため、独自のサンプル発話を見つけて作成するために費やす時間と労力を削減できます。Amazon Lex V2 で生成された発話は、編集または削除できます。このツールを使うと、インテント認識プロセス用のサンプル発話の作成を迅速化できます。

発話を生成できるようにするには、「生成 AI を使用した Lex V2 ボットの作成とパフォーマンスの最適化」の手順に従って生成 AI 機能を有効にします。

Amazon Lex V2 コンソールでこの機能にアクセスするには、コンソールロールに `bedrock:ListFoundationModels`、`bedrock:ListInferenceProfiles`、および アクセス`bedrock:InvokeModel`許可があることを確認します。

発話は、コンソールまたは API を使用して生成できます。

Console

1. ボットで任意のインテントの [サンプル発話] セクションに移動します (ビジュアル会話ビルダーの場合は、[開始] ブロックにあります)。
2. [発話を生成] ボタンを選択すると、5 つのサンプル発話が生成されます。インテントのサンプル発話が 25 個を超えると、[発話を生成] ボタンは無効になります。
3. 生成された発話には、生成された発話を既存の発話と区別する緑色のバナーが表示されます。
4. 発話にカーソルを合わせると、生成された発話を編集、削除、ソートするためのオプションが表示されます。

API

1. サンプル発話を生成するインテントとボット ID、バージョン、ロケールを入力して、GenerateBotElement リクエストを送信します。
2. レスポンスは SampleUtterance オブジェクトのリストを返します。各オブジェクトには生成された発話が含まれています。
3. 発話をインテントに追加するには、UpdateIntent リクエストを送信し、発話を `sampleUtterances` フィールドに追加します。

アシスト付きスロット解決を使用した Amazon Lex V2 のスロット値の明確化

> **Note**
>
> 生成 AI 機能を活用するには、最初に以下の前提条件を満たす必要があります。
>
> 1. Amazon Bedrock の使用料金については、「Amazon Bedrock の料金」を参照してください。
> 2. ボットロケールで生成 AI 機能を有効にします。これを行うには、「生成 AI を使用した Lex V2 ボットの作成とパフォーマンスの最適化」の手順に従います。

アシスト付きスロット解決を使用すると、ボットの会話フローで一部の組み込みスロットの精度を向上させることができます。アシスト付きスロット解決では、Amazon Bedrock の大規模言語モデル (LLM) を使用することで、一部の組み込みスロットの認識を向上させます。これにより、スロットの引き出しで得た顧客の応答をより正確に解釈できます。正常に解決できなかった発話の場合、Amazon Lex V2 は Amazon Bedrock を使用して 2 回目の解決を試みます。

アシスト付きスロット解決では、Amazon Bedrock の基盤モデルの機能を利用して、以下の組み込みスロットの精度を向上させることができます。

- `AMAZON.Alphanumeric` (正規表現のサポートなし)
- `AMAZON.City`
- `AMAZON.Country`
- `AMAZON.Date`

- AMAZON.Number
- AMAZON.PhoneNumber
- AMAZON.Confirmation

上記の組み込みスロットを使用するすべてのインテントで、アシスト付きスロット解決を有効にすることができます。アシスト付きスロット解決は、上記以外のカスタムスロットや Amazon 組み込みスロットには適用されません。

会話ログとメトリクスを使用して、Amazon Lex V2 ボットでアシスト付きスロット解決を有効にした後、精度の向上に関するデータを収集できます。

- 会話ログ - Amazon Bedrock をスロットの解決に使用した場合、解釈では interpretationSource が Bedrock になります。
- CloudWatch メトリクス - メトリクスは、CloudWatch メトリクスにリストされているディメンションの下に表示されます。詳細については、「Amazon CloudWatch による Amazon Lex のモニタリング」を参照してください。

記述的ボットビルダーを使用するには、「アシスト付きスロット解決に必要な Lex V2 のアクセス許可」の手順に従って IAM ロールに適切な権限があることを確認してください。

トピック

- Lex V2 で使用されるアシスト付きスロット解決の例
- 生成 AI 設定画面でアシスト付きスロット解決を有効にする
- Lex V2 のスロット設定でアシスト付きスロット解決を有効にする
- アシスト付きスロット解決に必要な Lex V2 のアクセス許可

Lex V2 で使用されるアシスト付きスロット解決の例

以下の例は、アシスト付きスロット解決を使用すると、ユーザーの発話を値にインテリジェントに変換できることを示しています。

AMAZON.Number

垂直	スロットタイプ	スロット名	スロットプロンプト	utterance	解決された値
旅行	AMAZON.Number	numberOfNightsStayed	旅行で何泊しましたか?	丸 1 週間、7 泊。	7
銀行	AMAZON.Number	numberOfPeopleOnTheAccount	アカウントの人数を教えてください。	私と妻。	2
旅行	AMAZON.Number	numberOfStops	途中の立ち寄り先の数を教えてください。	日本で 1 回。ロサンゼルスで 1 回。	2

AMAZON.AlphaNumeric

垂直	スロットタイプ	スロット名	スロットプロンプト	utterance	解決された値
レンタカー	AMAZON.AlphaNumeric	transactionId	トランザクション ID を教えてください。	アルファ ウィスキー エコー 8349 ロミオ ジュリエットだったと思います	AWE8349RJ
旅行	AMAZON.AlphaNumeric	confirmationCode	予約の確認番号を教えてください。	確認番号は BLT2UE です。	BLT2UE

AMAZON.Date

垂直	スロットタイプ	スロット名	スロットプロンプト	utterance	解決された値	currentDate
レンタカー	AMAZON.Date	dueDate	レンタル契約の期限はいつですか?	リースは来月 1 日に期限切れになります。	2023-12-01	2023-11-09
旅行	AMAZON.Date	returnDate	いつ戻りますか?	今日午後 7 時頃。	2023-11-09	2023-11-09

AMAZON.PhoneNumber

垂直	スロットタイプ	スロット名	スロットプロンプト	utterance	解決された値
保険	AMAZON.PhoneNumber	policyHolder	保険契約者の電話番号は何番ですか?	保険契約者の電話番号は 123-456-7890 です。	1234567890
リテール	AMAZON.PhoneNumber	phoneLookup	アカウントを検索できるように、電話番号を教えてください。	413-570-9617 だと思います。再確認させてください。	4135709617

AMAZON.Country

垂直	スロットタイプ	スロット名	スロットプロンプト	utterance	解決された値
旅行	AMAZON.Country	nativeCountry	出生国はどこですか?	私はインド人です。	インド

垂直	スロットタイプ	スロット名	スロットプロンプト	utterance	解決された値
銀行	AMAZON.Country	countryItinerary	デビットカードを使用して、どちらの国に旅行しますか?	ニューデリーに旅行します。	インド

AMAZON.City

垂直	スロットタイプ	Intent	質問	応答	解決された値
保険	AMAZON.City	policyHolderCity	保険契約者はどの都市に居住していますか?	私はスプリングフィールドに住んでいます。	スプリングフィールド
旅行	AMAZON.City	destinationCity	どちらの都市に旅行しますか?	東京に旅行します。	東京

AMAZON.Confirmation

垂直	スロットタイプ	スロット名	スロットプロンプト	utterance	解決された値
保険	AMAZON.Confirmation	policyExpired	保険契約の有効期限は切れていますか?	はい、残念ながら切れています。	はい
銀行	AMAZON.Confirmation	hasInvestments	何かに投資していますか?	まだ何にも投資していません。	いいえ

生成 AI 設定画面でアシスト付きスロット解決を有効にする

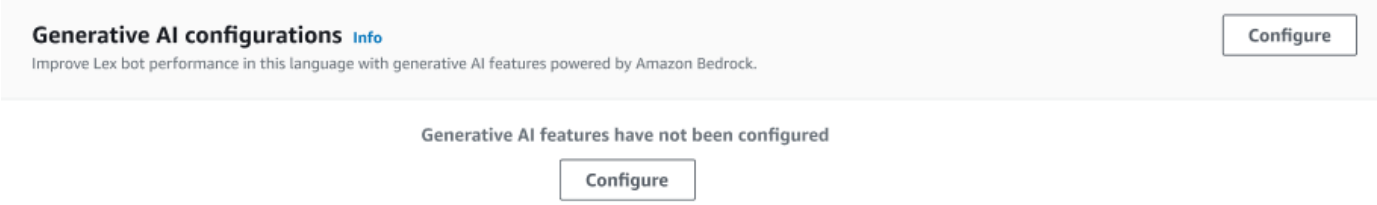
生成 AI 画面に移動すると、サポートされている組み込みスロットのアシスト付きスロット解決を有効にすることができます。

サポートされている組み込みスロットである場合は、スロットレベルでアシスト付きスロット解決を有効にするオプションが表示されます。

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/lexv2/home で Amazon Lex V2 コンソールを開きます。
2. ナビゲーションペインの [ボット] で、アシスト付きスロット解決で使用するボットを選択します。
3. 有効にするボットの言語として [英語 (米国)] を選択します。
4. 画面の [生成 AI 設定] セクションに移動します。
5. [Amazon Bedrock に移動] を選択してサインアップし、この機能を有効にします (まだ有効にしていない場合)。

> **Note**
>
> Amazon Bedrock の基盤モデルにアクセスできない場合は、[Amazon Bedrock に移動] を確認してください。[Amazon Bedrockに移動] をクリックして Amazon Bedrock ページに移動すると、サインアップして基盤モデルにアクセスできます。現在、アシスト付きスロット解決は Anthropic Claude をサポートしています。最良の結果を得るには、Anthropic Claude を使用することをお勧めします。

6. Amazon Bedrock の基盤モデルに既にアクセスできる場合は、[設定] ボタンが表示されます。このボタンをクリックして生成 AI 設定ページに移動し、Lex で生成 AI 機能を有効にします。



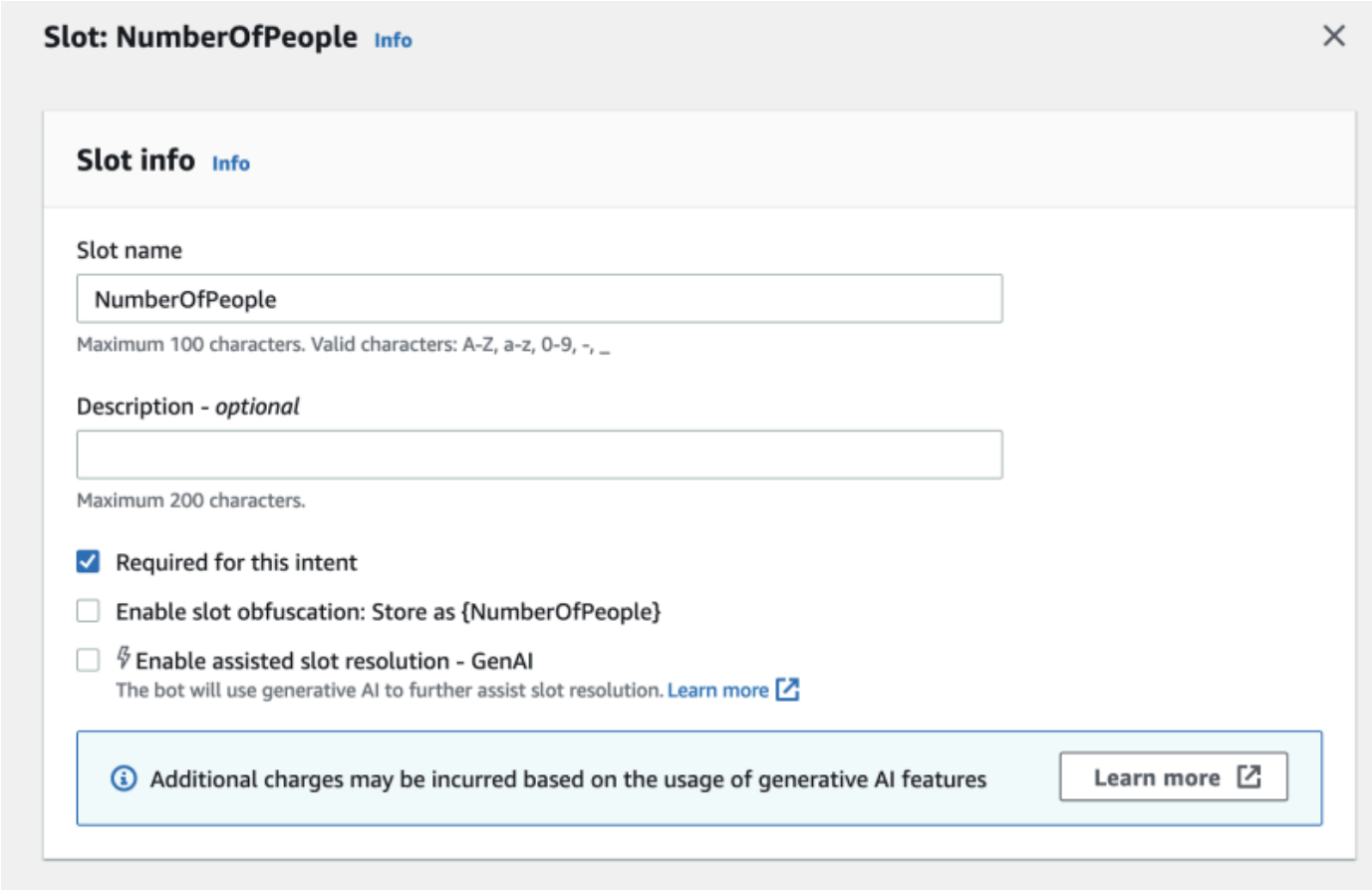
7. ボックスの右上隅にあるスライダーを右に動かして、[有効] 設定を選択します。
8. [有効化] ボタンを選択して、選択したスロットのアシスト付きスロット解決を有効にします。
9. アシスト付きスロット解決を無効にする場合は、リストからスロットを選択して [無効化] ボタンを選択します。

Lex V2 のスロット設定でアシスト付きスロット解決を有効にする

スロットが含まれている各インテントのスロットレベルに移動することで、サポートされている組み込みスロットのアシスト付きスロット解決を有効にすることができます。アシスト付きスロット解決を有効にするオプションを使用するには、スロットが前述のサポートされている組み込みスロットのいずれかである必要があります。アシスト付きスロット解決を有効にするオプションがスロットにない場合、オプションはグレー表示されます。

> **Note**
>
> 各スロットでアシスト付きスロット解決機能を有効にするには、まず生成 AI パネルでこの機能を有効にする必要があります。

1. AWS マネジメントコンソールにサインインし、Amazon Lex V2 コンソール (https://console.aws.amazon.com/lexv2/home) を開きます。
2. ナビゲーションペインの [ボット] で、アシスト付きスロット解決で使用するボットを選択します。
3. [すべての言語] で [英語 (米国)] を選択してリストを展開します。
4. 左側のパネルで [インテント] を選択し、選択したボットのインテントを一覧表示します。
5. [インテント] 画面で、変更するスロットが含まれているインテントを選択します。
6. インテント名を選択すると、インテントのスロットが表示されます。
7. [スロット] セクションで、[詳細オプション] ボタンを選択します。
8. [アシスト付きスロット解決を有効にする] チェックボックスをオンにして、この機能を有効にします。



9. 画面の右下隅にある [スロットを更新] ボタンを選択します。これにより、選択したスロットのアシスト付きスロット解決が有効になります。

API コールを行うことで、サポートされている組み込みスロットのアシスト付きスロット解決を有効にすることができます。

- 「生成 AI を使用した Lex V2 ボットの作成とパフォーマンスの最適化」の手順に従って、ボットロケールでアシスト付きスロット解決を有効にします。
- アシスト付きスロット解決を有効にするスロットを指定して、UpdateSlot リクエストを送信します。[slotResolutionSetting] フィールドで、slotResolutionStrategy の値を EnhancedFallback に設定します。アシスト付きスロット解決を有効にして新しいスロットを作成するには、代わりに CreateSlot リクエストを送信します。

アシスト付きスロット解決に必要な Lex V2 のアクセス許可

- この機能に Amazon Lex V2 コンソールからアクセスするには、コンソールのロールに bedrock:ListFoundationModels 権限と bedrock:ListInferenceProfiles 権限があることを確認してください。
- ボットに関連付けられた IAM ロールには、bedrock:InvokeModel 権限が必要です。Amazon Lex V2 コンソールでこの機能を有効にすると、ボットが Amazon Lex V2 によって生成されたサービスにリンクされたロールを使用している場合、ポリシーがボットロールに自動的に追加されます。

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "bedrock:InvokeModel"
            ],
            "Resource": [
                "arn:aws:bedrock:us-east-1::foundation-model/modelId"
            ]
        }
    ]
}
```

BedrockAgentIntent を使用した、Amazon Lex V2 での Amazon Bedrock エージェントの使用

> **Note**
>
> 生成 AI 機能を活用するには、最初に以下の前提条件を満たす必要があります。
>
> 1. Amazon Bedrock の使用料金については、「Amazon Bedrock の料金」を参照してください。

2. ボットロケールで生成 AI 機能を有効にします。これを行うには、「生成 AI を使用した Lex V2 ボットの作成とパフォーマンスの最適化」の手順に従います。

Amazon Bedrock エージェントを利用すると、包括的なタスク定義プロセスを経ることなく、顧客が要求する複雑なワークロードを処理できます。Amazon Lex V2 には、ボットに追加できる組み込み `AMAZON.BedrockAgentIntent` が用意されています。このインテントは、顧客のリクエストを認識、分析、推論し、最終的に応答することで、Amazon Bedrock の生成 AI 機能を活用します。また、必要なタスク (顧客の注文ステータスをチェックできる小売エージェントを定義したイメージなど) を達成するために、フォローアップの質問をすることもできます。顧客が注文ステータスを尋ねると、エージェントはまず `customerId` または関連する `emailId`をリクエストして詳細を取得し、最後に正しい注文ステータスで応答します)。また、AMAZON.BedrockAgentIntent を Bedrock ナレッジベースと統合して、顧客のクエリに直接回答することもできます。

「AMAZON.BedrockAgentIntent のアクセス許可」の手順に従って、IAM ロールに AMAZON.BedrockAgentIntent にアクセスするための適切なアクセス許可があることを確認します。

AMAZON.BedrockAgentIntent を利用するには、以下のいずれかのナレッジストアを設定しておく必要があります。

- Amazon Bedrock エージェント – 詳細については、「Bedrock エージェントの作成」を参照してください。
- Amazon Bedrock ナレッジベース — 詳細については、「ナレッジベースの構築」を参照してください。

AMAZON.BedrockAgentIntent を使用するには、「Bedrock エージェントインテント向けの Lex V2 で必要なアクセス許可」の手順に従って、IAM ロールに適切なアクセス許可を持たせます。

トピック

- 生成 AI 設定画面で Bedrock エージェントインテントを有効にする
- 組み込みインテントをボットに追加して Bedrock エージェントインテントを有効にする
- Bedrock エージェントインテント向けの Lex V2でに必要なアクセス許可
- セッション属性を含むサンプルリクエスト

生成 AI 設定画面で Bedrock エージェントインテントを有効にする

Bedrock エージェントインテントを有効にするには、生成 AI 画面に移動します。

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/lexv2/home で Amazon Lex V2 コンソールを開きます。
2. ナビゲーションペインの [ボット] のナビゲーションペインで、Bedrock エージェントインテントに使用するボットを選択します。
3. 有効にするボットの言語を選択します。
4. 画面の [生成 AI 設定] セクションに移動します。
5. [BedrockAgentIntent 設定] セクションで、[BedrockAgent インテントを作成] を選択します。

組み込みインテントをボットに追加して Bedrock エージェントインテントを有効にする

Amazon Lex V2 ボットに組み込みインテントを追加することで、Bedrock エージェントインテントを有効にできます。

> **Note**
>
> 個々のボッドで Bedrock エージェントインテント機能をアクティブ化するには、まず生成 AI パネルでこの機能をアクティブ化する必要があります。

1. AWS マネジメントコンソールにサインインし、Amazon Lex V2 コンソール (https://console.aws.amazon.com/lexv2/home) を開きます。
2. ナビゲーションペインの [ボット] で、Bedrock Agent インテントに使用するボットを選択します。
3. [すべての言語] で [英語 (米国)] を選択してリストを展開します。
4. [インテントを追加] を選択し、ドロップダウンメニューから [組み込みインテントを使用] を選択します。
5. AMAZON.BedrockAgentIntent の設定の詳細については、「AMAZON.BedrockAgentIntent」を参照してください。

Bedrock エージェントインテント向けの Lex V2でに必要なアクセス許可

- この機能に Amazon Lex V2 コンソールからアクセスするには、コンソールのロールに bedrock:ListFoundationModels 権限と bedrock:ListInferenceProfiles 権限があることを確認してください。
- ボットに関連付けられた IAM ロールには、AMAZON.BedrockAgentIntent に必要なアクセス権を持たせる必要があります。

  インテントで Amazon Bedrock ナレッジベースを使用している場合、Amazon Bedrock エージェント、および bedrock:InvokeModel と bedrock:RetrieveAndGenerate を使用している場合は、ボットロールに bedrock:InvokeAgent を呼び出すためのアクセス許可を持たせる必要があります。また、ボットの AMAZON.BedrockAgentIntent で指定したエージェントごとにステートメントをアタッチする必要があります (以下のポリシーの「Amazon Bedrock エージェントへのアクセス許可」または「Amazon Bedrock ナレッジベースへのアクセス許可」ステートメントを参照してください)。

  この機能を Amazon Lex コンソールで有効にすると、Amazon Lex によって生成されたサービスリンクロールをボットで使用している場合、ポリシーがボットロールに自動的に追加されます。

セッション属性を含むサンプルリクエスト

次の例は、AMAZON.BedrockAgentIntent を呼び出す方法と、レスポンスに入力されるセッションおよびリクエスト属性を示しています。これらの属性には Bedrock エージェントのレスポンスデータが含まれており、エージェントの出力、Amazon Bedrock ナレッジベースソース、アクショングループの呼び出しの詳細にアクセスするために使用できます。

```
{
    "sessionId": "user-session-123",
    "messages": [{
        "content": "Your order #12345 is currently being processed and will ship within
 2-3 business days. You will receive a tracking number via email once it ships.",
        "contentType": "PlainText"
    }],
    "sessionState": {
        "sessionAttributes": {
            "x-amz-lex:bedrock-agent-search-response": "Your order #12345 is currently
 being processed and will ship within 2-3 business days. You will receive a tracking
 number via email once it ships.",
```

```
            "x-amz-lex:bedrock-knowledge-base-search-response-source": "[{\"title\":
 \"Order Processing Guide\", \"uri\": \"s3://knowledge-base/orders/processing.pdf\",
 \"excerpt\": \"Standard orders typically ship within 2-3 business days...\"}]",
            "x-amz-lex:bedrock-agent-action-group-invocation-input":
 "{\"actionGroupName\": \"OrderLookup\", \"function\": \"getOrderStatus\", \"parameters
\": {\"orderId\": \"12345\", \"customerId\": \"67890\"}}",
            "x-amz-lex:bedrock-agent-knowledge-base-lookup-input": "{\"knowledgeBaseId
\": \"KB123456\", \"query\": \"order status processing time\", \"numberOfResults\": 3}"
        },
        "intent": {
            "name": "BedrockAgentIntent",
            "slots": {},
            "state": "Fulfilled",
            "confirmationState": "None"
        },
        "dialogAction": {
            "type": "ElicitIntent"
        }
    },
    "interpretations": [{
        "intent": {
            "name": "FallbackIntent",
            "slots": {}
        },
        "interpretationSource": "Lex"
    }],
    "requestAttributes": {
        "x-amz-lex:channels:platform": "Web",
        "x-amz-lex:accept-content-types": "PlainText",
        "x-amz-lex:bedrock-agent-search-response": "Your order #12345 is currently
 being processed and will ship within 2-3 business days. You will receive a tracking
 number via email once it ships.",
        "x-amz-lex:bedrock-knowledge-base-search-response-source": "[{\"title\":
 \"Order Processing Guide\", \"uri\": \"s3://knowledge-base/orders/processing.pdf\",
 \"excerpt\": \"Standard orders typically ship within 2-3 business days...\"}]",
        "x-amz-lex:bedrock-agent-action-group-invocation-input": "{\"actionGroupName
\": \"OrderLookup\", \"function\": \"getOrderStatus\", \"parameters\": {\"orderId\":
 \"12345\", \"customerId\": \"67890\"}}",
        "x-amz-lex:bedrock-agent-knowledge-base-lookup-input": "{\"knowledgeBaseId\":
 \"KB123456\", \"query\": \"order status processing time\", \"numberOfResults\": 3}"
    }
}
```

この例では、セッション属性は、BedrockAgentIntent がエージェントの回答、使用された Amazon Bedrock ナレッジベースのソース、アクショングループの呼び出し、およびレスポンスの生成に使用された Amazon Bedrock ナレッジベースのルックアップの詳細を含むレスポンスデータを入力する方法を示しています。

詳細については、「AMAZON.BedrockAgentIntent」を参照してください。

アシスト付き NLU を使用して Lex V2 のインテント分類とスロット解決を改善する

アシスト付き NLU は、大規模言語モデル (LLMs) を使用して Amazon Lex V2 のインテント分類およびスロット解決機能を改善する機能です。これにより、ボットが設定したインテントおよびスロット内に留まりながら、精度が向上します。この機能はボットコンテンツを生成または変更しません。この機能により、NLU システムの全体的な精度が向上し、ユーザーにとってよりシームレスで効果的な会話体験が得られます。

アシスト付き NLU 機能は、英語、スペイン語、ポルトガル語、カタルーニャ語、フランス語、イタリア語、ドイツ語のロケールで利用できます。具体的には、、、 `pt_` (pt_BR、pt_PT)`es_`、`ca_` (ca_ES)`en_`、 `fr_` (fr_CA、fr_FR)、 `it_` (it_IT)、 `de_` (de_AT、de_DE)、 `zh_` (zh_CN、zh_HK)`ja_JP`、、および で始まるロケールをサポートしますko_KR。サポートされているロケールの完全なリストについては、「Amazon Lex V2 でサポートされている言語と ロケール」の表を参照してください。

アシスト付き NLU を使用して、インテント分類とスロット解決を改善します。Amazon Lex V2 は Amazon Bedrock モデルを呼び出して、インテントの分類とボットのユースケースに合ったスロットタイプの解決を支援します。コンソールを使用して、ボットのアシスト付き NLU を有効にできます。

アシスト NLU モード

プライマリモードでは、Lex はデフォルトで LLM をユーザー入力を処理する主要な手段として使用してユーザーのインテントを決定し、スロット値を埋めます。

フォールバックモードでは、Lex は LLM を使用して、NLU によって決定された信頼スコアが設定されたしきい値より低いかどうかを判断したり、FallbackIntentにルーティングしたりします。また、従来の NLU が値をキャプチャしない場合、ユーザー入力からスロット値を決定します。

Console

Amazon Lex V2 ボットでのアシスト付き NLU の使用

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/lexv2/home で Amazon Lex V2 コンソールを開きます。
2. [ボット] ページで、アシスト付き NLU で使用するボットを選択します。
3. [ボットロケール] ページで、[アシスト付き NLU] セクションの [設定] をクリックします。
4. [ランタイム生成 AI 機能] セクションに、アシスト付き NLU 機能が表示されます。横のトグルボタンを使用して、LLM アシスト NLU 機能を有効にします。その後、プライマリモードまたはフォールバックモードを選択し、保存をクリックします。
5. [ボットロケール] ページの [アシスト付き NLU] セクションで、LLM アシスト付き NLU 機能が有効になっていることを確認します。
6. ボットを構築して、ランタイムで変更がボットに反映されていることを確認します。
7. ボットの構築が完了したら、コンソールのテストパネルを使用するか、LLM アシスト付き NLU 機能を有効にした後にテストセットを実行して、改善点を確認できます。

LLM アシスト付き NLU 機能を使用する際のボットの精度を改善するためのガイダンス

以下のベストプラクティスは、アシスト付き NLU 機能の効果を最大化するのに役立ちます。

1. インテント名を説明不要の名前にする — インテントのアクションや目的がすぐに伝わる名前を使用します。例えば、フライトを予約するインテントを作成している場合は、「BookFlight」という名前にします。
2. 名前をクリーンでシンプルに保つ — インテント名とスロット名にプレフィックス、サフィックス、または不要な単語を追加しないようにします。「Dev」や「Test」などの追加要素は LLM を混乱させ、目的を不明確にする可能性があります。
3. 詳細な説明を提供する — カスタムなインテントとスロットごとに、簡潔で有益な説明を含めます。これにより、特定の用途とコンテキストを説明し、人間と LLM の両方がその目的を理解しやすくなります。

> **Note**
>
> この機能を有効にすると、データが AWS リージョン間で処理される場合があります。クロスリージョン推論の詳細については、「https://docs.aws.amazon.com/bedrock/latest/userguide/cross-region-inference.html」を参照してください。

> **Important**
>
> ボットのドラフトバージョンでこの機能を有効にします。本番稼働用エイリアスで使用する前に、機能をテストします。

アシスト付き NLU の無効化

アシスト付き NLU 機能を無効にするには、次の手順に従います。

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/lexv2/home で Amazon Lex V2 コンソールを開きます。
2. [ボット] ページで、ボットを選択します。
3. [ボットロケール] ページで、[アシスト付き NLU] セクションの [設定] をクリックします。
4. [ランタイム生成 AI 機能] セクションで、[アシスト付き NLU] 機能をオフにし、[保存] をクリックします。
5. ボットを構築して変更を適用します。

意図的な曖昧さを解消して、あいまいなユーザー入力を解決する

インテントの曖昧さの排除は、複数のインテントが一致する可能性がある場合にあいまいなユーザー入力を解決するのに役立つ Assisted NLU の改善です。有効にすると、システムはユーザーに明確な質問を提示し、会話の精度を向上させるために正確なインテントを指定するのに役立ちます。システムは、インテント名と説明をコンテキストとして分析する大規模言語モデル (LLM) を使用し、ユーザーの発話のあいまいさに基づいて、一致する可能性が最も高いインテントを返します。LLM は、ユーザー入力が 1 つのインテントまたは複数のインテントに明確に一致しているかどうかを評価し、あいまいさを排除するのに十分な曖昧さがあるかどうかを評価し、候補インテントを提供します。

意図的なあいまいさの排除機能は、英語、スペイン語、ポルトガル語、カタルーニャ語、フランス語、イタリア語、ドイツ語、中国語、日本語、韓国語の口ケールで利用できます。具体的には、、、 pt_ (pt_BR、pt_PT)es_、 ca_ (ca_ES)en_、 fr_ (fr_CA、fr_FR)、 it_ (it_IT)、 de_ (de_AT、de_DE)、 zh_ (zh_CN、zh_HK)ja_JP、、および で始まる口ケールをサポートしますko_KR。サポートされている口ケールの完全なリストについては、「Amazon Lex V2 でサポートされている言語と口ケール」の表を参照してください。

意図的なあいまいさの排除には、次のオプションを設定できます。

インテントオプションの数

曖昧さを解消する必要がある場合にユーザーに提示するインテントの最大数 (2～5) を設定します。この設定は、システムがあいまいな入力を検出したときにユーザーに表示されるインテントオプションの数を決定します。デフォルト値は 3 です。これにより、選択を管理可能な状態を維持しながら、ユーザーに十分なオプションを与えることができます。

あいまいさの排除メッセージ

曖昧さを解消するオプションをユーザーに提示する前に表示されるカスタムメッセージを指定します。このメッセージはユーザーのコンテキストの設定に役立ち、ボットのトーンとブランドに合わせてカスタマイズできます。指定しない場合、デフォルトのメッセージが使用されます。

インテント表示名

曖昧さを解消するために、インテントにわかりやすい表示名を設定します。これは、インテント名が技術的であるか、エンドユーザーへの表示に適していない場合に推奨されます。表示名は、技術的なインテント名ではなく、あいまいさを解消する際にユーザーに表示されます。

Console

Amazon Lex V2 ボットでの意図的なあいまいさの排除の使用

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/lexv2/home で Amazon Lex V2 コンソールを開きます。
2. Bots ページで、意図的な曖昧さの排除で使用するボットを選択します。
3. [ボットロケール] ページで、[アシスト付き NLU] セクションの [設定] をクリックします。
4. アシスト NLU を有効にし、プライマリモードまたはフォールバックモードを選択します (意図的なあいまいさの排除は両方のモードで機能します)。
5. アシスト NLU 設定の「意図的なあいまいさの排除」セクションで、トグルボタンを使用して意図的なあいまいさの排除機能を有効にします。
6. 以下のオプション設定を構成します。
   - インテントオプションの数: 曖昧さの解消中にユーザーに提示するインテントの最大数 (2～5) を選択します。デフォルトは 3 です。
   - あいまいさの排除メッセージ: インテントオプションを表示するときに表示されるカスタムメッセージを指定します。指定しない場合、デフォルトのメッセージが使用されます。
7. 保存をクリックして設定を適用します。

8. オプションで、ユーザーエクスペリエンスを向上させるためにインテント表示名を設定します。

   a. 設定するボット内の各インテントに移動します。

   b. インテントエディタページで、表示名フィールドを見つけます。

   c. インテント名ではなく、あいまいさを解消する際にユーザーに表示されるわかりやすい名前を入力します。

9. ボットを構築して、実行時にボットに反映された変更を確認します。

意図的な曖昧さの排除機能を使用する際にボットの有効性を向上させるためのガイダンス

以下のベストプラクティスは、意図的なあいまいさの排除機能の効果を最大化するのに役立ちます。

1. Clear Intent Names and Descriptions: インテント名と説明がクリーンでクリアであり、他のインテントと重複しないようにします。これらは、曖昧さを解消するために LLM に提供される主な入力です。
2. 説明表示名: 現在のインテント名が技術的な場合は、インテントの目的を明確に伝える説明表示名を使用します。表示名は、インテント名に似ているか、一致している必要があります。
3. 適切な最大インテント: 好みとテストに基づいてインテントオプションの最大数を設定します。
4. カスタムメッセージ: ユーザーの入力を確認し、インテントオプションにつながる簡潔なあいまいさのないメッセージを作成します。
5. テストシナリオ: あいまいな発話でテストして、あいまいさのないエクスペリエンスがインテント名またはインテント表示名とカスタムメッセージで自然に感じられるようにし、あいまいさのないプロンプト中に正しいインテントオプションが表示されていることを確認します。

> **Important**
>
> ボットのドラフトバージョンでこの機能を有効にします。本番稼働用エイリアスで使用する前に、機能をテストします。

インテントの曖昧さの解消

意図的なあいまいさの排除機能を無効にするには、次の手順に従います。

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/lexv2/home で Amazon Lex V2 コンソールを開きます。
2. [ボット] ページで、ボットを選択します。
3. [ボットロケール] ページで、[アシスト付き NLU] セクションの [設定] をクリックします。
4. アシスト NLU 設定の「意図的なあいまいさの解除」セクションで、意図的なあいまいさの解除機能をオフにして、保存をクリックします。
5. ボットを構築して変更を適用します。

AI を活用した Bot Analyzer を使用してボットを最適化する

AI を活用した推奨事項を使用して、AWS のベストプラクティスに照らして Amazon Lex V2 ボット設定を分析します。Bot Analyzer は Amazon Bedrock の生成 AI 機能を使用して設定の問題を特定し、インテント分類とスロット解決のパフォーマンスを向上させるための実用的なガイダンスを提供します。

Bot Analyzer はボットのインテント設定を自動的に評価し、以下の推奨事項を提供します。

- インテントの分離の向上 - 複数の概念をグループ化する一般的なインテントを特定して解決する
- インテントの重複を排除する - ルーティングエラーの原因となるインテント間の同様の意味やフレーズを検出する
- スロットの使用を最適化する - スロットを適切に使用して、同様のインテントを組み合わせてエンティティ抽出を改善することをお勧めします
- 発話品質の向上 - サンプル発話カバレッジと多様性を分析してインテント分類を改善

前提条件

Bot Analyzer を使用する前に、以下を確認してください。

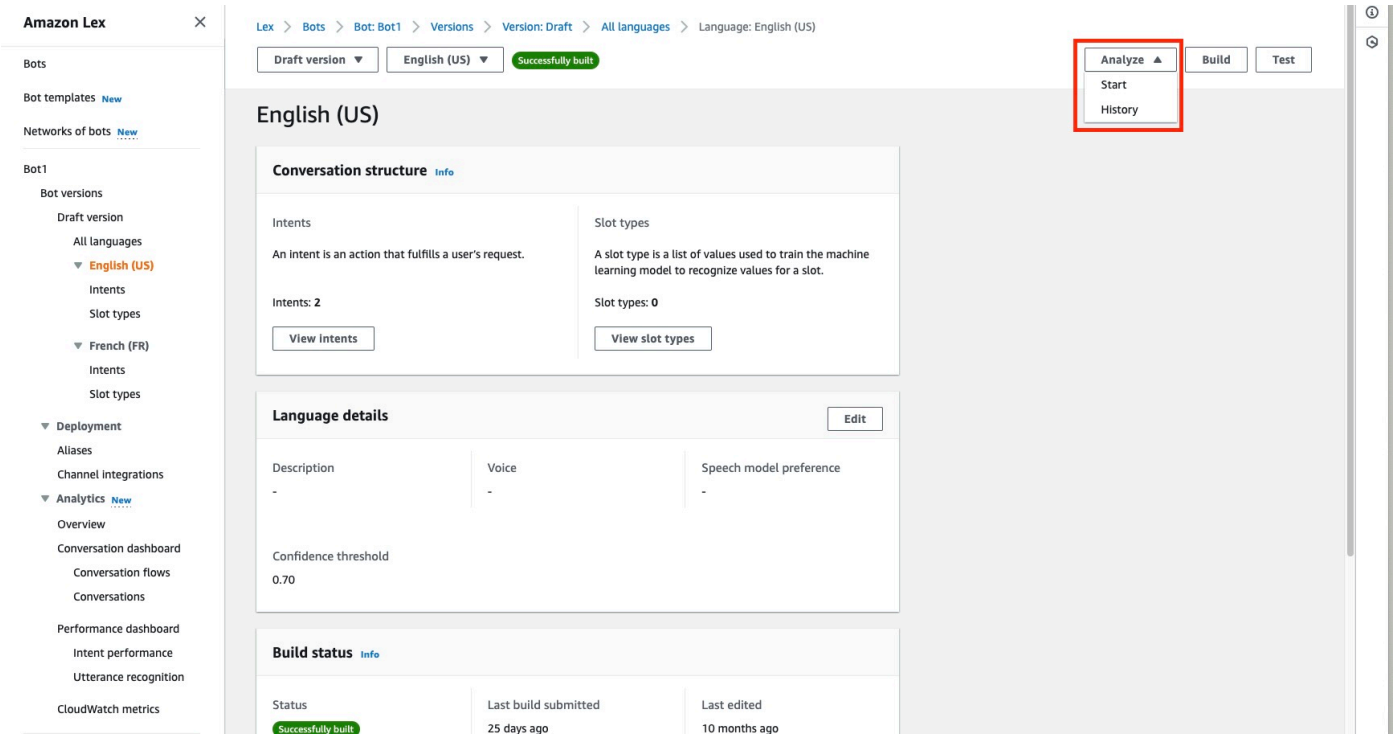
- ボットロケールが正常に構築されました
- 分析用のボットのバージョンは です。 `DRAFT`
- ボットロケールは、サポートされている英語ロケールの 1 つで す。`en_AU`、`en_GB`、`en_IN`、`en_US`、 `en_ZA`

Bot Analyzer は、コンソールまたは API で使用できます。

Console

分析を開始する

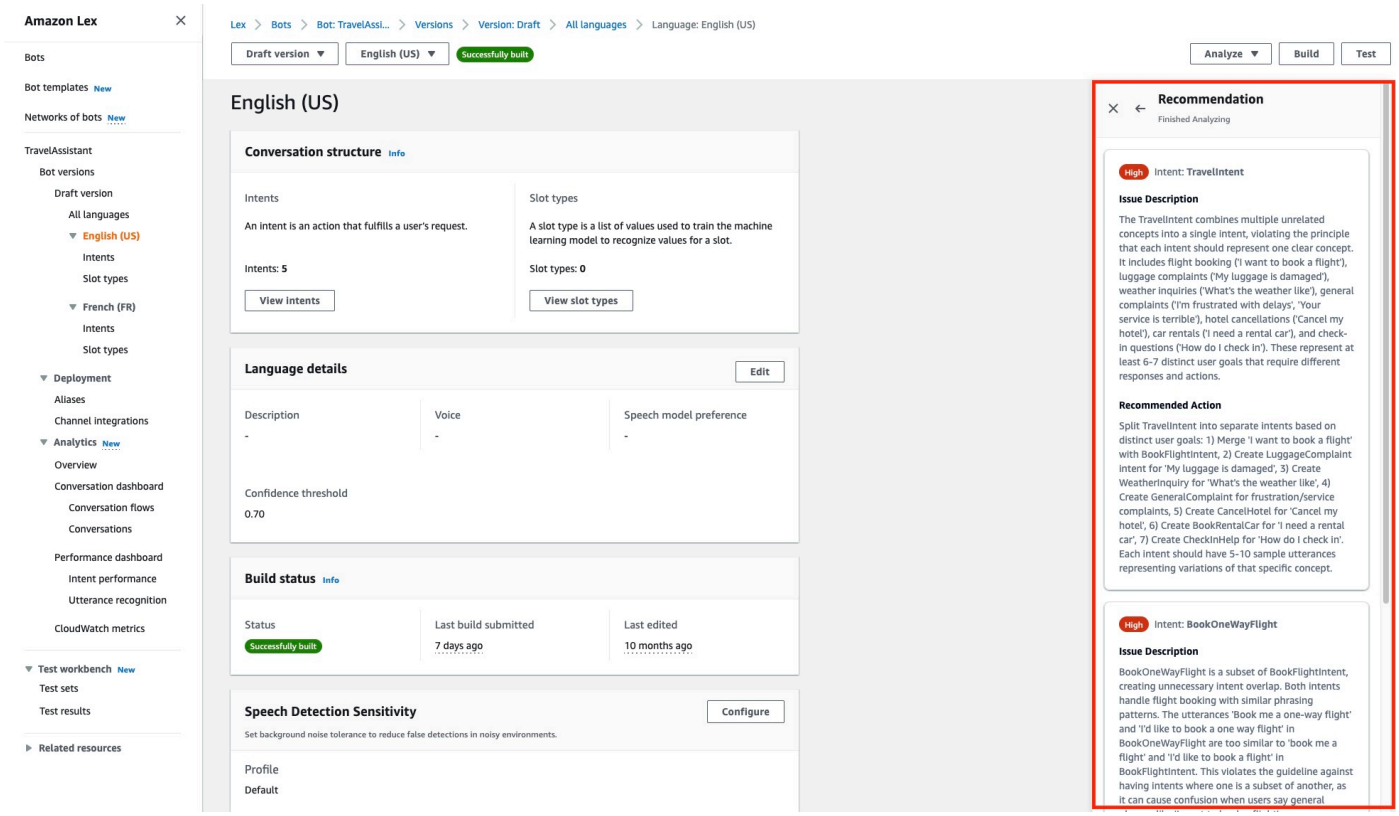
1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/lexv2/home で Amazon Lex V2 コンソールを開きます。
2. ボットを選択し、分析するボットロケールに移動します。
3. ボットロケールエディタで、分析ドロップダウンメニューをクリックします。
4. 開始を選択して分析を開始します。



通常、分析は数分以内に完了します。分析中、キャンセルする必要がある場合、開始ボタンは分析を停止に変わります。

レコメンデーションを表示する

分析が完了すると、右側のレコメンデーションパネルにレコメンデーションが表示されます。



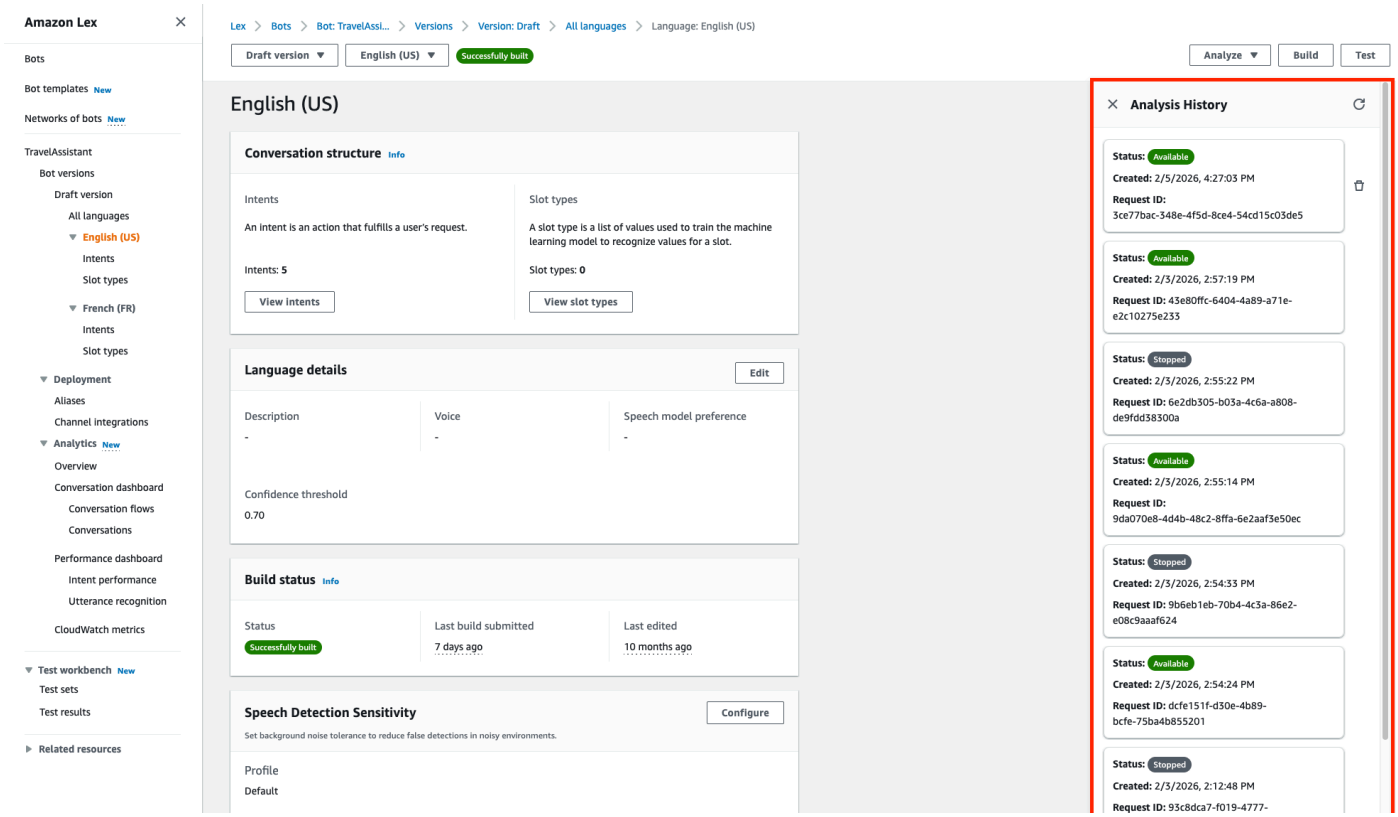
各レコメンデーションには以下が含まれます。

- Priority - 重要度が高、中、または低
- 問題の場所 - 影響を受ける特定のインテント
- 問題の説明 - 検出された設定の問題
- 修正案 - 問題を解決するための実用的なステップ

分析履歴の表示

以前の分析を表示するには:

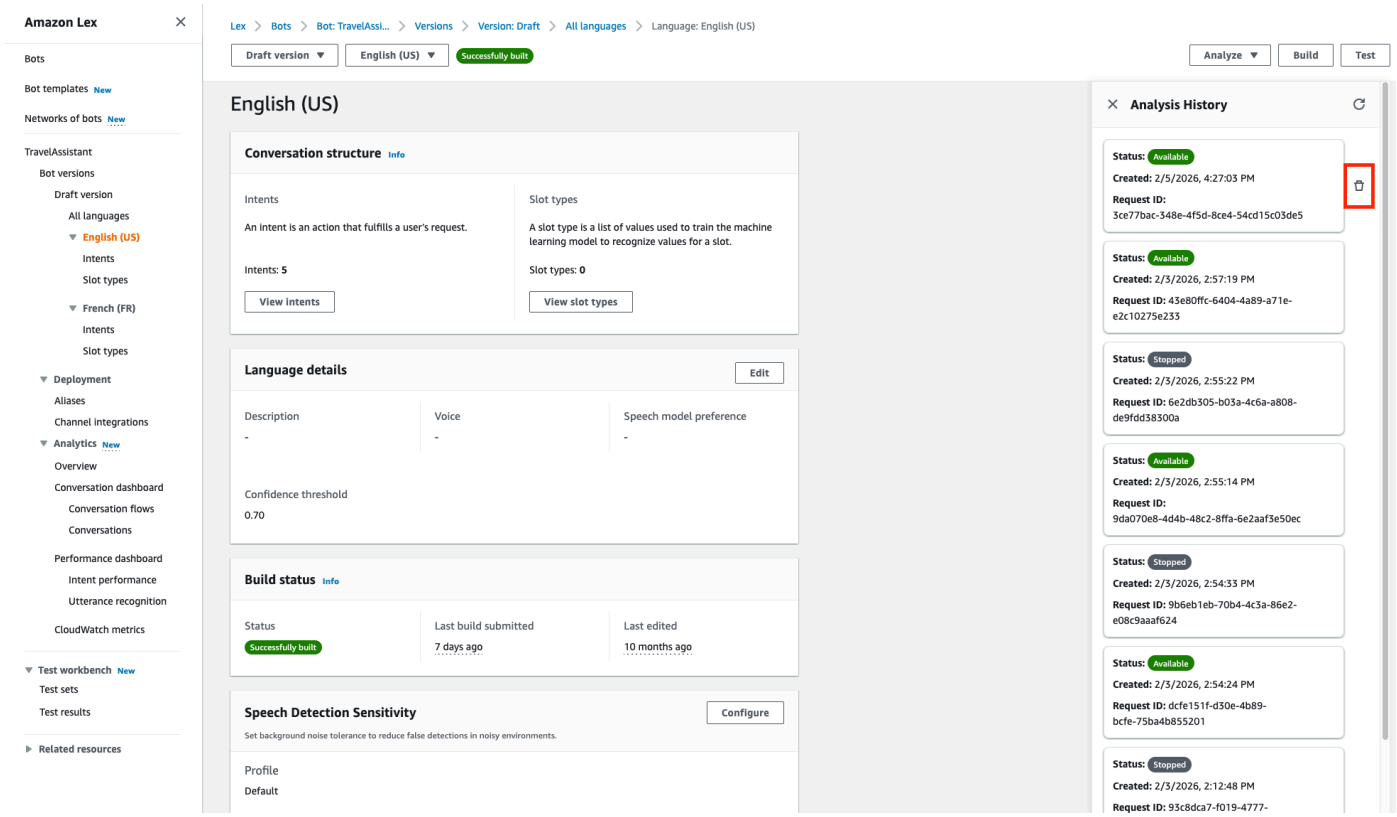
1. 分析ドロップダウンメニューをクリックします。
2. 履歴を選択します。
3. 分析履歴パネルには、過去の分析リクエストのステータスとタイムスタンプが表示されます。



推奨事項の削除

分析結果を削除するには:

1. 分析ドロップダウンメニューをクリックします。
2. [削除] を選択します。
3. 現在のレコメンデーションの削除を確認します。



API

分析を開始する

ボットロケールの分析を開始するStartBotAnalyzerリクエストを送信します。レスポンスは、 を含む HTTP 202 ステータスを返しますbotAnalyzerRequestId。この ID を書き留めておくと、分析ステータスを確認してレコメンデーションを取得するために必要になります。

分析ステータスを確認し、レコメンデーションを取得する

前のステップbotAnalyzerRequestIdの を使用してDescribeBotAnalyzerRecommendationリクエストを送信します。リクエストパスbotIdに を含めます。

レスポンスbotAnalyzerStatusの が の場合Available、分析は完了し、botAnalyzerRecommendationListフィールドにレコメンデーションが入力されます。各レコメンデーションには以下が含まれます。

- issueLocation - 問題が検出された場所
- priority - 重要度が高、中、または低
- issueDescription - 設定問題の詳細

- proposedFix - 問題を解決するための実用的なガイダンス

進行中の分析を停止する

進行中の分析をキャンセルする必要がある場合は、 botIdと を使用してStopBotAnalyzerリクエストを送信しますbotAnalyzerRequestId。

分析履歴の表示

ボットロケールの以前の分析のリストを取得するには、 ListBotAnalyzerHistory リクエストを送信します。botId および を指定localeIdして、ステータスとタイムスタンプを含むすべての過去の分析リクエストを表示します。

レコメンデーションを削除する

分析結果を削除するには、 botIdおよび を使用してDeleteBotAnalyzerRecommendationリクエストを送信しますbotAnalyzerRequestId。これにより、その分析に関連付けられたレコメンデーションが完全に削除されます。

> **Note**
>
> 推奨事項は 15 日後に自動的に削除されます。

関連トピック

- Amazon Lex インタラクションモデルを作成するためのベストプラクティス
- アシスト付き NLU を使用して Lex V2 のインテント分類とスロット解決を改善する
- アシスト付きスロット解決を使用した Amazon Lex V2 のスロット値の明確化

AMAZON.QnAIntent

> **Note**
>
> 生成 AI 機能を活用するには、最初に以下の前提条件を満たす必要があります。
>
> 1. Amazon Bedrock の使用料金については、「Amazon Bedrock の料金」を参照してください。

2. ボットロケールで生成 AI 機能を有効にします。これを行うには、「生成 AI を使用した Lex V2 ボットの作成とパフォーマンスの最適化」の手順に従います。

Amazon Bedrock FM を活用すると、顧客の質問にボット会話で回答できます。Amazon Lex V2 には、ボットに追加できる組み込み AMAZON.QnAIntent が用意されています。このインテントは、Amazon Bedrock の生成 AI 機能を活用して、顧客の質問を認識し、以下のナレッジストア (**Can you provide me details on the baggage limits for my international flight?** など) から回答を検索します。この機能により、Amazon Lex V2 のインテント内でタスク指向のダイアログを使用して、質問と回答を設定する必要が減ります。また、このインテントは、会話履歴に基づいてフォローアップの質問 (**What about domestic flight?** など) を認識し、質問に応じた回答を提供します。

「AMAZON.QnAIntent の権限」の手順に従って、IAM ロールが AMAZON.QnAIntent にアクセスするための適切な権限を持っていることを確認してください。

AMAZON.QnAIntent を利用するには、以下のナレッジストアのいずれかを設定しておく必要があります。

- Amazon OpenSearch Service データベース – 詳細については、「Amazon OpenSearch Service ドメインの作成と管理」を参照してください。
- Amazon Kendra インデックス – 詳細については、「インデックスの作成」を参照してください。
- Amazon Bedrock ナレッジベース — 詳細については、「ナレッジベースの構築」を参照してください。

AMAZON.QnAIntent は、次の 2 つの方法のいずれかでセットアップできます。

生成 AI 設定を使用してセットアップするには

1. Amazon Lex V2 コンソールの左側のナビゲーションペインで [ボット] を選択し、[ボット] セクションでインテントを追加するボットを選択します。
2. 左のナビゲーションペインで、インテントを追加する言語を選択します。
3. [生成 AI 設定] セクションで、[設定] を選択します。
4. [QnA 設定] セクションで、[QnA インテントを作成] を選択します。

組み込みインテントをボットに追加してセットアップするには

1. Amazon Lex V2 コンソールの左側のナビゲーションペインで [ボット] を選択し、[ボット] セクションでインテントを追加するボットを選択します。
2. 左側のナビゲーションペインで、インテントを追加する言語の下の [インテント] を選択します。
3. [インテントを追加] を選択し、ドロップダウンメニューから [組み込みインテントを使用] を選択します。
4. AMAZON.QnAIntent の設定の詳細については、「AMAZON.QnAIntent」を参照してください。

> **Note**
>
> AMAZON.QnAIntent が有効になるのは、発話がボット内に存在する他のインテントのいずれにも分類されない場合です。このインテントが有効になるのは、発話がボット内に存在する他のインテントのいずれにも分類されない場合です。スロット値を引き出す際に発話見逃しがあった場合、このインテントは有効にならないので注意してください。認識された場合、AMAZON.QnAIntent は、指定された Amazon Bedrock モデルを使用して設定済みのナレッジベースを検索し、顧客の質問に回答します。

トピック

- AMAZON.QnAIntent の権限

AMAZON.QnAIntent の権限

この機能に Amazon Lex V2 コンソールからアクセスするには、コンソールのロールに bedrock:ListFoundationModels 権限と bedrock:ListInferenceProfiles 権限があることを確認してください。

ボットに関連付けられた IAM ロールには、AMAZON.QnAIntent に必要な以下の権限を設定する必要があります。ボットロールには bedrock:InvokeModel を呼び出す権限が必要です。また、ボットの AMAZON.QnAIntent に指定するデータストアごとにステートメントをアタッチする必要があります (以下のポリシーの Permissions to access Amazon Kendra index、Permissions to access OpenSearch Service index、Permissions to access knowledge base in Amazon Bedrock ステートメントを参照してください)。この機能を Amazon Lex コンソールで有

効にすると、Amazon Lex が生成したサービスリンクロールをボットで使用している場合、ポリシーは自動的にボットロールに追加されます。

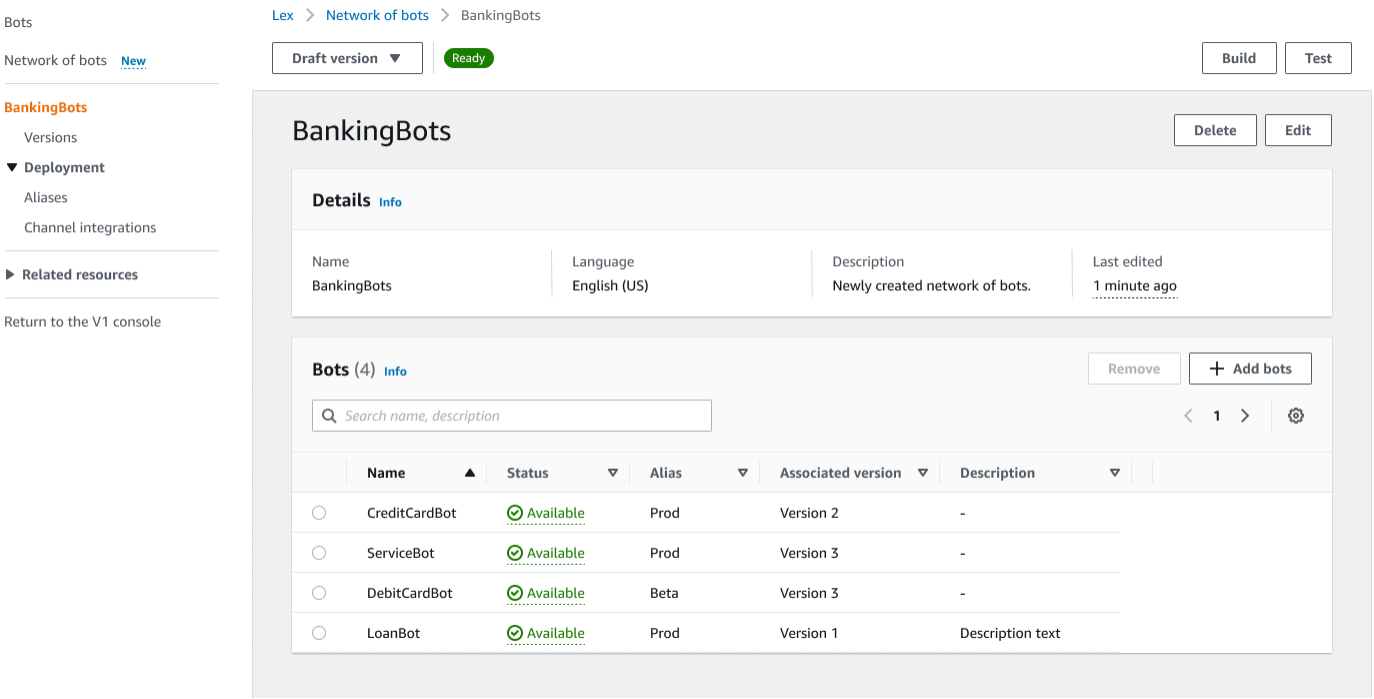
Lex V2 ボットのネットワークの作成

Network of Bots により、企業は複数のボットにわたって統一されたユーザーエクスペリエンスを提供できます。Network of Bots を使用すると、企業は複数のボットを単一のネットワークに追加して、柔軟で独立したボットのライフサイクル管理を実現できます。ネットワークは単一の統合インターフェースをエンドユーザーに公開し、ユーザー入力に基づいてリクエストを適切なボットにルーティングします。

改良されたボットが本番環境に導入されるにつれて、ボットを維持してネットワークに追加することで、チームが協力してさまざまなビジネスニーズを満たすボットのネットワークを構築できます。開発者は、複数のボットを単一のネットワークに統合することで、導入と改善を簡素化および迅速化できます。

ボットのネットワークは en-US 言語でのみ利用可能です。

> **Note**
>
> 現在、ボットのネットワークは 1 つのアカウントに制限されています。他のアカウントからボットを追加することはできません。



Lex V2 ボットのネットワークを作成する

AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/lexv2/home で Amazon Lex V2 コンソールを開きます。サイドメニューから [ボットのネットワーク] を選択します。ボットのネットワークを作成するには、少なくとも 1 つのボットを構築しておく必要があります。

ステップ 1: ボットのネットワーク設定を構成する

1. [詳細] セクションで、ネットワークの名前を入力し、必要に応じて説明を入力します。
2. IAM アクセス許可セクションで、Amazon CloudWatch などの他の AWS サービスにアクセスするためのアクセス許可を Amazon Lex V2 に付与する AWS Identity and Access Management (IAM) ロールを選択します。Amazon Lex V2 でロールを作成させるか、CloudWatch 権限を持つ既存のロールを選択できます。詳細については、「Amazon Lex V2 の Identity and Access Management」を参照してください。
3. Children's Online Privacy Protection Act (COPPA) セクションで、適切な応答を選択します。詳細については、「データプライバシー」を参照してください。
4. Idle session timeout セクションで、Amazon Lex V2 がユーザーとのセッションを開いたままにする期間を選択します。Amazon Lex V2 は、ボットが同じ変数で会話を再開できるように、セッション期間中はセッション変数を保持します。詳細については、「セッションタイムアウトの設定」を参照してください。
5. [言語設定の追加] セクションで、ボットがユーザーと対話するための音声を選択します。[音声サンプル]」にフレーズを入力し、[再生] を選択して音声を聞くことができます。
6. [詳細設定] セクションでは、ボットの識別に役立つタグを任意で追加できます。タグはアクセスの制御やリソースの監視に使用できます。詳細については、「リソースにタグを付ける」を参照してください。
7. [次へ] を選択し、ボットネットワークを作成してボットの追加に移ります。

ステップ 2: ボットを追加する

1. [ボット] セクションで、[+ ボットを追加] を選択します。
2. [ボットを追加] モーダルがポップアップ表示されます。[ボット] ドロップダウンメニューから追加するボットを選択し、[エイリアス]ドロップダウンメニューから使用したいボットのエイリアスを選択します。

エイリアスはドラフトバージョンではなく、ボットの番号付きバージョンを指している必要があります。ボットは 5 つまで追加できます。ボットは最大 25 の異なるネットワークに追加できます。

3. ネットワークにさらにボットを追加するには、[+ ボットの追加] を選択します。ボットを削除するには、削除するボットの横にある [削除] を選択します。ボットの追加が完了したら、[保存] を選択してモーダルを閉じます。
4. [保存] を選択してネットワークの作成を終了します。

Lex V2 ボットのネットワークを管理する

ボットのネットワークを作成すると、ネットワークを管理および構築できるページが表示されます。または、サイドメニューで [ボットのネットワーク] を選択し、管理するネットワークの名前を選択すると、このページにアクセスできます。

1. ネットワークの情報を編集するには、[詳細] セクションの上にある [編集] を選択します。ネットワークを削除するには、[詳細] セクションの上にある [削除] を選択します。
2. [ボット] セクションで、[+ ボットの追加] を選択してボットを追加できます。Amazon Lex V2 コンソールのサイドメニューにある [ボット] ページに移動すると、ボットを追加することもできます。追加するボットの横にあるラジオボタンを切り替え、[アクション] ドロップダウンメニューから [ボットのネットワークに追加] を選択します。

   表示されるモーダルの [ボットのネットワーク] ドロップダウンメニューから、ボットを追加するネットワークを選択します。[ボットエイリアス] ドロップダウンメニューから使用するボットのエイリアスを選択します。[追加] を選択して、選択したネットワークにボットを追加します。
3. ボットの横にあるラジオボタンを切り替えて [削除] を選択すると、ネットワークからボットを削除できます。
4. ネットワークの設定が完了したら、右上の [構築] を選択してネットワークを構築します。構築には数分かかります。構築が成功すると、ページ上部に完成を示す緑色のメッセージが表示されます。
5. ネットワークが構築されたら、右上の [テスト] を選択すると、右下隅にチャットウィンドウが表示されます。このチャットウィンドウを使ってネットワークのボットと会話したり、会話の流れや遷移が正しく設定されているかどうかを確認したりできます。

> **Note**
>
> ネットワーク内のボットを追加、削除、更新した場合は、ネットワークを再構築する必要があります。

Lex V2 ボットのネットワークのバージョン

ボットのネットワークにはさまざまなバージョンを作成できます。バージョンを管理するには、Amazon Lex V2 コンソールのサイドメニューからネットワークを選択し、[バージョン] を選択します。

1. [バージョンの作成] を選択して、ボットのネットワークの新しいバージョンを作成します。オプションとして説明を追加することができます。[作成] を選択して入力を作成します。
2. ボットのネットワークの特定のバージョンの横にあるラジオボタンを切り替えると、[エイリアスをバージョンに関連付ける] を選択して、エイリアスをそのバージョンに関連付けることができます。
3. ネットワークのバージョンを管理するには、[バージョン] セクションでバージョン名を選択します。次のページでは、バージョンの詳細を編集したり、バージョン内のボットとそれに関連するエイリアスを管理したりできます。

Lex V2 ボットのネットワークのエイリアス

エイリアスを使用してネットワークをデプロイできます。エイリアスを管理するには、Amazon Lex V2 コンソールのサイドメニューからネットワークを選択し、[エイリアス] を選択します。

1. [エイリアスを作成] を選択して新しいエイリアスを作成します。
2. エイリアスに名前を指定し、オプションで [エイリアスの詳細] セクションに [説明] を入力します。バージョンを選択してエイリアスを [バージョンと関連付ける] セクションに関連付け、[タグ] セクションにタグを追加できます。[作成] を選択して、エイリアスを作成します。
3. ネットワークのエイリアスを管理するには、[エイリアス] セクションでエイリアスの名前を選択します。次のページでは、エイリアスの詳細を編集したり、タグ、チャネル統合、リソースベースのポリシーを管理したりできます。ネットワークのバージョンとの関連付け履歴を表示することもできます。

Lex V2 ボットのネットワークのチャネル統合

ボットのネットワークをメッセージングプラットフォームと統合するには、Amazon Lex V2 コンソールのサイドメニューからボットのネットワークを選択します。次に、[チャネル統合] を選択します。

1. [チャネルを追加] を選択して、ネットワークを新しいチャネルと統合します。
2. [プラットフォーム] セクションの [プラットフォームを選択] で、ボットをデプロイするプラットフォームを選択します。IAM ロールが作成されます。[KMS キー] の下のドロップダウンメニューから情報を保護するためのキーを選択します。
3. [インテグレーション設定] チャネルで、[名前] と [説明] (オプション) を入力します。ドロップダウンメニューから [エイリアス] キーを選択します。
4. プラットフォームからアカウント SID と認証トークンを取得し、[アカウント SID] と [認証トークン] フィールドに入力します。詳細については、[ボットの統合] を参照してください。
5. [作成] を選択してチャンネル統合を完了します。

> **Note**
>
> ボットのネットワークは現在、Connect Customer の音声またはチャットでは使用できません。

本番環境用の Lex V2 ボットのデプロイ

ボットを作成してテストした時点で、デプロイして顧客とやり取りする準備が整います。このセクションでは、更新を行ったボットのバージョンを作成する方法を学びます。デプロイの準備が整ったら、エイリアスを使用してボットの異なるバージョンを示します。ボットをメッセージングプラットフォーム、モバイルアプリケーション、ウェブサイトと統合する方法について説明します。

トピック

- Lex V2 ボットを使用したバージョニングとエイリアス
- Java アプリケーションを使用して Amazon Lex V2 ボットと会話する
- グローバルレジリエンシーを使用した他のリージョンへのボットのデプロイ
- Amazon Lex V2 ボットのメッセージングプラットフォームとの統合
- Amazon Lex V2 ボットをコンタクトセンターと統合する

Lex V2 ボットを使用したバージョニングとエイリアス

Amazon Lex V2 は、クライアントアプリケーションが使用する実装を制御できるように、ボットのバージョニングおよびエイリアスとボットネットワークの作成をサポートしています。バージョンは、作業の番号付きスナップショットとして機能します。顧客に提供するボットのバージョンにエイリアスを指定できます。バージョンを作成する合間も、ユーザーエクスペリエンスに影響を与えずにボットの `Draft` バージョンを更新し続けることができます。

バージョン

Amazon Lex V2 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、ボットのバージョンの作成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、作業の段階別に作成して使用できます。

Lex V2 ボットのドラフトバージョン

Amazon Lex V2 ボットを作成する場合、バージョンは 1 つ (`Draft` バージョン) のみです。

`Draft` は、作業中のボットのコピーです。更新できるバージョンは `Draft` のみです。最初のバージョンを作成するまでは、`Draft` がボットの唯一のバージョンです。

Draft ボットのバージョンは TestBotAlias と関連づけられています。TestBotAlias 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 V2 では、TestBotAlias ボットのエイリアス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。

Lex V2 ボットのバージョンの作成

ボットの番号付きスナップショットを Amazon Lex V2 バージョンとして作成すると、そのバージョンの作成時点のボットを使用できます。作成したバージョンは、アプリケーションのドラフトバージョンで作業を続けても変更されずに残ります。

バージョンを作成するときに、バージョンに含める口ケールを選択できます。ボット内のすべての口ケールを選択する必要はありません。また、バージョンを作成するときに、以前のバージョンから口ケールを選択できます。例えば、ボットのバージョンが 3 つある場合は、Draft バージョンから 1 つの口ケールとバージョン 4 を作成するときのバージョン 2 の 1 つを選択できます。

Draft バージョンから口ケールを削除すると、それは番号付きバージョンからは削除されません。

ボットバージョンが 6 か月間使用されない場合、Amazon Lex V2 はそのバージョンを非アクティブとマークします。バージョンが非アクティブの場合、ボットでランタイムオペレーションを使用することはできません。ボットをアクティブにするには、バージョンに関連付けられているすべての言語を再構築します。

Amazon Lex V2 ボットの更新

Draft Amazon Lex V2 ボットのバージョンを更新できます。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 オペレーションを使用してリソースを更新した後で、いつでも作成できます。

Amazon Lex V2 ボットまたはバージョンの削除

Amazon Lex V2 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、ボットやバージョンを削除できます。

- DeleteBot
- DeleteBotVersion

Lex V2 ボットのエイリアス

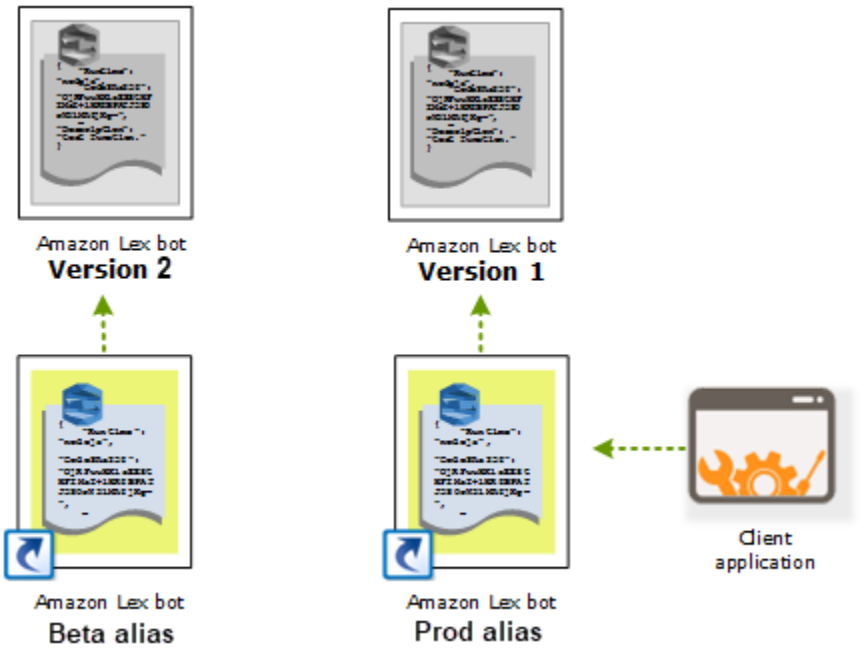
Amazon Lex V2 ボットでは、エイリアスがサポートされています。エイリアスはボットの特定バージョンを参照するポインタです。エイリアスを使用すると、クライアントアプリケーションが使用し

ているバージョンを簡単に更新できます。たとえば、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 を作成し、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。

エイリアスとは Amazon Lex V2 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。

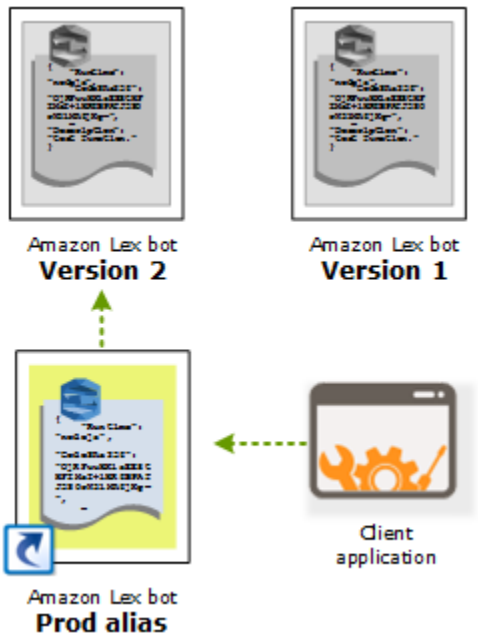
ボットを作成すると、Amazon Lex V2 は、ボットのテストに使える `TestBotAlias` というエイリアスを作成します。`TestBotAlias` エイリアスは常にボットの `Draft` バージョンと関連しています。使用するのは `TestBotAlias` テスト用のエイリアスで、Amazon Lex V2 ではエイリアスに対するランタイムリクエストの数が制限されています。

次の例では、Amazon Lex V2 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 UpdateBotAlias 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。



Amazon Lex Runtime V2 API を呼び出して顧客がボットとやり取りできるようにクライアントアプリケーションを設定する場合、顧客に使用してもらうバージョンを示すエイリアスを使用します。

> **Note**
>
> `Draft` 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作成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 V2 は進行中のすべてのセッションで現在のバージョンを使用します。新しいセッションでは、新しいバージョンが使用されます。

Java アプリケーションを使用して Amazon Lex V2 ボットと会話する

AWS SDK for Java 2.0 は、Java アプリケーションからボットと会話するために使用できるインターフェイスを提供します。SDK for Java を使用して、ユーザー用のクライアントアプリケーションを構築します。

次のアプリケーションは、演習 1: テンプレートからチャットボットを作成する で作成した OrderFlowers ボットと会話します。SDK for Java の `LexRuntimeV2Client` を使用して、RecognizeText オペレーションを呼び出してボットと会話を行います。

会話の出力は次のようになります。

```
User : I would like to order flowers
Bot  : What type of flowers would you like to order?
User : 1 dozen roses
Bot  : What day do you want the dozen roses to be picked up?
User : Next Monday
Bot  : At what time do you want the dozen roses to be picked up?
User : 5 in the evening
Bot  : Okay, your dozen roses will be ready for pickup by 17:00 on 2021-01-04.  Does
 this sound okay?
User : Yes
Bot  : Thanks.
```

クライアントアプリケーションと Amazon Lex V2 ボットの間で送信される JSON 構造については、「演習 2: 会話フローを確認する」を参照してください。

アプリケーションを実行するには、次の情報が必要です。

- botId - その作成時にボットに割り当てた識別子。ボットの Amazon Lex V2 コンソールでボット ID は、設定 ページを参照してください。
- botAliasId — ボットのエイリアスの作成時に割り当てた識別子。ボットのエイリアス ID は、Amazon Lex V2 コンソールの エイリアス ページで参照してください。リストにエイリアス ID が表示されない場合は、エイリアス ID で右上の歯車アイコンを選択してオンにします。
- localeId — ボットに使用したロケールの識別子。ロケールのリストについては、「Amazon Lex V2 でサポートされている言語とロケール」を参照してください。
- accessKey と secretKey — アカウントの認証キー。キーのセットがない場合は、AWS Identity and Access Management コンソールを使用して作成します。
- sessionId — Amazon Lex V2 ボットとのセッションの識別子。この場合、コードはランダムな UUID を使用します。
- リージョン — ボットが米国東部 (バージニア北部) リージョンにない場合は、リージョンを変更してください。

アプリケーションは、getRecognizeTextRequest という関数を使用してボットへの個別のリクエストを作成します。この関数は、Amazon Lex V2 に送信するために必要なパラメータを使用してリクエストを作成します。

```
package com.lex.recognizetext.sample;

import software.amazon.awssdk.auth.credentials.AwsBasicCredentials;
import software.amazon.awssdk.auth.credentials.AwsCredentialsProvider;
import software.amazon.awssdk.auth.credentials.StaticCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.lexruntimev2.LexRuntimeV2Client;
import software.amazon.awssdk.services.lexruntimev2.model.RecognizeTextRequest;
import software.amazon.awssdk.services.lexruntimev2.model.RecognizeTextResponse;

import java.net.URISyntaxException;
import java.util.UUID;


/**
 * This is a sample application to interact with a bot using RecognizeText API.
 */
public class OrderFlowersSampleApplication {

    public static void main(String[] args) throws URISyntaxException,
 InterruptedException {
        String botId = "";
        String botAliasId = "";
        String localeId = "en_US";
        String accessKey = "";
        String secretKey = "";
        String sessionId = UUID.randomUUID().toString();
        Region region = Region.US_EAST_1; // pick an appropriate region

        AwsBasicCredentials awsCreds = AwsBasicCredentials.create(accessKey,
 secretKey);
        AwsCredentialsProvider awsCredentialsProvider =
 StaticCredentialsProvider.create(awsCreds);

        LexRuntimeV2Client lexV2Client = LexRuntimeV2Client
                .builder()
                .credentialsProvider(awsCredentialsProvider)
                .region(region)
```

```
                .build();

        // utterance 1
        String userInput = "I would like to order flowers";
        RecognizeTextRequest recognizeTextRequest = getRecognizeTextRequest(botId,
botAliasId, localeId, sessionId, userInput);
        RecognizeTextResponse recognizeTextResponse =
lexV2Client.recognizeText(recognizeTextRequest);

        System.out.println("User : " + userInput);
        recognizeTextResponse.messages().forEach(message -> {
            System.out.println("Bot : " + message.content());
        });

        // utterance 2
        userInput = "1 dozen roses";
        recognizeTextRequest = getRecognizeTextRequest(botId, botAliasId, localeId,
sessionId, userInput);
        recognizeTextResponse = lexV2Client.recognizeText(recognizeTextRequest);

        System.out.println("User : " + userInput);
        recognizeTextResponse.messages().forEach(message -> {
            System.out.println("Bot : " + message.content());
        });

        // utterance 3
        userInput = "next monday";
        recognizeTextRequest = getRecognizeTextRequest(botId, botAliasId, localeId,
sessionId, userInput);
        recognizeTextResponse = lexV2Client.recognizeText(recognizeTextRequest);

        System.out.println("User : " + userInput);
        recognizeTextResponse.messages().forEach(message -> {
            System.out.println("Bot : " + message.content());
        });

        // utterance 4
        userInput = "5 in evening";
        recognizeTextRequest = getRecognizeTextRequest(botId, botAliasId, localeId,
sessionId, userInput);
        recognizeTextResponse = lexV2Client.recognizeText(recognizeTextRequest);

        System.out.println("User : " + userInput);
        recognizeTextResponse.messages().forEach(message -> {
```

```
            System.out.println("Bot : " + message.content());
        });

        // utterance 5
        userInput = "Yes";
        recognizeTextRequest = getRecognizeTextRequest(botId, botAliasId, localeId,
 sessionId, userInput);
        recognizeTextResponse = lexV2Client.recognizeText(recognizeTextRequest);

        System.out.println("User : " + userInput);
        recognizeTextResponse.messages().forEach(message -> {
            System.out.println("Bot : " + message.content());
        });
    }

    private static RecognizeTextRequest getRecognizeTextRequest(String botId, String
 botAliasId, String localeId, String sessionId, String userInput) {
        RecognizeTextRequest recognizeTextRequest = RecognizeTextRequest.builder()
                .botAliasId(botAliasId)
                .botId(botId)
                .localeId(localeId)
                .sessionId(sessionId)
                .text(userInput)
                .build();
        return recognizeTextRequest;
    }
}
```

グローバルレジリエンシーを使用した他のリージョンへのボットのデプロイ

グローバルレジリエンシーを使用すると、セカンダリリージョンにボットをレプリケートできます。セカンダリリージョンは、両リージョンでユーザーのボットを自動的にレプリケートすることでアクティブにすることができます。リージョンの停止が発生した場合に備え、バックアップリージョンを持つことができます。グローバルレジリエンシーがアクティブになると、作成された新しいボットは 2 番目の AWS リージョンにレプリケートされます。

> **Note**
>
> この機能は、米国東部 (バージニア北部)、米国西部 (オレゴン)、欧州西部 (ロンドン)、欧州中部 (フランクフルト) のリージョンで作成された Amazon Connect および Amazon Lex V2 インスタンスでのみ利用可能です。
> この機能を利用するには、Amazon Connect ソリューションアーキテクトまたはテクニカルアカウントマネージャーにお問い合わせください。

この機能を有効にすると、ペアになった AWS リージョンでの Amazon Lex V2 ボットとそのリソース、バージョン、エイリアスのレプリケーションをほぼリアルタイムで自動化できます。この機能を使用すると、オリジナルのボットとレプリカボットのバージョン番号をモニタリングして、ボットレプリカがオリジナルのボットと同期していることを確認できます。レプリケーションを有効にすると、ボットをレプリケートする事前定義された AWS リージョンをアクティブ化できます (リージョンは事前定義されたペアに基づいています)。ソースリージョンのソースボットの更新は、2 番目のリージョンのレプリケートされたボットに自動的に反映されます。

> **Note**
>
> ボットに対してグローバルレジリエンシーを有効にすると、既存のエイリアスとその関連バージョンがすべてレプリカリージョンにレプリケートされます。レプリケーションを有効にする前にエイリアスに関連付けられていないバージョンは、エイリアスに関連付けられるとレプリケートされます。レプリケーションを有効にした後に作成されたすべてのバージョンとエイリアスは、自動的にレプリケートされます。ユーザーは `ListBotVersionReplicas`と `ListBotAliasReplicas` を使用して、個々のバージョンとエイリアスのレプリケーションのステータスを確認できます。ボットミューテーションは、ボットからそのレプリカへの一方向です。レプリカボットは常にボットと同期されるため、ユーザーはレプリカボットを変更できません。

グローバルレジリエンシーの使用に関する追加情報:

- グローバルレジリエンシーは現在、事前に定義されたリージョンのペアでのみ機能します。

us–east–1	us-west-2
eu-west-2	eu-central-1

- ボットに対してグローバルレジリエンシーを有効にすると、既存のエイリアスとその関連バージョンがすべてレプリカリージョンにレプリケートされます。レプリケーションを有効にする前にエイリアスに関連付けられていないバージョンは、エイリアスに関連付けられるとレプリケートされます。レプリケーションを有効にした後に作成されたすべてのバージョンとエイリアスは、自動的にレプリケートされます。ユーザーは `ListBotVersionReplicas` と `ListBotAliasReplicas` を使用して、個々のバージョンとエイリアスのレプリケーションのステータスを確認できます。ボットミューテーションは、ボットからそのレプリカへの一方向です。レプリカボットは常にボットと同期されるため、ユーザーはレプリカボットを変更できません。
- 任意のエイリアスを任意のバージョンに関連付けることができます。バージョンがまだレプリケートされていない場合、エイリアスとの関連付け中にレプリケートされます。

機能制限:

- グローバルレジリエンシーは、CFAQ や発話生成などの LLM を使用するスロットで作成されたボットをレプリケートしません。
- グローバルレジリエンシーはボットのネットワークをレプリケートしませんが、ボットのネットワークの一部であるボットは引き続き個別にレプリケートできます。

トピック

- Lex V2 でボットをレプリケートし、ボットレプリカを管理する権限
- Lex V2 ボットを使用したグローバルレジリエンシーのデプロイ

Lex V2 でボットをレプリケートし、ボットレプリカを管理する権限

IAM ロールに AmazonLexFullAccess ポリシーがアタッチされている場合、ボットレプリカを作成および管理できます。

グローバルレジリエンシーに対して最小限の権限を持つロールを作成する場合は、次のステートメントを含む以下のポリシーを使用します。

- Amazon Lex V2 のサービスにリンクされたボットレプリケーションのロールにアクセスする権限。
- Amazon Lex V2 がユーザーに代わってボットレプリケーションのサービスリンクロールを作成できるようにする権限。
- ボットレプリケーション API を呼び出す権限。

以下のように変更することで、権限をさらに制限できます。

- * を特定のボットまたはボットエイリアス ID に置き換えて、権限を特定のボットまたはボットエイリアスに制限します。
- `lex BotReplica` アクションのサブセットを使用して、ロールを特定のアクションに制限します。

例については、ユーザーにボットレプリカの作成と表示を許可しますが、削除は許可しないを参照してください。

Lex V2 ボットを使用したグローバルレジリエンシーのデプロイ

グローバルレジリエンシーを使用すると、セカンダリリージョンにボットをレプリケートできます。セカンダリリージョンは、両リージョンでユーザーのボットを自動的にレプリケートすることでアクティブにすることができます。リージョンの停止が発生した場合に備え、バックアップリージョンを持つことができます。グローバルレジリエンシーがアクティブになると、作成された新しいボットは 2 番目の AWS リージョンにレプリケートされます。

グローバルレジリエンシー情報パネルにはデプロイの詳細が表示される

グローバルレジリエンシーパネルでは、次の情報にアクセスできます。

- ソースの詳細 – ボットのソースリージョン、レプリカタイプ、レプリケーションが有効になった日付、最後に作成されたバージョンに関する情報。この情報を使用して、ボットのイテレーションを追跡します。
- レプリケーションの詳細 – ボットレプリカを作成したら、レプリケートされたリージョン、レプリカタイプ、最後のバージョン同期日、および最後にレプリケートされたバージョンを追跡できます。この情報を使用して、ボットレプリカの同期を追跡します。
- ソースリージョン – グローバルレジリエンシーが有効にされたリージョン。ソースリージョンを変更して、両方のリージョンでボットをレプリケートできます。
- レプリカタイプ – リージョンに応じて、ボットが読み取り専用であるか、読み書きできるかどうかを示します。
- レプリカリージョン – グローバルレジリエンシーでソースボットをレプリケートするために使用されるセカンダリリージョン。グローバルレジリエンシーは現在、IAD/PDX と LDN/FRA のリージョンペアでのみ機能します。
- レプリケーションが有効になった日付 – ボットレプリカが有効になった日時。

- 最後に作成されたバージョン – ソースリージョンのレプリカに関連付けられた最後のボットバージョン。

Lex V2 ボットのグローバルレジリエンシーの有効化

Amazon Lex V2 コンソールでグローバルレジリエンシーを有効化する前に、ボットレプリケーションを有効にするユーザーにサービスリンクロール (SLR) を作成する権限があることを確認する必要があります。グローバルレジリエンシーは、CreateReplica が呼び出されたときに、これらの FAS 認証情報を使用して有効なアカウントに SLR を作成します。Amazon Lex V2 でグローバルレジリエンシーの SLR を設定する方法の詳細については、「AWS マネージドポリシー: AmazonLexFullAccess」を参照してください。

> **Note**
>
> この機能は、米国東部 (バージニア北部)、米国西部 (オレゴン)、欧州西部 (ロンドン)、欧州中部 (フランクフルト) のリージョンで作成された Amazon Connect および Amazon Lex V2 インスタンスでのみ利用可能です。
> この機能を利用するには、Amazon Connect ソリューションアーキテクトまたはテクニカルアカウントマネージャーにお問い合わせください。

グローバルレジリエンシーをアクティブ化し、2 番目のリージョンのボットレプリケーションを設定します。

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/lex/ で Amazon Lex コンソールを開きます。
2. 左側のナビゲーションパネルの [ボット] ナビゲーションからレプリケートするボットを選択します。
3. [デプロイ] > [Global Resiliency] を選択します。
4. ウィンドウの右上隅にある [レプリカを作成] ボタンを選択して、ボットのドラフトバージョンを作成します。

> **Note**
>
> レプリケートするボットと同じ名前のボットがセカンダリリージョンにないことを確認します (ボットには一意の名前を付ける必要があります)。

5. [Global Resiliency] に移動し、[レプリカを作成] をクリックします。このアクションにより、ボットのドラフトバージョンが作成されます (ステータスを確認したり、将来のビルドの詳細を確認したりする場合を除いて、グローバルレジリエンシータブに戻る必要はありません)。

   > **Note**
   >
   > グローバルレジリエンシーでレプリケーション用のエイリアスボットを作成するには、エイリアスに移動し、グローバルレジリエンシー対応ボットの [新しいエイリアスの作成] を選択します。すべてのエイリアスは、レプリケーションが有効になる前に作成された場合でもレプリケートされます。

6. グローバルレジリエンシーが有効になっているボットの [エイリアス] > [新しいエイリアスの作成] に移動します。すべてのエイリアスは、レプリケーションが有効になる前に作成された場合でもレプリケートされます。
7. グローバルレジリエンシーが有効になっているボットの [バージョン] > [新しいバージョンの作成] に移動します。エイリアスに関連付けられたバージョンは、レプリケーションが有効になる前に作成された場合でもレプリケートされます。

> **Note**
>
> お客様は、引き続きレプリケートされたボットのリソースベースのポリシーとタグを完全に管理できます。Lambda 関数と CloudWatch ロググループは、同じ識別子を使用して両方のリージョンにデプロイする必要があります。レプリカリージョンで Lambda 関数を再度関連付ける必要はありません。

グローバルレジリエンシーの無効化

グローバルレジリエンシーは、[Global Resiliency を無効化] ボタンを選択することで、いつでも無効化できます。このアクションにより、ソースボットと、それに関連付けられたエイリアスとバージョンが他のリージョンにレプリケートされることが停止されます。

Lex V2 ボットのグローバルレジリエンシーでの API の使用

以下の API を使用して、グローバルレジリエンシーで API コールを実行できます。グローバルレジリエンシー APIと Amazon Lex V2 の詳細については、「Amazon Lex V2 API ガイド」を参照してください。

- CreateBotReplica

  グローバルレジリエンシーを有効にし、レプリケートボットを作成します。replicaRegion が必須です。

  詳細については、Amazon Lex V2 API ガイド」の「CreateBotReplica」を参照してください。

- DeleteBotReplica

  グローバルレジリエンシーを無効にし、レプリケートボットを削除します。replicaRegion および botId が必須です。

  詳細については、Amazon Lex V2 API ガイド」のDeleteBotReplica」を参照してください。

- ListBotReplicas

  セカンダリゾーン内のレプリケートボットを一覧表示します。botId が必須です。

  詳細については、Amazon Lex V2 API ガイド」のListBotReplicas」を参照してください。

- DescribeBotReplica

  レプリケートボットの概要情報。replicaRegion および botId が必須です。

  詳細については、Amazon Lex V2 API ガイド」の「DescribeBotReplica」を参照してください。

Amazon Lex V2 ボットのメッセージングプラットフォームとの統合

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と Amazon Lex V2 ボットを連携させる方法について説明します。Amazon Lex V2 ボットがまだお持ちでない場合は、作成してください。このトピックでは、演習 1: テンプレートからチャットボットを作成する 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、どのようなボットでも使用可能です。

> **Note**
>
> Facebook、Slack、または Twilio の設定を保存する場合、Amazon Lex V2 は AWS KMS key 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へのチャネルを初めて作成すると、Amazon Lex V2 は AWS アカウントにデフォルトのカスタマーマネージドキー (aws/lex) を作成するか、独自のカスタマーマネージドキーを選択で

きます。Amazon Lex V2 は、対称キーのみをサポートします。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 V2 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。この属性を使用してボットの動作方法をカスタマイズします。詳細については、「Lex V2 ボットのリクエスト属性の設定」を参照してください。

メッセージングプラットフォームの一般的なリクエスト属性

属性	説明
x-amz-lex: チャネル: プラットフォーム	次のいずれかの値になります。 • `Facebook` • `Slack` • `Twilio`

Amazon Lex V2 ボットと Facebook Messenger の統合

Amazon Lex V2 ボットを Facebook Messenger でホストすることができます。そうすると、Facebook ユーザーはボットとインタラクションして、インテントを満たすことができます。

開始する前に、https://developers.facebook.com で Facebook デベロッパーアカウントにサインアップする必要があります。

以下のステップを実行する必要があります。

トピック

- ステップ 1: Facebook アプリケーションを作成する
- ステップ 2: Facebook Messenger と Amazon Lex V2 ボットを統合する
- ステップ 3: Lex V2 ボットと Facebook の統合を完了する
- ステップ 4: Facebook Messenger とボットの統合をテストする

ステップ 1: 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。

Facebook アプリケーションを作成するには

1. https://developers.facebook.com/apps を開く
2. [Create App (アプリの作成)] を選択します。
3. [アプリケーションの作成] ページで [ビジネス] を選択し、[次へ] を選択します。
4. [アドオンアプリ名]、[アプリ連絡先メールアドレス]、[ビジネスアカウント] の各項目で、アプリに適した選択を行います。アプリケーションの作成 を選択して続行します。
5. アプリケーションに商品を追加する から、メッセンジャー タイルからの セットアップ を選択します。
6. アクセストークン セクションで、ページの追加または削除 を選択します。
7. アプリで使用するページを選択し、次へ を選択します。
8. アプリケーションに許可されていることとは はデフォルトのままにして、完了 を選択します。
9. 確認ページで、[OK] を選択します。
10. アクセストークン セクションで、トークンを生成する を選択し、トークンをコピーします。このトークンを Amazon Lex V2 コンソールに入力します。
11. 左側のメニューから [設定] を選択し、[ベーシック] を選択します。
12. アプリケーションシークレットは、表示 を選んでから、シークレットをコピーします。このトークンを Amazon Lex V2 コンソールに入力します。

次のステップ

ステップ 2: Facebook Messenger と Amazon Lex V2 ボットを統合する

ステップ 2: Facebook Messenger と Amazon Lex V2 ボットを統合する

このステップでは、Amazon Lex V2 ボットを Facebook と連携させます。

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/lex/ で Amazon Lex コンソールを開きます。
2. ボットの一覧から、作成した Amazon Lex V2 ボットを選択します。
3. 左側のメニューで、チャネル統合 を選択し、チャンネルを追加 を選択します。
4. [チャネルの作成] で、以下の操作を行います。

a. プラットフォーム で、Facebook を選択します。

b. ID ポリシーで、チャネル情報を保護する AWS KMS キーを選択します。デフォルトのキーは Amazon Lex V2 が提供するものです。

c. 統合の設定 には、チャネル名とオプションの説明を指定します。使用するボットのバージョンを指すエイリアスを選択し、チャネルがサポートする言語を選択します。

d. [追加設定] では、次のように入力します。

- エイリアス - Amazon Lex V2 を呼び出しているアプリを識別する文字列です。任意の文字列を使用できます。この文字列を記録して、Facebook デベロッパーコンソールに入力します。
- ページアクセストークン - Facebook デベロッパーコンソールからコピーしたページアクセストークン。
- アプリケーションの秘密鍵 - Facebook デベロッパーコンソールからコピーした秘密鍵。

e. [作成] を選択します

f. Amazon Lex V2 には、ボットのチャンネルリストが表示されます。リストから、先ほど作成したチャネルを選択します。

g. コールバック URL から、コールバック URL を記録します。この URL を、Facebook デベロッパーコンソールに入力します。

次のステップ

ステップ 3: Lex V2 ボットと Facebook の統合を完了する

ステップ 3: Lex V2 ボットと Facebook の統合を完了する

このステップでは、Facebook デベロッパーコンソールを使用して、Amazon Lex V2 ボットとの統合を完了します。

Facebook Messenger との連携を完了するには

1. https://developers.facebook.com/apps を開く
2. アプリケーションの一覧から、Facebook Messenger と連携させるアプリケーションを選択します。
3. 左側のメニューで、メッセンジャー を選択し、設定 を選択します。
4. [Webhook] セクションで:

a. [コールバック URL を追加] を選択します。
b. コールバック URL の編集 に、以下を入力します。
   - コールバック URL - Amazon Lex V2 コンソールから記録したコールバック URL を入力します。
   - トークンの確認— Amazon Lex V2 コンソールで入力したエイリアスを入力します。
c. [Verify and Save] を選択します。
d. ページの横にある [Webhook] で [サブスクリプションを追加] を選択します。
e. 表示されるウィンドウで `messages` を選択し、[保存] をクリックします。

次のステップ

ステップ 4: Facebook Messenger とボットの統合をテストする

ステップ 4: Facebook Messenger とボットの統合をテストする

Facebook Messenger から Amazon Lex V2 ボットと会話を開始できるようになりました。

Facebook Messenger と Amazon Lex V2 ボットの連携をテストするために

1. ステップ 1 でボットに関連付けた Facebook ページを開きます。
2. [Messenger] ウィンドウで、「演習 1: テンプレートからチャットボットを作成する」に記載されたテスト発話を使用します。

Amazon Lex V2 ボットと Slack の統合

このトピックでは、Amazon Lex V2 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説明します。以下のステップを実行します。

トピック

- ステップ 1: 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 アプリケーションを作成する
- ステップ 3: Slack アプリケーションと Amazon Lex V2 ボットを統合する
- ステップ 4: Lex V2 ボットと Slack の統合を完了する
- ステップ 5: Lex V2 ボットと Slack の統合をテストする

ステップ 1: 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では、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 2: Slack アプリケーションを作成する

ステップ 2: 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な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。次のステップでは、この情報を使用して Amazon Lex V2 コンソールでボットを統合します。

Slack アプリケーションを作成するには

1. https://api.slack.com で Slack API コンソール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 3: Slack アプリケーションと Amazon Lex V2 ボットを統合する

ステップ 3: Slack アプリケーションと Amazon Lex V2 ボットを統合する

このセクションでは、作成した Slack アプリケーションを、チャネル統合を使用して作成した Amazon Lex V2 ボットと統合します。

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/lex/ で Amazon Lex コンソールを開きます。
2. ボットの一覧から、作成した Amazon Lex V2 ボットを選択します。
3. 左側のメニューで、チャネル統合 を選択し、チャンネルを追加 を選択します。
4. [チャネルの作成] で、以下の操作を行います。

   a. プラットフォーム は、Slack を選択します。

   b. ID ポリシーで、チャネル情報を保護する AWS KMS キーを選択します。デフォルトのキーは Amazon Lex V2 が提供するものです。

   c. 統合の設定 には、チャネル名とオプションの説明を指定します。使用するボットのバージョンを指すエイリアスを選択し、チャネルがサポートする言語を選択します。

> **Note**
>
> ボットが複数の言語で利用できる場合は、言語ごとに異なるチャネルと異なるアプリケーションを作成する必要があります。

d. [追加設定] では、次のように入力します。

- クライアント ID - Slack からクライアント ID を入力します。
- クライアントシークレット - Slack からクライアントシークレットを入力します。
- 検証トークン - Slack から検証トークンを入力します。
- 成功ページ URL - ユーザーが認証されたときに Slack が開くべきページの URL。通常、空欄にします。

5. 作成 を選択し、チャネルを作成します。
6. Amazon Lex V2 には、ボットのチャンネルリストが表示されます。リストから、先ほど作成したチャネルを選択します。
7. コールバック URL から、エンドポイントと OAuth エンドポイントを記録します。

次のステップ

ステップ 4: Lex V2 ボットと Slack の統合を完了する

ステップ 4: Lex V2 ボットと Slack の統合を完了する

このセクションでは、Slack API コンソールを使用して Slack アプリケーションとの統合を完了します。

1. https://api.slack.com で Slack API コンソールにサインインします。「ステップ 2: 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. [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 エンドポイントを追加します。[Add]、[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. [リクエスト URL] の値を、前のステップで Amazon Lex から返されたエンドポイントに更新することで、[インタラクティブ&ショートカット] 機能を更新します。ステップ 3 で保存したエンドポイントを入力し、[変更を保存] 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] の値を、前のステップで Amazon Lex から返された エンドポイントに設定します。
   - [ボットイベントをサブスクライブ] セクションで、[ボットユーザーイベントを追加] を選択し、**message.im** ボットイベントを追加して、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。
6. [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。複数のワークスペースで認証されている場合は、まずドロップダウンリストから右上隅にある正しいワークスペースを選択します。次に、[許可] を選択してボットがメッセージに応答すること許可します。

> **Note**
>
> 後で Slack アプリケーション設定を変更した場合は、このサブステップをやり直す必要があります。

次のステップ

[ステップ 5: Lex V2 ボットと Slack の統合をテストする]

ステップ 5: Lex V2 ボットと Slack の統合をテストする

ここでは、ブラウザウィンドウを使用して、Slack と Amazon Lex V2 ボットとの統合をテストします。

Slack アプリケーションをテストするには

1. Slack を起動します。左のメニューの [Direct Messages] のセクションで、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。
2. Slack アプリケーションでチャットに参加する。ボットはメッセージに応答します。

   演習 1: テンプレートからチャットボットを作成する を使用してボットを作成した場合、その演習で使用した会話例が使用できます。

Amazon Lex V2 ボットと Twilio SMS の統合

このトピックでは、Amazon Lex V2 ボットと Twilio シンプルメッセージングサービス (SMS) を統合する手順を説明します。以下のステップを実行します。

トピック

- ステップ 1: Twilio SMS アカウントを作成する
- ステップ 2: Twilio メッセージングサービスエンドポイントと Amazon Lex V2 ボットを統合する
- ステップ 3: Lex V2 ボットと Twilio の統合を完了する
- ステップ 4: Lex V2 ボットと Twilio の統合をテストする

ステップ 1: 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。

次のステップ

ステップ 2: Twilio メッセージングサービスエンドポイントと Amazon Lex V2 ボットを統合する

ステップ 2: Twilio メッセージングサービスエンドポイントと Amazon Lex V2 ボットを統合する

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/lex/ で Amazon Lex コンソールを開きます。
2. ボットの一覧から、作成した Amazon Lex V2 ボットを選択します。
3. 左側のメニューで、チャネル統合 を選択し、チャンネルを追加 を選択します。
4. [チャネルの作成] で、以下の操作を行います。
   a. [プラットフォーム] は、[Twilio] を選択します。
   b. ID ポリシーで、チャネル情報を保護する AWS KMS キーを選択します。デフォルトのキーは Amazon Lex V2 が提供するものです。
   c. 統合の設定 には、チャネル名とオプションの説明を指定します。使用するボットのバージョンを指すエイリアスを選択し、チャネルがサポートする言語を選択します。
   d. 追加設定 には、Twilio ダッシュボードからアカウント SID と認証トークンを入力します。
5. [作成] を選択します。
6. チャンネルの一覧から、先ほど作成したチャネルを選択します。
7. コールバック URL をコピーします。

次のステップ

ステップ 3: Lex V2 ボットと Twilio の統合を完了する

ステップ 3: Lex V2 ボットと Twilio の統合を完了する

Twilio コンソールを使用して、Amazon Lex V2 ボットと Twilio SMS の統合を完了します。

1. Twilio コンソール (https://www.twilio.com/console) を開きます。
2. 左側のメニューから、すべての製品 & サービス を選んでから、電話番号 を選びます。
3. 電話番号がある場合は、それを選択します。電話番号がない場合は、番号を購入する を選択して電話番号を取得してください。
4. メッセージング セクションの A MESSAGE COMES IN に、Amazon Lex V2 コンソールからコールバック URL を入力します。
5. [保存] を選択します。

次のステップ

ステップ 4: Lex V2 ボットと Twilio の統合をテストする

ステップ 4: Lex V2 ボットと Twilio の統合をテストする

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。携帯電話を使用して、Twilio 番号にメッセージを送信します。

演習 1: テンプレートからチャットボットを作成する を使用してボットを作成した場合、その演習で使用した会話例が使用できます。

Amazon Lex V2 ボットをコンタクトセンターと統合する

Amazon Lex V2 ボットをコンタクトセンターと統合して、Amazon Lex V2 ストリーミング API を使用してセルフサービスのユースケースを有効にすることができます。これらのボットは、テレフォニー上の対話型音声応答 (IVR) エージェントとして、またはコンタクトセンターに統合されたテキストベースの chatbot として使用します。ストリーミング API の詳細については、[Amazon Lex V2 ボットへの会話のストリーミング] を参照してください。

ストリーミング API を使用すると、以下の機能を有効にできます。

- 中断(「割り込み」) - プロンプトが完了する前に、発信者がボットに割り込み、質問に答えることができます。詳細については、「ユーザーによる Amazon Lex V2 ボットの中断を可能にする」を参照してください。
- 待機と続行 - 発信者は、通話中にクレジットカード番号や予約IDなどの追加情報を取得する時間が必要な場合、ボットに待機するように指示できます。詳細については、「一時停止中に、Amazon Lex V2 ボットによるユーザーの追加の情報提供の待機を可能にする」を参照してください。
- DTMF のサポート - 発信者は、音声または DTMF を介して情報を交換可能な方法で提供できます。
- SSML のサポート - Amazon Lex V2 ボットプロンプトを SSML タグで設定して、テキストからの音声生成をより詳細に制御できます。詳細については、Amazon Polly 開発者ガイド の「SSML ドキュメントからの音声を生成する」を参照してください。
- 設定可能なタイムアウト - Amazon Lex V2 が、「はい」や「いいえ」の質問への回答、日付やクレジットカード番号の入力など、お客様の発話入力を収集するまでの待ち時間を設定することができます。詳細については、「Lex V2 ボットでのユーザー入力キャプチャのタイムアウトの設定」を参照してください。

- フルフィルメントの進捗状況の更新 — インテントフルフィルメントのビジネスロジックの実行中に、フルフィルメントステータスに基づいて複数のメッセージで応答するようにボットを設定できます。フルフィルメントの開始時と完了時にボットがメッセージを返すように設定し、長時間実行されている Lambda 関数を定期的に更新するように設定できます。詳細については、「Lex V2 ボットのフルフィルメント進行状況の更新の設定」を参照してください。

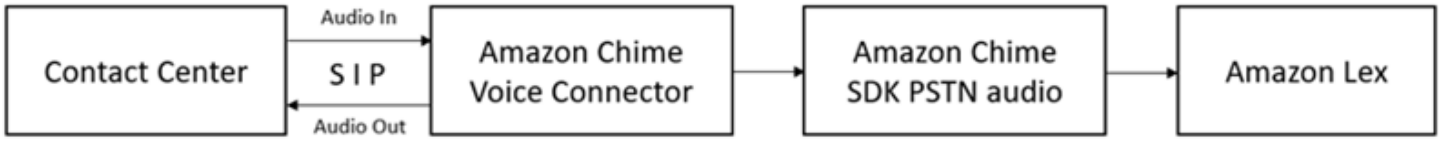
Amazon Chime SDK

Amazon Chime SDK を使用して、リアルタイム音声、動画、画面共有、およびメッセージング機能をウェブまたはモバイルアプリケーションに追加します。Amazon Chime SDK は公衆交換電話網 (PSTN) オーディオサービスを提供するため、 AWS Lambda 関数でカスタムテレフォニーアプリケーションを構築できます。

Amazon Chime PSTN オーディオは Amazon Lex V2 に統合されます。この統合を利用すると、コンタクトセンターの音声対話用の自動音声応答 (IVR) システムとして Amazon Lex V2 ボットにアクセスできます。これを使用して、以下のシナリオで PSTN オーディオサービスを使用して Amazon Lex V2 を統合します。

コンタクトセンターの統合 — Amazon Chime Voice Connector と Amazon Chime SDK PSTN Audio サービスを使用して Amazon Lex V2 ボットにアクセスできます。音声通信にセッション開始プロトコル (SIP) を使用するあらゆるコンタクトセンターアプリケーションで使用できます。この統合により、SIP サポートを備えた既存のオンプレミスまたはクラウドベースのコンタクトセンターに、自然言語の音声会話体験が追加されます。サポートされているコンタクトセンタープラットフォームのリストについては、「Amazon Chime Voice Connector リソース」を参照してください。

次の図は、SIP と Amazon Lex V2 を使用するコンタクトセンター間の統合を示しています。



ダイレクトテレフォニーサポート — Amazon Chime SDK にプロビジョニングされた電話番号を使用して Amazon Lex V2 ボットに直接アクセスするカスタマイズされた IVR ソリューションを構築できます。

詳細については、「Amazon Chime SDK ユーザーガイド」の次のトピックを参照してください。

- Amazon Chime Voice Connector を使用した SIP 統合
- Amazon Chime SDK PSTN オーディオサービスの使用

- Amazon Chime PSTN オーディオを Amazon Lex V2 に統合する

Amazon Chime SDK が Amazon Lex V2 にリクエストを送信する際、Lambda 関数と会話ログへのリクエスト属性に対してプラットフォーム固有の情報が含まれています。この情報をもとに、ボットにトラフィックを送っているコンタクトセンターアプリケーションを特定します。

一般的なリクエスト属性	値
x-amz-lex: チャネル: プラットフォーム	`Amazon Chime SDK PSTN Audio`

Connect のお客様

Connect Customer はオムニチャネルクラウドコンタクトセンターです。2～3 ステップで問い合わせセンターを設定し、どこからでもエージェントを追加して、顧客対応を開始することができます。詳細については、「 Connect Customer administrator guide」の「Get started with Connect Customer」を参照してください。

オムニチャネル通信を使用して、顧客向けにパーソナライズされたエクスペリエンスを作成できます。たとえば、顧客の好みや推定待機時間に基づいて、チャットや音声による問い合わせを提供できます。一方、エージェントは、1 つのインターフェイスですべての顧客に対応することができます。例えば、顧客とチャットしたり、ルーティングする際にタスクを作成したり、タスクに対応できます。

Connect Customer は顧客とのオーディオインタラクションに、Connect Customer Chat はテキストのみのインタラクションに使用できます。

詳細については、「 Connect Customer administrator guide」の以下のトピックを参照してください。

- Connect Customer とは
- Amazon Lex V2 ボットを追加する
- Connect Customer Get Customer 問い合わせブロック

コンタクトセンターが Amazon Lex V2 にリクエストを送信する際、プラットフォーム固有の情報が Lambda 関数と会話ログへのリクエスト属性として含まれています。この情報をもとに、どのコンタクトセンターアプリケーションがボットにトラフィックを送っているのかを特定します。

Amazon Connect の一般的なリクエスト属性

属性	値
x-amz-lex: チャネル: プラットフォーム	次のいずれかの値になります。 • `Connect` • `Connect Chat`

ジェネシスクラウド

Genesys Cloud は、エンタープライズコミュニケーション、コラボレーション、コンタクトセンター管理のためのクラウドサービススイートです。ジェネシスクラウドは AWS 上に構築されており、作業中の組織への安全なアクセスを提供する分散クラウド環境を使用します。

詳細については、Genesys Cloud Web サイトの以下のページを参照してください。

- Genesys Cloud コンタクトセンターについて
- Amazon Lex V2 との統合について

コンタクトセンターが Amazon Lex V2 にリクエストを送信する際、プラットフォーム固有の情報が Lambda 関数と会話ログへのリクエスト属性として含まれています。この情報をもとに、どのコンタクトセンターアプリケーションがボットにトラフィックを送っているのかを特定します。

Genesys Cloud の一般的なリクエスト属性

属性	値
x-amz-lex: チャネル: プラットフォーム	• `Genesys Cloud`

詳細はこちら

- Amazon Lex と Genesys Cloud でコンタクトセンターを強化する

Amazon Lex V2 ボットの会話について

ボットを作成したら、クライアントアプリケーションと Amazon Lex V2 ランタイムオペレーションを統合して、ボットと会話を行います。ユーザーがボットとの会話を開始すると、Amazon Lex V2 は セッション を作成します。セッションは、アプリケーションとボットの間で交換される情報をカプセル化します。詳細については、「Amazon Lex V2 ボットセッションについて」を参照してください。

一般的な会話には、ユーザーとボットの間の前後の流れが含まれます。例えば、次のようになります。

```
User : I'd like to make an appointment
Bot : What type of appointment would you like to schedule?
User : dental
Bot : When should I schedule your dental appointment?
User : Tomorrow
Bot : At what time do you want to schedule the dental appointment on 2021-01-01?
User : 9 am
Bot : 09:00 is available, should I go ahead and book your appointment?
User : Yes
Bot : Thank you. Your appointment has been set successfully.
```

RecognizeText または RecognizeUtterance API オペレーションを使用して、自分で会話を管理します。StartConversation オペレーションを使用する場合、Amazon Lex V2 がユーザーに代わって会話を管理します。

会話を管理するには、会話が論理的な終了に達するまで、ユーザーの発言をボットに送信する必要があります。現在の会話は、セッション状態に取り込まれます。セッションの状態は、ユーザーの発話があるたびに更新されます。セッションの状態には、会話の現在の状態が含まれており、ボットが応答で返します。 各ユーザーの発話に対応します。

会話は、次に示す状態のいずれかになります。

- ElicitIntent - ボットがまだユーザーの意図を決定していないことを示します。
- ElicitSlot - ボットがユーザーの意図を検出し、その意図を満たすために必要な情報を収集していることを示します。
- ConfirmIntent - ボットが、収集した情報が正しいかどうかをユーザーが確認するのを待っていることを示します。

- Closed - ユーザーの意図が完了し、ボットとの会話が論理的な終了に達したことを示します。

ユーザーは、最初のインテントが完了した後に、新しいインテントを指定することができます。詳細については、「Lex V2 ボットとの会話コンテキスト」を参照してください。

インテントは、次に示す状態のいずれかになります。

- InProgress - ボットがインテントを完了するために必要な情報を収集していることを示します。これは、`ElicitSlot` 会話状態と連動しています。
- Waiting - ボットが特定のスロットの情報をリクエストしたときに、ユーザーがボットに待機するよう要求したことを示します。
- Fulfilled - インテントに関連する Lambda 関数内のビジネスロジックが正常に実行されたことを示します。
- ReadyForFulfillment - ボットがインテントを満たすために必要なすべての情報を収集し、クライアントアプリケーションがフルフィルメントビジネスロジックを実行できることを示します。
- Failed - インテントが失敗したことを示します。

Amazon Lex V2 API を使用してボットとユーザー間の会話コンテキストとセッションを管理する方法については、以下のトピックを参照してください。

トピック

- Lex V2 ボットとの会話コンテキスト
- Amazon Lex V2 ボットセッションについて

Lex V2 ボットとの会話コンテキスト

会話コンテキストは、ユーザー、クライアントアプリケーション、または Lambda 関数がインテントを達成するために Amazon Lex V2 ボットに提供する情報です。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

- Lex V2 ボットのインテントコンテキストの設定
- Lex V2 ボットのインテントでのデフォルトスロット値の使用
- Lex V2 ボットのセッション属性の設定

- Lex V2 ボットのリクエスト属性の設定
- セッションタイムアウトを設定する
- Lex V2 ボットでのインテント間の情報共有
- Lex V2 ボットでの複雑な属性の設定

Lex V2 ボットのインテントコンテキストの設定

コンテキストに基づいて Amazon Lex V2 でインテントをトリガーできます。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 CreateIntent 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。コンテキストは、英語 (US) (en-US) ロケールでのみ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入力コンテキストの 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、RecognizeText オペレーション、または RecognizeUtterance 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 V2 は、出力コンテキストでインテントを満たすことでアクティブ化される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、`RecognizeText` オペレーション、または `RecognizeUtterance` 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 V2 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、「AWS Lambda関数を Amazon Lex V2 ボットに統合する」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。インテントが達成されると、Amazon Lex V2 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誘発する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。

Lex V2 ボットのインテントの出力コンテキスト

Amazon Lex V2 は、インテントが満たされると、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、「Lex V2 ボットのインテントでのデフォルトスロット値の使用」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 CreateIntent 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、RecognizeText または RecognizeUtterance 応答で返されます。

出力コンテキストを定義するときは、その有効期限、コンテキストが Amazon Lex V2 からのレスポンスに含まれる時間またはターン数も定義します。ターンは、アプリケーションから Amazon Lex V2 への 1 つのリクエストです。 V2 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 V2 に返します。Amazon Lex V2 は、発話がインテントを引き出す試みであると認識した場合、 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インテントに制限します。

Lex V2 ボットのインテントの入力コンテキスト

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 CreateIntent 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。

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。RecognizeText、RecognizeUtterance、または PutSession オペレーションを呼び出すときに、入力コンテキストを設定できます。

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 V2 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、「Lex V2 ボットのインテントでのデフォルトスロット値の使用」を参照してください。

Lex V2 ボットのインテントでのデフォルトスロット値の使用

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

CreateIntent 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": [
    {
        "botId": "string",
        "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. 最初の 2 つのデフォルト値のいずれも使用されていない場合、スロットにはデフォルト値がなく、Amazon Lex V2 は通常どおり値を取得します。

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誘発されることはありません。

Lex V2 ボットの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 V2 は、ボット用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 関数がセッション属性を追加または更新した場合、Amazon Lex V2 は新しい情報をクライアントアプリケーションに返します。

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。例えば、次のようになります。

- 初期化 - ピザの注文ボットにおいて、RecognizeText または RecognizeUtterance 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": "Vivian"}) として渡すと、Amazon Lex はプレースホルダーをその名前に置き換えます。そして「ビビアン、どのトッピングがいい？」というパーソナライズされたプロンプトをユーザーに送ります。

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 V2 は、セッションが終了するまで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、sessionAttributes フィールドに値を設定して RecognizeText または RecognizeUtterance オペレーションを呼び出すことで、リクエスト内でセッション属性を作成することができます。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 関数が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が Amazon Lex V2 へのリクエストにsessionAttributeフィールドを含めないたびに、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sessionAttributesフィールドを指定せずに RecognizeTextまたは RecognizeUtteranceオペレーションを呼び出す場合、Amazon Lex V2 は保存されたセッション属性 () を使用して Lambda 関数を呼び出します{"x": 1, "y": 2}。Lambda 関数がセッション属性を返さない場合、Amazon Lex V2 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 V2 は保存されたセッション属性を更新します。 V2 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 V2 に送信するには、属性のstring-to-string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"
}
```

RecognizeText のオペレーションでは、次のように、sessionAttributes 構造の sessionState フィールドを使って、マップをリクエストのボディに挿入します。

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

```
}
```

RecognizeUtterance のオペレーションでは、マップを base64 でエンコードし、x-amz-lex-session-state ヘッダーの一部として送信します。

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、「Lex V2 ボットでの複雑な属性の設定」を参照してください。

事前定義されたセッション属性

Amazon Lex には、事前定義されたセッション属性が用意されています。これらの属性は、Amazon Lex がボットに送信された情報を処理する方法を管理します。属性はセッション全体で保持されます。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。

コードフック Lambda タイムアウトの増加

デフォルトでは 30 秒であるボットのコードフック Lambda タイムアウト値を増やすには、x-amz-lex:codehook-timeout-msセッション属性を使用します。セッション属性値をミリ秒単位で設定します。最大タイムアウトは 120 秒です。

たとえば、ユーザーがx-amz-lex:codehook-timeout-ms」：「90000」を設定すると、コードフックの Lambda タイムアウトは 90 秒になります。

Lex V2 ボットの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 V2 に送信します。 V2 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、RecognizeUtterance の x-amz-lex-request-attributes ヘッダー、または RecognizeText 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、RecognizeUtterance レスポンスや RecognizeText 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。

Lex V2 ボットリクエストごとの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、RecognizeUtterance リクエストの amz-lex-request-attributes ヘッダーを使用するか、RecognizeText リクエストの requestAttributes フィールドを使用します。

Amazon Lex V2 にリクエスト属性を送信するには、属性のstring-to-string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、「Lex V2 ボットでの複雑な属性の設定」を参照してください。

Amazon Lex V2 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 および 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 プレーン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 V2 はレスポンスの [メッセージ] 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、「Message Groups」(メッセージグループ) を参照してください。

Lex V2 ボットの事前定義されたリクエスト属性の設定

Amazon Lex V2 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。

Lex V2 ボットでインテントスイッチを無効にする

ユーザーがインテントの確認中またはスロット誘発中にインテントを切り替えることができるかどうかを制御するには、x-amz-lex:intent-switch リクエスト属性を使用します。DISABLE に設定すると、この属性により、ユーザーは現在のインテントフローの完了処理中に別のインテントをトリガーできなくなります。

例えば、ユーザーがフライトを予約中であり、フライトの詳細を求められている場合、通常は他のインテントをトリガーする「天気の確認」や「ホテルの予約」などの発話は無視され、会話は現在の予約プロセスに集中し続けます。

セッションタイムアウトを設定する

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を提供しても、靴の購入が完了しなかった場合、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。もしユーザーが期限切れ前にセッションに戻れば、残りのスロットデータを提供し、購入を完了させることができます。

Amazon Lex V2 コンソールでは、ボットの作成時にセッションタイムアウトを設定します。AWS コマンドラインインターフェイス (AWS CLI) または API では、CreateBot オペレーションを使用してボットを作成するときに、idleSessionTTLInSeconds フィールドを設定してタイムアウトを設定します。

Lex V2 ボットでのインテント間の情報共有

Amazon Lex V2 は、インテント間の情報共有をサポートしています。インテント間で共有するには、出力コンテキストまたはセッション属性を使用します。

出力コンテキストを使用するには、インテントを作成または更新するときに、出力コンテキストを定義します。インテントが実行されると、Amazon Lex V2 からの応答は、コンテキストパラメータとしてインテントからのコンテキストとスロット値を含みます。これらのパラメータは、後続のインテントやアプリケーションコード、Lambda 関数でデフォルト値として使用することができます。

セッション属性を使用するには、Lambda またはアプリケーションコードで属性を設定します。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。

Lex V2 ボットでの複雑な属性の設定

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 RecognizeText オペレーションと RecognizeUtterance 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 RecognizeUtterance オペレーションで送信するには、次の JavaScript コードのように、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 RecognizeText オペレーションと RecognizeUtterance 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"
}
```

Amazon Lex V2 ボットセッションについて

ユーザーがボットとの会話を開始すると、Amazon Lex V2 は セッション を作成します。アプリケーションと Amazon Lex V2 の間で交換される情報は、会話のためのセッション状態を構成しています。リクエストを行うと、セッションは指定した識別子で識別されます。セッション識別子の詳細については、RecognizeText オペレーション、または RecognizeUtterance オペレーションの sessionId フィールドを参照してください。

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。

セッションの状態を更新するには、3 つの方法があります。

- RecognizeText オペレーション、または RecognizeUtterance オペレーションの呼び出しの一部として、セッション情報をインラインで渡します。
- 会話の各順番の後に呼び出される RecognizeText オペレーション、または RecognizeUtterance オペレーションで Lambda 関数を使用します。詳細については、「AWS Lambda関数を Amazon Lex V2 ボットに統合する」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 Amazon Lex V2 のランタイム API を使用して、セッションの状態を変更することです。
- 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使用します。オペレーションには、PutSession オペレーション、GetSession オペレーション、DeleteSession オペレーションがあります。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、ユーザーとの対話の状態、現在のインテントのために設定されたセッション属性とスロット値、Amazon Lex V2 がユーザーの発話と一致する可能性のあるインテントとして特定した他のインテントを含む、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。ボットが次に実行するダイアログアクションの種類や、Amazon Lex V2 がユーザーに送信するメッセージなど、セッションを設定することができます。これにより、ボットとの会話のフローを制御できます。Amazon Lex V2 がボットの次のアクションを決定するように、ダイアログアクション type フィールドを Delegate に設定します。

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。

PutSession オペレーションからのレスポンスには、RecognizeUtterance オペレーションと同じ情報が含まれます。RecognizeUtterance 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。

新しいセッションを開始する

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 操作を呼び出します。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。

インテントの切り替えをする

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 操作を呼び出します。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。

前のインテントを再開する

前のインテントを再開するには、GetSession オペレーションでインテントの状態を取得し、必要な対話を実行して、PutSession オペレーションでインテントを前の対話の状態に設定します。

- GetSession 操作を呼び出します。インテントの状態を保存する
- 別のインテントを満たすなど、別のインタラクションを実行します。
- 前のインテントのために保存された情報を使って、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。

スロット値の検証をする

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誘

発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。

AWS Lambda関数を Amazon Lex V2 ボットに統合する

AWS Lambda 関数を使用すると、定義したカスタム関数を使用して Amazon Lex V2 ボットの動作を拡張すると同時に、より適切に制御できます。Amazon Lex V2 は、インテントごとではなく、各言語のボットのエイリアスごとに 1 つの Lambda 関数を使用します。開始する前に、入力イベント内のどのフィールドから情報を引き出すか、および応答内のどのフィールドを操作して Lambda 関数から返すかを決定します。

Lambda 関数を Amazon Lex V2 ボットと統合するには、次の手順を実行します。

1. 選択したプログラミング言語AWS Lambdaを使用して で関数を作成し、スクリプトを書き込みます。
2. 関数が応答形式と一致する構造を返すことを確認してください。
3. Lambda 関数をデプロイします。
4. Lambda 関数をコンソールまたは API オペレーションの Amazon Lex V2 ボットエイリアスに関連付けます。
5. コンソールまたは API オペレーションで Lambda 関数を呼び出す会話ステージを選択します。
6. Amazon Lex V2 ボットを構築し、Lambda 関数がインテント通りに機能することをテストします。Amazon CloudWatch を使用して関数をデバッグします。

トピック

- AWS LambdaLex V2 の入力イベント形式
- AWS LambdaLex V2 のレスポンス形式
- Amazon Lex V2 の AWS Lambda関数の一般的な構造
- Amazon Lex V2 ボットの AWS Lambda関数の作成
- CloudWatch Logs のログを使用した Lambda 関数のデバッグ

AWS LambdaLex V2 の入力イベント形式

Lambda 関数を Amazon Lex V2 ボットに統合する最初のステップは、Amazon Lex V2 イベントのフィールドを理解し、これらのフィールドからスクリプトを書くときに使用する情報を決定することです。次の JSON オブジェクトに、Lambda 関数に渡される一般的な形式の Amazon Lex V2 イベントを示します。

> **Note**
>
> 入力形式は対応する messageVersion への変更なしに変わる場合があります。新しいフィールドがあっても、コードがエラーをスローすることはありません。

```
{
    "messageVersion": "1.0",
    "invocationSource": "DialogCodeHook | FulfillmentCodeHook",
    "inputMode": "DTMF | Speech | Text",
    "responseContentType": "audio/mpeg | audio/ogg | audio/pcm | text/plain;
 charset=utf-8",
    "sessionId": string,
    "inputTranscript": string,
    "invocationLabel": string,
    "bot": {
        "id": string,
        "name": string,
        "localeId": string,
        "version": string,
        "aliasId": string,
        "aliasName": string
    },
    "interpretations": [
        {
            "interpretationSource": "Bedrock | Lex",
            "intent": {
                // see Intent for details about the structure
            },
            "nluConfidence": number,
            "sentimentResponse": {
                "sentiment": "MIXED | NEGATIVE | NEUTRAL | POSITIVE",
                "sentimentScore": {
                    "mixed": number,
                    "negative": number,
                    "neutral": number,
                    "positive": number
                }
            }
        },
        ...
    ],
```

```
    "proposedNextState": {
        "dialogAction": {
            "slotToElicit": string,
            "type": "Close | ConfirmIntent | Delegate | ElicitIntent | ElicitSlot"
        },
        "intent": {
            // see Intent for details about the structure
        },
        "prompt": {
            "attempt": string
        }
    },
    "requestAttributes": {
        string: string,
        ...
    },
    "sessionState": {
        // see ####### for details about the structure
    },
    "transcriptions": [
        {
            "transcription": string,
            "transcriptionConfidence": number,
            "resolvedContext": {
                "intent": string
            },
            "resolvedSlots": {
                slot name: {
                    // see Slots for details about the structure
                },
                ...
            }
        },
        ...
    ]
}
```

入力イベントの各フィールドは次の通りです。

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 Amazon Lex V2 がサポートしているのは、メッセージバージョンのみです。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。

invocationSource

Lambda 関数を呼び出したコードフック。以下の値を指定できます。

`DialogCodeHook` — Amazon Lex V2 は、ユーザーからの入力後に Lambda 関数を呼び出します。

`FulfillmentCodeHook` — Amazon Lex V2 は、必要なスロットがすべて満たされ、インテントフルフィルメントの準備ができた後に Lambda 関数を呼び出します。

inputMode

ユーザー発話のモード。指定できる値は次のとおりです。

`DTMF` — ユーザーがタッチトーンキーパッド (デュアルトーンマルチ周波数) を使用して発話を入力します。

`Speech` — ユーザーが発話を行いました。

`Text` — ユーザーが発話を入力しました。

responseContentType

ユーザーに対するボットの応答の仕組み。`text/plain; charset=utf-8` は、最後の発話が書き込まれたことを示し、`audio` で始まる値は最後の発話が行われたことを示します。

sessionId

会話に使用される英数字のセッション識別子。

inputTranscript

ユーザーからの入力の文字起こし。

- テキスト入力の場合、これはユーザーが入力したテキストです。DTMF 入力の場合、これはユーザーが入力するキーです。

- 音声入力の場合は、インテントを呼び出すか、スロットを埋めるために、Amazon Lex V2 がユーザーの発話を変換したテキストです。

invocationLabel

Lambda 関数を呼び出した応答を示す値。初期応答、スロット、確認応答に呼び出しラベルを設定できます。

ボット

リクエストを処理したボットに関する情報。以下のフィールドで構成されます。

- id - その作成時にボットに割り当てた識別子。ボットの Amazon Lex V2 コンソールでボット ID は、設定 ページを参照してください。
- Name - ジョブを作成したときにジョブに指定する名前。
- localeId — ボットに使用したロケールの識別子。ロケールのリストについては、「Amazon Lex V2 でサポートされている言語とロケール」を参照してください。
- version - リクエストを処理したボットのバージョン。
- aliasId — ボットのエイリアスの作成時に割り当てた識別子。ボットのエイリアス ID は、Amazon Lex V2 コンソールの エイリアス ページで参照してください。リストにエイリアス ID が表示されない場合は、エイリアス ID で右上の歯車アイコンを選択してオンにします。
- aliasName — ボットエイリアスに付けた名前。

解釈

Amazon Lex V2 がユーザーの発話と一致する可能性があると考えるインテントに関する情報のリスト。各項目は、発話とインテントの一致に関する情報を次の形式で提供する構造です。

```
{
    "intent": {
        // see Intent for details about the structure
    },
    "interpretationSource": "Bedrock | Lex",
    "nluConfidence": number,
    "sentimentResponse": {
        "sentiment": "MIXED | NEGATIVE | NEUTRAL | POSITIVE",
        "sentimentScore": {
```

```
            "mixed": number,
            "negative": number,
            "neutral": number,
            "positive": number
        }
    }
}
```

構造内のフィールドは以下のとおりです。

- intent – インテントに関する情報が含まれる構造。構造の詳細については、「Intent」を参照してください。
- nluConfidence – インテントがユーザーのインテントに一致しているという Amazon Lex V2 の信頼度を示すスコア。
- sentimentResponse — 応答のセンチメントの分析。以下のフィールドが含まれます。
  - sentiment — 発話のセンチメントが POSITIVE、NEGATIVE、NEUTRAL、または MIXED であるかどうかを示します。
  - sentimentScore — 各センチメントを数値にマッピングした構造で、Amazon Lex V2 が発話によるセンチメントの伝達に関する信頼度を示します。
- interpretationSource – スロットが Amazon Lex V2 または Amazon Bedrock によって解決されるかどうかを示します。

proposedNextState

Lambda 関数が sessionState の dialogAction を Delegate に設定するとこのフィールドが表示され、会話の次のステップに関する Amazon Lex V2 の提案を確認できます。それ以外の場合、次の状態は、Lambda 関数からの応答で返される設定によって異なります。この構造は、次の両方に該当する場合のみ存在します。

1. invocationSource 値は DialogCodeHook である
2. 予測された dialogAction の type は ElicitSlot です。

この情報を使用して、会話の適切な時点で runtimeHints を追加できます。詳細については、「会話内のランタイムヒントによるスロット値の認識の向上」を参照してください。proposedNextState は、次のフィールドを含む構造です。

proposedNextState の構造は以下のとおりです。

```
"proposedNextState": {
    "dialogAction": {
        "slotToElicit": string,
        "type": "Close | ConfirmIntent | Delegate | ElicitIntent | ElicitSlot"
    },
    "intent": {
        // see Intent for details about the structure
    },
    "prompt": {
        "attempt": string
    }
}
```

- dialogAction — Amazon Lex V2 が提案する次のステップに関する情報が含まれます。構造内のフィールドは以下のとおりです。
  - slotToElicit — Amazon Lex V2で提案された通り、次に誘発するスロットです。このフィールドは、type が ElicitSlot の場合のみ表示されます。
  - type — Amazon Lex V2 が提案している会話の次のステップです。以下の値を指定できます。

    Delegate - Amazon Lex V2 が次のアクションを決定します。

    ElicitIntent - 次のアクションは、ユーザーからインテントを誘発することです。

    ElicitSlot - 次のアクションは、ユーザーからスロット値を誘発することです。

    Close — インテントフルフィルメントのプロセスを終了し、ユーザーからの応答が返されないことを示します。

    ConfirmIntent — 次のアクションは、スロットが正しく、インテントのフルフィルメントの準備ができているかどうかをユーザーに尋ねることです。
- intent — ユーザーが達成しようとしているとボットが判断したインテント。構造の詳細については、「Intent」を参照してください。
- prompt — Amazon Lex V2 がユーザーに次のスロットの入力を促した回数を指定する値に対応するフィールド attempt を含む構造。指定できる値は、1 回目の試行に対する Initial、それ以降の試行に対する Retry1、Retry2、Retry3、Retry4 および Retry5 です。

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を含む構造。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、「Lex V2 ボットのリクエスト属性の設定」を参照してください。

sessionState

ユーザーと Amazon Lex V2 ボット間の会話の現在の状態。構造の詳細については、「セッション状態」を参照してください。

文字起こし

Amazon Lex V2 がユーザーの発話と一致する可能性があると考える文字起こしのリスト。詳細については、「音声文字起こしの信頼度スコアを使用した Lex V2 ボットとの会話の改善」を参照してください。各項目は次の形式のオブジェクトで、1 つの可能な文字起こしに関する情報が含まれています。

```
{
    "transcription": string,
    "transcriptionConfidence": number,
    "resolvedContext": {
        "intent": string
    },
    "resolvedSlots": {
        slot name: {
            // see Slots for details about the structure
        },
        ...
    }
}
```

各フィールドについて以下に説明します。

- transcription – Amazon Lex V2 がユーザーの音声発話と一致する可能性があると考える文字起こし。
- transcriptionConfidence – インテントがユーザーのインテントに一致しているという Amazon Lex V2 の信頼度を示すスコア。
- resolvedContext — フィールド `intent` を含む構造で、発話に関係するインテントと一致します。

- resolvedSlots — 発話によって解決される各スロットの名前をキーとする構造。各スロット名は、そのスロットに関する情報を含む構造にマップされます。構造の詳細については、「Slots」を参照してください。

AWS LambdaLex V2 のレスポンス形式

Lambda 関数を Amazon Lex V2 ボットに統合する 2 番目のステップは、Lambda 関数応答のフィールドを理解し、操作するパラメータを決定することです。次の JSON オブジェクトは、Amazon Lex V2 に返される一般的な形式の Lambda レスポンスを示しています。

```
{
    "sessionState": {
        // see ####### for details about the structure
    },
    "messages": [
        {
            "contentType": "CustomPayload | ImageResponseCard | PlainText | SSML",
            "content": string,
            "imageResponseCard": {
                "title": string,
                "subtitle": string,
                "imageUrl": string,
                "buttons": [
                    {
                        "text": string,
                        "value": string
                    },
                    ...
                ]
            }
        },
        ...
    ],
    "requestAttributes": {
        string: string,
        ...
    }
}
```

応答の各フィールドは次の通りです。

sessionState

ユーザーと返したい Amazon Lex V2 ボット間の会話の状態。構造の詳細については、「セッション状態」を参照してください。このフィールドは常に必須です。

メッセージ

会話の次のターンに Amazon Lex V2 が返すメッセージのリスト。提供する `contentType` が `PlainText`、`CustomPayload`、または `SSML` の場合、顧客に返すメッセージを `content` フィールドに書き込みます。提供する `contentType` が `ImageResponseCard` の場合、`imageResponseCard` フィールドにカードの詳細を入力します。メッセージを指定しない場合、Amazon Lex V2 はボットの作成時に定義された適切なメッセージを使用します。

`dialogAction.type` が `ElicitIntent` または `ConfirmIntent` の場合、`messages` フィールドは必須です。

リスト内の各項目は次の形式の構造であり、ユーザーに返すメッセージに関する情報が含まれています。以下がその例です。

```
{
    "contentType": "CustomPayload | ImageResponseCard | PlainText | SSML",
    "content": string,
    "imageResponseCard": {
        "title": string,
        "subtitle": string,
        "imageUrl": string,
        "buttons": [
            {
                "text": string,
                "value": string
            },
            ...
        ]
    }
}
```

各フィールドの説明を以下に示します。

- [ContentType] — 使用するメッセージのタイプ。

  `CustomPayload` — アプリケーションのデータまたはメタデータを含むようにカスタマイズできる応答文字列。

ImageResponseCard — ユーザーが選択できるボタン付きの画像。詳細については、「ImageResponseCard」を参照してください。

PlainText — プレーンテキスト文字列。

SSML — 音声応答をカスタマイズするための音声合成マークアップ言語を含む文字列。

- content – ユーザーに送信するメッセージ。メッセージタイプが PlainText、CustomPayload、または SSML の場合、このフィールドを使用してください。
- imageResponseCard — ユーザーに表示する応答カードの定義が含まれます。メッセージタイプが ImageResponseCard の場合、このフィールドを使用してください。次のフィールドを含む構造にマップします。
  - title — レスポンスカードのタイトル。
  - subtitle — ボタンを選択するようユーザーに求めるプロンプト。
  - imageUrl — カードの画像へのリンク。
  - buttons — ボタンに関する情報を含む構造のリスト。各構造には、表示するテキストを含む text フィールドと、顧客がそのボタンを選択した場合に Amazon Lex V2 に送信される値を含む value フィールドが含まれます。最大 3 つのボタンを含めることができます。

requestAttributes

顧客への応答に必要な要求固有の属性を含む構造。詳細については「Lex V2 ボットのリクエスト属性の設定」を参照してください。このフィールドはオプションです。

応答内の必須フィールド

最低でも、Lambda 応答には sessionState オブジェクトを含める必要があります。その中で、dialogAction オブジェクトを提供し、type フィールドを指定します。提供する dialogAction の type によっては、Lambda 応答に他の必須フィールドがある場合があります。これらの要件は、最小限の実例とともに次のように説明されています。

受任者

Delegate は、Amazon Lex V2 が次のアクションを決定できるようにします。その他に必須フィールドはありません。

```
{
```

```
    "sessionState": {
        "dialogAction": {
            "type": "Delegate"
    }
}
```

ElicitIntent

ElicitIntent は、顧客にインテントを表明するよう促します。インテントを誘発するには、messages フィールドに少なくとも 1 つのメッセージを含める必要があります。

```
{
    "sessionState": {
        "dialogAction": {
            "type": "ElicitIntent"
    },
    "messages": [
        {
            "contentType": PlainText,
            "content": "How can I help you?"
        }
    ]
}
```

ElicitSlot

ElicitSlot は、顧客にスロット値を入力するよう求めます。dialogAction オブジェクトの slotToElicit フィールドにはスロットの名前を含める必要があります。sessionState オブジェクトには intent の name も含める必要があります。

```
{`
    "sessionState": {
        "dialogAction": {
            "slotToElicit": "OriginCity",
            "type": "ElicitSlot"
        },
        "intent": {
            "name": "BookFlight"
        }
    }
}
```

ConfirmIntent

ConfirmIntent は、顧客のスロット値と、そのインテントを達成する準備ができているかどうかを確認します。sessionState オブジェクトの intent の name と、確認する slots を含める必要があります。また、messages フィールドには、ユーザーにスロット値の確認を求めるメッセージを少なくとも 1 つ含める必要があります。メッセージに「はい」か「いいえ」の応答を求める必要があります。ユーザーの応答が「はい」であった場合、Amazon Lex V2 はインテントの confirmationState を Confirmed に設定します。ユーザーの応答が「いいえ」であった場合、Amazon Lex V2 はインテントの confirmationState を Denied に設定します。

```
{
    "sessionState": {
        "dialogAction": {
            "type": "ConfirmIntent"
        },
        "intent": {
            "name": "BookFlight",
            "slots": {
                "DepartureDate": {
                    "value": {
                        "originalValue": "tomorrow",
                        "interpretedValue": "2023-05-09",
                        "resolvedValues": [
                            "2023-05-09"
                    ]
                 }
                },
                "DestinationCity": {
                    "value": {
                        "originalValue": "sf",
                        "interpretedValue": "sf",
                        "resolvedValues": [
                            "sf"
                        ]
                    }
                },
                "OriginCity": {
                    "value": {
                        "originalValue": "nyc",
                        "interpretedValue": "nyc",
                        "resolvedValues": [
                            "nyc"
```

```
                        ]
                    }
                }
            }
        }
    },
    "messages": [
        {
            "contentType": PlainText,
            "content": "Okay, you want to fly from {OriginCity} to \
            {DestinationCity} on {DepartureDate}. Is that correct?"
        }
    ]
}
```

閉じる

Close は、インテントのフルフィルメントプロセスを終了し、これ以上ユーザーからの応答が予想されないことを通知します。sessionState オブジェクトには intent の name と state を含める必要があります。互換性のあるインテントステートはFailed、Fulfilled、および InProgressです。

```
"sessionState": {
    "dialogAction": {
        "type": "Close"
    },
    "intent": {
        "name": "BookFlight",
        "state": "Failed | Fulfilled | InProgress"
    }
}
```

Amazon Lex V2 の AWS Lambda関数の一般的な構造

Lambda レスポンスには、繰り返し発生する構造が数多くあります。これらの一般的な構造について詳しくは、このセクションで説明しています。

Intent

```
"intent": {
    "confirmationState": "Confirmed | Denied | None",
```

```
    "name": string,
    "slots": {
        // see Slots for details about the structure
    },
    "state": "Failed | Fulfilled | FulfillmentInProgress | InProgress |
 ReadyForFulfillment | Waiting",
    "kendraResponse": {
        // Only present when intent is KendraSearchIntent. For details, see
// https://docs.aws.amazon.com/kendra/latest/dg/API_Query.html#API_Query_ResponseSyntax
       }
}
```

intent フィールドは、次のフィールドを持つオブジェクトにマップされます。

confirmationState

ユーザーがインテントのスロットを確認し、インテントフルフィルメントの準備ができているかどうかを示します。以下の値を指定できます。

Confirmed — ユーザーはスロット値が正しいことを確認します。

Denied — ユーザーがスロット値が正しくないことを示しています。

None — ユーザーはまだ確認ステージに達していません。

名前

インテントの名前。

slots

インテントを達成するために必要なスロットに関する情報。構造の詳細については、「Slots」を参照してください。

state

インテントのフルフィルメントステータスを示します。以下の値を指定できます。

Failed — ボットはインテントを達成できませんでした。

Fulfilled — ボットはインテントのフルフィルメントを完了しました。

FulfillmentInProgress — ボットはインテントを達成している最中です。

InProgress — ボットは、インテントを達成するために必要なスロット値を誘発している最中です。

ReadyForFulfillment — ボットはインテントのすべてのスロット値を誘発し、インテントを実行する準備ができています。

Waiting — ボットはユーザーからの応答を待っています (会話のストリーミングに限定される)。

kendraResponse

Kendra 検索クエリの結果に関する情報が含まれます。このフィールドは、インテントが KendraSearchIntent の場合のみ表示されます。詳細については、「Kendra の Query API 呼び出しの応答構文」を参照してください。

Slots

slots フィールドは intent 構造内に存在し、そのインテントのスロットの名前をキーとする構造にマップされます。スロットが複数値スロットでない場合 (詳細は スロットで複数の値を使用する を参照)、次の形式の構造体にマップされます。shape は Scalar であることに注意してください。

```
{
    slot name: {
        "shape": "Scalar",
        "value": {
            "originalValue": string,
            "interpretedValue": string,
            "resolvedValues": [
                string,
                ...
            ]
        }
    }
}
```

スロットが複数値スロットの場合、マップ先のオブジェクトには values という別のフィールドが含まれます。このフィールドは構造のリストにマップされ、各構造には多値スロットを構成するスロットに関する情報が含まれます。リスト内の各オブジェクトの形式は、通常のスロットがマップされているオブジェクトの形式と一致します。shape は List ですが、values の下のコンポーネントスロットの shape は Scalar であることに注意してください。

```
{
```

```
    slot name: {
    "shape": "List",
    "value": {
        "originalValue": string,
        "interpretedValue": string,
        "resolvedValues": [
            string,
            ...
        ]
    },
    "values": [
        {
            "shape": "Scalar",
            "value": {
                "originalValue": string,
                "interpretedValue": string,
                "resolvedValues": [
                    string,
                    ...
                ]
            }
        },
        {
            "shape": "Scalar",
            "value": {
                "originalValue": string,
                "interpretedValue": string,
                "resolvedValues": [
                    string,
                    ...
                ]
            }
        },
        ...
    ]
}
```

スロットオブジェクトのフィールドは次のとおりです。

shape

スロットの形状。この値は、スロットに複数の値がある場合は List で (詳細については「スロットで複数の値を使用する」を参照)、そうでない場合は Scalar です。

値

ユーザーがスロットと Amazon Lex V2 の解釈に提供した値に関する情報を次の形式で含むオブジェクト。

```
{
    "originalValue": string,
    "interpretedValue": string,
    "resolvedValues": [
        string,
        ...
    ]
}
```

各フィールドについて以下に説明します。

- originalValue – Amazon Lex V2 がスロット値に関連していると判断したスロット誘発に対するユーザーのレスポンスの一部。
- interpretedValue – ユーザー入力を考慮して Amazon Lex V2 がスロットに対して決定する値。
- resolvedValues – Amazon Lex V2 がユーザー入力で解決できると判断した値のリスト。

値

複数値スロットを構成するスロットに関する情報を含むオブジェクトのリスト。各オブジェクトの形式は、上記で説明した shape および value フィールドを含む通常のスロットの形式と一致します。values は、スロットが複数の値で構成されている場合にのみ表示されます (詳細は「スロットで複数の値を使用する」を参照)。次の JSON オブジェクトは 2 つのコンポーネントスロットを示しています。

```
"values": [
    {
        "shape": "Scalar",
        "value": {
            "originalValue": string,
            "interpretedValue": string,
            "resolvedValues": [
                string,
                ...
            ]
        }
```

```
    },
    {
        "shape": "Scalar",
        "value": {
            "originalValue": string,
            "interpretedValue": string,
            "resolvedValues": [
                string,
                ...
            ]
        }
    },
    ...
]
```

セッション状態

sessionState フィールドは、ユーザーとの会話の状態に関する情報を含むオブジェクトにマップされます。オブジェクトに表示される実際のフィールドは、ダイアログアクションのタイプによって異なります。レスポンスの必須フィールドについては、「応答内の必須フィールド」を参照してください。sessionState オブジェクトの形式は次のとおりです。

```
"sessionState": {
    "activeContexts": [
        {
            "name": string,
            "contextAttributes": {
                string: string
            },
            "timeToLive": {
                "timeToLiveInSeconds": number,
                "turnsToLive": number
            }
        },
        ...
    ],
    "sessionAttributes": {
        string: string,
        ...
    },
    "runtimeHints": {
        "slotHints": {
            intent name: {
```

```
                slot name: {
                    "runtimeHintValues": [
                        {
                            "phrase": string
                        },
                        ...
                    ]
                },
                ...
            },
            ...
        }
    },
    "dialogAction": {
        "slotElicitationStyle": "Default | SpellByLetter | SpellByWord",
        "slotToElicit": string,
        "type": "Close | ConfirmIntent | Delegate | ElicitIntent | ElicitSlot"
    },
    "intent": {
        // see Intent for details about the structure
    },
    "originatingRequestId": string
}
```

各フィールドについて以下に説明します。

activeContexts

ユーザーがセッションで使用しているコンテキストに関する情報を含むオブジェクトのリスト。コンテキストを使用してインテントを認識しやすくし、制御します。コンテキストの詳細については、「Lex V2 ボットのインテントコンテキストの設定」を参照してください。各オブジェクトの形式は次のとおりです:

```
{
    "name": string,
    "contextAttributes": {
        string: string
    },
    "timeToLive": {
        "timeToLiveInSeconds": number,
        "turnsToLive": number
    }
}
```

各フィールドについて以下に説明します。

- name – コンテキストの名前。
- contextAttributes — コンテキストの属性名とそれらがマップされる値を含むオブジェクト。
- timeToLive — コンテキストがアクティブである期間を指定するオブジェクト。このオブジェクトには、次の一方または両方のフィールドを含めることができます。
  - timeToLiveInSeconds — コンテキストがアクティブのままである秒数。
  - turnsToLive – コンテキストがアクティブなままのターン数。

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。詳細については、「Lex V2 ボットのセッション属性の設定」を参照してください。オブジェクトの形式は次のとおりです:

```
{
    string: string,
    ...
}
```

runtimeHints

音声認識を改善するために、ユーザーがスロットに使用する可能性が高いフレーズのヒントを提供します。ヒントに入力した値によって、似たような言葉よりもそれらの値の音声認識が向上します。runtimeHints オブジェクトの形式は次のとおりです。

```
{
    "slotHints": {
        intent name: {
            slot name: {
                "runtimeHintValues": [
                    {
                        "phrase": string
                    },
                    ...
                ]
            },
            ...
        },
        ...
```

```
    }
}
```

slotHints フィールドは、ボットのインテントの名前をフィールドとするオブジェクトにマップされます。各インテント名は、そのインテントのスロットの名前をフィールドとするオブジェクトにマップされます。各スロット名は、オブジェクトのリストである 1 つのフィールド runtimeHintValues を持つ構造にマップされます。各オブジェクトには、ヒントに対応する phrase フィールドが含まれています。

dialogAction

Amazon Lex V2 が次に実行するアクションを決定します。オブジェクトの形式は次のとおりです。

```
{
    "slotElicitationStyle": "Default | SpellByLetter | SpellByWord",
    "slotToElicit": string,
    "type": "Close | ConfirmIntent | Delegate | ElicitIntent | ElicitSlot"
}
```

各フィールドについて以下に説明します。

- slotElicitationStyle — dialogAction の type が ElicitSlot である場合、ユーザーからの音声入力を Amazon Lex V2 がどのように解釈するかを決定します。詳細については、「会話中のスペルスタイルによるスロット値のキャプチャ」を参照してください。以下の値を指定できます。

  Default — Amazon Lex V2 は、音声入力をデフォルトの方法で解釈してスロットに対応します。

  SpellByLetter — Amazon Lex V2 は、ユーザーによるスロット値のスペルをリッスンします。

  SpellByWord — Amazon Lex V2 は、各文字に関連する単語 (「a as in apple」など) を使用して、ユーザーによるスロット値のスペルをリッスンします。
- slotToElicit — dialogAction の type が ElicitSlot の場合に、ユーザーから誘発するスロットを定義します。
- type — ボットが実行すべきアクションを定義します。以下の値を指定できます。

  Delegate - Amazon Lex V2 が次のアクションを決定できるようにします。

  ElicitIntent — 顧客にインテントを表現するよう促します。

`ConfirmIntent` - 顧客のスロット値と、そのインテントフルフィルメントの準備ができているかどうかを確認します。

`ElicitSlot` — インテントのスロット値を入力するよう顧客に促します。

`Close` — インテントのフルフィルメントプロセスを終了します。

intent

`intent` フィールドの構造については、「Intent」を参照してください。

originatingRequestId

リクエストの一意の識別子。Lambda レスポンスの場合、このフィールドはオプションです。

Amazon Lex V2 ボットの AWS Lambda関数の作成

Amazon Lex V2 ボットの Lambda 関数を作成するには、AWS マネジメントコンソールAWS Lambdaから にアクセスし、新しい関数を作成します。詳細については、 デAWS Lambdaベロッパーガイドを参照してくださいAWS Lambda。

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/lambda/ で AWS Lambdaコンソールを開きます。
2. 左側のサイドバーで [関数] を選択します。
3. [Create Function] (関数を作成) を選択します。
4. [一から作成] を選択して最小限のコードから始めるか、[ブループリントの使用] を選択してリストから一般的なユースケースのサンプルコードを選択するか、[コンテナイメージ] を選択して関数にデプロイするコンテナイメージを選択できます。[一から選択] を選択した場合は、次の手順に進んでください。

   a. 関数には、その機能を説明するわかりやすい関数名を付けてください。

   b. [ランタイム] の下のドロップダウンメニューから、関数を記述する言語を選択します。

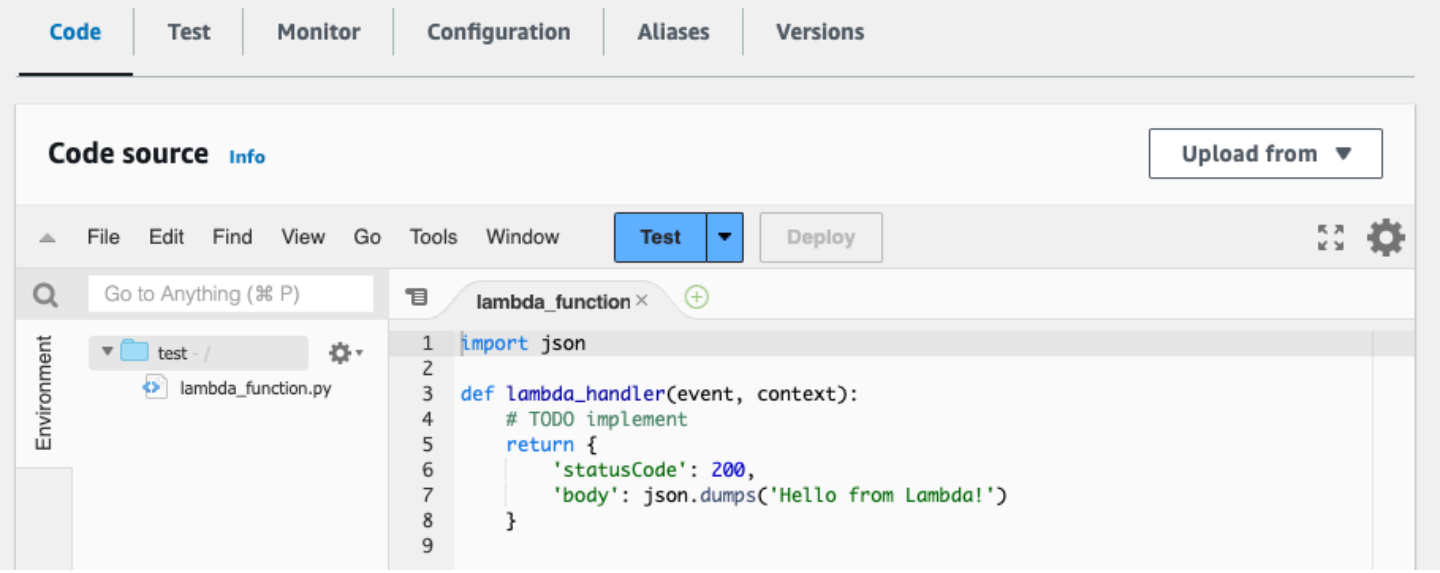
   c. 関数の命令セット [アーキテクチャ] を選択します。

   d. デフォルトでは、Lambda は基本的なアクセス許可でロールを作成します。既存のロールを使用するか、AWS ポリシーテンプレートを使用してロールを作成するには、デフォルトの実行ロールの変更メニューを展開し、オプションを選択します。

e. [詳細設定] メニューを展開して、その他のオプションを設定します。

5. [Create Function] (関数を作成) を選択します。

次の図は、新しい関数を最初から作成したときに表示される内容を示しています。



Lambda ハンドラー関数は、使用する言語によって異なります。少なくとも因数として `event` JSON オブジェクトを取ります。フィールドは、Amazon Lex V2 が AWS LambdaLex V2 の入力イベント形式 で提供する `event` にあります。「AWS LambdaLex V2 のレスポンス形式」で説明されている形式と一致する `response` JSON オブジェクトが最終的に返されるように、ハンドラー関数を変更します。

- 関数を記述し終えたら、[デプロイ] を選択して関数を使用できるようにします。

各ボットエイリアスは、最大で 1 つの Lambda 関数に関連付けられることに注意してください。ただし、Lambda コード内でボットに必要な数だけ関数を定義し、これらの関数を Lambda ハンドラー関数で呼び出すことができます。たとえば、同じボットエイリアスのすべてのインテントは同じ Lambda 関数を呼び出す必要がありますが、インテントごとに個別の関数を有効にするルーター関数を作成できます。次に、アプリケーションで使用または変更できるサンプルルーター関数を示します。

```
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
```

Amazon Lex V2 ボット会話でAWS Lambda関数を使用するタイミング

Lambda 関数はユーザーとの会話の次の時点で使用できます。

- インテントが認識された後の最初の応答。たとえば、ユーザーがピザを注文したいと言った後です。
- ユーザーからスロット値を誘発した後。たとえば、ユーザーが注文したいピザのサイズをボットに伝えた後などです。
- スロットを誘発するための各再試行の間。たとえば、認識しているサイズのピザを顧客が使用していない場合などです。
- インテントを確認する時点。たとえば、ピザの注文を確認する場合などです。
- インテントを達成するため。たとえば、ピザを注文する場合などです。
- インテントの達成後、ボットが会話を終了する前。たとえば、飲み物を注文するインテントに切り替える場合などです。

トピック

- コンソールを使用して Amazon Lex V2 ボットに AWS Lambda関数をアタッチする
- API オペレーションを使用して Amazon Lex V2 ボットに AWS Lambda関数をアタッチする

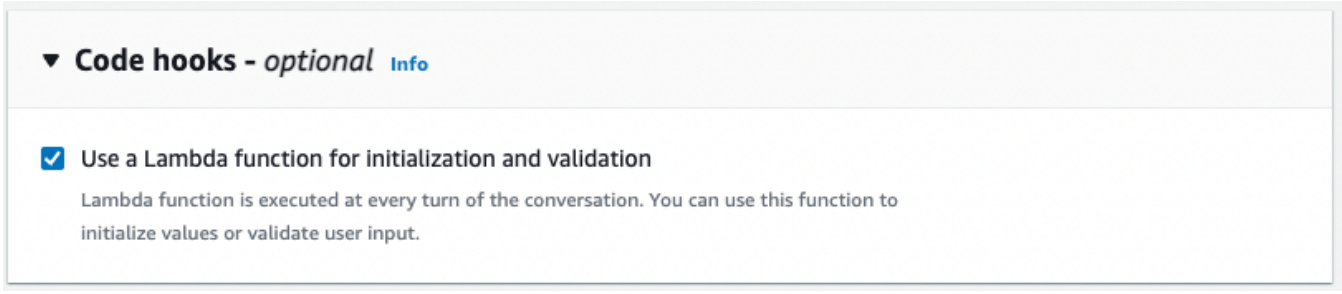
コンソールを使用して Amazon Lex V2 ボットに AWS Lambda関数をアタッチする

呼び出す前に、まず Lambda 関数を Amazon Lex V2 ボットエイリアスにアタッチする必要があります。1 つのボットエイリアスにアタッチできる Lambda 関数は 1 つだけです。以下の手順を実行して、AWS コンソールを使用して Lambda 関数をアタッチします。

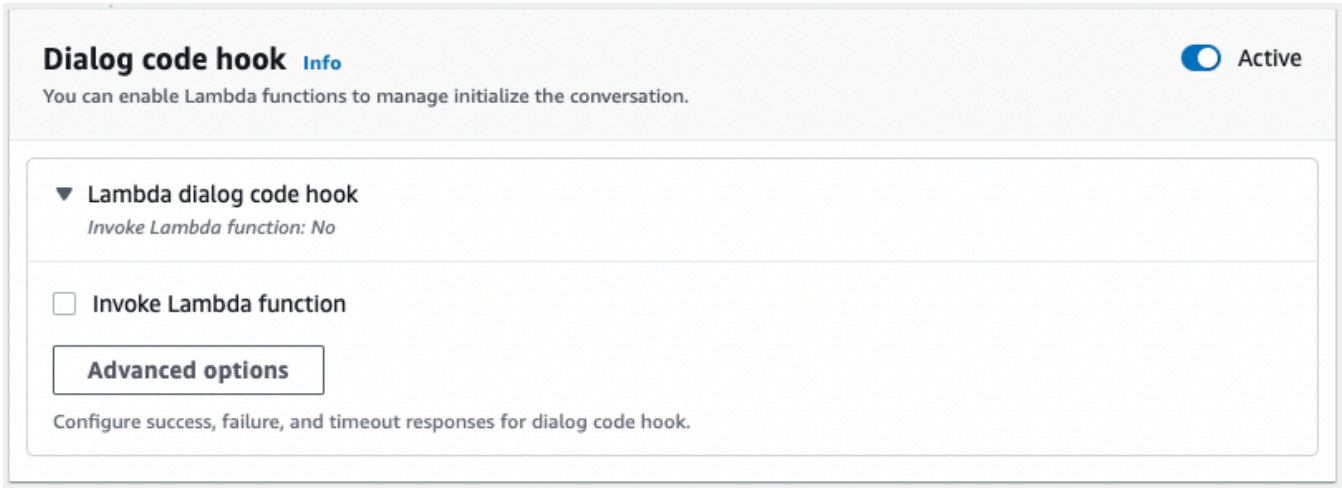
1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/lex/ で Amazon Lex コンソールを開きます。
2. 左側のパネルで [ボット] を選択し、ボットのリストから Lambda 関数を割り当てるボットの名前を選択します。
3. 左側のパネルから、[デプロイ] メニューの [エイリアス] を選択します。
4. エイリアスのリストから、Lambda 関数をアタッチするエイリアスの名前を選択します。
5. [言語] パネルで、Lambda 関数に使用する言語を選択します。言語がパネルに表示されていない場合は、[エイリアスの言語を管理] を選択して言語を追加します。
6. [ソース] ドロップダウンメニューで、アタッチする Lambda 関数の名前を選択します。
7. [Lambda 関数のバージョンまたはエイリアス] ドロップダウンメニューで、使用する Lambda 関数のバージョンまたはエイリアスを選択します。次に、[保存] を選択します。ボットがサポートする言語のすべてのインテントに同じ Lambda 関数が使用されます。

コンソールを使用して Lambda 関数を呼び出すための Amazon Lex V2 インテントの設定

1. ボットを選択したら、左側のメニューで Lambda 関数を呼び出すボットの言語の下にある [インテント] を選択します。
2. Lambda 関数を呼び出しインテントを選択し、インテントエディターを開きます。
3. Lambda コードフックを設定するためには 2 つのオプションがあります。
    1. 会話の各ステップ後に Lambda 関数を呼び出すには、以下の画像のように、インテントエディターの下部にある [コードフック] セクションまでスクロールし、[初期化と検証に Lambda 関数を使用する] チェックボックスを選択します。



2. または、会話ステージの [ダイアログコードフック] セクションを使用して Lambda 関数を呼び出します。[ダイアログコードフック] セクションは次のように表示されます。



Amazon Lex V2 がレスポンスのコードフックを呼び出す方法を制御する方法は 2 つあります。

- [アクティブ] ボタンを切り替えて、アクティブまたは非アクティブとマークします。コードフックがアクティブになると、Amazon Lex V2 はコードフックを呼び出します。コードフックが非アクティブの場合、Amazon Lex V2 はコードフックを実行しません。
- [Lambda ダイアログのコードフック] セクションを展開し、[Lambda 関数を呼び出す] チェックボックスを選択して、有効または無効のマークを付けます。コードフックは、アクティブとマークされている場合にのみ有効化または無効化できます。有効とマークされると、コードフックは通常どおり実行されます。無効にすると、コードフックは呼び出されず、Amazon Lex V2 はコードフックが正常に返されたかのように動作します。ダイアログコードフックが成功、失敗、またはタイムアウトした後のレスポンスを設定するには、[詳細オプション] を選択します。

Lambda コードフックは、以下の会話段階で呼び出すことができます。

- この関数を初期レスポンスとして呼び出すには、[初期応答] セクションまでスクロールし、[ユーザーの要求を確認するための応答] の横にある矢印を展開し、[詳細オプション]を

選択します。ポップアップメニューの下部にある [ダイアログコードフック] セクションを探してください。

- スロット誘発後に関数を呼び出すには、[スロット] セクションまでスクロールし、該当する [スロットのプロンプト] の横にある矢印を展開して、[詳細オプション] を選択します。ポップアップメニューの下部、[デフォルト値] のすぐ上にある [ダイアログコードフック] セクションを見つけてください。

  また、誘発のたびに関数を呼び出すこともできます。これを行うには、[スロットプロンプト] セクションの [ボット誘発情報] を展開し、[その他のプロンプトオプション] を選択し、[誘発のたびに Lambda コードフックを呼び出す] の横にあるチェックボックスを選択します。

- インテント確認用の関数を呼び出すには、[確認] セクションまでスクロールし、[インテントを確認するプロンプトを表示] の横にある矢印を展開してインテントを確認し、[詳細オプション] を選択します。ポップアップメニューの下部にある [ダイアログコードフック] セクションを探してください。

- インテントフルフィルメント用の関数を呼び出すには、[フルフィルメント] セクションまでスクロールします。[アクティブ] ボタンを切り替えて、コードフックをアクティブに設定します。[フルフィルメントが成功した場合] の横にある矢印を展開し、[詳細オプション] を選択します。[フルフィルメント Lambda コードフック] セクションの下にある [フルフィルメントに Lambda 関数を使用] の横にあるチェックボックスを選択して、コードフックを有効に設定します。

4. Lambda 関数を呼び出す会話ステージを設定したら、ボットを再度 [構築] して関数をテストします。

API オペレーションを使用して Amazon Lex V2 ボットに AWS Lambda関数をアタッチする

呼び出す前に、まず Lambda 関数を Amazon Lex V2 ボットエイリアスにアタッチする必要があります。1 つのボットエイリアスに関連付けることができる Lambda 関数 は 1 つだけです。以下の手順を実行して、API オペレーションを使用して Lambda 関数をアタッチします。

新しいボットエイリアスを作成する場合は、CreateBotAlias オペレーションを使用して Lambda 関数をアタッチします。Lambda 関数を既存のボットエイリアスにアタッチするには、UpdateBotAlias オペレーションを使用します。正しい設定を含むように `botAliasLocaleSettings` フィールドを変更します。

```
{
    "botAliasLocaleSettings" : {
        locale: {
            "codeHookSpecification": {
                "lambdaCodeHook": {
                    "codeHookInterfaceVersion": "1.0",
                    "lambdaARN": "arn:aws:lambda:region:account-id:function:function-
name"
                }
            },
            "enabled": true
        },
        ...
    }
}
```

1. botAliasLocaleSettings フィールドは、Lambda 関数をアタッチするロケールをキーとするオブジェクトにマップされます。サポートされているロケールと有効なキーとなるコードのリストについては、「サポートされている言語とロケール」を参照してください。
2. Lambda 関数lambdaARNの を検索するには、https://console.aws.amazon.com/lambda/home で AWS Lambdaコンソールを開き、左側のサイドバーで関数を選択し、ボットエイリアスに関連付ける関数を選択します。[関数概要] の右側にある [関数 ARN] の下の lambdaARN を確認します。リージョン、アカウント ID、関数名が含まれている必要があります。
3. Amazon Lex V2 がエイリアスの Lambda 関数を呼び出せるようにするには、enabled フィールドを true に設定します。

API オペレーションを使用して Lambda 関数を呼び出すための Amazon Lex V2 インテントの設定

インテント中に Lambda 関数の呼び出しを設定するには、新しいインテントを作成する場合は CreateIntent オペレーションを使用し、既存のインテントで関数を呼び出す場合は UpdateIntent オペレーションを使用します。インテントオペレーションでの Lambda 関数の呼び出しを制御するフィールドは、dialogCodeHook、initialResponseSetting、intentConfirmationSetting、および fulfillmentCodeHook です。

スロットの誘発中に関数を呼び出す場合、新しいスロットを作成する場合は CreateSlot オペレーションを使用し、既存のスロットで関数を呼び出すには UpdateSlot オペレーションを使用します。スロットオペレーションでの Lambda 関数の呼び出しを制御するフィールドは valueElicitationSetting オブジェクトの slotCaptureSetting フィールドです。

1. 会話が終わるたびに Lambda ダイアログコードフックが実行されるように設定するには、enabled フィールド内の次の DialogCodeHookSettings オブジェクトの dialogCodeHook フィールドを true に設定します。

```
"dialogCodeHook": {
    "enabled": boolean
}
```

2. または、関数を呼び出す会話ステージに対応する構造内の codeHook および/または elicitationCodeHook フィールドを変更して、会話の特定の時点でのみ実行されるように Lambda ダイアログコードフックを設定することもできます。Lambda ダイアログのコードフックをインテントフルフィルメントに使用するには、CreateIntent オペレーションまたは UpdateIntent オペレーションの fulfillmentCodeHook フィールドを使用します。これら 3 種類のコードフックの構造と用途は次のとおりです。

codeHook

codeHook フィールドは、会話の特定の段階で実行されるコードフックの設定を定義します。このオブジェクトは DialogCodeHookInvocationSetting オブジェクトで、次のような構造になっています。

```
"codeHook": {
    "active": boolean,
    "enableCodeHookInvocation": boolean,
    "invocationLabel": string,
    "postCodeHookSpecification": PostDialogCodeHookInvocationSpecification object,
}
```

- Amazon Lex V2 の active フィールドを trueに変更して、会話のその時点でコードフックを呼び出します。
- Amazon Lex V2 の enableCodeHookInvocation フィールドを true に変更して、コードフックが正常に動作するようにします。false をマークすると、Amazon Lex V2 はコードフックが正常に返されたかのように動作します。
- invocationLabel は、コードフックが呼び出されるダイアログステップを示します。
- postCodeHookSpecification フィールドを使用して、コードフックの成功、失敗、またはタイムアウトの後で発生するアクションとメッセージを指定します。

elicitationCodeHook

elicitationCodeHook フィールドは、1 つまたは複数のスロットを再利用する必要がある場合に実行するコードフックの設定を定義します。このシナリオは、スロットの誘発に失敗したり、インテントの確認が拒否されたりした場合に発生する可能性があります。elicitationCodeHook フィールドは、次の構造を持つ ElicitationCodeHookInvocationSetting オブジェクトです。

```
"elicitationCodeHook": {
    "enableCodeHookInvocation": boolean,
    "invocationLabel": string
}
```

- Amazon Lex V2 の enableCodeHookInvocation フィールドを true に変更して、コードフックが正常に動作するようにします。false をマークすると、Amazon Lex V2 はコードフックが正常に返されたかのように動作します。
- invocationLabel は、コードフックが呼び出されるダイアログステップを示します。

fulfillmentCodeHook

fulfillmentCodeHook フィールドは、インテントを達成するために実行するコードフックの設定を定義します。このフィールドは次の FulfillmentCodeHookSettings オブジェクトにマップされます。

```
"fulfillmentCodeHook": {
    "active": boolean,
    "enabled": boolean,
    "fulfillmentUpdatesSpecification": FulfillmentUpdatesSpecification object,
    "postFulfillmentStatusSpecification": PostFulfillmentStatusSpecification object
}
```

- Amazon Lex V2 の active フィールドを trueに変更して、会話のその時点でコードフックを呼び出します。
- Amazon Lex V2 の enabled フィールドを true に変更して、コードフックが正常に動作するようにします。false をマークすると、Amazon Lex V2 はコードフックが正常に返されたかのように動作します。
- fulfillmentUpdatesSpecification フィールドを使用して、インテントのフルフィルメント中にユーザーに最新情報が表示されるメッセージと、それに関連するタイミングを指定します。

- postFulfillmentStatusSpecification フィールドを使用して、コードフックの成功、失敗、またはタイムアウトの後で発生するメッセージとアクションを指定します。

active および enableCodeHookInvocation/enabled フィールドを true に設定することで、会話の次の時点で Lambda コードフックを呼び出すことができます。

初期応答中

インテントが認識された後の初期応答で Lambda 関数を呼び出すには、CreateIntent または UpdateIntent オペレーションの initialResponse フィールドにある codeHook 構造を使用します。initialResponse フィールドは次の InitialResponseSetting オブジェクトにマップされます。

```
"initialResponse": {
    "codeHook": {
        "active": boolean,
        "enableCodeHookInvocation": boolean,
        "invocationLabel": string,
        "postCodeHookSpecification": PostDialogCodeHookInvocationSpecification object,
    },
    "initialResponse": FulfillmentUpdatesSpecification object,
    "nextStep": PostFulfillmentStatusSpecification object,
    "conditional": ConditionalSpecification object
}
```

スロット誘発後、またはスロット再誘発中

スロット値を取得した後に Lambda 関数を呼び出すには、CreateSlot または UpdateSlot オペレーションの valueElicitation フィールド内の slotCaptureSetting フィールドを使用します。slotCaptureSetting フィールドは次の SlotCaptureSetting オブジェクトにマップされます。

```
"slotCaptureSetting": {
    "captureConditional": ConditionalSpecification object,
    "captureNextStep": DialogState object,
    "captureResponse": ResponseSpecification object,
    "codeHook": {
        "active": true,
        "enableCodeHookInvocation": true,
        "invocationLabel": string,
        "postCodeHookSpecification": PostDialogCodeHookInvocationSpecification object,
    },
```

```
    "elicitationCodeHook": {
        "enableCodeHookInvocation": boolean,
        "invocationLabel": string
    },
    "failureConditional": ConditionalSpecification object,
    "failureNextStep": DialogState object,
    "failureResponse": ResponseSpecification object
}
```

- スロットの誘発が成功した後に Lambda 関数を呼び出すには、codeHook フィールドを使用します。
- スロットの引き出が失敗し、Amazon Lex V2 がスロットの誘発を再試行した後に Lambda 関数を呼び出すには、elicitationCodeHook フィールドを使用します。

インテントの確認または拒否後

インテントを確認するときに Lambda 関数を呼び出すには、CreateIntent オペレーションまたは UpdateIntent オペレーションの intentConfirmationSetting フィールドを使用します。intentConfirmation フィールドは次の IntentConfirmationSetting オブジェクトにマップされます。

```
"intentConfirmationSetting": {
    "active": boolean,
    "codeHook": {
        "active": boolean,
        "enableCodeHookInvocation": boolean,
        "invocationLabel": string,
        "postCodeHookSpecification": PostDialogCodeHookInvocationSpecification object,
    },
    "confirmationConditional": ConditionalSpecification object,
    "confirmationNextStep": DialogState object,
    "confirmationResponse": ResponseSpecification object,
    "declinationConditional": ConditionalSpecification object,
    "declinationNextStep": FulfillmentUpdatesSpecification object,
    "declinationResponse": PostFulfillmentStatusSpecification object,
    "elicitationCodeHook": {
        "enableCodeHookInvocation": boolean,
        "invocationLabel": string,
    },
    "failureConditional": ConditionalSpecification object,
    "failureNextStep": DialogState object,
```

```
    "failureResponse": ResponseSpecification object,
    "promptSpecification": PromptSpecification object
}
```

- ユーザーがインテントとそのスロットを確認した後に Lambda 関数を呼び出すには、codeHook フィールドを使用します。
- ユーザーがインテント確認を拒否し、Amazon Lex V2 がスロット誘発を再試行した後に Lambda 関数を呼び出すには、elicitationCodeHook フィールドを使用します。

インテントフルフィルメント中

インテントを確認するために Lambda 関数を呼び出すには、CreateIntent オペレーションまたは UpdateIntent オペレーションの fulfillmentCodeHook フィールドを使用します。fulfillmentCodeHook フィールドは次の FulfillmentCodeHookSettings オブジェクトにマップされます。

```
{
    "active": boolean,
    "enabled": boolean,
    "fulfillmentUpdatesSpecification": FulfillmentUpdatesSpecification object,
    "postFulfillmentStatusSpecification": PostFulfillmentStatusSpecification object
}
```

3. Lambda 関数を呼び出す会話ステージを設定したら、BuildBotLocale オペレーションを使用して関数をテストするためにボットを再構築します。

CloudWatch Logs のログを使用した Lambda 関数のデバッグ

Amazon CloudWatch Logs は、Lambda 関数のデバッグに役立つ API 呼び出しとメトリクスを追跡するためのツールです。コンソールまたは API 呼び出しでボットをテストすると、CloudWatch は会話の各ステップを記録します。Lambda コードで印刷関数を使用すると、CloudWatch はその関数も表示します。

Lambda 関数の CloudWatch Logs を表示するには

1. AWS マネジメントコンソールにサインインし、CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. 左側のバーの [ログ] メニューで [ロググループ] を選択します。

3. Lambda 関数のロググループを選択します。このグループは /aws/lambda/*function-name* 形式である必要があります。
4. ログストリームのリストには、ボットの各セッションのログが含まれています。表示するログストリームを選択します。
5. [ログイベント] のリストで、[タイムスタンプ] の横にある右矢印を選択すると、そのイベントの詳細が展開されます。Lambda コードから印刷したものはすべてログイベントとして表示されます。この情報を使用してコードをデバッグします。
6. コードをデバッグしたら、必ず Lambda 関数をデプロイし、コンソールを使用している場合は、ボットの動作を再テストする前にテストウィンドウをリロードしてください。

Lex V2 でのユーザーとボットのインタラクションのカスタマイズ

ユーザーのデフォルト動作を拡張したり調整したりすることで、ボットによるユーザーとのやりとりをカスタマイズできる以下の機能について説明します。

トピック

- ボットとの会話におけるユーザー発話の感情分析
- 信頼度スコアを使用した会話精度の向上
- Lex V2 ボットで使用する音声文字起こしのカスタマイズ
- オーディオフィラーを使用してボットの応答性を向上させる

ボットとの会話におけるユーザー発話の感情分析

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。

ボットの感情分析を有効にするには、コンソールを使用するか、Amazon Lex API を使用します。ボットのエイリアスで感情分析を有効にします。Amazon Lex コンソールを使用します。

1. エイリアスを選択します。
2. [詳細] で [編集] を選択します。
3. [感情分析を有効にする] を選択し、感情分析をオンまたはオフにします。
4. [確定] を選択し、変更を保存します。

API を使用している場合は、`detectSentiment` フィールドを `true` に設定して CreateBotAlias オペレーションを呼び出します。

感情分析を有効にすると、RecognizeText と RecognizeUtterance オペレーション操作の応答は、他のメタデータと一緒に interpretations の構造体の sentimentResponse というフィールドを返します。sentimentResponse フィールドには、センチメント分析の結果を含む sentiment 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、RecognizeText または RecognizeUtterance 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。

```
sentimentResponse {
    "sentimentScore": {
        "mixed": 0.030585512690246105,
        "positive": 0.94992071056365967,
        "neutral": 0.0141543131828308,
        "negative": 0.00893945890665054
    },
    "sentiment": "POSITIVE"
}
```

Amazon Lex は、あなたに代わって Amazon Comprehend を呼び出し、ボットが処理するすべての発話に含まれる感情を判断します。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 の感情分析の仕組みについては、「Amazon Comprehendデベロッパーガイド」の「感情を判断する」を参照してください。

信頼度スコアを使用した会話精度の向上

Amazon Lex V2 では、ユーザーが言っていることを判断するために 2 つのステップを使用します。1 つ目は自動音声認識 (ASR) で、ユーザーの音声発話の文字起こしを作成します。もう 1 つは自然言語理解 (NLU) で、ユーザーの発話の意味を判断して、ユーザーのインテントやスロットの価値を認識します。

デフォルトで、Amazon Lex V2 は ASR と NLU からの最も可能性の高い結果を返します。場合によっては、Amazon Lex V2 が最も可能性の高い結果を判断するのが困難なことがあります。その場合、結果が正しい可能性を示す信頼度スコアとともに、考えられる複数の結果が返されます。信頼度スコアは、Amazon Lex V2 が提供する評価で、結果における相対的確信度を示します。信頼度スコアは 0.0 ~ 1.0 です。

専門分野の知識と信頼スコアを併用することで、ASR または NLU の結果の正しい解釈を判断できます。

ASR (文字起こし) 信頼度スコアは、Amazon Lex V2 が特定の文字起こしが正しいとどの程度確信しているかを示す評価です。NLU (インテント) 信頼度スコアは、上部の文字起こしで指定されているインテントが正しいことを Amazon Lex V2 がどの程度確信しているかを示す評価です。アプリケーションに最も適した信頼度スコアを使用してください。

トピック

- Lex V2 でのインテント信頼スコアを使用したインテント選択の改善
- 音声文字起こしの信頼度スコアを使用した Lex V2 ボットとの会話の改善

Lex V2 でのインテント信頼スコアを使用したインテント選択の改善

ユーザーが発話を行うと、Amazon Lex V2 は自然言語理解 (NLU) を使用してユーザーの要求を理解し、適切な意図を返します。デフォルトでは、Amazon Lex V2 は、ボットによって定義された最も可能性の高い意図を返します。

場合によっては、Amazon Lex V2 が最も可能性の高い意図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、似たような意図が2つある場合などです。適切な意図を決定するために、解釈のリストで、ドメイン知識と NLU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 V2 が提供する評価で、意図が正しい意図であるという確信度を示します。

解釈内の 2 つの意図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、最初の意図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 V2 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 V2 の改善に基づいて変更される場合があります。

Amazon Lex V2 は、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、各レスポンスの `interpretations` 構造で関連するスコアとともに返します。次の JSON コード

は、RecognizeText オペレーションからのレスポンスにおける interpretations の構造を示しています。

```
"interpretations": [
   {
      "intent": {
         "confirmationState": "string",
         "name": "string",
         "slots": {
            "string" : {
               "value": {
                  "interpretedValue": "string",
                  "originalValue": "string",
                  "resolvedValues": [ "string" ]
               }
            }
         },
         "state": "string"
      },
      "nluConfidence": number
   }
]
```

AMAZON.FallbackIntent

Amazon Lex V2 は、次の 2 つの状況で最上位インテントとして AMAZON.FallbackIntent を返します。

1. 考えられるすべてのインテントの信頼スコアが信頼度しきい値を下回っている場合。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。AMAZON.KendraSearchIntent が設定されている場合、この状況で Amazon Lex V2 は同様にそれを返します。
2. AMAZON.FallbackIntent の解釈の信頼度が、他のすべてのインテントの解釈の信頼度よりも高い場合。

Amazon Lex V2 は、AMAZON.FallbackIntent の信頼スコアを表示しないことに注意してください。

信頼度しきい値の設定と変更

信頼度しきい値は 0.00 から 1.00 の間の数値である必要があります。以下の方法で、ボットの言語ごとにしきい値を設定できます。

Amazon Lex V2 コンソールの使用

- [言語を追加] を使用してボットに言語を追加するときにしきい値を設定するには、[信頼度スコアのしきい値] パネルに希望する値を挿入します。
- しきい値を更新するには、ボットの言語の [言語詳細] パネルで [編集] を選択します。次に、[信頼度スコアしきい値] パネルに希望する値を挿入します。

API オペレーションの使用

- しきい値を設定するには、CreateBotLocale オペレーションの `nluIntentConfidenceThreshold` パラメータを設定します。
- 信頼度しきい値を更新するには、UpdateBotLocale オペレーションの `nluIntentConfidenceThreshold` パラメータを設定します。

セッションの管理

Amazon Lex V2 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。

Lex V2 ボットでの Lambda 関数の使用

Lambda 関数を使用する場合、Amazon Lex V2 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 V2 がユーザーの発話の最も可能性の高い意図として特定した意図を含む、`currentIntent` と呼ばれるフィールドを含んでいます。JSON 構造は、ユーザーの意図を満たす可能性のある最大 4 つの追加意図を含む `alternativeIntents` フィールドも含んでいます。各意図には、Amazon Lex V2 がその意図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。

別の意図を使用するには、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、「AWS Lambda関数を Amazon Lex V2 ボットに統合する」を参照してください。

Lex V2 ボットでのセッション管理 API の使用

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 V2 が選択した意図よりも、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意図を変更し、ユーザーが次に対話する意図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 V2 ボットセッションについて」を参照してください。

音声文字起こしの信頼度スコアを使用した Lex V2 ボットとの会話の改善

ユーザーが音声発話を行うと、Amazon Lex V2 は自動音声認識 (ASR) を使用して、解釈される前にユーザーの要求を書き起こします。Amazon Lex V2 はデフォルトで、最も可能性の高い音声文字起こしを使用して解釈します。

場合によっては、音声文字起こしが複数あることもあります。たとえば、ユーザーが「マイ・ネーム・イズ・ジョン」が「マイ・ネーム・イズ・フアン」と聞こえるような曖昧な発話をする場合があります。このような場合は、曖昧性解消の手法を使用するか、ドメインの知識と文字起こし信頼度スコアを組み合わせて、文字起こしリスト内のどの文字起こしが正しいかを判断できます。

Amazon Lex V2 には、Lambda コードフック関数へのリクエストのユーザー入力用に、一番上の文字起こしと最大 2 つの代替文字起こしが含まれています。各文字起こしには、それが正しい文字起こしであることを示す信頼度スコアが含まれています。各文字起こしには、ユーザー入力から推測されるスロット値も含まれます。

2 つの文字起こしの信頼スコアを比較して、両者の間にあいまいさがあるかどうかを判断できます。例えば、ある文字起こしの信頼スコアが 0.95 で、別の文字起こしの信頼スコアが 0.65 の場合、最初の文字起こしはおそらく正しく、それらの間のあいまいさは低いでしょう。2 つの文字起こしの信頼スコアが 0.75 と 0.72 であれば、両者のあいまいさは高いと言えます。ドメインの知識があれば区別できる可能性はあります。

たとえば、信頼スコアが 0.75 と 0.72 の 2 つの文字起こしの推定スロット値が「ジョン」と「ホアン」の場合、データベース内のユーザーにこれらの名前の存在を問い合わせて、文字起こしの 1 つを削除できます。「ジョン」がデータベース内のユーザーではなく、「ホアン」がユーザーである場合は、ダイアログコードフックを使用して、ファーストネームの推定スロット値を「ホアン」に変更できます。

Amazon Lex V2 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 V2 の改善に基づいて変更される場合があります。

音声文字起こしの信頼度スコアは、英語 (GB) (en_GB) と英語 (米国) (en_US) の言語でのみ利用できます。信頼度スコアは 8 kHz 音声入力でのみサポートされます。Amazon Lex V2 コンソールのテストウィンドウからの音声入力では 16 kHz の音声入力が使用されるため、文字起こし信頼性スコアは提供されません。

> **Note**
>
> 既存のボットで音声文字変換信頼度スコアを使用するには、まずボットを再構築する必要があります。既存のバージョンのボットは、文字起こし信頼度スコアをサポートしていません。それらを使用するには、ボットの新しいバージョンを作成する必要があります。

信頼度スコアは複数の会話デザインパターンに使用できます。

- 騒がしい環境や信号品質が悪いために最も高い信頼スコアがしきい値を下回った場合は、ユーザーに同じ質問をしてより高品質の音声をキャプチャするように促すことができます。
- 「ジョン」と「ホアン」のように、複数の文字起こしのスロット値の信頼スコアが類似している場合は、値を既存のデータベースと比較して入力を省くか、2 つの値のうちの 1 つを選択するようユーザーに促すことができます。たとえば、「ジョンなら 1、ホアンなら 2 と言ってください。」と設定します。
- ビジネスロジックで、信頼スコアが最上位の文字起こしに近い代替文字起こしの特定のキーワードに基づいてインテントを切り替える必要がある場合は、ダイアログコードフックの Lambda 関数を使用するか、セッション管理オペレーションを使用してインテントを変更できます。詳細については、「セッション管理」を参照してください。

Amazon Lex V2 は、Lambda コードフック関数へのユーザーの入力について、最大 3 つの文字起こしを含む次の JSON 構造を送信します。

```
"transcriptions": [
    {
        "transcription": "string",
        "rawTranscription": "string",
        "transcriptionConfidence": "number",
        },
        "resolvedContext": {
            "intent": "string"
        },
```

```
            "resolvedSlots": {
                "string": {
                    "shape": "List",
                    "value": {
                        "originalValue": "string",
                        "resolvedValues": [
                            "string"
                        ]
                    },
                    "values": [
                        {
                            "shape": "Scalar",
                            "value": {
                                "originalValue": "string",
                                "resolvedValues": [
                                    "string"
                                ]
                            }
                        },
                        {
                            "shape": "Scalar",
                            "value": {
                                "originalValue": "string",
                                "resolvedValues": [
                                    "string"
                                ]
                            }
                        }
                    ]
                }
            }
        }
    ]
```

JSON 構造には、文字起こしテキスト、発話で解決されたインテント、および発話で検出されたスロットの値が含まれます。テキストユーザー入力の場合、文字起こしには信頼スコアが 1.0 の文字起こしが 1 つ含まれます。

文字起こしの内容は、会話の順番や認識されたインテントによって異なります。

最初のターン、インテント誘発では、Amazon Lex V2 が文字起こしの上位 3 つを決定します。一番上の文字起こしでは、インテント、および文字起こし内の推定スロット値が返されます。

次のターン、つまりスロット誘発では、結果は各文字起こしの推定インテントによって次のように異なります。

- 一番上の文字起こしの推定インテントが前のターンと同じで、他のすべての文字起こしが同じインテントである場合、
  - すべての文字起こしには推定されたスロット値が含まれます。

- 一番上の文字起こしの推定インテントが前のターンと異なり、他のすべての文字起こしに前のインテントがある場合、
  - 一番上の文字起こしには、新しいインテントの推定スロット値が含まれます。
  - 他の文字起こしには、前のインテントとそのインテントの推定スロット値が含まれています。

- 一番上の文字起こしの推測されたインテントが前のターンと異なり、1 つの文字起こしが前のインテントと同じで、1 つの文字起こしが別のインテントである場合、
  - 一番上の文字起こしには、発話に含まれる新しい推測されたインテントと推測されたスロット値が含まれます。
  - 以前に推測されたインテントを含む文字起こしには、そのインテントの推測されたスロット値が含まれています。
  - インテントが異なる文字起こしには、推測されたインテント名も推測されたスロット値もありません。

- 一番上の文字起こしの推測されたインテントが前のターンと異なり、他のすべての文字起こしのインテントが異なる場合、
  - 一番上の文字起こしには、発話に含まれる新しい推測されたインテントと推測されたスロット値が含まれます。
  - その他の文字起こしには、推測されたインテントや推測されたスロット値が含まれていません。

- 上位 2 つの文字起こしの推測されたインテントが前のターンと同じで異なっていて、3 番目の文字起こしのインテントが異なる場合、
  - 上位 2 つの文字起こしには、発話に含まれる新しい推測されたインテントと推測されたスロット値が含まれます。

- 3 番目の文字起こしにはインテント名も解決済みのスロット値もありません。

セッション管理

Amazon Lex V2 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します。または、カスタムアプリケーションでセッション管理 API を使用することもできます。

Lex V2 ボットでの Lambda 関数の使用

Lambda 関数を使用する場合、Amazon Lex V2 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 V2 が発話として特定した可能な文字変換を含む `transcriptions` と呼ばれるフィールドを含んでいます。`transcriptions` フィールドには、1～3 つの可能な文字起こしが含まれており、それぞれに信頼スコアが付けられています。

別の文字起こしから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。代替文字起こしのスロット値を使用するには、Lambda 関数レスポンスの `intent` フィールドに値を設定します。詳細については、「AWS Lambda関数を Amazon Lex V2 ボットに統合する」を参照してください。

Lex V2 で Lambda を使用する場合のコードの例

次のコード例は Python Lambda 関数で、音声文字変換を使用してユーザーの会話エクスペリエンスを向上させます。

サンプルコードを使用するには、以下が必要です。

- 英語 (GB) (en_GB) または英語 (GB) (en_US) のうち 1 つの言語を使ったボット。
- 1 つのインテント、OrderBirthStone。インテント定義の [コードフック] セクションで [初期化と検証に Lambda 関数を使用する] が選択されていることを確認してください。
- インテントには「BirthMonth」と「Name」の 2 つのスロットが必要であり、どちらも `AMAZON.AlphaNumeric` タイプとなります。
- Lambda 関数が定義されたエイリアス。詳細については、「Amazon Lex V2 ボットの AWS Lambda関数の作成」を参照してください。

```
import time
import os
import logging
```

```
logger = logging.getLogger()
logger.setLevel(logging.DEBUG)


# --- Helpers that build all of the responses ---

def elicit_slot(session_attributes, intent_request, slots, slot_to_elicit, message):
    return {
        'sessionState': {
            'dialogAction': {
                'type': 'ElicitSlot',
                'slotToElicit': slot_to_elicit
            },
            'intent': {
                'name': intent_request['sessionState']['intent']['name'],
                'slots': slots,
                'state': 'InProgress'
            },
            'sessionAttributes': session_attributes,
            'originatingRequestId': 'e3ab4d42-fb5f-4cc3-bb78-caaf6fc7cccd'
        },
        'sessionId': intent_request['sessionId'],
        'messages': [message],
        'requestAttributes': intent_request['requestAttributes'] if 'requestAttributes'
 in intent_request else None
    }


def close(intent_request, session_attributes, fulfillment_state, message):
    intent_request['sessionState']['intent']['state'] = fulfillment_state
    return {
        'sessionState': {
            'sessionAttributes': session_attributes,
            'dialogAction': {
                'type': 'Close'
            },
            'intent': intent_request['sessionState']['intent'],
            'originatingRequestId': '3ab4d42-fb5f-4cc3-bb78-caaf6fc7cccd'
        },
        'messages': [message],
        'sessionId': intent_request['sessionId'],
        'requestAttributes': intent_request['requestAttributes'] if 'requestAttributes'
 in intent_request else None
    }
```

```
def delegate(intent_request, session_attributes):
    return {
        'sessionState': {
            'dialogAction': {
                'type': 'Delegate'
            },
            'intent': intent_request['sessionState']['intent'],
            'sessionAttributes': session_attributes,
            'originatingRequestId': 'abc'
        },
        'sessionId': intent_request['sessionId'],
        'requestAttributes': intent_request['requestAttributes'] if 'requestAttributes'
 in intent_request else None
    }


def get_session_attributes(intent_request):
    sessionState = intent_request['sessionState']
    if 'sessionAttributes' in sessionState:
        return sessionState['sessionAttributes']

    return {}


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


""" --- Functions that control the behavior of the bot --- """


def order_birth_stone(intent_request):
    """
    Performs dialog management and fulfillment for ordering a birth stone.
    Beyond fulfillment, the implementation for this intent demonstrates the following:
    1) Use of N best transcriptions to re prompt user when confidence for top
 transcript is below a threshold
    2) Overrides resolved slot for birth month from a known fixed list if the top
 transcript
    is not accurate.
    """
```

```
    transcriptions = intent_request['transcriptions']

    if intent_request['invocationSource'] == 'DialogCodeHook':
        # Disambiguate if there are multiple transcriptions and the top transcription
        # confidence is below a threshold (0.8 here)
        if len(transcriptions) > 1 and transcriptions[0]['transcriptionConfidence'] <
 0.8:
            if transcriptions[0]['resolvedSlots'] is not {} and 'Name' in
 transcriptions[0]['resolvedSlots'] and \
                    transcriptions[0]['resolvedSlots']['Name'] is not None:
                return prompt_for_name(intent_request)
            elif transcriptions[0]['resolvedSlots'] is not {} and 'BirthMonth' in
 transcriptions[0]['resolvedSlots'] and \
                    transcriptions[0]['resolvedSlots']['BirthMonth'] is not None:
                return validate_month(intent_request)

    return continue_conversation(intent_request)


def prompt_for_name(intent_request):
    """
    If the confidence for the name is not high enough, re prompt the user with the
 recognized names
    so it can be confirmed.
    """
    resolved_names = []
    for transcription in intent_request['transcriptions']:
        if transcription['resolvedSlots'] is not {} and 'Name' in
 transcription['resolvedSlots'] and \
                transcription['resolvedSlots']['Name'] is not None:
            resolved_names.append(transcription['resolvedSlots']['Name']['value']
['originalValue'])
    if len(resolved_names) > 1:
        session_attributes = get_session_attributes(intent_request)
        slots = get_slots(intent_request)
        return elicit_slot(session_attributes, intent_request, slots, 'Name',
                           {'contentType': 'PlainText',
                            'content': 'Sorry, did you say your name is {} ?'.format("
 or ".join(resolved_names))})
    else:
        return continue_conversation(intent_request)


def validate_month(intent_request):
```

```
    """
    Validate month from an expected list, if not valid looks for other transcriptions
 and to see if the month
    recognized there has an expected value. If there is, replace with that and if not
 continue conversation.
    """

    expected_months = ['january', 'february', 'march']
    resolved_months = []
    for transcription in intent_request['transcriptions']:
        if transcription['resolvedSlots'] is not {} and 'BirthMonth' in
 transcription['resolvedSlots'] and \
                transcription['resolvedSlots']['BirthMonth'] is not None:
            resolved_months.append(transcription['resolvedSlots']['BirthMonth']
['value']['originalValue'])

    for resolved_month in resolved_months:
        if resolved_month in expected_months:
            intent_request['sessionState']['intent']['slots']['BirthMonth']
['resolvedValues'] = [resolved_month]
            break

    return continue_conversation(intent_request)


def continue_conversation(event):
    session_attributes = get_session_attributes(event)

    if event["invocationSource"] == "DialogCodeHook":
        return delegate(event, session_attributes)


# --- Intents ---


def dispatch(intent_request):
    """
    Called when the user specifies an intent for this bot.
    """

    logger.debug('dispatch sessionId={},
 intentName={}'.format(intent_request['sessionId'],

 intent_request['sessionState']['intent']['name']))
```

```
    intent_name = intent_request['sessionState']['intent']['name']

    # Dispatch to your bot's intent handlers
    if intent_name == 'OrderBirthStone':
        return order_birth_stone(intent_request)

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


# --- Main handler ---


def lambda_handler(event, context):
    """
    Route the incoming request based on intent.
    The JSON body of the request is provided in the event slot.

    """
    # By default, treat the user request as coming from the America/New_York time
 zone.
    os.environ['TZ'] = 'America/New_York'
    time.tzset()
    logger.debug('event={}'.format(event))

    return dispatch(event)
```

セッション管理 API を使用した別のインテントまたはスロット値の選択

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 V2 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することができます。そうすれば、ユーザーが次に操作するインテントが、選択済みのインテントになります。

また、PutSession オペレーションを使用して intent 構造内のスロット値を変更し、代替文字起こしの値を使用するようにすることもできます。

詳細については、「Amazon Lex V2 ボットセッションについて」を参照してください。

Lex V2 ボットで使用する音声文字起こしのカスタマイズ

ボットのデフォルト動作により、音声の文字起こしが不正確になる場合があります。あまり一般的でない、または混同されやすい単語や名前をボットが認識できるようにするために、以下の機能を利用できます。

トピック

- 音声認識モデルの設定
- カスタム語彙による音声認識の向上
- 音声アクティビティの検出感度の設定
- スピーカーダイアライゼーションを使用してプライマリスピーカーに集中する
- 会話内のランタイムヒントによるスロット値の認識の向上
- 会話中のスペルスタイルによるスロット値のキャプチャ

音声認識モデルの設定

Amazon Lex V2 には、ボットの音声認識機能の精度とパフォーマンスを最適化するために選択できるさまざまな音声認識モデルが用意されています。音声モデルの設定を設定して、ユースケースに最適なモデルを選択できます。

音声認識モデルタイプ

Amazon Lex V2 は、次の音声認識モデルをサポートしています。

標準モデル

標準音声認識モデルは、一般的なユースケースで信頼性の高い音声認識パフォーマンスを提供します。このモデルは、幅広いオーディオ条件にわたって一貫した精度を提供し、ほとんどの会話型 AI アプリケーションに適しています。

ニューラルモデル

ニューラル音声認識モデルは、精度を高め、自然な音声パターン、アクセント、バックグラウンドノイズをより適切に処理します。このモデルは、高度なニューラルネットワークアーキテクチャを使用して、特に困難なオーディオ環境で認識パフォーマンスを向上させます。

ディープグラム

Deepgram は、アカウントと API キーを作成するユーザー向けのパブリックspeech-to-text (STT) API を提供します。パブリックオファリングの詳細については、https://deepgram.com/ を参照してください。

音声モデルの設定

ボットロケールを作成または更新するときに、音声モデルの設定を設定できます。音声モデル設定により、Amazon Lex V2 がボットの音声入力を処理するために使用する認識モデルが決まります。

音声モデルの設定を構成するには:

1. Amazon Lex V2 コンソールで、ボットに移動し、設定するロケールを選択します。
2. ボットロケール設定で、音声認識設定セクションを見つけます。
3. 音声モデル設定で、次のいずれかのオプションを選択します。
   - 標準 - 標準の音声認識モデルを使用して、一般的なユースケース全体で信頼性の高いパフォーマンスを実現します。
   - ニューラル - ニューラル音声認識モデルを使用して、精度を高め、自然な音声パターンをより適切に処理します。
   - Deepgram - Deepgram の Listen API を音声認識に使用します。セットアップ手順については、「」を参照してくださいthe section called “Deepgram 音声モデルの設定”。
4. 変更を保存して、音声モデルの設定をボットロケールに適用します。

> **Note**
>
> 音声モデル設定を指定しない場合、Amazon Lex V2 はデフォルトで標準モデルを使用します。

適切な音声モデルの選択

ボットの音声認識モデルを選択するときは、次の要素を考慮してください。

- オーディオ品質 - ボットがバックグラウンドノイズ、さまざまなオーディオ品質、または困難な音響条件でオーディオを処理する場合、ニューラルモデルの方が精度が向上する可能性があります。

- スピーカーの多様性 - ボットがさまざまなアクセントや音声パターンを持つユーザーとやり取りする場合、ニューラルモデルの強化された自然言語処理機能によって認識パフォーマンスが向上する可能性があります。
- パフォーマンス要件 - 標準モデルは一貫したパフォーマンスを提供し、制御されたオーディオ環境とクリアな音声入力を持つアプリケーションには十分です。

特定のユースケースで両方のモデルをテストして、アプリケーションの精度とパフォーマンスの最適なバランスを提供する を決定できます。

Deepgram 音声モデルの設定

Deepgram は、リアルタイムおよびバッチ処理をサポートする高度な AI speech-to-text機能を提供するサードパーティーの音声認識サービスです。Deepgram は、さまざまなオーディオ条件、複数の言語、さまざまなユースケースの特殊なモデルにわたって精度を向上させます。Deepgram のサービスの詳細については、https://deepgram.com/ を参照してください。Deepgram を音声認識モデルの設定として使用するには、1 回限りのセットアッププロセスを完了して Deepgram API キーを設定し、AWS Secrets Manager に安全に保存する必要があります。

> **Important**
>
> Deepgram はサードパーティーのサービスであり、GDPR、FedRAMP、AWS のサービスが準拠するその他のコンプライアンス標準などの特定の規制フレームワークに準拠していない可能性があります。この統合を使用する前に、Deepgram のコンプライアンスドキュメントと組織の要件を確認してください。

リージョンエンドポイントの選択

Amazon Lex V2 は、AWS リージョンに基づいて適切な Deepgram API エンドポイントを自動的に選択して、パフォーマンスとデータのローカル性を最適化します。

- EU リージョン: eu-プレフィックス (eu-west-1、、 などeu-central-1) を持つ AWS リージョンにデプロイされたボットの場合eu-west-2、Amazon Lex V2 は Deepgram EU エンドポイント () を使用しますapi.eu.deepgram.com。
- 他のすべてのリージョン: 他のすべての AWS リージョンにデプロイされたボットの場合、Amazon Lex V2 はグローバル Deepgram エンドポイント () を使用しますapi.deepgram.com。

このエンドポイントの選択は自動であり、現在カスタマイズできません。同じ Deepgram API キーは両方のエンドポイントで機能します。Amazon Lex V2 は、Amazon Connect がカスタムエンドポイント設定に提供する `apiTokenRegion`パラメータをサポートしていません。Amazon Connect のエンドポイント設定オプションの詳細については、「Amazon Connect 管理者ガイド」の「サードパーティー STT プロバイダーのエンドポイントとリージョン」を参照してください。 Amazon Connect

Deepgram API キーの作成

Amazon Lex V2 で Deepgram を使用する前に、Deepgram から API キーを取得する必要があります。

Deepgram API キーを作成するには:

1. https://console.deepgram.com/ で Deepgram コンソールにログインします。
2. 左側のナビゲーションペインで、API キーを選択します。
3. 新しい API キーの作成 を選択します。
4. 手順に従って API キーを作成し、次のセクションで使用するようにコピーします。

> **Important**
>
> API キーを安全に保存してください。次のセクションで AWS Secrets Manager を設定するには、これが必要です。

AWS Secrets Manager での API キーの保存

Deepgram API キーに安全にアクセスするには、Amazon Lex V2 用の AWS Secrets Manager に保存する必要があります。 Amazon Lex シークレットには、 をキー`apiToken`とし、 Deepgram API キーを値とする単一のキーと値のペアが含まれている必要があります。

> **Important**
>
> シークレットで使用する対称 KMS キーを作成する必要があります。デフォルトの AWS マネージド KMS キーは Amazon Lex V2 では機能しません。 V2

Deepgram API キーを Secrets Manager に保存するには:

1. AWS Secrets Manager のコンソールを「https://console.aws.amazon.com/secretsmanager/」で開きます。
2. 新しいシークレットを保存 を選択します。
3. [Secret type] (シークレットタイプ) で、[Other type of secret] (他の種類のシークレット) を選択します。
4. 次のいずれかの方法を使用してシークレットを設定します。
   - キーと値のペアメソッド: キーと値のペアの下に、 をキーapiTokenとして、 Deepgram API キーを値として 1 つのキーと値のペアを追加します。
   - プレーンテキストメソッド: プレーンテキストに、次の構造を持つ JSON オブジェクトを入力します。

```
{
  "apiToken": "your-deepgram-api-key-here"
}
```

5. [次へ] を選択します。
6. シークレットの名前を入力し、次へを選択します。
7. (オプション) 必要に応じてシークレットのローテーションを設定し、次へを選択します。
8. シークレット設定を確認し、保存を選択します。
9. シークレットを作成したら、シークレットに移動し、ARN をコピーします。この ARN は、ボットを設定するときに必要になります。

Secrets Manager のリソースポリシーの設定

Amazon Lex V2 が Deepgram API キーを取得できるようにするには、シークレットにリソースポリシーをアタッチする必要があります。

以下は、Amazon Lex V2 がシークレットを取得できるようにするリソースポリシーの例です。

```
{
  "Version": "2012-10-17"       ,
  "Statement": [
    {
      "Sid": "LexTrust",
      "Effect": "Allow",
```

```
      "Principal": {
        "Service": "lex.amazonaws.com"
      },
      "Action": "secretsmanager:GetSecretValue",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "<YOUR_ACCOUNT_ID>"
        },
        "ArnLike": {
          "aws:SourceArn": "arn:aws:lex:us-east-1:<YOUR_ACCOUNT_ID>:bot-alias/*/*"
        }
      }
    }
  ]
}
```

を実際の AWS アカウント ID <YOUR_ACCOUNT_ID>に置き換え、デプロイに必要な ARN パターンでリージョンを調整します。

Deepgram を使用するようにボットを設定する

Secrets Manager で Deepgram API キーを設定したら、音声認識に Deepgram を使用するように Amazon Lex V2 ボットを設定できます。

ボットの Deepgram を設定するには:

1. Amazon Lex V2 コンソールで、ボットに移動し、設定するロケールを選択します。
2. 音声モデル設定で、Deepgram を選択します。
3. Deepgram 設定の追加フィールドが表示されます。
   - モデル ID (オプション) - 特定のモデルを使用する場合は、Deepgram モデル ID を指定します。使用可能なモデルについては、Deepgram モデルのドキュメントを参照してください。空白のままにすると、API のデフォルトモデルが使用されます。
   - シークレット ARN (必須) - Deepgram API キーを含む AWS Secrets Manager で作成したシークレットの ARN を入力します。
4. 変更を保存して、Deepgram 音声モデル設定をボットロケールに適用します。

これで、音声認識に Deepgram を使用するようにボットが設定されました。ボットをテストして、音声認識が Deepgram 統合で期待どおりに機能していることを確認します。

Deepgram 統合のトラブルシューティング

Deepgram 統合で問題が発生した場合は、以下を確認してください。

- API キーの有効性: Deepgram API キーが有効で、有効期限が切れていないことを確認します。
- シークレット設定: AWS Secrets Manager のシークレットに正しいキー名 (`apiToken`) と API キー値が含まれていることを確認します。
- リソースポリシー: シークレットのリソースポリシーがAmazon Lex V2 が正しいアカウント ID と ARN パターンでアクセスすることを許可していることを確認します。
- KMS キー: デフォルトの AWS マネージドキーではなく、カスタマーマネージド対称 KMS キーを使用していることを確認します。
- モデル ID: モデル ID を指定した場合は、それが有効な Deepgram モデル識別子であることを確認します。

サポートの詳細については、Amazon Lex V2 CloudWatch ログを参照するか、AWS サポートにお問い合わせください。

カスタム語彙による音声認識の向上

特定の言語でカスタム語彙を作成することで、ボットとの音声会話を処理する方法についてより多くの情報を Amazon Lex V2 に提供することができます。カスタム語彙は、音声入力で Amazon Lex V2 に認識させる特定の語句のリストです。これらは通常、Amazon Lex V2 では認識されない固有名詞またはドメイン固有の単語です。

たとえば、テクニカルサポートボットがあるとします。カスタム語彙に「バックアップ」を追加すると、音声が「パックアップ」のように聞こえたとしても、ボットが音声を「バックアップ」として正しく文字起こしできるようになります。カスタム語彙は、金融サービス向けの「ソルベンシー」や「Cognito」や「Monitron」などの固有名詞など、音声に含まれるまれな単語の認識にも役立ちます。

カスタム語彙の基本

- カスタム語彙は、ボットへの音声入力の文字起こしに役立ちます。インテントやスロット値を認識するには、サンプル発話を提供する必要があります。
- カスタム語彙は特定の言語に固有のものです。カスタムボキャブラリーは言語ごとに個別に設定する必要があります。カスタム語彙は、英語 (英国)、英語 (米国)、韓国語 (韓国)、イタリア語 (イタリア)、ドイツ語 (ドイツ)、フランス語 (カナダ)、フランス語 (フランス)、スペイン語 (スペイン)、スペイン語 (米国)、日本語 (日本)、標準中国語 (PRC)、スウェーデン語 (スウェーデン)、

アラビア語湾岸方言 (アラブ首長国連邦)、ヒンディー語 (インド)、オランダ語 (オランダ)、ノルウェー語 (ノルウェー)、ポーランド語 (ポーランド)、フィンランド語 (フィンランド)、ポルトガル語 (ポルトガル)、ポルトガル語 (ブラジル)、カタロニア語 (スペイン) でサポートされています。

- カスタムボキャブラリーは、Amazon Lex V2 がサポートするコンタクトセンターインテグレーションで利用できます。Amazon Lex V2 コンソールのテストウィンドウは、2022 年 7 月 31 日以降に構築されたすべての Amazon Lex V2 ボットのカスタムボキャブラリーをサポートしています。テストウィンドウでカスタムボキャブラリーに関する問題が発生した場合は、ボットを再構築してやり直してください。

Amazon Lex V2 はカスタムボキャブラリーを使用してインテントとスロットの両方を誘発します。インテントとスロットには同じカスタム語彙ファイルが使用されます。スロットタイプを追加するときに、スロットのカスタム語彙機能を選択的にオフにできます。

インテントの誘発 — インテントを誘発するためのカスタム語彙を作成できます。これらのフレーズは、ボットがユーザーのインテントを判断する際の書き起こしに使用されます。たとえば、カスタム語彙に「バックアップ」というフレーズを設定した場合、音声が「写真をパックアップしてくれませんか」と聞こえても。Amazon Lex V2 はユーザー入力を「写真をバックアップしてくれませんか?」と書き起こします。重みを 0、1、2、3 に設定することで、各フレーズのブーストの程度を指定できます。`displayAs` フィールドを追加して、最終的な音声テキスト変換出力のフレーズの代替表現を指定することもできます。

インテントを誘発する際の文字起こしを改善するために使用されるカスタム語彙フレーズは、スロットを誘発する際の文字起こしには影響しません。インテント誘発のためのカスタム語彙の作成方法については、「インテントやスロットを誘発するためのカスタム語彙の作成」を参照してください。

カスタムスロットの誘発 — カスタム語彙を使用して、音声会話のスロット認識を向上させることができます。Amazon Lex V2 ボットがスロット値を認識しやすくするには、カスタムスロットを作成し、そのカスタムスロットにスロット値を追加してから、[スロット値をカスタム語彙として使用] を選択します。スロット値の例には、製品名、カタログ、固有名詞などがあります。カスタムボキャブラリーでは、「はい」や「いいえ」などの一般的な単語やフレーズを使用しないでください。

スロット値を追加すると、ボットがカスタムスロットへの入力を予想しているとき、スロット認識を向上させるためにこれらの値を使用します。これらの値は、インテントを誘発する際の文字起こしには使用されません。詳細については、「スロットタイプの追加」を参照してください。

カスタム語彙を作成するためのベストプラクティス

インテントの誘発

- カスタムボキャブラリーは、特定の単語や句をターゲットにしている場合に最も効果的です。Amazon Lex V2 ですぐに認識されない単語だけをカスタム語彙に追加してください。
- 文字起こしでその単語が認識されない頻度と、入力された単語がどれだけまれであるかに基づいて、単語にどの程度の重みを与えるかを決定します。発音しにくい単語の場合は重みを高くする必要があります。
- 代表的なテストセットを使用して、重みが適切かどうかを判断してください。会話ログのオーディオログをオンにすると、音声テストセットを収集できます。
- カスタム語彙には、「on」、「it」、「to」、「yes」、「no」などの短い単語を使用しないでください。

カスタムスロットの誘発

- 認識が予想されるカスタムスロットタイプに値を追加します。そのスロット値がどれほど一般的または珍しいものであっても、カスタムスロットタイプで使用できるすべてのスロット値を加算します。
- カスタムスロットタイプにカタログ値または商品名や投資信託などのエンティティのリストが含まれている場合にのみ、このオプションを有効にします。
- スロットタイプが「はい」、「いいえ」、「わからない」、「多分」などの一般的なフレーズや、「1」、「2」、「3」などの一般的な語句をキャプチャするために使用される場合は、このオプションを無効にしてください。
- 最高のパフォーマンスを得るには、スロット値とシノニムの数を 500 以下に制限してください。

頭字語など、1 文字ずつ個別に発音する単語は、ピリオドとスペースで区切って 1 文字で入力します。「J. P. Morgan」や「A. W. S.」など、フレーズの一部でない限り、個々の文字は使用しないでください。 頭字語の入力には、大文字と小文字のどちらでも使用できます。

インテントやスロットを誘発するためのカスタム語彙の作成

Amazon Lex V2 コンソールを使用してカスタム語彙を作成および管理することも、Amazon Lex V2 API オペレーションを使用することもできます。コンソールを使用してカスタム語彙を作成する方法は 2 つあります。

コンソール

カスタム語彙をコンソールにインポートする:

1. https://console.aws.amazon.com/lexv2/home から Amazon Lex V2 コンソールを開きます。

2. ボットの一覧から、カスタム語彙を追加するボットを選択します。
3. ボットの詳細ページの [言語の追加] セクションで、[言語を表示] を選択します。
4. 言語のリストから、カスタム語彙を入力する言語を選択します。

コンソールから直接新しいカスタム語彙を作成します。

1. 言語詳細ページの [カスタム語彙] セクションで [作成] をクリックします。編集ウィンドウが開きますが、カスタム語彙は表示されません。
2. 必要に応じて、フレーズ、DisplayAs、重みの入力を追加します。フィールドを更新するか、リストから削除することで、追加した項目をさらにインライン編集できます。
3. [保存] をクリックします。注意: 新しいカスタム語彙は、[保存] をクリックした後にのみボットに保存されます。
4. このページで引き続きインライン編集を行い、編集が完了したら [保存] をクリックします。
5. このページでは、右上のドロップダウンメニューから、カスタム語彙ファイルのインポート、エクスポート、削除を行うこともできます。

API

ListCustomVocabularyItems API を使用してカスタム語彙のエントリを表示します。

1. ListCustomVocabularyItems オペレーションを使用してカスタム語彙のエントリを表示します。リクエスト本文は以下のようになります。

```
{
   "maxResults": number,
   "nextToken": "string"
}
```

2. maxResults と nextToken はリクエスト本文のオプションフィールドであることに注意してください。
3. ListCustomVocabularyItems 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。

```
{
   "botId": "string",
   "botVersion": "string",
   "localeId": "string",
   "customVocabularyItems": [
```

```
        {
            "itemId": "string",
            "phrase": "string",
            "weight": number,
            "displayAs": "string"
        }
    ]
}
```

BatchCreateCustomVocabularyItem API を使用してカスタム語彙のエントリを作成します。

1. ボットの口ケールにまだカスタム語彙が作成されていない場合は、StartImport を使用してカスタム語彙を作成する手順に従ってください。
2. カスタム語彙を作成したら、BatchCreateCustomVocabularyItem オペレーションを使用して新しいカスタム語彙エントリを作成します。リクエスト本文は以下のようになります。

```
{
   "customVocabularyItemList": [
        {
            "phrase": "string",
            "weight": number,
            "displayAs": "string"
        }
   ]
}
```

3. weight と displayAs はリクエスト本文のオプションフィールドであることに注意してください。
4. BatchCreateCustomVocabularyItem のレスポンスは次のようになります。

```
{
    "botId": "string",
    "botVersion": "string",
    "localeId": "string",
    "errors": [
        {
            "itemId": "string",
            "errorMessage": "string",
            "errorCode": "string"
        }
    ],
```

```
    "resources": [
        {
            "itemId": "string",
            "phrase": "string",
            "weight": number,
            "displayAs": "string"
        }
    ]
}
```

5. これはバッチオペレーションであるため、いずれかの項目の作成に失敗してもリクエストは失敗しません。エラーリストには、その特定のエントリでオペレーションが失敗した理由に関する情報が含まれます。リソースリストには、正常に作成されたすべてのエントリが含まれます。
6. BatchCreateCustomVocabularyItem では、次のようなエラーが表示されることが考えられます。

   - RESOURCE_DOES_NOT_EXIST: カスタム語彙が存在しません。このオペレーションを呼び出す前に、カスタム語彙の作成手順に従ってください。
   - DUPLICATE_INPUT: 入力リストに重複するフレーズが含まれています。
   - RESOURCE_ALREADY_EXISTS: エントリに指定されたフレーズが、カスタム語彙にすでに存在します。
   - INTERNAL_SERVER_FAILURE: リクエストの処理中、バックエンドでエラーが発生しました。これはサービスの停止または別の問題を示している可能性があります。

BatchDeleteCustomVocabularyItem API を使用して既存のカスタム語彙エントリを削除します。

1. ボットの口ケールにまだカスタム語彙が作成されていない場合は、StartImport を使用してカスタム語彙を作成する手順に従ってください。
2. カスタム語彙を作成したら、BatchDeleteCustomVocabularyItem オペレーションを使用して既存のカスタム語彙エントリを削除します。リクエスト本文は以下のようになります。

```
{
   "customVocabularyItemList": [
        {
            "itemId": "string"
        }
   ]
```

```
}
```

3. BatchDeleteCustomVocabularyItem のレスポンスは次のようになります。

```
{
    "botId": "string",
    "botVersion": "string",
    "localeId": "string",
    "errors": [
        {
            "itemId": "string",
            "errorMessage": "string",
            "errorCode": "string"
        }
    ],
    "resources": [
        {
            "itemId": "string",
            "phrase": "string",
            "weight": number,
            "displayAs": "string"
        }
    ]
}
```

4. これはバッチオペレーションであるため、いずれかの項目の削除に失敗してもリクエストは失敗しません。エラーリストには、その特定のエントリでオペレーションが失敗した理由に関する情報が含まれます。リソースリストには、正常に削除されたすべてのエントリが含まれます。
5. BatchDeleteCustomVocabularyItem では、次のようなエラーが表示されることが考えられます。

   - RESOURCE_DOES_NOT_EXIST: 削除しようとしているカスタム語彙エントリは存在しません。
   - INTERNAL_SERVER_FAILURE: リクエストの処理中、バックエンドでエラーが発生しました。これはサービスの停止または別の問題を示している可能性があります。

BatchUpdateCustomVocabularyItem API を使用して既存のカスタム語彙エントリを更新します。

1. ボットの口ケールにまだカスタム語彙が作成されていない場合は、StartImport を使用してカスタム語彙を作成する手順に従ってください。

2. カスタム語彙を作成したら、BatchUpdateCustomVocabularyItem オペレーションを使用して既存のカスタム語彙エントリを更新します。リクエスト本文は以下のようになります。

```
{
   "customVocabularyItemList": [
        {
            "itemId": "string",
            "phrase": "string",
            "weight": number,
            "displayAs": "string"
        }
   ]
}
```

3. weight と displayAs はリクエスト本文のオプションフィールドであることに注意してください。

4. BatchUpdateCustomVocabularyItem のレスポンスは次のようになります。

```
{
    "botId": "string",
    "botVersion": "string",
    "localeId": "string",
    "errors": [
        {
            "itemId": "string",
            "errorMessage": "string",
            "errorCode": "string"
        }
    ],
    "resources": [
        {
            "itemId": "string",
            "phrase": "string",
            "weight": number,
            "displayAs": "string"
        }
    ]
}
```

5. これはバッチオペレーションであるため、いずれかの項目の削除に失敗してもリクエストは失敗しません。エラーリストには、その特定のエントリでオペレーションが失敗した理由に関する情報が含まれます。リソースリストには、正常に更新されたすべてのエントリが含まれます。

6. BatchUpdateCustomVocabularyItem では、次のような エラーが表示されることが考えられます。

   - RESOURCE_DOES_NOT_EXIST: 更新しようとしているカスタム語彙エントリは存在しません。
   - DUPLICATE_INPUT: 入力リストに重複する itemIds が含まれています。
   - RESOURCE_ALREADY_EXISTS: エントリに指定されたフレーズが、カスタム語彙にすでに存在します。
   - INTERNAL_SERVER_FAILURE: リクエストの処理中、バックエンドでエラーが発生しました。これはサービスの停止または別の問題を示している可能性があります。

カスタム語彙ファイルの作成

カスタム語彙ファイルは、認識するフレーズ、強調するための重み、および音声文字起こし内のフレーズを置き換える displayAs フィールドを含む値をタブで区切ったリストです。ブースト値が高いフレーズは、音声入力に表示されるときに使用される可能性が高くなります。

カスタム語彙ファイルの名前は **CustomVocabulary.tsv** とし、インポートする前に zip ファイルに圧縮する必要があります。zip ファイルは 300 MB 未満である必要があります。カスタム語彙の語句の最大文字数は 500 です。

- フレーズ: 認識する必要がある 1 ~ 4 単語。フレーズ内の単語はスペースで区切ります。ファイル内のボキャブラリーが重複することは許可されません。フレーズフィールドは必須です。
- 重み - フレーズ認識をブーストする度合いを指定します。値は整数 0、1、2、3 です。重みを指定しない場合、デフォルト値は 1 です。文字起こしでその単語が認識されない頻度と、入力された単語がどれだけまれであるかに基づいて、重みを決定します。重みが 0 の場合、ブーストは適用されず、エントリは displayAs フィールドを使った置換を行う場合にのみ使用されます。
- displayAs – 文字起こし出力におけるフレーズの表示方法を定義します。これはカスタム語彙のオプションフィールドです。

カスタム語彙ファイルには、「phrase」、「weight」、および「displayAs」というヘッダーを含むヘッダー行が含まれている必要があります。ヘッダーはどのような順序でもかまいませんが、上記の命名法に従う必要があります。

次はカスタム語彙ファイルの例です。phrase、weight、displayAs を区切るのに必要なタブ文字は、「[TAB]」というテキストで表されます。この例を使用する場合は、テキストをタブ文字に置き換えてください。

```
phrase[TAB]weight[TAB]displayAs
Newcastle[TAB]2
Hobart[TAB]2[TAB]Hobart, Australia
U. Dub[TAB]1[TAB]University of Washington, Seattle
W. S. U.[TAB]3
Issaquah
Kennewick
```

音声アクティビティの検出感度の設定

音声アクティビティ検出 (VAD) は、音声信号に音声が存在するタイミングを決定するテクノロジーです。Amazon Lex V2 は VAD を使用して、実際の音声とバックグラウンドノイズを区別することで、音声認識の精度を最適化します。VAD 感度レベルを設定して、さまざまな音響環境でボットのパフォーマンスを向上させることができます。

VAD の感度レベルについて

Amazon Lex V2 には、ボットロケール用に設定できる 3 つの VAD 感度レベルが用意されています。

デフォルト

一般的なバックグラウンドノイズレベルを持つほとんどの環境に適した標準感度レベル。これは、一般的なユースケースで推奨される設定です。

HighNoiseTolerance

中程度のバックグラウンドノイズに対する耐性が向上しました。この設定は、多忙なオフィスや小売環境など、一貫した中程度のノイズレベルを持つ環境でボットが動作する場合に使用します。

MaximumNoiseTolerance

高レベルのバックグラウンドノイズの最大許容値。この設定は、コールセンター、製造フロア、環境ノイズが大きい屋外の場所など、非常にノイズの多い環境で使用します。

> **Note**
>
> ノイズ耐性レベルが高いと、システムが音声と考えるものに対してより寛容になり、非常に静かな環境で偽陰性につながる可能性があります。予想される音響環境に最適な感度レベルを選択します。

VAD の機密性の設定

Amazon Lex V2 コンソール、AWS CLI、または SDKs を使用して、ボットロケールを作成または更新するときに VAD の機密性を設定できます。

Amazon Lex V2 コンソールの使用

コンソールで VAD 感度を設定するには

1. https://console.aws.amazon.com/lexv2/ で Amazon Lex V2 コンソールを開きます。
2. リストからボットを選択します。
3. 左側のナビゲーションペインで、ボット言語を選択します。
4. 設定する言語を選択するか、言語の追加を選択して新しい言語を追加します。
5. 音声検出感度セクションで、次のいずれかのオプションを選択します。
   - デフォルト - 一般的な環境の標準感度
   - 高ノイズ耐性 - ノイズが中程度の環境の場合
   - 最大耐ノイズ性 - 非常にノイズの多い環境の場合
6. 保存を選択して変更を適用します。

AWS CLI または SDKs

次の API オペレーションで `speechDetectionSensitivity`パラメータを使用して VAD 感度を設定できます。

- `CreateBotLocale` - 新しいボットロケールを作成するときに VAD の機密性を設定する
- `UpdateBotLocale` - 既存のボットロケールの VAD 感度を変更する
- `DescribeBotLocale` - 現在の VAD 感度設定を表示する

Example AWS CLI での VAD 感度の設定

```
aws lexv2-models create-bot-locale \
    --bot-id "AIDACKCEVSQ6C2EXAMPLE" \
    --bot-version "DRAFT" \
    --locale-id "en_US" \
    --nlu-intent-confidence-threshold 0.40 \
    --speech-detection-sensitivity "HighNoiseTolerance"
```

VAD 設定のベストプラクティス

- ターゲット環境でテストする - ボットがデプロイされる実際の音響条件に基づいて VAD 感度を設定します。
- デフォルトから開始 - デフォルト設定から開始し、パフォーマンステストとユーザーフィードバックに基づいて調整します。
- パフォーマンスのモニタリング - Amazon Lex V2 分析と会話ログを使用して、音声認識の精度をモニタリングし、必要に応じて VAD 感度を調整します。
- ユースケースを検討する - 機密性レベルが高いほどノイズの多い環境には有益ですが、ヘッドセットを備えたカスタマーサービスセンターなどの制御された環境では必要ではない場合があります。

スピーカーダイアライゼーションを使用してプライマリスピーカーに集中する

スピーカーダイアライゼーションは、音声会話中にボットをプライマリ (ルード) スピーカーに維持します。他の音声やバックグラウンドチャターがマイクに到達すると、ボットは周囲の音に反応するのではなく、プライマリスピーカーに集中し続けます。例としては、次の部屋のテレビ、オープンオフィスでのサイド会話、発信者の近くで話している人などがあります。その結果、音声会話がスムーズになり、バックグラウンド音声によって開始されるターンと中断されるプロンプトが少なくなります。

> **Note**
>
> スピーカーダイアライゼーションは、テレフォニーコールなど、StartConversation オペレーションを使用する音声会話のストリーミングに適用されます。8 kHz シングルチャネル (モノラル) オーディオが必要です。Amazon Lex V2 は、テキスト会話や非ストリーミ

ング RecognizeText および RecognizeUtterance オペレーションにスピーカーダイアライゼーションを適用しません。ストリーミング API の詳細については、「」を参照してくださいAmazon Lex V2 ボットへの会話のストリーミング。
スピーカーダイアライゼーションは、Amazon Lex V2 が利用可能なすべての AWS リージョンで使用できます。 V2 リージョンのリストについては、Amazon Web Services 全般のリファレンスの「Amazon Amazon Lex V2 エンドポイントとクォータ」を参照してください。

話者ダイアライゼーションの仕組み

Amazon Lex V2 は常に音声アクティビティ検出 (VAD) を使用して、音声ストリームに音声が存在するタイミングを判断します。VAD は「誰かが話しているのか」という質問に答えます。しかし、「誰が話しているのか？」ではないため、十分な音量の近くの音声でターンを開始できます。スピーカーダイアライゼーションは、ボットロケールに対して有効にするオプションの設定です。VAD の上に 2 番目のチェックが追加されるため、 はプライマリスピーカーのアクティビティに基づいて開始と終了を切り替えます。VAD の詳細については、「」を参照してくださいthe section called “音声アクティビティの検出感度の設定”。

スピーカーダイアライゼーションを有効にすると、Amazon Lex V2 はその決定に 2 番目のシグナルを追加します。

- スピーカーアクティビティ - オーディオの 100 ミリ秒フレームごとにスピーカーがアクティブであるストリーミングダイアライゼーションモデルラベル。最大 4 人の同時スピーカーを追跡します。
- プライマリスピーカーの選択 - Amazon Lex V2 は、検出された各スピーカーのラウドネスを追跡し、最もラウドなスピーカーをプライマリスピーカーとして扱います。このアプローチは、発信者がマイクに最も近いことを前提としています。
- 複合決定 - ダイアライゼーションによって、プライマリスピーカー以外のスピーカーが話していると判断された場合、Amazon Lex V2 はそのオーディオを非音声として扱います。このオーディオは、ターンを開始したり、プロンプトを中断したり、音声認識に到達したりしません。プライマリスピーカーが話していて、ダイアライゼーションが個別のバックグラウンドスピーカーを識別しない場合、Amazon Lex V2 は VAD 決定を使用します。音声キャプチャの開始と終了時にダイアライゼーションゲート。Amazon Lex V2 が既にキャプチャした音声から音声をフィルタリングしません。

このモデルは、ある話者と別の話者を、発言内容ではなく音声特性で区別します。したがって、すべてのボットのロケールと言語で機能します。

プライマリスピーカーの変更

短いラウドサウンドのため、プライマリスピーカーは変更されません。別の話者が主話者になるのは、複数の連続したフレームでアクティブに話している間に、最もラウドトラックされた話者になった後のみです。せき、ドアスラム、短い感嘆符は会話を引き継ぐことはありません。

ウォームアップ期間

このモデルでは、話者追跡を確立するために約 1 秒の音声が必要です。この間、ダイアライゼーションは音声を抑制しないため、モデルのウォームアップ中に発信者の音声の開始がドロップされることはありません。

発信者が最初に話す前に

主な話者は、会話の中でこれまでで最も大きな話者です。バックグラウンドスピーカーがモデルが追跡する最初の音声である場合、そのスピーカーがプライマリスピーカーになります。発信者は、最も大きな追跡対象スピーカーになるのに十分な時間だけ話した後に引き継ぎます。スピーカーダイアライゼーションは、このウィンドウ中の会話を保護しません。大きなバックグラウンド会話でも、発信者が話す前にターンを開始したり、プロンプトを中断したりできます。

自動フォールバック

Amazon Lex V2 は、各会話のダイアライゼーションをモニタリングします。ダイアライゼーションが VAD が検出したほぼすべての音声のフィルタリングを開始すると、Amazon Lex V2 はまずダイアライゼーション状態をリセットします。条件が持続する場合、Amazon Lex V2 は現在のユーザーの残りのターンにダイアライゼーションの適用を停止し、VPD のみを継続します。Amazon Lex V2 は、次のターンの開始時にダイアライゼーションを再度有効にします。ダイアライゼーションの問題により、会話が失敗したり削除されたりすることはありません。

話者ダイアライゼーションではできないこと

- 話者が誰であるかは特定されません。登録、話者名、ユーザーアカウントとの関連付けはありません。ボットは、プライマリスピーカーと他のスピーカーのみを区別します。
- 音声をスピーカーごとのチャネルに分割したり、トランスクリプトでスピーカーラベルを返したりすることはありません。唯一の効果は、Amazon Lex V2 が発信者の音声として扱うオーディオです。
- プライマリスピーカーが話している間にキャプチャされた音声から重複するバックグラウンド音声は削除されません。プライマリスピーカーのターンと重複する音声は、引き続き書き起こされます。ダイアライゼーションは、プライマリスピーカーが停止するとすぐにキャプチャを終了します。

- 会話中の話者を追跡しません。すべての会話は、話者履歴なしで始まります。
- VAD なしでは使用されません。ダイアライゼーションは、置き換えるのではなく VAD の決定を絞り込みます。
- 生成 AI エージェントまたはカスタム推論エンドポイントを使用するボットには適用されません。
- 4 人以上の人が同時に話すと、精度が低下する可能性があります。

スピーカーダイアライゼーションの設定

話者ダイアライゼーションはボットロケール設定であるため、ボットの一部の言語では有効にし、他の言語では無効にすることができます。Amazon Lex V2 コンソール、または AWS CLI と SDKs を使用して、ボットロケールを作成または更新するときに設定できます。

Using the console

1. https://console.aws.amazon.com/lexv2/ で Amazon Lex V2 コンソールを開きます。
2. リストからボットを選択します。
3. 左側のナビゲーションペインで、ボット言語を選択します。
4. 設定する言語を選択するか、言語の追加を選択して新しい言語を追加します。
5. 「スピーカーダイアライゼーション」セクションで、「設定」を選択します。
6. スピーカーダイアライゼーションを有効にするをオンにします。
7. 保存を選択して変更を適用します。
8. 実行時に変更が有効になるように言語を構築します。

言語の詳細ページには、設定の現在の状態が有効または無効と表示されます。

Using the API

スピーカーダイアライゼーションは、次の API オペレーションの `speakerDiarizationSettings`パラメータを使用して設定できます。

- `CreateBotLocale` - 新しいボットロケールのスピーカーダイアライゼーションを設定します。
- `UpdateBotLocale` - 既存のボットロケールのスピーカーダイアライゼーションを変更します。
- `DescribeBotLocale` - 現在のスピーカーダイアライゼーション設定を表示します。

構造には 1 つの必須メンバーがあります。

enabled

Amazon Lex V2 がこのボットロケールのプライマリスピーカーに音声検出を制限するかどうか。有効な値は、true および false です。

インポートファイルのボットロケールspeakerDiarizationSettingsに を含めることで、エクスポートおよびインポート時に設定がボットとともに移動するようにすることもできます。

Example AWS CLI を使用してスピーカーダイアライゼーションを有効にする

```
aws lexv2-models update-bot-locale \
    --bot-id "ABCDE12345" \
    --bot-version "DRAFT" \
    --locale-id "en_US" \
    --nlu-intent-confidence-threshold 0.40 \
    --speaker-diarization-settings '{
        "enabled": true
    }'
```

Example AWS CLI を使用して現在の設定を表示する

```
aws lexv2-models describe-bot-locale \
    --bot-id "ABCDE12345" \
    --bot-version "DRAFT" \
    --locale-id "en_US" \
    --query "speakerDiarizationSettings"
```

speakerDiarizationSettings は によってではなくDescribeBotLocale、 によって返されますListBotLocales。複数の口ケールを監査するには、口ケールDescribeBotLocaleごとに を呼び出します。

> **Note**
>
> 設定を変更したら、 BuildBotLocaleオペレーションを使用してボットロケールを構築します。エイリアスが番号付きボットバージョンを提供する場合は、他のボットロケール設定と同様に、バージョンから新しいDRAFTバージョンを作成し、そのバージョンを指すようにエイリアスを更新します。詳細については、「the section called “Lex V2 ボットを使用したバージョニングとエイリアス”」を参照してください。

Amazon Lex V2 が 設定を解釈する方法

`{"enabled": true}`

音声検出はプライマリスピーカーに制限されます。

`{"enabled": false}`

スピーカーダイアライゼーションはオフになっています。Amazon Lex V2 は VAD のみを使用します。

未設定

設定を一度も設定していないボットではオフになっています。

UpdateBotLocale リクエストspeakerDiarizationSettingsから を省略すると、 は保存された値を保持するため、他のボットロケール設定を変更する更新ではスピーカーダイアライゼーションはオフになりません。この機能が有効になっているボットロケールでこの機能をオフにするには、フィールドを省略するのではなく、{"enabled": false}明示的に を送信します。

話者ダイアライゼーションのベストプラクティス

- 発信者の環境がノイズの多い場所で有効にする - 利点は、テレビがオンになっている自宅の発信者、スピーカーフォンの発信者、オープンオフィス、会議室など、他のスピーカーからの干渉です。one-on-oneクワイエットコールでは、この機能は目立つ効果がないように設計されています。
- 最も大きな音声が発信者として扱われることを期待する - プライマリスピーカーはラウドネスによって選択されます。発信者と家族、発信者とインタープリタなど、2 人のユーザーが通話を共有できます。この場合、Amazon Lex V2 は、クワイエットパーソンが長時間話者になるまで、そのクワイエットパーソンをバックグラウンドとして扱います。2 人が順番に話す会話フローでは、話者のダイアライゼーションをオフにしておきます。

- 長い開始プロンプトの割り込みをオフにする - 発信者が話す前にスピーカーダイアライゼーションが役に立ちません。ボットが長いプロンプトで開き、発信者が常にバックグラウンド会話を行う場合は、そのプロンプトfalseのx-amz-lex:allow-interrupt:*:*セッション属性を に設定します。その後、プロンプトが再生され、発信者が話しかけると、話者のダイアライゼーションが引き継がれます。詳細については、「the section called “Lex V2 ボットでの割り込み動作の仕組み”」を参照してください。
- 中断設定をそのままにしておく - スピーカーダイアライゼーションにより、他の音声による中断が軽減されます。呼び出し元からの中断の仕組みは変更されないため、割り込みとタイムアウトの設定を個別に調整します。
- 代表的な音声でテストする - テレビ、会話の重複、サイドチャットなど、ボットが実際に遭遇するバックグラウンドノイズを含むテストコールで変更を検証します。本番環境にロールアウトする前に、 設定をオフにして、結果を同じ呼び出しと比較します。
- 一度に 1 つの言語をロールアウトする - この設定は 1 つのボットロケールに適用されるため、1 つの言語で有効にし、会話メトリクスとログを確認してから、他の言語に展開できます。

会話内のランタイムヒントによるスロット値の認識の向上

ランタイムヒントを使用すると、Amazon Lex V2 にコンテキストに基づいたスロット値のセットを与えることができます。これにより、音声会話での認識が向上し、スロットの解決が向上します。ランタイムヒントを使用すると、実行時にスロット値の解決の候補となるフレーズのリストを表示できます。

たとえば、フライト予約ボットを操作するユーザーがサンフランシスコ、ジャカルタ、ソウル、モスクワに頻繁に旅行する場合、目的地を誘発する際にこれらの 4 つの都市のリストを含むランタイムヒントを設定して、頻繁に旅行する都市の認識を高めることができます。

ランタイムヒントは、英語 (US) および英語 (UK) のみで使用できます。以下のスロットタイプで使用できます。

- カスタムスロットタイプ
- AMAZON.City
- AMAZON.Country
- AMAZON.FirstNameA
- AMAZON.LastName
- AMAZON.State
- AMAZON.StreetName

ランタイムヒントの基本

- ランタイムヒントは、ユーザーからスロット値を誘発する場合にのみ使用されます。
- ランタイムヒントを使用する場合、ヒントの値は類似の値よりも優先されます。たとえば、フードオーダーボットの場合、メニュー項目のリストをランタイムヒントとして設定し、カスタムスロットの食品項目では、似たような響きの「フェラ」よりも「フィレ」を優先するように誘導できます。
- ユーザー入力がランタイムヒントで提供される値と異なる場合、元のユーザー入力がスロットに使用されます。
- カスタムスロットタイプでは、ボット作成時にカスタムスロットに含まれていなくても、ランタイムヒントとして提供された値がスロットの解決に使用されます。
- ランタイムヒントは 8 kHz 音声入力でのみサポートされます。Amazon Lex V2 がサポートするコンタクトセンターインテグレーションで利用できます。Amazon Lex V2 コンソールのテストウィンドウからの音声入力では 16 kHz の音声入力が使用されるため、ランタイムヒントは提供されません。

> **Note**
>
> 既存のボットでランタイムヒントを使用する前に、まずボットを再構築する必要があります。既存のバージョンのボットはランタイムヒントをサポートしていません。それらを使用するには、ボットの新しいバージョンを作成する必要があります。

PutSession、RecognizeText、RecognizeUtterance、または StartConversation オペレーションを使用してAmazon Lex V2 にランタイムヒントを送信できます。また、Lambda 関数を使用してランタイムヒントを追加できます。

会話の開始時にランタイムヒントを送信してボットで使用される各スロットのヒントを設定したり、会話中のセッション状態の一部としてヒントを送信したりできます。`runtimeHints` 属性は、スロットをそのスロットのヒントにマップします。

Amazon Lex V2 にランタイムヒントを送信すると、セッションが終了するまで会話のたびにヒントが保持されます。null `runtimeHints` 構造を送信すると、既存のヒントが使用されます。ヒントは次の方法で変更できます。

- 新しい `runtimeHints` 構造をボットに送信します。新しい構造の内容が既存の構造と置換されます。

- 空の runtimeHints 構造をボットに送信します。これにより、ボットのランタイムヒントがクリアされます。

コンテキストのスロット値の追加

ユーザーから次に予想される発話に関する情報がアプリケーションに含まれているときに、想定されるスロット値をランタイムヒントとして指定することで、ボットにコンテキストを追加します。Lambda ダイアログコードフックをボットに追加し (詳細については「AWS Lambda関数を Amazon Lex V2 ボットに統合する」を参照)、AWS LambdaLex V2 の入力イベント形式 の [proposedNextState] フィールドを使用して、ユーザーとの会話を改善するために含める必要のあるランタイムヒントを決定します。

たとえば、銀行アプリでは、特定のユーザーのアカウントニックネームのリストを生成し、そのリストを使用してユーザーがアクセスしたいアカウントを引き出すことができます。

ボットがユーザー入力を解釈するのに役立つコンテキストがあれば、会話の開始時にランタイムヒントを送信します。たとえば、ユーザーの電話番号がわかっていれば、この情報を使用してユーザーを検索できます。これにより、ユーザーの認証情報を確認するためにユーザーの名前を求める場合に、PutSession または StartConversation オペレーションを使用して名と姓のヒントをボットに渡すことができます。

会話中に、あるスロット値から別のスロット値に役立つ情報を収集する場合があります。たとえば、自動車ケアアプリでは、ユーザーのアカウント番号がわかっていれば、顧客が所有している車を調べて、別のスロットにヒントとして渡すことができます。

頭字語など、1 文字ずつ個別に発音する単語は、ピリオドとスペースで区切って 1 文字で入力します。「J.P. Morgan」や「A.W.S」などフレーズの一部でない限り、個々の文字は使用しないでください。頭字語の入力には、大文字と小文字のどちらでも使用できます。

スロットへのヒントの追加

ランタイムヒントをスロットに追加するには、sessionState 構造の一部である runtimeHints 構造を使用します。以下は、runtimeHints 構造の例です。「MakeAppointment」インテントの「FirstName」と「LastName」の 2 つのスロットのヒントが表示されます。

```
{
    "sessionState": {
        "intent": {},
        "activeContexts": [],
        "dialogAction": {},
```

```
        "originatingRequestId": {},
        "sessionAttributes": {},
        "runtimeHints": {
            "slotHints": {
                "MakeAppointment": {
                    "FirstName": {
                        "runtimeHintValues": [
                            {
                                "phrase": "John"
                            },
                            {
                                "phrase": "Mary"
                            }
                        ]
                    },
                    "LastName": {
                        "runtimeHintValues": [
                            {
                                "phrase": "Stiles"
                            },
                            {
                                "phrase": "Major"
                            }
                        ]
                    }
                }
            }
        }
    }
}
```

Lambda 関数を使用して、会話中にランタイムヒントを追加することもできます。ランタイムヒントを追加するには、Lambda 関数が Amazon Lex V2 に送信するレスポンスのセッション状態に `runtimeHints` 構造を追加します。詳細については、「AWS LambdaLex V2 のレスポンス形式」を参照してください。

リクエストには有効な `intentName` と `slotName` を指定する必要があります。指定しない場合、Amazon Lex V2 はランタイムエラーを返します。

会話中のスペルスタイルによるスロット値のキャプチャ

Amazon Lex V2 には、名、姓、E メールアドレス、英数字の識別子などのユーザー固有の情報をキャプチャするための組み込みスロットが用意されています。例えば、`AMAZON.LastName` スロッ

トを使用して「Jackson」や「Garcia」などの姓をキャプチャできます。ただし、Amazon Lex V2 は、「Xiulan」のように発音しにくい姓や、ロケールでは一般的ではない姓と混同されることがあります。このような名前をキャプチャするために、文字単位のスペル または 単語単位のスペル でスペルアウトするスタイルで、ユーザーに入力を求めることができます。

Amazon Lex V2 では、3つの スロットエリシテーションスタイル を用意しています。スロットエリシテーションを設定すると、Amazon Lex V2 がユーザーからのスペルを解釈する方法が変わります。

文字単位のスペル — このスタイルでは、フレーズ全体ではなくスペルを聞くようにボットに指示できます。例えば、「Xiulan」などの姓をキャプチャするには、姓を一度に 1 文字ずつスペルアウトするようにユーザーに指示します。ボットはスペルをキャプチャし、文字を単語に変換します。例えば、ユーザーが「x i u l a n」と言うと、ボットは姓を「xiulan」としてキャプチャします。

単語単位のスペル — 特に電話での音声会話で、「t」、「b」、「p」など音が似ているものがあります。英数字の値またはスペリングをキャプチャして誤った値になる場合、その文字とともに識別語を入力するようユーザーに促すことができます。例えば、予約 ID のリクエストに対する音声応答が「abp 123」の場合、ボットは「abb 123」というフレーズを認識してしまう可能性があります。これが間違った値である場合は、「alpha の a、boy の b、peter の P、one two three」と入力するようにユーザーに依頼できます。するとボットは「abp 123」と認識します。

単語単位のスペルを使用する場合は、以下の形式を使用できます。

英語 (米国) 、英語 (英国) 、英語 (オーストラリア) の場合:

- 「as in」(as in apple)
- 「for」(for apple)
- 「like」(like apple)

ドイツ語 (de-DE):

- 「wie」 (p wie ピザ)
- 「für」 (p für ピザ)

フランス語 (fr-FR):

- "comme" (p comme パリ)
- 「pour」 (pchanging paris)

スペイン語 (es-419、es-US、es-ES) の場合:

- 「como」 (u como union)
- 「de」 (u de union)

デフォルト — これは、単語の発音を使用したスロットキャプチャの自然なスタイルです。例えば、「John Stiles」などの名前を自然にキャプチャできます。スロットエリシテーションスタイルが指定されていない場合、ボットはデフォルトのスタイルを使用します。AMAZON.AlphaNumeric および AMAZON.UKPostal コードスロットタイプでは、デフォルトのスタイルでは文字単位によるスペル入力がサポートされています。

「Xiulan」という名前が文字と単語を組み合わせで話される場合、「x-ray の x、i、u、lion の l、a、n」のように、スロットエリシテーションスタイルは単語単位のスペルスタイルに設定する必要があります。文字単位のスタイルでそれは認識されません。

より良いエクスペリエンスを得るには、自然な会話スタイルでスロット値をキャプチャする音声インターフェイスを作成する必要があります。ナチュラルスタイルを使用して正しくキャプチャされない入力については、ユーザーに再度プロンプトを表示して、スロットエリシテーションスタイルを文字単位または単語単位のスペルに設定できます。

次のスロットタイプでは、英語 (米国) 、英語 (英国) 、英語 (オーストラリア) 、ドイツ語 (de-DE)、フランス語 (fr-FR)、スペイン語 (es-419、es-US、es-ES) のspell-by-wordおよびスspell-by-letterスタイルを使用できます。

- AMAZON.AlphaNumeric
- AMAZON.EmailAddress
- AMAZON.FirstNameA
- AMAZON.LastName
- AMAZON.UK 郵便番号 (英語 (米国) 、英語 (英国) 、英語 (オーストラリア) のみ)
- カスタムスロットタイプ

スペルチェックを有効にする

ユーザーからスロットを引き出すときに、実行時に文字単位のスペルと単語単位のスペルを有効にします。PutSession、RecognizeText、RecognizeUtterance、または StartConversation オペレーションでスペルスタイルを設定できます。Lambda 関数を使用して、文字単位のスペルと単語単位のスペルを有効にすることもできます。

スペルスタイルは、前述の API オペレーションのいずれかのリクエストで、または Lambda レスポンスを設定するときに、sessionState フィールドの dialogAction フィールドを使用して設定します (詳細については「AWS LambdaLex V2 のレスポンス形式」を参照)。ダイアログアクションタイプが ElicitSlot で、引き出すスロットがサポートされているスロットタイプの場合のみ、スタイルを設定することができます。

次の JSON コードは dialogAction フィールドは、単語単位のスペルスタイルを使用するように設定されています。

```
"dialogAction": {
    "slotElicitationStyle": "SpellByWord",
    "slotToElicit": "BookingId",
    "type": "ElicitSlot"
}
```

slotElicitationStyle フィールドは、SpellByLetter、SpellByWord、または Default に設定できます。値を指定しない場合、Default となります。

> **Note**
>
> コンソールから文字単位のスペルや単語単位のスペルを有効にすることはできません。

Lambda と Lex V2 を使用する場合のコード例

通常、最初のスロット値を解決しようとしてうまくいかなかった場合は、スペリングスタイルを変更します。次のコード例は Python Lambda 関数で、2 回目のスロットの解決時に単語単位のスペルスタイルを使用します。

サンプルコードを使用するには、以下が必要です。

- 1 つの言語を持つボット、英語 (GB) (en_GB)。
- 1 つのインテント、「アカウントを確認したい」という 1 つのサンプル発話を含む「CheckAccount」。インテント定義の [コードフック] セクションで [初期化と検証に Lambda 関数を使用する]が選択されていることを確認してください。
- インテントには、AMAZON.UKPostalCode 組み込みタイプのスロット「PostalCode」が 1 つ必要です。
- Lambda 関数が定義されたエイリアス。詳細については、「Amazon Lex V2 ボットの AWS Lambda関数の作成」を参照してください。

```
import json
import time
import os
import logging

logger = logging.getLogger()
logger.setLevel(logging.DEBUG)

# --- Helpers that build all of the responses ---

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

def get_session_attributes(intent_request):
    sessionState = intent_request['sessionState']
    if 'sessionAttributes' in sessionState:
        return sessionState['sessionAttributes']
    return {}

def get_slot(intent_request, slotName):
    slots = get_slots(intent_request)
    if slots is not None and slotName in slots and slots[slotName] is not None:
        logger.debug('resolvedValue={}'.format(slots[slotName]['value']
['resolvedValues']))
        return slots[slotName]['value']['resolvedValues']
    else:
        return None

def elicit_slot(session_attributes, intent_request, slots, slot_to_elicit,
 slot_elicitation_style, message):
    return {'sessionState': {'dialogAction': {'type': 'ElicitSlot',
                                              'slotToElicit': slot_to_elicit,
                                              'slotElicitationStyle':
 slot_elicitation_style
                                              },
                             'intent': {'name': intent_request['sessionState']
['intent']['name'],
                                        'slots': slots,
                                        'state': 'InProgress'
                                        },
                             'sessionAttributes': session_attributes,
                             'originatingRequestId': 'REQUESTID'
                             },
```

```
            'sessionId': intent_request['sessionId'],
            'messages': [ message ],
            'requestAttributes': intent_request['requestAttributes']
            if 'requestAttributes' in intent_request else None
            }

def build_validation_result(isvalid, violated_slot, slot_elicitation_style,
 message_content):
    return {'isValid': isvalid,
            'violatedSlot': violated_slot,
            'slotElicitationStyle': slot_elicitation_style,
            'message': {'contentType': 'PlainText',
            'content': message_content}
            }

def GetItemInDatabase(postal_code):
    """
    Perform database check for transcribed postal code. This is a no-op
    check that shows that postal_code can't be found in the database.
    """
    return None

def validate_postal_code(intent_request):

    postal_code = get_slot(intent_request, 'PostalCode')

    if GetItemInDatabase(postal_code) is None:
        return build_validation_result(
            False,
            'PostalCode',
            'SpellByWord',
            "Sorry, I can't find your information. " +
            "To try again, spell out your postal " +
            "code using words, like a as in apple."
        )
    return {'isValid': True}

def check_account(intent_request):
    """
    Performs dialog management and fulfillment for checking an account
    with a postal code. Besides fulfillment, the implementation for this
    intent demonstrates the following:
    1) Use of elicitSlot in slot validation and re-prompting.
    2) Use of sessionAttributes to pass information that can be used to
```

```
        guide a conversation.
    """
    slots = get_slots(intent_request)
    postal_code = get_slot(intent_request, 'PostalCode')
    session_attributes = get_session_attributes(intent_request)

    if intent_request['invocationSource'] == 'DialogCodeHook':
        # Validate the PostalCode slot. If any aren't valid,
        # re-elicit for the value.
        validation_result = validate_postal_code(intent_request)
        if not validation_result['isValid']:
            slots[validation_result['violatedSlot']] = None
            return elicit_slot(
                session_attributes,
                intent_request,
                slots,
                validation_result['violatedSlot'],
                validation_result['slotElicitationStyle'],
                validation_result['message']
            )

        return close(
            intent_request,
            session_attributes,
            'Fulfilled',
            {'contentType': 'PlainText',
             'content': 'Thanks'
             }
        )

def close(intent_request, session_attributes, fulfillment_state, message):
    intent_request['sessionState']['intent']['state'] = fulfillment_state
    return {
        'sessionState': {
            'sessionAttributes': session_attributes,
            'dialogAction': {
                'type': 'Close'
            },
            'intent': intent_request['sessionState']['intent'],
            'originatingRequestId': 'xxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
        },
        'messages': [ message ],
        'sessionId': intent_request['sessionId'],
```

```
        'requestAttributes': intent_request['requestAttributes'] if 'requestAttributes'
 in intent_request else None
    }

# --- Intents ---

def dispatch(intent_request):
    """
    Called when the user specifies an intent for this bot.
    """
    intent_name = intent_request['sessionState']['intent']['name']
    response = None

    # Dispatch to your bot's intent handlers
    if intent_name == 'CheckAccount':
        response = check_account(intent_request)

    return response

# --- Main handler ---

def lambda_handler(event, context):
    """
    Route the incoming request based on the intent.

    The JSON body of the request is provided in the event slot.
    """

    # By default, treat the user request as coming from
    # Eastern Standard Time.
    os.environ['TZ'] = 'America/New_York'
    time.tzset()

    logger.debug('event={}'.format(json.dumps(event)))
    response = dispatch(event)
    logger.debug("response={}".format(json.dumps(response)))

    return response
```

オーディオフィラーを使用してボットの応答性を向上させる

オーディオフィラーは、ユーザーの発話の終わりからボットのレスポンスの開始までの一時停止中に、ライトメロディーやソフトキーストロークなどの短いバックグラウンドオーディオを再生します。これにより、処理の遅延がマスクされ、音声会話が自然に感じられるようになります。

> **Note**
>
> 起動時に、音声フィラーはspeech-to-speechやり取りをサポートし、がunifiedSpeechSettings設定したボットロケールで使用できます。追加の会話モードのサポートは、今後数か月間に展開されます。

使用可能なオーディオフィラータイプ

Amazon Lex V2 には 7 つの組み込みフィラーサウンドがあり、次の 2 つのファミリーで構成されています。

- Melody - チッパーチム
- Melody - Curious Crawl
- Melody - Rising Ripple
- Melody - 患者の Ping
- Melody - ポンダリングポン
- 入力 - キー
- 入力 - クワイエットクワイアティ

Amazon Lex V2 コンソールのオーディオプレビューの再生ボタンを使用して、ボットロケールに保存する前に各オプションを聞きます。

タイミングパラメータ

3 つのタイミングパラメータを調整して、オーディオフィラーが再生されるタイミングとボットレスポンスへの移行方法を制御できます。

`startDelayInMilliseconds`

オーディオフィラーの再生を開始する前に、ユーザーの発話が終了してから待機する時間。有効な範囲は 500 ~ 5000 ミリ秒です。デフォルトは 1000 です。

`minimumPlayDurationInMilliseconds`

ボットレスポンスの準備が早くなっても、オーディオフィラーが開始してから再生される最小時間。有効な範囲は 1000 ~ 5000 ミリ秒です。デフォルトは 3000 です。

`responseDeliveryDelayInMilliseconds`

オーディオフィラーの再生の終了とボットのレスポンスの開始の間に挿入されたサイレント遅延。有効な範囲は 200 ~ 1000 ミリ秒です。デフォルトは 500 です。

オーディオフィラーの設定

Amazon Lex V2 コンソール、Amazon Connect Conversational AI デザイナー、または AWS CLI と SDKs を使用して、ボットロケールを作成または更新するときにオーディオフィラーを設定できます。 V2

Using the console

1. https://console.aws.amazon.com/lexv2/ で Amazon Lex V2 コンソールを開きます。
2. リストからボットを選択します。
3. 左側のナビゲーションペインで、ボット言語を選択します。
4. 設定する言語を選択するか、言語の追加を選択して新しい言語を追加します。
5. オーディオフィラーセクションで、オーディオフィラーを有効にするを選択します。
6. メロディーまたは入力オプションからオーディオタイプを選択します。選択したフィラーを聞くには、オーディオプレビューの再生を使用します。
7. 必要に応じて、開始遅延、最小再生時間、レスポンスバッファのタイミングスライダーを調整します。
8. 保存を選択して変更を適用します。

Using the Amazon Connect Conversational AI designer

1. Amazon Connect 管理ウェブサイトを開き、ボットの対話型 AI デザイナーに移動します。
2. 設定する言語 (ロケール) を開きます。

3. Audio Filler セクションで、オーディオフィラーを有効にし、オーディオタイプを選択します。
4. 開始遅延、最小再生時間、レスポンスバッファ値を調整します。
5. 変更内容を保存します。デザイナーは、基盤となる Amazon Lex V2 ボットロケールaudioFillerSettingsにも同じを適用します。

Using the API

オーディオフィラーは、次の API オペレーションの audioFillerSettingsパラメータを使用して設定できます。

- CreateBotLocale - 新しいボットロケールのオーディオフィラーを設定します。
- UpdateBotLocale - 既存のボットロケールのオーディオフィラーを変更します。
- DescribeBotLocale - 現在のオーディオフィラー設定を表示します。

Example AWS CLI を使用してオーディオフィラーを設定する

```
aws lexv2-models update-bot-locale \
    --bot-id "bot-1234567890abcdef0" \
    --bot-version "DRAFT" \
    --locale-id "en_US" \
    --nlu-intent-confidence-threshold 0.40 \
    --audio-filler-settings '{
        "enabled": true,
        "audioType": "MELODY_CHIPPER_CHIME",
        "startDelayInMilliseconds": 1000,
        "minimumPlayDurationInMilliseconds": 3000,
        "responseDeliveryDelayInMilliseconds": 500
    }'
```

AI エージェント中間メッセージを含むオーディオフィラー

オーディオフィラーは AI エージェントの中間メッセージと連動します。AI エージェントが発信者に中間メッセージを送信すると (例えば、「確認させてください」) 、開始遅延タイマーは元の呼び出しからではなく、その中間メッセージの末尾から測定されます。これにより、音声フィラーがエー

ジェントの音声と重複するのを防ぎ、発信者が経験する遅延が、聞いた最新の音声から測定されます。

ダイアログおよびフルフィルメントコードフックを備えたオーディオフィラー

オーディオフィラーは、Lambda 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によって生じる処理ギャップ中にも再生されます。同じタイミングパラメータが適用されるため、ボットが処理を AI エージェント、コードフック、またはその両方に同じターンで委任するかどうかにかかわらず、発信者は一貫したエクスペリエンスを聞きます。

オーディオフィラーのベストプラクティス

- フィラーをブランドの音声に一致させます。コンシューマーエクスペリエンスや小売エクスペリエンスにはメロディーフィラーを使用し、ユーザーがボットがタスクに積極的に取り組んでいることを期待する場合はフィラーを入力します。
- レイテンシープロファイルの開始遅延を調整します。ほとんどのボットレスポンスが よりも高速の場合`startDelayInMilliseconds`、フィラーが再生されることはほとんどありません。レイテンシーの多いワークロードの遅延を短縮し、応答の速いボットの場合はそれを増やします。
- 高速ボットの最小再生時間は短くします。高速ボット`minimumPlayDurationInMilliseconds`を長くすると、レスポンスの準備ができたらフィラーを保持することで、認識されるレイテンシーが追加されます。
- 代表的なトラフィックでテストします。実稼働環境にロールアウトする前に、現実的な会話でフィラーの選択とタイミングを検証します。

Lex V2 のボットパフォーマンスのモニタリング

モニタリングは、Amazon Lex V2 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、会話ログを使用してユーザーとチャットボット間の会話をモニタリングし、発話統計を使用してボットが検出および見逃す発話を判断し、Amazon CloudWatch Logs と を使用して Amazon Lex V2 AWS CloudTrail をモニタリングする方法について説明します。Amazon Lex V2 ランタイムとチャンネル関連付けの説明もあります。

これらのツールとメトリクスを使用して、ボットのパフォーマンスを向上させるためにどのような方向性やアクションを取ることができるかを理解してください。

トピック

- Analytics によるビジネスパフォーマンスの測定
- Lex V2 ボットの会話ログの有効化
- Lex V2 のエラーログを使用したエラーのログ記録
- Lex V2 でのオペレーションメトリクスのモニタリング
- Test Workbench による Lex V2 ボットのパフォーマンス評価

Analytics によるビジネスパフォーマンスの測定

Analytics では、ボットと顧客とのやり取りの成功率と失敗率に関連する指標を使用して、ボットのパフォーマンスを評価できます。ボットと顧客間の会話の流れのパターンを視覚化することもできます。Analytics はこれらの指標をグラフやチャートにまとめることで、エクスペリエンスを効率化します。Analytics には、結果をフィルタリングして、インテント、スロット、発話、会話に関する問題や問題を特定するのに役立つツールが用意されています。このデータを活用してボットを改良し、カスタマーエクスペリエンスを改善できます。

> **Note**
>
> ユーザーが Analytics にアクセスするには、AWS マネージドポリシー: AmazonLexFullAccess または 分析 API 権限を含むカスタムポリシーまたはのいずれかを IAM ロールにアタッチする必要があります。カスタムポリシーを使用してユーザー権限を処理する方法の詳細については、「分析のアクセス許可の管理」を参照してください。AWS マネージドポリシー: AmazonLexReadOnly が顧客の IAM ロールにアタッチされている場

合、ユーザーが Analytics ダッシュボードにアクセスするためにユーザーの IAM ロールに追加べき権限がありません、というエラーが表示されます。

Analytics にアクセスするには

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/lexv2/home で Amazon Lex V2 コンソールを開きます。
2. ナビゲーションペインの [ボット] で、Analytics に表示したいボットを選択します。
3. [Analytics] の下で、表示するセクションを選択します。

トピック

- 主な定義
- 結果のフィルタリング
- 概要: ボットパフォーマンスの概要
- 会話ダッシュボード: ボット会話の要約
- パフォーマンスダッシュボード: ボットのインテントと発話のメトリクスの概要が表示されます。
- 分析に API を使用する
- 分析のアクセス許可の管理

主な定義

このトピックでは、ボット分析を解釈するのに役立つ重要な定義を紹介します。これらの定義は、インテント、スロット、会話、発話の 4 つのコンテキストにおけるボットのパフォーマンスに関係しています。以下のフィールドは多くのパフォーマンスメトリクスに関係します。

- `Intent` オブジェクトの `state` フィールド。
- SessionState オブジェクト内の `dialogAction` オブジェクトの `type` フィールド。

インテント

Amazon Lex V2 では、インテントを以下の方法で分類しています。

- 成功 — ボットはインテントを正常に実行しました。以下のいずれかの状況に該当します:
  - インテント `state` は `ReadyForFulfillment` で、`dialogAction` の `type` が `Close` です。

  - インテント state は Fulfilled で、dialogAction の type が Close です。
- 失敗 — ボットはインテントを実行できませんでした。インテントの状態。以下のいずれかの状況が該当します。
  - インテントの state が Failed で、dialogAction の type が Close の場合 (例えば、ユーザーが確認プロンプトを拒否した場合など)。
  - ボットはインテントが完了する前に AMAZON.FallbackIntent に切り替わります。
- 切り替え — ボットは別のインテントを認識し、元のインテントが成功または失敗に分類される前に、代わりにそのインテントに切り替えます。
- 除外 — インテントが成功または失敗に分類されるまで、顧客は応答しません。

スロット

Amazon Lex V2 では、スロットを以下の方法で分類しています。

- 成功 — ボットはスロットを埋め、確認ステップで別のスロットに正常に移行しました。
- 失敗 — 再試行回数が上限に達したにもかかわらず、ボットはスロットを埋めることができませんでした。
- 除外 — スロットが成功または失敗に分類されるまで、顧客は応答または別のインテントに切り替わりません。

会話

顧客が Amazon Lex V2 に対してランタイムコールを行うと、sessionIdが提供され、Amazon Lex V2 は originatingRequestId を生成します。ボットに設定したセッションタイムアウト (idleSessionTTLInSeconds) 内に顧客が応答しない場合、セッションは期限切れになります。顧客が同じ sessionId を使用してセッションに戻ると、Amazon Lex V2 は新しい originatingRequestId を生成します。

分析では、会話は sessionId と originatingRequestId を独自に組み合わせたものです。Amazon Lex V2 では、会話を以下の方法で分類しています。

- 成功 — 会話の最終意図は成功と分類されます。
- 失敗 — 会話の最終インテントが失敗しました。Amazon Lex V2 がデフォルトで AMAZON.FallbackIntent に設定されている場合も、会話は失敗します。
- 除外 — 会話が成功または失敗に分類されるまで、顧客は応答しません。

発話

Amazon Lex V2 では、発話を以下の方法で分類しています。

- 検出 - Amazon Lex V2 が、ボット用に設定されたインテントを呼び出そうとしていると認識します。
- 見逃し - Amazon Lex V2 が発話を認識しません。

結果のフィルタリング

各ページの上部で、ボット分析の結果をフィルタリングできます。

また、次のパラメータでフィルタリングできます。

- 時間 — 結果を相対時間範囲または絶対時間範囲でフィルタリングできます。開始時刻と終了時刻を選択すると、Amazon Lex V2 は開始時刻後に開始され、終了時刻より前に終了した会話を取得します。
  - 相対範囲 — 過去 1 日の結果を表示するには [1d]、過去 1 週間は [1w]、過去 1 か月は [1m] を選択します。

    その他のオプションについては、[カスタム] を選択し、[相対範囲] メニューで期間を選択します。期間をより細かく制御するには、[カスタム範囲] を選択し、[期間] フィールドに数値を入力して、ドロップダウンメニューから [時間の単位] を選択します。
  - 絶対範囲 — [カスタム] を選択し、[絶対範囲] メニューを選択すると、指定した時間範囲内の会話をフィルタリングできます。カレンダーで開始日と終了日を選択するか、YYYY/MM/DD 形式で入力できます。

  > **Note**
  >
  > 分析時間範囲には、次の制限があります。
  >
  > - 開始日は、現在の日付から過去 365 日以内である必要があります。
  > - 終了日は、開始日から 1 か月を超えることはできません。

- ボットフィルター — ロケール、エイリアス、ボットのバージョンでフィルタリングするには、[すべてのロケール]、[すべてのエイリアス]、[すべてのバージョン] というラベルの付いたドロップダウンメニューを選択します。

- モダリティ — 歯車アイコンを選択し、[モダリティ] ドロップダウンメニューを選択して、[スピーチ] または [テキスト] のどちらの結果を表示するかを選択します。
- チャネル — 歯車アイコンを選択し、[チャネル] ドロップダウンメニューを選択して、結果を表示するチャネルを選択します。チャネル統合の詳細については、「Amazon Lex V2 ボットのメッセージングプラットフォームとの統合」および「Amazon Connect コンタクトセンター」を参照してください。

概要: ボットパフォーマンスの概要

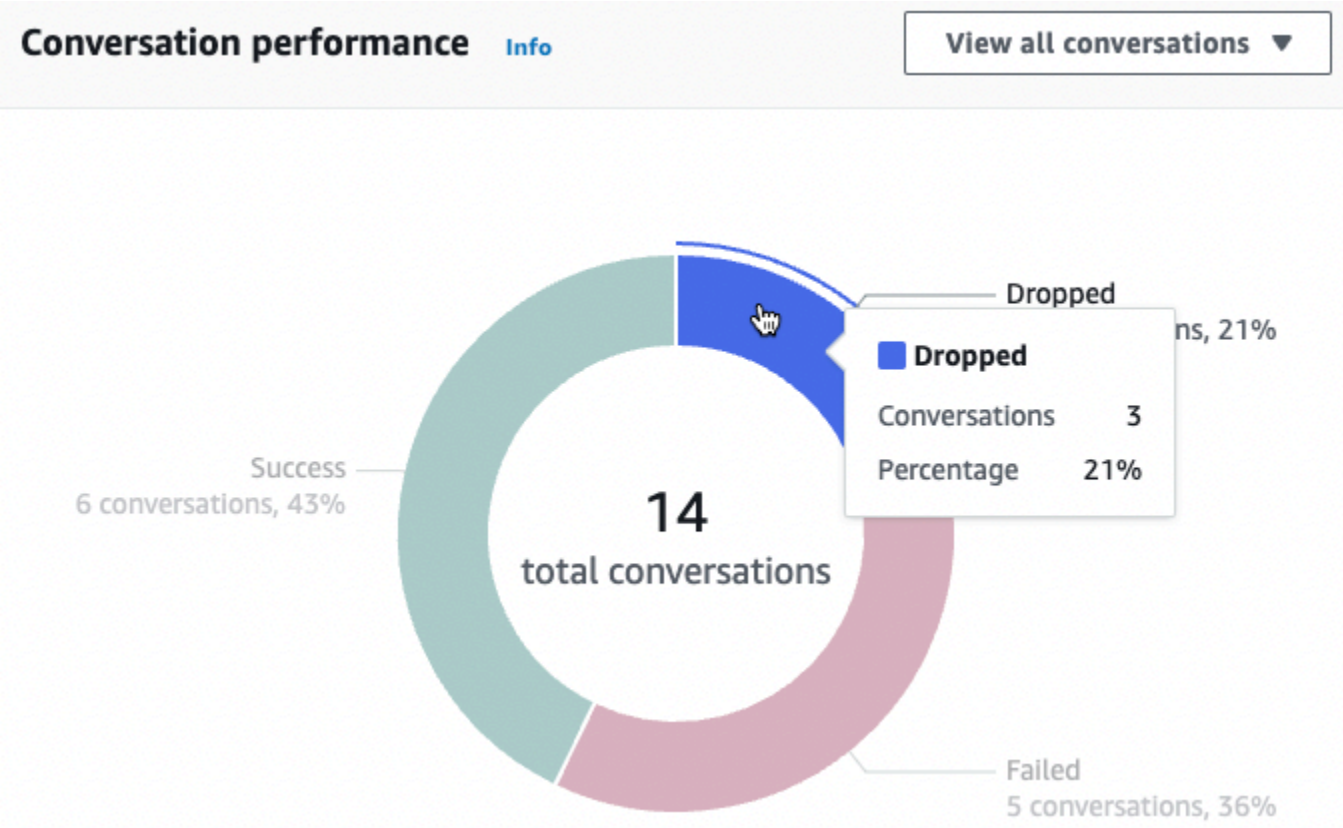
概要ページには、会話、発話認識、インテントの使用に関するボットのパフォーマンスがまとめられています。概要には次のセクションがあります。

- 会話パフォーマンス
- 発話認識率
- 会話パフォーマンス履歴
- 使用されたインテントの上位 5 位
- 失敗したインテントの上位 5 位

会話パフォーマンス

このグラフを使用して、成功、失敗、除外に分類された会話の数と割合を追跡できます。会話のリストにアクセスするには、[すべての会話を表示] を選択してドロップダウンメニューを表示します。ボットとのすべてのユーザー会話のリストを表示するか、特定の結果 (成功、失敗、または除外) を伴う会話をフィルターするかを選択できます。これらのリンクをクリックすると、[会話ダッシュボード] の [会話] サブセクションに移動します。詳細については、「会話」を参照してください。

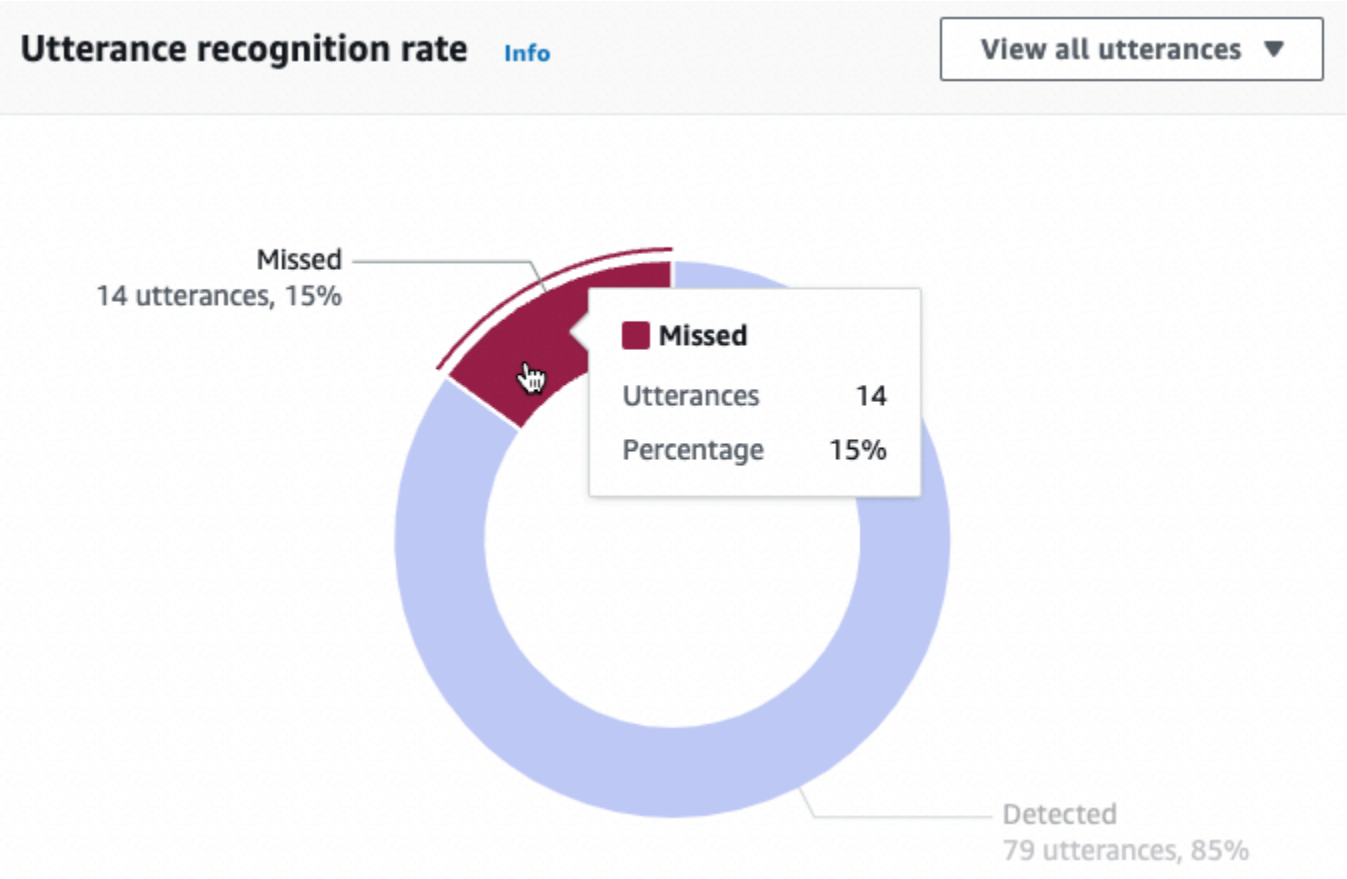
その結果になった会話の数と割合を示すボックスを表示するには、次の画像のように、グラフのセグメントにカーソルを合わせます。



発話認識率

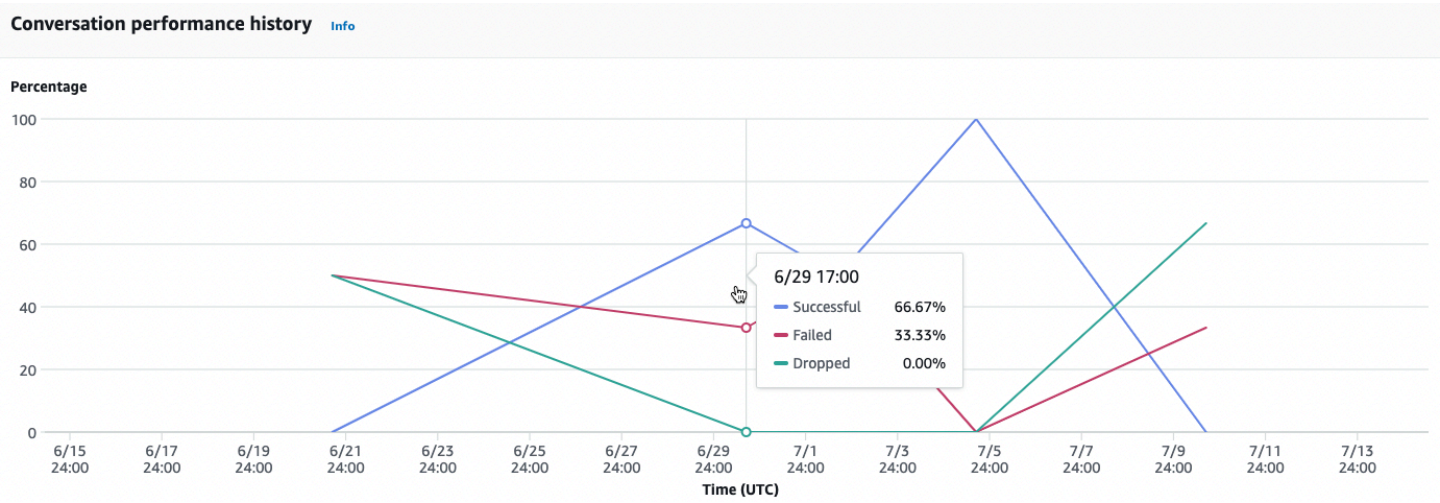
このグラフを使用して、ボットによって検出された発話と、見逃された発話の数と割合を追跡できます。発話のリストにアクセスするには、[発話を表示] を選択してドロップダウンメニューを表示します。すべてのユーザー発話のリストを表示するか、特定の結果 ([見逃した]、または [検出] を含む発話をフィルタリングするかを選択できます。これらのリンクをクリックすると、[パフォーマンスダッシュボード] の [発話認識] サブセクションに移動します。詳細については、[発話を表示する] を参照して 発話認識 に移動します。

会話の数と割合を示すボックスを表示するには、次の画像のように、グラフのセグメントにカーソルを合わせます。



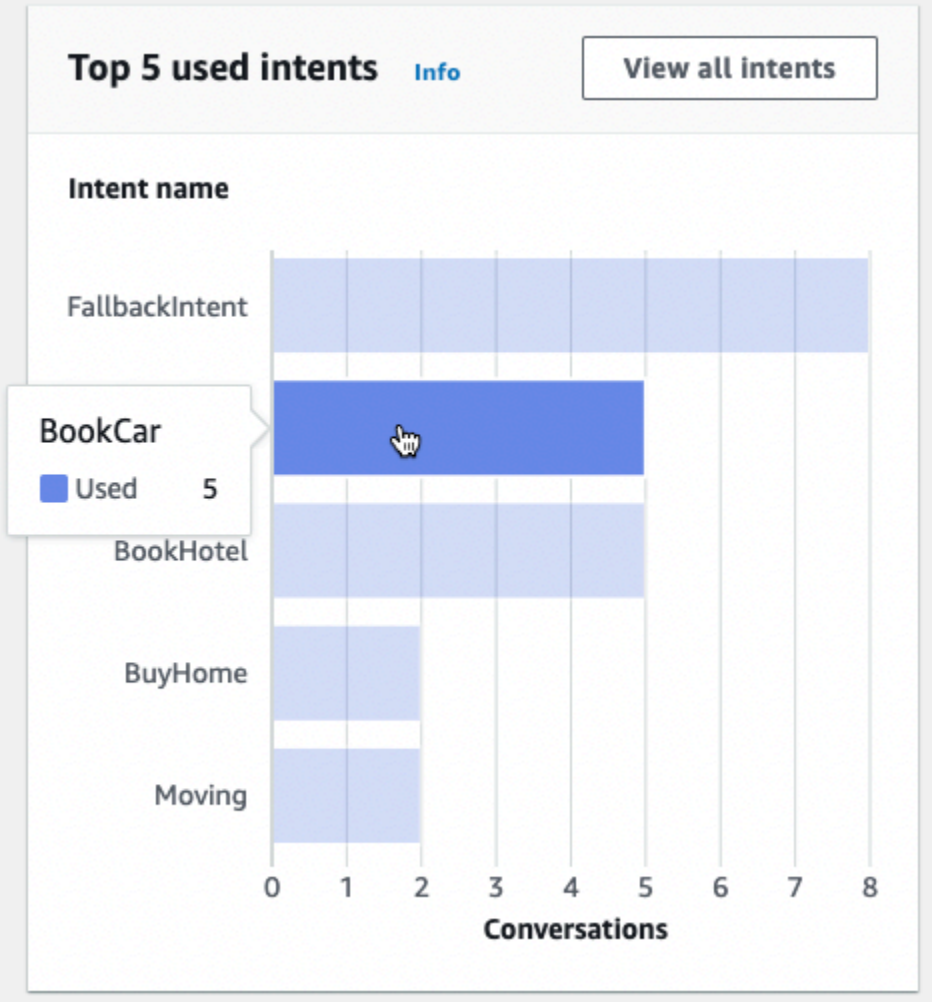
会話パフォーマンス履歴

このグラフを使用して、フィルターで設定した時間範囲で、成功、失敗、および除外に分類された会話の割合を追跡できます。特定の期間に特定の結果をもたらした会話の割合を確認するには、次の図のようにその間隔にカーソルを合わせます。



使用されたインテントの上位 5 位

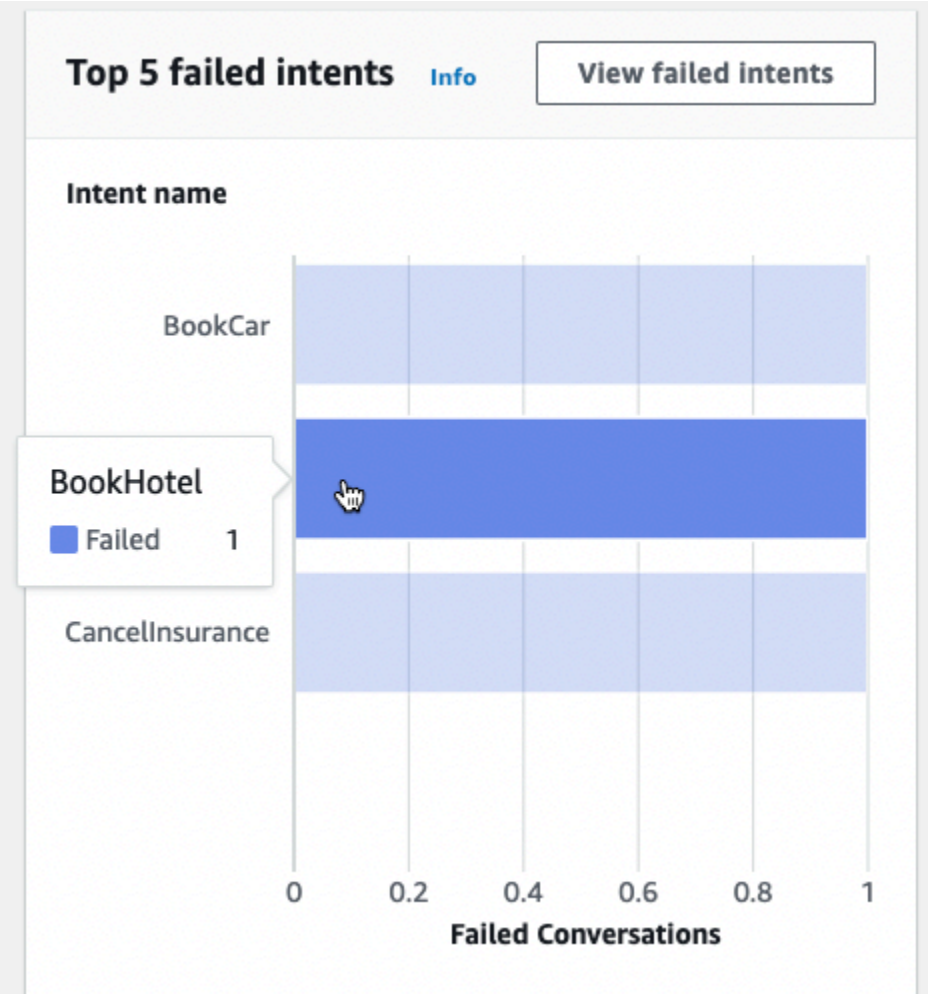
このグラフを使用して、顧客がボットで使用したインテントの上位 5 つを特定してください。棒にカーソルを合わせると、以下の画像のように、ボットがそのインテントを認識した回数が表示されます。



[すべてのインテントを表示] を選択すると、[パフォーマンスダッシュボード] の [インテントのパフォーマンス] サブセクションに移動します。ここでは、インテントを満たしているボットのパフォーマンスのメトリクスを確認できます。詳細については、「インテントパフォーマンス」を参照してください。

失敗したインテントの上位 5 位

このチャートを使用して、ボットが履行に失敗したインテントの上位 5 つを特定してください (失敗したインテントの定義については、「インテント」を参照してください)。棒にカーソルを合わせると、以下の画像のように、ボットがそのインテントの履行に失敗した回数が表示されます。



[失敗したインテントを表示] を選択すると、[パフォーマンスダッシュボード] の [インテントのパフォーマンス] サブセクションに移動します。ここでは、ボットが履行に失敗したインテントのメトリクスを確認できます。詳細については、「インテントパフォーマンス」を参照してください。

会話ダッシュボード: ボット会話の要約

会話ダッシュボードは、顧客とボットとの会話 (会話の定義については「会話」を参照) の指標を視覚化します。

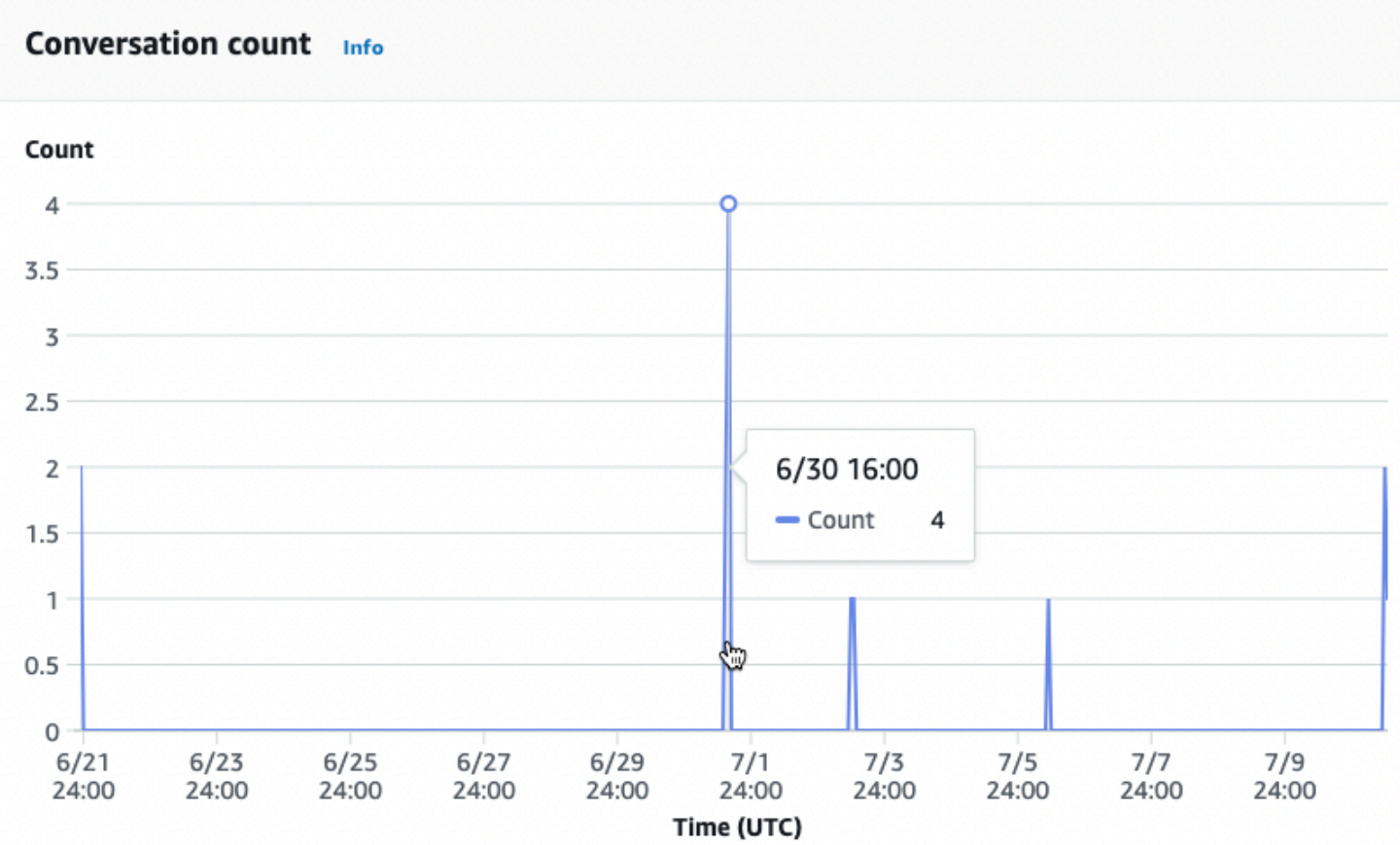
概要には、ボットとのユーザー会話に関する以下の情報が含まれます。数値はフィルター設定に基づいて計算されます。

- 会話の合計 — ボットとの会話の合計数。
- 平均会話時間 — ユーザーとボットとの会話の平均時間 (分と秒)。形式は mm:ss です。
- 会話あたりの平均ターン数 — 会話が続く平均ターン数です。

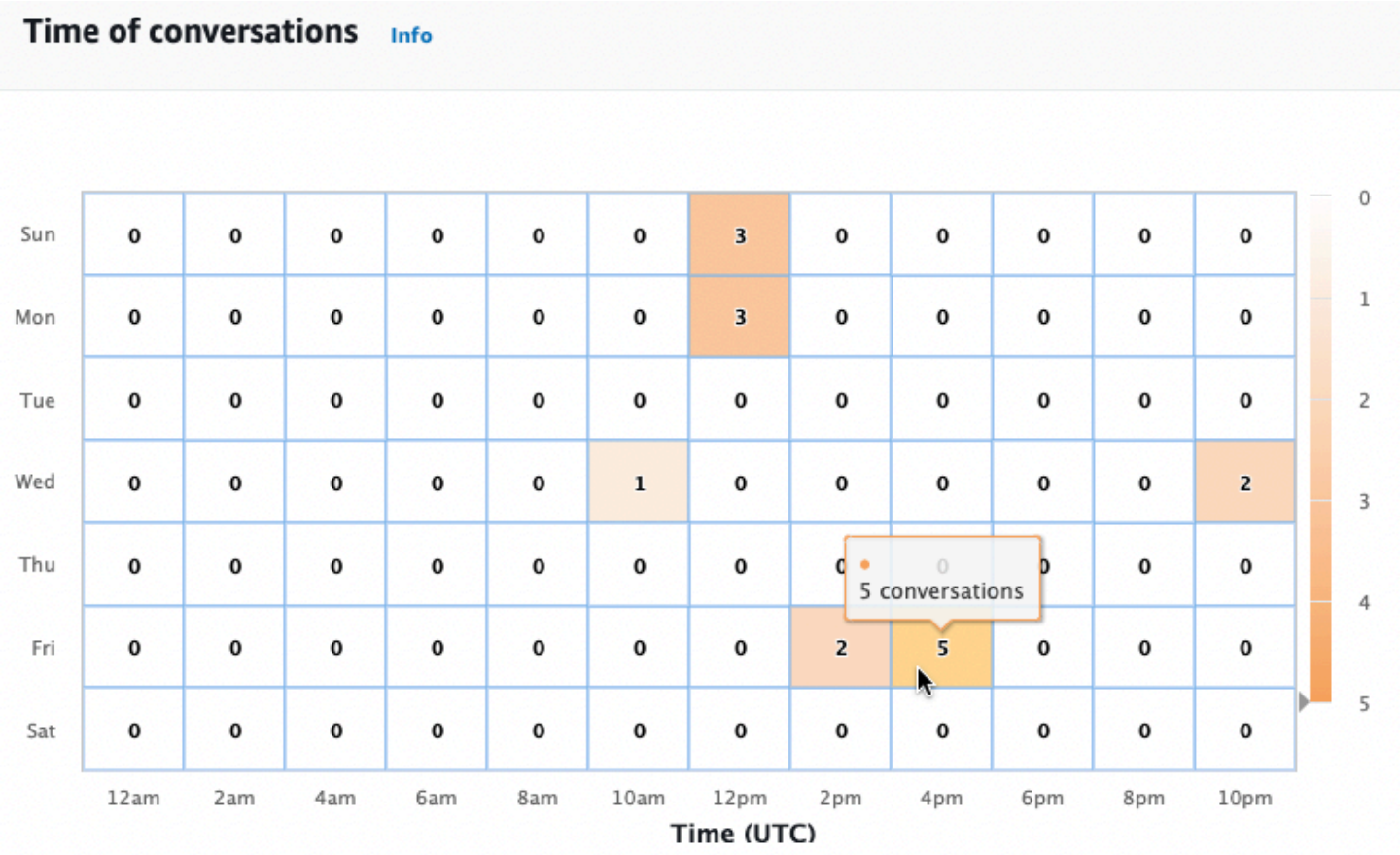
[会話数] セクションと [メッセージ数] セクションにはそれぞれ、フィルターで指定した期間における会話数とメッセージ数を示すグラフがあります。特定の時間セグメントにカーソルを合わせると、そのセグメント内の会話またはメッセージの数が表示されます。タイムセグメントのサイズは、指定した時間範囲によって異なります。

- 1 週間未満 — 1 時間ごとにカウントが表示されます。
- 1 週間以上 — 日ごとにカウントが表示されます。

ホバリング動作の例を次の画像に示します。



[会話時間] セクションには、フィルターで指定した時間範囲内で、各曜日の 2 時間間隔にボットと顧客との間で行われた会話の数が表示されます。濃い色のセルは、より多くの会話が行われた時間を示します。セルにカーソルを合わせると、その時間帯から始まる 2 時間以内の会話数が表示されます。たとえば、次の画像のアクションは、UTC の午後 4 時 から午後 6 時 までの間に発生した会話の数を示しています。



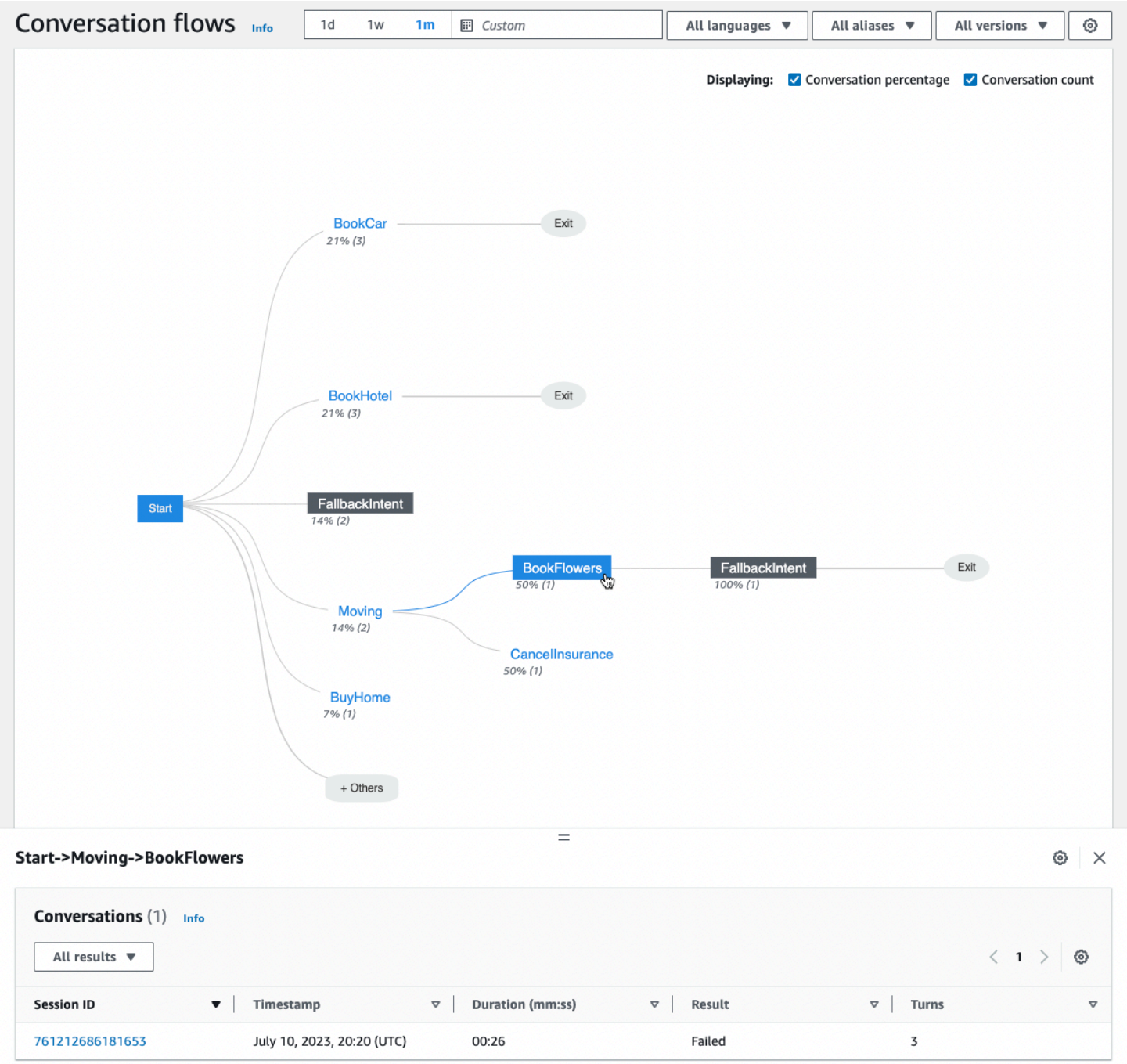
[会話ダッシュボード] には、会話フローと会話という 2 つのツールがあります。ツールにアクセスするには、左側のナビゲーションペインの [会話ダッシュボード] でツールを選択します。

会話フロー

会話フローを使用すると、ボットとの会話で顧客が取るインテントの順序を視覚化できます。各インテントの下には、会話のその時点でそのインテントを呼び起こした会話の割合と数が表示されます。割合と数切り替えるには、上部にある [会話の割合] と [会話数] を選択します。デフォルトでは、会話のその時点で最も頻度の高い 5 つのインテントが、頻度の降順に表示されます。[+ その他] を選択すると、すべてのインテントが表示されます。

インテントを選択すると、新しいブランチ列に展開され、会話のその時点で取られたインテントが頻度の降順で一覧表示されます。

会話フロー内のノードを選択すると、下のウィンドウを展開して、そのインテントの順序に従った会話のリストを表示できます。会話に対応する [セッション ID] を選択すると、その会話の詳細が表示されます。次の図は、会話フローと、下部にある拡張された [会話] ウィンドウを示しています。



会話

会話ツールには、ボットとの会話のリストが表示されます。列を選択して、その列を基準に昇順または降順でソートできます。

会話を結果でフィルタリングするには、[すべての結果] を選択し、[成功]、[失敗]、または [除外] を選択します。

会話を期間でフィルタリングするには

1. [会話を期間でフィルタリングする] とマークされた検索バーを選択します。
2. 次のいずれかの方法でフィルターを定義します。
   - 定義済みのオプションを使用します。
     a. [期間] を選択します。
     b. = (等しい)、> (より大きい)、< (より小さい) 演算子から選択します。
     c. 時間の長さを選択します。
   - 「Duration {operator} {number} sec」の形式で入力してください。たとえば、30 秒を超える会話をすべて検索するには、**`Duration > 30 sec`** と入力します。時間の長さを秒単位で指定します。

メタデータ、インテントの使用状況、文字起こしなど、セッションに関する詳細情報を表示するには、会話の [セッション ID] を選択します。

> **Note**
>
> 会話は `sessionId` と `originatingRequestId` を独自に組み合わせたものであるため、テーブルには同じ `sessionId` が複数回表示されることがあります。

[詳細] セクションには、次のメタデータがあります。

- タイムスタンプ — 会話の日付と開始時間を指定します。時刻は hh:mm:ss 形式です。
- 時間 — 会話の継続時間を mm:ss 形式で指定します。この期間にセッションタイムアウト時間 (`idleSessionTTLInSeconds`) は含まれません。
- 結果 — 会話が成功、失敗、除外のいずれに分類されたかを指定します。これらの結果の詳細については、「会話」を参照してください。
- モード — 会話が `Speech`、`Text`、または `DTMF` (タッチトーンキーパッドの押下) のいずれであったかを指定します。複数のモードで構成される会話は、`Multimode` です。
- チャネル — 該当する場合、会話が行われたチャネルを指定します。「Amazon Lex V2 ボットのメッセージングプラットフォームとの統合」を参照してください。
- 言語 — ボットの言語を指定します。

ボットが会話の中で誘発したインテントが [詳細] の下に表示されます。[インテントに移動] を選択すると、インテント工ディタでそのインテントに移動します。[文字起こしにスナップ] を選択すると、ボットがインテントを誘発した最初のインスタンスまで文字起こしが自動的にスクロールされます。

インテント名の横にある右矢印を選択すると、スロットの名前、ボットが各スロットに対して誘発した値、ボットが各スロットを誘発しようとした回数など、インテントで誘発されたスロットの詳細が表示されます。

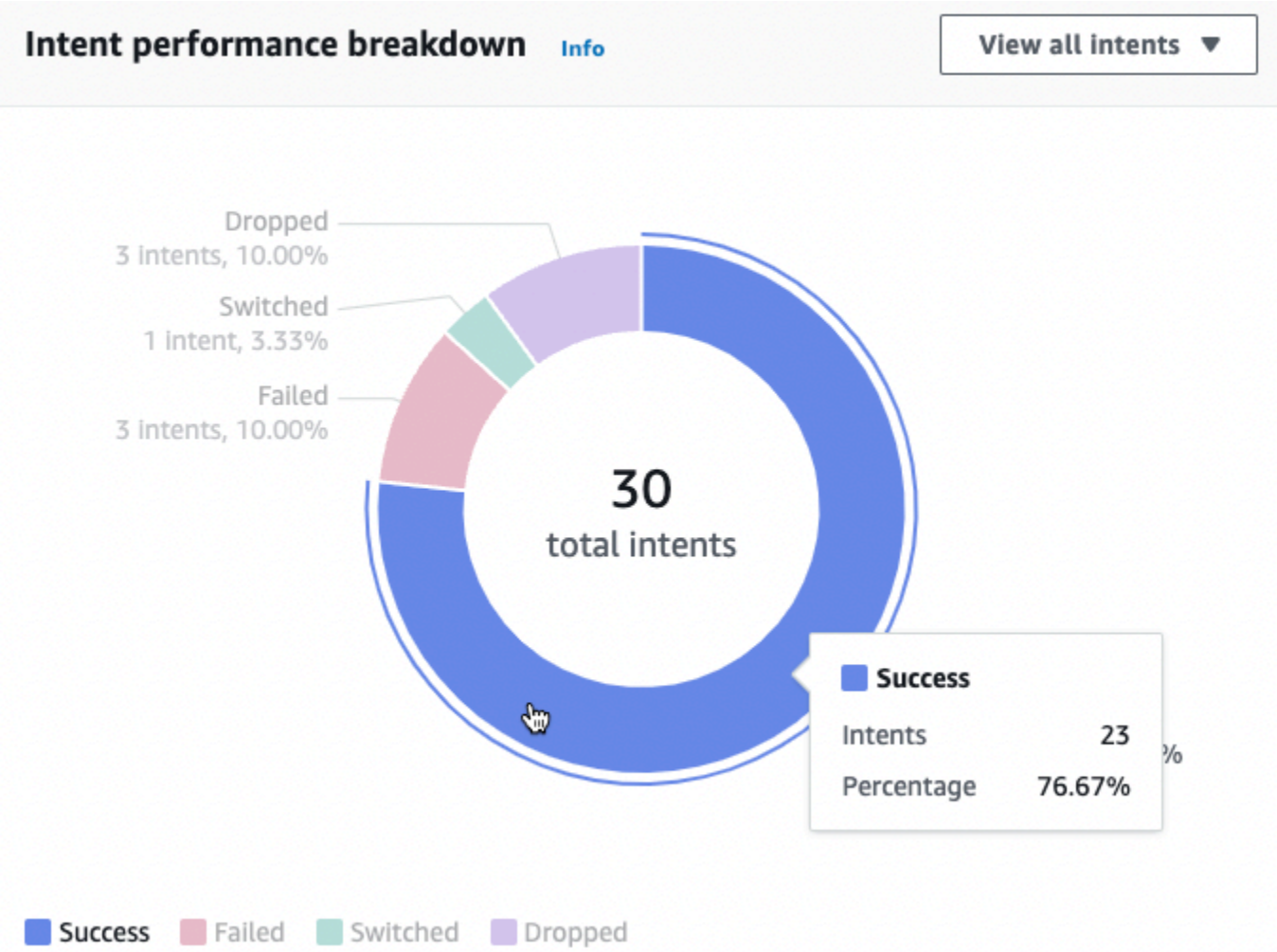
[文字起こし] では、会話の発話や、インテントやスロットを誘発する際のボットの行動を確認できます。ユーザーの発話は左側に表示され、ボットの発話は右側に表示されます。[このセッションの文字起こしをフィルタ] とマークされた検索バーを使用して、文字起こし内のテキストを検索します。[表示中:] の横には、会話の各ターンの下に表示される 3 つの情報が表示され、表示するかしないかを選択できます。

- タイムスタンプ — 発話の時間を指定します。
- インテント状態 — 発話中にボットが誘発しているインテントと、該当する場合はインテントの結果を指定します。インテント状態には以下のものがあります。
  - 呼び出されたインテント: ###### — ユーザーが呼び出しているインテントをボットが識別しました。
  - 切り替えられたインテント: ###### — ボットは発話に基づいて別のインテントに切り替わりました。
  - ######: 成功 — ボットはインテントを実行しました。
- スロット状態 — 発話中にボットが誘発しているスロット (該当する場合) と、顧客が提供する値を指定します。

パフォーマンスダッシュボード: ボットのインテントと発話のメトリクスの概要が表示されます。

パフォーマンスダッシュボードでは、ボットのインテントフルフィルメントと発話認識のパフォーマンスに関する詳細を確認できます。

[インテントパフォーマンスの内訳] セクションには、ボットがインテントを呼び出した合計回数と、インテントが成功、失敗、ドロップ、切り替えに分類された回数と割合が表示されます。これらの定義の説明については「インテント」を参照してください。グラフのセグメントにカーソルを合わせて、その結果になった会話の数と割合を示すボックスを表示します (次の画像を参照)。



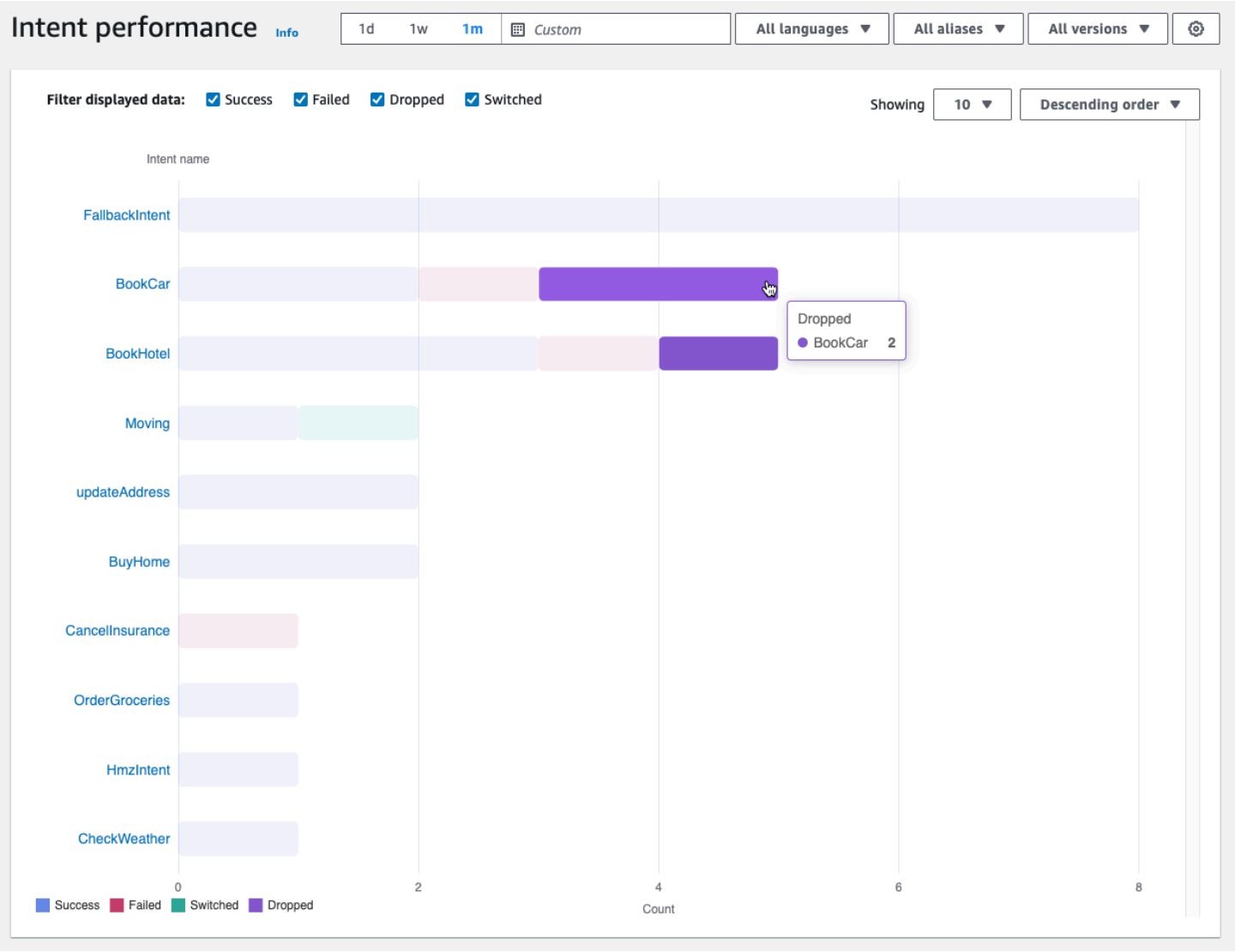
[すべてのインテントを表示] を選択するとドロップダウンメニューが表示され、そこからボットが誘発したインテントのリストを表示できます。特定の結果 (成功、失敗、ドロップ、切り替え) のインテントを表示するように選択することもできます。これらのリンクをクリックすると、[パフォーマンスダッシュボード] の [インテントパフォーマンス] サブセクションに移動します。詳細については、「インテントパフォーマンス」を参照してください。

[発話認識] セクションには、見逃された発話と検出された発話の数がまとめられています。[詳細を表示] を選択すると、ボットの発話リストに移動します。[発話見逃し] の数字を選択すると聞き逃した発話のリストが表示され、[検出された発話] の数字を選択すると、ボットの検出された発話のリストが表示されます。詳細については、「発話認識」を参照してください。

左側のサイドバーのパフォーマンスダッシュボードで [インテントパフォーマンス] と [発話認識] を選択すると、ボットのインテントと発話の詳細が表示されます。

インテントパフォーマンス

このダッシュボードには、ボットで使用されたインテントのパフォーマンスが、使用頻度の低い順にまとめられています。各インテントの横にあるバーには、インテントが成功、失敗、ドロップ、切り替えに分類された回数が視覚的に表示されます。これらの定義の説明については「インテント」を参照してください。以下の画像のように、バーの特定の部分にカーソルを合わせると、そのインテントを使用した会話とその結果の数が表示されます。



> **Note**
>
> ダッシュボードには、一連のフィルター設定の上位 1,000 件の結果が表示されます。より的を絞った結果を得るには、詳細なフィルター設定を行います。

グラフの上部にある [成功]、[失敗]、[ドロップ]、[切り替え] の各チェックボックスを使用して、表示するインテントステータスを切り替えることができます。

[表示] の右側にあるドロップダウンメニューを選択して、表示するインテントの数や、インテントを頻度の昇順または降順で表示するかどうかを調整します。

インテント名を選択すると、インテントパフォーマンスの内訳、スロットパフォーマンス、インテントスイッチの 3 つのグラフが表示されるページに移動します。

[インテントパフォーマンスの内訳] セクションには、ボットがインテントを使用した合計回数と、インテントフルフィルメントが成功、失敗、ドロップ、切り替えに分類された回数と割合が表示されます。これらの定義の説明については「インテント」を参照してください。グラフのセグメントにカーソルを合わせると、そのインテントフルフィルメントによってその結果が得られた回数と割合が表示されます。

[スロットパフォーマンスセクション] には、現在のインテントに属するスロットのメトリクスが表示されます。列を基準にソートするには、その列を 1 回選択して昇順にソートし、2 回選択して降順にソートします。検索バーを使用して特定のスロットを検索したり、ページ番号ボタンを使用してスロット間を移動したりできます。

> **Note**
>
> ダッシュボードには、一連のフィルター設定の上位 1,000 件の結果が表示されます。より的を絞った結果を得るには、詳細なフィルター設定を行います。

[インテントスイッチ] セクションには、ボットが現在のインテントから別のインテントに切り替えたインスタンスが、以下の情報とともに一覧表示されます。

- ステージ — ボットがインテントを切り替えた会話のステージ。
- インテントの切り替え先 — ボットが現在のインテントを切り替えたインテント。
- セッション数 — ステージとインテントの切り替え先の組み合わせが発生したセッションの数。

> **Note**
>
> ダッシュボードには、一連のフィルター設定の上位 1,000 件の結果が表示されます。より的を絞った結果を得るには、詳細なフィルター設定を行います。

発話認識

このページには、ボットが見逃したり検出したりしたすべての発話が一覧表示され、ボットのトレーニングに役立つサンプル発話をインテントに追加するためのツールも用意されています。これらの定義の説明については「発話」を参照してください。上部のタブを使用して、[発話見逃し] と [検出された発話] のリストを切り替えます。

> **Note**
>
> ダッシュボードには、一連のフィルター設定の上位 1,000 件の結果が表示されます。より的を絞った結果を得るには、詳細なフィルター設定を行います。

インテントに発話を追加するには:

1. インテントのサンプル発話として追加する発話の横にあるチェックボックスを選択します。
2. [インテントに追加] を選択し、[インテント] の下のドロップダウンメニューで発話を追加したいインテントを選択します。
3. [追加] を選択します。

分析に API を使用する

このセクションでは、ボットの分析情報を取得するために使用する API オペレーションについて説明します。

> **Note**
>
> ListUtteranceMetrics と ListUtteranceAnalyticsData を使用するには、発話関連の分析へのアクセスを提供する ListAggregatedUtterances オペレーションを実行する権限が IAM ロールに必要です。詳細と IAM ロールに適用する IAM ポリシーについては、「Lex V2 会話の発話統計の表示」を参照してください。

- 次の API オペレーションでは、ボットのサマリーメトリクスを取得します。
  - ListSessionMetrics
  - ListIntentMetrics

  - ListIntentStageMetrics
  - ListUtteranceMetrics
- 次の API オペレーションは、セッションと発話をするためのメタデータのリストを取得します。
  - ListSessionAnalyticsData
  - ListUtteranceAnalyticsData
- ListIntentPaths オペレーションは、顧客がボットと会話する際に取るインテントの順序に関するメトリクスを取得します。

結果のフィルタリング

Analytics API リクエストでは、`startTime` と `endTime` を指定する必要があります。API は、`startTime` の後に始まり、`endTime` の前に終了したセッション、インテント、インテントステージ、または発話を返します。

`filters` は、Analytics API リクエストのオプションフィールドです。AnalyticsSessionFilter、AnalyticsIntentFilter、AnalyticsIntentStageFilter、または AnalyticsUtteranceFilter オブジェクトのリストにマップされます。各オブジェクトで、フィールドを使用してフィルター条件となる式を作成します。たとえば、次のフィルターをリストに追加すると、ボットは 30 秒を超える会話を検索します。

```
{
    "name": "Duration",
    "operator": "GT",
    "value": "30 sec",
}
```

ボットのメトリクスを取得する

`ListSessionMetrics`、`ListIntentMetrics`、`ListIntentStageMetrics` および `ListUtteranceMetrics` オペレーションを使用して、セッション、インテント、インテントステージ、発話のサマリーメトリクスを取得します。

これらのオペレーションでは、以下の必須フィールドに入力してください。

- `startTime` と `endTime` を入力して、結果を取得する時間範囲を定義します。
- `metrics` で計算するメトリクス、AnalyticsSessionMetric、AnalyticsIntentMetric、AnalyticsIntentStageMetric、または

AnalyticsUtteranceMetric オブジェクトのリストを指定します。各オブジェクトで、name フィールドを使用して、Sum、Average、Max 数値を計算するかどうかを指定する statistic フィールドと、Ascending と Descending 順序のどちらで結果をソートするかを指定する order フィールドを指定します。

> **Note**
>
> metrics と binBy オブジェクトの両方に、order フィールドがあります。order でのソートを指定できるのは、2 つのオブジェクトのうちの 1 つのみです。

リクエストの残りのフィールドはオプションです。次の方法で検索結果をフィルタリングし、整理できます。

- 結果のフィルタリング — filters フィールドを使用して結果をフィルタリングします。詳細については、「結果のフィルタリング」を参照してください。
- 結果をカテゴリ別にグループ化 — groupBy フィールド、単一の AnalyticsSessionResult、AnalyticsIntentResult、AnalyticsIntentStageResult、または AnalyticsUtteranceResult オブジェクトを含むリストを指定します。オブジェクトで、結果をグループ化する基準となるカテゴリの name フィールドを指定します。

  リクエストで groupBy フィールドを指定すると、レスポンスの results オブジェクトには、groupByKeys、AnalyticsSessionGroupByKey、AnalyticsIntentGroupByKey、AnalyticsIntentStageGr オブジェクトのリストが含まれます。各オブジェクトには、リクエストで指定した name、および value フィールドにはそのカテゴリのメンバーが含まれます。
- 結果を時間ごとにビニングする — binBy フィールド、単一の AnalyticsBinBySpecification オブジェクトを含むリストを指定します。オブジェクトで、会話が開始した時点までに結果をビニングするには name フィールドを ConversationStartTime で指定し、発話が行われた日までに結果をビニングするには UtteranceTimestamp で指定します。結果を interval フィールドにビニングする時間間隔と、order フィールド内で時間をAscending と Descending のどちらでソートするかを指定します。

  リクエストで binBy フィールドを指定すると、レスポンス内の results オブジェクトには AnalyticsBinKey オブジェクトのリスト binKeys が含まれ、各オブジェクトには、リクエストで指定した name と、value フィールドでそのビンを定義する時間間隔が含まれます。

> **Note**
>
> metrics と binBy オブジェクトの両方に、order フィールドがあります。order でのソートを指定できるのは、2 つのオブジェクトのうちの 1 つのみです。

以下のフィールドを使用してレスポンスの表示を処理します。

- 1 つのレスポンスで返される結果の数を制限するには、maxResults フィールドに 1～1,000 の数値を指定します。
- 結果の数が maxResults フィールドで指定した数よりも多い場合、レスポンスには nextToken が含まれます。リクエストを再度行い、この値を nextToken フィールドに入力して次の結果のバッチを返します。

ListUtteranceMetricsを使用している場合は、attributes フィールドで返す属性を指定できます。このフィールドは、1 つの AnalyticSutteranceAttribute オブジェクトを含むリストにマップされます。発話時に Amazon Lex V2 が使用していたインテントを返すように LastUsedIntent フィールドで name を指定します。

レスポンスでは、results フィールドが AnalyticsSessionResult、AnalyticsIntentResult、AnalyticsIntentStageResult、または AnalyticsUtteranceResult オブジェクトのリストにマップされます。各オブジェクトには、指定したメソッドから作成されたビンまたはグループに加えて、要求したメトリクスの要約統計の値を返す metrics フィールドが含まれています。

ボットのセッションと発話のメタデータを取得する

ListSessionAnalyticsData オペレーションと ListUtteranceAnalyticsData オペレーションを使用して、個々のセッションと発話に関するメタデータを取得します。

必須の startTime と endTime フィールドを入力して、結果を取得する時間範囲を定義します。

リクエストの残りのフィールドはオプションです。結果をフィルタリングしてソートするには:

- 結果のフィルタリング — filters フィールドを使用して結果をフィルタリングします。詳細については、「結果のフィルタリング」を参照してください。

- 結果のソート — SessionDataSortBy オブジェクトまたは UtteranceDataSortBy オブジェクトを含む sortBy フィールドを使用して結果をソートします。name フィールド内で並べ替える値と、order フィールドで Descending と Ascending 順のどちらでソートするかを指定します。

以下のフィールドを使用してレスポンスの表示を処理します。

- 1 つのレスポンスで返される結果の数を制限するには、maxResults フィールドに 1 ~ 1,000 の数値を指定します。
- 結果の数が maxResults フィールドで指定した数よりも多い場合、レスポンスには nextToken が含まれます。リクエストを再度行い、この値を nextToken フィールドに入力して次の結果のバッチを返します。

レスポンスで、sessions または utterances フィールドが SessionSpecification オブジェクトまたは UtteranceSpecification オブジェクトのリストにマップされます。各オブジェクトには、1 つのセッションまたは発話のメタデータが含まれています。

インテントパス分析データの取得

ListIntentPaths オペレーションを使用して、顧客がボットと会話する際に取るインテントの順序に関するメトリクスを取得します。

このオペレーションでは、以下の必須フィールドに入力してください。

- startTime と endTime を入力して、結果を取得する時間範囲を定義します。
- メトリクスを取得するインテントの順序を定義するには、intentPath を指定します。パス内のインテントはフォワードスラッシュで区切ります。たとえば、intentPath フィールドに **/BookCar/BookHotel** を入力すると、ユーザーが BookCar と BookHotel インテントをその順序で呼び出した回数の詳細が表示されます。

オプションの filters フィールドを使用して、結果をフィルタリングします。詳細については、結果のフィルタリングを参照してください。

Lex V2 会話の発話統計の表示

発話統計を使用して、ユーザーがボットに送信している発話を特定できます。Amazon Lex V2 が正常に検出した発話と、検出されなかった発話の両方を確認できます。この情報を使用して、ボットをチューニングできるようになります。

例えば、Amazon Lex V2 が見つからないという発話をユーザーが送信している場合、その発話をインテントに追加できます。インテントのドラフトバージョンは新しい発話で更新され、ボットにデプロイする前にテストできます。

Amazon Lex V2 が、ボット用に設定されたインテントを呼び出そうとしていると認識した場合、発話が検出されます。Amazon Lex V2 が発話を認識せず、代わりに AMAZON.FallbackIntent を呼び出すと、発話が失われます。

発話統計は ListUtteranceMetrics API と ListAggregatedUtterance API を使用して表示できます。

発話の統計は、以下の条件で ListUtteranceMetrics API を使用して生成されません。

- コンソールでボットを作成した際、児童オンラインプライバシー保護法の設定が [はい] と設定されていた、または childDirected フィールドが true に設定されていた場合は、CreateBot オペレーションでボットを作成した時です。

ListUtteranceMetrics API には次のような追加機能もあります。

- 検出された発話のインテントのマッピングなど、より多くの情報を入手できます。
- より多くのフィルタリング機能 (チャネルとモードを含む)。
- 保持日付範囲が長い (30 日間)。
- データストレージをオプトアウトしていても API を使用できます。聞き逃した発話や検出された発話のコンソール機能は ListUtteranceMetrics API に依存します。

発話の統計は、以下の条件で ListAggregatedUtterance API を使用して生成されません。

- コンソールでボットを作成した際、児童オンラインプライバシー保護法の設定が [はい] と設定されていた、または childDirected フィールドが true に設定されていた場合は、CreateBot オペレーションでボットを作成した時です。
- 1 つ以上のスロットでスロットの難読化を使用しています。
- Amazon Lex の改善への参加をオプトアウトしました。

ListAggregatedUtterance API には次のような機能もあります。

- 詳細情報を減らします (発話のインテントのマッピングなし)。
- 限定されたフィルタリング機能 (チャネルとモードを含まない)。

- 保持日付範囲が短い (15 日間)。

発話統計を使用すると、特定の発話が検出された、または見逃されたか、ボットインタラクションで発話が最後に使用されたかを確認できます。

Amazon Lex V2 は、ユーザーがボットと対話している間、発話を継続的に保存します。コンソールまたは、ListAggregatedUtterances オペレーションを使用して統計をクエリできます。データ保持期間は 15 日間で、ユーザーがデータストレージをオプトアウトした場合は使用できません。DeleteUtterances オペレーションまたはデータストレージのオプトアウトにより、発話を削除することができます。AWS アカウントを閉じると、すべての発話が削除されます。保存された発話は、サーバーマネージドキーで暗号化されます。

ボットバージョンを削除すると、そのバージョンの発話統計は ListUtteranceMetrics の場合は最大 30 日間、ListAggregatedUtterances を使用する場合は 15 日間使用できます。Amazon Lex V2 コンソールでは、削除されたバージョンの統計を表示できません。削除されたバージョンの統計を表示するには、ListAggregatedUtterances と ListUtteranceMetrics オペレーション両方を使用します。

ListAggregatedUtterances と ListUtteranceMetrics API の両方で、発話は発話のテキストによって集計されます。例えば、顧客が「ピザを注文したい」というフレーズを使用したすべてのインスタンスは、応答の同じ行に集約されます。RecognizeUtterance オペレーションを使用するとき、使用されるテキストは入力文字起こしです。

ListAggregatedUtterances と ListUtteranceMetrics API を使用するには、以下のポリシーをロールに適用してください。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ListAggregatedUtterancesPolicy",
            "Effect": "Allow",
            "Action": "lex:ListAggregatedUtterances",
            "Resource": "*"
        }
    ]
}
```

分析のアクセス許可の管理

分析へのユーザーアクセスを提供するには、そのロールが分析用の API オペレーションを呼び出すことを許可するポリシーを IAM ロールにアタッチします。AWS マネージドポリシー: AmazonLexFullAccess を IAM ロールにアタッチして Amazon Lex API オペレーションへのフルアクセスを提供することも、分析へのアクセス権限のみを許可するカスタムポリシーを作成して IAM ロールにアタッチすることもできます。

分析用のアクセス許可を含むカスタムポリシーを作成するには

1. まず IAM ロールを作成する必要がある場合は、「IAM ユーザーにアクセス許可を委任するロールの作成」で説明されているステップに従います。
2. 「IAM ポリシーの作成」のステップに従い、次の JSON オブジェクトを使用してポリシーを作成します。IAM ロールの特定のボットへの分析アクセスを有効にするには、各ボットの ARN を `Resource` フィールドに追加します。*`region`*、*`account-id`*、*`BOTID`* をボットに対応する値に置き換えます。また、ステートメントの識別子 *`AnalyticsActions`* を、選択した名前で置き換えることができます。
3. 「IAM アイデンティティアクセス許可の追加と削除」の手順に従って、分析アクセス許可を付与するロールに作成したポリシーをアタッチします。
4. これで、指定したボットの分析を閲覧するアクセス許可がロールに付与されたはずです。

Lex V2 ボットの会話ログの有効化

会話ログを使用して、ボットとのユーザー会話を保存します。これらのログを確認して、ボットとユーザーとのやり取りに関する問題を特定し、その情報を基にボットの行動を修正してください。このセクションでは、ユーザーのプライバシーを保護するためにスロット値を難読化する方法についても説明します。

トピック

- Lex V2 の会話ログによる会話のログ記録
- Lex V2 の会話ログでスロット値を隠す
- Lex V2 の会話ログの選択的なキャプチャ

Lex V2 の会話ログによる会話のログ記録

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。RecognizeText オペレーションのテキストをログに記録できます。RecognizeUtterance 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文字起こしを取得できます。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 CreateBotAlias または UpdateBotAlias オペレーションを使用します。エイリアスの会話ログを有効にすると、RecognizeText または RecognizeUtterance オペレーションを使用して、そのエイリアスが設定された 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
- Lex V2 ボットの会話ログの設定
- Lex V2 のテキストログの Amazon CloudWatch Logs での表示
- 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 V2 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。Amazon Lex V2 がこれらのリソースに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。 Amazon Lex

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される 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 コンソール、IAM API、または を使用して AWS Command Line Interface 、ロールとポリシーを作成できます。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。

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 V2 を追加します。これにより、Amazon Lex V2 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。

   JSON

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

```
        "Service": "lexv2.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 V2 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。
2. AWS CLI で、CloudWatch Logs ロググループに書き込みアクセス許可を付与する 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 V2 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。

   JSON

```
{
  "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許可を付与する 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 V2 に渡すアクセス許可が必要です。ユーザーが Amazon Lex V2 サービスにロールを渡すには、IAM 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 AWS Identity and Access Management ユーザーガイドの「AWS サービスにロールを渡すアクセス許可をユーザーに付与する」および「IAM および AWS STS 条件コンテキストキー」を参照してください。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::111122223333:role/role-name",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "lexv2.amazonaws.com"
                },
                "StringLike": {
                    "iam:AssociatedResourceARN":
 "arn:aws:lex:region:123456789012:bot:bot-name:bot-alias"
                }
            }
        }
    ]
}
```

Lex V2 ボットの会話ログの設定

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 conversationLogSettings フィールド がある CreateBotAlias または UpdateBotAlias オペレーションを使用します。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 アカウントの 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 V2 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 ユーザーガイドのロググループとログストリーミングの操作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 V2 ボットと同じリージョンにあることが必要です。S3 バケットの作成の詳細については、「Amazon Simple Storage Service スタートガイド」の「バケットの作成」を参照してください。

コンソールを使用して会話ログを管理する場合、コンソールはサービスロールを更新して、ロググループと S3 バケットにアクセスできるようにします。

コンソールを使用しない場合、Amazon Lex V2 が設定済みのロググループまたはバケットの書き込みを有効にするポリシーのある IAM ロールを提供する必要があります。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 AWS Command Line Interface、次の例のように、 `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。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 iam create-service-linked-role \
    --aws-service-name lexv2.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。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::111122223333: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. [https://console.aws.amazon.com/lexv2/] で、Amazon Lex V2 コンソールを開いてください。
2. リストからボットを選択します。
3. 左側のメニューで、[エイリアス] 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を選択します。
5. [会話ログ] セクションで、[会話ログを管理する]を選択します。
6. テキストログの場合は、[有効化] 選択し、次に、Amazon CloudWatch Logs のロググループ名を入力します。
7. オーディオログの場合は、[有効化]を選択し、次に、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. [Save] (保存) を選択して、会話のログ記録を開始します。必要に応じて、Amazon Lex V2 は CloudWatch Logs ロググループおよび選択した S3 バケットにアクセスするためのアクセス許可でサービスロールを更新します。

Lex V2 の会話ログの無効化

コンソールを使用してログをオフにするには

1. [https://console.aws.amazon.com/lexv2/] で、Amazon Lex V2 コンソールを開いてください。
2. リストからボットを選択します。
3. 左側のメニューで、[エイリアス] 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を選択します。
5. [会話ログ] セクションで、[会話ログを管理する]を選択します。
6. ログ記録をオフにするには、テキストロギング、オーディオロギング、またはその両方を無効にします。
7. 会話のログ記録を停止するには、[Save] (保存) を選択します。

Lex V2 のテキストログの Amazon CloudWatch Logs での表示

Amazon Lex V2 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 V2 コンソールを使用してログを表示するには

1. [https://console.aws.amazon.com/lexv2/] で、Amazon Lex V2 コンソールを開いてください。
2. リストからボットを選択します。
3. 左側のメニューから、[分析] を選択し、[CloudWatch メトリクス] を選択します。
4. 「CloudWatch メトリクス」ページで、ボットのメトリクスを表示します。

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 V2 コンソールでログのログストリーミングプレフィックスを見つけるか、DescribeBotAlias オペレーションを使用できます。

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。

message-version

メッセージスキーマバージョン。

ボット

顧客がやり取りしているボットに関する詳細。

messages

ボットがユーザーに返送した応答。

utteranceContext

この発話の処理に関する情報。

- `runtimeHints`—ユーザーの入力を文字起こしして解釈するために使用されるランタイムコンテキスト。詳細については、「会話内のランタイムヒントによるスロット値の認識の向上」を参照してください。
- `slotElicitationStyle` —ユーザー入力を解釈するために使用されるスロット誘発スタイル。詳細については、「会話中のスペルスタイルによるスロット値のキャプチャ」を参照してください。

sessionState

ユーザーとボットの間の会話の現在の状態。詳細については、「Amazon Lex V2 ボットの会話について」を参照してください。

解釈

Amazon Lex V2 が判断したインテントのリストが、ユーザーの発話を満たしている可能性があります。信頼度スコアを使用した会話精度の向上。

interpretationSource

スロットが Amazon Lex V2 または Amazon Bedrock によって解決されるかどうかを示します。値: Lex | Bedrock

sessionId

会話が行われているユーザーセッションの識別子。

inputTranscript

ユーザーからの入力の文字起こし。

- テキスト入力の場合、これはユーザーが入力したテキストです。DTMF 入力の場合、これはユーザーが入力するキーです。
- 音声入力の場合は、インテントを呼び出すか、スロットを埋めるために、Amazon Lex V2 がユーザーの発話を変換したテキストです。

rawInputTranscript

テキスト処理が適用される前のユーザー入力の未加工の文字起こし。注: テキスト処理は en-US および en-GB ロケールのみを対象としています。

文字起こし

ユーザー入力の書き起こし候補のリスト。詳細については、「音声文字起こしの信頼度スコアを使用した Lex V2 ボットとの会話の改善」を参照してください。

rawTranscription

音声文字起こしの信頼度スコアを使用する。詳細については、「音声文字起こしの信頼度スコアを使用した Lex V2 ボットとの会話の改善」を参照してください。

missedUtterance

Amazon Lex V2 がユーザーの発話を認識できたかどうかを示します。

requestId

Amazon Lex V2 はユーザー入力のリクエスト ID を生成しました。

timestamp

ユーザー入力のタイムスタンプ。

developerOverride

会話フローがダイアログコードフックを使用して更新されたかどうかを示します。ダイアログコードフックの使用方法の詳細については、AWS Lambda関数を Amazon Lex V2 ボットに統合する を参照してください。

inputMode

入力のタイプを示します。オーディオ、DTMF、テキストのいずれでもかまいません。

requestAttributes

ユーザーの入力を処理するときに使用されるリクエスト属性。

audioProperties

音声会話ログが有効で、ユーザー入力が音声形式だった場合、音声入力の合計時間、音声の再生時間、音声の無音時間が含まれます。また、オーディオファイルへのリンクも含まれています。

bargeIn

ユーザー入力によって前回のボットレスポンスが中断されたかどうかを示します。

responseReason

レスポンスが生成された理由。次のいずれかの値を指定できます。

- `UtteranceResponse` — ユーザー入力へのレスポンス
- `StartTimeout` — ユーザーが入力しなかったときにサーバーが生成するレスポンス
- `StillWaitingResponse` — ユーザーがボットに待機をリクエストしたときにサーバーが生成するレスポンス
- `FulfillmentInitiated` — フルフィルメントが開始されようとしているというサーバーが生成するレスポンス
- `FulfillmentStartedResponse` — フルフィルメントが開始されたというサーバーが生成するレスポンス
- `FulfillmentUpdateResponse` — フルフィルメントの進行中にサーバーが生成する定期的なレスポンス

- FulfillmentCompletedResponse — フルフィルメントが完了したときにサーバーが生成するレスポンス。

operationName

ボットと対話するのに使用される API。これは、PutSession、RecognizeText、RecognizeUtterance、または StartConversation のいずれかになります。

```
{
    "message-version": "2.0",
    "bot": {
        "id": "string",
        "name": "string",
        "aliasId": "string",
        "aliasName": "string",
        "localeId": "string",
        "version": "string"
    },
    "messages": [
        {
            "contentType": "PlainText | SSML | CustomPayload | ImageResponseCard",
            "content": "string",
            "imageResponseCard": {
                "title": "string",
                "subtitle": "string",
                "imageUrl": "string",
                "buttonsList": [
                    {
                        "text": "string",
                        "value": "string"
                    }
                ]
            }
        }
    ],
    "utteranceContext": {
        "activeRuntimeHints": {
            "slotHints": {
                "string": {
                    "string": {
                        "runtimeHintValues": [
```

```
                            {
                                "phrase": "string"
                            },
                            {
                                "phrase": "string"
                            }
                        ]
                    }
                }
            }
        },
        "slotElicitationStyle": "string"
    },
    "sessionState": {
        "dialogAction": {
            "type": "Close | ConfirmIntent | Delegate | ElicitIntent | ElicitSlot",
            "slotToElicit": "string"
        },
        "intent": {
            "name": "string",
            "slots": {
                "string": {
                    "value": {
                       "interpretedValue": "string",
                       "originalValue": "string",
                       "resolvedValues": [ "string" ]
                    }
                 },
                "string": {
                    "shape": "List",
                    "value": {
                        "originalValue": "string",
                        "interpretedValue": "string",
                        "resolvedValues": [ "string" ]
                    },
                    "values": [
                        {
                            "shape": "Scalar",
                            "value": {
                                "originalValue": "string",
                                "interpretedValue": "string",
                                "resolvedValues": [ "string" ]
                            }
                        },
```

```
                        {
                            "shape": "Scalar",
                            "value": {
                                "originalValue": "string",
                                "interpretedValue": "string",
                                "resolvedValues": [ "string" ]
                            }
                        }
                    ]
                }
            },
            "kendraResponse": {
                // Only present when intent is KendraSearchIntent. For details, see
                // https://docs.aws.amazon.com/kendra/latest/dg/
API_Query.html#API_Query_ResponseSyntax
                },
            "state": "InProgress | ReadyForFulfillment | Fulfilled | Failed",
            "confirmationState": "Confirmed | Denied | None"
        },
        "originatingRequestId": "string",
        "sessionAttributes": {
            "string": "string"
        },
        "runtimeHints": {
            "slotHints": {
                "string": {
                    "string": {
                        "runtimeHintValues": [
                            {
                                "phrase": "string"
                            },
                            {
                                "phrase": "string"
                            }
                        ]
                    }
                }
            }
        }
     },
   "dialogEventLogs": [
        {
   // only for conditional
     "conditionalEvaluationResult":[
```

```
      // all the branches until true

     {
     "conditionalBranchName": "string",
     "expressionString": "string",
     "evaluatedExpression": "string",
     "evaluationResult": "true | false"
     }
    ],
  "dialogCodeHookInvocationLabel": "string",
  "response": "string",
  "nextStep": {
        "dialogAction": {
            "type": "Close | ConfirmIntent | Delegate | ElicitIntent | ElicitSlot",
            "slotToElicit": "string"
        },
       "intent": {
                          "name": "string",
           "slots": {
               }
        }
       }
    ]
    "interpretations": [
        {
            "interpretationSource": "Bedrock | Lex",
            "nluConfidence": "string",
            "intent": {
                "name": "string",
                "slots": {
                    "string": {
                        "value": {
                            "originalValue": "string",
                            "interpretedValue": "string",
                            "resolvedValues": [ "string" ]
                        }
                    },
                    "string": {
                        "shape": "List",
                        "value": {
                            "interpretedValue": "string",
                            "originalValue": "string",
                            "resolvedValues": [ "string" ]
                        },
```

```
                        "values": [
                            {
                                "shape": "Scalar",
                                "value": {
                                    "interpretedValue": "string",
                                    "originalValue": "string",
                                    "resolvedValues": [ "string" ]
                                }
                            },
                            {
                                "shape": "Scalar",
                                "value": {
                                    "interpretedValue": "string",
                                    "originalValue": "string",
                                    "resolvedValues": [ "string" ]
                                }

                            }
                        ]
                    }
                },
                "kendraResponse": {
                    // Only present when intent is KendraSearchIntent. For details, see

                    // https://docs.aws.amazon.com/kendra/latest/dg/
API_Query.html#API_Query_ResponseSyntax
                    },
                "state": "InProgress | ReadyForFulfillment | Fulfilled | Failed",
                "confirmationState": "Confirmed | Denied | None"
                },
            "sentimentResponse": {
                "sentiment": "string",
                "sentimentScore": {
                    "positive": "string",
                    "negative": "string",
                    "neutral": "string",
                    "mixed": "string"
                }
            }
        }
    ],
    "sessionId": "string",
    "inputTranscript": "string",
    "rawInputTranscript": "string",
```

```
    "transcriptions": [
        {
            "transcription": "string",
            "rawTranscription": "string",
            "transcriptionConfidence": "number",
            },
            "resolvedContext": {
                "intent": "string"
            },
            "resolvedSlots": {
                "string": {
                    "name": "slotName",
                    "shape": "List",
                    "value": {
                        "originalValue": "string",
                        "resolvedValues": [
                            "string"
                        ]
                    }
                }
            }
        }
    ],
    "missedUtterance": "bool",
    "requestId": "string",
    "timestamp": "string",
    "developerOverride": "bool",
    "inputMode": "DTMF | Speech | Text",
    "requestAttributes": {
        "string": "string"
    },
    "audioProperties": {
        "contentType": "string",
        "s3Path": "string",
        "duration": {
            "total": "integer",
            "voice": "integer",
            "silence": "integer"
        }
    },
    "bargeIn": "string",
    "responseReason": "string",
    "operationName": "string"
```

```
}
```

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 S3 のオーディオログへのアクセス

Amazon Lex V2 は、会話のオーディオログを S3 バケットに保存します。

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 V2 コンソールまたは conversationLogSettings フィールドがある DescribeBotAlias オペレーションレスポンスに表示されます。

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 V2 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

成功のメトリクスは、Amazon Lex V2 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 V2 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 V2 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 V2 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 V2 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。

Lex V2 の会話ログでスロット値を隠す

Amazon Lex V2 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、ログ記録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 V2 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:
    My name is John Stiles
After:
    My name is {full_name}
```

発話に括弧文字 ({}) が含まれている場合、Amazon Lex V2 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:
    My name is {John Stiles}
After:
    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、`RecognizeText` および `RecognizeUtterance` 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 V2 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 V2 は発話を難読化しません。

2 回目以降のターンでは、Amazon Lex V2 は誘発するスロットを認識し、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 V2 がスロット値を認識すると、値は難読化されます。Amazon Lex V2 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 V2 は、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 V2 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。

どのスロットを難読化するかは、コンソールまたは Amazon Lex V2 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、スロットの `obfuscationSetting` フィールドに `DEFAULT_OBFUSCATION` を設定します。これは、CreateSlot または UpdateSlot オペレーションを呼び出すときです。

Lex V2 の会話ログの選択的なキャプチャ

選択的会話ログキャプチャでは、ユーザーはライブ会話のテキストと音声データを使用して会話ログをキャプチャする方法を選択できます。

選択的会話ログキャプチャ機能を有効にして出力をキャプチャするには、Amazon Lex V2 コンソールで機能を有効にし、API 設定で必要なセッション属性を有効にして、選択した出力をログからキャプチャする必要があります。

選択的会話ログキャプチャには次のオプションを選択できます。

- テキストのみ
- オーディオ専用
- テキストとオーディオ

会話の特定の部分をキャプチャし、オーディオログに音声、テキスト、または両方をキャプチャするかどうかを選択できます。

> **Note**
>
> 選択的会話ログキャプチャは Amazon Lex V2 でのみ機能します。

トピック

- 選択的会話ログキャプチャを管理する
- 選択的会話ログキャプチャの例

選択的会話ログキャプチャを管理する

Lex コンソールを使用して、選択的会話ログキャプチャ設定を有効にし、選択的会話ログキャプチャを有効にするスロットを選択できます。

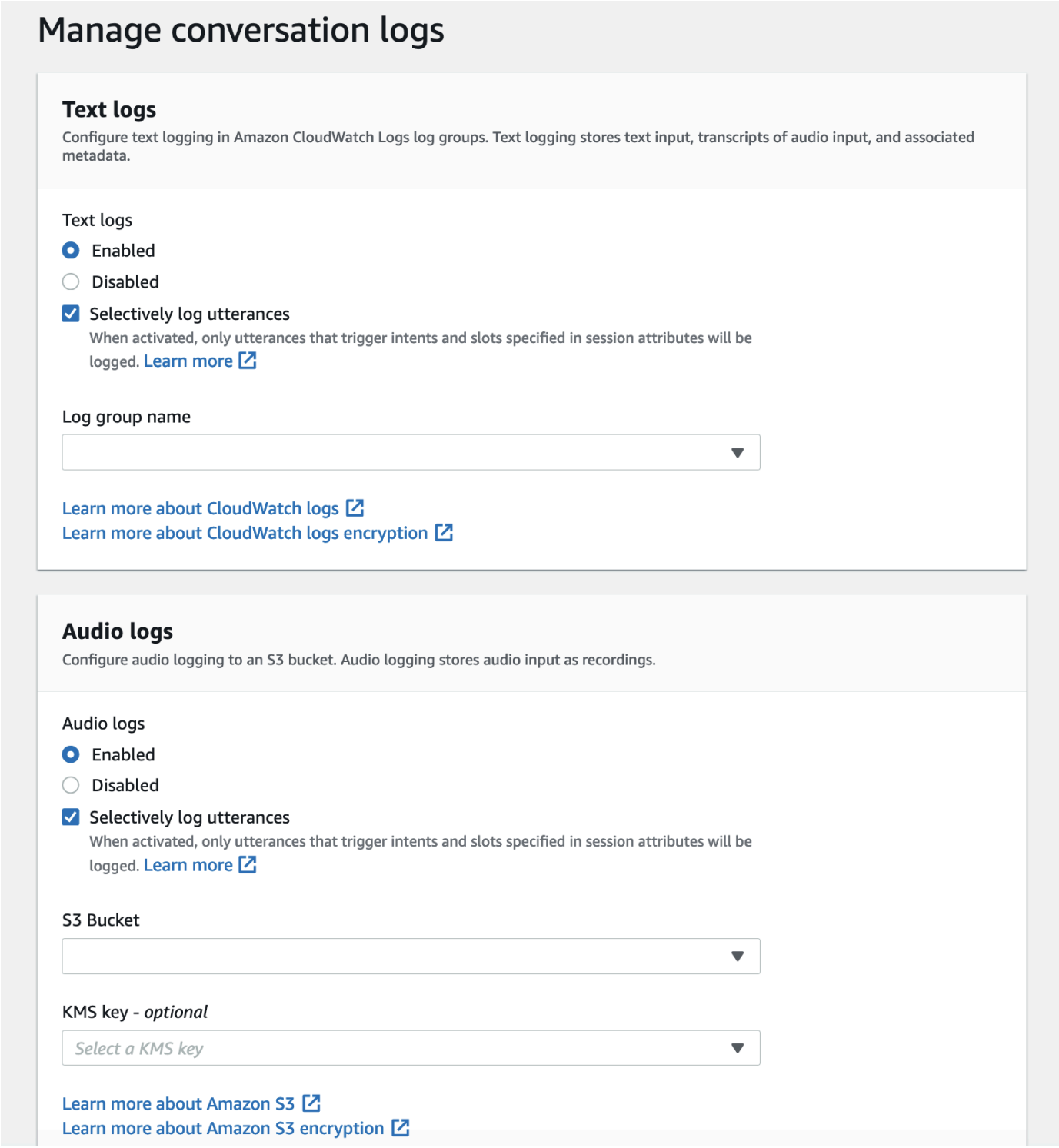
Amazon Lex V2 コンソールで選択的会話ログキャプチャを有効にします。

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/lexv2/home で Amazon Lex V2 コンソールを開きます。
2. 左側のパネルから [ボット] を選択し、選択的会話ログキャプチャを有効にするボットを選択します。既存のボットを選択するか、新しく作成します。

3. 左側のパネルの [デプロイ] セクションで、選択したボットの [エイリアス] を選択します。
4. ボットのエイリアスを選択し、[会話ログの管理] を選択します。
5. [会話ログの管理] パネルの [テキストログ] で、ラジオボタンを選択してテキストログを有効にするか無効にするかを選択します。テキストログで [有効] を選択した場合は、ロググループ名を入力するか、ドロップダウンメニューから既存のロググループ名を選択する必要があります。テキストファイルを選択的に記録する場合は、[発話を選択的に記録する] のチェックボックスを選択します。

   > **Note**
   >
   > テキストや音声のログを有効にするには、ビルド時の BotAlias 設定の会話ログ設定 (テキストおよび/または音声)で [発話を選択的にログに記録する] チェックボックスを選択します。このオプションを選択するには、CloudWatch ロググループと Amazon S3 バケットを設定する必要があります。

6. [オーディオログ] セクションで、ラジオボタンを選択してオーディオログを有効にするか無効にするかを選択します。オーディオログに [有効] を選択した場合は、Amazon S3 バケットの場所と (オプション) オーディオデータを暗号化するための KMS キーを指定する必要があります。オーディオファイルを選択的に記録する場合は、[発話を選択的に記録する] のチェックボックスを選択します。

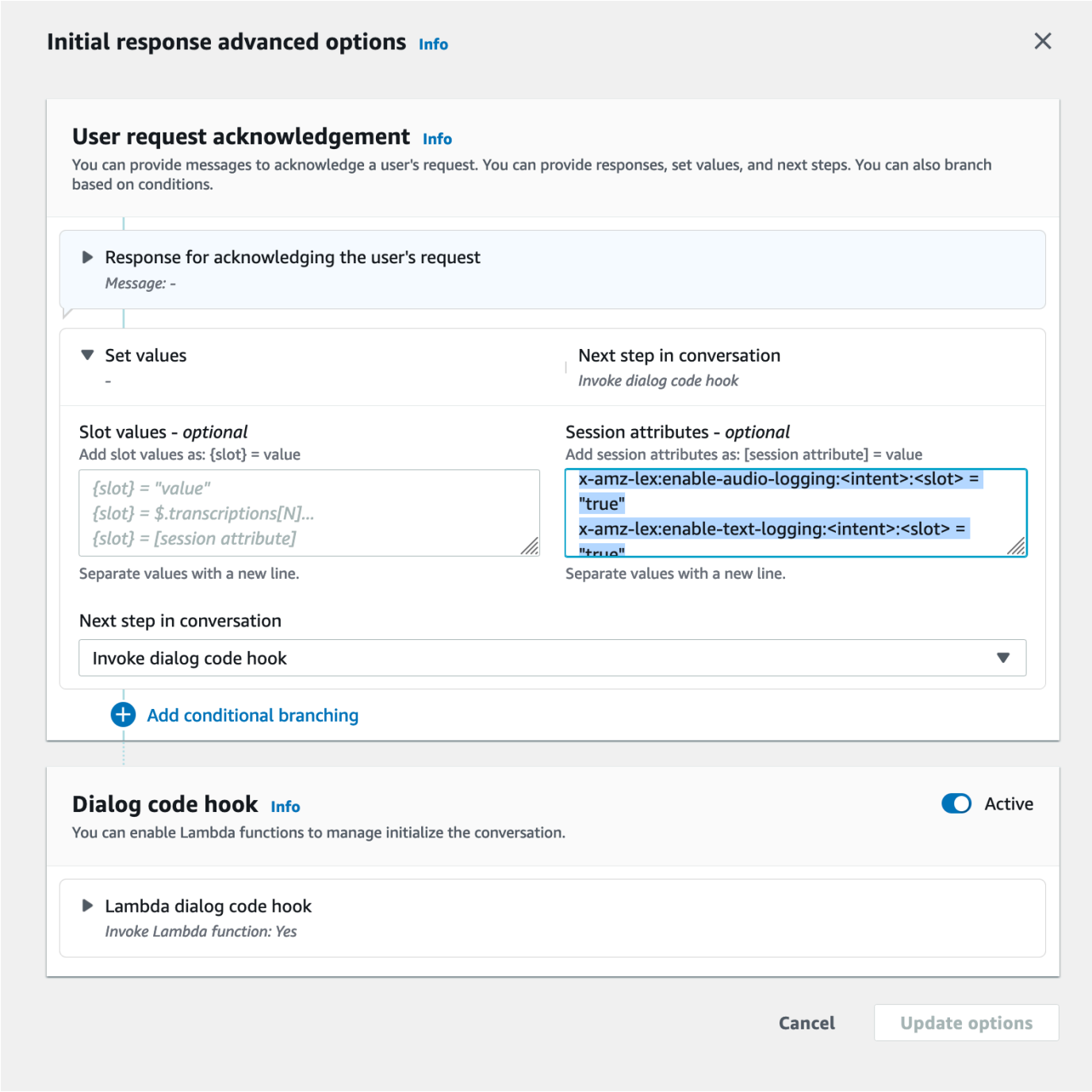


7. パネルの右下隅にある [保存] を選択して、選択した会話ログのキャプチャ設定を保存します。

Amazon Lex V2 コンソールで選択的会話ログキャプチャを有効にします。

1. [インテント] に移動し、[インテント名]、[初期応答]、[詳細設定]、[設定値]、[セッション属性] を選択します。
2. 選択的会話ログキャプチャを有効にするインテントとスロットに基づいて、次の属性を設定します。

   - `x-amz-lex:enable-audio-logging:intent:slot = "true"`
   - `x-amz-lex:enable-text-logging:intent:slot = "true"`



> **Note**
>
> 会話の特定のスロットのみを含む発話をキャプチャするように `x-amz-lex:enable-audio-logging:intent:slot = "true"` を設定します。発話を記録するアクションは、セッション属性表現、および対応するフラグ値と比較して、*#####*: 発話内の*###*

#の評価によって異なります。発話をログに記録するには、セッション属性内の少なくとも 1 つの式でその発話を許可し、ログ記録を有効化フラグを true に設定する必要があります。#####と####の値も同様に "*" にできます。スロット値やインテント値が "*" の場合、"*" のスロット値やインテント値のいずれかがその値と一致することを意味します。x-amz-lex:enable-audio-logging と同様に、x-amz-lex:enable-text-logging という新しいセッション属性がテキストログの制御に使用されます。

3. [更新オプション] を選択し、更新された設定を含むようにボットを構築します。

> **Note**
>
> IAM ロールには、Amazon S3 バケットにデータを書き込み、KMS キーを使用してデータを暗号化するためのアクセス許可が必要です。Lex は CloudWatch Logs ロググループおよび選択した Amazon S3 バケットにアクセスするための Lex アクセス許可で IAM ロールを更新します。

選択的会話ログキャプチャの使用に関するガイドライン:

[会話ログ設定] でテキストログやオーディオログを有効にしている場合にのみ、テキストログやオーディオログの選択的会話ログキャプチャを有効にできます。テキストログやオーディオログの選択的会話ログキャプチャを有効にすると、会話のすべてのインテントとスロットの記録が無効になります。特定のインテントやスロットのテキストまたはオーディオログを生成するには、それらのインテントとスロットのテキストまたはオーディオ選択型会話ログキャプチャセッション属性を「true」に設定する必要があります。

- 選択的会話ログキャプチャが有効になっていて、プレフィックスが x-amz-lex: enable-audio-logging のセッション属性が存在しない場合、すべての発話の記録はデフォルトで無効になります。このシナリオは x-amz-lex:enable-text-logging についても当てはまります。
- セッション属性の少なくとも 1 つの表現で許可されている場合、発話ログはテキストや音声会話のセグメントのみに保存されます。
- セッション属性で定義されているテキストや音声の選択的会話ログキャプチャの設定は、ボットエイリアスの会話ログ設定でテキストや音声の選択的会話ログキャプチャが有効になっている場合にのみ有効になります。有効になっていない場合、セッション属性は無視されます。

- 選択的会話ログキャプチャが有効になっている場合、セッション属性によるロギングが有効になっていない SessionState、Interpretations、Transcriptions のスロット値は、生成されたテキストログで難読化されます。
- オーディオログやテキストログを作成するかどうかの決定は、ボットが誘発したスロットを選択的な会話ログキャプチャセッション属性と照合することによって評価されます。ただし、インテント誘発ターンは例外で、ユーザーがインテント誘発とともにスロット値を入力できるインテント誘発ターンは例外です。インテント誘発ターンでは、現在のターンに埋まっているスロットが選択的会話ログキャプチャセッション属性と照合される。
- 空になったと見なされるスロットは、ターン終了時のセッション状態から算出されます。したがって、Dialog Codehook Lambda によってセッション状態のスロットに加えられた変更は、選択的会話ログキャプチャの動作に影響します。
- インテント誘発ターンで、ユーザーが複数のスロット値を指定した場合、テキスト/オーディオのログは、テキスト/オーディオのセッション属性でそのターンに埋まったすべてのスロットの記録が許可されている場合にのみ生成されます。
- 推奨される運用アプローチは、セッションの開始時に選択的会話ログキャプチャセッション属性を設定し、セッション中は変更しないことです。
- 機密データを含むスロットがある場合は、常にスロットの難読化を有効にする必要があります。

選択的会話ログキャプチャの例

選択的会話ログキャプチャのビジネスユースケースの例を次に示します。

ユースケース:

あるフィンテック企業は、Amazon Lex V2 ボットを利用して IVR システムをサポートし、ユーザーが請求書の支払いを実行できるようにしています。コンプライアンスと監査の要件を満たすためには、ユーザーが提供した承認同意の音声録音を保存する必要があります。ただし、一般的なオーディオログを有効にすると、CardNumber、CVV、その他の情報などの機密スロットをオーディオログ内の難読化できないため、非準拠になってしまい現実的ではありません。代わりに、オーディオログの会話ログを選択的にキャプチャできるようにし、承認同意を得た発話のオーディオログのみを生成するようにセッション属性を設定することができます。

BotAlias 設定:

- テキストログ有効: true
- テキストログ、選択的ロギング有効: false

- オーディオログ有効: true
- オーディオログ:選択的ロギング有効: true

セッション属性:

```
x-amz-lex:enable-audio-logging:PayBill:AuthorizationConsent = "true"
```

サンプル会話:

- ユーザー (オーディオ入力):「請求書番号 35XU68 で支払いをしたいのですが。」
- ボット:「支払うべき金額は何ドルですか?」
- ユーザー (オーディオ入力):「235」
- ボット:「クレジットカード番号は何ですか?」
- ユーザー (オーディオ入力):「9239829722200348」
- ボット:「末尾が0348のクレジットカード番号で、235 ドルをお支払いいただきます。『235 ドルの支払いを許可します』と言ってください。」
- ユーザー (オーディオ入力):「235 ドルの支払いを許可します。」
- ボット:「請求書の支払いが完了しました。」

会話ログ出力:

この場合、すべてのターンのテキストログが生成されます。ただし、PayBill インテント内の AuthorizationConsent スロットが誘発された特定のターンのオーディオログのみが録音され、それ以外のターンのオーディオログは生成されません。

Lex V2 のエラーログを使用したエラーのログ記録

エラーログを有効にして、ボットとのやりとりを保存します。これらのエラーログを使用して、ボットのパフォーマンスを確認し、会話に関するエラーのトラブルシューティングを行うことができます。

エラーログは 1 つのバージョン向けに設定されています。各バージョンで、エラーログの設定を別々にすることができます。テキストログは、テキスト入力を CloudWatch Logs に保存します。AWS KMS カスタマーマネージド CMK を使用して、テキストログの暗号化を有効にできます。

エラーログの IAM ポリシー

選択したログ記録のタイプに応じて、Amazon Lex V2 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。Amazon Lex V2 がこれらのリソースに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。 Amazon Lex

エラーログ用の 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される 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 コンソール、IAM API、または を使用して AWS Command Line Interface 、ロールとポリシーを作成できます。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。

エラーログの IAM ロールを作成するには

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。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::111122223333:role/role"
        }
    ]
}
```

Lex V2 のエラーログの有効化

Amazon Lex V2 コンソールを使用してエラーログをオンにするには

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/lex/ で Amazon Lex コンソールを開きます。

2. [ボット] のリストから、エラーログを有効にするボットを選択します。
3. 左側のメニューから、[バージョン] を選択します。
4. [バージョン] のリストで、エラーログを設定するバージョンを選択します。
5. [バージョンの詳細] セクションで、[有効化] を選択します。
6. [Save] (保存) を選択して、会話のログ記録を開始します。必要に応じて、Amazon Lex V2 は CloudWatch Logs ロググループにアクセスするためのアクセス許可でサービスロールを更新します。

Lex V2 のエラーログの無効化

Amazon Lex V2 コンソールを使用してエラーログをオフにするには

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/lex/ で Amazon Lex コンソールを開きます。
2. [ボット] のリストから、エラーログを有効にするボットを選択します。
3. 左側のメニューから、[バージョン] を選択します。
4. [バージョン] のリストで、エラーログを設定するバージョンを選択します。
5. [バージョンの詳細] セクションで、[無効化] を選択します。
6. 会話のログ記録を停止するには、[Save] (保存) を選択します。

Lex V2 でのオペレーションメトリクスのモニタリング

Amazon CloudWatch と AWS CloudTrail は、Amazon Lex V2 と統合してボットとのユーザーインタラクションをモニタリングするのに役立つ 2 つの AWS サービスです。これらのサービスを使用して、アクションを記録し、ほぼリアルタイムのデータを送信し、条件が満たされた場合の通知と自動アクションの設定を行います。

トピック

- Amazon CloudWatch によるオペレーションメトリクスの測定
- for Lex V2 AWS CloudTrail でのイベントの表示

Amazon CloudWatch によるオペレーションメトリクスの測定

Amazon CloudWatch を使用して Amazon Lex V2 をモニタリングすることで、raw データを収集し、ほぼリアルタイムで読み取り可能なメトリクスに加工することができます。これらの統計は 15 か月間保持されるため、履歴情報にアクセスし、ウェブアプリケーションまたはサービスの動作をより的確に把握できます。また、特定のしきい値を監視するアラームを設定し、これらのしきい値に達したときに通知を送信したりアクションを実行したりできます。詳細については、Amazon CloudWatch ユーザーガイド を参照してください。

Amazon Lex V2 サービスは、AWS/Lex 名前空間の以下のメトリクスをレポートします。

メトリクス	説明
AssistedSlotResolutionModelAccessDeniedErrorCount	Amazon Lex V2 が Amazon Bedrock へのアクセスを拒否された回数。 RecognizeUtterance または StartConversation オペレーションの有効なディメンション: • BotId、BotAliasId、LocaleId、Operation、InputMode、ModelType、Model • BotId、BotVersion、LocaleId、Operation、InputMode、ModelType、Model RecognizeText の有効なディメンション: • BotId、BotAliasId、LocaleId、Operation、ModelType、Model • BotId、BotVersion、LocaleId、Operation、ModelType、Model 単位: 数
AssistedSlotResolutionModelInvocationCount	Amazon Bedrock が呼び出された回数。 RecognizeUtterance または StartConversation オペレーションの有効なディメンション: • BotId、BotAliasId、LocaleId、Operation、InputMode、ModelType、Model

メトリクス	説明
	• BotId、BotVersion、LocaleId、Operation、InputMode、ModelType、Model RecognizeText の有効なディメンション: • BotId、BotAliasId、LocaleId、Operation、ModelType、Model • BotId、BotVersion、LocaleId、Operation、ModelType、Model 単位: 数
AssistedSlotResolutionModelSystemErrorCount	Amazon Bedrock を呼び出したときに 5xx が発生した回数。 RecognizeUtterance または StartConversation オペレーションの有効なディメンション: • BotId、BotAliasId、LocaleId、Operation、InputMode、ModelType、Model • BotId、BotVersion、LocaleId、Operation、InputMode、ModelType、Model RecognizeText の有効なディメンション: • BotId、BotAliasId、LocaleId、Operation、ModelType、Model • BotId、BotVersion、LocaleId、Operation、ModelType、Model 単位: 数

メトリクス	説明
AssistedSlotResolutionModelThrottlingErrorCount	Amazon Lex が Amazon Bedrock によってスロットリングされた回数。 RecognizeUtterance または StartConversation オペレーションの有効なディメンション: • BotId、BotAliasId、LocaleId、Operation、InputMode、ModelType、Model • BotId、BotVersion、LocaleId、Operation、InputMode、ModelType、Model RecognizeText の有効なディメンション: • BotId、BotAliasId、LocaleId、Operation、ModelType、Model • BotId、BotVersion、LocaleId、Operation、ModelType、Model 単位: 数
AssistedSlotResolutionResolvedSlotCount	Amazon Bedrock がスロット値を返した回数。 RecognizeUtterance または StartConversation オペレーションの有効なディメンション: • BotId、BotAliasId、LocaleId、Operation、InputMode、ModelType、Model • BotId、BotVersion、LocaleId、Operation、InputMode、ModelType、Model RecognizeText の有効なディメンション: • BotId、BotAliasId、LocaleId、Operation、ModelType、Model • BotId、BotVersion、LocaleId、Operation、ModelType、Model 単位: 数

メトリクス	説明
KendraIndexAccessError	Amazon Lex V2 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。 • オペレーション、BotId、BotAliasID、LocaleID 単位: 数
KendraLatency	AMAZON.KendraSearchIntent からのリクエストに Amazon Kendra が応答する所要時間。 有効なディメンション: • オペレーション、BotID、BotVersion、LocaleID • オペレーション、BotId、BotAliasID、LocaleID 単位: ミリ秒
KendraSuccess	Amazon Lex V2 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。 有効なディメンション: • オペレーション、BotID、BotVersion、LocaleID • オペレーション、BotId、BotAliasID、LocaleID 単位: 数
KendraSystemErrors	Amazon Lex V2 が Amazon Kendra インデックスをクエリできなかった回数。 有効なディメンション: • オペレーション、BotId、BotAliasID、InputMode、LocaleID 単位: 数

メトリクス	説明
KendraThrottledEvents	Amazon Kendra が AMAZON.KendraSearchIntent からの リクエストをスロットルした回数。 有効なディメンション: • オペレーション、BotId、BotAliasID、InputMode LocaleID 単位: 数
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。 RecognizeUtterance または StartConversation オペレーションの有効なディメンション: • オペレーション、BotID、BotVersion、InputMode、LocaleID • オペレーション、BotId、BotAliasID、InputMode、LocaleID 他のオペレーションの有効なディメンション: • オペレーション、BotID、BotVersion、LocaleID • オペレーション、BotId、BotAliasID、LocaleID 単位: 数
RuntimeInvalidLambdaResponses	指定された期間の無効な AWS Lambda レスポンスの数。 有効なディメンション: • オペレーション、BotId、BotAliasID、InputMode、LocaleID 単位: 数

メトリクス	説明
RuntimeLambdaErrors	指定期間中の Lambda ランタイムエラーの数。 有効なディメンション: • オペレーション、BotId、BotAliasID、InputMode、LocaleID 単位: 数
RuntimePollyErrors	指定期間中の無効な Amazon Polly レスポンスの数。 有効なディメンション: • オペレーション、BotId、BotAliasID、InputMode、LocaleID 単位: 数
RuntimeRequestCount	指定期間中のランタイムリクエストの数。 RecognizeUtterance または StartConversation オペレーションの有効なディメンション: • オペレーション、BotID、BotVersion、InputMode、LocaleID • オペレーション、BotId、BotAliasID、InputMode、LocaleID 他のオペレーションの有効なディメンション: • オペレーション、BotID、BotVersion、LocaleID • オペレーション、BotId、BotAliasID、LocaleID 単位: 数

メトリクス	説明
RuntimeRequestLength	Amazon Lex V2 ボットとの会話の合計の長さ。StartConversation オペレーションにのみ適用されます。 有効なディメンション: • BotAliasID、BotId、LocaleID、オペレーション • BotId、BotAliasID、LocaleID、オペレーション 単位: ミリ秒
RuntimeSuccessfulRequestLatency **Important** このメトリクスは RuntimeSuccessfulRequestLatency で、RuntimeSuccessfulRequestLatency ではありません。	リクエストが成功してから、レスポンスが返されるまでのレイテンシー。 RecognizeUtterance または StartConversation オペレーションの有効なディメンション: • オペレーション、BotID、BotVersion、InputMode、LocaleID • オペレーション、BotId、BotAliasID、InputMode、LocaleID 他のオペレーションの有効なディメンション: • オペレーション、BotID、BotVersion、LocaleID • オペレーション、BotId、BotAliasID、LocaleID 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。 RecognizeUtterance または StartConversation オペレーションの有効なディメンション: • オペレーション、BotID、BotVersion、InputMode、LocaleID • オペレーション、BotId、BotAliasID、InputMode、LocaleID 他のオペレーションの有効なディメンション: • オペレーション、BotID、BotVersion、LocaleID • オペレーション、BotId、BotAliasID、LocaleID 単位: 数
RuntimeThrottledEvents	スロットルされたイベントの数。Amazon Lex V2 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 のサービス制限」を参照してください。 RecognizeUtterance または StartConversation オペレーションの有効なディメンション: • オペレーション、BotID、BotVersion、InputMode、LocaleID • オペレーション、BotId、BotAliasID、InputMode、LocaleID 他のオペレーションの有効なディメンション: • オペレーション、BotID、BotVersion、LocaleID • オペレーション、BotId、BotAliasID、LocaleID 単位: 数

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。 RecognizeUtterance または StartConversation オペレーションの有効なディメンション: • オペレーション、BotID、BotVersion、InputMode、LocaleID • オペレーション、BotId、BotAliasID、InputMode、LocaleID 他のオペレーションの有効なディメンション: • オペレーション、BotID、BotVersion、LocaleID • オペレーション、BotId、BotAliasID、LocaleID 単位: 数

Amazon Lex V2 メトリクスでサポートされるディメンションは以下のとおりです。

ディメンション	説明
Operation	Amazon Lex V2 オペレーションの名前 — RecognizeText 、RecognizeUtterance 、StartConversation 、GetSession 、PutSession 、DeleteSession — エントリを生成しました。
BotId	ボットの一意の英数字識別子。
BotAliasId	ボットエイリアスの一意の英数字識別子。
BotVersion	ボットの数値バージョン。
InputMode	ボットへの入力のタイプ (スピーチ、テキスト、または DTMF)。
LocaleId	en-US や fr-CA など、ボットの口ケールの識別子。

ディメンション	説明
Model	Amazon Bedrock の大規模言語モデルのモデル ID を示します。
ModelType	Amazon Bedrock から呼び出す大規模言語モデルのタイプを示します。

for Lex V2 AWS CloudTrail でのイベントの表示

Amazon Lex V2 は AWS CloudTrail、Amazon Lex V2 のユーザー、ロール、または AWS のサービスによって実行されたアクションを記録するサービスである Amazon Lex と統合されています。CloudTrail は、Amazon Lex V2 の API コールをイベントとしてキャプチャします。キャプチャされた呼び出しには、Amazon Lex V2 コンソールからの呼び出しと、Amazon Lex V2 API オペレーションへのコード呼び出しが含まれます。証跡を作成する場合は、Amazon Lex V2 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 [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 V2 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。

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 のAmazon Lex V2 情報

CloudTrail は、アカウント作成時に AWS アカウントで有効になります。Amazon Lex V2 でアクティビティが発生すると、そのアクティビティは CloudTrail イベントとイベント履歴の他の AWS サービス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、「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 V2 のイベントなど、AWS アカウントのイベントの継続的な記録については、証跡を作成します。追跡により、CloudTrail はログファイルを 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 Amazon S3 バケットにログファイルが配信されます。さらに、その他の AWS サービスを設定して、CloudTrail ログで収集されたデータをより詳細に分析し、それに基づく対応を行うことができます。詳細については、以下を参照してください。

- 追跡を作成するための概要
- CloudTrail がサポートされているサービスと統合

- 「CloudTrail の Amazon SNS 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 V2 は、「モデル構築 API V2」に記載されているすべてのアクションのログ記録をサポートしています。

各イベントまたはログエントリには、誰がリクエストを生成したかという情報が含まれます。アイデンティティ情報は、以下を判別するのに役立ちます。

- リクエストがルートまたは AWS Identity and Access Management IAM ユーザー認証情報を使用して行われたかどうか。
- 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 AWS の別のサービスによって生成されたかどうか。

詳細については、「CloudTrail userIdentity エレメント」を参照してください。

Amazon Lex V2 ログファイルエントリの理解

「トレイル」は、指定した Amazon 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 のログファイルは、単一か複数のログエントリを含みます。イベントは、任意の出典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、リクエストパラメータなどに関する情報が含まれます。CloudTrail ログファイルは、パブリック API 呼び出し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。

次の例は、CreateBotAlias アクションを示す CloudTrail ログエントリです。

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "ID of caller:temporary credentials",
        "arn": "arn:aws:sts::111122223333:assumed-role/role name/role ARN",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
```

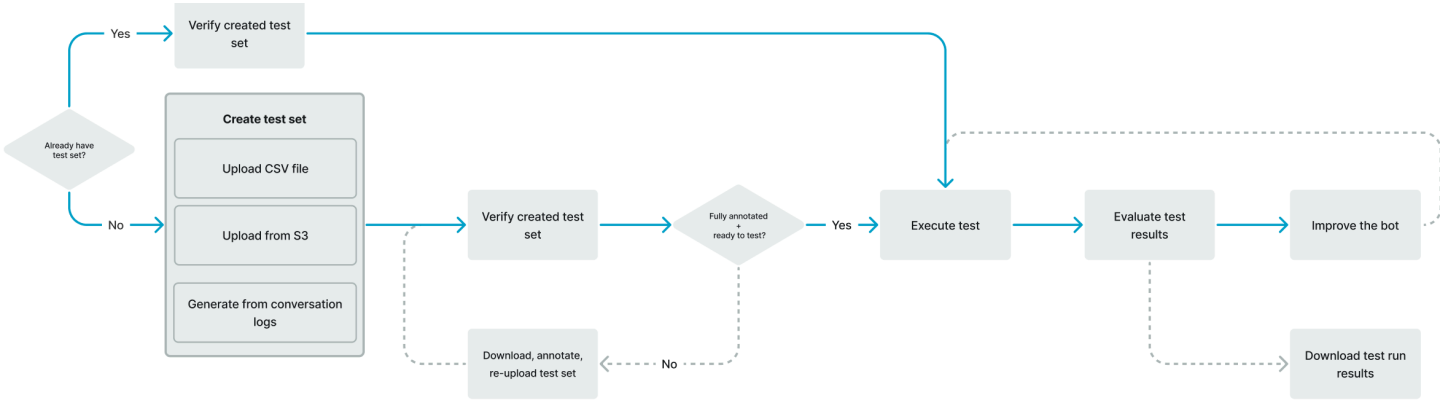
```
                "principalId": "ID of caller",
                "arn": "arn:aws:iam::111122223333:role/role name",
                "accountId": "111122223333",
                "userName": "role name"
            },
            "webIdFederationData": {},
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "creation date"
            }
        }
    },
    "eventTime": "event timestamp",
    "eventSource": "lex.amazonaws.com",
    "eventName": "CreateBotAlias",
    "awsRegion": "Region",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "user agent",
    "requestParameters": {
        "botAliasLocaleSettingsMap": {
            "en_US": {
                "enabled": true
            }
        },
        "botId": "bot ID",
        "botAliasName": "bot aliase name",
        "botVersion": "1"
    },
    "responseElements": {
        "botAliasLocaleSettingsMap": {
            "en_US": {
                "enabled": true
            }
        },
        "botAliasId": "bot alias ID",
        "botAliasName": "bot alias name",
        "botId": "bot ID",
        "botVersion": "1",
        "creationDateTime": creation timestamp
    },
    "requestID": "unique request ID",
    "eventID": "unique event ID",
    "readOnly": false,
    "eventType": "AwsApiCall",
```

```
    "recipientAccountId": "111122223333"
}
```

Test Workbench による Lex V2 ボットのパフォーマンス評価

ボットのパフォーマンスを向上させるには、ボットのパフォーマンスを大規模に評価できます。テスト評価の結果は、簡単な表とグラフに表示されます。

Test Workbench を使用して、既存の文字起こしデータを使用する参照テストセットを作成できます。導入前にボットをテストしてパフォーマンスを評価したり、テスト結果の内訳を大規模に表示したりできます。



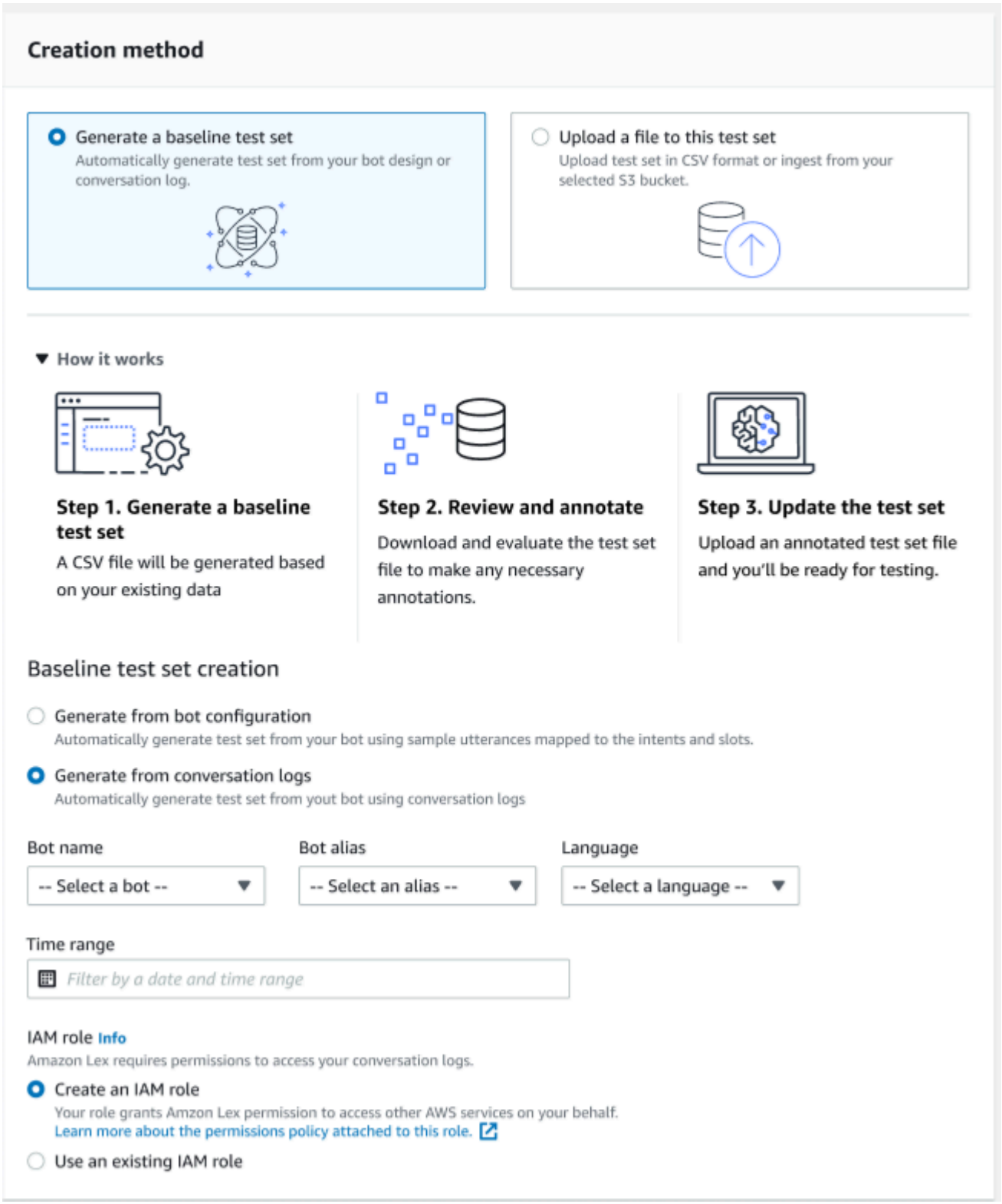
ユーザーは Test Workbench を使用してボットのベースラインパフォーマンスを設定できます。これには、単一入力または会話形式の発話のインテントとスロットパフォーマンスが対象となります。テストセットが正常に読み込まれたら、既存のプリプロダクションボットまたはプロダクションボットに対してテストセットを実行できます。Test Workbench は、スロット充填やインテントの分類を改善できる可能性を見極めるのに役立ちます。

トピック

- Test Workbench のテストセットの生成
- テストセットを管理する
- テストを実行する
- Test Workbench でのテストセットのカバレッジ
- テスト結果を表示する
- Test Workbench のテスト結果の詳細

Test Workbench のテストセットの生成

テストセットを作成して、ボットのパフォーマンスを評価できます。CSV ファイル形式のテストセットをアップロードするか、会話ログからテストセットを生成してテストセットを生成します。テストセットには音声またはテキスト入力を含めることができます。



テストセットで検証エラーが発生した場合は、テストセットを削除してテストセットデータの別のリストに置き換えるか、スプレッドシート編集プログラムを使用して CSV ファイル内のデータを編集します。

テストセットを作成するには:

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/lex/ で Amazon Lex コンソールを開きます。
2. 左側のパネルから [Test workbench] を選択します。
3. Test Workbench の下のオプションから [テストセット] を選択します。
4. コンソールの [テストセットを作成] ボタンを選択します。
5. [詳細] に、テストセット名と説明 (オプション) を入力します。
6. [ベースラインテストセットを生成する] を選択します。
7. [会話ログから生成する] を選択します。
8. ドロップダウンメニューから [ボット名]、[ボットエイリアス]、[言語] を選択します。
9. 会話ログからベーステストを生成する場合は、必要に応じて [時間範囲] と [IAM ロール] を選択します。基本 Amazon Lex V2 アクセス許可でロールを作成させるか、既存のロールを選択できます。
10. 作成するテストセットの [オーディオ] または [テキスト] のモダリティを選択します。注: Test Workbench では、最大 50k のテキストファイル、最大 5 時間のオーディオファイルをインポートできます。
11. テスト結果を保存する Amazon S3 の場所を選択し、オプションで KMS キーを追加して出力文字起こしを暗号化します。
12. [作成] を選択します。

既存のテストセットを CSV ファイル形式でアップロードするか、テストセットを更新するには:

1. 左側のパネルから [Test workbench] を選択します。
2. Test Workbench の下のオプションから [テストセット] を選択します。
3. コンソールで [このテストセットにファイルをアップロード] を選択します。
4. [Amazon S3 バケットからアップロード] または [コンピュータからアップロード] を選択します。注: テンプレートから作成した CSV ファイルをアップロードできます。[CSV テンプレート] をクリックして、テンプレートを含む ZIP ファイルをダウンロードします。
5. [Amazon Lex の基本的なアクセス許可を持つロールの作成] または [Role ARN の既存ロールの使用] を選択します。
6. 作成するテストセットの [オーディオ] または [テキスト] のモダリティを選択します。注: Test Workbench では、最大 50k のテキストファイル、最大 5 時間のオーディオファイルをインポートできます。

7. テスト結果を保存する Amazon S3 の場所を選択し、オプションで KMS キーを追加して出力文字起こしを暗号化します。
8. [作成] を選択します。

操作が成功すると、テストセットをテストする準備ができたことを示す確認メッセージが表示され、ステータスには [テスト準備完了] と表示されます。

テストセットを正常に作成するためのヒント

- コンソールで Test Workbench 用の IAM ロールを作成することも、IAM ロールを段階的に設定することもできます。詳細については、「Test Workbench 用の IAM ロールを作成する」を参照してください。
- テストを実行する前に、[差異の検証]ボタンを使用して、テストセットとボット定義に不一致がないか検証します。テストセットで使用されているインテントとスロットの命名規則がボットと一致している場合は、テストを実行してください。異常が見つかった場合は、テストセットを修正してテストセットを更新し、[差異の検証] を選択します。矛盾がなくなるまでこのシーケンスをもう一度繰り返し、テストを実行します。
- Test Workbench では、[予想される出力スロット] 列のさまざまなスロット値フォーマットを使用してテストできます。どのビルトインスロットでも、ユーザー入力に入力した値 (例:日付 = 明日) を選択することも、絶対解決値 (例:Date = 2023-03-21) を指定することもできます。ビルトインスロットとその絶対値について詳しくは、「組み込みのスロット」を参照してください。
- [予想される出力スロット] 列の一貫性と読みやすさを考慮して、等号の前後にスペースを入れる「SlotName = SlotValue」 (例: AppointmentType = クリーニング) の規則に従ってください。
- ボットに複合スロットが含まれる場合は、[予想される出力スロット] でスロット名のサブスロットをピリオドで区切って定義します (「Car.Color」など)。他の構文や句読点を使用することはできません。
- ボットに複数値スロットが含まれている場合は、[予想される出力スロット] に複数のスロット値をカンマで区切って入力します (「FlowerType = バラ、ユリ」)。他の構文や句読点を使用することはできません。
- テストセットは必ず有効な会話ログから作成してください。
- スロット: スロットの値は、CSV 形式のインテント列の後の同じ列に表示されます。
- ユーザーターンからの DTMF 入力は、想定どおりの文字起こしとして解釈され、Amazon S3 の場所は表示されません。

Test Workbench を使用した、テストセット内でのテストケースの作成

Test Workbench の結果は、ボットの定義とそれに対応するテストセットによって異なります。ボット定義の情報を使用してテストセットを生成し、改善が必要な領域を特定できます。現在のボット設計と顧客との会話に関する知識を考慮すると、ボットにとって正しく解釈するのが難しいと思われる (またはわかっている) 例を含むテストデータセットを作成します。

プロダクションボットから学んだことを基に、定期的にインテントを見直してください。ボットのサンプル発話とスロット値を引き続き追加し、調整してください。ランタイムヒントなど、利用可能なオプションを使用してスロットの解像度を改善することを検討してください。ボットの設計と開発は、連続的なサイクルによる反復的なプロセスです。

テストセットを最適化するためのヒントは他にもいくつかあります。

- テストセットで頻繁に使用されるインテントとスロットを含む最も一般的なユースケースを選択してください。
- インテントやスロットを顧客が参照するさまざまな方法を調べてみましょう。これには、ステートメント、質問、コマンドなどのユーザー入力が含まれ、その長さは最小限から拡張までさまざまです。
- さまざまなスロット数のユーザー入力を含めることができます。
- ボットがサポートするカスタムスロット値によく使われる同義語や略語 (「root canal」、「canal」、「RC」など) を含めてください。
- 組み込みスロット値のバリエーション (「tomorrow」、「asap」、「the next day」など) を含めてください。
- 誤解されやすいユーザー入力 (「ink」、「ankle」、「anchor」など) を収集して、ボットによる音声モダリティの堅牢性を調べます。

CSV ファイルからの Test Workbench のテストセットの作成

Amazon Lex V2 コンソールで提供されている CSV ファイルテンプレートからテストセットを作成するには、CSV スプレッドシートエディタを使用して値を直接入力します。テストセットは、1 人のユーザー発話と、次の列に記録された複数回にわたる会話で構成されたカンマ区切り値 (CSV) ファイルです。

- 行番号 — この列は、テストする行の合計数を記録する増分カウンタです。

- 会話番号 — この列は会話のターン数を記録します。単一入力の場合、この列は空白のままにして「-」または「N/A」を入力できます。会話の場合、会話内の各ターンに同じ会話番号が割り当てられます。
- ソース — この列は「ユーザー」または「エージェント」に設定されます。単一入力の場合、常に「ユーザー」に設定されます。
- 入力 — この列には、ユーザーの発話またはボットプロンプトが含まれます。
- 予想される出力インテント — この列には、入力されたインテントが反映されます。
- インテント期待出力スロット 1 — この列には、ユーザー入力で最初に誘発されたスロットをキャプチャします。テストセットには、ユーザー入力のスロットごとに「予想される出力スロット X」という列が含まれている必要があります。

入力が 1 つだけのテストセットの例:

行番号	会話番号	ソース	Input	予想される出力インテント	予想される出力スロット 1	予想される出力スロット 2
1		ユーザー	book a cleaning appointment tomorrow	MakeAppointment	AppointmentType = cleaning	Date = tomorrow
2	該当なし	ユーザー	book a cleaning appointment on April 15th	MakeAppointment	AppointmentType = cleaning	Date = 4/15/23
3	該当なし	ユーザー	book appointment for December first	MakeAppointment	Date = December first	

行番号	会話番号	ソース	Input	予想される出力インテント	予想される出力スロット 1	予想される出力スロット 2
4	該当なし	ユーザー	book a cleaning appointment	MakeAppointment	AppointmentType = cleaning	
1		ユーザー	予約を取ってくれますか?	MakeAppointment		

会話を含むテストセットの例

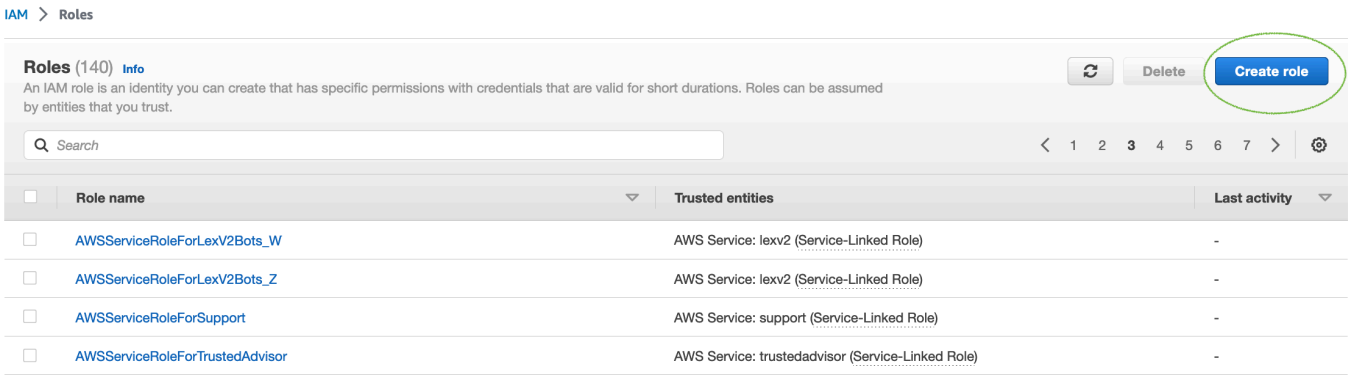
行番号	会話番号	ソース	Input	予想される出力インテント	予想される出力スロット 1	予想される出力スロット 2	予想される出力スロット 3
1	1	ユーザー	book an appointment	MakeAppointment			
2	1	[エージェント]	どのタイプを予約しますか?	MakeAppointment			
3	1	ユーザー	cleaning	MakeAppointment	AppointmentType = cleaning		
4	1	[エージェント]	いつ予約をすればいいですか?	MakeAppointment			

行番号	会話番号	ソース	Input	予想される出力インテント	予想される出力スロット 1	予想される出力スロット 2	予想される出力スロット 3
5	1	ユーザー	tomorrow	MakeAppointment		Date = tomorrow	
6	2	ユーザー	book a root canal appointment today	MakeAppointment	AppointmentType = root canal	Date = today	
7	2	[エージェント]	何時に予約をすればいいですか?	MakeAppointment			
8	2	ユーザー	eleven a.m.	MakeAppointment			Time = eleven a.m.

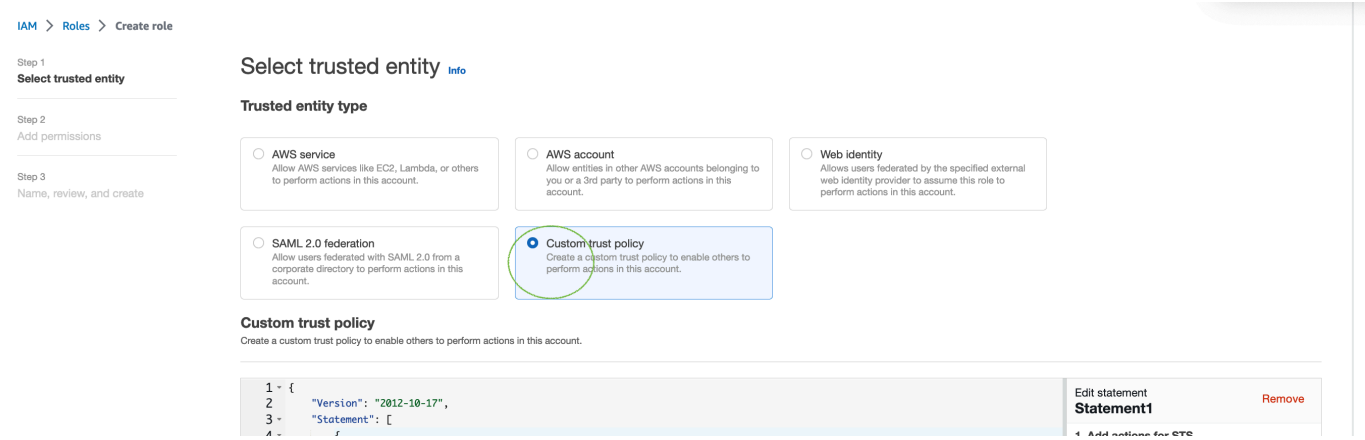
Test Workbench 用の IAM ロールを作成する

Test Workbench 用の IAM ロールを作成するには

1. 「IAM ユーザーの作成」の手順に従って、test-workbench コンソールへのアクセスに使用できる IAM ユーザーを作成します。
2. [Create role] ボタンを選択します。



3. [カスタム信頼ポリシー] のオプションを選択します。



4. 以下に信頼ポリシーを入力し、[次へ] をクリックします。

   JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "sid4",
      "Effect": "Allow",
      "Principal": {
        "Service": "lexv2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

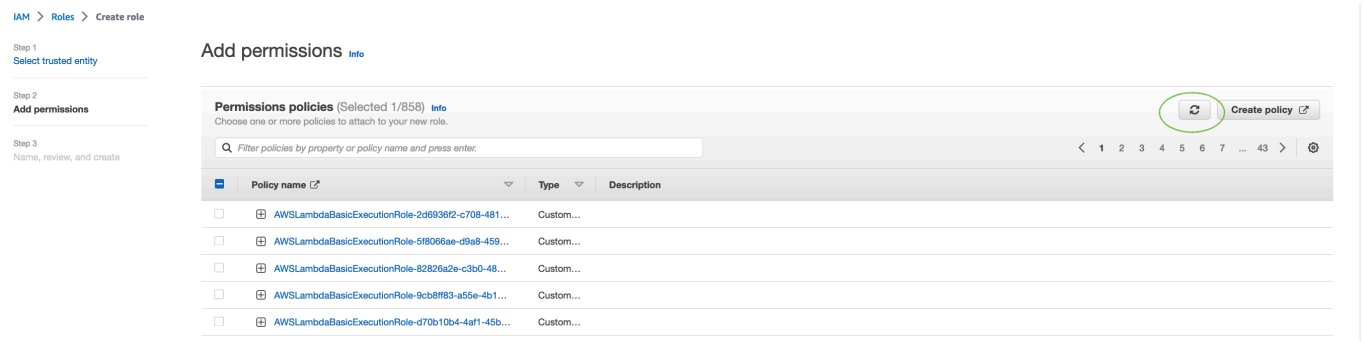
5. [ポリシーの作成] ボタンを選択します。

6. ブラウザに新しいタブが開くので、以下のポリシーを入力して [次: タグ] ボタンをクリックします。

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:FilterLogEvents"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "lex:*"
            ],
            "Resource": "*"
        }
    ]
}
```

7. 「LexTestWorkbenchPolicy」などのポリシー名を入力し、[ポリシーの作成] をクリックします。
8. ブラウザで前のタブに戻り、以下のように [更新] ボタンをクリックしてポリシーのリストを更新します。



9. 6 番目のステップで使用したポリシー名を入力してポリシーのリストを検索し、ポリシーを選択します。
10. [次へ] ボタンを選択します。
11. ロール名を入力し、[ロールを作成] ボタンをクリックします。
12. Amazon Lex V2 コンソールで Test Workbench 用のプロンプトが表示されたら、新しい IAM ロールを選択します。

テストワークベンチ用の IAM ロールを作成する - 高度な機能

テストワークベンチ IAM ロールのアクセス許可の設定

このセクションでは、Test Workbench アクセス許可の最小特権アクセスコントロールを実装するための AWS Identity and Access Management (IAM) アイデンティティベースのポリシーの例をいくつか示します。

1. テストワークベンチが S3 のオーディオファイルを読み取るためのポリシー – このポリシーにより、テストワークベンチはテストセットで使用されているオーディオファイルを読み取ることができます。以下のポリシーは、テストセット内のオーディオファイルの Amazon S3 の場所を指すように、*S3BucketName* と *S3Path* を更新するよう適宜変更する必要があります。

   JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "TestWorkbenchS3AudioFilesReadOnly",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
```

```
        "s3:GetObjectVersion"
      ],
      "Resource": [
        "arn:aws:s3:::S3BucketName/S3Path/*"
      ]
    }
  ]
}
```

2. テストワークベンチがテストセットと結果を Amazon S3 バケットに読み書きするためのポリシー – このポリシーにより、テストワークベンチはテストセットの入力と結果を保存できます。以下のポリシーを変更して、*S3BucketName* をテストセットデータが保存される Amazon S3 バケットに更新する必要があります。テストワークベンチは、これらのデータを Amazon S3 バケットのみに保存し、Lex Service インフラストラクチャには保存しません。そのため、テストワークベンチは、適切に機能させるために Amazon S3 バケットへのアクセスが必要です。

   JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "TestSetDataUploadWithEncryptionOnly",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3:::S3BucketName/*/lex_testworkbench/test_set/*",
        "arn:aws:s3:::S3BucketName/*/lex_testworkbench/test_execution/*",
        "arn:aws:s3:::S3BucketName/*/lex_testworkbench/
test_set_discrepancy_report/*"
      ],
      "Condition": {
        "StringEquals": {
          "s3:x-amz-server-side-encryption": "aws:kms"
        }
      }
    },
    {
      "Sid": "TestSetDataGetObject",
      "Effect": "Allow",
```

```
      "Action": [
        "s3:GetObject",
        "s3:GetObjectVersion"
      ],
      "Resource": [
        "arn:aws:s3:::S3BucketName/*/lex_testworkbench/test_set/*",
        "arn:aws:s3:::S3BucketName/*/lex_testworkbench/test_execution/*",
        "arn:aws:s3:::S3BucketName/*/lex_testworkbench/
test_set_discrepancy_report/*"
      ]
    },
    {
      "Sid": "TestSetListS3Objects",
      "Effect": "Allow",
      "Action": [
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::S3BucketName"
      ]
    }
  ]
}
```

3. テストワークベンチが CloudWatch Logs を読み取るためのポリシー – このポリシーにより、テストワークベンチは Amazon CloudWatch Logs に保存されている Lex 会話テキストログからテストセットを生成できます。以下のポリシーを変更して、*####*、*AwsAccountId*、*LogGroupName* を更新する必要があります。

4. テストワークベンチが Lex ランタイムを呼び出すためのポリシー – このポリシーにより、テストワークベンチは Lex ボットに対してテストセットを実行できます。以下のポリシーを変更して、*#####*、*AwsAccountId*、*BotId* を更新する必要があります。テストワークベンチは Lex 環境で任意のボットをテストできるため、リソースを「arn:aws:lex:*Region*:*AwsAccountId*:bot-alias/*」に置き換えて、テストワークベンチにアカウント内のすべての Amazon Lex V2 ボットへのアクセスを許可できます。

5. (オプション) テストワークベンチがテストセットデータを暗号化および復号化するポリシー – テストワークベンチがカスタマーマネージド KMS キーを使用してテストセットの入力と結果を Amazon S3 バケットに保存するように設定されている場合、テストワークベンチには KMS キーへの暗号化と復号化の両方のアクセス許可が必要です。以下のポリシーを変更して、*Region*、*AwsAccountId*、および *KmsKeyId* を更新する必要があります。ここで、*KmsKeyId* はカスタマーマネージド KMS キーの ID です。

6. (オプション) テストワークベンチがオーディオファイルを復号化するためのポリシー – オーディオファイルがカスタマーマネージド KMS キーを使用して S3 バケットに保存されている場合、テストワークベンチには KMS キーへの復号化アクセス許可が必要です。以下のポリシーを変更して、*Region*、*AwsAccountId*、および *KmsKeyId* を更新する必要があります。ここで、*KmsKeyId* は、オーディオファイルの暗号化に使用されるカスタマーマネージド KMS キーの ID です。

テストセットを管理する

テストセットウィンドウからテストセットをダウンロード、更新、削除できます。または、使用可能なテストセットのリストを使用して、テストセットファイルを編集したり、手動で注釈を付けたりすることもできます。エラーやその他の入力の問題が発生した場合は、もう一度アップロードして検証を再試行してください。

テストセットレコードからテストセットファイルをダウンロードするには:

1. テストセットのリストからテストセットの名前を選択します。
2. テストセットレコードウィンドウで、画面右側の [テスト入力] セクションにある [ダウンロード] ボタンを選択します。
3. ウィンドウ上部にテストセットに関する検証エラーの詳細が表示されている場合は、[ダウンロード] ボタンを選択します。ファイルは [ダウンロード] フォルダに保存されます。テストセット内の検証エラーは、テストセット CSV ファイルのエラーメッセージから修正できます。検証ステップで特定されたエラーを見つけ、行を修正または削除し、ファイルをアップロードして検証ステップを再試行します。
4. テストセットを正常にダウンロードすると、緑色のバナーメッセージが表示されます。

テストセットのリストからテストセットをダウンロードするには:

1. テストセットのリストで、ダウンロードするテストセット項目の横にあるラジオボタンを選択します。
2. 右上の [アクション] メニューから [ダウンロード] を選択します。
3. テストセットを正常にダウンロードすると、緑色のバナーメッセージが表示されます。ファイルは [ダウンロード] フォルダに保存されます。

テストワークベンチでサポートされているテストセット列

以下に、テストワークベンチでサポートされているテストセット列の完全なリストと、Amazon Lex V2 でそれらを使用する方法の手順を示します。

Column Name (列名)	テストセットタイプ	値の型	複数列のサポート	説明
[Line Number] (行番号)	テキストとオーディオ	Number	いいえ	これは Amazon Lex V2 によって無視されるユーザー列です。 V2 これは、テストセットの作成者がテストセット行をソートおよびフィルタリングするのを支援することを目的としています。「行 #」は代替列名として使用できます。
会話番号	テキストとオーディオ	Number	いいえ	この列を使用すると、会話内に行をまとめて配置できます。「会話 #」は代替列名として使用できます。
ソース	テキストとオーディオ	列挙型 (「ユーザー」、「エージェント」)	いいえ	この列の値は、行がユーザー用かエージェント用かを示します。「会話参加者」は代替列名

Column Name (列名)	テストセットタイプ	値の型	複数列のサポート	説明
				として使用できます。
Input	テキスト	String	いいえ	この列は、テキストテストセットのトランスクリプトを追加するために使用されます。「ユーザー」行ではテキスト入力を使用する必要があります。エージェントプロンプトは、「エージェント」行で使用する必要があります。

Column Name (列名)	テストセットタイプ	値の型	複数列のサポート	説明
想定される文字起こし	音声	String	いいえ	この列は、オーディオテストセットのトランスクリプトを追加するために使用されます。オーディオファイルの想定される文字起こしは、オーディオ入力がある「ユーザー」行で使用する必要があります。DTMF 入力は、DTMF 入力がある「ユーザー」行で使用できます。エージェントプロンプトは、「エージェント」行で使用する必要があります。

Column Name (列名)	テストセットタイプ	値の型	複数列のサポート	説明
S3 オーディオの場所	音声	String	いいえ	この列はオーディオファイルの場所を追加するために使用され、オーディオテストセットにのみ適用可能です。S3 パスは、オーディオ入力がある「ユーザー」行で使用する必要があります。このフィールドは、DTMF 入力がある「ユーザー」行では空のままにする必要があります。「エージェント」行では、このフィールドは空のままにする必要があります。

Column Name (列名)	テストセットタイプ	値の型	複数列のサポート	説明
入力コンテキストタグ	テキストとオーディオ	String	はい	この列は、テストセットの行の実行中に Lex への入力で使用される入力コンテキストの名前を指定するために使用されます。これは、「Lex V2 ボットのインテントコンテキストの設定」の入力コンテキストを指します。テストワークベンチはコンテキストの名前のみをサポートすることに注意してください。コンテキスト内のパラメータはサポートされません。「入力コンテキストタグ 1」、「入力コンテキストタグ 2」などの名前の複数の列を使用できます。

Column Name (列名)	テストセットタイプ	値の型	複数列のサポート	説明
リクエスト属性	テキストとオーディオ	String	はい	この列は、テストセットの行の実行中に Lex への入力で使用されるリクエスト属性を指定するために使用されます。列の値は、「<request-attribute-name> = <request-attribute-value>」の形式で指定する必要があります。読みやすいように、「=」の周囲にスペースを追加できます。たとえば、request-attribute-foo = これはダミーレスポンス request-attribute-foo = 'これは「ダミーレスポンス」' request-attribute-foo = "これは「ダミーレスポンス」です。「リクエスト属性 1」、「リクエスト属性 2」など

Column Name (列名)	テストセットタイプ	値の型	複数列のサポート	説明
				の名前の複数の列を使用できます。

Column Name (列名)	テストセットタイプ	値の型	複数列のサポート	説明
セッション属性	テキストとオーディオ	String	はい	この列は、テストセットの行の実行中に Lex への入力で使用されるセッション属性を指定するために使用されます。 • 列の値は、「<session-attribute-name> = <session-attribute-value>」の形式で指定する必要があります。読みやすいように、「=」の周囲にスペースを追加できます。 • 例: • session-attribute-foo = これはダミーレスポンスです • session-attribute-foo = 'これは "ダミーレスポンス" です'

Column Name (列名)	テストセットタイプ	値の型	複数列のサポート	説明
				• session-attribute-foo = "これは 'ダミーレスポンス' です" • 「セッション属性 1」、「セッション属性 2」などの名前の複数の列を使用できます。

Column Name (列名)	テストセットタイプ	値の型	複数列のサポート	説明
ランタイムヒント	テキストとオーディオ	String	はい	この列は、テストセットの行の実行中に Lex への入力で使用されるインテント内のスロットのランタイムヒントを指定するために使用されます。以下に例を示します。 • 列の値は、「<intent-name>.<slot-name> = <slot-value>」の形式で指定する必要があります。読みやすくするために、「=」の周囲にスペースを追加できます。 • 例: • IntentNameFoo.slotNameFoo = ダミー値 • IntentNameFoo.slot

Column Name (列名)	テストセットタイプ	値の型	複数列のサポート	説明
				NameFoo = "'ダミー値"' • IntentNameFoo.slotNameFoo = "'ダミー値"' • テストワークベンチは、ランタイムヒントの複合スロットをサポートしていません。 • 「ランタイムヒント 1」、ランタイムヒント 2」などの名前の複数の列を使用できます。

Column Name (列名)	テストセットタイプ	値の型	複数列のサポート	説明
バージイン	音声	ブール値	いいえ	この列は、テストセット内の行の Lex Runtime にオーディオファイルを送信するときにテストワークベンチがバージインするかどうかを指定するために使用されます。 • ストリーミング API のオーディオテストセットにのみ適用可能です。 • この列は、非ストリーミング API モードでテストセットを実行するときは無視されます。
予想される出力インテント	テキストとオーディオ	String	いいえ	この列は、テストセット内の行の Lex からの出力で想定されるインテントの名前を指定するために使用されます。

Column Name (列名)	テストセットタイプ	値の型	複数列のサポート	説明

予想される出力スロット	テキストとオーディオ	String	はい	この列は、テストセット内の行の実行中に Lex からの出力で想定されるスロット値を指定するために使用されます。
				• 列の値は、「<slot - name> = <slot-value>」の形式で指定する必要があります。読みやすくするために、「=」の周囲にスペースを追加できます。
				• 複数値スロットでも複合スロットでもないスロットの例:
				  • slotNameFoo = ダミー値
				  • slotNameFoo = '"ダミー値"'

Column Name (列名)	テストセットタイプ	値の型	複数列のサポート	説明
				• slotNameFoo = "'ダミー値'" • 複数値スロットの例: • slotNameFoo = value1、value2 • slotNameFoo = value1、"Foo's item" • slotNameFoo = value1、'value2' • スロット名が「Car」で、サブスロット名が「Make」であるときの複合スロットの例: • Car.Make = Toyota • Car.Make = "Toyota" • Car.Make = 'Toyota' • 「想定される出力スロット

Column Name (列名)	テストセットタイプ	値の型	複数列のサポート	説明
				1」、「想定される出力スロット 2」などの名前の複数の列を使用できます。

Column Name (列名)	テストセットタイプ	値の型	複数列のサポート	説明
想定される出力コンテキストタグ	テキストとオーディオ	String	はい	この列は、テストセットの行の Lex からの出力で想定される出力コンテキストの名前を指定するために使用されます。 • これは、「Lex V2 ボットのインテントコンテキストの設定」の出力コンテキストを指します。 • テストワークベンチはコンテキストの名前のみをサポートし、コンテキスト内のパラメータはまだサポートしていないことに注意してください。 • 「予想される出力コンテキストタグ 1」、「予想される出力コンテキストタグ

Column Name (列名)	テストセットタイプ	値の型	複数列のサポート	説明
				2」などの名前の複数の列を使用できます。

Test Workbench でのテスト検証エラーの表示

検証エラーを報告するテストセットを修正できます。これらの検証エラーは、テストセットをテストする準備ができていない場合に生成されます。Test Workbench では、テストセット入力 CSV ファイルのどの必須列の値が期待どおりの形式になっていないかを確認できます。

テスト検証エラーを表示するには:

1. テストセットのリストから、確認する検証エラーのステータスを報告するテストセットの名前を選択します。テストセットの名前は、テストセットに関する詳細に移動するアクティブリンクです。
2. テストセットレコードは、画面上部に検証エラーの詳細を表示します。[詳細を表示] を選択すると、検証エラーに関するレポートが表示されます。
3. エラーレポートウィンドウで、[行番号] と [エラータイプ] を確認して、エラーが発生した場所を確認します。エラーのリストが長い場合は、エラーレポートを [ダウンロード] するよう選択できます。
4. テストセットの入力 CSV ファイルにリストされているエラーを元のテストファイルと比較して問題を修正し、テストセットをもう一度アップロードします。

次の表に、入力 CSV 検証エラーメッセージとシナリオを示します。

シナリオ	エラーメッセージ	注意事項
テストセットのファイルサイズ超過	テストセットのファイルサイズが 200 MB を超えています。小さいファイルを提供して、リクエストを再試行してください。	

シナリオ	エラーメッセージ	注意事項
テストセットが最大レコード数を超えています	入力ファイルには、サポートされている最大数である 200,000 件を超えるレコードが含まれていました。	
空のテストセットをアップロードする	インポートされたテストセットが空です。空でないテストセットを指定して、リクエストを再試行してください。	
空の列ヘッダー名	列ヘッダー行: 列番号 5 に空の列名が見つかりました。	
認識されない列ヘッダー名	列ヘッダー行: 列番号 2 の列名 'ダミー' を認識できませんでした。	
重複数列ヘッダー名	列ヘッダー行: 同じまたは同等の 'S3 オーディオリンク' と 'S3 オーディオリンク' の列が複数見つかりました。これらの列の 1 つを削除するか、名前を変更します。	
複数値の列名が制限を超えました	列ヘッダー行:'予想される出力スロット' の列数が、サポートされている最大数 6 を超えました。'予想される出力スロット' の列をいくつか削除して、やり直してください。	複数値列でサポートされる列の最大数は 6 です。

シナリオ	エラーメッセージ	注意事項
テキストまたはオーディオ関連の列ヘッダーは存在しません。	テキストまたは音声会話の列が見つかりませんでした。テキスト会話には {'テキスト入力'} 列を使用してください。音声会話には {'S3 オーディオリンク', '予想される文字起こし'} 列を使用してください。	音声必須列: {'S3 オーディオリンク', '予想される文字起こし'} テキスト必須列: {'テキスト入力'}
テキスト関連と音声関連の両方のコラムヘッダーが存在します	テキスト会話と音声会話の両方の列が見つかりました。テキスト会話には {'テキスト入力'} 列を、音声会話には {'S3 オーディオリンク', '予想される文字起こし'} 列を使用できます。	音声必須列: {'S3 オーディオリンク', '予想される文字起こし'} テキスト必須列: {'テキスト入力'}
必須列がありません	必須列 ["予想される出力インテント"] が見つかりませんでした。	必須列: {"行 #", "ソース", "予想される出力インテント"}
ヘッダーのない列にデータが見つかりました	行番号 6 の列番号 8 にデータが見つかりましたが、対応する列には列ヘッダーがありませんでした。	
必須列のデータが見つかりません	Row=12: 必須列の値が見つかりません: {"Source", "Expected Output Intent"}	

シナリオ	エラーメッセージ	注意事項
重複する会話 ID が見つかりました	行番号 39 の前の会話に対して、会話番号 '19' が表示されました。2 つの会話に同じ会話番号が指定されていないことを確認してください。そのためには、会話番号のすべての行がグループ化されていることを確認してください。	
入力された会話 ID は無効です	'会話 #' 列に無効な値 'test_conversation' が見つかりました。ユーザー行の場合、この列の値は数値か N/A (該当なし) である必要があります。	
行番号に数値以外の値が入力されています	'行 #' 列に数値以外の値 'test_line' が見つかりました。値は数値である必要があります。	
会話 ID がエージェント行に見つかりません。	'会話 #' 列に値が見つかりませんでした。エージェント行には値を入力する必要があります。	
数字以外の会話 ID がエージェント行に入力されています	[会話 #] 列に数値以外の値「test_conversation」が見つかりました。エージェント行の場合、その値は数値でなければなりません。	
S3 の場所が無効	無効な値 'bucket/folder' が提供されました。有効な形式は S3://<bucketName>/<keyName> です。	

シナリオ	エラーメッセージ	注意事項
S3 バケット名が無効	無効な S3 バケット名 'test_bucket' が入力されました。バケット名を確認してください。	
S3 オーディオの場所はフォルダです	指定されたオーディオの場所 "S3://bucket/folder' が無効です。S3 フォルダを指しています。	
無効なインテント名	インテント 'intent @name 'に無効な文字が含まれていました。インテント名を確認してください。	正規表現チェック: ^([0-9a-zA-Z][_-]?)+$
無効なスロット名	スロット 'Slot @Name 'に無効な文字がありました。スロット名を確認してください。	正規表現: ^([0-9a-zA-Z][_-]?)+$It 先頭や末尾にドット (.) を付けてはいけません
親スロットに指定されたスロット値	サブスロット'Address.City' と親スロット 'Address' にスロット値が指定されました。値はサブスロットにのみ指定してください。	CST の親スロットにはスロット値があってはなりません
コンテキスト名に無効な文字があります	コンテキスト名 'context @1 'に無効な文字が含まれていました。コンテキスト名を確認してください。	正規表現:^ ([A-zA-Z] _?) +$
無効なスロットのスペルスタイル	無効な値 'test' が提供されました。すべて大文字であることを確認してください。有効な値は ["Default", "SpellByLetter", "SpellByWord"] です。	サポートされている値 ["Default", "SpellByLetter", "SpellByWord"]

シナリオ	エラーメッセージ	注意事項
参加者またはソースはエージェントまたはユーザーでなければなりません	無効な値 'ボット' が提供されました。有効な値は ["エージェント", "ユーザー"] です。	サポートされている列挙型: "エージェント", "ユーザー"
行番号は 10 進数であってはなりません	無効な値 '10.1' が提供されました。分数のない有効な数値でなければなりません。	
会話番号は 10 進数であってはなりません。	無効な値 '10.1' が提供されました。分数のない有効な数値でなければなりません。	
行番号は範囲内でなければなりません	無効な値 '92233720 368547758071' が提供されました。これは、1 以上で、9223372036854775807 以下である必要があります。	
割り込み列はブーリアン値のみを受け付けます	無効な値 'test' が提供されました。'true' や 'false' などの有効なブール値でなければなりません。代わりに 'yes' と 'no' を使用することもできます。	使用できる値:"True"、 "true"、 "T"、 "Yes"、 "yEs"、 "Y"、 "1"、 "1.0"、"False"、 "false"、 "F"、 "No"、 "no"、 "N"、 "0"、 "0.0"
想定されるスロット、セッション属性、リクエスト属性は (=) で区切る必要があります	値 'slotName:slotValue' に '=' はありません。このような値は '<key>=<value>' の形式でキーと値のペアとして指定する必要があります。	例: slotName = slotType
予想されるスロット、セッション属性、リクエスト属性にはキーと値のペアが必要です	'=slotValue' に '=' より前のキーがありません。このような値は '<key>=<value>' の形式でキーと値のペアとして指定する必要があります。	例: slotName = slotType

シナリオ	エラーメッセージ	注意事項
無効な末尾の引用符	'Foo's item"‘ に誤った引用が見つかりました。引用文字 `"` で始まりますが、同じ引用文字で終わるわけではありません。	例: `"Foo's item", KFC`
真ん中の引用が無効です	`"Foo's" Burger, etc` に誤った引用が見つかりました。コンテンツ内に引用文字 `"` が含まれています。一重引用符を含む値は二重引用符で囲む必要があり、その逆も同様です。	正しい例: `"Foo's item", KFC`
必須の引用文字	`key = Foo's item` には一重引用符または二重引用符が含まれていますが、引用符で囲まれていません。一重引用符を含む値は二重引用符内に囲む必要があり、その逆も同様です。	
列内で重複キーが繰り返されています。	キー `key1` が `セッション属性 3` と `セッション属性 1` の 2 つの列で繰り返されました。	
ランタイムヒントの形式が無効です	ランタイムヒントに提供された `BookFlight.Car"` というキーが無効です。ランタイムヒントの場合、キーは形式 <intentName>.<slotName> である必要があります。	キーの中央に '.' がある必要がある場合、そのキーからインテント名とスロット名を誘発することはできません。このような誤った形式の例としては、"BookFlight"、".BookFlight.Car"、"BookFlight.Car." があります。

シナリオ	エラーメッセージ	注意事項
無効なランタイムヒントキーのインテント名	ランタイムヒントのインテント `intent @name `が無効です。インテント名を確認してください。	正規表現チェック: ^([0-9a-zA-Z][_-]?)+$
無効なランタイムヒントキーのスロット名	ランタイムヒントの `Slot @Name `に無効なスロット名が見つかりました。スロット名を確認してください。	正規表現: ^([0-9a-zA-Z][_-]?)+$It 先頭や末尾にドット (.) を付けてはいけません

Test Workbench でのテストセットの削除

テストセットのリストからテストセットを簡単に削除できます。

テストセットを削除するには:

1. 左側のメニューから [テストセット] のリストに移動すると、テストセットのリストが表示されます。
2. テストセットのリストから、削除するテストセットを選択します。
3. 右上の [アクション] ドロップダウンメニューに移動し、[削除] を選択します。
4. テストセットが削除されたことを確認するメッセージが表示されます。

テストセットの詳細を編集します。

テストセットのリストでテストセット名と詳細を編集できます。名前や詳細は後で追加または更新できます。ただし、ボットまたは文字起こしデータを使用してテストを実行する前に、テストセットを更新する必要があります。

テストセットの詳細を編集するには:

1. 左側のメニューからテストセットのリストに移動すると、テストセットのリストが表示されます。
2. テストセットのリストから、編集するテストセットのチェック ボックスを選択します。
3. 右上の [アクション] ドロップダウンメニューに移動し、[詳細の編集] を選択します。

4. テストセットが正常に編集されたことを確認するメッセージが表示されます。

テストセットを更新する

テストセットから項目を更新、修正、変更、または削除して、ベースライン結果を最適化したり、テストセットで発生した可能性のあるその他のエラーを修正したりすることができます。

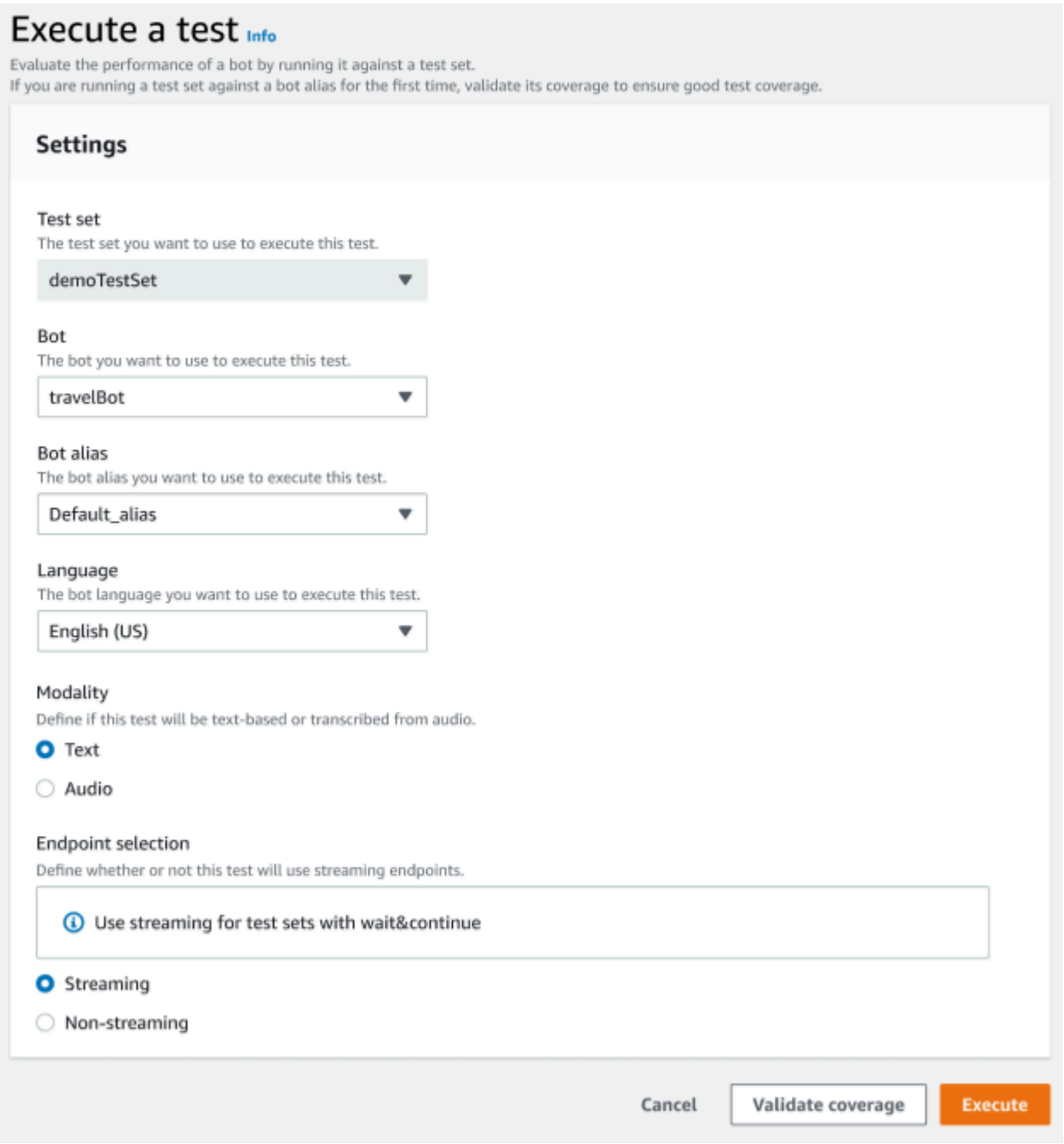
修正したテストセットをアップロードする前に、テストセットをダウンロードして検証エラーを修正できます。「テスト検証エラーを表示する」を参照してください。

テストセットを更新するには:

1. テストセットレコードから、右上の [テストセットの更新] ボタンを選択します。
2. Amazon S3 アカウントからアップロードするファイルを選択するか、コンピュータから CSV テストファイルをアップロードします。注: テストセットを更新すると、既存のデータが上書きされます。
3. [更新] ボタンを選択します。
4. テストセットが正常に更新されたことを確認するメッセージが表示されます。注: テストセットの複雑さとサイズによっては、この操作に数分かかる場合があります。
5. テストセットが正常に更新されたことを確認するメッセージが表示され、[ステータス] に [テスト準備完了] と表示されます。

テストを実行する

テストセットを実行するには、テストセットに対してテストを実行する適切なボットを選択する必要があります。テストセットのドロップダウンメニューから AWS アカウントからボットを選択できます。この操作では、選択したボットを検証済みのテストデータと照合してテストし、テストセットのベースラインデータに対するパフォーマンスメトリクスを報告します。



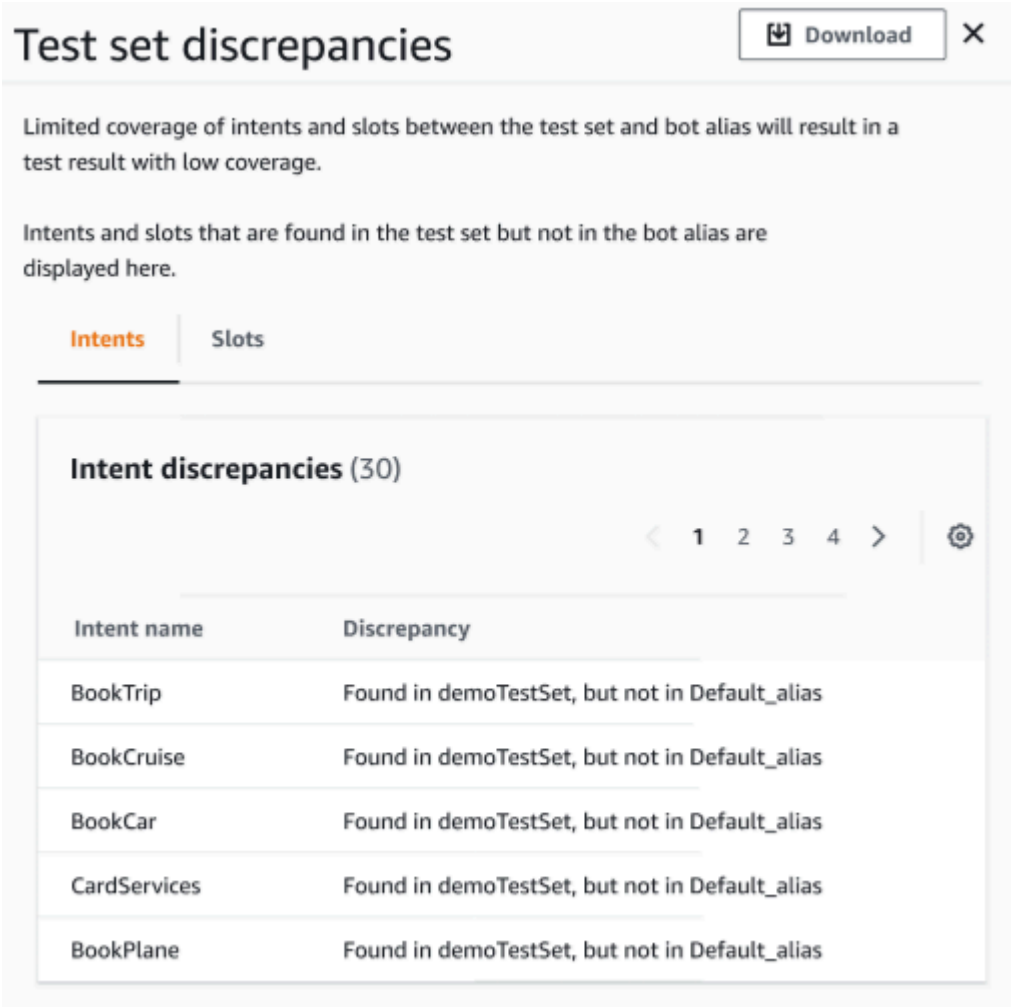
Test Workbench でテストを実行するには

1. テストセットのレコードページで、[テストを実行] を選択します。
2. テストで使用するテストセットを選択します。
3. テストで使用するボットの名前を [ボット] ドロップダウンメニューから選択します。
4. 該当する場合は、[ボットエイリアス] ドロップダウンメニューからボットエイリアスを選択します。

5. [言語] 選択項目から、英語版を選択します。
6. モダリティタイプとして [テキスト] または [音声] を選択します。
7. Amazon S3 の場所を選択します。 (オーディオのみ)
8. ボットのエンドポイントセレクションを選択します。(ストリーミングのみ)
9. [カバレッジを検証] ボタンを選択して、テストを実行する準備ができていることを確認します。検証ステップでエラーがある場合は、前のパラメータを確認して修正してください。
10. [実行] を選択してテストを実行します。
11. テストが正常に実行されたことを確認するメッセージが表示されます。

Test Workbench でのテストセットのカバレッジ

テストセットとボットの間のインテントとスロットの範囲が限られているため、期待どおりのパフォーマンスを測定できる可能性があります。テストを実行する前に、テストセットのカバレッジを確認することをお勧めします。



検証力バレッジを確認するには

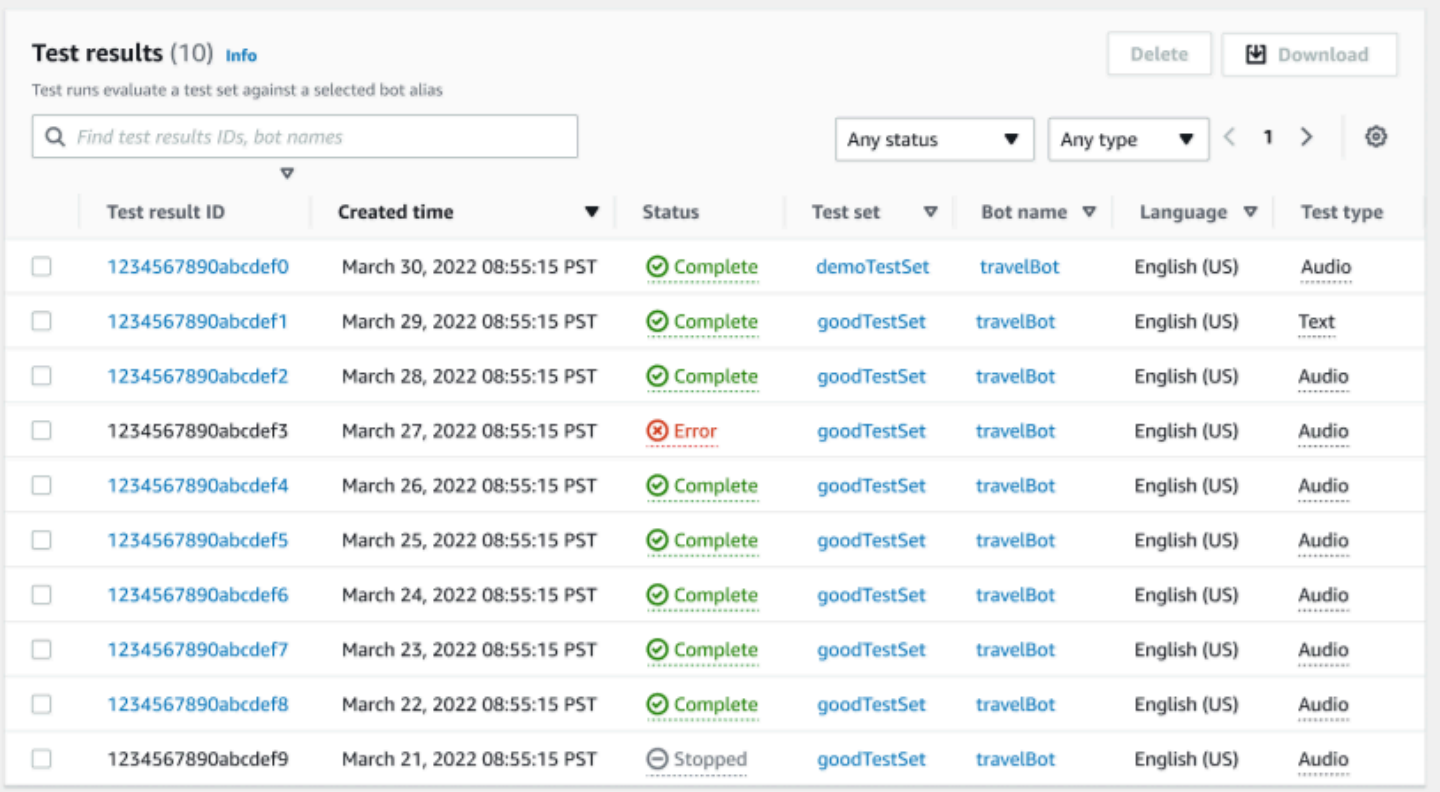
1. テストセットのレコードで、[力バレッジを検証] ボタンを選択します。
2. このメッセージは、テストセットと選択したボットの間の力バレッジを検証中であることを示しています。
3. 操作が完了すると、[力バレッジ検証が成功しました] というメッセージが表示されます。
4. ウィンドウの下部にある [詳細を表示] ボタンを選択します。
5. インテントとスロットのテストセットの不一致を確認するには、それぞれのタブを選択します。[ダウンロード] ボタンを選択すると、このデータを CSV 形式でダウンロードできます。
6. テストセットデータ、ボットインテント、スロットの検証結果を確認します。問題を特定し、ボットのテストセットのアーキテクチャを変更して結果を改善します。CSV ファイルに変更を加えたら、編集したテストセットとボットをアップロードしてテストを実行します。注: 検証力バレッジはテストセットに対して実行され、ボットに対しては実行されません。ボットに含まれていてもテストセットには存在しないインテントは対象外です。

テスト結果を表示する

Test Workbench からのテスト結果を解釈して、ボットと顧客との会話が失敗している箇所や、顧客がインテントを達成するために何度も試みる必要がある箇所を特定します。

テスト結果でこれらの問題を特定することにより、ボットのリアルタイムの文字起こし値とより一致したさまざまなトレーニングデータや発話を使用してインテントのパフォーマンスを向上させ、ボットのパフォーマンスを最適化できます。

パフォーマンスに差異があったインテントとスロットを詳細に表示できます。差異があるインテントやスロットを特定したら、さらに掘り下げて発話や会話の流れを確認できます。



テスト結果をレビューするには:

1. 左側のメニューからテストセットのリストに移動し、Test Workbench の [テスト結果] オプションを選択します。注: テスト結果に成功すると、[ステータス] に完了と表示されます。
2. 確認するテスト結果の [テスト結果 ID] を選択します。

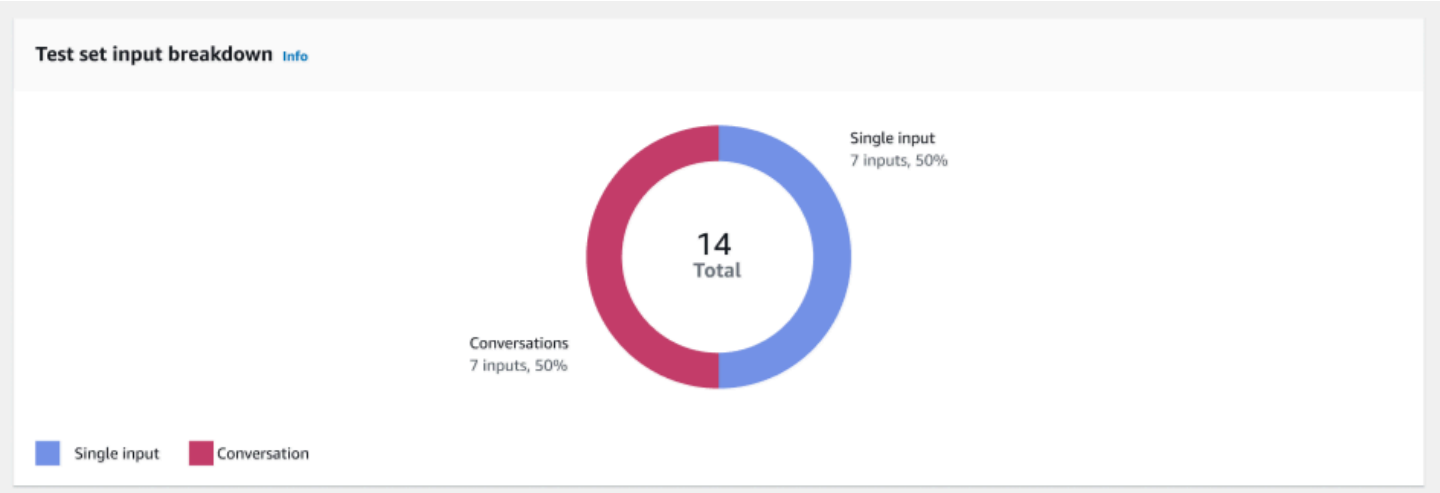
Test Workbench のテスト結果の詳細

テスト結果には、テストセットの詳細、使用されたインテント、使用されたスロットが表示されます。また、全体的な結果、会話結果、インテント、スロット結果など、テストセット全体の入力の内訳も表示されます。

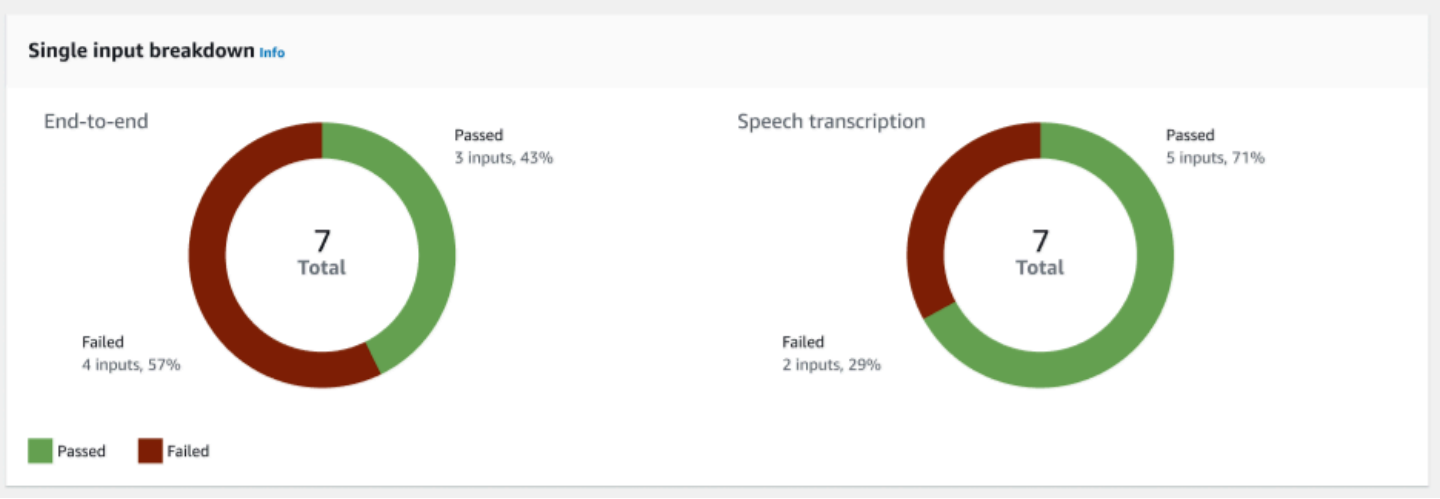
テスト結果には、次のようなすべてのテスト関連情報が含まれます。

- テストの詳細メタデータ
- 総合結果
- 会話の結果
- インテントとスロットの結果
- 詳細結果

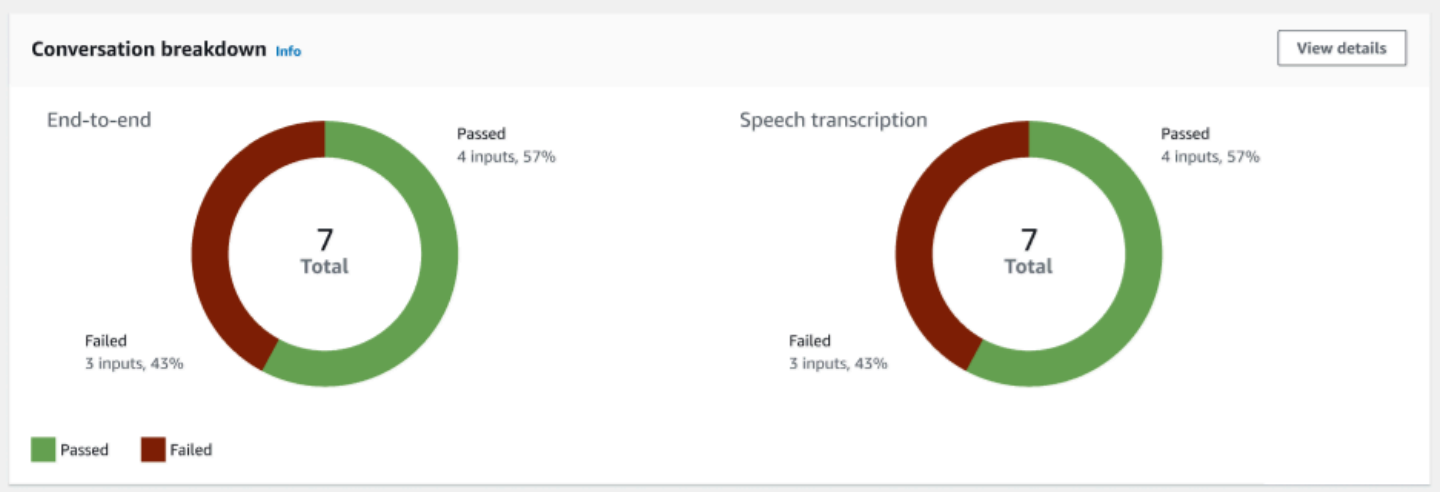
[総合結果] タブ:



テストセットの入力内訳 — このチャートには、テストセット内の会話数と単一入力発話数の内訳が表示されます。

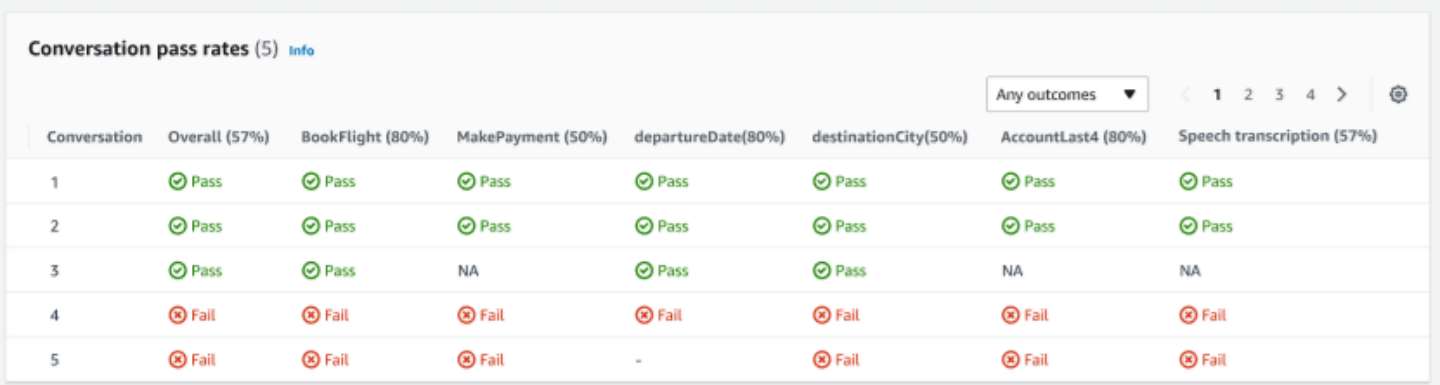


単一入力の内訳 — エンドツーエンドの会話と音声文字起こしを含む 2 つのグラフを表示します。合格した入力と不合格だった入力の数が各チャートに示されます。注: 音声文字起こしチャートはオーディオテストセットでのみ表示されます。



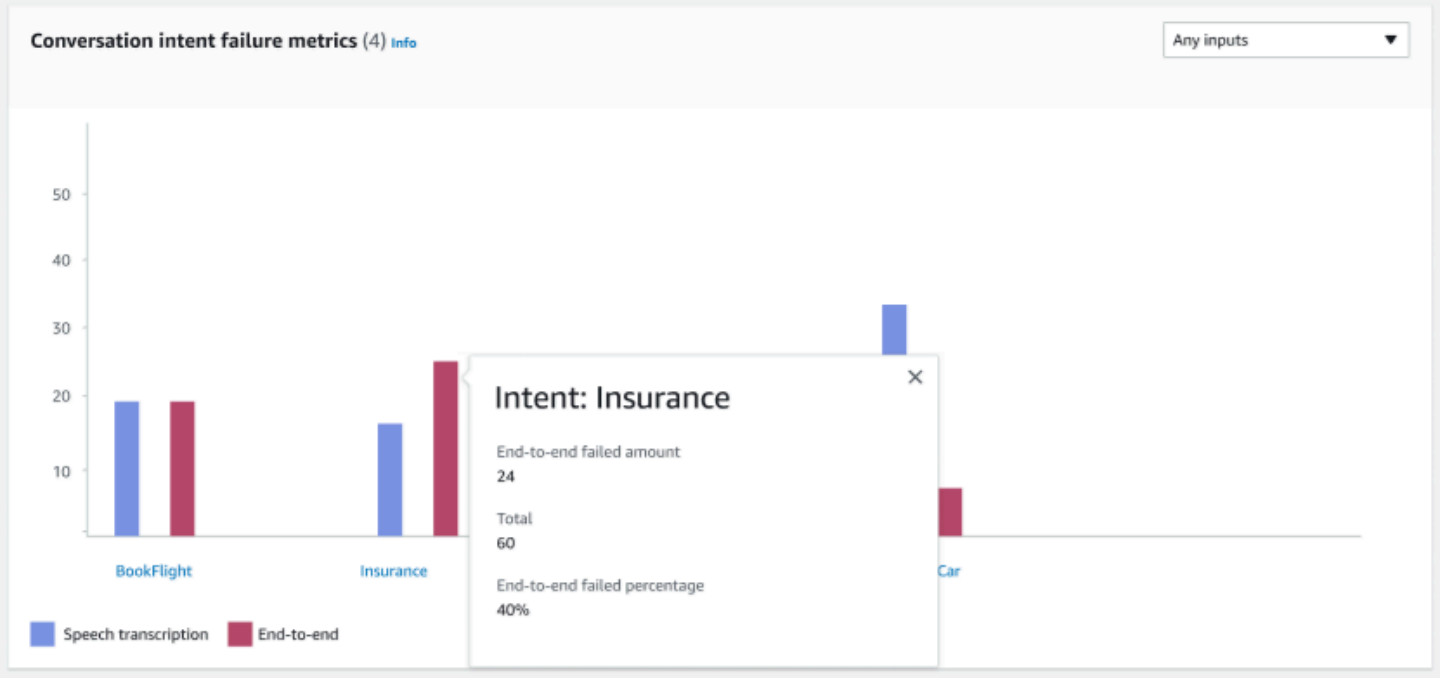
会話の内訳 — エンドツーエンドの会話と音声文字起こしを含む 2 つのグラフを表示します。合格した入力と不合格だった入力の数が各チャートに示されます。注: 音声文字起こしチャートはオーディオテストセットでのみ表示されます。

[会話の結果] タブ:



Conversation	Overall (57%)	BookFlight (80%)	MakePayment (50%)	departureDate(80%)	destinationCity(50%)	AccountLast4 (80%)	Speech transcription (57%)
1	Pass	Pass	Pass	Pass	Pass	Pass	Pass
2	Pass	Pass	Pass	Pass	Pass	Pass	Pass
3	Pass	Pass	NA	Pass	Pass	NA	NA
4	Fail	Fail	Fail	Fail	Fail	Fail	Fail
5	Fail	Fail	Fail	-	Fail	Fail	Fail

会話合格率 — 会話合格率テーブルを使用して、テストセット内の各会話でどのインテントとスロットが使用されているかを確認できます。どのインテントまたはスロットが不合格だったかを確認し、各インテントとスロットの合格率を確認することで、会話のどこで失敗したかを視覚化できます。

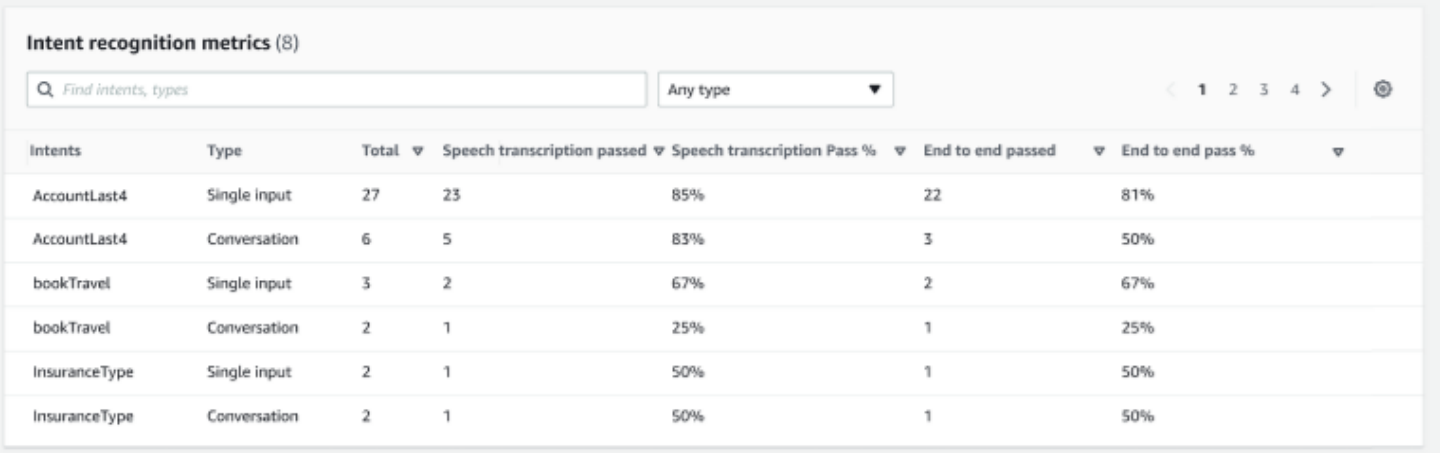


会話インテント不合格メトリクス — このメトリクスは、テストセットの中でパフォーマンスが最も悪かったインテントの上位 5 つを示します。このパネルには、ボットの会話ログまたは文字起こしに基づいて、成功または失敗したインテントの割合または数のグラフが表示されます。インテントが成功しても、会話全体が成功したわけではありません。これらのインテントは、どのインテントが前または後に来たかにかかわらず、インテントの値にのみ適用されます。



会話スロット不合格メトリクス — このメトリクスは、テストセットの中でパフォーマンスが最も悪かったスロットの上位 5 つを示します。インテント内の各スロットの成功率を示しました。棒グラフには、インテント内の各スロットの音声文字起こしとエンドツーエンドの会話の両方が表示されます。

[インテントとスロットの結果] タブ:

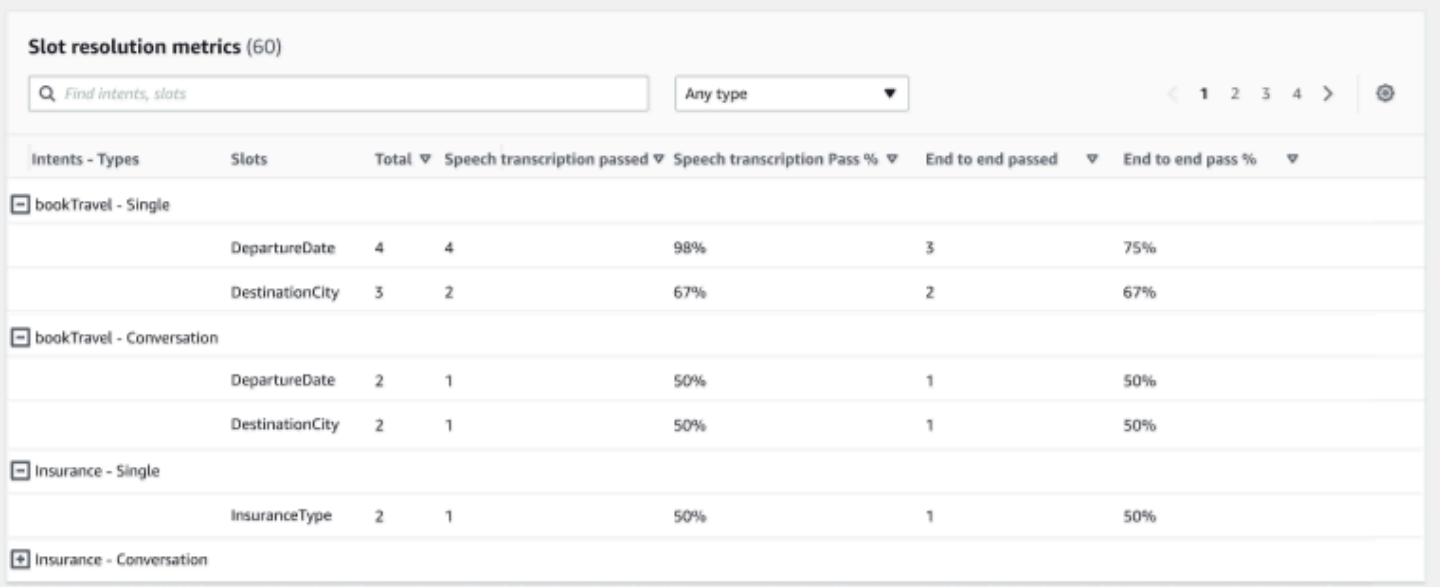


Intent recognition metrics (8)

Find intents, types | Any type | 1 2 3 4

Intents	Type	Total	Speech transcription passed	Speech transcription Pass %	End to end passed	End to end pass %
AccountLast4	Single input	27	23	85%	22	81%
AccountLast4	Conversation	6	5	83%	3	50%
bookTravel	Single input	3	2	67%	2	67%
bookTravel	Conversation	2	1	25%	1	25%
InsuranceType	Single input	2	1	50%	1	50%
InsuranceType	Conversation	2	1	50%	1	50%

インテント認識メトリクス — 正常に認識されたインテントの数を示す表が表示されます。音声文字起こしとエンドツーエンドの会話の合格率を表示します。



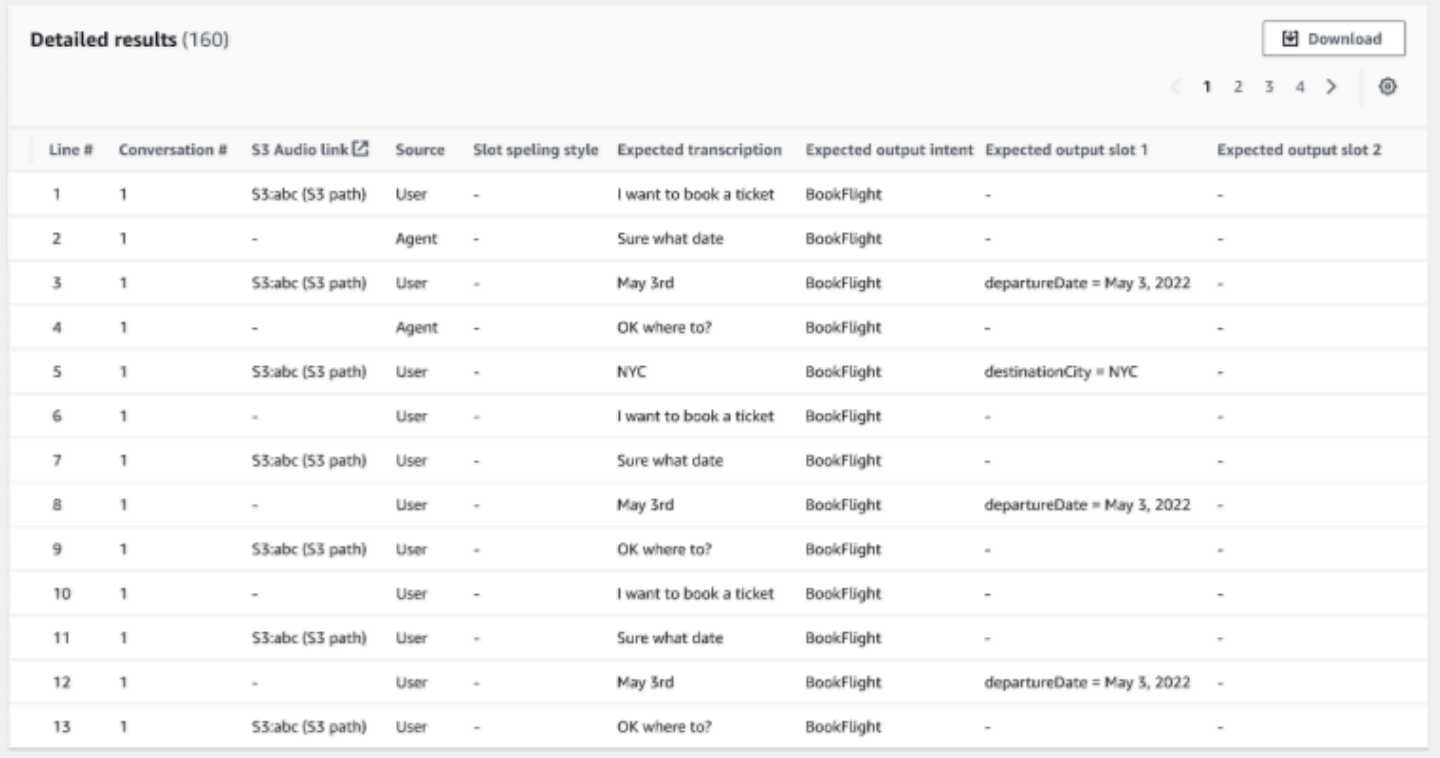
Slot resolution metrics (60)

Find intents, slots | Any type | 1 2 3 4

Intents - Types	Slots	Total	Speech transcription passed	Speech transcription Pass %	End to end passed	End to end pass %
bookTravel - Single						
	DepartureDate	4	4	98%	3	75%
	DestinationCity	3	2	67%	2	67%
bookTravel - Conversation						
	DepartureDate	2	1	50%	1	50%
	DestinationCity	2	1	50%	1	50%
Insurance - Single						
	InsuranceType	2	1	50%	1	50%
Insurance - Conversation						

スロット解決メトリクス — インテントとスロットを別々に表示し、会話または単一入力で使用された各インテントの各スロットの成功率と失敗率を表示します。音声文字起こしとエンドツーエンドの会話の合格率を表示します。

[詳細結果] タブ:



Detailed results (160)

Download

1 2 3 4

Line #	Conversation #	S3 Audio link	Source	Slot speling style	Expected transcription	Expected output intent	Expected output slot 1	Expected output slot 2
1	1	S3:abc (S3 path)	User	-	I want to book a ticket	BookFlight	-	-
2	1	-	Agent	-	Sure what date	BookFlight	-	-
3	1	S3:abc (S3 path)	User	-	May 3rd	BookFlight	departureDate = May 3, 2022	-
4	1	-	Agent	-	OK where to?	BookFlight	-	-
5	1	S3:abc (S3 path)	User	-	NYC	BookFlight	destinationCity = NYC	-
6	1	-	User	-	I want to book a ticket	BookFlight	-	-
7	1	S3:abc (S3 path)	User	-	Sure what date	BookFlight	-	-
8	1	-	User	-	May 3rd	BookFlight	departureDate = May 3, 2022	-
9	1	S3:abc (S3 path)	User	-	OK where to?	BookFlight	-	-
10	1	-	User	-	I want to book a ticket	BookFlight	-	-
11	1	S3:abc (S3 path)	User	-	Sure what date	BookFlight	-	-
12	1	-	User	-	May 3rd	BookFlight	departureDate = May 3, 2022	-
13	1	S3:abc (S3 path)	User	-	OK where to?	BookFlight	-	-

詳細結果 — ユーザーとエージェントの発話、各スロットの予想される出力と想定される文字起こしを含む詳細な表を会話ログに表示します。このレポートは [ダウンロード] ボタンを選択してダウンロードできます。

次の表に、結果不合格エラーメッセージとシナリオを示します。

シナリオ	エラーメッセージ	アクション
インテントが一致しません	BookFlight インテントを期待していましたが、BookHotel インテントでした。	会話の他のターンをスキップする
スロット誘発が一致しません	departureDate スロットが誘発される予定でしたが、cabinType でした。	会話の他のターンをスキップする
スロット値が一致しない	予想されるスロット値と実際のスロット値が一致しません。	会話の他のターンを続ける

シナリオ	エラーメッセージ	アクション
バックツーバックのエージェントプロンプトが表示されない	このターン、ボットがエージェントプロンプトを返すはずでしたが、受信されませんでした。	会話の他のターンをスキップする
文字起こしが一致しない	予想される文字起こしが、実際の文字起こしと一致しませんでした。	会話の他のターンを続ける
オプションスロットが誘発されない	次のターンで cabinType スロットを誘発することが期待されていますが、その前に現在のインテントが達成されています。	会話の他のターンをスキップする
スロットが認識されない	このターンで departureDate スロットは認識されませんでした。	会話の他のターンをスキップする
連続して追加されたエージェントプロンプト	ユーザーターンが予想されましたが、エージェントプロンプトでした	会話の他のターンをスキップする

Amazon Lex V2 ボットへの会話のストリーミング

Amazon Lex V2 ストリーミング API を使用して、Amazon Lex V2 ボットとアプリケーション間の双方向ストリーミングをスタートできます。ストリーミングをスタートして、ボットでボットとユーザー間の会話を管理できます。ボットは、ユーザーからのレスポンスを処理するコードの記述なしに、ユーザー入力に応答します。ボットは次のことを実行できます。

- プロンプトの再生中にユーザーからの中断を処理します。詳細については、「ユーザーによる Amazon Lex V2 ボットの中断を可能にする」を参照してください。
- ユーザーによる情報入力を待機します。たとえば、ボットはユーザーがクレジットカード情報を収集するのを待機できます。詳細については、「一時停止中に、Amazon Lex V2 ボットによるユーザーの追加の情報提供の待機を可能にする」を参照してください。
- デュアルトーンマルチ周波数 (DTMF) と音声入力の両方を同じストリーミングで受け取ります。
- アプリケーションから会話を管理するよりも、ユーザー入力で一時停止を最適に処理します。

Amazon Lex V2 ボットは、アプリケーションから送信されたデータに応答するだけでなく、会話の状態に関する情報をアプリケーションに送信します。この情報を使用して、アプリケーションの顧客への応答方法を変更できます。

Amazon Lex V2 ボットは、ボットとアプリケーション間の接続もモニタリングします。接続がタイムアウトしたかどうかを判断できます。

API を使用して Amazon Lex V2 ボットへのストリーミングをスタートするには、「Amazon Lex V2 ボットへの会話ストリーミングの開始」を参照してください。

アプリケーションから Amazon Lex V2 ボットへのストリーミングをスタートすると、ユーザーからの音声入力またはテキスト入力を受け入れるようにボットを設定できます。また、ユーザーが入力に応答して音声またはテキストを受信するかどうかを選択できます。

Amazon Lex V2 ボットがユーザーからの音声入力を受け入れるように設定している場合、テキスト入力は受け取れません。テキスト入力を受け入れるようにボットを設定した場合、ユーザーは入力テキストのみを使用して通信できます。

Amazon Lex V2 ボットがストリーミング音声入力を受け取ると、ボットはユーザーの話の始まりと終わりを判断します。ユーザーからの一時停止や中断を処理します。DTMF (デュアルトーンマルチ周波数) 入力と音声入力を同じストリーミングで受け取ることもできます。これにより、ユーザーが

ボットとより自然にやり取りできるようになります。ウェルカムメッセージとプロンプトをユーザーに提示できます。ユーザーがこれらのメッセージおよびプロンプトを中断できるように設定することもできます。

双方向ストリーミングをスタートするとき、Amazon Lex V2 により HTTP/2 プロトコル が使用されます。アプリケーションとボットは、一連の イベント として単一のストリーミングでデータを交換します。イベントの値は次のいずれかを指定できます。

- ユーザーからのテキスト、音声、または DTMF の入力。
- アプリケーションから Amazon Lex V2 ボットへのシグナル送信。これには、メッセージの音声の再生が完了したこと、またはユーザーがセッションから切断されたことを示します。

イベントの詳細については、「Amazon Lex V2 ボットへの会話ストリーミングの開始」を参照してください。イベントをエンコードする方法については、「イベントストリームエンコード」を参照してください。

トピック

- Amazon Lex V2 ボットへの会話ストリーミングの開始
- イベントストリームエンコード
- ユーザーによる Amazon Lex V2 ボットの中断を可能にする
- 一時停止中に、Amazon Lex V2 ボットによるユーザーの追加の情報提供の待機を可能にする
- Lex V2 ボットのフルフィルメント進行状況の更新の設定
- Lex V2 ボットでのユーザー入力キャプチャのタイムアウトの設定

Amazon Lex V2 ボットへの会話ストリーミングの開始

ユーザーと Amazon Lex V2 ボット間のストリーミングをアプリケーション内スタートする StartConversation オペレーションを使用します。アプリケーションからの POST リクエストにより、アプリケーションと Amazon Lex V2 ボット間の接続が確立されます。これにより、アプリケーションとボットがイベントを通じて相互の情報交換をスタートできます。

> **Note**
>
> StartConversation を使用する場合、発話オーディオの再生時間が の設定値を超えるとmax-length-ms、Amazon Lex V2 は指定された再生時間でオーディオを切断します。

StartConversation オペレーションは、次の SDK でのみサポートされます。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for JavaScript v3
- AWS SDK for Ruby v3

ConfigurationEvent は、アプリケーションが Amazon Lex V2 ボットに送信する必要のある最初のイベントです。このイベントには、レスポンスタイプの形式などの情報が含まれます。設定イベントで使用できるパラメータを次に示します。

- responseContentType - ボットがユーザー入力に対しテキストまたは音声で応答するかどうかを決定します。
- sessionState - あらかじめ決められたインテントやダイアログの状態など、ボットとのストリーミングセッションに関連する情報。
- welcomeMessages - ボットとの会話の開始時の、ユーザーに対して再生されるウェルカムメッセージを指定します。これらのメッセージは、ユーザーが何かしらの入力をする前に再生されます。ウェルカムメッセージを起動されるには、sessionState と dialogAction パラメータの値も指定する必要があります。
- disablePlayback - ボットが発信者の入力を聞き始める前に、クライアントからのキューを待機するかどうかを決定します。デフォルトでは再生が有効になっているため、このフィールドの値は false となります。
- requestAttributes - リクエストに関する追加情報を提供します。

前述のパラメータの値を指定する方法については、「StartConversation オペレーションの ConfigurationEvent データタイプ」を参照してください。

ボットとアプリケーションの間の各ストリーミングには、1 つの設定イベントのみ持つことができます。アプリケーションが設定イベントを送信した後、ボットはアプリケーションから追加の通信を受け取ることができます。

ユーザーが音声を使用して Amazon Lex V2 ボットと通信するように指定した場合、アプリケーションはその会話中にボットに次のイベントを送信できます。

- AudioInputEvent – 最大サイズが 320 バイトの音声のチャンクが含まれます。アプリケーションは、サーバーからボットにメッセージを送信するために、複数の音声入力イベントを使用する必要

があります。ストリーミング内のすべての音声入力イベントは、同じ音声の形式である必要があります。

- DTMFInputEvent – DTMF 入力をボットに送信します。DTMF キーを押すごとに、1 つのイベントに対応します。
- PlaybackCompletionEvent – ユーザーの入力によるレスポンスが再生されたことをサーバーに通知します。ユーザーに音声レスポンスを送信する場合は、再生完了イベントを使用する必要があります。もし構成イベントの `disablePlayback` が `true` の場合、この機能は使用できません。
- DisconnectionEvent – ユーザーが会話から切断されたことをボットに通知します。

ユーザーがテキストを使用してボットと通信するように指定した場合、アプリケーションはその会話中にボットに次のイベントを送信できます。

- TextInputEvent – アプリケーションからボットに送信されるテキスト。最大 512 文字まで 1 つのテキスト入力イベントに追加できます。
- PlaybackCompletionEvent – ユーザーの入力によるレスポンスが再生されたことをサーバーに通知します。音声をユーザーに再生する場合は、このイベントを使用する必要があります。もし構成イベントの `disablePlayback` が `true` の場合、この機能は使用できません。
- DisconnectionEvent – ユーザーが会話から切断されたことをボットに通知します。

Amazon Lex V2 ボットに送信するすべてのイベントは、正しい形式でエンコードする必要があります。詳細については、「イベントストリームエンコード」を参照してください。

すべてのイベントにはイベント ID があります。ストリーミングで発生する可能性のある問題のトラブルシューティングに役立つように、各入力イベントに一意のイベント ID を割り当てます。その後、処理に失敗した場合のトラブルシューティングをボットで行うことができます。

Amazon Lex V2 では、イベントごとにタイムスタンプも使用されます。イベント ID に加えてこれらのタイムスタンプを使用して、ネットワーク伝送に関する問題のトラブルシューティングに役立てることができます。

ユーザーと Amazon Lex V2 ボット間の会話中に、ボットはユーザーに応答して次のアウトバウンドイベントを送信できます。

- IntentResultEvent – Amazon Lex V2 がユーザーの発話から判断したインテントが含まれます。各内部結果イベントには以下が含まれます。
  - inputMode - ユーザーの発話のタイプ。有効な値は `Speech`、`DTMF`、または `Text` です。

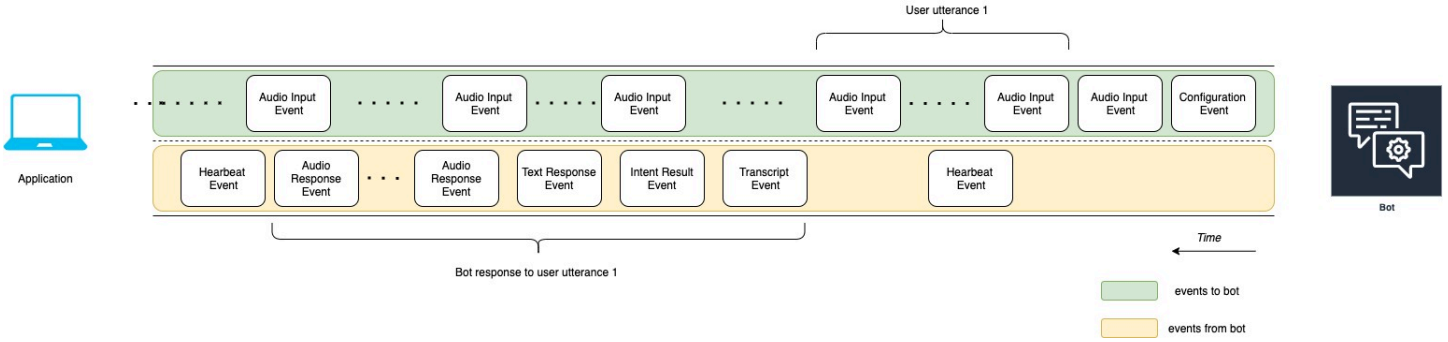
  - interpretations - Amazon Lex V2 がユーザーの発話から判断する解釈。
  - requestAttributes - Lambda 関数でリクエスト属性を変更していない場合、この属性は会話のスタート時に渡された属性と同じになります。
  - sessionId - 会話に使用されるセッション識別子。
  - sessionState - Amazon Lex V2 とのユーザーのセッションの状態。
- TranscriptEvent – ユーザーがアプリケーションへの入力を行った場合、このイベントには、ボットに対するユーザーの発話の内容が記録されます。ユーザー入力がない場合、`TranscriptEvent` はアプリケーションでは受け取られません。

  アプリケーションに送信されるトランスクリプトイベントの値は、会話モードとして音声 (スピーチおよび DMTF) とテキストのどちらを指定したかによって異なります。
  - 音声入力のトランスクリプト - ユーザーがボットと話している場合、トランスクリプトイベントはユーザーの音声の書き起こしになります。これは、ユーザーが話し始めた時点から話し終わるまでのすべてのスピーチのトランスクリプトです。
  - DTMF 入力のトランスクリプト - ユーザーがキーパッドで入力している場合、トランスクリプトイベントには、ユーザーが入力で押したすべての数字が含まれます。
  - テキスト入力のトランスクリプト - ユーザーがテキスト入力を行っている場合、トランスクリプトイベントにはユーザーの入力したすべてのテキストが含まれます。
- TextResponseEvent – テキスト形式のボットのレスポンスを含んでいます。デフォルトで、テキストレスポンスが返されます。音声レスポンスを返すように Amazon Lex V2 を設定した場合、このテキストは音声レスポンスの生成に使用されます。各テキストレスポンスイベントには、ボットがユーザーに返すメッセージオブジェクトの配列が含まれています。
- AudioResponseEvent – `TextResponseEvent` で生成されたテキストから合成された音声レスポンスを含みます。音声レスポンスイベントを受信するには、音声レスポンスを提供するように Amazon Lex V2 を設定する必要があります。すべての音声レスポンスイベントで同じ音声の形式になります。各イベントには 100 バイト以下の音声のチャンクが含まれます。Amazon Lex V2 は、空の音声のチャンクを `bytes` フィールドセットで `null` に送り、アプリケーションに音声レスポンスイベントの終了を示します。
- PlaybackInterruptionEvent – ボットがアプリケーションに送信したレスポンスをユーザーが中断すると、Amazon Lex V2 はこのイベントをトリガーしてレスポンスの再生を停止します。
- HeartbeatEvent - Amazon Lex V2 はこのイベントを定期的に返して、アプリケーションとボットの間の接続がタイムアウトしないようにします。

Amazon Lex V2 ボットを使用する際の音声会話イベントの時系列

次の図表は、ユーザーと Amazon Lex V2 ボット間のストリーミングの会話音声を示しています。アプリケーションは継続的に音声をボットにストリーミングし、ボットは音声からユーザー入力を探します。この例では、ユーザーとボットの両方が音声を使用して通信しています。各図表は、ユーザーの発話とその発話に対するボットの反応に対応しています。

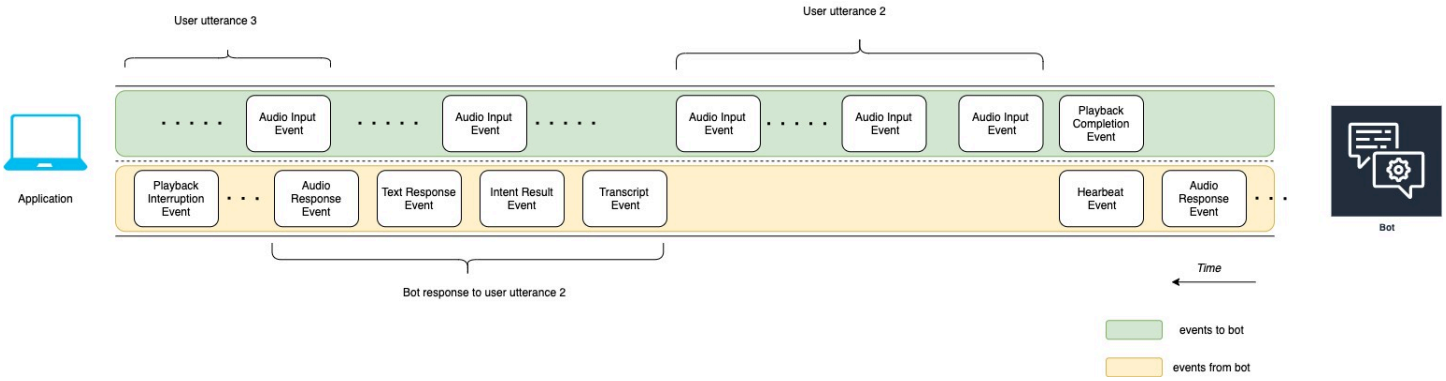
次の図表は、アプリケーションとボットの間の会話の始まりを示しています。ストリーミングは時間軸ゼロ (t0) から始まります。



次のリストは、前の図表のイベントについて説明しています。

- t0: アプリケーションはボットに設定イベントを送信してストリーミングをスタートします。
- t1: アプリケーションは音声データをストリーミングします。このデータは、アプリケーションからの一連の入力イベントに分割されます。
- t2: ユーザーの発話 1 では、ユーザーが話し始めたときにボットが音声入力イベントを検出します。
- t2: ユーザーが話している間、ボットはハートビートイベントを送信して接続を維持します。接続がタイムアウトしないように、これらのイベントを断続的に送信します。
- t3: ボットはユーザーの発話の終わりを検出します。
- t4: ボットは、ユーザーのスピーチのトランスクリプトを含むトランスクリプトイベントをアプリケーションに送信します。これが ユーザーの発話に対するボットのレスポンス 1 の始まりです。
- t5: ボットは、ユーザーが実行したいアクションを示すインテントの結果イベントを送信します。
- t6: ボットは、テキストレスポンスイベントでレスポンスをテキストとして提供し始めます。
- t7: ボットは、ユーザーのために再生するよう、一連の音声レスポンスイベントをアプリケーションに送信します。
- t8: ボットは別のハートビートイベントを送信して、断続的に接続を維持します。

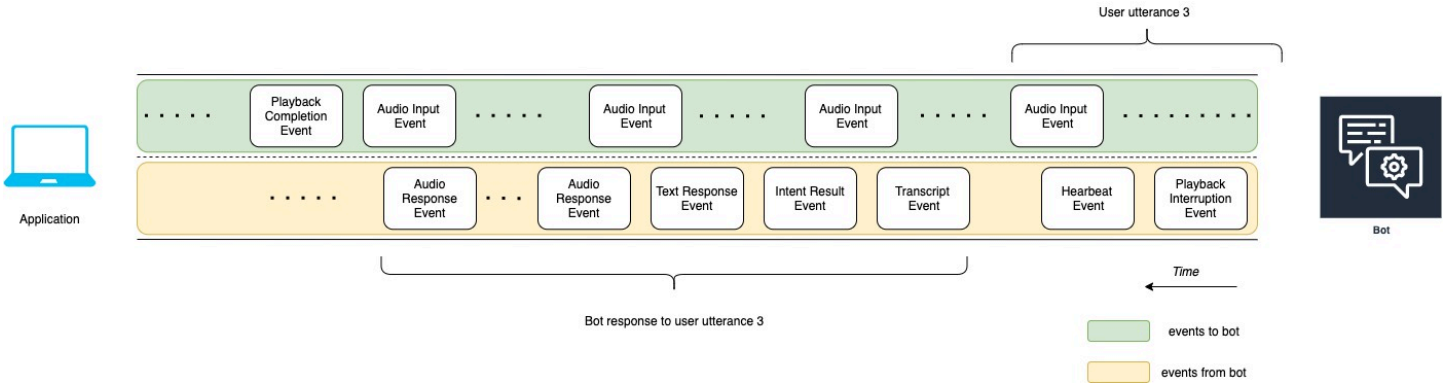
次の図表は、前の図表の続きです。これは、ユーザーのために音声レスポンスの再生を停止したことを示すために、ボットに再生完了イベントを送信するアプリケーションを示します。アプリケーションは ユーザーの発話に対するボットのレスポンス 1 をユーザーに再生します。ユーザーは ユーザー発話に対するボット応答 1 に対して ユーザーの発話 2 で応答します。



次のリストは、前の図表のイベントについて説明しています。

- t10: アプリケーションは、ユーザーにボットのメッセージの再生が終了したことを示す再生完了イベントを送信します。
- t11: アプリケーションはユーザーのレスポンスである ユーザーの発話 2 をボットに送信します。
- t12: ユーザーの発話に対するボットのレスポンス 2 では、ボットはユーザーの発話の終了まで待機し、その後音声レスポンスの提供を開始します。
- t13: ボットは ユーザーの発話に対するボットのレスポンス 2 をアプリケーションに送信している間、ユーザーの発話 3 のスタートを検出します。ボットは ユーザーの発話に対するボットのレスポンス 2 を停止し、再生中断イベントを送信します。
- t14: ボットは、ユーザーがプロンプトを中断したことを知らせるために、再生中断イベントをアプリケーションに送信します。

次の図表は、ユーザーの発話に対するボットのレスポンス 3 であり、ボットがユーザーの発話に応答した後も会話が続くことを示しています。



API を使用してストリーミングの会話をスタートする

Amazon Lex V2 ボットへのストリーミングをスタートすると、次のタスクが実行されます。

1. サーバーへの初期接続を作成します。
2. セキュリティ認証情報およびボットの詳細を設定します。ボットの詳細には、ボットが DTMF と音声入力、またはテキスト入力を受け取るかどうかが含まれています。
3. イベントをサーバーに送信します。これらのイベントは、ユーザーからのテキストデータまたは音声データです。
4. サーバーから送信されたイベントを処理します。このステップでは、ボットの出力がテキストまたはスピーチとしてユーザーに提示されるかどうかを決定します。

次のコード例では、Amazon Lex V2 ボットとローカルマシンとのストリーミングの会話を初期化します。必要に応じてコードを変更できます。

次のコードは、 AWS SDK for Java を使用したリクエストの例であり、ボットへの接続をスタートし、ボットの詳細と認証情報を設定します。

```
package com.lex.streaming.sample;

import software.amazon.awssdk.auth.credentials.AwsBasicCredentials;
import software.amazon.awssdk.auth.credentials.AwsCredentialsProvider;
import software.amazon.awssdk.auth.credentials.StaticCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.lexruntimev2.LexRuntimeV2AsyncClient;
import software.amazon.awssdk.services.lexruntimev2.model.ConversationMode;
import software.amazon.awssdk.services.lexruntimev2.model.StartConversationRequest;

import java.net.URISyntaxException;
```

```
import java.util.UUID;
import java.util.concurrent.CompletableFuture;

/**
 * The following code creates a connection with the Amazon Lex bot and configures the
 bot details and credentials.
 * Prerequisite: To use this example, you must be familiar with the Reactive streams
 programming model.
 * For more information, see
 * https://github.com/reactive-streams/reactive-streams-jvm.
 * This example uses AWS SDK for Java for Amazon Lex V2.
 * <p>
 * The following sample application interacts with an Amazon Lex bot with the streaming
 API. It uses the Audio
 * conversation mode to return audio responses to the user's input.
 * <p>
 * The code in this example accomplishes the following:
 * <p>
 * 1. Configure details about the conversation between the user and the Amazon Lex bot.
 These details include the conversation mode and the specific bot the user is speaking
 with.
 * 2. Create an events publisher that passes the audio events to the Amazon Lex bot
 after you establish the connection. The code we provide in this example tells your
 computer to pick up the audio from
 * your microphone and send that audio data to Amazon Lex.
 * 3. Create a response handler that handles the audio responses from the Amazon Lex
 bot and plays back the audio to you.
 */
public class LexBidirectionalStreamingExample {

    public static void main(String[] args) throws URISyntaxException,
 InterruptedException {
        String botId = "";
        String botAliasId = "";
        String localeId = "";
        String accessKey = "";
        String secretKey = "";
        String sessionId = UUID.randomUUID().toString();
        Region region = Region.region_name; // Choose an AWS Region where the Amazon
 Lex Streaming API is available.

        AwsCredentialsProvider awsCredentialsProvider = StaticCredentialsProvider
                .create(AwsBasicCredentials.create(accessKey, secretKey));
```

```
        // Create a new SDK client. You need to use an asynchronous client.
        System.out.println("step 1: creating a new Lex SDK client");
        LexRuntimeV2AsyncClient lexRuntimeServiceClient =
LexRuntimeV2AsyncClient.builder()
                .region(region)
                .credentialsProvider(awsCredentialsProvider)
                .build();


        // Configure the bot, alias and locale that you'll use to have a conversation.
        System.out.println("step 2: configuring bot details");
        StartConversationRequest.Builder startConversationRequestBuilder =
StartConversationRequest.builder()
                .botId(botId)
                .botAliasId(botAliasId)
                .localeId(localeId);

        // Configure the conversation mode of the bot. By default, the
        // conversation mode is audio.
        System.out.println("step 3: choosing conversation mode");
        startConversationRequestBuilder =
startConversationRequestBuilder.conversationMode(ConversationMode.AUDIO);

        // Assign a unique identifier for the conversation.
        System.out.println("step 4: choosing a unique conversation identifier");
        startConversationRequestBuilder =
startConversationRequestBuilder.sessionId(sessionId);

        // Start the initial request.
        StartConversationRequest startConversationRequest =
startConversationRequestBuilder.build();

        // Create a stream of audio data to the Amazon Lex bot. The stream will start
after the connection is established with the bot.
        EventsPublisher eventsPublisher = new EventsPublisher();

        // Create a class to handle responses from bot. After the server processes the
user data you've streamed, the server responds
        // on another stream.
        BotResponseHandler botResponseHandler = new
BotResponseHandler(eventsPublisher);

        // Start a connection and pass in the publisher that streams the audio and
process the responses from the bot.
```

```
        System.out.println("step 5: starting the conversation ...");
        CompletableFuture<Void> conversation =
 lexRuntimeServiceClient.startConversation(
                startConversationRequest,
                eventsPublisher,
                botResponseHandler);

        // Wait until the conversation finishes. The conversation finishes if the
 dialog state reaches the "Closed" state.
        // The client stops the connection. If an exception occurs during the
 conversation, the
        // client sends a disconnection event.
        conversation.whenComplete((result, exception) -> {
            if (exception != null) {
                eventsPublisher.disconnect();
            }
        });

        // The conversation finishes when the dialog state is closed and last prompt
 has been played.
        while (!botResponseHandler.isConversationComplete()) {
            Thread.sleep(100);
        }

        // Randomly sleep for 100 milliseconds to prevent JVM from exiting.
        // You won't need this in your production code because your JVM is
        // likely to always run.
        // When the conversation finishes, the following code block stops publishing
 more data and informs the Amazon Lex bot that there is no more data to send.
        if (botResponseHandler.isConversationComplete()) {
            System.out.println("conversation is complete.");
            eventsPublisher.stop();
        }
    }
}
```

次のコードは、 AWS SDK for Java を使用したリクエストの例であり、ボットにイベントを送信します。この例のコードは、コンピュータのマイクを使用して音声イベントを送信します。

```
package com.lex.streaming.sample;
```

```
import org.reactivestreams.Publisher;
import org.reactivestreams.Subscriber;
import
 software.amazon.awssdk.services.lexruntimev2.model.StartConversationRequestEventStream;

/**
 * You use the Events publisher to send events to the Amazon Lex bot. When you
 establish a connection, the bot uses the
 * subscribe() method and enables the events publisher starts sending events to
 * your computer. The bot uses the "request" method of the subscription to make more
 requests. For more information on the request method, see https://github.com/reactive-
streams/reactive-streams-jvm.
 */
public class EventsPublisher implements Publisher<StartConversationRequestEventStream>
 {

    private AudioEventsSubscription audioEventsSubscription;

    @Override
    public void subscribe(Subscriber<? super StartConversationRequestEventStream>
 subscriber) {
        if (audioEventsSubscription == null) {

            audioEventsSubscription = new AudioEventsSubscription(subscriber);
            subscriber.onSubscribe(audioEventsSubscription);

        } else {
            throw new IllegalStateException("received unexpected subscription
 request");
        }
    }

    public void disconnect() {
        if (audioEventsSubscription != null) {
            audioEventsSubscription.disconnect();
        }
    }

    public void stop() {
        if (audioEventsSubscription != null) {
            audioEventsSubscription.stop();
        }
    }
```

```
    public void playbackFinished() {
        if (audioEventsSubscription != null) {
            audioEventsSubscription.playbackFinished();
        }
    }
}
```

次のコードは、 AWS SDK for Java を使用したリクエストの例であり、ボットからのレスポンスを処理します。この例のコードは、Amazon Lex V2 が音声レスポンスを再生するように設定します。

```
package com.lex.streaming.sample;

import javazoom.jl.decoder.JavaLayerException;
import javazoom.jl.player.advanced.AdvancedPlayer;
import javazoom.jl.player.advanced.PlaybackEvent;
import javazoom.jl.player.advanced.PlaybackListener;
import software.amazon.awssdk.core.async.SdkPublisher;
import software.amazon.awssdk.services.lexruntimev2.model.AudioResponseEvent;
import software.amazon.awssdk.services.lexruntimev2.model.DialogActionType;
import software.amazon.awssdk.services.lexruntimev2.model.IntentResultEvent;
import software.amazon.awssdk.services.lexruntimev2.model.PlaybackInterruptionEvent;
import software.amazon.awssdk.services.lexruntimev2.model.StartConversationResponse;
import
 software.amazon.awssdk.services.lexruntimev2.model.StartConversationResponseEventStream;
import
 software.amazon.awssdk.services.lexruntimev2.model.StartConversationResponseHandler;
import software.amazon.awssdk.services.lexruntimev2.model.TextResponseEvent;
import software.amazon.awssdk.services.lexruntimev2.model.TranscriptEvent;

import java.io.IOException;
import java.io.UncheckedIOException;
import java.util.concurrent.CompletableFuture;

/**
 * The following class is responsible for processing events sent from the Amazon Lex
 bot. The bot sends multiple audio events,
 * so the following code concatenates those audio events and uses a publicly available
 Java audio player to play out the message to
 * the user.
```

```
 */
public class BotResponseHandler implements StartConversationResponseHandler {

    private final EventsPublisher eventsPublisher;

    private boolean lastBotResponsePlayedBack;
    private boolean isDialogStateClosed;
    private AudioResponse audioResponse;


    public BotResponseHandler(EventsPublisher eventsPublisher) {
        this.eventsPublisher = eventsPublisher;
        this.lastBotResponsePlayedBack = false;// At the start, we have not played back
 last response from bot.
        this.isDialogStateClosed = false; // At the start, the dialog state is open.
    }

    @Override
    public void responseReceived(StartConversationResponse startConversationResponse) {
        System.out.println("successfully established the connection with server.
 request id:" + startConversationResponse.responseMetadata().requestId()); // would
 have 2XX, request id.
    }

    @Override
    public void onEventStream(SdkPublisher<StartConversationResponseEventStream>
 sdkPublisher) {

        sdkPublisher.subscribe(event -> {
            if (event instanceof PlaybackInterruptionEvent) {
                handle((PlaybackInterruptionEvent) event);
            } else if (event instanceof TranscriptEvent) {
                handle((TranscriptEvent) event);
            } else if (event instanceof IntentResultEvent) {
                handle((IntentResultEvent) event);
            } else if (event instanceof TextResponseEvent) {
                handle((TextResponseEvent) event);
            } else if (event instanceof AudioResponseEvent) {
                handle((AudioResponseEvent) event);
            }
        });
    }

    @Override
```

```
    public void exceptionOccurred(Throwable throwable) {
        System.err.println("got an exception:" + throwable);
    }

    @Override
    public void complete() {
        System.out.println("on complete");
    }

    private void handle(PlaybackInterruptionEvent event) {
        System.out.println("Got a PlaybackInterruptionEvent: " + event);
    }

    private void handle(TranscriptEvent event) {
        System.out.println("Got a TranscriptEvent: " + event);
    }



    private void handle(IntentResultEvent event) {
        System.out.println("Got an IntentResultEvent: " + event);
        isDialogStateClosed =
DialogActionType.CLOSE.equals(event.sessionState().dialogAction().type());
    }

    private void handle(TextResponseEvent event) {
        System.out.println("Got an TextResponseEvent: " + event);
        event.messages().forEach(message -> {
            System.out.println("Message content type:" + message.contentType());
            System.out.println("Message content:" + message.content());
        });
    }

    private void handle(AudioResponseEvent event) {//Synthesize speech
        // System.out.println("Got a AudioResponseEvent: " + event);
        if (audioResponse == null) {
            audioResponse = new AudioResponse();
            //Start an audio player in a different thread.
            CompletableFuture.runAsync(() -> {
                try {
                    AdvancedPlayer audioPlayer = new AdvancedPlayer(audioResponse);

                    audioPlayer.setPlayBackListener(new PlaybackListener() {
                        @Override
                        public void playbackFinished(PlaybackEvent evt) {
```

```
                            super.playbackFinished(evt);

                            // Inform the Amazon Lex bot that the playback has
 finished.
                            eventsPublisher.playbackFinished();
                            if (isDialogStateClosed) {
                                lastBotResponsePlayedBack = true;
                            }
                        }
                    });
                    audioPlayer.play();
                } catch (JavaLayerException e) {
                    throw new RuntimeException("got an exception when using audio
 player", e);
                }
            });
        }

        if (event.audioChunk() != null) {
            audioResponse.write(event.audioChunk().asByteArray());
        } else {
            // The audio audio prompt has ended when the audio response has no
            // audio bytes.
            try {
                audioResponse.close();
                audioResponse = null;  // Prepare for the next audio prompt.
            } catch (IOException e) {
                throw new UncheckedIOException("got an exception when closing the audio
 response", e);
            }
        }
    }

    // The conversation with the Amazon Lex bot is complete when the bot marks the
 Dialog as DialogActionType.CLOSE
    // and any prompt playback is finished. For more information, see
    // https://docs.aws.amazon.com/lexv2/latest/dg/API_runtime_DialogAction.html.
    public boolean isConversationComplete() {
        return isDialogStateClosed && lastBotResponsePlayedBack;
    }

}
```

音声で入力イベントに応答するようにボットを設定するには、最初に Amazon Lex V2 の音声イベントを受信し、ユーザーからの入力イベントに対する音声レスポンスを提供するようにボットを設定する必要があります。

次のコードは、Amazon Lex V2 からオーディオイベントをサブスクライブする AWS SDK for Java 例です。

```
package com.lex.streaming.sample;

import org.reactivestreams.Subscriber;
import org.reactivestreams.Subscription;
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.services.lexruntimev2.model.AudioInputEvent;
import software.amazon.awssdk.services.lexruntimev2.model.ConfigurationEvent;
import software.amazon.awssdk.services.lexruntimev2.model.DisconnectionEvent;
import software.amazon.awssdk.services.lexruntimev2.model.PlaybackCompletionEvent;
import
 software.amazon.awssdk.services.lexruntimev2.model.StartConversationRequestEventStream;

import javax.sound.sampled.AudioFormat;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.DataLine;
import javax.sound.sampled.LineUnavailableException;
import javax.sound.sampled.TargetDataLine;
import java.io.IOException;
import java.io.UncheckedIOException;
import java.nio.ByteBuffer;
import java.util.Arrays;
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.LinkedBlockingQueue;
import java.util.concurrent.atomic.AtomicLong;

public class AudioEventsSubscription implements Subscription {
    private static final AudioFormat MIC_FORMAT = new AudioFormat(8000, 16, 1, true,
 false);
    private static final String AUDIO_CONTENT_TYPE = "audio/lpcm; sample-rate=8000;
 sample-size-bits=16; channel-count=1; is-big-endian=false";
    //private static final String RESPONSE_TYPE = "audio/pcm; sample-rate=8000";
```

```
    private static final String RESPONSE_TYPE = "audio/mpeg";
    private static final int BYTES_IN_AUDIO_CHUNK = 320;
    private static final AtomicLong eventIdGenerator = new AtomicLong(0);

    private final AudioInputStream audioInputStream;
    private final Subscriber<? super StartConversationRequestEventStream> subscriber;
    private final EventWriter eventWriter;
    private CompletableFuture eventWriterFuture;


    public AudioEventsSubscription(Subscriber<? super
StartConversationRequestEventStream> subscriber) {
        this.audioInputStream = getMicStream();
        this.subscriber = subscriber;
        this.eventWriter = new EventWriter(subscriber, audioInputStream);
        configureConversation();
    }

    private AudioInputStream getMicStream() {
        try {
            DataLine.Info dataLineInfo = new DataLine.Info(TargetDataLine.class,
MIC_FORMAT);
            TargetDataLine targetDataLine = (TargetDataLine)
AudioSystem.getLine(dataLineInfo);

            targetDataLine.open(MIC_FORMAT);
            targetDataLine.start();

            return new AudioInputStream(targetDataLine);
        } catch (LineUnavailableException e) {
            throw new RuntimeException(e);
        }
    }

    @Override
    public void request(long demand) {
        // If a thread to write events has not been started, start it.
        if (eventWriterFuture == null) {
            eventWriterFuture = CompletableFuture.runAsync(eventWriter);
        }
        eventWriter.addDemand(demand);
    }

    @Override
```

```
    public void cancel() {
        subscriber.onError(new RuntimeException("stream was cancelled"));
        try {
            audioInputStream.close();
        } catch (IOException e) {
            throw new UncheckedIOException(e);
        }
    }

    public void configureConversation() {
        String eventId = "ConfigurationEvent-" +
String.valueOf(eventIdGenerator.incrementAndGet());

        ConfigurationEvent configurationEvent = StartConversationRequestEventStream
                .configurationEventBuilder()
                .eventId(eventId)
                .clientTimestampMillis(System.currentTimeMillis())
                .responseContentType(RESPONSE_TYPE)
                .build();

        System.out.println("writing config event");
        eventWriter.writeConfigurationEvent(configurationEvent);
    }

    public void disconnect() {

        String eventId = "DisconnectionEvent-" +
String.valueOf(eventIdGenerator.incrementAndGet());

        DisconnectionEvent disconnectionEvent = StartConversationRequestEventStream
                .disconnectionEventBuilder()
                .eventId(eventId)
                .clientTimestampMillis(System.currentTimeMillis())
                .build();

        eventWriter.writeDisconnectEvent(disconnectionEvent);

        try {
            audioInputStream.close();
        } catch (IOException e) {
            throw new UncheckedIOException(e);
        }

    }
```

```
    //Notify the subscriber that we've finished.
    public void stop() {
        subscriber.onComplete();
    }

    public void playbackFinished() {
        String eventId = "PlaybackCompletion-" +
String.valueOf(eventIdGenerator.incrementAndGet());

        PlaybackCompletionEvent playbackCompletionEvent =
StartConversationRequestEventStream
                .playbackCompletionEventBuilder()
                .eventId(eventId)
                .clientTimestampMillis(System.currentTimeMillis())
                .build();

        eventWriter.writePlaybackFinishedEvent(playbackCompletionEvent);
    }

    private static class EventWriter implements Runnable {
        private final BlockingQueue<StartConversationRequestEventStream> eventQueue;
        private final AudioInputStream audioInputStream;
        private final AtomicLong demand;
        private final Subscriber subscriber;

        private boolean conversationConfigured;

        public EventWriter(Subscriber subscriber, AudioInputStream audioInputStream) {
            this.eventQueue = new LinkedBlockingQueue<>();

            this.demand = new AtomicLong(0);
            this.subscriber = subscriber;
            this.audioInputStream = audioInputStream;
        }

        public void writeConfigurationEvent(ConfigurationEvent configurationEvent) {
            eventQueue.add(configurationEvent);
        }

        public void writeDisconnectEvent(DisconnectionEvent disconnectionEvent) {
            eventQueue.add(disconnectionEvent);
        }
```

```
        public void writePlaybackFinishedEvent(PlaybackCompletionEvent
playbackCompletionEvent) {
            eventQueue.add(playbackCompletionEvent);
        }

        void addDemand(long l) {
            this.demand.addAndGet(l);
        }

        @Override
        public void run() {
            try {

                while (true) {
                    long currentDemand = demand.get();

                    if (currentDemand > 0) {
                        // Try to read from queue of events.
                        // If nothing is in queue at this point, read the audio events
directly from audio stream.
                        for (long i = 0; i < currentDemand; i++) {

                            if (eventQueue.peek() != null) {
                                subscriber.onNext(eventQueue.take());
                                demand.decrementAndGet();
                            } else {
                                writeAudioEvent();
                            }
                        }
                    }
                }
            } catch (InterruptedException e) {
                throw new RuntimeException("interrupted when reading data to be sent to
server");
            } catch (Exception e) {
                e.printStackTrace();
            }
        }

        private void writeAudioEvent() {
            byte[] bytes = new byte[BYTES_IN_AUDIO_CHUNK];

            int numBytesRead = 0;
            try {
```

```
                numBytesRead = audioInputStream.read(bytes);
                if (numBytesRead != -1) {
                    byte[] byteArrayCopy = Arrays.copyOf(bytes, numBytesRead);

                    String eventId = "AudioEvent-" +
 String.valueOf(eventIdGenerator.incrementAndGet());

                    AudioInputEvent audioInputEvent =
 StartConversationRequestEventStream
                            .audioInputEventBuilder()

 .audioChunk(SdkBytes.fromByteBuffer(ByteBuffer.wrap(byteArrayCopy)))
                            .contentType(AUDIO_CONTENT_TYPE)
                            .clientTimestampMillis(System.currentTimeMillis())
                            .eventId(eventId).build();

                    //System.out.println("sending audio event:" + audioInputEvent);
                    subscriber.onNext(audioInputEvent);
                    demand.decrementAndGet();
                    //System.out.println("sent audio event:" + audioInputEvent);
                } else {
                    subscriber.onComplete();
                    System.out.println("audio stream has ended");
                }

            } catch (IOException e) {
                System.out.println("got an exception when reading from audio stream");
                System.err.println(e);
                subscriber.onError(e);
            }
        }
    }
}
```

次の AWS SDK for Java 例では、入力イベントに音声応答を提供するように Amazon Lex V2 ボットを設定します。

```
package com.lex.streaming.sample;

import java.io.IOException;
```

```
import java.io.InputStream;
import java.io.UncheckedIOException;
import java.util.Optional;
import java.util.concurrent.LinkedBlockingQueue;
import java.util.concurrent.TimeUnit;

public class AudioResponse extends InputStream{

    // Used to convert byte, which is signed in Java, to positive integer (unsigned)
    private static final int UNSIGNED_BYTE_MASK = 0xFF;
    private static final long POLL_INTERVAL_MS = 10;

    private final LinkedBlockingQueue<Integer> byteQueue = new LinkedBlockingQueue<>();

    private volatile boolean closed;

    @Override
    public int read() throws IOException {
        try {
            Optional<Integer> maybeInt;
            while (true) {
                maybeInt = Optional.ofNullable(this.byteQueue.poll(POLL_INTERVAL_MS,
 TimeUnit.MILLISECONDS));

                // If we get an integer from the queue, return it.
                if (maybeInt.isPresent()) {
                    return maybeInt.get();
                }

                // If the stream is closed and there is nothing queued up, return -1.
                if (this.closed) {
                    return -1;
                }
            }
        } catch (InterruptedException e) {
            throw new IOException(e);
        }
    }

    /**
     * Writes data into the stream to be offered on future read() calls.
     */
    public void write(byte[] byteArray) {
        // Don't write into the stream if it is already closed.
```

```
        if (this.closed) {
            throw new UncheckedIOException(new IOException("Stream already closed when
 attempting to write into it."));
        }

        for (byte b : byteArray) {
            this.byteQueue.add(b & UNSIGNED_BYTE_MASK);
        }
    }

    @Override
    public void close() throws IOException {
        this.closed = true;
        super.close();
    }
}
```

イベントストリームエンコード

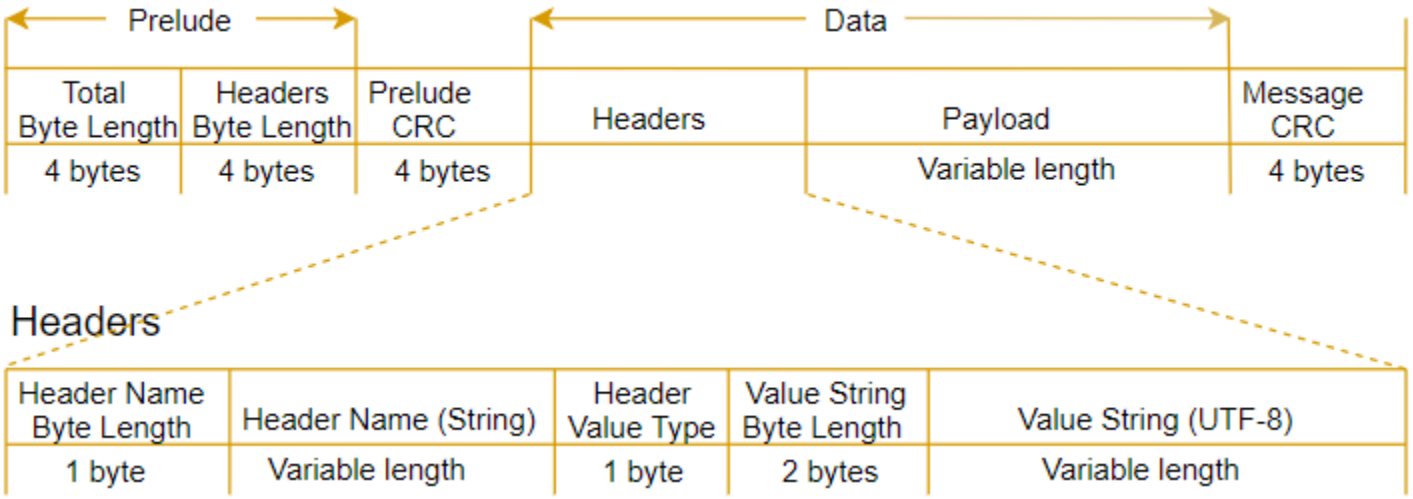
イベントストリームエンコードは、クライアントとサーバーの間のメッセージを使用して双方向通信を提供します。Amazon Lex V2 のストリーミングサービスに送信されるデータフレームは、この形式でエンコードされます。Amazon Lex V2 からのレスポンスでも、このエンコードが使用されます。

各メッセージは、prelude と data の 2 つのセクションで構成されています。prelude セクションには、メッセージの全バイト長と、すべてのヘッダーの合計バイト長が含まれています。data セクションには、ヘッダーとペイロードが含まれています。

各セクションは、4 バイトのビッグエンディアン整数 CRC チェックサムで終わります。メッセージ CRC チェックサムには、prelude セクションと data セクションが含まれています。Amazon Lex V2 は、CRC32 (別名 GZIP CRC32) を使用して、両方の CRC を計算します。CRC32 の詳細については、「GZIP ファイル形式仕様バージョン 4.3」を参照してください。

合計メッセージオーバーヘッド (prelude と両方のチェックサムを含む) は 16 バイトです。

次の図は、メッセージとヘッダーを構成するコンポーネントを示します。メッセージごとに複数のヘッダーがあります。



各メッセージには、以下のコンポーネントが含まれます。

- Prelude: 常に 8 バイト の固定サイズ (4 バイト × 2 フィールド)。
  - 最初の 4 バイト: 合計バイト長です。メッセージ全体のビッグエンディアン整数バイト長です (4 バイト長のフィールド自体を含む)。
  - 次の 4 バイト: ヘッダーバイト長。メッセージのヘッダー部分のビッグエンディアン整数バイト長 (ヘッダー長フィールド自体を除く)。
- Prelude CRC: メッセージの prelude 部分の 4 バイト CRC チェックサム (CRC 自体を除く)。バッファオーバーランなどのエラーを引き起こすことなく、Amazon Lex V2 で、破損したバイト長の情報を即時に検出できるように、prelude にはメッセージ CRC とは別の CRC があります。
- ヘッダー: メッセージの種類、コンテンツの種類など、メッセージに注釈を付けるメタデータ。メッセージには複数のヘッダーがあります。ヘッダーは、キーが UTF-8 文字列であるキーと値のペアです。ヘッダーは、メッセージのヘッダー部分に任意の順序で表示することができ、任意のヘッダーは一度だけ表示することができます。必要なヘッダータイプについては、以下のセクションを参照してください。
- ペイロード: Amazon Lex に送信される音声またはテキストコンテンツ。
- メッセージ CRC: メッセージの先頭からチェックサムの先頭までの 4 バイトの CRC チェックサム。つまり、CRC を除き、メッセージ内のすべてのものを含みます。

各ヘッダーには、以下のコンポーネントが含まれます。フレームごとに複数のヘッダーがあります。

- ヘッダー名のバイト長: ヘッダー名のバイトの長さ。
- ヘッダー名: ヘッダータイプを示すヘッダの名前。有効な値については、次のフレームの説明を参照してください。

- ヘッダー値のタイプ: ヘッダー値の種類を示す列挙。
- 値の文字列のバイト長: ヘッダー値の文字列のバイト長。
- ヘッダー値: ヘッダー文字列の値。このフィールドの有効な値は、ヘッダーのタイプによって異なります。有効な値については、次のフレームの説明を参照してください。

ユーザーによる Amazon Lex V2 ボットの中断を可能にする

Amazon Lex V2 ボットとアプリケーションの間で双方向の音声ストリーミングをスタートすると、プロンプトを送信している間にユーザー入力を聞き取るようにボットを設定できます。この入力は割り込みとして設定できます。これにより、ユーザーはボットが再生を完了する前にプロンプトを中断できます。この設定は、CVV コードの入力を求められる場合など、ユーザーが質問に対する回答をすでに知っている可能性がある状況に使用できます。

ボットは、アプリケーションが `PlaybackCompletion` イベントを送信する前にユーザー入力を検出すると、ユーザーがプロンプトを中断したことを知ることができます。ユーザーがボットを中断すると、ボットは `PlaybackInterruptionEvent` を送信します。

デフォルトで、ユーザーはボットがアプリケーションにストリーミングしているプロンプトを中断できます。この設定は Amazon Lex V2 コンソールで変更できます。

スロットを編集して、プロンプトに対するユーザーの応答方法を変更できます。スロットはインテントの一部であり、ユーザーがお客様の必要とする情報を提供する手段です。各スロットには、その情報を提供するための、ユーザーへのプロンプトが表示されます。スロットの詳細については、「Amazon Lex V2 のコア概念」を参照してください。

ユーザーがプロンプトを中断できるかどうかを変更するには (コンソール)

1. AWS マネジメントコンソールにサインインし、Amazon Lex V2 コンソールで Amazon Lex V2 コンソールV2を開きます。
2. [ボット] で、ボットを選択します。
3. [言語] で、ボットの言語を選択します。
4. [View intents (インテントの表示)] を選択します。
5. インテントを選択します。
6. [スロット] で、スロットを選択します。
7. [詳細オプション] で、[スロットプロンプト] を選択します。
8. [その他のプロンプトオプション] を選択します。

9. [ユーザーは、プロンプトが読み込まれるときにプロンプトを中断できます] を選択または選択解除します。

この機能をテストするには、2 つのスロットを持つボットを作成し、ユーザーが 1 つのスロットのプロンプトを中断できないように指定します。中断可能なプロンプトを中断すると、ボットは再生中断イベントを送信します。中断が不可能なプロンプトを中断しても、プロンプトは引き続き再生されます。

一時停止中に、Amazon Lex V2 ボットによるユーザーの追加の情報提供の待機を可能にする

Amazon Lex V2 ボットからアプリケーションへの双方向のストリーミングをスタートするときに、ユーザーが追加情報を提供するのを待機するようにボットを設定できます。ユーザーがプロンプトに応答する準備ができていない場合があります。たとえば、財布が別の部屋にあるため、ユーザーがクレジットカード情報を提供する準備ができていない場合があります。

Amazon Lex V2 ボットの [Wait and continue (待機と続行)] の機能を使用して、ユーザーは「ちょっと待って」などのフレーズを言うことで、ボットに情報の発見と提供を待機させることができます。この機能を有効にすると、ボットはユーザーに情報提供をするよう定期的にリマインダーを送信します。書き起こすユーザーの発話がないため、トランスクリプトイベントは返送されません。

Amazon Lex V2 ボットは、ストリーミングの会話を自動的に管理します。この機能を有効にするのに追加のコードを書き込む必要はありません。ボットがユーザーから待機するように求められたら、`state` の `Intent` は `Waiting` であり、`type` の `DialogAction` は `ElicitSlot` です。この情報を使用して、必要に応じてアプリケーションをカスタマイズできます。たとえば、ユーザーがクレジットカードを探しているときに音楽を再生するようにアプリケーションを設定できます。

個々のスロットに対して、待機と続行の機能を有効にします。スロットの詳細については、「Amazon Lex V2 のコア概念」を参照してください。

待機と続行を可能にするには

1. AWS マネジメントコンソールにサインインし、Amazon Lex V2 コンソールで Amazon Lex V2 コンソールV2を開きます。
2. [ボット] で、ボットを選択します。
3. [言語] で、ボットの言語を選択します。
4. [View intents (インテントの表示)] を選択します。

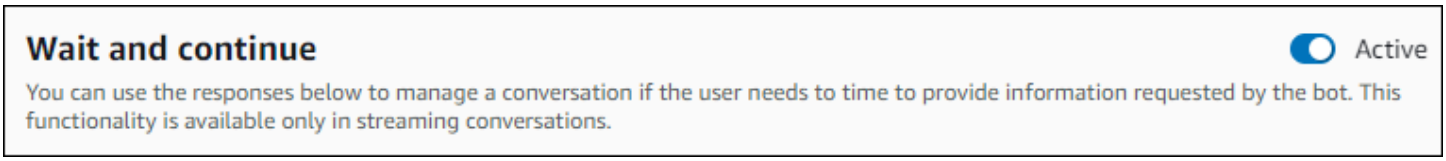
5. インテントを選択します。
6. [スロット] で、スロットを選択します。
7. [詳細オプション] で、[待機と続行] を選択します。
8. [待機と続行] で、次のフィールドを指定します。

   - [ユーザーがボットを待機させるときのレスポンス] - これは、ユーザーが追加情報を待機するように要求したときにボットが応答する方法です。
   - [ユーザーがボットを待機させ続ける必要がある場合のレスポンス] - これは、ボットが情報をまだ待機していることをユーザーに知らせるために送信するレスポンスです。ボットがユーザーにリマインドする頻度を変更できます。
   - [ユーザーが続行したいときのレスポンス] - これは、ユーザーが要求された情報を持っているときのボットのレスポンスです。

すべてのボットのレスポンスに対して、複数のレスポンスのバリエーションを用意でき、そのうちの 1 つをランダムにユーザーに提示することができます。また、ユーザーがこれらのレスポンスを中断できるかどうかを選択することもできます。

待機と継続の機能をテストするには、ユーザー入力を待機し、Amazon Lex V2 ボットへのストリーミングをスタートするようにボットを設定します。ボットへのストリーミングの詳細については、「API を使用してストリーミングの会話をスタートする」を参照してください。

待機と続行のレスポンスをオフにする必要がある場合があります。アクティブ なトグルを使用して、待機と続行のレスポンスを使用するかどうかを設定します。



> **Note**
>
> 待機と続行は、ca-ES、de-AT、de-DE、en-AU、en-GB、en-IN、en-US、en-ZA、es-419、es-ES、es-US、fr-CA、fr-FR、it-IT、ja-JP、ko-KR、pt-BR、pt-PT、zh-CN、zh-HK の各ロケールで利用できます。

Lex V2 ボットのフルフィルメント進行状況の更新の設定

インテントのフルフィルメント Lambda 関数が呼び出されると、ボットは関数が完了するまでレスポンスを送信しません。Lambda 関数の完了に数秒以上かかる場合、ユーザーはボットが応答しないと思うかもしれません。これに対処するには、フルフィルメント Lambda 関数の実行中にユーザーにアップデートを送信するようにボットを設定して、ボットが引き続きリクエストを処理していることをユーザーに知ることができます。

インテントにフルフィルメントアップデートを追加すると、ボットはフルフィルメントの開始時とフルフィルメントの進行中に定期的に応答します。スタートレスポンスを設定するときに、ボットがレスポンスを送信するまでの遅延を指定できます。これにより、フルフィルメントが比較的早く完了しないケースをサポートできます。アップデートレスポンスを構成するときは、アップデートを送信する頻度を指定します。また、フルフィルメント関数の実行時間を制限するようにタイムアウトを構成します。

フルフィルメント後のレスポンスをボットに追加することもできます。これにより、ボットはフルフィルメントの成功、失敗、またはタイムアウトのどちらによって異なるレスポンスを送信できます。

フルフィルメントのアップデートは、StartConversation オペレーションを使用してボットと対話する場合のみに使用されます。StartConversation、RecognizeText、および RecognizeUtterance オペレーションでボットと対話する際に、ポストフルフィルメントアップデートを使用することができます。

フルフィルメントのアップデート

フルフィルメントのアップデートは、Lambda 関数がインテントを満たしている間に送信されます。フルフィルメントのアップデートを有効にすると、フルフィルメントの開始時に送信されるスタートレスポンスと、フルフィルメントの進行中に定期的に送信されるアップデートレスポンスを提供します。

アップデートレスポンスを指定するときは、フルフィルメント機能を実行できる期間を決定するタイムアウトも指定します。タイムアウトの長さは、最大 15 分 (900 秒) まで指定できます。

コンソールで `active` を [false] に設定するか、CreateIntent または UpdateIntent オペレーションを使用してフルフィルメントアップデートをオフにすると、フルフィルメント更新に指定されたタイムアウトは使用されず、代わりにデフォルトのタイムアウトである 30 秒が使用されます。

フルフィルメント機能がタイムアウトした場合、Amazon Lex V2 は次の 3 つのうちいずれかを行います。

- フルフィルメント後のレスポンスが構成され、アクティブ — タイムアウトレスポンスを返します。
- フルフィルメント後のレスポンスが構成されており、アクティブではない場合 - 例外を返します。
- フルフィルメント後のレスポンスが構成されていません - 例外を返します。

レスポンスを開始

Amazon Lex V2 は、ストリーミング会話中に Lambda フルフィルメント関数が呼び出されたときにスタートレスポンスを返します。通常、インテントの実行には時間がかかり、待たなければならないことをユーザーに伝えます。`RecognizeText` あるいは `RecognizeUtterance` の操作を行った場合、スタートレスポンスは返されません。

最大 5 つのレスポンスメッセージを指定できます。Amazon Lex V2 は、ユーザーへのメッセージを 1 つ選んで再生します。

Lambda 関数が呼び出されてからスタートレスポンスが返されるまでの遅延時間を設定できます。遅延が完了する前に Lambda 関数が作業を完了した場合、スタートレスポンスは返されません。

コンソール内の `active` トグルまたは FulfillmentUpdatesSpecification 構体を使用して、フルフィルメント後のレスポンスのオン/オフを切り替えることができます。`active` が [false] の場合、スタートレスポンスは再生されません。

レスポンスをアップデートする

Amazon Lex は、Lambda フルフィルメント機能の実行中に、ストリーミング会話で定期的にアップデート応答を返します。`RecognizeText` または `RecognizeUtterance` の操作では、アップデートレスポンスは再生されません。アップデートレスポンスが再生される頻度を設定できます。たとえば、フルフィルメント関数の実行中に 30 秒ごとにアップデートレスポンスを再生して、プロセスが実行中であり、引き続き待機する必要があることをユーザーに知らせることができます。

最大 5 つのアップデートメッセージを指定できます。Amazon Lex V2 は、ユーザーへのメッセージを選んで再生します。複数のメッセージを使用すると、アップデートが繰り返されるのを防ぎます。

フルフィルメント Lambda 関数の実行中にユーザーが音声、DTMF、またはテキストを介して入力を提供した場合、Amazon Lex V2 はユーザーにアップデートレスポンスを返します。

Lambda 関数が最初のアップデート期間が終了する前に作業を完了した場合、アップデートレスポンスは返されません。

コンソール内の active トグルまたは FulfillmentUpdatesSpecification 構体を使用して、更新レスポンスのオン/オフを切り替えることができます。メトリック active が [false] の場合、アップデートレスポンスは返されません。

フルフィルメント後のレスポンス

Amazon Lex V2 は、フルフィルメント関数の終了時にフルフィルメント後のレスポンスを返します。フルフィルメント後のレスポンスは、ストリーミング会話をするだけでなく、あらゆるインテントを満たすときに使用できます。フルフィルメント後のレスポンスでは、機能が完了したこととその結果をユーザーに知らせます。

コンソール内の active トグルまたは PostFulfillmentStatusSpecification 構造を使用して、フルフィルメント後のレスポンスのオン/オフを切り替えることができます。active が [false] の場合、レスポンスは再生されません。

フルフィルメント後のレスポンスには、次の 3 種類があります。

- 成功 — フルフィルメント Lambda 関数の作業が正常に完了したときに返されます。フルフィルメント後のレスポンスがアクティブでない場合。Amazon Lex V2 は、次に設定されたアクションを実行します。
- タイムアウト — 設定されたタイムアウト期間が経過する前に Lambda 関数が作業を完了しなかった場合に返されます。フルフィルメント後のレスポンスがアクティブでない場合、Amazon Lex V2 は例外を返します。
- 失敗 - Lambda 関数がレスポンスで Failed ステータスを返すとき、または Amazon Lex V2 がインテントを遂行する際にエラーに遭遇したときに返されます。フルフィルメント後のレスポンスがアクティブでない場合、Amazon Lex V2 は例外を返します。

各タイプのメッセージは最大 5 つまで指定できます。Amazon Lex V2 は、ユーザーへのメッセージを 1 つ選んで再生します。

フルフィルメント開始応答やフルフィルメント更新応答とは異なり、フルフィルメント後の応答は、ストリーミング会話と非ストリーミング会話の両方を再生します。

また、Lambda 関数がフルフィルメント後のメッセージを返すように設定することで、これらのメッセージを上書きすることもできます。

> **Note**
>
> インテントに終了応答がある場合、フルフィルメント後の応答の後に返されます。

Lex V2 のフルフィルメント後の例

フルフィルメント後の対応について理解を深めるために、*BookTrip* ボットを例にとってみましょう。*BookFlight* インテントには、顧客のフライトを航空会社で予約するフルフィルメント Lambda 関数が設定されています。*BookFlight* のスロットが解放されると、Amazon Lex V2 はフルフィルメント Lambda 関数を呼び出します。フルフィルメントプロセスでは、次の 3 つの結果のいずれかで設定できます。

- 成功 — フライトは正常に予約されました。
- タイムアウト — 設定したフルフィルメント Lambda 実行時間よりも予約処理に時間がかかっています (たとえば、割り当てられた時間内に航空会社に連絡できない場合)。
- 失敗 — 別の理由で予約が失敗しました。

フルフィルメント後の対応を活用することで、こうした状況において顧客により有意義な応答を提供することができます。それぞれの状況の例は次のとおりです。

- 成功の応答 —「チケットの予約が完了し、確認メールを送信しました。ご不明な点がございましたら、そのメールに記載されている連絡先情報を使用して、お気軽にお問い合わせください。」
- タイムアウトび応答 —「システムが込み合っているため、チケットの予約に予想以上に時間がかかっています。お客様のリクエストが処理待ちになっています。このリクエストに対応する参照番号を記載したメールを送信しました。チケットを予約したら、予約確認書をお送りします。ご不明な点がございましたら、そのメールに記載されている連絡先情報を使用して、お気軽にお問い合わせください。」

  > **Note**
  >
  > タイムアウトメッセージを設定しない場合、Amazon Lex V2 はユースケースに対応する 4XX エラーをスローします。

- 失敗の応答 —「申し訳ございません。チケットを予約できませんでした。予約時に発生した問題の詳細を記載したメールを送信しました。」

Lex V2 ボットでのユーザー入力キャプチャのタイムアウトの設定

Amazon Lex V2 ストリーミング API を使用すると、ボットはユーザー入力の発話を自動的に検出できます。インテントまたはスロットを作成するときに、発話の最大持続時間、ユーザー入力待機中のタイムアウト、DTMF 入力の終了文字など、発話の側面を設定できます。ユースケースに合わせてボットの動作をカスタマイズできます。例えば、クレジットカード番号の桁数を 16 に制限できます。

ボットとの会話を開始するときにセッション属性を使用してタイムアウトを設定し、必要に応じて Lambda 関数で上書きすることもできます。

属性の設定キーには次の構文を使用します。

```
x-amz-lex:<InputType>:<BehaviorName>:<IntentName>:<SlotName>
```

InputType は、**audio**、**dtmf**、あるいは **text** のいずれかです。

インテントやスロット名として指定することで、ボット内のすべての * インテントやスロットにデフォルトの設定を行うことができます。インテント - またはスロット固有の設定は、デフォルト設定よりも優先されます。

Amazon Lex V2 は、方法を管理するための定義済みのセッション属性を提供します。StartConversation オペレーションは、ボットへのテキスト、音声、または DTMF 入力で動作します。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。

* をインテント、スロットまたはスロット名として指定することで、ボット内のすべてのインテント、スロット、またはサブスロットにデフォルトの設定を行うことができます。インテントまたはスロット固有の設定は、デフォルト設定よりも優先されます。以下のすべてのタイムアウトにこれらのパターンを使用してください。

複合スロットのサブスロットの場合は、. で区切ることができます。例えば、次のようになります。

```
<slotName>.<subSlotName>
```

```
x-amz-lex:allow-interrupt:<intentName>:<slotName>.<subSlotName>
```

式	シナリオ
Intent:Slot.SubSlot	「Slot」という名前の複合スロット内の「SubSlot」という名前のサブスロットにのみ適用されます
Intent:Slot.*	「Slot」という名前の複合スロット内の任意のサブスロットにのみ適用されます
Intent:*.SubSlot	複合スロット内の「SubSlot」という名前のサブスロットにのみ適用されます
Intent:*.*	複合スロット内の任意のサブスロットにのみ適用されます

Lex V2 ボットでの割り込み動作の仕組み

ボットの割り込み動作を設定できます。属性は Amazon Lex V2 で定義されます。

割り込みを許可

```
x-amz-lex:allow-interrupt:<intentName>:<slotName>
```

Amazon Lex V2 ボットが再生するプロンプトをユーザーが中断できるかどうかを定義します。選択的にオフにすることができます。

デフォルト: True

音声入力のタイムアウトの設定

セッション属性を使用して、ボットとの音声インタラクションのタイムアウト値を設定できます。属性は Amazon Lex V2 で定義されます。これらの属性により、Amazon Lex V2 がお客様の発話を終えてから入力音声を収集するまでの時間を指定することができます。

これらの属性はすべて `x-amz-lex:audio` 名前空間にあります。

発話の最大長さ

```
x-amz-lex:audio:max-length-ms:<intentName>:<slotName>
```

Amazon Lex V2 が、音声入力を切り捨ててアプリケーションに音声を返すまでの待ち時間を定義します。長時間の回答が予想される場合や、お客様に情報を提供する時間を長く取りたい場合などは、入力の長さを長くすることができます。

デフォルト: 55,000 ミリ秒 (55 秒) 。最大値は 55,000 ミリ秒 (55 秒) です

`max-length-ms` 属性を 55,000 ミリ秒以上に設定した場合、値はデフォルトで 55,000 ミリ秒になります。

音声タイムアウト

```
x-amz-lex:audio:start-timeout-ms:<intentName>:<slotName>
```

ボットが顧客は話さないと見なすまでのに待ち時間。お客様が話す前に情報を探したり、思い出したりするのに時間がかかるような場合には、時間を長くすることができます。例えば、お客様がクレジットカードを取り出す時間を設けて、番号を入力できるようにするなどです。

デフォルト = 4,000 ミリ秒 (4 秒)

無音タイムアウト

```
x-amz-lex:audio:end-timeout-ms:<intentName>:<slotName>
```

お客様が発話を止めてから、ボットが発話の終了と判断するまでの時間。入力している間の無音の時間を見込む場合は、時間を延長できます。

デフォルト = 600 ミリ秒 (0.6 秒)

音声入力を許可

```
x-amz-lex:allow-audio-input:<intentName>:<slotName>
```

この属性を有効にして、ボットが音声モダリティ経由でのみユーザー入力を受け付けるようにすることができます。このフラグが false に設定されている場合、ボットは音声入力を受け付けません。デフォルトでは、値は true に設定されます。

デフォルト: True

テキスト入力のタイムアウト

次のセッション属性を使用して、ボットのテキスト会話モードでの動作を指定します。

この属性は x-amz-lex:text 名前空間。

スタートタイムアウトしきい値

```
x-amz-lex:text:start-timeout-ms:<intentName>:<slotName>
```

テキスト入力を再度促すまでのボットの待機時間。お客様がテキスト入力をする前に、情報を探したり思い出したりする時間を確保したい場合には、時間を長くすることができます。例えば、お客様が注文した商品の詳細を確認する時間を長く取りたい場合などです。あるいは、より早くお客様にプロンプトを出すために、しきい値を下げることも可能です。

デフォルト = 30,000 ミリ秒 (30 秒)

DTMF 入力の設定

次のセッション属性を使用して、音声会話を使用しているときに Amazon Lex V2 ボットが DTMF 入力に応答する方法を指定します。

これらの属性はすべて x-amz-lex:dtmf 名前空間にあります。

削除文字

```
x-amz-lex:dtmf:deletion-character:<intentName>:<slotName>
```

蓄積された DTMF ディジットをクリアし、直ちに入力を終了させる DTMF 文字です。

デフォルト: *

終了文字

```
x-amz-lex:dtmf:end-character:<intentName>:<slotName>
```

入力をすぐに終了する DTMF 文字。ユーザーがこの文字を押さなければ、終了タイムアウト後に入力が終了します。

デフォルト= #

終了タイムアウト

```
x-amz-lex:dtmf:end-timeout-ms:<intentName>:<slotName>
```

最後の DTMF 文字入力から、入力が終了したと判断するまでのボットの待ち時間。

デフォルト = 5000 ミリ秒 (5 秒)

発話あたりの許可される DTMF の最大ディジット数

```
x-amz-lex:dtmf:max-length:<intentName>:<slotName>
```

発話で許可される DTMF ディジットの最大数。例えば、この値を 16 に設定して、クレジットカード番号に入力できる文字数を制限できます。この値を増やすことはできません。

デフォルト = 1,024 文字

DTMF 入力を許可

ボットが受け付ける入力のタイプは、セッション属性を使用して設定できます。属性は Amazon Lex V2 で定義されます。

```
x-amz-lex:allow-dtmf-input:<intentName>:<slotName>
```

この属性を有効にして、ボットが DTMF モダリティ経由でのみユーザー入力を受け付けるようにすることができます。このフラグが false に設定されている場合、ボットは DTMF 入力を受け付けません。デフォルトでは、値は true に設定されます。

デフォルト: True

Lex V2 でのボットのインポートとエクスポート

ボット定義、ボットロケール、またはカスタム語彙をエクスポートし、再度インポートして新しいリソースを作成するか、AWS アカウントの既存のリソースを上書きできます。例えば、テストアカウントからボットをエクスポートし、本番アカウントにそのボットのコピーを作成することができます。ある AWS リージョンから別のリージョンにボットをコピーすることもできます。

エクスポートしたリソースは、インポートする前に変更することができます。例えば、ボットをエクスポートしてから、スロットの JSON ファイルを編集して、特定のスロットからスロット値を引き出す発話を追加または削除できます。定義の編集が終了したら、修正したファイルをインポートすることができます。

トピック

- Lex V2 からのボットのエクスポート
- Lex V2 でのボットのインポート
- インポートまたはエクスポート時のパスワードの使用
- Lex V2 でボットをインポートおよびエクスポートするための JSON 形式

Lex V2 からのボットのエクスポート

コンソールまたは `CreatExport` オペレーションを使用し、ボット、ボットロケール、またはカスタム語彙をエクスポートします。エクスポートするリソースを指定し、エクスポートを開始する際に、.zip ファイルを保護するための任意のパスワードを指定することができます。.zip ファイルをダウンロードした後、パスワードを使用してファイルにアクセスしないと、使用することができません。詳細については、「インポートまたはエクスポート時のパスワードの使用」を参照してください。

エクスポート は非同期オペレーションです。エクスポートを開始したら、コンソールまたは `DescribeExport` オペレーションでエクスポートの進行状況をモニタリングすることができます。エクスポートが完了すると、コンソールまたは `DescribeExport` オペレーションでステータスが `COMPLETED` と表示され、コンソールがエクスポートされた .zip ファイルをブラウザにダウンロードします。`DescribeExport` オペレーションにより、Amazon Lex V2 はエクスポート結果をダウンロードできる署名済みの Amazon S3 URL を提供します。ダウンロード URL は 5 分間だけ利用可能ですが、再度 `DescribeExport` オペレーションを行うことで、新しい URL を取得することができます。

リソースのエクスポート履歴は、コンソールまたは ListExports オペレーションで確認することができます。その結果、現在のステータスとエクスポートが表示されます。エクスポートは 7 日間履歴に残ります。

Draft バージョンのボットまたはボットロケールをエクスポートする場合、エクスポート中に Draft バージョンのボットまたはボットロケールが変更されている可能性があるため、JSON ファイル内の定義に一貫性がない状態になる可能性があります。エクスポート中に Draft バージョンが変更され、その変更内容がエクスポートファイルに含まれない場合があります。

ボットロケールをエクスポートすると、Amazon Lex は、ロケール、カスタム語彙、インテント、スロットタイプ、スロットなど、ロケールを定義するすべての情報をエクスポートします。

ボットをエクスポートすると、Amazon Lex は、インテント、スロットタイプ、スロットなど、ボットに定義されたすべてのロケールをエクスポートします。次の項目は、ぼっととともにエクスポートされません。

- ボットエイリアス
- ボットに関連するロール ARN
- ボットとボットのエイリアスに関連するタグ
- ボットエイリアスに関連する Lambda コードフック

ロール ARN とタグは、ボットをインポートする際にリクエストパラメータとして入力されます。必要に応じて、インポート後にボットエイリアスを作成し、Lambda コードフックを割り当てる必要があります。

コンソールまたは DeleteExport オペレーションを使用して、エクスポートと関連する .zip ファイルを削除することができます。

コンソールを使用してボットをエクスポートする例については、「Lex V2 ボットのエクスポート (コンソール)」を参照してください。

Lex V2 でボットをエクスポートするために必要な IAM 権限

ボット、ボットロケール、およびカスタムボキャブラリーをエクスポートするには、エクスポートを実行するユーザーが次の IAM 権限を持っている必要があります。

API	• 必須 IAM アクション	[リソース]
CreateExport	• CreateExport	ボット

API	• 必須 IAM アクション	[リソース]
UpdateExport	• UpdateExport	ボット
DescribeExport	• DescribeExport • DescribeBot • DescribeCustomVocabulary • DescribeLocale • DescribeIntent • DescribeSlot • DescribeSlotType • ListLocale • ListIntent • ListSlot • ListSlotType	ボット
カスタムボキャブラリーの DescribeExport	• DescribeExport • DescribeCustomVocabulary	ボット
DeleteExport	• DeleteExport	ボット
ListExports	• ListExports	*

IAM ポリシーの例については、「ユーザーにボットとボットロケールのエクスポートを許可する」を参照してください。

Lex V2 ボットのエクスポート (コンソール)

ボット一覧、バージョン一覧、またはバージョンの詳細ページからボットをエクスポートできます。バージョンを選択すると、Amazon Lex V2 はそのバージョンをエクスポートします。以下の説明は、ボットの一覧からエクスポートを開始することを想定していますが、バージョンから開始する場合も手順は同じです。

コンソールを使用して、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、https://console.aws.amazon.com/lexv2/home で Amazon Lex V2 コンソールを開きます。
2. 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。
3. アクション から、エクスポート を選択します。
4. ボットのバージョン、プラットフォーム、およびエクスポート形式を選択します。
5. (オプション) .zip ファイルのパスワードを入力します。パスワードを設定することで、出力アーカイブを保護することができます。
6. [エクスポート] をクリックします。

エクスポートを開始した後、ボットの一覧に戻ります。エクスポートの進捗状況をモニタリングするには、インポート/エクスポートの履歴 リストを使用します。エクスポートのステータスが 完了 になると、コンソールは自動的に .zip ファイルをコンピュータにダウンロードします。

エクスポートを再度ダウンロードするには、[インポート/エクスポート] のリストで、エクスポートを選択し、ダウンロード を選択します。ダウンロードした .zip ファイルにパスワードを付けることができます。

ボット言語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、https://console.aws.amazon.com/lexv2/home で Amazon Lex V2 コンソールを開きます。
2. ボットの一覧から、エクスポートする言語のボットを選択します。
3. 言語を追加するから、言語を表示する を選択します。
4. すべての言語 リストから、エクスポートする言語を選択します。
5. アクション から、エクスポート を選択します。
6. ボットのバージョン、プラットフォーム、および形式を選択します。
7. (オプション) .zip ファイルのパスワードを入力します。パスワードを設定することで、出力アーカイブを保護することができます。
8. [エクスポート] をクリックします。

エクスポートを開始すると、言語のリストに戻ります。エクスポートの進捗状況をモニタリングするには、インポート/エクスポートの履歴 リストを使用します。エクスポートのステータスが 完了 になると、コンソールは自動的に .zip ファイルをコンピュータにダウンロードします。

エクスポートを再度ダウンロードするには、[インポート/エクスポート] のリストで、エクスポートを選択し、ダウンロード を選択します。ダウンロードした .zip ファイルにパスワードを付けることができます。

Lex V2 でのボットのインポート

コンソールを使用して、以前にエクスポートしたボット、ボットロケール、またはカスタム語彙をインポートするには、ローカルコンピュータ上のファイルの場所と、ファイルのロックを解除するための任意のパスワードを指定します。例については、Lex V2 ボットのインポート (コンソール)を参照してください。

API を使用する場合、リソースのインポートは 3 つのステップで行います。

1. `CreateUploadUrl` オペレーションを使用してアップロード URL を作成します。コンソールを使用する場合は、アップロード URL を作成する必要はありません。
2. リソースの定義を含む .zip ファイルをアップロードします。
3. `StartImport` オペレーションでインポートを開始します。

アップロード URL は、書き込み権限を持つ、署名付き Amazon S3 URL です。URL は生成されてから 5 分間使用可能になります。.zip ファイルをパスワードで保護する場合は、インポート開始時にパスワードを指定する必要があります。詳細については、「インポートまたはエクスポート時のパスワードの使用」を参照してください。

インポートは非同期プロセスです。コンソールまたは `DescribeImport` オペレーションを使用して、インポートの進行状況をモニタリングできます。

ボットまたはボットロケールをインポートするとき、インポートファイル内のリソース名と Amazon Lex V2 の既存のリソース名が競合することがあります。Amazon Lex V2 は、次の 3 つの方法で競合を処理することができます。

- 競合時の失敗 - インポートが停止し、インポート .zip ファイルからリソースはインポートされません。
- 上書き - Amazon Lex V2 は、インポート .zip ファイルからすべてのリソースをインポートし、既存のリソースをインポートファイルの定義で置き換えます。
- 追加 - Amazon Lex V2 は、インポート .zip ファイルからすべてのリソースをインポートし、既存のリソースにインポートファイルの定義を追加します。ボットロケールでのみ使用できます。

リソースへのインポートの一覧は、コンソールまたは ListImports オペレーションで確認することができます。インポートは 7 日間リストに残ります。コンソールまたは DescribeImport オペレーションを使用して、特定のインポートに関する詳細を確認することができます。

また、コンソールまたは DeleteImport オペレーションを使用して、インポートと関連する .zip ファイルを削除することができます。

コンソールを使用してボットをインポートする例については、「Lex V2 ボットのインポート (コンソール)」を参照してください。

インポートに必要な IAM 権限

ボット、ボットロケール、およびカスタムボキャブラリーをインポートするには、インポートを実行するユーザーが次の IAM 権限を持っている必要があります。

API	必須 IAM アクション	[リソース]
CreateUploadUrl	• CreateUploadUrl	*
ボットとボットロケールの StartImport	• StartImport • iam:PassRole • CreateBot • CreateCustomVocabulary • CreateLocale • CreateIntent • CreateSlot • CreateSlotType • UpdateBot • UpdateCustomVocabulary • UpdateLocale • UpdateIntent • UpdateSlot • UpdateSlotType • DeleteBot • DeleteCustomVocabulary	1. 新しいボットをインポートするには:ボット、ボットのエイリアス。 2. 既存のボットを上書きするには: ボット。 3. 新しいロケールをインポートするには: ボット。

API	必須 IAM アクション	[リソース]
	• DeleteLocale • DeleteIntent • DeleteSlot • DeleteSlotType	
カスタムボキャブラリーの StartImport	• StartImport • CreateCustomVocabulary • DeleteCustomVocabulary • UpdateCustomVocabulary	ボット
DescribeImport	• DescribeImport	ボット
DeleteImport	• DeleteImport	ボット
ListImports	• ListImports	*

IAM ポリシーの例については、「ユーザーにボットとボットロケールのインポートを許可する」を参照してください。

Lex V2 ボットのインポート (コンソール)

コンソールを使用して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、https://console.aws.amazon.com/lexv2/home で Amazon Lex V2 コンソールを開きます。
2. アクション から、インポート を選択します。
3. 入力ファイルで、ボットに名前を付けて、ボットを定義する JSON ファイルを含む .zip ファイルを選択します。
4. .zip ファイルがパスワードで保護されている場合は、.zip ファイルのパスワードを入力してください。アーカイブのパスワード保護は任意ですが、コンテンツの保護に役立ちます。
5. ボットのアクセス権限を定義する IAM ロールを作成または入力します。
6. ボットが児童オンラインプライバシー保護法 (COPPA) の適用を受けるかどうかを示します。

7. ボットにアイドルタイムアウトの設定を行います。値を指定しない場合は、zip ファイルの値が使用されます。.zip ファイルにタイムアウトの設定がない場合、Amazon Lex V2 はデフォルトの 300 秒 (5 分) を使用します。
8. (オプション) ボットにタグを追加します。
9. 同じ名前の既存ボットを上書きすることについて警告を出すかどうかを選択します。警告を有効にすると、インポートするボットが既存ボットを上書きしてしまう場合、警告が表示され、そのボットはインポートされません。警告を無効にすると、インポートされたボットは既存のボットを同名のボットに置き換わります。
10. [インポート] を選択します。

インポートを開始した後、ボットの一覧に戻ります。インポートの進捗状況をモニタリングするには、インポート/エクスポートの履歴 リストを使用します。インポートのステータスが 完了 になったら、ボットのリストからボットを選択して、ボットの修正または構築ができます。

ボット言語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、https://console.aws.amazon.com/lexv2/home で Amazon Lex V2 コンソールを開きます。
2. ボットの一覧から、言語をインポートするボットを選択します。
3. 言語を追加する から、言語を表示する を選択します。
4. アクション から、インポート を選択します。
5. 入力ファイル から、インポートする言語を含むファイルを選択します。.zip ファイルを保護した場合は、パスワード にパスワードを入力してください。
6. 言語で、インポートする言語を選択します。言語は、インポートファイルの言語と一致する必要はありません。ある言語から別の言語へインテントをコピーできます。
7. 音声で、音声インタラクションに使用する Amazon Polly の音声を選択するか、テキスト専用ボットを使用する場合は なし を選択します。
8. 信頼度スコアの閾値では、代替インテントを返す際に、Amazon Lex V2 が `AMAZON.FallbackIntent`、`AMAZON.KendraSearchIntent` またはその両方を挿入する閾値を入力します。
9. 既存の言語の上書きについて警告を出すかどうかを選択します。警告を有効にすると、インポートする言語が既存の言語を上書きしてしまう場合、警告が表示され、その言語はインポートされません。警告を無効にすると、インポートされた言語が既存の言語を置き換えます。
10. [インポート] を選択すると、言語のインポートが開始されます。

インポートを開始した後、言語一覧に戻ります。インポートの進捗状況をモニタリングするには、インポート/エクスポートの履歴 リストを使用します。インポートのステータスが 完了 になったら、ボットのリストから言語を選択して、ボットの修正または構築ができます。

インポートまたはエクスポート時のパスワードの使用

Amazon Lex V2 は、エクスポートアーカイブをパスワードで保護したり、標準の .zip ファイル圧縮を使用して保護されたインポートアーカイブを読み取ったりできます。インポートおよびエクスポートアーカイブは、必ずパスワードで保護する必要があります。

Amazon Lex V2 は、 S3 バケットにエクスポートアーカイブを送信し、署名付き S3 URL で利用可能になります。URL は 5 分間のみ利用可能です。アーカイブは、ダウンロード URL にアクセスすると、すべてのユーザーが利用できます。アーカイブ内のデータを保護するために、リソースをエクスポートするときにパスワードを指定します。URL の有効期限が切れた後にアーカイブを取得する必要がある場合は、コンソールまたは `DescribeExport` オペレーションで新しい URL を取得することができます。

エクスポートアーカイブのパスワードを紛失した場合は、インポート/エクスポート履歴テーブルから のダウンロード を選択するか、`UpdateExport` オペレーションにより既存のファイルに新しいパスワードを作成することができます。エクスポートの履歴テーブルで ダウンロード を選択し、パスワードを指定しない場合、Amazon Lex V2 は保護されていない zip ファイルをダウンロードします。

Lex V2 でボットをインポートおよびエクスポートするための JSON 形式

リソースの部分を記述する JSON 構造を含む .zip ファイルを使用して、Amazon Lex V2 からボット、ボットロケール、またはカスタムボキャブラリーをインポートおよびエクスポートします。リソースをエクスポートすると、Amazon Lex V2 が .zip ファイルを作成し、Amazon S3 の署名付き URL を使用して利用できるようにします。リソースをインポートする際には、JSON 構造を含む .zip ファイルを作成し、S3 の署名付き URL にアップロードする必要があります。

Amazon Lex は、ボットをエクスポートするときに、.zip ファイルに以下のようなディレクトリ構造を作成します。ボットロケールをエクスポートする際、ロケールの下の構造のみがエクスポートされます。カスタム語彙をエクスポートすると、カスタム語彙の下の構造だけがエクスポートされます。

```
BotName_BotVersion_ExportID_LexJson.zip
```

```
            -or-
BotName_BotVersion_LocaleId_ExportId_LEX_JSON.zip
        --> manifest.json
        --> BotName
        ----> Bot.json
        ----> BotLocales
        ------> Locale_A
        --------> BotLocale.json
        --------> Intents
        ----------> Intent_A
        ------------> Intent.json
        ------------> Slots
        --------------> Slot_A
        ----------------> Slot.json
        --------------> Slot_B
        ----------------> Slot.json
        ----------> Intent_B
                      ...
        --------> SlotTypes
        ----------> SlotType_A
        ------------> SlotType.json
        ----------> SlotType_B
                      ...
        --------> CustomVocabulary
        ------------> CustomVocabulary.json

        ------> Locale_B
                      ...
```

マニフェストファイル構造

マニフェストファイルには、エクスポートファイルのメタデータが含まれています。

```
{
    "metadata": {
        "schemaVersion": "1.0",
        "fileFormat": "LexJson",
        "resourceType": "Bot | BotLocale | CustomVocabulary"
    }
}
```

ボットファイル構造

ボットファイルには、ボットの設定情報が含まれています。

```
{
    "name": "BotName",
    "identifier": "identifier",
    "version": "number",
    "description": "description",
    "dataPrivacy": {
        "childDirected": true | false
    },
    "idleSessionTTLInSeconds": seconds
}
```

ボットロケールファイル構造

ボットロケールファイルには、ボットのロケールまたは言語に関する説明が含まれています。ボットをエクスポートする際、.zip ファイルに複数のボットロケールファイルが存在する可能性があります。ボットロケールをエクスポートする際、zip ファイルにはロケールが 1 つだけ含まれます。

```
{
    "name": "locale name",
    "identifier": "locale ID",
    "version": "number",
    "description": "description",
    "voiceSettings": {
        "voiceId": "voice",
        "engine": "standard | neural
    },
    "nluConfidenceThreshold": number
}
```

インテントファイル構造

インテントファイルには、インテントの設定情報が含まれています。.zip ファイルには、特定のロケールのインテントごとに 1 つのインテントファイルがあります。

以下は、サンプルの BookTrip ボットの BookCar インテントの JSON 構造の例です。インテントの JSON 構造の完全な例については、CreateIntent オペレーションを参照してください。

```
{
    "name": "BookCar",
    "identifier": "891RWHHICO",
    "description": "Intent to book a car.",
    "parentIntentSignature": null,
    "sampleUtterances": [
        {
            "utterance": "Book a car"
        },
        {
            "utterance": "Reserve a car"
        },
        {
            "utterance": "Make a car reservation"
        }
    ],
    "intentConfirmationSetting": {
        "confirmationPrompt": {
            "messageGroupList": [
                {
                    "message": {
                        "plainTextMessage": {
                            "value": "OK, I have you down for a {CarType} hire in
 {PickUpCity} from {PickUpDate} to {ReturnDate}.  Should I book the reservation?"
                        },
                        "ssmlMessage": null,
                        "customPayload": null,
                        "imageResponseCard": null
                    },
                    "variations": null
                }
            ],
            "maxRetries": 2
        },
        "declinationResponse": {
            "messageGroupList": [
                {
                    "message": {
                        "plainTextMessage": {
                            "value": "OK, I have cancelled your reservation in
 progress."
                        },
                        "ssmlMessage": null,
```

```
                        "customPayload": null,
                        "imageResponseCard": null
                    },
                    "variations": null
                }
            ]
        }
    },
    "intentClosingSetting": null,
    "inputContexts": null,
    "outputContexts": null,
    "kendraConfiguration": null,
    "dialogCodeHook": null,
    "fulfillmentCodeHook": null,
    "slotPriorities": [
        {
            "slotName": "DriverAge",
            "priority": 4
        },
        {
            "slotName": "PickUpDate",
            "priority": 2
        },
        {
            "slotName": "ReturnDate",
            "priority": 3
        },
        {
            "slotName": "PickUpCity",
            "priority": 1
        },
        {
            "slotName": "CarType",
            "priority": 5
        }
    ]
}
```

スロットファイル構造

スロットファイルには、インテント内のスロットの設定情報が含まれています。特定の口ケールのスロットごとに、.zip ファイル内に 1 つのスロットファイルがあります。

次の例は、BookTrip サンプルボットの BookCar インテントで、顧客がレンタルしたい車のタイプを選択できるスロットの JSON 構造です。スロットの JSON 構造の完全な例については、CreateSlot オペレーションを参照してください。

```
{
    "name": "CarType",
    "identifier": "KDHJWNGZGC",
    "description": "Type of car being reserved.",
    "multipleValuesSetting": {
        "allowMutlipleValues": false
    },
    "slotTypeName": "CarTypeValues",
    "obfuscationSetting": null,
    "slotConstraint": "Required",
    "defaultValueSpec": null,
    "slotValueElicitationSetting": {
        "promptSpecification": {
            "messageGroupList": [
                {
                    "message": {
                        "plainTextMessage": {
                            "value": "What type of car would you like to rent?  Our
 most popular options are economy, midsize, and luxury"
                        },
                        "ssmlMessage": null,
                        "customPayload": null,
                        "imageResponseCard": null
                    },
                    "variations": null
                }
            ],
            "maxRetries": 2
        },
        "sampleValueElicitingUtterances": null,
        "waitAndContinueSpecification": null,
    }
}
```

次の例に、複雑なスロットの JSON 構造を示します。

```
{
  "name": "CarType",
  "identifier": "KDHJWNGZGC",
```

```
  "description": "Type of car being reserved.",
  "multipleValuesSetting": {
      "allowMutlipleValues": false
  },
  "slotTypeName": "CarTypeValues",
  "obfuscationSetting": null,
  "slotConstraint": "Required",
  "defaultValueSpec": null,
  "slotValueElicitationSetting": {
      "promptSpecification": {
          "messageGroupList": [
              {
                  "message": {
                      "plainTextMessage": {
                          "value": "What type of car would you like to rent?  Our most
popular options are economy, midsize, and luxury"
                      },
                      "ssmlMessage": null,
                      "customPayload": null,
                      "imageResponseCard": null
                  },
                  "variations": null
              }
          ],
          "maxRetries": 2
      },
      "sampleValueElicitingUtterances": null,
      "waitAndContinueSpecification": null,
  },
  "subSlotSetting": {
    "slotSpecifications": {
      "firstname": {
        "valueElicitationSetting": {
          "promptSpecification": {
            "allowInterrupt": false,
            "messageGroupsList": [
              {
                "message": {
                  "imageResponseCard": null,
                  "ssmlMessage": null,
                  "customPayload": null,
                  "plainTextMessage": {
                    "value": "please provide firstname"
                  }
```

```
              },
              "variations": null
            }
          ],
          "maxRetries": 2,
          "messageSelectionStrategy": "Random"
        },
        "defaultValueSpecification": null,
        "sampleUtterances": [
          {
            "utterance": "my name is {firstName}"
          }
        ],
        "waitAndContinueSpecification": null
      },
      "slotTypeId": "AMAZON.FirstName"
    },
    "eyeColor": {
      "valueElicitationSetting": {
        "promptSpecification": {
          "allowInterrupt": false,
          "messageGroupsList": [
            {
              "message": {
                "imageResponseCard": null,
                "ssmlMessage": null,
                "customPayload": null,
                "plainTextMessage": {
                  "value": "please provide eye color"
                }
              },
              "variations": null
            }
          ],
          "maxRetries": 2,
          "messageSelectionStrategy": "Random"
        },
        "defaultValueSpecification": null,
        "sampleUtterances": [
          {
            "utterance": "eye color is {eyeColor}"
          },
          {
            "utterance": "I have eyeColor eyes"
```

```
            }
          ],
          "waitAndContinueSpecification": null
        },
        "slotTypeId": "7FEVCB2PQE"
      }
    },
    "expression": "(firstname OR eyeColor)"
  }
}
```

スロットタイプファイル構造

スロットタイプファイルには、言語またはロケールで使用されるカスタムスロットタイプの設定情報が含まれています。特定のロケールのカスタムスロットタイプごとに、.zip ファイル内に 1 つのスロットタイプがあります。

以下は、BookTrip のサンプルボットで使用可能な車類をリストしたスロットタイプの JSON 構造です。スロットタイプの JSON 構造の完全な例については、CreateSlotType オペレーションを参照してください。

```
{
    "name": "CarTypeValues",
    "identifier": "T1YUHGD9ZR",
    "description": "Enumeration representing possible types of cars available for
 hire",
    "slotTypeValues": [{
        "synonyms": null,
        "sampleValue": {
            "value": "economy"
        }
    }, {
        "synonyms": null,
        "sampleValue": {
            "value": "standard"
        }
    }, {
        "synonyms": null,
        "sampleValue": {
            "value": "midsize"
        }
    }, {
```

```
        "synonyms": null,
        "sampleValue": {
            "value": "full size"
        }
    }, {
        "synonyms": null,
        "sampleValue": {
            "value": "luxury"
        }
    }, {
        "synonyms": null,
        "sampleValue": {
            "value": "minivan"
        }
    }],
    "parentSlotTypeSignature": null,
    "valueSelectionSetting": {
        "resolutionStrategy": "TOP_RESOLUTION",
        "advancedRecognitionSetting": {
            "audioRecognitionStrategy": "UseSlotValuesAsCustomVocabulary"
        },
        "regexFilter": null
    }
}
```

次の例に、複雑なスロットタイプの JSON 構造を示します。

```
{
  "name": "CarCompositeType",
  "identifier": "TPA3CC9V",
  "description": null,
  "slotTypeValues": null,
  "parentSlotTypeSignature": null,
  "valueSelectionSetting": {
    "regexFilter": null,
    "resolutionStrategy": "CONCATENATION"
  },
  "compositeSlotTypeSetting": {
    "subSlots": [
      {
        "name": "model",
        "slotTypeId": "MODELTYPEID" # custom slot type Id for model
      },
```

```
      {
        "name": "city",
        "slotTypeId": "AMAZON.City"
      },
      {
        "name": "country",
        "slotTypeId": "AMAZON.Country"
      },
      {
        "name": "make",
        "slotTypeId": "MAKETYPEID" # custom slot type Id for make
      }
    ]
  }
}
```

以下は、カスタム文法を使用して顧客の発話を理解するスロットタイプです。詳細については、「文法スロットタイプ」を参照してください。

```
{
  "name": "custom_grammar",
  "identifier": "7KEAQIQKPX",
  "description": "Slot type using a custom grammar",
  "slotTypeValues": null,
  "parentSlotTypeSignature": null,
  "valueSelectionSetting": null,
  "externalSourceSetting": {
    "grammarSlotTypeSetting": {
      "source": {
        "kmsKeyArn": "arn:aws:kms:Region:123456789012:alias/customer-grxml-key",
        "s3BucketName": "grxml-test",
        "s3ObjectKey": "grxml_files/grammar.grxml"
      }
    }
  }
}
```

カスタム語彙ファイル構造

カスタム語彙ファイルには、単一言語または口ケールのカスタム語彙のエントリが含まれます。.zip ファイルには、カスタム語彙を含む口ケールごとに 1 つのカスタム語彙ファイルがあります。

以下は、レストランの注文を受けるボット用のカスタム語彙ファイルです。ボットには口ケールごとに 1 つのファイルがあります。

```
{
    "customVocabularyItems": [
        {
            "weight": 3,
            "phrase": "wafers"
        },
        {
            "weight": null,
            "phrase": "extra large"
        },
        {
            "weight": null,
            "phrase": "cremini mushroom soup"
        },
        {
            "weight": null,
            "phrase": "ramen"
        },
        {
            "weight": null,
            "phrase": "orzo"
        }
    ]
}
```

Lex V2 でのリソースのタグ付け

Amazon Lex V2 ボットとボットエイリアス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 リソースに付けるラベルです。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 リソースの特定と整理。多くの AWS リソースはタグ付けをサポートしているため、異なるサービスのリソースに同じタグを割り当てて、リソースが同じであること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。AWS Billing and Cost Managementダッシュボードでタグをアクティブ化します。AWS はタグを使用してコストを分類し、月単位のコスト配分レポートを提供します。Amazon Lex V2 では、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。詳細については、「AWS Billing and Cost Management ユーザーガイド」の「コスト配分タグを使用する」を参照してください。
- リソースへのアクセスの制御。Amazon Lex V2 でタグを使用して、Amazon Lex V2 リソースへのアクセスを制御するポリシーを作成できます。これらのポリシーは、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。

AWS マネジメントコンソール、、AWS Command Line Interfaceまたは Amazon Lex V2 API を使用してタグを操作できます。

コンソールまたは API を使用したリソースのタグ付け

Amazon Lex V2 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 V2 API を使用している場合は、次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。

- CreateBot および CreateBotAlias - 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。
- ListTagsForResource - リソースに関連付けられているタグの表示。

- 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 V2 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:aws:lex:${Region}:${account}:bot/${bot-id}`
- ボットエイリアス - 次のような ARN を使用します。
  - `arn:aws:lex:${Region}:${account}:bot-alias/${bot-id}/${bot-alias-id}`

`bot-id` および `bot-alias-id` 値は、10 文字の英数字の大文字文字列です。

Lex V2 を使用する場合のタグの制限

Amazon Lex V2 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。

- キーの最大数 — コンソールを使用して 50、API を使用して 200 です。
- キーの最大長 - 128 文字
- 値の最大長 - 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
- キーのプレフィックス`aws:`として を使用しないでください。AWS 用に予約されています。

リソースのタグ付け (コンソール)

コンソールを使用して、ボットまたはボットエイリア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 マネジメントコンソールにサインインし、https://console.aws.amazon.com/lex/ で Amazon Lex コンソールを開きます。
2. [ボットの作成] を選択します。
3. [詳細設定] の [ボットの設定] セクションで、[新しいタグを追加] を選択します。ボットと `TestBotAlias` エイリアスにタグを追加できます。

4. [次へ] を選択し、ボットの作成を続行します。

ボットエイリアスの作成時にタグを追加するには

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/lex/ で Amazon Lex コンソールを開きます。
2. ボットエイリアスを追加するボットを選択します。
3. 左側のメニューから、[エイリアス] を選択し、[エイリアスの作成] を選択します。
4. [一般的な情報] で、[タグ] から [新しいタグを追加] を選択します。
5. [作成] を選択します。

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 マネジメントコンソールにサインインし、https://console.aws.amazon.com/lex/ で Amazon Lex コンソールを開きます。
2. 変更するボットを選択します。
3. 左側のメニューから [設定] を選択し、[編集] を選択します。
4. [タグ] で、変更を実施します。
5. [保存] を選択して、ボットに加えた変更を保存します。

既存のエイリアスのタグを追加、削除、または変更するには、

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/lex/ で Amazon Lex コンソールを開きます。
2. 変更するボットを選択します。
3. 左側のメニューから、[エイリアス] を選択します。エイリアスのリストから、変更するエイリアスを選択します。
4. [エイリアスの詳細] の [タグ] で、[タグを変更する] を選択します。
5. [タグの管理] で、変更を実施します。
6. [保存] を選択して、エイリアスに加えた変更を保存します。

Amazon Lex V2 のセキュリティ

クラウドセキュリティは AWS の最優先事項です。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 AWS のお客様の責任共有です。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 AWS のサービスを実行するインフラストラクチャを保護する責任を担います。また、AWS では安全に使用できるサービスも用意されています。AWS コンプライアンスプログラムの一環として、サードパーティーによる監査が入り、定期的にセキュリティの有効性をテストおよび検証しています。Amazon Lex V2 に適用するコンプライアンスプログラムの詳細については、コンプライアンスプログラムによる対象範囲内の AWS のサービスをご参照ください。
- クラウド内のセキュリティ – お客様の責任はお客様が使用する AWS のサービスによって決まります。また、ユーザーは、データの機密性、会社の要件、適用される法律や規制など、その他の要因についても責任を負います。

このドキュメントは、Amazon Lex V2 使用時における責任共有モデルの適用法を理解するのに役立ちます。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 V2 を設定する方法について説明します。また、Amazon Lex V2 リソースのモニタリングや保護に役立つ、他の AWS のサービスの使用方法についても説明します。

トピック

- Amazon Lex V2 のデータ保護
- Amazon Lex V2 のための Identity and Access Management
- Amazon Lex V2 でのログ記録とモニタリング
- Amazon Lex V2 のコンプライアンス検証
- Amazon Lex V2 の耐障害性
- Amazon Lex V2 でのインフラストラクチャセキュリティ
- Amazon Lex V2 とインターフェイス VPC エンドポイント (AWS PrivateLink)

Amazon Lex V2 のデータ保護

Amazon Lex V2 は、AWS 責任共有モデルに準拠しています。これには、データ保護に関する規制とガイドラインが含まれています。AWS は、すべての AWS サービスを実行するグローバルインフラストラクチャの保護を担います。AWS では、カスタマーコンテンツと個人データを処理するためのセキュリティ設定管理など、このインフラストラクチャでホストされているデータの制御が維持されます。AWS のお客様と APN パートナーは、データコントローラーまたはデータプロセッサの役割を果たし、AWS クラウドに配置する個人データを担当します。

データ保護目的の場合、AWS アカウント認証情報を保護して AWS Identity and Access Management (IAM) で個々のユーザーアカウントをセットアップし、そのユーザーに各自の職務を果たすために必要なアクセス許可のみが付与されるようにすることをお勧めします。また、次の方法でデータを保護することもお勧めします:

- 各アカウントで多要素認証 (MFA) を使用します。
- SSL/TLS を使用して AWS リソースと通信します。

- で API とユーザーアクティビティのログ記録を設定します AWS CloudTrail。
- AWS 暗号化ソリューションを、AWS サービス内のすべてのデフォルトのセキュリティ管理と一緒に使用します。
- Amazon Macie などの高度なマネージドセキュリティサービスを使用します。これにより、Amazon S3 に保存される個人データの検出と保護が支援されます。

顧客のアカウント番号などの機密の識別情報は、[名前] フィールドなどの自由形式のフィールドに配置しないことを強くお勧めします。これは、コンソール、API、AWS CLI、または AWS SDK を使用して Amazon Lex V2 または他の AWS のサービスを使用する場合も同様です。 SDKs Amazon Lex V2 や他のサービスに入力したすべてのデータは、診断ログに取り込まれる可能性があります。外部サーバーへの URL を指定するときは、そのサーバーへのリクエストを検証するための認証情報を URL に含めないでください。

データ保護の詳細については、AWS セキュリティブログのブログ投稿「AWS 責任共有モデルと GDPR」を参照してください。

保管中の暗号化

Amazon Lex V2 は、ユーザー発話およびユーザーが保存するその他の情報を暗号化します。

トピック

- サンプル発話
- 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。この情報は保管中に暗号化され、ボットの構築とカスタマーエクスペリエンスを作成するためのみに使用されます。

セッション属性

セッション属性には、Amazon Lex V2 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 V2 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 関数がセッション属性を追加または更新した場合、Amazon Lex V2 は新しい情報をクライアントアプリケーションに返します。

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、ランタイム時 Amazon Lex V2 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。

転送中の暗号化

Amazon Lex V2 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。

Amazon Lex V2 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 V2 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、追加料金なしで使用できる AWS のサービス です。

トピック

- オーディエンス
- アイデンティティを使用した認証
- ポリシーを使用したアクセスの管理
- Amazon Lex V2 で IAM が機能する仕組み
- Amazon Lex V2 のアイデンティティベースのポリシー例
- Amazon Lex V2 内のリソースベースのポリシーの例
- Amazon Lex V2 の AWS 管理ポリシー V2
- Amazon Lex V2 のサービスリンクロール
- Amazon Lex V2 のサービスロールの使用 V2
- Amazon Lex V2 のサービスロールへの移行
- Amazon Lex V2 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

オーディエンス

AWS Identity and Access Management (IAM) の使用方法は、ロールによって異なります。

- サービスユーザー - 機能にアクセスできない場合は、管理者にアクセス許可をリクエストします (「Amazon Lex V2 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)。
- サービス管理者 - ユーザーアクセスを決定し、アクセス許可リクエストを送信します (「Amazon Lex V2 で IAM が機能する仕組み」を参照)
- IAM 管理者 - アクセスを管理するためのポリシーを作成します (「Amazon Lex V2 のアイデンティティベースのポリシー例」を参照)

アイデンティティを使用した認証

認証は、アイデンティティ認証情報を使用して AWS にサインインする方法です。、IAM ユーザー AWS アカウントのルートユーザー、または IAM ロールを引き受けることで認証される必要があります。

AWS IAM アイデンティティセンター (IAM Identity Center)、シングルサインオン認証、Google/Facebook 認証情報などの ID ソースからの認証情報を使用して、フェデレーティッド ID としてサインインできます。サインインの詳細については、「AWS サインイン ユーザーガイド」の「AWS アカウントにサインインする方法」を参照してください。

プログラムによるアクセスの場合、AWS はリクエストに暗号で署名するための SDK と CLI を提供します。詳細については、IAM ユーザーガイドの「API リクエストの AWS 署名バージョン 4」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、まず、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 AWS アカウント ルートユーザーと呼ばれる 1 つのサインインアイデンティティから始めます。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。ルートユーザー認証情報を必要とするタスクについては、「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。

フェデレーテッドアイデンティティ

ベストプラクティスとして、人間のユーザーが一時的な認証情報 AWS のサービス を使用して にアクセスするには、ID プロバイダーとのフェデレーションを使用する必要があります。

フェデレーティッド ID は、エンタープライズディレクトリ、ウェブ ID プロバイダー、または ID ソースからの認証情報 AWS のサービス を使用して Directory Service にアクセスするユーザーです。フェデレーテッドアイデンティティは、一時的な認証情報を提供するロールを引き受けます。

アクセスを一元管理する場合は、 AWS IAM アイデンティティセンターをお勧めします。詳細については、「AWS IAM アイデンティティセンター ユーザーガイド」の「IAM アイデンティティセンターとは」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、特定の個人やアプリケーションに対する特定のアクセス許可を持つアイデンティティです。長期認証情報を持つ IAM ユーザーの代わりに一時的な認証情報を使用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「ID プロバイダーとのフェデレーションを使用して一時的な認証情報を使用して AWS にアクセスする必要がある」を参照してください。

IAM グループは、IAM ユーザーの集合を指定し、大量のユーザーに対するアクセス許可の管理を容易にします。詳細については、「IAM ユーザーガイド」の「IAM ユーザーに関するユースケース」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を持つアイデンティであり、一時的な認証情報を提供します。ユーザーから IAM ロール (コンソール) に切り替えるか、AWS CLI または AWS API オペレーションを呼び出すことで、ロールを引き受けることができます。詳細については、「IAM ユーザーガイド」の「ロールを引き受けるための各種方法」を参照してください。

IAM ロールは、フェデレーションユーザーアクセス、一時的な IAM ユーザーのアクセス許可、クロスアカウントアクセス、クロスサービスアクセス、および Amazon EC2 で実行するアプリケーションに役立ちます。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。

ポリシーを使用したアクセスの管理

AWS でアクセスを制御するには、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、ID またはリソースに関連付けられている場合のアクセス許可を定義します。AWS は、プリンシパルがリクエストを行うときにこれらのポリシーを評価します。大半のポリシーは JSON ドキュメントとして AWS に保存されます。JSON ポリシードキュメントの詳細については、「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、ポリシーを使用して、どのプリンシパルがどのリソースに対して、どのような条件でアクションを実行できるかを定義することで、誰が何にアクセスできるかを指定します。

デフォルトでは、ユーザーやロールにアクセス許可はありません。IAM 管理者は IAM ポリシーを作成してロールに追加し、このロールをユーザーが引き受けられるようにします。IAM ポリシーは、オペレーションの実行方法を問わず、アクセス許可を定義します。

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のポリシーは、アイデンティティ (ユーザー、グループ、またはロール) にアタッチできる JSON アクセス許可ポリシードキュメントです。これらのポリシーは、アイデンティティがどのリソースに対してどのような条件下でどのようなアクションを実行できるかを制御します。アイデンティティベースポリシーの作成方法については、IAM ユーザーガイド の カスタマー管理ポリシーでカスタム IAM アクセス許可を定義する 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、インラインポリシー (単一の ID に直接埋め込む) または管理ポリシー (複数の ID にアタッチされたスタンドアロンポリシー) にすることができます。管理ポリシーと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「管理ポリシーとインラインポリシーのいずれかを選択する」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。例としては、IAM ロール信頼ポリシーや Amazon S3 バケットポリシーなどがあります。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。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。

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 IAM の AWS マネージドポリシーを使用することはできません。

その他のポリシータイプ

AWS は、より一般的なポリシータイプによって付与されるアクセス許可の最大数を設定できる追加のポリシータイプをサポートしています。

- アクセス許可の境界 – アイデンティティベースのポリシーで IAM エンティティに付与することのできるアクセス許可の数の上限を設定します。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 (SCP) - AWS Organizations内の組織または組織単位の最大のアクセス許可を指定します。詳細については、「AWS Organizations ユーザーガイド」の「サービスコントロールポリシー」を参照してください。
- リソースコントロールポリシー (RCP) – は、アカウント内のリソースで利用できる最大数のアクセス許可を定義します。詳細については、「AWS Organizations ユーザーガイド」の「リソースコントロールポリシー (RCP)」を参照してください。
- セッションポリシー – ロールまたはフェデレーションユーザーの一時セッションを作成する際にパラメータとして渡される高度なポリシーです。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アクセス許可を理解するのがさらに複雑になります。複数のポリシータイプが関連するとき、リクエストを許可するかどうかを AWS が決定する方法の詳細については、IAM ユーザーガイドのポリシーの評価ロジックを参照してください。

Amazon Lex V2 で IAM が機能する仕組み

IAM を使用して Amazon Lex V2 へのアクセスを管理する前に、Amazon Lex V2 で利用できる IAM の機能について学びます。

Amazon Lex V2 で使用できる IAM の機能

IAM 機能	Amazon Lex V2 サポート
アイデンティティベースのポリシー	あり
リソースベースのポリシー	あり
ポリシーアクション	あり
ポリシーリソース	あり
ポリシー条件キー	いいえ
ACL	いいえ
ABAC (ポリシー内のタグ)	あり
一時的な認証情報	いいえ
プリンシパルアクセス権限	あり
サービスロール	あり
サービスリンクロール	部分的

Amazon Lex V2 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については、IAM ユーザーガイドの「IAM と連携する AWS のサービス」を参照してください。

Amazon Lex V2 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のポリシーをサポート: はい

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ポリシーの作成方法については、「IAM ユーザーガイド」の「カスタマー管理ポリシーでカスタム IAM アクセス許可を定義する」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。JSON ポリシーで使用できるすべての要素について学ぶには、「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 V2 のアイデンティティベースのポリシー例

Amazon Lex V2 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 V2 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 V2 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をサポート: はい

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コントロールできます。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、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、ユーザー、ロール、フェデレーティッドユーザー、または AWS のサービスを含めることができます。

Amazon Lex V2 では、クロスアカウントまたはクロスリージョンポリシーを使用できません。クロスアカウントまたはクロスリージョン ARN を持つリソースのポリシーを作成すると、Amazon Lex V2 はエラーを返します。

Amazon Lex V2 サービスは、ボットポリシーとボットエイリアスポリシーと呼ばれるリソースベースのポリシーをサポートしています。このポリシーは、ボットまたはボットエイリアスにアタッチされます。これらのポリシーは、ボットまたはボットエイリアスに対してアクションを実行できるプリンシパルを定義します。

アクションは特定のリソースでのみ使用できます。例えば、`UpdateBot` アクションはボットリソースでのみ使用でき、`UpdateBotAlias` アクションは、ボットエイリアスリソースでのみ使用できます。ポリシーで指定されたリソースで使用できないアクションをポリシーで指定すると、Amazon

Lex V2 はエラーを返します。アクションの一覧と、使用できるリソースのリストについては、次の表を参照してください。

アクション	リソースベースのポリシーのサポート	[リソース]
BuildBotLocale	サポート	BotId
CreateBot	いいえ	
CreateBotAlias	いいえ	
CreateBotChannel [アクセス許可のみ]	サポート	BotId
CreateBotLocale	サポート	BotId
CreateBotVersion	サポート	BotId
CreateExport	サポート	BotId
CreateIntent	サポート	BotId
CreateResourcePolicy	サポート	BotId、BotAliasId
CreateSlot	サポート	BotId
CreateSlotType	サポート	BotId
CreateUploadUrl	いいえ	
DeleteBot	サポート	BotId、BotAliasId
DeleteBotAlias	サポート	BotAliasId
DeleteBotChannel [アクセス許可のみ]	サポート	BotId
DeleteBotLocale	サポート	BotId
DeleteBotVersion	サポート	BotId

アクション	リソースベースのポリシーのサポート	[リソース]
DeleteExport	サポート	BotId
DeleteImport	サポート	BotId
DeleteIntent	サポート	BotId
DeleteResourcePolicy	サポート	BotId、BotAliasId
DeleteSession	サポート	BotAliasId
DeleteSlot	サポート	BotId
DeleteSlotType	サポート	BotId
DescribeBot	サポート	BotId
DescribeBotAlias	サポート	BotAliasId
DescribeBotChannel [アクセス許可のみ]	サポート	BotId
DescribeBotLocale	サポート	BotId
DescribeBotVersion	サポート	BotId
DescribeExport	サポート	BotId
DescribeImport	サポート	BotId
DescribeIntent	サポート	BotId
DescribeResourcePolicy	サポート	BotId、BotAliasId
DescribeSlot	サポート	BotId
DescribeSlotType	サポート	BotId
GetSession	サポート	BotAliasId

アクション	リソースベースのポリシーのサポート	[リソース]
ListBotAliases	サポート	BotId
ListBotChannels [アクセス許可のみ]	サポート	BotId
ListBotLocales	サポート	BotId
ListBots	いいえ	
ListBotVersions	サポート	BotId
ListBuiltInIntents	いいえ	
ListBuiltIntSlotTypes	いいえ	
ListExports	いいえ	
ListImports	いいえ	
ListIntents	サポート	BotId
ListSlots	サポート	BotId
ListSlotTypes	サポート	BotId
PutSession	サポート	BotAliasId
RecognizeText	サポート	BotAliasId
RecognizeUtterance	サポート	BotAliasId
StartConversation	サポート	BotAliasId
StartImport	サポート	BotId、BotAliasId
TagResource	いいえ	
UpdateBot	サポート	BotId

アクション	リソースベースのポリシーのサポート	[リソース]
UpdateBotAlias	サポート	BotAliasId
UpdateBotLocale	サポート	BotId
UpdateBotVersion	サポート	BotId
UpdateExport	サポート	BotId
UpdateIntent	サポート	BotId
UpdateResourcePolicy	サポート	BotId、BotAliasId
UpdateSlot	サポート	BotId
UpdateSlotType	サポート	BotId
UntagResource	いいえ	

リソースベースのポリシーをボットまたはボットエイリアスにアタッチする方法については、「Amazon Lex V2 内のリソースベースのポリシーの例」を参照してください。

Amazon Lex V2 内のリソースベースのポリシーの例

Amazon Lex V2 リソースベースのポリシーの例を表示するには、「Amazon Lex V2 内のリソースベースのポリシーの例」を参照してください。

Amazon Lex V2 のポリシーアクション

ポリシーアクションをサポート: 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対してどのような条件下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。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アクセス許可を付与するポリシーで使用されます。

Amazon Lex V2 アクションのリストを確認するには、「サービス認可リファレンス」の「Amazon Lex V2 で定義されるアクション」を参照してください。

Amazon Lex V2 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"
         ]
```

Amazon Lex V2 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 V2 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 V2 のポリシーリソース

ポリシーリソースのサポート: 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のリソースに対してどのような条件下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。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。リソースレベルのアクセス許可をサポートしないアクションの場合は、ワイルドカード (*) を使用して、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します。

```
"Resource": "*"
```

Amazon Lex V2 リソースのタイプとその ARN のリストを確認するには、「サービス認可リファレンス」 の「Amazon Lex V2 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 V2 で定義されるアクション」を参照してください。

Amazon Lex V2 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 V2 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 V2 のポリシー条件キー

サービス固有のポリシー条件キーをサポート: なし

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対してどのような条件下でアクションを実行できるかということです。

`Condition` 要素は、定義された基準に基づいてステートメントが実行される時期を指定します。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して、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。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 V2 の条件キーのリストを確認するには、「サービス認可リファレンス」の「Amazon Lex V2 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 V2 で定義されるアクション」を参照してください。

Amazon Lex V2 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 V2 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 V2 のアクセスコントロールリスト (ACL)

ACLsをサポート: なし

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アクセス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 Lex V2 での属性ベースのアクセスコントロール (ABAC)

ABAC をサポート (ポリシーのタグ): はい

属性ベースのアクセス制御 (ABAC) は、タグと呼ばれる属性に基づいてアクセス許可を定義する認可戦略です。IAM エンティティと AWS リソースにタグをアタッチし、プリンシパルのタグがリソースのタグと一致するときにオペレーションを許可する ABAC ポリシーを設計できます。

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ありです。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 認可でアクセス許可を定義する」を参照してください。ABAC をセットアップする手順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。

Amazon Lex V2 での一時的な認証情報の使用

一時的な認証情報をサポート: いいえ

一時的な認証情報は、AWS リソースへの短期的なアクセスを提供し、フェデレーションまたは切り替えロールの使用時に自動的に作成されます。AWS では、長期のアクセスキーを使用する代わりに、一時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 ユーザーガイド」の「IAM の一時的な認証情報」および「IAM と連携するAWS のサービス」を参照してください。

Amazon Lex V2 のクロスサービスプリンシパルのアクセス許可

転送アクセスセッション (FAS): はい

転送アクセスセッション (FAS) は、 を呼び出すプリンシパルのアクセス許可と AWS のサービス、ダウンストリームサービス AWS のサービス へのリクエストをリクエストする を使用します。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 V2 のサービスロール

サービスロールをサポート: はい

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 のサービスに許可を委任するロールを作成する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 V2 の機能が破損する可能性があります。Amazon Lex V2 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 V2 のサービスリンクロール

サービスにリンクされたロールのサポート: 一部

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アクセス許可を表示できますが、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の作成または管理の詳細については、IAM と連携する AWS のサービスを参照してください。表の「サービスリンクロール」列に Yes と記載されたサービスを見つけます。サービスにリンクされたロールに関するドキュメントをサービスで表示するには、[はい] リンクを選択します。

Amazon Lex V2 のアイデンティティベースのポリシー例

デフォルトで、ユーザーとロールには Amazon Lex V2 リソースを作成または変更する権限がありません。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。

これらのサンプルの 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 ポリシーを作成する (コンソール)」を参照してください。

Amazon Lex V2 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、「https://docs.aws.amazon.com/service-authorization/latest/reference/list_amazonlexv2.htmlサービス認可リファレンス」の「Amazon Lex V2 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

- ポリシーに関するベストプラクティス
- Amazon Lex V2 コンソールを使用する
- ユーザーがボットに関数を追加できるようにする
- ユーザーがボットにチャネルを追加できるようにする
- ボットの作成と更新をユーザーに許可する
- 自動 Chatbot デザイナーの使用をユーザーに許可する
- ユーザーが AWS KMS キーを使用してファイルを暗号化および復号できるようにする

- ユーザーにボットの削除を許可する
- ユーザーにボットとの会話を許可する
- 特定のユーザーにリソースベースのポリシーの管理を許可する
- ユーザーにボットとボットロケールのエクスポートを許可する
- ユーザーにカスタム語彙のエクスポートを許可する
- ユーザーにボットとボットロケールのインポートを許可する
- ユーザーにカスタム語彙のエクスポートを許可する
- ユーザーが Amazon Lex から Amazon Lex V2 にボットを移行できるようにする
- 自分の権限の表示をユーザーに許可する
- Amazon Lex V2 でのビジュアル会話ビルダーを使用して、ユーザーが会話フローを描画できるようにする
- ユーザーにボットレプリカの作成と表示を許可しますが、削除は許可しない

ポリシーに関する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 V2 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では、 AWS アカウントに費用が発生する場合があります。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 – ユーザーとワークロードにアクセス許可の付与を開始するには、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することで、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 管理ポリシー」または「職務機能用の AWS 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 ポリシーでアクセス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、サービスアクション

が などの特定の を通じて使用されている場合に 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 CloudFormation。詳細については、IAM ユーザーガイド の IAM JSON ポリシー要素:条件 を参照してください。

- IAM アクセスアナライザー 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 アクセスアナライザー 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 でポリシーを検証する 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、MFA をオンにしてセキュリティを強化します。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 を使用した安全な API アクセス 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 V2 コンソールを使用する

Amazon Lex V2 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 V2 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。

ユーザーとロールが引き続き Amazon Lex V2 コンソールを使用できるようにするには、ユーザーにコンソールアクセス権が必要です。コンソールアクセスを持つユーザーの作成の詳細については、「IAM ユーザーガイド」の「AWS アカウントでの IAM ユーザーの作成」を参照してください。

ユーザーがボットに関数を追加できるようにする

この例では、IAM ユーザーが Amazon Comprehend、センチメント分析、および Amazon Kendra クエリのアクセス許可を Amazon Lex V2 ボットに追加できるようにするポリシーを示しています。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Id1",
            "Effect": "Allow",
            "Action": "iam:PutRolePolicy",
            "Resource": "arn:aws:iam::*:role/aws-service-role/
lexv2.amazonaws.com/AWSServiceRoleForLexV2Bots*"
        },
        {
            "Sid": "Id2",
            "Effect": "Allow",
            "Action": "iam:GetRolePolicy",
            "Resource": "arn:aws:iam::*:role/aws-service-role/
lexv2.amazonaws.com/AWSServiceRoleForLexV2Bots*"
        }
    ]
}
```

ユーザーがボットにチャネルを追加できるようにする

この例は、IAM ユーザーがボットにメッセージングチャネルを追加できるようにするポリシーです。ユーザーは、メッセージングプラットフォームにボットをデプロイする前に、このポリシーを設定しておく必要があります。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Id1",
            "Effect": "Allow",
            "Action": "iam:PutRolePolicy",
            "Resource": "arn:aws:iam::*:role/aws-service-role/
channels.lexv2.amazonaws.com/AWSServiceRoleForLexV2Channels*"
        },
```

```
        {
            "Sid": "Id2",
            "Effect": "Allow",
            "Action": "iam:GetRolePolicy",
            "Resource": "arn:aws:iam::*:role/aws-service-role/
channels.lexv2.amazonaws.com/AWSServiceRoleForLexV2Channels*"
        }
    ]
}
```

ボットの作成と更新をユーザーに許可する

この例では、IAM ユーザーがボットの作成と更新を許可するポリシーの例を示しています。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。

自動 Chatbot デザイナーの使用をユーザーに許可する

この例では、IAM ユーザーが自動 Chatbot デザイナーを実行できるようにするポリシーの例を示しています。

ユーザーが AWS KMS キーを使用してファイルを暗号化および復号できるようにする

この例では、IAM ユーザーが AWS KMS カスタマーマネージドキーを使用してデータを暗号化および復号化できるようにするポリシーの例を示します。

ユーザーにボットの削除を許可する

この例では、IAM ユーザーにボットの削除を許可するポリシーの例を示しています。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。

JSON

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

```
                "lex:DeleteBot",
                "lex:DeleteBotLocale",
                "lex:DeleteBotAlias",
                "lex:DeleteIntent",
                "lex:DeleteSlot",
                "lex:DeleteSlottype"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:lex:us-east-1:123412341234:bot/*",
                "arn:aws:lex:us-east-1:123412341234:bot-alias/*"
            ]
        }
    ]
}
```

ユーザーにボットとの会話を許可する

この例では、IAM ユーザーにボットとの会話を許可するポリシーの例を示しています。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。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "lex:StartConversation",
                "lex:RecognizeText",
                "lex:RecognizeUtterance",
                "lex:GetSession",
                "lex:PutSession",
                "lex:DeleteSession"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:lex:us-east-1:123412341234:bot-alias/*"
        }
    ]
}
```

特定のユーザーにリソースベースのポリシーの管理を許可する

次の例では、リソースベースのポリシーを管理する権限を特定のユーザーに付与します。これにより、ボットとボットのエイリアスに関連付けられたポリシーへのコンソールと API アクセスを許可します。

ユーザーにボットとボットロケールのエクスポートを許可する

次の IAM アクセス許可ポリシーにより、ユーザーはボットまたはボットのロケールの作成、更新、およびエクスポートの取得が可能になります。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "lex:CreateExport",
                "lex:UpdateExport",
                "lex:DescribeExport",
                "lex:DescribeBot",
                "lex:DescribeBotLocale",
                "lex:ListBotLocales",
                "lex:DescribeIntent",
                "lex:ListIntents",
                "lex:DescribeSlotType",
                "lex:ListSlotTypes",
                "lex:DescribeSlot",
                "lex:ListSlots",
                "lex:DescribeCustomVocabulary"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:lex:us-east-1:123456789012:bot/*"
            ]
        }
    ]
}
```

ユーザーにカスタム語彙のエクスポートを許可する

次の IAM アクセス許可ポリシーでは、ユーザーがボットのロケールからカスタム語彙をエクスポートできます。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "lex:CreateExport",
                "lex:UpdateExport",
                "lex:DescribeExport",
                "lex:DescribeCustomVocabulary"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:lex:us-east-1:123456789012:bot/*"
            ]
        }
    ]
}
```

ユーザーにボットとボットロケールのインポートを許可する

次の IAM アクセス許可ポリシーは、ユーザーがボットまたはボットのロケールをインポートし、インポートのステータスをチェックできるようにします。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "lex:CreateUploadUrl",
                "lex:StartImport",
                "lex:DescribeImport",
```

```
                "lex:CreateBot",
                "lex:UpdateBot",
                "lex:DeleteBot",
                "lex:CreateBotLocale",
                "lex:UpdateBotLocale",
                "lex:DeleteBotLocale",
                "lex:CreateIntent",
                "lex:UpdateIntent",
                "lex:DeleteIntent",
                "lex:CreateSlotType",
                "lex:UpdateSlotType",
                "lex:DeleteSlotType",
                "lex:CreateSlot",
                "lex:UpdateSlot",
                "lex:DeleteSlot",
                "lex:CreateCustomVocabulary",
                "lex:UpdateCustomVocabulary",
                "lex:DeleteCustomVocabulary",
                "iam:PassRole"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:lex:us-east-1:123456789012:bot/*",
                "arn:aws:lex:us-east-1:123456789012:bot-alias/*"
            ]
        }
    ]
}
```

ユーザーにカスタム語彙のエクスポートを許可する

次の IAM アクセス許可ポリシーでは、ユーザーがボットのロケールにカスタム語彙をインポートできます。

JSON

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

```
                "lex:CreateUploadUrl",
                "lex:StartImport",
                "lex:DescribeImport",
                "lex:CreateCustomVocabulary",
                "lex:UpdateCustomVocabulary",
                "lex:DeleteCustomVocabulary"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:lex:us-east-1:123456789012:bot/*"
            ]
        }
    ]
}
```

ユーザーが Amazon Lex から Amazon Lex V2 にボットを移行できるようにする

次の IAM アクセス許可ポリシーでは、ユーザーが Amazon Lex から Amazon Lex V2 へのボットの移行をスタートできます。

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 か AWS API を使用してプログラム的に、このアクションを完了する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": "*"
        }
    ]
}
```

Amazon Lex V2 でのビジュアル会話ビルダーを使用して、ユーザーが会話フローを描画できるようにする

次の IAM アクセス許可ポリシーを使用すると、ユーザーが Amazon Lex V2 でのビジュアル会話ビルダーを使用して会話フローを描画できます。

ユーザーにボットレプリカの作成と表示を許可しますが、削除は許可しない

次のアクセス許可を IAM ロールにアタッチして、ボットレプリカの作成と表示のみを許可できます。`lex:DeleteBotReplica` を省略することで、ロールがボットレプリカを削除できないようにします。詳細については、「Lex V2 でボットをレプリケートし、ボットレプリカを管理する権限」を参照してください。

Amazon Lex V2 内のリソースベースのポリシーの例

あるリソースベースのポリシーは、ボットやボットのエイリアスなどのリソースにアタッチされています。リソースベースのポリシーによって、リソースにアクセスできるユーザーと、そのユーザーが実行できるアクションを指定できます。例えば、ユーザーが特定のボットを変更できるようにするリソースベースのポリシーを追加したり、特定のボットエイリアスに対してユーザーがランタイムオペレーションを使用することができます。

リソースベースのポリシーを使用すると、他の AWS のサービスがアカウント内のリソースにアクセスすることを許可できます。例えば、Connect Customer に Amazon Lex V2 ボットへのアクセスを許可できます。

ボットまたはボットエイリアスを作成する方法については、「Amazon Lex V2 ボットの使用」を参照してください。

トピック

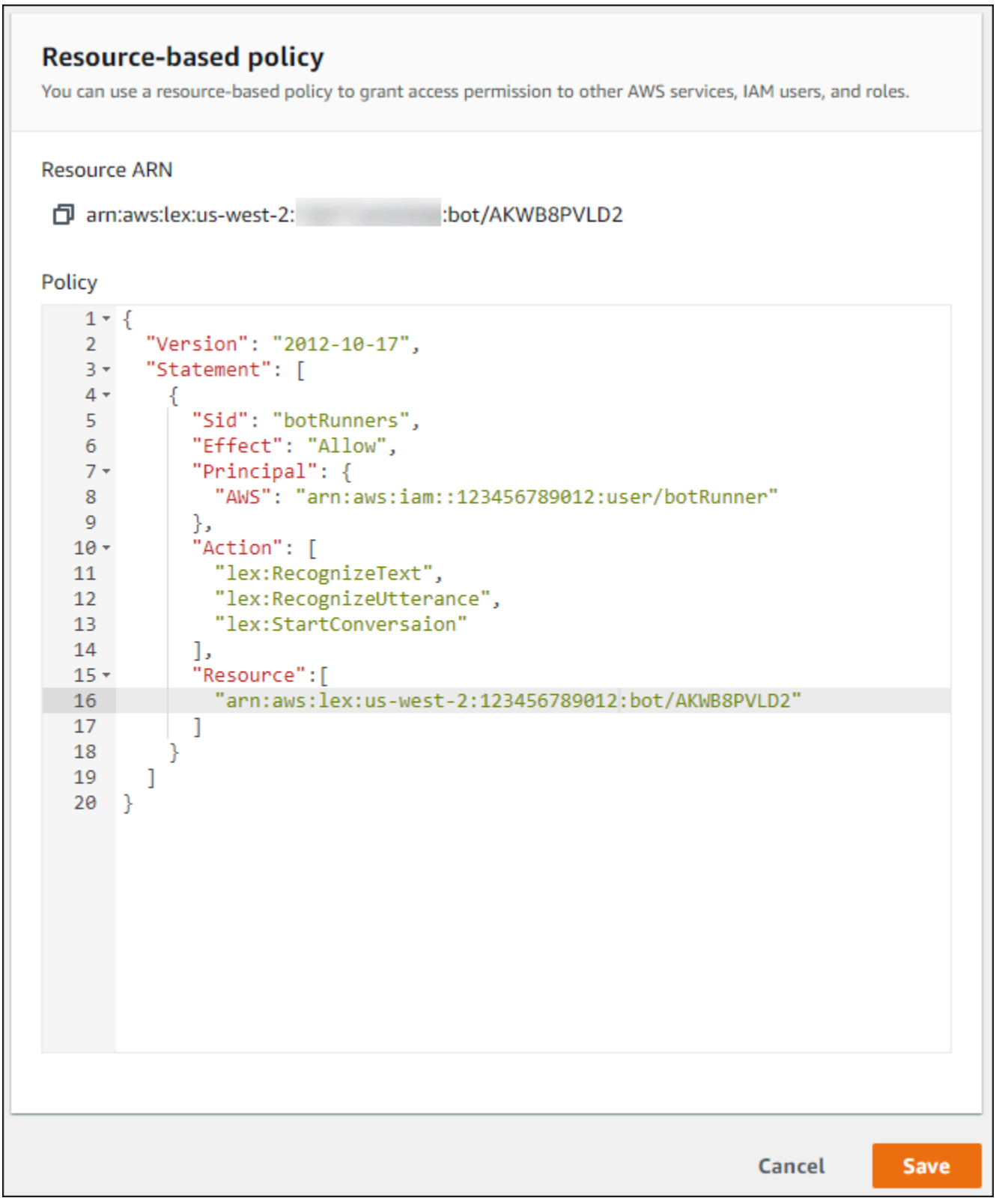
- コンソールを使用してリソースベースのポリシーを指定します。
- API を使用してリソースベースのポリシーを指定する
- IAM ロールがボットを更新し、ボットのエイリアスを一覧表示できるようにする。
- ユーザーにボットとの会話を許可する
- AWS サービスに特定の Amazon Lex V2 ボットの使用を許可する

コンソールを使用してリソースベースのポリシーを指定します。

Amazon Lex V2 コンソールを使用して、ボットとボットエイリアスのリソースベースのポリシーを管理できます。ポリシーの JSON 構造を入力すると、コンソールがそれをリソースに関連付けます。リソースにポリシーがすでに関連付けられている場合は、コンソールを使用してポリシーを表示および変更できます。

ポリシーエディタでポリシーを保存すると、コンソールはポリシーの構文をチェックします。ポリシーに、存在しないユーザーやリソースでサポートされていないアクションなどのエラーが含まれている場合は、エラーを返し、ポリシーを保存しません。

次に、コンソール内のボットのリソースベースのポリシーエディタを示します。ボットエイリアスのポリシーエディタも同様です。


Resource-based policy
You can use a resource-based policy to grant access permission to other AWS services, IAM users, and roles.
Resource ARN
arn:aws:lex:us-west-2: :bot/AKWB8PVLD2
Policy
{
"Version": "2012-10-17",
"Statement": [
{
"Sid": "botRunners",
"Effect": "Allow",
"Principal": {
"AWS": "arn:aws:iam::123456789012:user/botRunner"
},
"Action": [
"lex:RecognizeText",
"lex:RecognizeUtterance",
"lex:StartConversaion"
],
"Resource":[
"arn:aws:lex:us-west-2:123456789012:bot/AKWB8PVLD2"
]
}
]
}
Cancel
Save

ボットのポリシーエディタを開くには

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/lex/ で Amazon Lex コンソールを開きます。
2. [ボット] リストから、ポリシーを編集するボットを選択します。
3. [リソースベースのポリシー] セクションで、[編集]を選択します。

ボットエイリアスのポリシーエディタを開くには

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/lex/ で Amazon Lex コンソールを開きます。
2. [ボット] リストから、編集するエイリアスを含むボットを選択します。
3. 左側のメニューから、[エイリアス] を選択し、[エイリアスの作成] を選択します。
4. [リソースベースのポリシー] セクションで、[編集]を選択します。

API を使用してリソースベースのポリシーを指定する

API オペレーションを使用して、ボットとボットのエイリアスのリソースベースのポリシーを管理できます。ポリシーを作成、更新、および削除するオペレーションがあります。

CreateResourcePolicy

指定されたポリシーステートメントを持つ新しいリソースポリシーをボットまたはボットのエイリアスに追加します。

CreateResourcePolicyStatement

新しいリソースポリシーステートメントをボットまたはボットのエイリアスに追加します。

DeleteResourcePolicy

ボットまたはボットのエイリアスからリソースポリシーを削除します。

DeleteResourcePolicyStatement

ボットまたはボットのエイリアスからリソースポリシーステートメントを削除します。

DescribeResourcePolicy

リソースポリシーとポリシーリビジョンを取得します。

UpdateResourcePolicy

ボットまたはボットのエイリアスの既存のリソースポリシーを新しいリソースポリシーに置き換えます。

例

Java

次の例では、リソースベースのポリシーオペレーションを使用してリソースベースのポリシーを管理する方法を示しています。

```
        /*
         * Create a new policy for the specified bot alias
         * that allows a role to invoke lex:UpdateBotAlias on it.
         * The created policy will have revision id 1.
         */

        CreateResourcePolicyRequest createPolicyRequest =
                CreateResourcePolicyRequest.builder()
                        .resourceArn("arn:aws:lex:Region:123456789012:bot-
alias/MYBOTALIAS/TSTALIASID")
                        .policy("{\"Version\": \"2012-10-17\",\"Statement
\": [{\"Sid\": \"BotAliasEditor\",\"Effect\": \"Allow\",\"Principal\":
 {\"AWS\": \"arn:aws:iam::123456789012:role/BotAliasEditor\"},\"Action\":
 [\"lex:UpdateBotAlias\"],\"Resource\":[\"arn:aws:lex:Region:123456789012:bot-
alias/MYBOTALIAS/TSTALIASID\"]]}")

        lexmodelsv2Client.createResourcePolicy(createPolicyRequest);

        /*
         * Overwrite the policy for the specified bot alias with a new policy.
         * Since no expectedRevisionId is provided, this request overwrites the
 current revision.
         * After this update, the revision id for the policy is 2.
         */
        UpdateResourcePolicyRequest updatePolicyRequest =
        UpdateResourcePolicyRequest.builder()
                        .resourceArn("arn:aws:lex:Region:123456789012:bot-
alias/MYBOTALIAS/TSTALIASID")
                        .policy("{\"Version\": \"2012-10-17\",\"Statement
\": [{\"Sid\": \"BotAliasEditor\",\"Effect\": \"Deny\",\"Principal\":
 {\"AWS\": \"arn:aws:iam::123456789012:role/BotAliasEditor\"},\"Action\":
```

```
 [\"lex:UpdateBotAlias\"],\"Resource\":[\"arn:aws:lex:Region:123456789012:bot-
alias/MYBOTALIAS/TSTALIASID\"]]}")

        lexmodelsv2Client.updateResourcePolicy(updatePolicyRequest);

        /*
         * Creates a statement in an existing policy for the specified bot alias
         * that allows a role to invoke lex:RecognizeText on it.
         * This request expects to update revision 2 of the policy. The request will
 fail
         * if the current revision of the policy is no longer revision 2.
         * After this request, the revision id for this policy will be 3.
         */

        CreateResourcePolicyStatementRequest createStatementRequest =
                CreateResourcePolicyStatementRequest.builder()
                        .resourceArn("arn:aws:lex:Region:123456789012:bot-
alias/MYBOTALIAS/TSTALIASID")
                        .effect("Allow")

 .principal(Principal.builder().arn("arn:aws:iam::123456789012:role/
BotRunner").build())
                        .action("lex:RecognizeText")
                        .statementId("BotRunnerStatement")
                        .expectedRevisionId(2)
                        .build();

        lexmodelsv2Client.createResourcePolicyStatement(createStatementRequest);

        /*
         * Deletes a statement from an existing policy for the specified bot alias
 by statementId.
         * Since no expectedRevisionId is supplied, the request will remove the
 statement from
         * the current revision of the policy for the bot alias.
         * After this request, the revision id for this policy will be 4.
         */
        DeleteResourcePolicyRequest deleteStatementRequest =
                DeleteResourcePolicyRequest.builder()
                        .resourceArn("arn:aws:lex:Region:123456789012:bot-
alias/MYBOTALIAS/TSTALIASID")
                        .statementId("BotRunnerStatement")
                        .build();
```

```
        lexmodelsv2Client.deleteResourcePolicy(deleteStatementRequest);

        /*
         * Describe the current policy for the specified bot alias
         * It always returns the current revision.
         */
        DescribeResourcePolicyRequest describePolicyRequest =
                DescribeResourcePolicyRequest.builder()
                        .resourceArn("arn:aws:lex:Region:123456789012:bot-
alias/MYBOTALIAS/TSTALIASID")
                        .build();

        lexmodelsv2Client.describeResourcePolicy(describePolicyRequest);


        /*
         * Delete the current policy for the specified bot alias
         * This request expects to delete revision 3 of the policy. Since the
 revision id for
         * this policy is already at 4, this request will fail.
         */
        DeleteResourcePolicyRequest deletePolicyRequest =
                DeleteResourcePolicyRequest.builder()
                        .resourceArn("arn:aws:lex:Region:123456789012:bot-
alias/MYBOTALIAS/TSTALIASID")
                        .expectedRevisionId(3);
                        .build();

        lexmodelsv2Client.deleteResourcePolicy(deletePolicyRequest);
```

IAM ロールがボットを更新し、ボットのエイリアスを一覧表示できるようにする。

次の例では、特定の IAM ロールに対して Amazon Lex V2 モデル構築 API オペレーションを呼び出して、既存のボットを変更する権限を付与します。ユーザーはボットのエイリアスを一覧表示してボットを更新できますが、ボットまたはボットのエイリアスは削除できません。

JSON

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

```
        {
            "Sid": "botBuilders",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:role/BotBuilder"
            },
            "Action": [
                "lex:ListBotAliases",
                "lex:UpdateBot"
            ],
            "Resource": [
                "arn:aws:lex:us-east-1:123456789012:bot/MYBOT"
            ]
        }
    ]
}
```

ユーザーにボットとの会話を許可する

次の例では、ボットの単一のエイリアスで Amazon Lex V2 ランタイム API オペレーションを呼び出す権限を特定のユーザーに付与します。

ユーザーには、ボットエイリアスを更新または削除する許可が特に拒否されています。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "botRunners",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:user/botRunner"
            },
            "Action": [
                "lex:RecognizeText",
                "lex:RecognizeUtterance",
                "lex:StartConversation",
                "lex:DeleteSession",
                "lex:GetSession",
                "lex:PutSession"
```

```
            ],
            "Resource": [
                "arn:aws:lex:us-east-1:123456789012:bot-alias/MYBOT/MYBOTALIAS"
            ]
        },
        {
            "Sid": "botRunners",
            "Effect": "Deny",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:user/botRunner"
            },
            "Action": [
                "lex:UpdateBotAlias",
                "lex:DeleteBotAlias"
            ],
            "Resource": [
                "arn:aws:lex:us-east-1:123456789012:bot-alias/MYBOT/MYBOTALIAS"
            ]
        }
    ]
}
```

AWS サービスに特定の Amazon Lex V2 ボットの使用を許可する

次の例では、Amazon Lex V2 ランタイム API オペレーションを呼び出すアクセス許可を AWS Lambda と Connect Customer に付与します。 Amazon Lex

条件ブロックはサービスプリンシパルに必要であり、グローバルコンテキストキー `AWS:SourceAccount` および `AWS:SourceArn` を使用する必要があります。

`AWS:SourceAccount` は、Amazon Lex V2 ボットを呼び出しているアカウント ID です。

`AWS:SourceArn` は、Amazon Lex V2 ボットエイリアスの呼び出し元の Connect Customer Service インスタンスまたは Lambda 関数のリソース ARN です。

Amazon Lex V2 の AWS 管理ポリシー V2

AWS 管理ポリシーは、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 管理ポリシーは、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ように設計されているため、ユーザー、グループ、ロールにアクセス許可の割り当てを開始できます。

AWS 管理ポリシーは、すべての AWS のお客様が使用できるため、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注意してください。ユースケースに固有のカスタマー管理ポリシーを定義して、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されているアクセス許可は変更できません。AWS が AWS 管理ポリシーで定義されたアクセス許可を更新すると、その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 ID (ユーザー、グループ、ロール) に影響します。AWS は、新しい AWS のサービス が起動されたとき、または既存のサービスで新しい API オペレーションが利用可能になったときに、AWS 管理ポリシーを更新する可能性が最も高いです。

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` ポリシーを IAM アイデンティティ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 モデル構築サービス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。

アクセス許可の詳細

このポリシーには、以下のアクセス許可が含まれています。

- `lex` — モデル構築サービス内の Amazon Lex V2 および Amazon Lex リソースへの読み取り専用アクセス。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AmazonLexReadOnlyStatement1",
            "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:DescribeBotRecommendation",
                "lex:DescribeBotReplica",
                "lex:DescribeBotVersion",
                "lex:DescribeExport",
                "lex:DescribeImport",
                "lex:DescribeIntent",
                "lex:DescribeResourcePolicy",
                "lex:DescribeSlot",
                "lex:DescribeSlotType",
                "lex:ListBots",
                "lex:ListBotLocales",
                "lex:ListBotAliases",
                "lex:ListBotAliasReplicas",
                "lex:ListBotChannels",
                "lex:ListBotRecommendations",
                "lex:ListBotReplicas",
                "lex:ListBotVersions",
                "lex:ListBotVersionReplicas",
                "lex:ListBuiltInIntents",
                "lex:ListBuiltInSlotTypes",
                "lex:ListExports",
                "lex:ListImports",
                "lex:ListIntents",
                "lex:ListRecommendedIntents",
                "lex:ListSlots",
                "lex:ListSlotTypes",
                "lex:ListTagsForResource",
```

```
                "lex:SearchAssociatedTranscripts",
                "lex:ListCustomVocabularyItems"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS マネージドポリシー: AmazonLexRunBotsOnly

AmazonLexRunBotsOnly ポリシーを IAM アイデンティティにアタッチできます。

このポリシーは、Amazon Lex V2 および Amazon Lex 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。

アクセス許可の詳細

このポリシーには、以下のアクセス許可が含まれています。

- lex — Amazon Lex V2 および Amazon Lex 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。

JSON

```
{
    "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 マネージドポリシー: AmazonLexFullAccess

AmazonLexFullAccess ポリシーを IAM アイデンティティにアタッチできます。

このポリシーは、Amazon Lex V2 および Amazon Lex リソースを作成、読み取り、更新、および削除し、Amazon Lex V2 および Amazon Lex 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。

アクセス許可の詳細

このポリシーには、以下のアクセス許可が含まれています。

- lex — Amazon Lex V2 および Amazon Lex モデル構築およびランタイムサービス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。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AmazonLexFullAccessStatement1",
            "Effect": "Allow",
```

```
            "Action": [
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:DescribeAlarms",
                "cloudwatch:DescribeAlarmsForMetric",
                "kms:DescribeKey",
                "kms:ListAliases",
                "lambda:GetPolicy",
                "lambda:ListFunctions",
                "lambda:ListAliases",
                "lambda:ListVersionsByFunction",
                "lex:*",
                "polly:DescribeVoices",
                "polly:SynthesizeSpeech",
                "kendra:ListIndices",
                "iam:ListRoles",
                "s3:ListAllMyBuckets",
                "logs:DescribeLogGroups",
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "AmazonLexFullAccessStatement2",
            "Effect": "Allow",
            "Action": [
                "bedrock:ListFoundationModels"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "bedrock:InvokeModel"
            ],
            "Resource": "arn:aws:bedrock:*::foundation-model/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "lambda:AddPermission",
                "lambda:RemovePermission"
            ],
```

```
            "Resource": "arn:aws:lambda:*:*:function:AmazonLex*",
            "Condition": {
                "StringEquals": {
                    "lambda:Principal": "lex.amazonaws.com"
                }
            }
        },
        {
            "Sid": "AmazonLexFullAccessStatement3",
            "Effect": "Allow",
            "Action": [
                "iam:GetRole",
                "iam:GetRolePolicy"
            ],
            "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*",
                "arn:aws:iam::*:role/aws-service-role/
replication.lexv2.amazonaws.com/AWSServiceRoleForLexV2Replication*"
            ]
        },
        {
            "Sid": "AmazonLexFullAccessStatement4",
            "Effect": "Allow",
            "Action": [
                "iam:CreateServiceLinkedRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/aws-service-role/lex.amazonaws.com/
AWSServiceRoleForLexBots"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:AWSServiceName": "lex.amazonaws.com"
                }
            }
        },
        {
```

```
            "Sid": "AmazonLexFullAccessStatement5",
            "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"
                }
            }
        },
        {
            "Sid": "AmazonLexFullAccessStatement6",
            "Effect": "Allow",
            "Action": [
                "iam:CreateServiceLinkedRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/aws-service-role/lexv2.amazonaws.com/
AWSServiceRoleForLexV2Bots*"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:AWSServiceName": "lexv2.amazonaws.com"
                }
            }
        },
        {
            "Sid": "AmazonLexFullAccessStatement7",
            "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"
```

```
                }
            }
        },
        {
            "Sid": "AmazonLexFullAccessStatement8",
            "Effect": "Allow",
            "Action": [
                "iam:CreateServiceLinkedRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/aws-service-role/
replication.lexv2.amazonaws.com/AWSServiceRoleForLexV2Replication*"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:AWSServiceName": "replication.lexv2.amazonaws.com"
                }
            }
        },
        {
            "Sid": "AmazonLexFullAccessStatement9",
            "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*",
                "arn:aws:iam::*:role/aws-service-role/
replication.lexv2.amazonaws.com/AWSServiceRoleForLexV2Replication*"
            ]
        },
        {
            "Sid": "AmazonLexFullAccessStatement10",
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
```

```
            ],
            "Resource": [
                "arn:aws:iam::*:role/aws-service-role/lex.amazonaws.com/
AWSServiceRoleForLexBots"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "lex.amazonaws.com"
                    ]
                }
            }
        },
        {
            "Sid": "AmazonLexFullAccessStatement11",
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/aws-service-role/lexv2.amazonaws.com/
AWSServiceRoleForLexV2Bots*"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "lexv2.amazonaws.com"
                    ]
                }
            }
        },
        {
            "Sid": "AmazonLexFullAccessStatement12",
            "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"
                    ]
                }
            }
        },
        {
            "Sid": "AmazonLexFullAccessStatement13",
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/aws-service-role/
replication.lexv2.amazonaws.com/AWSServiceRoleForLexV2Replication*"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "lexv2.amazonaws.com"
                    ]
                }
            }
        }
    ]
}
```

AWS マネージドポリシー: AmazonLexReplicationPolicy

IAM エンティティに AmazonLexReplicationPolicy をアタッチすることはできません。このポリシーは、Amazon Lex V2 がユーザーに代わってアクションを実行することを許可するサービスリンクロールにアタッチされます。詳細については、「Amazon Lex V2 のサービスリンクロール」を参照してください。

このポリシーは、Amazon Lex V2 がユーザーに代わってリージョン間で AWS リソースをレプリケートできるようにする管理アクセス許可を付与します。このポリシーをアタッチすると、ロールがボット、ロケール、バージョン、エイリアス、インテント、スロットタイプ、スロット、カスタム語彙などのリソースを簡単にレプリケートできるようになります。

アクセス許可の詳細

このポリシーには、以下のアクセス許可が含まれています。

- lex – プリンシパルが他のリージョンにリソースをレプリケートできるようにします。
- iam — プリンシパルが IAM のロールを渡すことを許可します。これは、Amazon Lex V2 に他のリージョンにリソースをレプリケートする権限を与えるために必要です。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
 {
  "Sid": "ReplicationPolicyStatement1",
  "Effect": "Allow",
  "Action": [
   "lex:BuildBotLocale",
   "lex:ListBotLocales",
   "lex:CreateBotAlias",
   "lex:UpdateBotAlias",
   "lex:DeleteBotAlias",
   "lex:DescribeBotAlias",
   "lex:CreateBotVersion",
   "lex:DeleteBotVersion",
   "lex:DescribeBotVersion",
   "lex:CreateExport",
   "lex:DescribeBot",
   "lex:UpdateExport",
   "lex:DescribeExport",
   "lex:DescribeBotLocale",
   "lex:DescribeIntent",
   "lex:ListIntents",
   "lex:DescribeSlotType",
   "lex:ListSlotTypes",
   "lex:DescribeSlot",
   "lex:ListSlots",
   "lex:DescribeCustomVocabulary",
   "lex:StartImport",
   "lex:DescribeImport",
   "lex:CreateBot",
```

```
    "lex:UpdateBot",
    "lex:DeleteBot",
    "lex:CreateBotLocale",
    "lex:UpdateBotLocale",
    "lex:DeleteBotLocale",
    "lex:CreateIntent",
    "lex:UpdateIntent",
    "lex:DeleteIntent",
    "lex:CreateSlotType",
    "lex:UpdateSlotType",
    "lex:DeleteSlotType",
    "lex:CreateSlot",
    "lex:UpdateSlot",
    "lex:DeleteSlot",
    "lex:CreateCustomVocabulary",
    "lex:UpdateCustomVocabulary",
    "lex:DeleteCustomVocabulary",
    "lex:DeleteBotChannel",
    "lex:ListTagsForResource",
    "lex:TagResource",
    "lex:UntagResource",
    "lex:CreateResourcePolicy",
    "lex:DeleteResourcePolicy",
    "lex:DescribeResourcePolicy",
    "lex:UpdateResourcePolicy"
   ],
   "Resource": [
    "arn:aws:lex:*:*:bot/*",
    "arn:aws:lex:*:*:bot-alias/*"
   ]
  },
  {
   "Sid": "ReplicationPolicyStatement2",
   "Effect": "Allow",
   "Action": [
    "lex:CreateUploadUrl",
    "lex:ListBots"
   ],
   "Resource": "*"
  },
  {
   "Sid": "ReplicationPolicyStatement3",
   "Effect": "Allow",
   "Action": [
```

```
    "iam:PassRole"
   ],
   "Resource": "*",
   "Condition": {
    "StringEquals": {
     "iam:PassedToService": "lexv2.amazonaws.com"
    }
   }
  }
     ]
}
```

AWS マネージドポリシー: AmazonLexV2BedrockAgentPolicy

Amazon Bedrock エージェント用の Amazon Lex V2 ポリシー

[応答]

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Sid": "LexV2TrustPolicy",
      "Principal": {
        "Service": "lexv2.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "{accountId}"
        }
      }
    }
  ]
}
```

AWS マネージドポリシー: AmazonLexV2BedrockKnowledgeBasePolicy

Amazon Bedrock ナレッジベースの Amazon Lex V2 ポリシー

[応答]

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Sid": "LexV2TrustPolicy",
      "Principal": {
        "Service": "lexv2.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
     "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "{accountId}"
        }
      }
    }
  ]
}
```

AWS マネージドポリシー: AmazonLexV2BedrockAgentPolicyInternal

Amazon Bedrock エージェント用の Amazon Lex V2 内部ポリシー

[応答]

JSON

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

```
      "Effect": "Allow",
      "Sid": "LexV2InternalTrustPolicy",
      "Principal": {
        "Service": "lexv2.aws.internal"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "{accountId}"
        }
      }
    }
  ]
}
```

AWS マネージドポリシー: AmazonLexV2BedrockKnowledgeBasePolicyInternal

Amazon Bedrock ナレッジベースの Amazon Lex V2 内部ポリシー

[応答]

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "LexV2InternalTrustPolicy",
      "Effect": "Allow",
      "Principal": {
        "Service": "lexv2.aws.internal"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "{accountId}"
        }
      }
    }
  ]
}
```

Amazon Lex V2 の AWS 管理ポリシーの更新

このサービスがこれらの変更の追跡を開始してからの Amazon Lex V2 の AWS 管理ポリシーの更新に関する詳細を表示します。 V2 このページへの変更に関する自動アラートについては、Amazon Lex V2 Amazon Lex V2 のドキュメント履歴 ページのRSSフィードを購読してください。

変更	説明	日付
AmazonLexReplicationPolicy – 更新されたポリシー	Amazon Lex V2 は、タグと ResourceBasedPolicy のレプリケーションを許可するようにポリシーを更新しました。	2025 年 6 月 24 日
AmazonLexV2BedrockKnowledgeBasePolicyInternal – 新しいポリシー	Amazon Lex V2 は、Amazon Bedrock ナレッジベースリソースのレプリケーションを許可する新しいポリシーを追加しました。	2024 年 8 月 30 日
AmazonLexV2BedrockAgentPolicyInternal – 新しいポリシー	Amazon Lex V2 は、Amazon Bedrock エージェントリソースのレプリケーションを許可する新しいポリシーを追加しました。	2024 年 8 月 30 日
AmazonLexV2BedrockKnowledgeBasePolicy – 新しいポリシー	Amazon Lex V2 は、Amazon Bedrock ナレッジベースリソースのレプリケーションを許可する新しいポリシーを追加しました。	2024 年 8 月 30 日
AmazonLexV2BedrockAgentPolicy – 新しいポリシー	Amazon Lex V2 は、Amazon Bedrock エージェントリソー	2024 年 8 月 30 日

変更	説明	日付
	スのレプリケーションを許可する新しいポリシーを追加しました。	
AmazonLexReadOnly — 既存のポリシーに対する更新	Amazon Lex V2 は、ボットリソースの読み取り専用アクセスレプリカを許可する新しい権限を追加しました。	2024 年 5 月 10 日
AmazonLexFullAccess — 既存のポリシーに対する更新	Amazon Lex V2 は、ボットリソースを他のリージョンにレプリケーションする新しい権限を追加しました。	2024 年 4 月 16 日
AmazonLexFullAccess — 既存のポリシーに対する更新	Amazon Lex V2 は、ボットリソースを他のリージョンにレプリケーションする新しい権限を追加しました。	2024 年 1 月 31 日
AmazonLexReplicationPolicy – 新しいポリシー	Amazon Lex V2 は、ボットリソースを他のリージョンにレプリケーションする新しいポリシーを追加しました。	2024 年 1 月 31 日
AmazonLexReadOnly — 既存のポリシーに対する更新	Amazon Lex V2 では、カスタム語彙項目をリストアップするため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。	2022 年 11 月 29 日
AmazonLexFullAccess — 既存のポリシーに対する更新	Amazon Lex V2 では、Amazon Lex V2 モデル構築サービス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

変更	説明	日付
AmazonLexReadOnly — 既存のポリシーに対する更新	Amazon Lex V2 では、Amazon Lex V2 自動 Chatbot デザイナー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 12 月 1 日
AmazonLexFullAccess — 既存のポリシーに対する更新	Amazon Lex V2 では、Amazon Lex V2 モデル構築サービス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 V2 では、Amazon Lex V2 モデル構築サービス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 V2 では、Amazon Lex V2 ランタイムサービス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 V2 が変更の追跡を開始しました。	Amazon Lex V2 は、AWS 管理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 V2 のサービスリンクロール

Amazon Lex V2 は、AWS Identity and Access Management (IAM) サービスにリンクされたロールを使用します。サービスリンクロールは、Amazon Lex V2 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 V2 によって事前定義されており、サービスがユーザーに代わって他の AWS サービスを呼び出すために必要なすべてのアクセス許可が含まれています。

必要なアクセス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 V2 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アクセス許可は Amazon Lex V2 が定義し、別段の定義がない限り、Amazon Lex V2 のみがそのロールを引き受けることができます。定義される許可は信頼ポリシーと許可ポリシーに含まれており、その許可ポリシーを他の IAM エンティティにアタッチすることはできません。

サービスリンクロールをサポートする他のサービスについては、「IAM と連携する AWS サービス」で「サービスリンクロール」列が「はい」になっているサービスを探してください。サービスにリンクされたロールに関するサービスのドキュメントを表示するには、「はい」のリンクをクリックします。

サービスリンクロールの作成、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) に許可するにはアクセス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「サービスリンクロールの許可」を参照してください。

サービスリンクロールを削除するには、まず関連リソースを削除します。これにより、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ように、Amazon Lex V2 リソースを保護します。

トピック

- Amazon Lex V2 のサービスリンクロールの作成
- Amazon Lex V2 のサービスリンクロールの編集
- Amazon Lex V2 のサービスリンクロールの削除
- Amazon Lex V2 のサービスリンクロール許可
- Amazon Lex V2 サービスリンクロールがサポートされるリージョン

Amazon Lex V2 のサービスリンクロールの作成

AWS マネジメントコンソール、AWS CLI、または AWS API で関連するアクションを実行すると (Amazon Lex V2 のサービスリンクロール許可詳細については「」を参照)、Amazon Lex V2 によっ

てサービスにリンクされたロールが自動的に作成されるため、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。

このサービスリンク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。

Amazon Lex V2 のサービスリンクロールの編集

Amazon Lex V2 では、 のサービスリンクロールを編集することはできません。サービスリンク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 を使用してロールの説明を編集することはできます。詳細については、「IAM ユーザーガイド」の「サービスリンクロールの編集」を参照してください。

Amazon Lex V2 のサービスリンクロールの削除

サービスリンク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、サービスリンク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 V2 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。

Amazon Lex V2 で特定のサービスリンクロールのリソースを削除する手順については、Amazon Lex V2 のサービスリンクロール許可 のロールに固有のセクションを参照してください。

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

サービスにリンクされたロールに関連するリソースを削除したら、IAM コンソール、AWS CLI、または AWS API を使用して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「サービスリンクロールの削除」を参照してください。

Amazon Lex V2 のサービスリンクロール許可

Amazon Lex V2 は、以下のプレフィックスを持つサービスリンクロールを使用します。

トピック

- AWSServiceRoleForLexV2Bots _
- AWSServiceRoleForLexV2Channels _
- AWSServiceRoleForLexV2Replication

AWSServiceRoleForLexV2Bots _

AWSServiceRoleForLexV2Bots_ — ロールは、ボットを他の必要なサービスに接続する権限を与えます。このロールには、lexv2.amazonaws.com サービスがロールを引き受けることを許可する信頼ポリシーが含まれており、以下のアクションを実行するための権限も含まれています。

- Amazon Polly を使用して、アクションがサポートするすべての Amazon Lex V2 リソースで音声を合成します。
- Amazon Comprehend センチメント分析を使用するようにボットが設定されている場合は、アクションがサポートするすべての Amazon Lex V2 リソースでセンチメントを検出します。
- S3 バケットにオーディオログを保存するようにボットが設定されている場合は、指定されたバケットにオブジェクトを置きます。
- ボットがオーディオログとテキストログを保存するように設定されている場合は、ログストリーミングを作成し、指定したロググループにログを配置します。
- ボットが AWS KMS キーを使用してデータを暗号化するように設定されている場合は、特定のデータキーを生成します。
- ボットが `KendraSearchIntent` インテントを使用するように設定されている場合は、指定された Amazon Kendra インデックスへのクエリアクセスを行います。

ロールを作成するには

Amazon Lex V2 は、ボットを作成するたびに、アカウントにランダムなサフィックスを持つ新しい AWSServiceRoleForLexV2Bots _ ロールを作成します。ボットに機能を追加すると、Amazon Lex V2 によってロールが変更されます。例えば、Amazon Comprehend のセンチメント分析をボットに追加すると、Amazon Lex V2 は `lex:DetectSentiment` アクションに対する権限をサービスロールに追加します。

ロールを削除するには

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/lex/ で Amazon Lex コンソールを開きます。

2. 左側のナビゲーションペインから [ボット] を選択し、サービスリンクロールを削除するボットを選択します。
3. ボットの任意のバージョンを選択します。
4. IAM アクセス許可のランタイムロールは、バージョンの詳細にあります。
5. [ボット] ページに戻り、削除するボットの横にあるラジオボタンを選択します。
6. [アクション] を選択して、[削除] を選択します。
7. 「サービスリンクロールの削除 」の手順に従って、IAMロールを削除します。

AWSServiceRoleForLexV2Channels _

AWSServiceRoleForLexV2Channels_ — ロールは、アカウント内のボットを一覧表示し、ボットの会話 API を呼び出す権限を与えます。このロールには、channels.lexv2.amazonaws.com サービスがロールを引き受けることを許可する信頼ポリシーが含まれています。ボットがチャネルを使用してメッセージングサービスと通信するように設定されている場合、AWSServiceRoleForLexV2Channels_ ロールのアクセス許可ポリシーにより、Amazon Lex V2 は以下のアクションを完了することを許可します。

- アカウント内のすべてのボットに対する権限を一覧表示します。
- テキストを認識し、セッションを取得し、指定したボットエイリアスにセッション許可を設定します。

ロールを作成するには

チャネル統合を作成してメッセージングプラットフォームにボットをデプロイすると、Amazon Lex V2 は各チャネルごとにランダムなサフィックス付きの新しいサービスリンクロールをアカウント内に作成します。

ロールを削除するには

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/lex/ で Amazon Lex コンソールを開きます。
2. 左のナビゲーションペインで [ボット] を選択します。
3. ボットを選択します。
4. 左側のナビゲーションペインで、[デプロイ] から [チャネル統合] を選択します。
5. サービスリンクロールを削除するチャネルを選択します。

6. [IAM アクセス許可のランタイムロール] が [全般設定] にある
7. [削除] を選択し、[削除] をもう一度選択して、チャネルを削除します。
8. 「サービスリンクロールの削除 」の手順に従って、IAMロールを削除します。

AWSServiceRoleForLexV2Replication

AWSServiceRoleForLexV2Replication ロールは、2 つ目のリージョンにボットをレプリケートする権限を与えます。このロールには、replication.lexv2.amazonaws.com サービスがロールを引き受けることを許可する信頼ポリシーと、以下のアクションのアクセス許可を許可する AmazonLexReplicationPolicy AWS 管理ポリシーが含まれています。

- ボットの IAM ロールをレプリカボットに渡して、レプリカボットに適切な権限を再作成します。
- 他のリージョンでボットとボットリソース (バージョン、エイリアス、インテント、スロット、カスタム語彙など) を作成および管理します。

ロールを作成するには

ボットのグローバルレジリエンシーを有効にすると、Amazon Lex V2 はアカウントに AWSServiceRoleForLexV2Replication サービスリンクロールを作成します。サービスリンクロールを作成するための権限を Amazon Lex V2 サービスに付与する正しい権限があることを確認します。

AWSServiceRoleForLexV2Replication で使用される Amazon Lex V2 リソースを削除して、ロールを削除できるようにするには

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/lex/ で Amazon Lex コンソールを開きます。
2. Global Resiliency が有効になっているボットを選択します。
3. [デプロイ] で [グローバルレジリエンシー] を選択します。
4. [グローバルレジリエンシーの無効化] を選択します。
5. グローバルレジリエンシーが有効になっているすべてのボットに対してこのプロセスを繰り返します。
6. 「サービスリンクロールの削除 」の手順に従って、IAMロールを削除します。

Amazon Lex V2 サービスリンクロールがサポートされるリージョン

Amazon Lex V2 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、「AWS のリージョンとエンドポイント」を参照してください。

Amazon Lex V2 のサービスロールの使用 V2

Amazon Lex V2 は IAM サービスロールを使用して、実行時にユーザーに代わって他の AWS のサービスにアクセスします。サービスロールは、ボットの構築と実行に必要なアクセス許可を提供します。サービスロールは、 アカウントの標準カスタマー管理 IAM ロールです。AWS Organizations で設定したサービスコントロールポリシー (SCPs) を尊重し、Amazon Lex V2 が使用するアクセス許可を完全に制御できます。

AmazonLexServiceRole-

AmazonLexServiceRole- ロールは、ボットを他の必要なサービスに接続するためのアクセス許可を付与します。このロールには、lexv2.amazonaws.com サービスがロールを引き受けることを許可する信頼ポリシーが含まれており、以下のアクションを実行するための権限も含まれています。

- ボットがtext-to-speechを使用するように設定されている場合は、Amazon Polly を使用して、アクションがサポートするすべての Amazon Lex V2 リソースで音声を合成します。
- Amazon Comprehend センチメント分析を使用するようにボットが設定されている場合は、アクションがサポートするすべての Amazon Lex V2 リソースでセンチメントを検出します。
- S3 バケットにオーディオログを保存するようにボットが設定されている場合は、指定されたバケットにオブジェクトを置きます。
- ボットがオーディオログとテキストログを保存するように設定されている場合は、ログストリーミングを作成し、指定したロググループにログを配置します。
- ボットが AWS KMS キーを使用してデータを暗号化するように設定されている場合は、特定のデータキーを生成します。
- ボットが Amazon Bedrock 生成 AI 機能を使用するように設定されている場合は、指定された Amazon Bedrock モデル、推論プロファイル、ナレッジベースを呼び出します。

ロールの作成

Amazon Lex V2 は、ボットを作成し、新しいロールの作成を選択するときに、アカウントにランダムなサフィックスを持つ新しい AmazonLexServiceRole- ロールを作成します。ボットに機能を追加

すると、Amazon Lex V2 によってロールが変更されます。例えば、Amazon Comprehend 感情分析をボットに追加すると、Amazon Lex V2 は `comprehend:DetectSentiment`アクションのアクセス許可をサービスロールに追加します。

Amazon Lex V2 サービスプリンシパル () を信頼している限り、自分で作成して設定した既存の IAM ロールを選択することもできます`lexv2.amazonaws.com`。 V2

ロールの編集または削除

サービスロールは標準のカスタマー管理 IAM ロールであるため、他の IAM ロールと同じ方法で管理します。IAM コンソール、AWS CLI、または IAM API を使用します。ロールの信頼ポリシーとアタッチされたアクセス許可ポリシーは、IAM コンソールでいつでも確認できます。

> **Important**
>
> サービスロールを削除する前に、まだボットが使用していないことを確認してください。ボットが依存するロールを削除すると、ボットの実行に必要なアクセス許可が失われます。

Amazon Lex V2 サービスロールでサポートされているリージョン

Amazon Lex V2 は、サービスが利用可能なすべてのリージョンでサービスロールの使用をサポートしています。詳細については、「AWS のリージョンとエンドポイント」を参照してください。

Amazon Lex V2 のサービスロールへの移行

Amazon Lex V2 は、AWS マネージドサービスリンクロール (SLRs) からボットランタイムアクセス許可のカスタマーマネージドサービスロールに移行しています。このトピックでは、Amazon Lex V2 がこの変更を行う理由、既存のボットの移行方法、移行のタイムラインについて説明します。

Amazon Lex V2 がサービスロールに移行する理由

サービスにリンクされたロールは、Amazon Lex V2 に直接リンクされ、AWS によって管理される IAM ロールです。サービスにリンクされたロールは、AWS Organizations で設定したサービスコントロールポリシー (SCPs) を尊重しません。

サービスロールは、自分のアカウントで所有および管理している IAM ロールです。サービスロールは標準のカスタマー管理ロールであるため、組織で設定したサービスコントロールポリシー (SCPs) が優先されます。これにより、Amazon Lex V2 がユーザーに代わって使用するアクセス許可を完全

に制御できます。サービスロールに移行すると、Amazon Lex V2 が IAM のベストプラクティスと整合し、セキュリティとコンプライアンス体制が向上します。

既存のボットをサービスロールに移行する

サービスにリンクされたロールを使用する既存のボットを、Amazon Lex V2 コンソールから直接カスタマーマネージドサービスロールに移行できます。

> **Note**
>
> Amazon Lex V2 は、2026 年 9 月 1 日より前にサービスロールへの移行ボタンを使用して単一ボットの移行を行う予定です。ボタンが使用可能になったら、次の手順を実行してボットを移行します。

1. https://console.aws.amazon.com/lexv2/home で Amazon Lex V2 コンソールを開きます。
2. 左側のナビゲーションペインで、Bots を選択し、移行するボットを選択します。
3. ボットのドラフトバージョンページで、ボットがサービスにリンクされたロールを使用している場合、ボットがサービスにリンクされたロールを使用していることを示すバナーが表示されます。サービスロールへの移行を選択します。
4. Amazon Lex V2 は、新しいカスタマーマネージドサービスロールを作成し、ボットに必要なアクセス許可をアタッチし、新しいロールを使用するようにボットを更新します。
5. 移行が完了すると、サービスにリンクされたロールバナーとサービスロールへの移行ボタンがボットに表示されなくなり、ボットがサービスロールを使用するようになりました。

> **Note**
>
> サービスロールへの移行ボタンは、サービスにリンクされたロールを使用しているボットに対してのみ表示されます。移行では、元のサービスにリンクされたロールは変更または削除されません。移行が完了しない場合、ボットは既存のサービスにリンクされたロールを引き続き使用し、完全に機能し続けます。

移行タイムライン

サービスにリンクされたロールからサービスロールへの移行は、次のリストで説明されているタイムラインに従います。

- 新しいボット – コンソールで作成された新しいボットは、デフォルトでサービスロールになりました。サービスロールを使用して新しいボットを作成することをお勧めします。
- 1 つのボットの移行 – ワンクリックでサービスロールへの移行ボタンは、サービスにリンクされたロールを使用するボットにのみ表示されます。ワンクリックで、ボットをサービスロールに移行できます。Amazon Lex V2 は、これを 2026 年 9 月 1 日までに利用可能にする予定です。
- 複数のボットの移行 – ウィザードにボットが一覧表示されるため、複数のサービスにリンクされたロールボットを選択し、一度にサービスロールに移行できます。Amazon Lex V2 は、これを 2026 年 9 月 15 日までに利用可能にする予定です。
- サービスにリンクされたロールのサポート終了 – サービスにリンクされたロールのサポートは 2026 年末までに終了します。

Amazon Lex V2 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 V2 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 V2 でアクションを実行する権限がない
- iam:PassRole を実行する権限がない
- 管理者として Amazon Lex V2 へのアクセスを他のユーザーに許可したい
- ユーザーにプログラム的なアクセス権を付与する
- AWS アカウント以外のユーザーに Amazon Lex V2 リソースへのアクセスを許可したい

Amazon Lex V2 でアクションを実行する権限がない

AWS マネジメントコンソール から、アクションを実行する権限がないと通知された場合、管理者に問い合わせ、サポートを依頼する必要があります。管理者とは、サインイン認証情報を提供した担当者です。

以下のエラー例は、`mateojackson` IAM ユーザーがコンソールを使用して架空の *`my-example-widget`* リソースに関する詳細情報を表示しようとしているが、架空の `lex:`*`GetWidget`* 権限がないという場合に発生します。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 lex:GetWidget on resource: my-example-widget
```

この場合、Mateo は、`lex:GetWidget` アクションを使用して `my-example-widget` リソースにアクセスできるように、ポリシーの更新を管理者に依頼します。

iam:PassRole を実行する権限がない

`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 V2 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アクセス許可が必要です。

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 V2 でアクションを実行しようと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アクセス許可が必要です。Mary には、ロールをサービスに渡すアクセス許可がありません。

```
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。サインイン資格情報を提供した担当者が管理者です。

管理者として Amazon Lex V2 へのアクセスを他のユーザーに許可したい

他のユーザーが Amazon Lex V2 にアクセスすることを許可するには、アクセスが必要なユーザーまたはアプリケーションにアクセス許可を付与する必要があります。 AWS IAM アイデンティティセンター を使用してユーザーとアプリケーションを管理する場合は、アクセスレベルを定義するアクセス許可セットをユーザーまたはグループに割り当てます。アクセス許可セットは、ユーザーまたはアプリケーションに関連付けられている IAM ロールに自動的に IAM ポリシーを作成して割り当てます。詳細については、「AWS IAM アイデンティティセンター 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。

IAM アイデンティティセンターを使用していない場合は、アクセスを必要としているユーザーまたはアプリケーションの IAM エンティティ (ユーザーまたはロール) を作成する必要があります。次に、Amazon Lex V2 の適切なアクセス許可を付与するエンティティにポリシーをアタッチする必要があります。アクセス許可が付与されたら、ユーザーまたはアプリケーション開発者に認証情報

を提供します。これらの認証情報を使用して AWS にアクセスします。IAM ユーザー、グループ、ポリシー、アクセス許可の作成の詳細については、「IAM ユーザーガイド」の「IAM アイデンティティ」と「IAM のポリシーとアクセス許可」を参照してください。

ユーザーにプログラム的なアクセス権を付与する

アクセスキー ID とシークレットアクセスキーを取得する方法については、AWS 全般のリファレンスの「AWS 認証情報の理解と取得」を参照してください。

AWS アカウント以外のユーザーに Amazon Lex V2 リソース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 V2 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 V2 で IAM が機能する仕組み」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 外部で認証されたユーザー (ID フェデレーション) へのアクセスの許可 を参照してください。
- クロスアカウントアクセスにおけるロールとリソースベースのポリシーの使用方法の違いについては、「IAM ユーザーガイド」の「IAM でのクロスアカウントのリソースへのアクセス」を参照してください。

Amazon Lex V2 でのログ記録とモニタリング

モニタリングは、Amazon Lex V2 および AWS のその他のソリューションの信頼性、可用性、およびパフォーマンスを維持するための重要な部分です。AWS には、Amazon Lex V2 をモニタリングし

て異常を検出した場合に報告し、必要に応じて自動的に対処するために、次のモニタリングツールが用意されています。

- 「Amazon CloudWatch」は、AWS リソースと、AWS で実行されるアプリケーションをリアルタイムで監視します。メトリクスの収集と追跡、カスタマイズしたダッシュボードの作成、および指定したメトリクスが指定したしきい値に達したときに通知またはアクションを実行するアラームの設定を行うことができます。例えば、CloudWatch で Amazon EC2 インスタンスの CPU 使用率などのメトリクスを追跡し、必要に応じて新しいインスタンスを自動的に起動できます。詳細については、「Amazon CloudWatch ユーザーガイド」を参照してください。
- AWS CloudTrail は、AWS アカウントによって、または AWS アカウントに代わって行われた API コールおよび関連イベントをキャプチャし、指定した Amazon S3 バケットにログファイルを配信します。AWS を呼び出したユーザーとアカウント、呼び出し元 IP アドレス、および呼び出し日時を特定できます。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。

Amazon Lex V2 のコンプライアンス検証

サードパーティーの監査者は、複数の AWS コンプライアンスプログラムの一環として Amazon Lex V2 のセキュリティとコンプライアンスを評価します。Amazon Lex V2 は HIPAA 対象サービスです。これは、PCI、SOC、および ISO に準拠しています。

新しい AWS サインアップエクスペリエンスは、規制されたワークロード向けには設計されていません。新しい AWS サインアップエクスペリエンスを使用しているが、規制対象のワークロードに AWS を使用する場合は、AWS (アドバンスド) にサインアップするか、AWS 環境の高度な機能をアクティブ化できます。

AWS のサービス が特定のコンプライアンスプログラムの範囲内にあるかどうかを確認するには、「コンプライアンスAWS のサービス プログラムによるスコープ」の「コンプライアンス」を参照して、関心のあるコンプライアンスプログラムを選択します。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。

を使用して、サードパーティーの監査レポートをダウンロードできます AWS Artifact。詳細については、「Downloading Reports in AWS Artifact」を参照してください。

を使用する際のお客様のコンプライアンス責任 AWS のサービス は、お客様のデータの機密性、貴社のコンプライアンス目的、適用可能な法律および規制によって決まります。を使用する際のコンプライアンス責任の詳細については AWS のサービス、「AWS セキュリティドキュメント」を参照してください。

Amazon Lex V2 の耐障害性

AWS の 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として構築されます。AWS リージョンには、低レイテンシー、高いスループット、そして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し隔離されたアベイラビリティーゾーンがあります。アベイラビリティーゾーンでは、ゾーン間で中断することなく自動的にフェー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、フォールトトレランス、および拡張性が優れています。

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 V2 には、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をサポートするのに役立ついくつかの機能が用意されています。

> **Note**
>
> Amazon Lex V2 のグローバルレジリエンシーの詳細については、「グローバルレジリエンシー」を参照してください。

Amazon Lex V2 で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 V2 は、ホワイトペーパー「Amazon Web Services: 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手順で保護されています。

AWS が公開した API コールを使用して、ネットワーク経由で Amazon Lex V2 にアクセスします。クライアントで Transport Layer Security (TLS) 1.0 以降がサポートされ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。

Amazon Lex V2 とインターフェイス VPC エンドポイント (AWS PrivateLink)

VPC と Amazon Lex V2 とのプライベート接続を確立するには、インターフェイス VPC エンドポイントを作成します。インターフェイスエンドポイントは、インターネットゲートウェイAWS PrivateLink、NAT デバイス、VPN 接続、または AWS Direct Connect 接続なしで Amazon Lex V2 APIs にプライベートにアクセスできるテクノロジーである を利用しています。VPC のインスタンスは、パブリック IP アドレスがなくても Amazon Lex V2 API と通信できます。VPC と Amazon Lex V2 との間のトラフィックは、Amazon ネットワークを離れません。

各インターフェースエンドポイントは、サブネット内の 1 つ以上の Elastic Network Interface によって表されます。

詳細については、「Amazon VPC ユーザーガイド」の「インターフェイス VPC エンドポイント (AWS PrivateLink)」を参照してください。

Amazon Lex V2 VPC エンドポイントに関する考慮事項

Amazon Lex V2 用の VPC エンドポイントを設定する前に、「Amazon VPC ユーザーガイド」の「インターフェイスエンドポイントのプロパティと制限」を確認してください。

Amazon Lex V2 は、VPC からのすべての API アクションの呼び出しをサポートしています。

Amazon Lex V2 用のインターフェイス VPC エンドポイントの作成

Amazon Lex V2 サービスの VPC エンドポイントを作成できます。 AWS Command Line Interface 詳細については、「Amazon VPC ユーザーガイド」のインターフェースエンドポイントの作成を参照してください。

Amazon Lex V2 用の VPC エンドポイントを作成するには、次のサービス名を使用します。

- com.amazonaws.*region*.models-v2-lex
- com.amazonaws.*region*.runtime-v2-lex

エンドポイントのプライベート DNS を有効にすると、リージョンのデフォルト DNS 名 (`runtime-v2-lex.us-east-1.amazonaws.com` など) を使用して、Amazon Lex V2 への API リクエストを実行できます。

詳細については、「Amazon VPC ユーザーガイド」の「インターフェイスエンドポイントを介したサービスへのアクセス」を参照してください。

Amazon Lex V2 用の VPC エンドポイントポリシーの作成

VPC エンドポイントに Amazon Lex V2 へのアクセスをコントロールするエンドポイントポリシーをアタッチできます。このポリシーでは、以下の情報を指定します。

- アクションを実行できるプリンシパル。
- 実行可能なアクション。
- アクションを実行できるリソース。

詳細については、「Amazon VPC ユーザーガイド」の「VPC エンドポイントでサービスへのアクセスを制御する」を参照してください。

例: Amazon Lex V2 アクションの VPC エンドポイントポリシー

Amazon Lex V2 のエンドポイントポリシーの例を次に示します。このポリシーは、エンドポイントにアタッチされると、すべてのリソースのすべてのプリンシパルに対して、登録されている Amazon Lex V2 アクションへのアクセスを許可します。

```
{
   "Statement":[
      {
         "Principal":"*",
         "Effect":"Allow",
         "Action":[
            "lex:RecognizeText",
            "lex:RecognizeUtterance",
            "lex:StartConversation",
            "lex:DeleteSession",
            "lex:GetSession",
            "lex:DeleteSession"
         ],
         "Resource":"*"
      }
   ]
}
```

SDK を使用した Amazon Lex V2 のコード例 AWS SDKs

次のコード例は、 AWS Software Development Kit (SDK) で Amazon Lex V2 を使用する方法を示しています。

シナリオは、1 つのサービス内から、または他の AWS のサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Amazon Lex V2 の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

コードの例

- SDK を使用した Amazon Lex V2 のシナリオ AWS SDKs
  - ウェブサイトの訪問者を引き付ける Amazon Lex V2 チャットボットを作成する

SDK を使用した Amazon Lex V2 のシナリオ AWS SDKs

次のコード例は、 AWS SDKs を使用して Amazon Lex V2 で一般的なシナリオを実装する方法を示しています。 V2 これらのシナリオでは、Amazon Lex V2 内で複数の関数を呼び出すか、他の関数と組み合わせて特定のタスクを実行する方法を示します AWS のサービス。各シナリオには、完全なソースコードへのリンクが含まれており、そこからコードの設定方法と実行方法に関する手順を確認できます。

シナリオは、サービスアクションをコンテキストで理解するのに役立つ中級レベルの経験を対象としています。

例

- ウェブサイトの訪問者を引き付ける Amazon Lex V2 チャットボットを作成する

ウェブサイトの訪問者を引き付ける Amazon Lex V2 チャットボットを作成する

次のコード例は、Web サイトの訪問者を引き込むチャットボットを作成する方法を示しています。

Java

SDK for Java 2.x

Amazon Lex V2 API を使用してウェブアプリケーション内に Chatbot を作成し、ウェブサイトの訪問者を引き付ける方法を示します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- Amazon Comprehend
- Amazon Lex V2
- Amazon Translate

JavaScript

SDK for JavaScript (v3)

Amazon Lex V2 API を使用してウェブアプリケーション内に Chatbot を作成し、ウェブサイトの訪問者を引き付ける方法を示します。

完全なソースコードとセットアップと実行の手順については、 デ AWS SDK for JavaScript ベロッパーガイドのAmazon Lex V2 チャットボットの構築」の完全な例を参照してください。

この例で使用されているサービス

- Amazon Comprehend
- Amazon Lex V2
- Amazon Translate

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Amazon Lex V2 の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

ガイドラインとベストプラクティス

以下のガイドラインとベストプラクティスを参考にして、ボットの動作と顧客との対話を最適化してください。

リクエストへの署名

API リファレンス のすべての Amazon Lex V2 のモデル構築および実行時のすべてのリクエストは、リクエストの認証に署名 V4 を使用します。リクエストの認証の詳細については、AWS 全般のリファレンスの「署名バージョン 4 の署名プロセス」を参照してください。

機密情報の保護

ランタイム API オペレーションの RecognizeText と RecognizeUtterance では、必須パラメーターとしてセッション 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、ボットとの会話を一意に識別するために使用します。

ユーザーの発話からスロット値をキャプチャする

Amazon Lex V2 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 12 個の列挙値を持つカスタムタイプ `ZodiacSign` のスロットで、`Aries` ~ `Pisces` です。たとえば、ユーザーが「地球の未来を教えて」と言ったとします。

- Amazon Lex V2 では、次のいずれかのアクションを実行すると、「地球」が ZodiacSign であると推測されます。
  - CreateSlotType オペレーションを使用して、`valueSelectionStrategy` フィールドを `ORIGINAL_VALUE` に設定します。
  - コンソールで [値を拡大] を選択します。
- 次のいずれかのアクションを実行してスロットタイプに定義した値に認識を制限すると、Amazon Lex V2 は値「地球」を認識しません。

- CreateSlotType オペレーションを使用して、valueSelectionStrategy フィールドを TOP_RESOLUTION に設定します。
- コンソールで [スロット値とシノニムに制限] を選択します。

スロット値にシノニムを定義すると、それらはスロット値と同じと認識されます。ただし、シノニムの代わりにスロット値が返されます。

Amazon Lex V2 はこの値をクライアントアプリケーションまたは Lambda 関数に渡すため、フルフィルメントアクティビティで使用する前に、スロット値が有効な値であることを確認する必要があります。

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。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。

スロット値の頭字語

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)

日付と時刻用の組み込みスロット

AMAZON.Date と AMAZON.Time 組み込みスロットタイプは、日時 (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 V2 がリクエストを受け取った日時と、リクエストを処理しているリージョンで解決されます。

AMAZON.Time 組み込みスロットタイプで、ユーザーが時間を午前か午後かを指定しない場合、時間があいまいになります。その場合、Amazon Lex V2 はユーザーに再度プロンプトを表示します。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。

ボットのトレーニングデータのあいまいさ回避

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 V2 がユーザー入力を理解する能力が低下します。ボットに 2 つのインテント (OrderPizzaとOrderDrink) があり、サンプル発話として「注文したい」を含めたとします。ボットを構築しても、Amazon Lex V2 ではこの発話を

特定のインテントにマッピングできません。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 V2 は高い信頼性におけるインテントを選択できません。

同じ発話を持つ 2 つのインテントがランタイムにある場合、入力コンテキストを使用して Amazon Lex V2 が実行時に 2 つのインテントを区別できるようにします。詳細については、「インテントコンテキストのセッティング」を参照してください。

TSTALIASID エイリアスの使用

- ボットの TSTALIASID エイリアスはドラフト版を指し、手動テストにのみ使用してください。Amazon Lex では、TSTALIASID エイリアス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。
- ドラフトバージョンのボットを更新すると、Amazon Lex はボットの TSTALIASID 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シャットダウンします。ドラフト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 TSTALIASID alias のボットを使用しないでください。代わりに、別のバージョンとエイリアス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。ボットのバージョンを変更すると、その変更は TSTALIASID エイリアスによりすぐに反映されます。

クォータ

制限とも呼ばれるサービスクォータは、AWS アカウントで許可されるサービスリソースの最大数です。詳細については、AWS 全般のリファレンスの「AWS サービスクォータ」を参照してください。

一部の Service Quotas は調整または増やすことができます。以下の表の [調整可能] 列を参照してクォータを調整できるかどうかを確認し、[セルフサービス] 列を参照して、Service quotas コンソールからクォータ調整をリクエストできるかどうかを確認してください。セルフサービスではなく調整可能なクォータを増やす サポート には、 にお問い合わせください。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。

> **Note**
>
> 文字制限は Unicode コード単位の数として計算されます。ほとんどの場合、1 つの Unicode 文字は 1 つの Unicode コード単位に相当します。特殊文字の中には 1 単位を超えるものもあり、単位の数はエンコーディングによって異なる場合があります。文字列長の計算の詳細については、こちらのドキュメントを参照してください。

ビルド時のクォータ

ボットの作成時には、次の最大クォータが適用されます。

説明	デフォルト	引き上げ可能	セルフサービス
AWS アカウントあたりのボット数	100	はい	はい
AWS アカウントあたりのボットチャネルの関連付け	5,000	いいえ	該当なし
AWS アカウントあたりの並列ロケールビルド	5	はい	いいえ

説明	デフォルト	引き上げ可能	セルフサービス
ボットネットワークあたりのボット数	5	いいえ	該当なし
ボットあたりのボットネットワーク数	25	いいえ	該当なし
ボットあたりのバージョン	100	いいえ	該当なし
各ボットのロケールごとのインテント	• en-AU、en-GB、および en-US で 1,000 • 他のすべてのロケールで 250	はい	いいえ
各ボットのロケールごとのスロット	• en-AU、en-GB、および en-US で 4,000 • 他のすべてのロケールで 2000	いいえ	該当なし
ボットロケールあたりのカスタムスロットタイプ	• en-AU、en-GB、および en-US で 250 • 他のすべてのロケールで 100	いいえ	該当なし
ボットロケールあたりのカスタムスロットタイプ値とシノニム	50,000	いいえ	該当なし

説明	デフォルト	引き上げ可能	セルフサービス
ボットロケールあたりのサンプル発話の合計文字	• en-AU、en-GB、および en-US で 2,000,000 • 他のすべてのロケールで 200,000	いいえ	該当なし
ボットエイリアスあたりのボットチャネルの関連付け	10	いいえ	該当なし
インテントあたりのスロット	100	いいえ	該当なし
インテントあたりのサンプル発話	1,500	はい	はい
サンプル発話あたりの文字	500	いいえ	該当なし
テキストレスポンスの長さ	4,000	いいえ	該当なし
スロットあたりのサンプル発話	10	はい	はい
サンプルスロットの発話あたりの文字数	500	いいえ	該当なし
スロットあたりのプロンプト数	30	いいえ	該当なし
カスタムスロットタイプあたりの値とシノニム	10,000	いいえ	該当なし
カスタムスロットタイプ値あたりの文字	500	いいえ	該当なし

説明	デフォルト	引き上げ可能	セルフサービス
チャンネルアソシエーション名のキャラクター	100	いいえ	該当なし
リージョンごとのアカウント内のすべてのボットでの同時に実行される自動チャットボットデザイナーの分析ジョブ数	10	いいえ	該当なし
カスタム文法、スロットタイプ XML ファイルのサイズ	100 KB	いいえ	該当なし

ランタイムクォータ

実行時に次の最大クォータが適用されます。

説明	デフォルト	引き上げ可能	セルフサービス
RecognizeText と RecognizeUtterance の入力テキストサイズ	1024 文字	いいえ	該当なし
`RecognizeUtterance` 操作での音声入力の長さ	55 秒	はい	いいえ
`RecognizeUtterance` ヘッダーのサイズ	16 KB	いいえ	該当なし

説明	デフォルト	引き上げ可能	セルフサービス
RecognizeUtterance に対するリクエストヘッダーとセッションヘッダーの組み合わせのサイズ	12 KB	いいえ	該当なし
RecognizeText、RecognizeUtterance、または StartConversation TestBotalias のテキストモードでの同時会話数の最大値	2	いいえ	該当なし
RecognizeText、RecognizeUtterance、または StartConversation その他のエイリアスでのテキストモードでの同時会話数の最大値	50	はい	いいえ
TestBotalias の RecognizeUtterance の同時音声モード会話数の最大値	2	いいえ	該当なし

説明	デフォルト	引き上げ可能	セルフサービス
エイリアスの RecognizeUtterance の同時音声モード会話数の最大値	125	はい	いいえ
TestBotalias の StartConversation の同時音声モード会話数の最大値	2	いいえ	該当なし
エイリアスの StartConversation の同時音声モード会話数の最大値	400	はい	いいえ
同時セッション管理オペレーションの最大数 (PutSession 、GetSession 、または DeleteSession) TestBotaliaS を使用する場合	2	いいえ	該当なし
同時セッション管理オペレーションの最大数 (PutSession 、GetSession 、または DeleteSession) 他のエイリアスを使用する場合	50	はい	いいえ

説明	デフォルト	引き上げ可能	セルフサービス
Lambda 関数の最大入力サイズ	12 KB	いいえ	該当なし
Lambda 関数の最大出力サイズ	50 KB	いいえ	該当なし
Lambda 関数の最大出力サイズ (base-64 エンコーディング後)	12 KB	いいえ	該当なし
Lambda 関数の最大タイムアウト	30 秒	はい	いいえ
1 回の会話の最大継続時間	15 分	いいえ	いいえ
オーディオの最大長	55 秒	いいえ	いいえ

Amazon Lex V1 から V2 への移行ガイド

Amazon Lex V2 コンソールと API により、ボットの構築と管理が容易になります。ボットを移行する際に、Amazon Lex V2 API の改善点を理解するため、こちらのガイドをご利用ください。

Amazon Lex コンソールまたは API を使用してボットを移行します。詳細については、Amazon Lex デベロッパーガイド の「ボットを移行する」を参照してください。

Amazon Lex V2 の概要

1つのボットに複数の言語を追加できるので、単一のリソースとして管理できます。シンプルな情報アーキテクチャにより、ボットのバージョンを効率的に管理できます。「会話フロー」、ボット設定の部分的な保存、発話の一括アップロードなどの機能により、より柔軟な対応が可能です。

ボット内の複数の言語

Amazon Lex V2 API を使用して、複数の言語を追加できます。各言語を個別に追加、変更、および構築します。スロットタイプなどのリソースは、言語レベルでスコープが設定されています。異なる言語間をすばやく移動して、会話を比較してより正確にすることができます。コンソールの 1 つのダッシュボードを使用して、すべての言語の発話を確認し、より迅速な分析と繰り返しが可能になります。ボットオペレータは、1 つのボット設定ですべての言語のアクセス権限とロギング操作を管理できます。Amazon Lex V2 ボットと会話するには、ランタイムパラメータとして言語を指定する必要があります。詳細については、「Amazon Lex V2 でサポートされている言語と口ケール」を参照してください。

簡略化された情報アーキテクチャ

Amazon Lex V2 API は、インテントとスロットタイプが言語にスコープされた簡略化された情報アーキテクチャ (IA) に従います。インテントやスロットタイプなどのリソースが個別にバージョン管理されないように、ボットレベルでバージョン管理を行います。デフォルトでは、ボットは ドラフト バージョンで作成されます。ドラフトバージョンは、変更可能で、変更のテストに使用されます。ドラフトバージョンから番号付きスナップショットを作成することができます。バージョンに含める言語を選択します。ボット内のすべてのリソース (言語、インテント、スロットタイプ) は、ボットのバージョンを作成する際にアーカイブされます。詳細については、「バージョン」を参照してください。

ビルダーの生産性向上

ビルダーの生産性向上ツールや機能が追加され、ボットの設計プロセスをより柔軟にコントロールできるようになりました。

部分的な設定の保存

Amazon Lex V2 API では、開発中に部分的に変更した内容を保存することができます。たとえば、削除されたスロットタイプを参照するスロットを保存できます。この柔軟性により、作業内容を保存し、後でその作業に戻ることができます。ボットを構築する前に、これらの変更を解決することができます。Amazon Lex V2 では、スロット、バージョン、エイリアスに部分保存を適用することができます。

リソースの名前変更

Amazon Lex V2 では、リソースの作成後に名前を変更できます。リソース名を使用して、各リソースにユーザーフレンドリーなメタデータを関連付けます。Amazon Lex V2 API では、すべてのリソースに一意の 10 文字のリソース ID が割り当てられます。すべてのリソースにリソース名があります。次のリソースの名前を変更できます。

- ボット
- Intent
- スロットタイプ
- スロット
- エイリアス

リソース ID を使用して、リソースの読み取りと変更ができます。 AWS Command Line Interface または Amazon Lex V2 API を使用して Amazon Lex V2 を操作する場合は、特定のコマンドにリソース IDs が必要です。

Lambda 関数の管理の簡素化

Amazon Lex V2 API では、インテントごとに関数を定義するのではなく、言語ごとに 1 つの Lambda 関数を定義します。Lambda 関数は言語のエイリアスで設定され、ダイアログとフルフィルメントコードのフックの両方に使用されます。ダイアログとフルフィルメントコードのフックは、インテントごとに個別に有効または無効を選択することができます。詳細については、「AWS Lambda関数を Amazon Lex V2 ボットに統合する」を参照してください。

詳細な設定

Amazon Lex V2 API は、音声およびインテントの分類の信頼度の閾値をボットから言語スコープに移動します。センチメント分析フラグは、ボットスコープからエイリアススコープに移動します。ボットスコープでのセッションタイムアウトとプライバシー設定、エイリアススコープの会話ログは変更されません。

デフォルトのフォールバックインテント

Amazon Lex V2 API は、言語を作成するときに、デフォルトのフォールバックインテントを追加します。特定のエラー処理プロンプトの代わりに、ボットのエラー処理を設定するために使用します。

セッション変数の更新の最適化

Amazon Lex V2 API では、セッション API に依存することなく、RecognizeText および RecognizeUtterance オペレーションで直接セッションの状態を更新することができます。

を使用した Amazon Lex V2 リソースの作成 AWS CloudFormation

Amazon Lex V2 は AWS CloudFormation、AWS リソースのモデル化とセットアップに役立つサービスである と統合されているため、リソースとインフラストラクチャの作成と管理に費やす時間を短縮できます。必要なすべての AWS リソース (Amazon Lex V2 チャットボットなど) を記述するテンプレートを作成し、それらのリソースを CloudFormation プロビジョニングして設定します。

を使用すると CloudFormation、テンプレートを再利用して Amazon Lex V2 リソースを一貫して繰り返しセットアップできます。リソースを一度記述し、複数の AWS アカウント およびリージョンで同じリソースを何度もプロビジョニングします。

Amazon Lex V2 と CloudFormation テンプレート

Amazon Lex V2 および関連サービスのリソースをプロビジョニングして設定するには、CloudFormation テンプレートについて理解しておく必要があります。テンプレートは、JSON や YAML でフォーマットされたテキストファイルです。これらのテンプレートには、CloudFormation スタックにプロビジョニングしたいリソースを記述します。JSON または YAML に慣れていない場合は、 CloudFormation デザイナー を使用して CloudFormation テンプレートの使用を開始できます。詳細については、 AWS CloudFormation ユーザーガイドのCloudFormation 「デザイナーとは」を参照してください。

Amazon Lex V2 は、 CloudFormationで以下のリソースの作成をサポートしています:

- AWS::Lex::Bot
- AWS::Lex::BotAlias
- AWS::Lex::BotVersion
- AWS::Lex::ResourcePolicy

これらのリソースの JSON テンプレートと YAML テンプレートの例を含む詳細については、「AWS CloudFormation ユーザーガイドの「Amazon Lex V2 リソースタイプのリファレンス」を参照してください。

の詳細 CloudFormation

詳細については CloudFormation、次のリソースを参照してください。

- AWS CloudFormation
- AWS CloudFormation ユーザーガイド
- CloudFormation API リファレンス
- AWS CloudFormation コマンドラインインターフェイスユーザーガイド

Amazon Lex V2 のドキュメント履歴

- ドキュメント最新更新日: 2025 年 10 月 27 日

次の表に、Amazon Lex V2 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を受け取るには、RSS フィードにサブスクライブできます。

変更	説明	日付
機能更新	待機と続行が追加の口ケールで利用可能になりました (ca-ES、de-AT、de-DE、es-419、es-ES、es-US、fr-CA、fr-FR、it-IT、ja-JP、ko-KR、pt-BR、pt-PT、zh-CN、zh-HK)。詳細については、「一時停止中に、Amazon Lex V2 ボットによるユーザーの追加の情報提供の待機を可能にする」を参照してください。	2025 年 10 月 27 日
機能更新	AMAZON.Currency および AMAZON.Confirmation の組み込みスロットタイプが、すべての口ケールをサポートするようになりました。詳細については、「組み込みスロットタイプ」を参照してください。	2025 年 8 月 18 日
ドキュメントの更新	アシスト付き NLU 機能のドキュメントが更新され、例と説明が改善されました。詳細については、「アシスト付き NLU」を参照してください。	2025 年 8 月 4 日

新機能	AMAZON.BedrockAgentIntent 組み込みインテントを使用して、Amazon Lex V2 ボットを Amazon Bedrock エージェントおよびナレッジベースに接続できます。詳細については、「AMAZON.BedrockAgentIntent 組み込みインテント」を参照してください。	2025 年 7 月 3 日
AWS 管理ポリシーの更新	Amazon Lex V2 は、グローバルレジリエンシーを使用する場合にタグと ResourceBasedPolicy のレプリケーションを許可するようにポリシーを更新しました。「AmazonLexReplicationPolicy」を参照してください。	2025 年 6 月 24 日
新機能	アシスト付き NLU を使用して、Amazon Lex V2 のインテント分類とスロット解決を改善できるようになりました。この機能は、大規模言語モデル (LLM) を使用して、ボットが設定したインテントとスロット内に留まりながら精度を向上させます。詳細については、「アシスト付き NLU」を参照してください。	2025 年 6 月 11 日
機能 (2025-06-03) の更新	Amazon Lex V2 で、カスタム語彙で 17 の新しい言語を使用できます。詳細については、「カスタム語彙」を参照してください。	2025 年 6 月 3 日カスタム語彙で 17 の新しい言語を使用できます

新機能 (2024-12-01)	QinConnect 組み込みインテントを使用して、Amazon Lex V2 ボットを Amazon Connect に接続できます。詳細については、「QinConnect 組み込みインテント」を参照してください。	2024 年 12 月 1 日
新機能 (2024-10-01)	Amazon Lex V2 では、米国およびスペインのロケールに AMAZON.Currency の組み込みスロットを使用できます。詳細については、「AMAZON.Currency」を参照してください。	2024 年 10 月 1 日
機能 (2024-07-26) の更新	Amazon Lex V2 で、生成 AI 向けの QnA 組み込みインテントが、Bedrock ナレッジベースとガードレールをサポートするようになりました。詳細については、「AMAZON.QnAIntent」を参照してください。	2024 年 7 月 26 日
機能 (2024-07-10) の更新	Amazon Lex V2 で、生成 AI 向けの QnA 組み込みインテントが、Anthropic Claude 3 Haiku および Anthropic Claude 3 Sonnet モデルをサポートするようになりました。詳細については、「生成 AI を使用してボットの作成とパフォーマンスを最適化する」を参照してください。	2024 年 7 月 10 日

AWS 管理ポリシー (2024-05-10) の更新	Amazon Lex V2 は AmazonLexReadOnly マネージドポリシーに新しいアクセス許可を追加し、他のリージョンにレプリケートされたボットリソースに読み取りアクセスを付与しました。	2024 年 5 月 10 日
AWS 管理ポリシー (2024-04-15) の更新	Amazon Lex V2 は AmazonLexFullAccess マネージドポリシーに新しいアクセス許可を追加し、他のリージョンにレプリケートされたボットリソースを更新できるようになりました。	2024 年 4 月 22 日
リージョンの拡大	Amazon Lex V2 が AWS GovCloud (米国西部) (us-gov-west-1) で利用可能になりました。	2024 年 3 月 22 日
AWS 管理ポリシー (2024-03-07) の更新	Amazon Lex V2 は AmazonLexReplicationPolicy マネージドポリシーに新しいアクセス許可を追加し、他のリージョンにレプリケートされたボットリソースを更新できるようになりました。	2024 年 3 月 7 日
新しい 関数	fn.LENGTH() 関数を使用して、Amazon Lex V2 の文字列値の長さを決定できます。詳細については、「条件付き分岐 - 関数」を参照してください。	2024 年 3 月 4 日

機能 (2024-02-28) の更新	Amazon Lex V2 の生成 AI 機能用の QnA 組み込みスロットが GA になりました。詳細については、「生成 AI を使用してボットの作成とパフォーマンスを最適化する」を参照してください。	2024 年 2 月 1 日
AWS 管理ポリシー (2024-02-08) の更新	Amazon Lex V2 は AmazonLexFullAccess マネージドポリシーに新しいアクセス許可を追加し、ボットリソースを他のリージョンにレプリケートできるようになりました。	2024 年 2 月 8 日
新しい マネージドポリシー	Amazon Lex V2 は、他のリージョンにボットリソースをレプリケートするアクセス許可を付与するマネージドポリシーを追加しました。詳細については、「AmazonLexReplicationPolicy」を参照してください。	2024 年 2 月 8 日
新機能 (2024-02-08)	グローバルレジリエンシーを使用して、Amazon Lex V2 の 2 番目の AWS リージョンにボットをレプリケートできます。詳細については、「グローバルレジリエンシー」を参照してください。	2024 年 2 月 8 日

新機能 (2023-11-29)	Amazon Lex V2 で生成 AI 機能を利用できるようになりました。詳細については、「生成 AI を使用してボットの作成とパフォーマンスを最適化する」を参照してください。	2023 年 11 月 29 日
新機能 (2023-11-08)	Amazon Lex V2 では、選択的ロギングを使用して、インテントまたはスロットレベルでテキストや音声をキャプチャできるようになりました。詳細については、「選択的ロギング」を参照してください。	2023 年 11 月 8 日
新機能 (2023-08-17)	Amazon Lex V2 では、組み込みスロットで `AMAZON.Confirmation` を使用して「はい/いいえ/たぶん/わからない」の各応答を判断できるようになりました。詳細については、「組み込みスロットタイプ」を参照してください。	2023 年 8 月 17 日
新機能 (2023-07-18)	分析ダッシュボードを使用して、インテントとスロットのパフォーマンスメトリクスのほか、他の会話メトリクスを表示できます。詳細については、「Analytics」を参照してください。	2023 年 7 月 18 日

新機能 (2023-06-06)	Test Workbench を使うと、ボットの精度とフルフィルメント成功率を向上させることができます。詳細は、「Test Workbench」を参照してください。	2023 年 6 月 6 日
新機能 (2023-02-23)	一部の人気業種向けのテンプレートからボットを作成できるようになりました。詳細については、「ボットテンプレート」を参照してください。	2023 年 2 月 23 日
新機能 (2023-02-09)	複数のボットを組み合わせてボットのネットワークを構築し、統合されたカスタマーエクスペリエンスを構築できるようになりました。詳細については、「ボットのネットワーク」を参照してください。	2023 年 2 月 9 日
新機能 (2022-12-06)	Amazon Lex V2 では、湾岸アラビア語 (アラブ首長国連邦)、広東語 (香港)、フィンランド語 (フィンランド)、ノルウェー語 (ノルウェー)、ポーランド語 (ポーランド)、スウェーデン語 (スウェーデン) の各ロケールがサポートされるようになりました。詳細については、「Amazon Lex V2 でサポートされている言語とロケール」を参照してください。	2022 年 12 月 6 日

AWS 管理ポリシーに更新	Amazon Lex V2 は AmazonLexReadOnly マネージドポリシーに新しいアクセス許可を追加し、カスタム語彙項目を表示できるようになりました。	2022 年 11 月 29 日
新機能 (2022-11-07)	Amazon Lex V2 では、コンソールまたは API を使用して音声テキストへの出力をカスタマイズすることで、フレーズや単語の代替表現を表示できます。詳細については、「音声認識用のカスタム語彙の作成」を参照してください。	2022 年 11 月 7 日
新機能 (2022-10-28)	Amazon Lex V2 では、音声認識中にフレーズがどの程度ブーストされるかを表す重み属性をアイテム要素に追加できます。詳細については、「文法の重み」を参照してください。	2022 年 10 月 28 日
新機能 (2022-10-19) (2022-10-19) - 2	Amazon Lex V2 では、最終的な音声書き起こし出力でフレーズや単語の代替表現を表示できます。詳細については、「音声認識用のカスタム語彙の作成」を参照してください。	2022 年 10 月 19 日

新機能 (2022-10-19) (2022-10-19)	Amazon Lex V2 では、AMAZON.FreeFormInput を使用して、エンドユーザーからの単語や文字で構成された自由形式の入力をキャプチャできます。詳細については、「組み込みスロットタイプ」を参照してください。	2022 年 10 月 19 日
新機能 (2022-10-14)	Amazon Lex V2 がヒンディー語 (インド) ロケールとオランダ語 (オランダ) ロケールをサポートするようになりました。詳細については、「Amazon Lex V2 でサポートされている言語と口ケール」を参照してください。	2022 年 10 月 14 日
新機能 (2022-09-22)	Amazon Lex V2 がユーザー入力を管理する方法が更新されました。会話フローの任意の時点で Amazon Lex V2 がテキスト、音声、または DTMF 入力を受け入れるかどうかを選択できるようになりました。問い合わせ属性の詳細については、「設定可能な属性」を参照してください。	2022 年 9 月 22 日

新機能 (2022-09-14)	Amazon Lex V2 が会話フローを管理する方法が更新されました。ビジュアル会話ビルダーは、会話パスを簡単に設計および視覚化できるドラッグアンドドロップの会話ビルダーです。詳細については、「ビジュアル会話ビルダー」を参照してください。	2022 年 9 月 14 日
新機能 (2022-09-09)	Amazon Lex V2 が複雑なスロットを構築する方法が更新されました。スロット内に複雑なサブスロットを作成して、複雑な会話デザインのインテントを管理できるようになりました。詳細については、「複合スロットの作成」を参照してください。	2022 年 9 月 9 日
新機能 (2022-08-17)	Amazon Lex V2 がユーザーとの会話パスを管理する方法が更新されました。会話における次のステップの順序を決めることで、複雑な会話パスを作成できるようになりました。詳細については、「会話パスの作成」を参照してください。	2022 年 8 月 17 日

新機能 (2022-07-05)	Amazon Lex V2 がユーザーとの会話を管理する方法が更新されました。プロンプトの順序を決めることで、複雑な会話を作成できるようになりました。詳細については、「プロンプトの設定」を参照してください。	2022 年 7 月 5 日
新機能 (2022-05-03)	Amazon Lex V2 がユーザーとの会話を管理する方法が更新されました。条件を使用して複雑な会話を作成できるようになりました。詳細については、「新しい会話フローについて」を参照してください。	2022 年 5 月 3 日
新機能 (2022-03-22)	組み込み文法スロットタイプ用の業界文法の例を追加しました。詳細については、「業界文法」を参照してください。	2022 年 3 月 22 日
新機能 (2022-03-18)	Amazon Lex V2 と Amazon Chime SDK の統合に関するドキュメントを追加しました。詳細については、「Amazon Chime SDK」を参照してください。	2022 年 3 月 18 日

新機能 (2022-01-27)	Amazon Lex V2 が音声文字起こしの信頼スコアを提供するようになりました。このスコアを使用して、ユーザーからの正しい応答を決定できます。詳細については、「音声文字起こし信頼度スコアの使用」を参照してください。	2022 年 1 月 27 日
新機能 (2022-01-13)	ボットの精度を向上させるために、スロットにコンテキストヒントや動的ヒントを追加できるようになりました。詳細については、「ヒントを使った精度の向上」を参照してください。	2022 年 1 月 13 日
新機能 (2022-01-12)	Amazon Lex V2 では、音声入力の音声認識を改善するためのカスタムボキャブラリーのサポートが追加されました。詳細については、「音声認識を向上させるためのカスタム語彙の作成」を参照してください。	2022 年 1 月 12 日
新機能 (2022-01-07)	Amazon Lex V2 が をサポートするようになりましたAWS PrivateLink。詳細については、「VPC エンドポイント (AWS PrivateLink)」を参照してください。	2022 年 1 月 7 日

新機能 (2022-01-03)	Amazon Lex V2 でカタルーニャ語 (スペイン) ロケールをサポートするようになりました。詳細については、「Amazon Lex V2 でサポートされている言語とロケール」を参照してください。	2022 年 1 月 3 日
新機能 (2021-12-20) (2021-12-20) - 2	AWS CloudFormationが Amazon Lex V2 をサポートするようになりました。V2 詳細については、「AWS CloudFormation リソース」を参照してください。	2021 年 12 月 20 日
新機能 (2021-12-20) (2021-12-20)	独自のカスタム文法を使用してスロットタイプを作成できるようになりました。詳細については、「カスタム文法スロットタイプの使用」を参照してください。	2021 年 12 月 20 日
新機能 (2021-12-16)	Amazon Lex V2 で、ポルトガル語 (ブラジル)、ポルトガル語 (ポルトガル)、北京語 (PRC)、北京語 (PRC) がサポートされるようになりました。詳細については、「Amazon Lex V2 でサポートされている言語とロケール」を参照してください。	2021 年 12 月 16 日

新機能 (2021-12-01)	Amazon Lex V2 では、コンタクトセンターの文字起こしからチャットボット作成を開始できるように、Automated Chatbot Designer のプレビューが提供されます。詳細については、「自動 Chatbot デザイナの使用 (プレビュー)」を参照してください。	2021 年 12 月 1 日
新機能 (2021-11-19) (2021-11-19) - 2	Amazon Polly ニューラルテキスト読み上げ (NTTS) の音声をユーザーとの音声会話に使用できるようになりました。詳細については、Amazon Polly の音声 を参照してください。	2021 年 11 月 19 日
新機能 (2021-11-19) (2021-11-19)	Amazon Lex V2 の理解が難しいスロット値を入力するために、文字ごとのスペルとスペルバイワードスタイルを使用できるようになりました。詳細については、「スペルスタイルを使用してスロット値をキャプチャする」を参照してください。	2021 年 11 月 19 日
新機能 (2021-11-09)	Amazon Lex V2 が英語 (南アフリカ) ロケールをサポートするようになりました。詳細については、「Amazon Lex V2 でサポートされている言語とロケール」を参照してください。	2021 年 11 月 9 日

新機能 (2021-11-05)	Amazon Lex V2 がドイツ語 (オーストリア) ロケールをサポートするようになりました。詳細については、「Amazon Lex V2 でサポートされている言語と口ケール」を参照してください。	2021 年 11 月 5 日
新機能 (2021-10-07)	フルフィルメント関数の開始時および関数の実行中に定期的に再生される更新メッセージをユーザーに提供できるようになりました。また、機能の完了時にフルフィルメントのステータスをユーザーに知らせるメッセージを作成することもできます。詳細については、「フルフィルメント進捗の更新の設定」を参照してください。	2021 年 10 月 7 日
リージョンの拡張 (2021-09-22)	Amazon Lex V2 が、アフリカ (ケープタウン) (af-south-1) およびアジアパシフィック (ソウル) (ap-northeast-2) で利用可能になりました。	2021 年 9 月 22 日
新機能 (2021-09-22)	ユーザーがボットに送信する発話の統計情報を表示できるようになりました。詳細については、「発話統計の表示」を参照してください。	2021 年 9 月 22 日

新機能 (2021-09-09)	Amazon Lex V2 が韓国語 (韓国) ロケールをサポートするようになりました。詳細については、「Amazon Lex V2 でサポートされている言語とロケール」を参照してください。	2021 年 9 月 9 日
新機能 (2021-07-27)	Amazon Lex V2 では、英国の郵便番号にスロットタイプが組み込まれています。詳細については、「Amazon.UK 郵便番号」を参照してください。	2021 年 7 月 27 日
新機能 (2021-07-15)	Amazon Lex V2 が英語 (インド) ロケールをサポートするようになりました。詳細については、「Amazon Lex V2 でサポートされている言語とロケール」を参照してください。	2021 年 7 月 15 日
新機能 (2021-07-13)	Amazon Lex V2 では、Amazon Lex V1 から Amazon Lex V2 API にボットを移行するツールが提供されるようになりました。詳細については、「Amazon Lex デベロッパーガイド」の「ボットの移行」を参照してください。	2021 年 7 月 13 日

新機能 (2021-06-26)	スロット値を null に設定し、会話フローをループバックするように設定することで、すでに入力されているスロットを再誘発できるように Amazon Lex V2 ボットを設定できるようになりました。この機能により、顧客のレスポンスに基づくより動的な会話が可能になります。詳細については、「スロットの再誘発」を参照してください。	2021 年 6 月 26 日
新しいドキュメント	Amazon Lex V2 で、ボット開発に関する包括的なガイドラインとベストプラクティスが提供されるようになり、より効果的な会話体験を実現できるようになりました。詳細については、「Amazon Lex V2 ボット開発のガイドライン」を参照してください。	2021 年 6 月 26 日
新機能 (2021-06-15)	Amazon Lex V2 では、英語 (米国) 言語で 1 つのスロットに複数の値を受け入れることができるようになりました。詳細については、「スロットで複数の値を使用する」を参照してください。	2021 年 6 月 15 日
新機能 (2021-06-11)	英語用のより大きなボットを構築できるようになりました。詳細については、「クォータ」を参照してください。	2021 年 6 月 11 日

新機能 (2021-05-20)	Amazon Lex V2 リソースベースのポリシーを使用して、ボットとボットのエイリアスへのアクセスを管理します。詳細については、「Amazon Lex V2 のリソースベースのポリシー」を参照してください。	2021 年 5 月 20 日
新機能 (2021-05-18)	Amazon Lex V2 では、ボットとボットのロケールをインポートおよびエクスポートできるようになりました。この機能を使用して、アカウントと AWS リージョン間でボットとボットロケールをコピーできます。詳細については、「インポートとエクスポート」を参照してください。	2021 年 5 月 18 日
リージョン拡張 (2021-05-17)	Amazon Lex V2 が、カナダ (中部) (ca-central-1) で利用可能になりました。	2021 年 5 月 17 日
新機能 (2021-04-01)	Amazon Lex V2 が日本語 (日本) ロケールをサポートするようになりました。詳細については、「Amazon Lex V2 でサポートされている言語とロケール」を参照してください。	2021 年 4 月 1 日

新機能 (2021-03-12)	Amazon Lex V2 では、3 つの新しい組み込みスロットタイプとして、AMAZON.City 、AMAZON.Country 、AMAZON.State をサポートするようになりました。	2021 年 3 月 12 日
新しいガイド	これは Amazon Lex V2 ユーザーガイドの最初のリリースです。	2021 年 1 月 21 日

API リファレンス

API リファレンスは独立したドキュメントになっています。

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。

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。