

---

# Amazon Lex

## Guia do desenvolvedor



## Amazon Lex: Guia do desenvolvedor

Copyright © 2019 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

## Table of Contents

|   |     |
|---|-----|
| O que é o Amazon Lex?                                       | 1   |
| Você é um usuário iniciante do Amazon Lex?                  | 2   |
| Como ele funciona   | 3   |
| Modelo de programação                                       | 4   |
| Operações de API de criação de modelo                       | 5   |
| Operações de API de runtime                                 | 6   |
| Funções do Lambda como ganchos de código                    | 6   |
| Permissões de serviço                                       | 9   |
| Criação de políticas com base em recursos para o AWS Lambda | 9   |
| Excluindo funções vinculadas a serviço                      | 10  |
| Gerenciamento de mensagens                                  | 10  |
| Tipos de mensagens  | 10  |
| Contextos para a configuração de mensagens                  | 11  |
| Formatos de mensagem suportados                             | 15  |
| Grupos de mensagens   | 15  |
| Cartões de resposta   | 16  |
| Gerenciar contexto da conversa                              | 20  |
| Definição dos atributos da sessão                           | 20  |
| Definição de atributos de solicitação                       | 21  |
| Definição do tempo limite da sessão                         | 23  |
| Compartilhamento de informações entre intenções             | 24  |
| Configuração de atributos complexos                         | 24  |
| Gerenciamento de sessões                                    | 25  |
| Alternância entre intenções                                 | 26  |
| Como retomar uma intenção anterior                          | 27  |
| Como iniciar uma nova sessão                                | 27  |
| Validação de valores de slot                                | 27  |
| Opções de implantação                                       | 28  |
| Intenções integradas e tipos de slot                        | 28  |
| Intenções integradas  | 28  |
| Tipos de slot integrados                                    | 28  |
| Tipos de slot personalizados                                | 33  |
| Conceitos básicos   | 35  |
| Etapa 1: Configurar uma conta                               | 35  |
| Cadastre-se na AWS  | 35  |
| Criar um usuário da IAM                                     | 36  |
| Próxima etapa   | 36  |
| Etapa 2: Configurar a AWS CLI                               | 36  |
|   | 37  |
| Etapa 3: Conceitos básicos (console)                        | 37  |
| Exercício 1: Criar um bot usando um esquema                 | 38  |
| Exercício 2: Criar bot personalizado                        | 64  |
| Exercício 3: publique uma versão e crie um alias            | 75  |
| Etapa 4: Conceitos básicos (AWS CLI)                        | 76  |
| Exercício 1: Crie um bot                                    | 76  |
| Exercício 2: Adicione um novo enunciado                     | 88  |
| Exercício 3: Adicionar uma função do Lambda                 | 92  |
| Exercício 4: Publique uma versão                            | 95  |
| Exercício 5: Crie um alias                                  | 99  |
| Exercício 6: Limpar   | 100 |
| Controle de versão e aliases                                | 102 |
| Controle de versão  | 102 |
| A versão \$LATEST   | 102 |
| Publicação de uma versão de recurso do Amazon Lex           | 102 |

|   |     |
|---|-----|
| Atualização de um recurso do Amazon Lex .....                                       | 103 |
| Exclusão de um recurso ou de uma versão do Amazon Lex .....                         | 103 |
| Aliases .....   | 104 |
| Uso de funções do Lambda .....  | 106 |
| Evento de entrada de função do Lambda e formato de resposta .....                   | 106 |
| Formato de eventos de entrada .....   | 106 |
| Formato de resposta .....   | 109 |
| Esquemas do Amazon Lex e do AWS Lambda .....  | 113 |
| Implantação de bots .....   | 114 |
| Implantação de um bot do Amazon Lex em uma plataforma de sistema de mensagens ..... | 114 |
| Integração com o Facebook .....   | 116 |
| Integrar com o Kik .....  | 118 |
| Integração com o Slack .....  | 121 |
| Integração com o SMS do Twilio .....  | 125 |
| Implantação de um bot do Amazon Lex em aplicativos móveis .....                     | 128 |
| Importar e exportar .....   | 129 |
| Exportação e importação em formato do Amazon Lex .....                              | 129 |
| Exportação em formato do Amazon Lex .....   | 130 |
| Importação em formato do Amazon Lex .....   | 130 |
| Formato JSON para importação e exportação .....                                     | 131 |
| Exportar para uma habilidade do Alexa .....   | 134 |
| Exemplos de bot .....   | 135 |
| Exemplo de bot: ScheduleAppointment .....   | 135 |
| Visão geral do esquema de bot (ScheduleAppointment) .....                           | 137 |
| Visão geral do esquema da função do Lambda (lex-make-appointment-python) .....      | 138 |
| Etapa 1: Criar um bot do Amazon Lex .....   | 138 |
| Etapa 2: Criar uma função do Lambda .....   | 140 |
| Etapa 3: atualizar a intenção - configuração de um hook de código .....             | 141 |
| Etapa 4: implantar o bot na plataforma do Facebook Messenger .....                  | 142 |
| Detalhes do fluxo de informações .....  | 142 |
| Exemplo de bot: BookTrip .....  | 154 |
| Etapa 1: análise de esquema .....   | 155 |
| Etapa 2: Criar um bot do Amazon Lex .....   | 157 |
| Etapa 3: Criar uma função do Lambda .....   | 160 |
| Etapa 4: Adicionar a função do Lambda como um gancho de código .....                | 160 |
| Detalhes do fluxo de informações .....  | 163 |
| Exemplo: uso de um cartão de resposta .....   | 176 |
| Exemplo: Atualização de enunciados .....  | 180 |
| Exemplo: Integração com um site .....   | 181 |
| Segurança .....   | 182 |
| Proteção de dados .....   | 182 |
| Criptografia em repouso .....   | 183 |
| Criptografia em trânsito .....  | 184 |
| Gerenciamento de chaves .....   | 184 |
| Identity and Access Management .....  | 184 |
| Público .....   | 184 |
| Autenticação com identidades .....  | 185 |
| Gerenciamento do acesso usando políticas .....                                      | 187 |
| Saiba mais .....  | 188 |
| Como o Amazon Lex funciona com o IAM .....  | 188 |
| Exemplos de políticas baseadas em identidade .....                                  | 191 |
| Exemplos de política baseada em recursos .....                                      | 196 |
| Solução de problemas .....  | 197 |
| Monitoramento .....   | 199 |
| Monitoramento do Amazon Lex com o CloudWatch .....                                  | 199 |
| Registro em log de chamadas da API do Amazon Lex com o AWS CloudTrail .....         | 204 |
| Validação de conformidade .....   | 207 |

|   |     |
|---|-----|
| Resiliência .....                             | 208 |
| Segurança da infraestrutura .....             | 208 |
| Diretrizes e limites .....                    | 209 |
| Diretrizes gerais .....                       | 209 |
| Limites .....                                 | 211 |
| Limites gerais .....                          | 211 |
| Limites de serviço de tempo de execução ..... | 211 |
| Limites de criação de modelos .....           | 212 |
| API Reference .....                           | 216 |
| Actions .....                                 | 216 |
| Amazon Lex Model Building Service .....       | 217 |
| Amazon Lex Runtime Service .....              | 340 |
| Data Types .....                              | 365 |
| Amazon Lex Model Building Service .....       | 365 |
| Amazon Lex Runtime Service .....              | 392 |
| Histórico de documentos .....                 | 401 |
| AWS Glossary .....                            | 403 |

# O que é o Amazon Lex?

O Amazon Lex é um serviço da AWS para a criação de interfaces de conversa para qualquer aplicativo que usa voz e texto. Com o Amazon Lex, o mesmo mecanismo de conversa da plataforma do Amazon Alexa agora está disponível para todos os desenvolvedores, permitindo que você crie chatbots sofisticados em linguagem natural em aplicativos novos e existentes. O Amazon Lex fornece a funcionalidade e a flexibilidade avançadas de compreensão de linguagem natural (NLU) e o reconhecimento automático de fala (ASR) para permitir a criação de experiências de usuário altamente envolventes com interações realistas por conversa e a criação de novas categorias de produtos.

O Amazon Lex permite que qualquer desenvolvedor crie chatbots de conversa rapidamente. Com o Amazon Lex, nenhuma experiência em deep learning é necessária — para criar um bot, você só precisa especificar o fluxo de conversa básico no console do Amazon Lex. O Amazon Lex gerencia o diálogo e ajusta dinamicamente as respostas na conversa. Usando o console, você pode criar, testar e publicar o chatbot de texto ou voz. Em seguida, você pode adicionar as interfaces de conversa aos bots em dispositivos móveis, aplicativos Web e plataformas de bate-papo (por exemplo, Facebook Messenger).

O Amazon Lex fornece integração predefinida com o AWS Lambda e você pode integrá-lo facilmente com muitos outros serviços na plataforma da AWS, incluindo o Amazon Cognito, o AWS Mobile Hub, o Amazon CloudWatch e o Amazon DynamoDB. A integração com o Lambda fornece aos bots acesso a conectores empresariais sem servidor, pré-criados para vinculação a dados em aplicativos SaaS, como o Salesforce, o HubSpot ou o Marketo.

Alguns dos benefícios do uso do Amazon Lex incluem:

- Simplicidade – o Amazon Lex orienta você durante o uso do console para criar seu próprio chatbot em minutos. Você fornece apenas algumas frases de exemplo e o Amazon Lex cria um modelo completo de linguagem natural por meio do qual o bot pode interagir usando voz e texto para fazer perguntas, obter respostas e concluir tarefas sofisticadas.
- Tecnologias de deep learning democratizadas – com a mesma tecnologia da Alexa, o Amazon Lex fornece as tecnologias ASR e NLU para criar um sistema de compreensão de linguagem falada (SLU). Por meio do SLU, o Amazon Lex analisa a entrada de linguagem natural falada e de texto, compreende a intenção por trás da entrada e atende à intenção do usuário chamando a função apropriada do negócio.

O reconhecimento da fala e a compreensão da linguagem natural são alguns dos problemas mais difíceis de serem resolvidos em ciência da computação e exigem que algoritmos sofisticados de deep learning sejam treinados em enormes quantidades de dados e infraestrutura. O Amazon Lex coloca as tecnologias de deep learning ao alcance de todos os desenvolvedores, com a mesma tecnologia do Alexa. Os chatbots do Amazon Lex convertem fala de entrada em texto e compreendem a intenção do usuário para gerar uma resposta inteligente, assim, você pode se concentrar na criação de seus bots com valor agregado diferenciado para seus clientes, para definir categorias de produtos totalmente novas possibilitadas pelas interfaces de conversa.

- Implantação e escalabilidade contínuas – Com o Amazon Lex, você pode criar, testar e implantar seus chatbots diretamente no console do Amazon Lex. O Amazon Lex permite que você publique facilmente seus chatbots de voz ou texto para uso em dispositivos móveis, aplicativos Web e serviços de bate-papo (por exemplo, Facebook Messenger). O Amazon Lex escala automaticamente, portanto, você não precisa se preocupar com o provisionamento de hardware e o gerenciamento da infraestrutura para potencializar sua experiência de bot.

- Integração incorporada com a plataforma da AWS – o Amazon Lex tem interoperabilidade nativa com outros serviços da AWS, como o Amazon Cognito, o AWS Lambda, o Amazon CloudWatch e o AWS Mobile Hub. Você pode aproveitar o poder da plataforma da AWS para segurança, monitoramento, autenticação do usuário, lógica de negócios, armazenamento e desenvolvimento de aplicativos móveis.
- Economia – com o Amazon Lex, não há custos iniciais nem taxas mínimas. Você será cobrado apenas pelas solicitações de texto ou fala feitas. A definição de preço conforme o uso e o baixo custo por solicitação fazem do serviço uma maneira econômica de criar interfaces de conversa. Com o nível gratuito do Amazon Lex, você pode testar o Amazon Lex com facilidade e sem nenhum investimento inicial.

## Você é um usuário iniciante do Amazon Lex?

Se estiver usando o Amazon Lex pela primeira vez, recomendamos ler as seções a seguir nesta ordem:

1. [Conceitos básicos do Amazon Lex \(p. 35\)](#) – nesta seção, você define sua conta e testa o Amazon Lex.
2. [API Reference \(p. 216\)](#) – esta seção fornece exemplos adicionais que podem ser usados para explorar o Amazon Lex.

# Amazon Lex: Como ele funciona

O Amazon Lex permite criar aplicativos usando uma interface de texto ou de fala com a mesma tecnologia usada pelo Amazon Alexa. Veja a seguir as etapas comuns que você executa ao trabalhar com Amazon Lex:

1. Crie e configure um bot com uma ou mais intenções a que você deseja oferecer suporte. Configure o bot para que ele entenda o objetivo do usuário (intenção), inicie uma conversa com o usuário para obter informações e cumpra a intenção do usuário.
2. Teste o bot. Você pode usar o cliente da janela de teste fornecido pelo console do Amazon Lex.
3. Publique uma versão e crie um alias.
4. Implante o bot. Você pode implantar o bot em plataformas como aplicações móveis ou plataformas de mensagens, como Facebook Messenger.

Antes de começar a usar, familiarize-se com os seguintes conceitos principais e a terminologia do Amazon Lex:

- Bot – um bot executa tarefas automatizadas, como encomendar uma pizza, reservar um hotel, encomendar flores e assim por diante. Um bot do Amazon Lex é desenvolvido pelos recursos de Automatic Speech Recognition (ASR) e do Natural Language Understanding (NLU), a mesma tecnologia usada pelo Amazon Alexa.

Os bots do Amazon Lex podem compreender a entrada do usuário fornecida por texto ou fala e conversar em linguagem natural. É possível criar funções do Lambda e adicioná-las como ganchos de código em sua configuração de intenção para executar a validação de dados do usuário e tarefas de atendimento.

- Intenção – uma intenção representa uma ação que o usuário deseja executar. Crie um bot para oferecer suporte a uma ou mais intenções relacionadas. Por exemplo, você pode criar um bot que peça pizza e bebidas. Para cada intenção, forneça as seguintes informações obrigatórias:
  - Nome da intenção – um nome descritivo para a intenção. Por exemplo, **OrderPizza**.
  - Enunciados de exemplo – como um usuário pode transmitir a intenção. Por exemplo, um usuário pode dizer "Posso pedir uma pizza" ou "Quero pedir uma pizza".
  - Como atender à intenção – como você deseja atender à intenção depois que o usuário fornecer todas as informações necessárias (por exemplo, fazer um pedido em uma pizzeria local). Recomendamos criar uma função do Lambda para atender à intenção.

Opcionalmente, você pode configurar a intenção para que o Amazon Lex simplesmente retorne as informações de volta ao aplicativo cliente para executar o atendimento necessário.

Além de intenções personalizadas, como encomendar uma pizza, o Amazon Lex também fornece intenções integradas para configurar seu bot rapidamente. Para obter mais informações, consulte [Intenções integradas e tipos de slot \(p. 28\)](#).



- Slot – uma intenção pode exigir zero ou mais slots ou parâmetros. Você adiciona slots como parte da configuração de intenção. Em tempo de execução, o Amazon Lex solicita ao usuário valores específicos do slot. O usuário deve fornecer valores para todos os slots necessários para que o Amazon Lex possa atender à intenção.

Por exemplo, a intenção `OrderPizza` exige slots como tamanho da pizza, tipo de massa e número de pizzas. Na configuração de intenção, você adiciona esses slots. Para cada slot, você fornece o tipo de slot e uma solicitação para o Amazon Lex enviar ao cliente para obter os dados do usuário. Um usuário pode responder com um valor de slot que inclui palavras adicionais, como "large pizza please (pizza grande, por favor)" ou "let's stick with small (vamos querer a pequena)", e o Amazon Lex ainda compreenderá o valor do slot pretendido.

- Tipo de slot – cada slot tem um tipo. Você pode criar tipos de slot personalizados ou usar tipos de slot integrados. Por exemplo, você pode criar e usar os seguintes tipos de slot para a intenção `OrderPizza`:
  - Tamanho – com valores de enumeração `Small`, `Medium` e `Large`.
  - Massa – com valores de enumeração `Thick` e `Thin`.

O Amazon Lex também fornece tipos de slot integrado. Por exemplo, `AMAZON.NUMBER` é um tipo de slot integrado que você pode usar para o número de pizzas pedidas. Para obter mais informações, consulte [Intenções integradas e tipos de slot \(p. 28\)](#).

Os tópicos a seguir fornecem informações adicionais. Recomendamos que você analise-as em ordem e, em seguida, explore os exercícios [Conceitos básicos do Amazon Lex \(p. 35\)](#).

#### Tópicos

- [Modelo de programação \(p. 4\)](#)
- [Permissões de serviço \(p. 9\)](#)
- [Gerenciamento de mensagens \(p. 10\)](#)
- [Gerenciar contexto da conversa \(p. 20\)](#)
- [Gerenciamento de sessões com a API do Amazon Lex \(p. 25\)](#)
- [Opções de implantação de bot \(p. 28\)](#)
- [Intenções integradas e tipos de slot \(p. 28\)](#)
- [Tipos de slot personalizados \(p. 33\)](#)

## Modelo de programação

Um bot é o principal tipo de recurso no Amazon Lex. Os outros tipos de recursos no Amazon Lex são intenção, tipo de slot, alias e associação de canal de bot.

Você cria um bot usando o console do Amazon Lex ou a API de criação de modelos. O console fornece uma interface gráfica de usuário que você usa para criar um bot pronto para produção para seu aplicativo. Se preferir, você poderá usar a API de criação de modelos por meio da AWS CLI ou de seu próprio programa personalizado para criar um bot.

Depois de criar um bot, você o implanta em uma das [plataformas compatíveis](#) ou o integra em seu próprio aplicativo. Quando um usuário interage com o bot, o aplicativo cliente envia solicitações ao bot usando a

API de tempo de execução do Amazon Lex. Por exemplo, quando um usuário diz "I want to order pizza (Quero encomendar uma pizza)", o cliente envia essa entrada do usuário ao Amazon Lex usando uma das operações da API de tempo de execução. Os usuários podem fornecer entrada de voz ou texto.

Você também pode criar funções do Lambda e usá-las em uma intenção. Use esses ganchos de código de função do Lambda para executar atividades em tempo de execução, como inicialização, validação da entrada do usuário e atendimento da intenção. As seções a seguir fornecem informações adicionais.

#### Tópicos

- [Operações de API de criação de modelo \(p. 5\)](#)
- [Operações de API de runtime \(p. 6\)](#)
- [Funções do Lambda como ganchos de código \(p. 6\)](#)

## Operações de API de criação de modelo

Para criar programaticamente bots, intenções e tipos de slot, use a API de operações de criação de modelo. Você também pode usar a API de criação de modelo para gerenciar, atualizar e excluir os recursos para o seu bot. As operações de API de criação de modelo incluem:

- [PutBot \(p. 311\)](#), [PutBotAlias \(p. 319\)](#), [PutIntent \(p. 323\)](#) e [PutSlotType \(p. 333\)](#) para criar e atualizar bots, aliases de bot, intenções e tipos de slot, respectivamente.
- [CreateBotVersion \(p. 219\)](#), [CreateIntentVersion \(p. 224\)](#) e [CreateSlotTypeVersion \(p. 230\)](#) para criar e publicar versões de seus bots, intenções e tipos de slot, respectivamente.
- [GetBot \(p. 252\)](#) e [GetBots \(p. 270\)](#) para obter um bot específico ou uma lista de bots que você criou, respectivamente.
- [GetIntent \(p. 288\)](#) e [GetIntents \(p. 293\)](#) para obter uma intenção específica ou uma lista de intenções que você criou, respectivamente.
- [GetSlotType \(p. 299\)](#) e [GetSlotTypes \(p. 302\)](#) para obter um tipo de slot específico ou uma lista de tipos de slot que você criou, respectivamente.
- [GetBuiltinIntent \(p. 276\)](#), [GetBuiltinIntents \(p. 278\)](#) e [GetBuiltinSlotTypes \(p. 280\)](#) para obter uma intenção integrada do Amazon Lex, uma lista de intenções integradas do Amazon Lex ou uma lista de tipos de slot integrado que você pode usar em seu bot, respectivamente.
- [GetBotChannelAssociation \(p. 263\)](#) e [GetBotChannelAssociations \(p. 267\)](#) para obter uma associação entre seu bot e uma plataforma de mensagens ou uma lista de associações entre seu bot e plataformas de mensagens, respectivamente.
- [DeleteBot \(p. 234\)](#), [DeleteBotAlias \(p. 236\)](#), [DeleteBotChannelAssociation \(p. 238\)](#), [DeleteIntent \(p. 242\)](#) e [DeleteSlotType \(p. 246\)](#) para remover recursos desnecessários em sua conta.

Você pode usar a API de criação de modelo para criar ferramentas personalizadas a fim de gerenciar os recursos do Amazon Lex. Por exemplo, há um limite de 100 versões para bots, intenções e tipos de slot. Você pode usar a API de criação de modelo para criar uma ferramenta que exclui automaticamente versões antigas quando o bot se aproxima do limite.

Para garantir que apenas uma operação atualize um recurso por vez, o Amazon Lex usa somas de verificação. Quando usa uma operação de API `Put` — [PutBot \(p. 311\)](#), [PutBotAlias \(p. 319\)](#), [PutIntent \(p. 323\)](#) ou [PutSlotType \(p. 333\)](#) — para atualizar um recurso, você deve passar a soma de verificação atual do recurso na solicitação. Se duas ferramentas tentarem atualizar um recurso ao mesmo tempo, elas oferecem a mesma soma de verificação atual. A primeira solicitação a alcançar o Amazon Lex corresponde à soma de verificação atual do recurso. Quando a segunda solicitação chega, a soma de verificação é diferente. A segunda ferramenta recebe uma exceção `PreconditionFailedException` e a atualização é encerrada.

As operações `Get` — [GetBot \(p. 252\)](#), [GetIntent \(p. 288\)](#) e [GetSlotType \(p. 299\)](#) — são eventualmente consistentes. Se você usar uma operação `Get` imediatamente após criar ou modificar um

recurso com uma das operações `Put`, as alterações poderão não ser retornadas. Depois de uma operação `Get` retornar a atualização mais recente, ela sempre retornará esse recurso atualizado até que o recurso seja modificado novamente. Você pode determinar se um recurso atualizado foi retornado ao analisar a soma de verificação.

## Operações de API de runtime

Os aplicativos cliente usam as seguintes operações de API em tempo de execução para se comunicar com o Amazon Lex.

- [PostContent \(p. 347\)](#) – usa entrada de texto ou de fala e retorna as informações da intenção e uma mensagem de texto ou de fala para transmitir ao usuário. No momento, o Amazon Lex oferece suporte aos seguintes formatos de áudio:

Formatos de áudio de entrada – LPCM e Opus

Formatos de áudio de saída – MPEG, OGG e PCM

A operação `PostContent` oferece suporte a entrada de áudio em 8 kHz e 16 kHz. Os aplicativos em que o usuário final se comunica com o Amazon Lex por telefone, como uma central de atendimento automatizada, podem passar áudio 8 kHz diretamente.

- [PostText \(p. 355\)](#) – usa texto como entrada e retorna informações da intenção e uma mensagem de texto para transmitir ao usuário.

Seu aplicativo cliente usa a API de tempo de execução para chamar um determinado bot do Amazon Lex para processar enunciados — entrada de texto ou de fala do usuário. Por exemplo, suponha que um usuário diga "Quero pizza". O cliente envia essa entrada do usuário a um bot usando uma das operações da API de tempo de execução do Amazon Lex. A partir da entrada do usuário, o Amazon Lex reconhece que a solicitação do usuário destina-se à intenção `OrderPizza` definida no bot. O Amazon Lex engaja o usuário em uma conversa para obter as informações necessárias ou dados do slot, como o tamanho da pizza, a cobertura e o número de pizzas. Depois que o usuário fornece todos os dados necessários do slot, o Amazon Lex chama o gancho de código da função do Lambda para atender à intenção ou retorna os dados da intenção para o cliente, dependendo de como a intenção está configurada.

Use a operação [PostContent \(p. 347\)](#) quando seu bot usa entrada de voz. Por exemplo, um aplicativo de call center automatizado pode enviar voz para um bot do Amazon Lex, e não para um agente, para resolver indagações do cliente. Você pode usar o formato de áudio de 8 kHz para enviar áudio diretamente do telefone para o Amazon Lex.

A janela de teste no console do Amazon Lex usa a API [PostContent \(p. 347\)](#) para enviar solicitações de texto e de fala para o Amazon Lex. Use esta janela de teste nos exercícios [Conceitos básicos do Amazon Lex \(p. 35\)](#).

## Funções do Lambda como ganchos de código

Você pode configurar seu bot do Amazon Lex para chamar uma função do Lambda como um gancho de código. O hook de código pode servir para várias finalidades:

- Personaliza a interação do usuário — por exemplo, quando Joe pergunta pelas coberturas de pizza disponíveis, você pode usar o conhecimento anterior das opções de Joe para exibir um subconjunto de coberturas.

- Valida a entrada do usuário — suponha que Jen queira escolher flores depois do horário de funcionamento. Você pode validar o horário que Jen inseriu e enviar uma resposta adequada.
- Atende à intenção do usuário — depois de Joe fornecer todas as informações sobre seu pedido de pizza, o Amazon Lex pode chamar uma função do Lambda para fazer o pedido em uma pizzeria local.

Quando configura uma intenção, você especifica as funções do Lambda como ganchos de código nos seguintes locais:

- Gancho de código do diálogo para inicialização e validação — essa função do Lambda é chamada em cada entrada do usuário, supondo que o Amazon Lex entendeu a intenção do usuário.
- Gancho de código de atendimento — essa função do Lambda é chamada depois que o usuário fornece todos os dados do slot necessários para atender à intenção.

Você escolhe a intenção e define os ganchos de código no console do Amazon Lex, conforme mostrado na captura de tela a seguir:

## OrderFlowers Latest ▾

### ▼ Sample utterances ⓘ

*e.g. I would like to book a flight.* +

I would like to pick up flowers ✕

I would like to order some flowers ✕

Order flowers ✕

### ▼ Lambda initialization and validation ⓘ

☒ Initialization and validation code hook

Lambda Function Name ▾

### ▼ Slots ⓘ

| Priority | Required                            | Name                 | Slot type          |            | Prompt                 |
|----------|-------------------------------------|----------------------|--------------------|------------|------------------------|
|          |                                     | <i>e.g. Location</i> | <i>e.g. A...</i> ▾ |            | <i>e.g. What city?</i> |
| 1.       | <input checked="" type="checkbox"/> | FlowerType           | Flowe... ▾         | 1 ▾        | What type of flo       |
| 2.       | <input checked="" type="checkbox"/> | PickupDate           | AMA... ▾           | Built-in ▾ | What day do yo         |
| 3.       | <input checked="" type="checkbox"/> | PickupTime           | AMA... ▾           | Built-in ▾ | At what time do        |

### ▼ Confirmation prompt ⓘ

☒ Confirmation prompt

Confirm

Okay, your {FlowerType} will be ready for pickup by {Picku} ⚙

Cancel (if the user says "no")

Okay, I will not place your order. ⚙

### ▼ Fulfillment ⓘ

☒ AWS Lambda function ☐ Return parameters to client

Você também pode definir hooks de código usando os campos `dialogCodeHook` e `fulfillmentActivity` na operação [PutIntent](#) (p. 323).

Uma função do Lambda pode executar a inicialização, a validação e o atendimento. Os dados do evento que a função do Lambda recebe têm um campo que identifica o chamador como um gancho de código de diálogo ou de atendimento. É possível usar essas informações para executar a parte adequada de seu código.

Você pode usar uma função do Lambda para criar um bot que pode navegar por diálogos complexos. Você usa o campo `dialogAction` na resposta da função do Lambda para direcionar o Amazon Lex para executar ações específicas. Por exemplo, você pode usar a ação do diálogo `ElicitSlot` para dizer ao Amazon Lex para solicitar ao usuário um valor de slot que não é necessário. Você pode usar a ação de diálogo `ElicitIntent` para estimular uma nova intenção quando o usuário conclui a anterior.

Para obter mais informações, consulte [Uso de funções do Lambda](#) (p. 106).

## Permissões de serviço

O Amazon Lex usa AWS Identity and Access Management (IAM) [funções vinculadas ao serviço](#). O Amazon Lex assume essas funções para chamar serviços da AWS em nome de seus bots e canais de bot. As funções existem dentro de sua conta, mas estão vinculadas aos casos de uso do Amazon Lex e têm permissões predefinidas. Somente o Amazon Lex pode assumir essas funções, e você não pode modificar suas permissões. Você pode excluí-las depois de excluir seus recursos relacionados usando o IAM. Isso protege seus recursos do Amazon Lex, pois você não pode remover permissões necessárias inadvertidamente.

O Amazon Lex usa duas funções vinculadas ao serviço do IAM.

- `AWSServiceRoleForLexBots` — o Amazon Lex usa essa função vinculada ao serviço para chamar o Amazon Polly para sintetizar respostas de fala para o seu bot.
- `AWSServiceRoleForLexChannels` — o Amazon Lex usa essa função vinculada ao serviço para postar texto em seu bot ao gerenciar canais.

Você não precisa criar manualmente nenhuma dessas funções. Quando você cria seu primeiro bot usando o console, o Amazon Lex cria a função `AWSServiceRoleForLexBots` para você. Quando você associa pela primeira vez um bot a um canal de sistema de mensagens, o Amazon Lex cria a função `AWSServiceRoleForLexChannels` para você.

## Criação de políticas com base em recursos para o AWS Lambda

Ao invocar funções do Lambda, o Amazon Lex usa políticas com base em recursos. Uma política baseada em recursos está anexada a um recurso; ela permite especificar quem tem acesso ao recurso e quais ações podem ser executadas nele. Isso permite que você defina permissões escopo mais específicas entre as funções do Lambda e as intenções que você criou. Também permite que você veja essas permissões em uma única política quando gerenciar funções do Lambda que têm muitas origens de eventos.

Para obter mais informações, consulte [Uso de políticas com base em recursos para o AWS Lambda \(Políticas de função do Lambda\)](#) no Guia do desenvolvedor do AWS Lambda.

Para criar políticas com base em recursos para intenções que você associa a uma função do Lambda, você pode usar o console do Amazon Lex. Ou você pode usar a interface de linha de comando da AWS (AWS CLI). Na CLI da AWS, use a API [AddPermission](#) do Lambda com o campo `Principal` definido

como `lex.amazonaws.com` e o `SourceArn` definido como o ARN da intenção que tem permissão para chamar a função.

## Excluindo funções vinculadas a serviço

Você pode usar o console do IAM, a CLI do IAM ou a API do IAM para excluir as funções vinculadas ao serviço `AWSServiceRoleForLexBots` e `AWSServiceRoleForLexChannels`. Para obter mais informações, consulte [Exclusão de uma função vinculada ao serviço](#) no Guia do usuário do IAM.

## Gerenciamento de mensagens

### Tópicos

- [Tipos de mensagens \(p. 10\)](#)
- [Contextos para a configuração de mensagens \(p. 11\)](#)
- [Formatos de mensagem suportados \(p. 15\)](#)
- [Grupos de mensagens \(p. 15\)](#)
- [Cartões de resposta \(p. 16\)](#)

Ao criar um bot, você pode configurar mensagens de esclarecimento ou informativas que deseja que ele envie ao cliente. Considere os seguintes exemplos:

- Você pode configurar seu bot com a solicitação de esclarecimento a seguir:

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

O Amazon Lex envia esta mensagem ao cliente, caso não entenda a intenção do usuário.

- Suponha que você crie um bot para oferecer suporte a uma intenção chamada `OrderPizza`. Para pedidos de pizza, você deseja que os usuários forneçam informações como tamanho da pizza, cobertura e tipo de massa. Você pode configurar as seguintes solicitações:

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

Assim que o Amazon Lex determina a intenção do usuário de encomendar uma pizza, ele envia estas mensagens ao cliente para obter informações do usuário.

Esta seção explica como projetar interações do usuário em sua configuração de bot.

## Tipos de mensagens

A mensagem pode ser uma solicitação ou instrução.

- Normalmente, a solicitação é uma pergunta que presume uma resposta do usuário.
- A instrução é apenas informativa. Ela não presume uma resposta.

A mensagem pode incluir referências a um slot e a atributos de sessão. Em tempo de execução, Amazon Lex substitui essas referências por valores reais.

Para se referir a valores de slots que foram definidos, use a seguinte sintaxe:

```
{SlotName}
```

Para se referir a atributos de sessão, use a seguinte sintaxe:

```
[AttributeName]
```

As mensagens podem incluir valores de slot e atributos de sessão.

Por exemplo, suponha que você configure a seguinte mensagem na intenção OrderPizza de seu bot:

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

Essa mensagem refere-se tanto ao slot (PizzaTopping) quanto aos atributos de sessão (FirstName e DeliveryTime). Em tempo de execução, o Amazon Lex substitui estes espaços reservados por valores e retorna a seguinte mensagem para o cliente:

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

Para incluir colchetes ([]) ou chaves ({}), em uma mensagem, use o caractere de escape de barra invertida (\). Por exemplo, a seguinte mensagem inclui chaves e colchetes:

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

O texto retornado ao aplicativo cliente é semelhante a este:

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

Para obter mais informações sobre atributos de sessão, consulte as operações de API [PostText \(p. 355\)](#) e [PostContent \(p. 347\)](#). Para ver um exemplo, consulte [Exemplo de bot: BookTrip \(p. 154\)](#).

As funções do Lambda também podem gerar mensagens e retorná-las ao Amazon Lex para envio ao usuário. Se você adicionar funções do Lambda ao configurar sua intenção, poderá criar mensagens dinamicamente. Se você fornecer as mensagens ao configurar o bot, poderá eliminar a necessidade de criar uma solicitação na função do Lambda.

## Contextos para a configuração de mensagens

Ao criar o bot, você pode criar mensagens em contextos diferentes, como solicitações de esclarecimento no bot, solicitações de valores de slot e mensagens de intenções. O Amazon Lex escolhe uma mensagem apropriada em cada contexto para retornar ao usuário. Você pode fornecer um grupo de mensagens para cada contexto. Se você fizer isso, o Amazon Lex selecionará aleatoriamente uma mensagem no grupo. Você pode também especificar o formato da mensagem ou agrupar as mensagens. Para obter mais informações, consulte [Formatos de mensagem suportados \(p. 15\)](#).

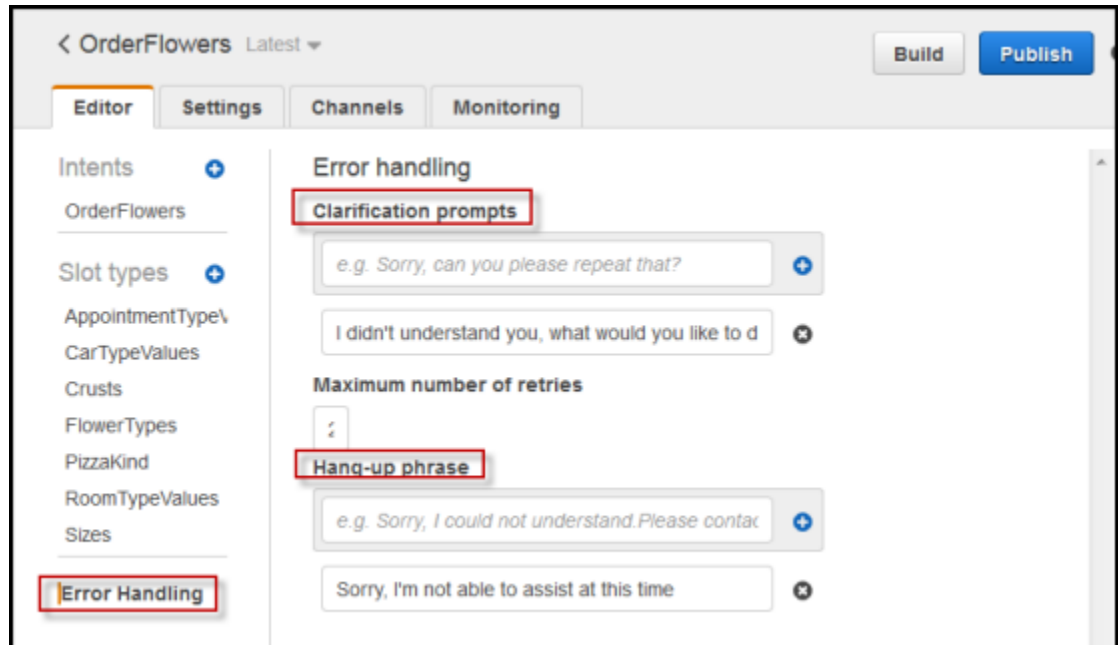
Se você tiver uma função do Lambda associada a uma intenção, poderá substituir qualquer uma das mensagens configuradas no momento da compilação. No entanto, para usar qualquer uma dessas mensagens, não é preciso ter uma função do Lambda.

## Mensagens de bot

Você pode configurar seu bot com solicitações de esclarecimento e mensagens de encerramento. Em tempo de execução, o Amazon Lex usa a solicitação de esclarecimento quando não compreende



a intenção do usuário. Você pode configurar o número de vezes que o Amazon Lex deve solicitar esclarecimento antes de desligar com a mensagem de encerramento. Você configura as mensagens em nível de bot na seção Error Handling (Tratamento de erros) do console do Amazon Lex, da seguinte forma:



Com a API, você configura mensagens definindo os campos `clarificationPrompt` e `abortStatement` na operação `PutBot` (p. 311).

Se usar uma função do Lambda com uma intenção, a função do Lambda pode retornar uma resposta direcionando o Amazon Lex a solicitar uma intenção do usuário. Se a função do Lambda não fornecer essa mensagem, o Amazon Lex usará a solicitação de esclarecimento.

## Solicitações de slot

Você deve especificar pelo menos uma mensagem de esclarecimento para os slots necessários em uma intenção. Em tempo de execução, o Amazon Lex usa uma dessas mensagens para solicitar que o usuário forneça um valor para o slot. Por exemplo, para um slot `cityName`, o seguinte é um prompt válido:

Which city would you like to fly to?

Você pode definir uma ou mais solicitações para cada slot usando o console. Você pode também criar grupos de solicitações usando a operação `PutIntent` (p. 323). Para obter mais informações, consulte [Grupos de mensagens](#) (p. 15).

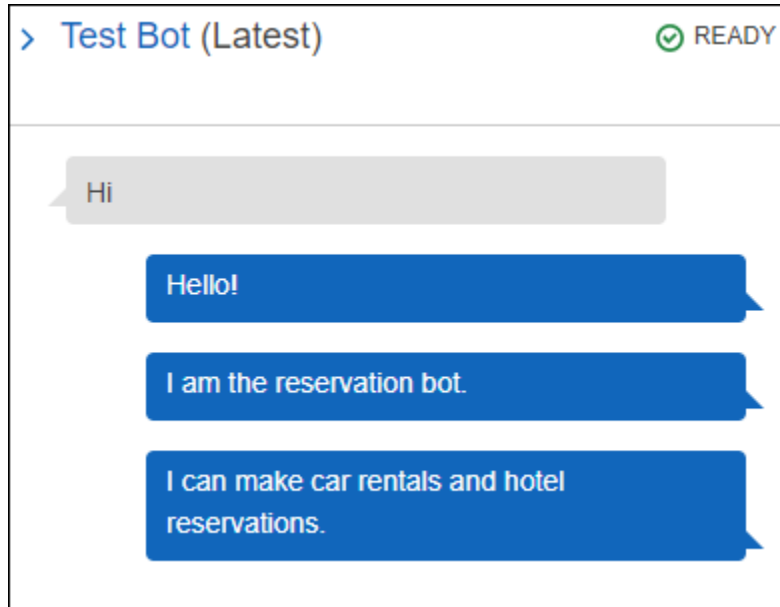
## Respostas

No console, use a seção Responses (Respostas) para criar conversas dinâmicas e envolventes para seu bot. Você pode criar um ou mais grupos de mensagens para uma resposta. Em tempo de execução, o Amazon Lex cria uma resposta selecionando uma mensagem de cada grupo de mensagens. Para obter mais informações sobre grupos de mensagens, consulte [Grupos de mensagens](#) (p. 15).

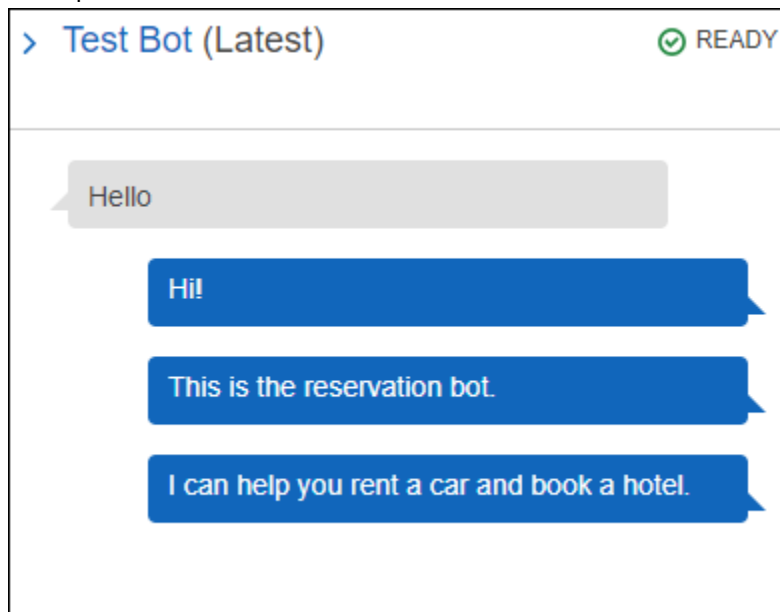
Por exemplo, o primeiro grupo de mensagens pode conter diferentes saudações: "Olá", "Oi" e "Saudações". O segundo grupo de mensagens pode conter diferentes formas de introdução: "Sou o bot de reserva" e "Este é o bot de reserva". Um terceiro grupo de mensagens pode informar os recursos do bot:

"Posso ajudar com aluguel de carro e reservas de hotéis", "Você pode alugar carros e fazer reservas de hotel" e "Eu posso ajudar você a alugar um carro e reservar um hotel".

O Lex usa uma mensagem de cada um dos grupos de mensagens para criar as respostas dinamicamente em uma conversa. Por exemplo, uma interação pode ser:

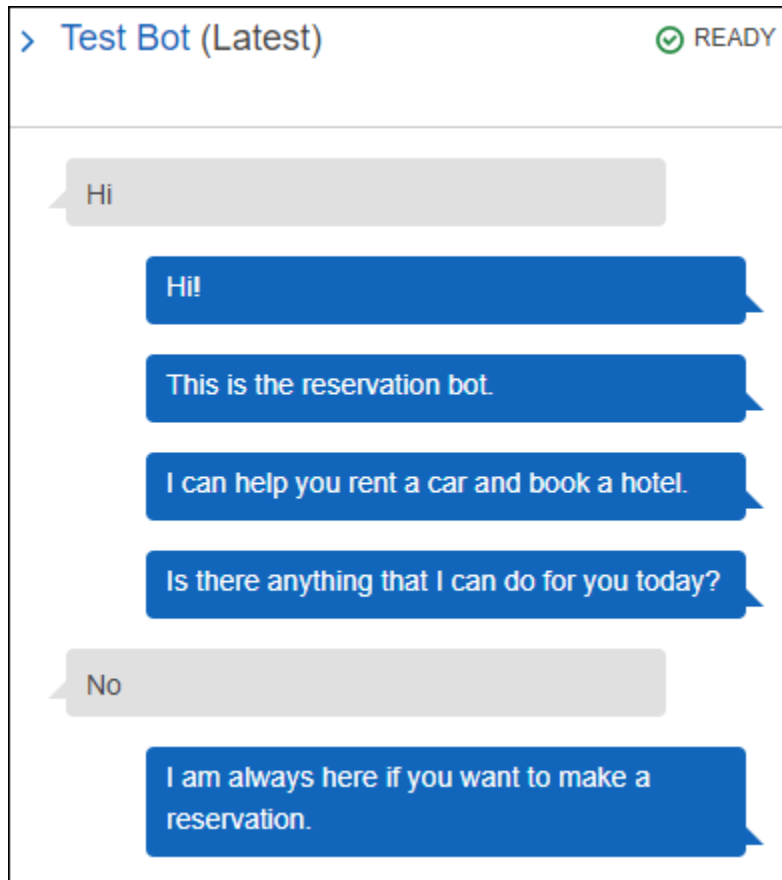


Outra poderia ser:

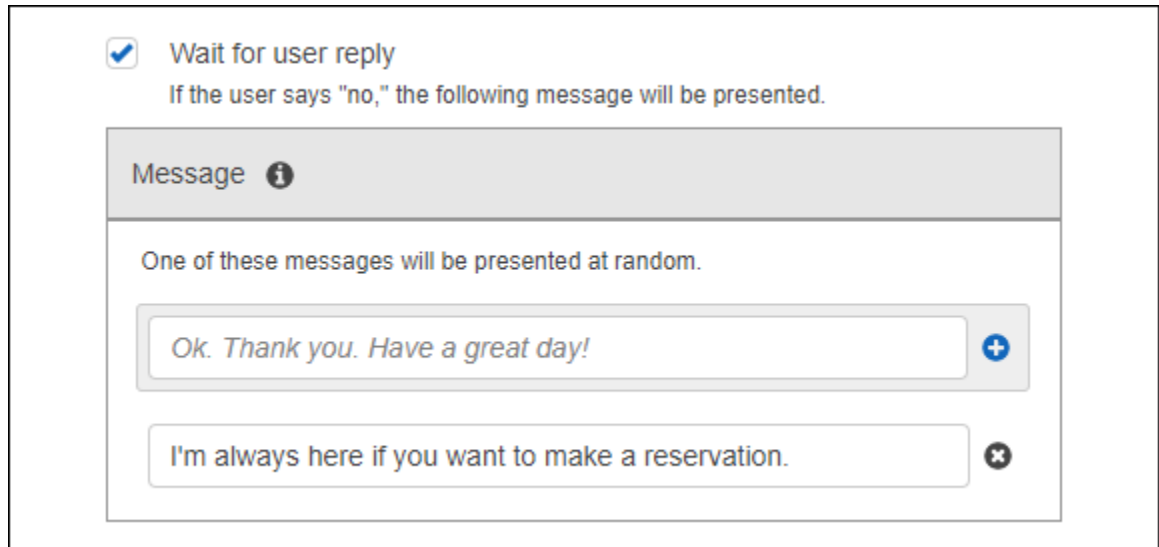


Em qualquer um dos casos, o usuário pode responder com uma nova intenção, como a intenção `BookCar` ou `BookHotel`.

Você pode configurar o bot para fazer uma pergunta complementar na resposta. Por exemplo, para a interação anterior, você pode criar um quarto grupo de mensagens com as seguintes perguntas: "Posso ajudar com um carro ou um hotel?", "Você gostaria de fazer uma reserva agora?" e "Há algo que possa fazer para você?". Para mensagens que incluam "Não" como resposta, você pode criar uma solicitação de acompanhamento. Por exemplo:



Para criar uma solicitação de acompanhamento, escolha Wait for user reply (Aguardar resposta do usuário). Em seguida, digite a mensagem ou as mensagens que você deseja enviar quando o usuário disser "Não". Ao criar uma resposta para usar como solicitação de acompanhamento, você deve especificar também uma instrução apropriada quando a resposta para a instrução for "Não". Por exemplo:



Para adicionar respostas a uma intenção com a API, use a operação `PutIntent`. Para especificar uma resposta, defina o campo `conclusionStatement` na solicitação `PutIntent`. Para definir uma

solicitação de acompanhamento, defina o campo `followUpPrompt` e inclua a instrução a ser enviada quando o usuário disser "Não". Você não pode definir o campo `conclusionStatement` e o campo `followUpPrompt` na mesma intenção.

## Formatos de mensagem suportados

Quando você usa a operação [PostText \(p. 355\)](#) ou a operação [PostContent \(p. 347\)](#) com o cabeçalho `Accept` definido como `text/plain; charset=utf8`, o Amazon Lex oferece suporte a mensagens nos seguintes formatos:

- `PlainText` — a mensagem contém texto sem formatação UTF-8.
- `SSML` — a mensagem contém texto formatado para saída de voz.
- `CustomPayload` — a mensagem contém um formato personalizado que você criou para seu cliente. Você pode definir a carga útil para atender às necessidades de seu aplicativo.
- `Composite` — a mensagem é uma coleção de mensagens, uma de cada grupo de mensagens. Para obter mais informações sobre grupos de mensagens, consulte [Grupos de mensagens \(p. 15\)](#).

Por padrão, o Amazon Lex retorna qualquer uma das mensagens definidas para determinada solicitação. Por exemplo, se você definir cinco mensagens para obter um valor de slot, o Amazon Lex escolherá uma das mensagens aleatoriamente e a retornará para o cliente.

Se quiser que o Amazon Lex retorne um tipo específico de mensagem para o cliente em uma solicitação de tempo de execução, defina o parâmetro de solicitação `x-amzn-lex:accept-content-types`. A resposta é limitada pelo tipo ou pelos tipos solicitados. Se houver mais de uma mensagem do tipo especificado, o Amazon Lex retornará uma aleatoriamente. Para obter mais informações sobre o cabeçalho `x-amz-lex:accept-content-types`, consulte [Como configurar o tipo de resposta \(p. 22\)](#).

## Grupos de mensagens

Um grupo de mensagens é um conjunto de respostas adequadas para determinada solicitação. Use grupos de mensagens quando desejar que seu bot crie respostas dinamicamente em uma conversa. Quando o Amazon Lex retorna uma resposta ao aplicativo cliente, ele escolhe aleatoriamente uma mensagem de cada grupo. Você pode criar no máximo cinco grupos de mensagens para cada resposta. Cada grupo pode conter no máximo cinco mensagens. Para obter exemplos de como criar grupos de mensagens no console, consulte [Respostas \(p. 12\)](#).

Para criar um grupo de mensagens, você pode usar o console ou as operações [PutBot \(p. 311\)](#), [PutIntent \(p. 323\)](#) ou [PutSlotType \(p. 333\)](#) para atribuir um número de grupo a uma mensagem. Se você não criar um grupo de mensagens ou se criar apenas um grupo, o Amazon Lex enviará uma única mensagem no campo `Message`. Os aplicativos cliente recebem várias mensagens em uma resposta somente quando você cria mais de um grupo de mensagens no console ou mais de um grupo de mensagens ao criar ou atualizar uma intenção com a operação [PutIntent \(p. 323\)](#).

Quando o Amazon Lex envia uma mensagem de um grupo, o campo `Message` da resposta contém um objeto JSON de escape que contém as mensagens. O exemplo a seguir mostra o conteúdo do campo `Message` quando ele contém várias mensagens.

### Note

O exemplo é formatado por motivo de legibilidade. Uma resposta não contém retornos de carro (CR).

```
{ \"messages\": [
```

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

Você pode definir o formato das mensagens. O formato pode ser um dos seguintes:

- PlainText — o formato da mensagem é texto sem formatação UTF-8.
- SSML — o formato da mensagem é Speech Synthesis Markup Language (SSML).
- CustomPayload — o formato da mensagem é um formato personalizado especificado por você.

Para controlar o formato das mensagens que as operações `PostContent` e `PostText` retornam no campo `Message`, defina o atributo de solicitação `x-amz-lex:accept-content-types`. Por exemplo, se você definir o cabeçalho como a seguir, você receberá apenas texto sem formatação e mensagens SSML na resposta:

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

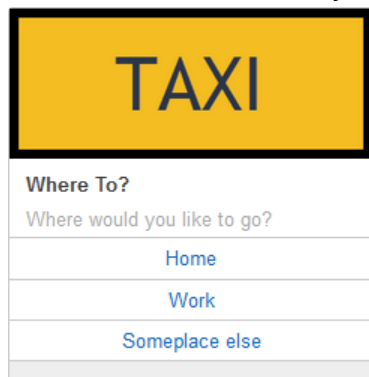
Se solicitar um formato específico de mensagem e um grupo de mensagens não contiver uma mensagem com esse formato, você obterá uma exceção `NoUsableMessageException`. Quando usar um grupo de mensagens para agrupar mensagens por tipo, não use o cabeçalho `x-amz-lex:accept-content-types`.

Para obter mais informações sobre o cabeçalho `x-amz-lex:accept-content-types`, consulte [Como configurar o tipo de resposta](#) (p. 22).

## Cartões de resposta

Um cartão de resposta contém um conjunto de respostas adequadas a uma solicitação. Use cartões de resposta para simplificar interações para seus usuários e aumentar a precisão de seu bot ao reduzir erros tipográficos nas interações de texto. Você pode enviar um cartão de resposta para cada solicitação que o Amazon Lex envia a seu aplicativo cliente. Você pode usar cartões de resposta com o Facebook Messenger, Slack e Twilio e com seus próprios aplicativos clientes.

Por exemplo, em um aplicativo de táxi, você pode configurar uma opção no cartão de resposta para "Casa" e defina o valor para o endereço residencial do usuário. Quando o usuário seleciona essa opção, o Amazon Lex recebe o endereço inteiro como o texto de entrada.



| Where To?                   |
|-----------------------------|
| Where would you like to go? |
| Home                        |
| Work                        |
| Someplace else              |

Você pode definir um cartão de resposta para os seguintes prompts:

- Declaração de conclusão
- Prompt de confirmação

- Prompt de acompanhamento
- Declaração de rejeição
- Enunciados de tipo de slot

Você pode definir apenas um cartão de resposta para cada prompt.

Você configura cartões de resposta ao criar uma intenção. Você pode definir um cartão de resposta estático no momento da criação usando o console ou a operação [PutIntent](#) (p. 323). Ou você pode definir um cartão de resposta dinâmica em tempo de execução em uma função do Lambda. Se você definir cartões de resposta estática e dinâmica, a o cartão de resposta dinâmica terá precedência.

O Amazon Lex envia cartões de resposta no formato que o cliente compreende. Ele transforma cartões de resposta para Facebook Messenger, Slack e Twilio. Para outros clientes, o Amazon Lex envia uma estrutura JSON na resposta [PostText](#) (p. 355). Por exemplo, se o cliente for o Facebook Messenger, o Amazon Lex transformará o cartão de resposta para um modelo genérico. Para obter mais informações sobre os modelos genéricos do Facebook Messenger, consulte [Modelo genérico](#) no site do Facebook. Para ver um exemplo de estrutura JSON, consulte [Gerando cartões de resposta dinamicamente](#) (p. 19).

Você pode usar cartões de resposta somente com a operação [PostText](#) (p. 355). Você não pode usar cartões de resposta com a operação [PostContent](#) (p. 347).

## Definição de cartões de resposta estática

Defina cartões de resposta estática com a operação [PutBot](#) (p. 311) ou o console do Amazon Lex ao criar uma intenção. Um cartão de resposta estática é definido ao mesmo tempo que a intenção. Use um cartão de resposta estática quando as respostas são fixas. Suponha que você está criando um bot com uma intenção que tem um slot para sabor. Ao definir o slot de sabor, você especifica prompts, como mostrado na seguinte captura de tela do console:

| Slots ⓘ  |          |           |             |                        |      |
|----------|----------|-----------|-------------|------------------------|------|
| Priority | Required | Name      | Slot type   | Prompt                 |      |
|          |          |           | e.g. AMA... | e.g. What city?        | ⚙️ + |
| 1.       | ✓        | teaSize   | teaSize     | What size iced tea wc  | ⚙️ ✖ |
| 2.       | ✓        | teaFlavor | teaFlavor   | Would you like a flavo | ⚙️ ✖ |

Ao definir solicitações, você tem a opção de associar um cartão de resposta e definir detalhes na operação [PutBot](#) (p. 311) ou no console do Amazon Lex, conforme mostrado no exemplo a seguir:

teaFlavor Prompts

Maximum number of retries

2

Corresponding utterances

e.g. I would like to go to {toCity}

+

Prompt response cards

0

Card image

Card title

What Flavor?

Card subtitle

What flavor tea would

Preview

Facebook

Button value

lemon

Button title

Lemon

raspberry

Raspberry

plain

Plain

None

e.g. Button title

None

e.g. Button title

Delete card

What Flavor?

What flavor tea would you like?

Lemon

Raspberry


Plain

Cancel

Save

Agora, suponha que você integrou o bot ao Facebook Messenger. O usuário pode clicar nos botões para escolher uma sabor, como mostrado na ilustração a seguir:

What flavor tea would you like?



What flavor?  
What flavor tea would you like?

Lemon

Raspberry

Plain

Para personalizar o conteúdo de um cartão de resposta, você pode consultar atributos da sessão. Em tempo de execução, o Amazon Lex substitui essas referências pelos valores adequados dos atributos de sessão. Para obter mais informações, consulte [Definição dos atributos da sessão \(p. 20\)](#). Para ver um exemplo, consulte [Exemplo: uso de um cartão de resposta \(p. 176\)](#).

## Gerando cartões de resposta dinamicamente

Para gerar cartões de resposta dinamicamente em tempo de execução, use a função de inicialização e validação do Lambda para a intenção. Use um cartão de resposta dinâmica quando as respostas forem determinadas em tempo de execução na função do Lambda. Em resposta à entrada do usuário, a função do Lambda gera um cartão de resposta e o retorna na seção `dialogAction` da resposta. Para obter mais informações, consulte [Formato de resposta \(p. 109\)](#).

A seguir, uma resposta parcial de uma função do Lambda que mostra o elemento `responseCard`. Ele gera uma experiência do usuário semelhante à mostrada na seção anterior.

```
responseCard: {  
  "version": 1,  
  "contentType": "application/vnd.amazonaws.card.generic",  
  "genericAttachments": [  
    {  
      "title": "What Flavor?",  
      "subtitle": "What flavor do you want?",  
      "imageUrl": "Link to image",
```



```
"attachmentLinkUrl": "Link to attachment",
"buttons": [
  {
    "text": "Lemon",
    "value": "lemon"
  },
  {
    "text": "Raspberry",
    "value": "raspberry"
  },
  {
    "text": "Plain",
    "value": "plain"
  }
]
}
```

Para ver um exemplo, consulte [Exemplo de bot: ScheduleAppointment \(p. 135\)](#).

## Gerenciar contexto da conversa

Contexto de conversa são as informações que um usuário, seu aplicativo ou uma função do Lambda fornece a um bot do Amazon Lex para atender a uma intenção. O contexto da conversa inclui dados de slot que o usuário fornece, atributos de solicitação definidos pelo aplicativo cliente e atributos de sessão que o aplicativo cliente e as funções do Lambda criam.

### Tópicos

- [Definição dos atributos da sessão \(p. 20\)](#)
- [Definição de atributos de solicitação \(p. 21\)](#)
- [Definição do tempo limite da sessão \(p. 23\)](#)
- [Compartilhamento de informações entre intenções \(p. 24\)](#)
- [Configuração de atributos complexos \(p. 24\)](#)

## Definição dos atributos da sessão

Os Atributos de sessão contêm informações específicas do aplicativo que são passadas entre um bot e um aplicativo cliente durante uma sessão. O Amazon Lex passa atributos da sessão para todas as funções do Lambda configuradas para um bot. Se uma função do Lambda adicionar ou atualizar atributos da sessão, o Amazon Lex passará as novas informações de volta para o aplicativo cliente. Por exemplo:

- No exemplo [Exercício 1: Criar um bot do Amazon Lex usando um esquema \(console\) \(p. 38\)](#), o bot usa o atributo de sessão `price` para manter o preço das flores. A função do Lambda define esse atributo com base nos tipos de flores encomendados. Para obter mais informações, consulte [Etapa 5 \(opcional\): Revise os detalhes do fluxo de informações \(console\) \(p. 52\)](#).
- No exemplo [Exemplo de bot: BookTrip \(p. 154\)](#), o bot usa o atributo de sessão `currentReservation` para manter uma cópia dos dados do tipo de slot durante a conversa para fazer uma reserva em hotel ou alugar um carro. Para obter mais informações, consulte [Detalhes do fluxo de informações \(p. 163\)](#).

Use atributos de sessão em suas funções do Lambda para inicializar um bot e personalizar solicitações e cartões de resposta. Por exemplo:

- Inicialização — em um bot de pedido de pizza, o aplicativo cliente passa o local do usuário como um atributo de sessão na primeira chamada à operação [PostContent](#) (p. 347) ou [PostText](#) (p. 355). Por exemplo, "Location": "111 Maple Street". A função do Lambda usa essas informações para encontrar a pizzeria mais próxima para fazer o pedido.
- Personalizar solicitações — configure solicitações e cartões de resposta para fazer referência a atributos de sessão. Por exemplo, "Olá, [FirstName], quais coberturas você quer?" Se você passar o nome do usuário como um atributo de sessão ({ "FirstName": "Jo" }), o Amazon Lex substituirá o nome pelo espaço reservado. Em seguida, ele envia uma solicitação personalizada para o usuário: "Oi, Jo, quais coberturas você quer?"

Os atributos da sessão são persistidos durante a vigência da sessão. O Amazon Lex os armazena em um armazenamento de dados criptografados até o final da sessão. O cliente pode criar atributos de sessão em uma solicitação usando o [PostContent](#) (p. 347) ou a operação [PostText](#) (p. 355) com o campo `sessionAttributes` definido como um valor. Uma função do Lambda pode criar um atributo de sessão em uma resposta. Depois que o cliente ou uma função do Lambda cria um atributo de sessão, o valor do atributo armazenado é usado sempre que o aplicativo cliente não incluir o campo `sessionAttribute` em uma solicitação para o Amazon Lex.

Por exemplo, suponha que você tenha dois atributos de sessão {"x": "1", "y": "2"}. Se o cliente chamar a operação `PostContent` ou `PostText` sem especificar o campo `sessionAttributes`, o Amazon Lex usará a função do Lambda com os atributos de sessão armazenados ({ "x": 1, "y": 2 }). Se a função do Lambda não retornar atributos de sessão, o Amazon Lex retornará os atributos de sessão armazenados ao aplicativo cliente.

Se o aplicativo cliente ou uma função do Lambda passar atributos de sessão, o Amazon Lex atualizará as informações dos atributos de sessão armazenados. Passar um valor existente, como { "x": 2 }, atualiza o valor armazenado. Se você inserir um novo conjunto de atributos de sessão, como { "z": 3 }, os valores existentes serão removidos e apenas o novo valor será mantido. Quando um mapa vazio, {}, é passado, os valores armazenados são apagados.

Para enviar atributos de sessão para o Amazon Lex, crie um mapa de string para string dos atributos. As considerações a seguir mostram como mapear atributos de sessão:

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

Para a operação `PostText`, insira o mapa no corpo da solicitação usando o campo `sessionAttributes`, como a seguir:

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

Para a operação `PostContent`, codifique o mapa em base64 e o envie como o cabeçalho `x-amz-lex-session-attributes`.

Se você estiver enviando dados binários ou estruturados em um atributo de sessão, deve primeiro transformar os dados em uma string simples. Para obter mais informações, consulte [Configuração de atributos complexos](#) (p. 24).

## Definição de atributos de solicitação

Atributos da solicitação contêm informações específicas de solicitação e aplicam-se apenas à solicitação atual. Um aplicativo cliente envia essas informações ao Amazon Lex. Use atributos de solicitação para

passar informações que não precisam ser mantidas durante toda a sessão. Você pode criar seus próprios atributos de solicitação ou usar atributos predefinidos. Para enviar atributos de solicitação, use o cabeçalho `x-amz-lex-request-attributes` em uma [the section called "PostContent" \(p. 347\)](#) ou no campo `requestAttributes` em uma solicitação [the section called "PostText" \(p. 355\)](#). Como os atributos de solicitação não são persistentes entre as solicitações como os atributos de sessão, eles não são retornados em respostas `PostContent` ou `PostText`.

#### Note

Para enviar informações que são mantidas nas solicitações, use atributos de sessão.

O namespace `x-amz-lex:` é reservado para os atributos de solicitação predefinidos. Não crie atributos de solicitação com o prefixo `x-amz-lex:`.

## Definição de atributos de solicitação predefinidos

O Amazon Lex fornece atributos de solicitação predefinidos para gerenciar a maneira como ele processa as informações enviadas a seu bot. Os atributos não são mantidos durante toda a sessão. Por isso, você deve enviar os atributos predefinidos em cada solicitação. Todos os atributos predefinidos estão no namespace `x-amz-lex:`.

Além dos atributos predefinidos a seguir, o Amazon Lex fornece atributos predefinidos para plataformas de sistemas de mensagens. Para obter uma lista desses atributos, consulte [Implantação de um bot do Amazon Lex em uma plataforma de sistema de mensagens \(p. 114\)](#).

### Como configurar o tipo de resposta

Se tiver dois aplicativos clientes com diferentes recursos, talvez seja necessário restringir o formato das mensagens em uma resposta. Por exemplo, talvez você queira restringir as mensagens enviadas a um cliente web a textos sem formatação, mas permitir que um cliente móvel use o texto sem formatação e Speech Synthesis Markup Language (SSML). Para definir o formato das mensagens retornadas pelas operações [PostContent \(p. 347\)](#) e [PostText \(p. 355\)](#), use o atributo de solicitação `x-amz-lex:accept-content-types`.

Você pode definir o atributo para qualquer combinação dos seguintes tipos de mensagem:

- `PlainText` — a mensagem contém texto sem formatação UTF-8.
- `SSML` — a mensagem contém texto formatado para saída de voz.
- `CustomPayload` — a mensagem contém um formato personalizado que você criou para seu cliente. Você pode definir a carga útil para atender às necessidades de seu aplicativo.

O Amazon Lex retorna apenas mensagens com o tipo especificado no campo `Message` da resposta. Você pode definir mais de um valor. Para isso, separe os valores com uma vírgula. Se estiver usando grupos de mensagens, cada um deverá conter pelo menos uma mensagem do tipo especificado. Do contrário, você receberá um erro `NoUsableMessageException`. Para obter mais informações, consulte [Grupos de mensagens \(p. 15\)](#).

#### Note

O atributo de solicitação `x-amz-lex:accept-content-types` não tem efeito sobre o conteúdo do corpo HTML. O conteúdo da resposta de uma operação `PostText` sempre é texto sem formatação UTF-8. O corpo da resposta de uma operação `PostContent` contém dados no formato definido no cabeçalho `Accept` da solicitação.

### Definição do fuso horário preferido

Para definir o fuso horário usado para resolver datas de forma que seja relativo ao fuso horário do usuário, use o atributo de solicitação `x-amz-lex:time-zone`. Se um fuso horário não está especificado no atributo `x-amz-lex:time-zone`, o padrão depende da região que você está usando para o seu bot.

| Região                            | Fuso horário padrão |
|-----------------------------------|---------------------|
| Leste dos EUA (Norte da Virgínia) | America/New_York    |
| Oeste dos EUA (Oregon)            | America/Los_Angeles |
| UE (Irlanda)                      | Europe/Dublin       |

Por exemplo, se o usuário responde `tomorrow` em resposta à solicitação "Which day would you like your package delivered? (Quando você deseja que o pacote seja entregue?)" a data real em que o pacote será entregue dependerá do fuso horário do usuário. Por exemplo, quando ele é 01:00 de 16 de setembro em Nova York, são 22:00 de 15 de setembro em Los Angeles. Se uma pessoa em Los Angeles encomendar um pacote a ser entregue "amanhã" usando o fuso horário padrão, o pacote será entregue no dia 17, e não no dia 16. No entanto, se você definir o `x-amz-lex:time-zone` atributo de solicitação `America/Los_Angeles`, o pacote será entregue no dia 16.

Você pode definir o atributo para qualquer um nomes de fuso horário IANA (Internet Assigned Number Authority). Para ver a lista de nomes de fuso horário, consulte a [Lista de fusos horários do banco de dados tz](#) na Wikipédia.

## Definição de atributos de solicitação definidos pelo usuário

O atributo de solicitação definido pelo usuário são os dados que você envia para seu bot em cada solicitação. Você enviar as informações no `amz-lex-request-attributes` cabeçalho de uma solicitação `PostContent` ou no campo `requestAttributes` de uma solicitação `PostText`.

Para enviar atributos de solicitação ao Amazon Lex, crie um mapa de string para string dos atributos. As considerações a seguir mostram como mapear atributos de solicitação:

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

Para a operação `PostText`, insira o mapa no corpo da solicitação usando o campo `requestAttributes`, como a seguir:

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

Para a operação `PostContent`, codifique o mapa em base64 e o envie como o cabeçalho `x-amz-lex-request-attributes`.

Se você está enviando dados binários ou estruturados em um atributo de solicitação, você deve primeiro transformar os dados em uma string simples. Para obter mais informações, consulte [Configuração de atributos complexos](#) (p. 24).

## Definição do tempo limite da sessão

O Amazon Lex retém informações de contexto — dados de slot e atributos de sessão — até o fim de uma sessão de conversa. Para controlar o tempo de duração de uma sessão em um bot, defina o tempo limite da sessão. Por padrão, a duração da sessão é de 5 minutos, mas você pode especificar qualquer duração entre 0 e 1.440 minutos (24 horas).

Por exemplo, suponha que você crie um bot `ShoeOrdering` que seja compatível com intenções como `OrderShoes` e `GetOrderStatus`. Quando o Amazon Lex detecta que a intenção do usuário é encomendar sapatos, ele pede os dados do slot. Por exemplo, ele pergunta o tamanho, a cor, a marca, etc. Se o usuário fornecer alguns dados do slot, mas não finalizar a compra de sapato, o Amazon Lex memorizará todos os dados do slot e os atributos da sessão inteira. Se o usuário retornar para a sessão antes que ela expire, ele ou ela pode fornecer os dados de slot restantes e concluir a compra.

No console do Amazon Lex, você define o tempo limite de sessão ao criar um bot. Com a interface de linha de comando da AWS (CLI da AWS) ou API, você define esse limite ao criar ou atualizar um bot com a [PutBot \(p. 311\)](#) operação configurando o campo `idleSessionTTLInSeconds`.

## Compartilhamento de informações entre intenções

O Amazon Lex oferece suporte ao compartilhamento de informações entre intenções. Para compartilhar entre intenções, use atributos de sessão.

Por exemplo, um usuário do bot `ShoeOrdering` começa a pedir sapatos. O bot inicia uma conversa com o usuário, coletando dados de slot como tamanho, cor e marca do sapato. Quando o usuário faz um pedido, a função do Lambda; que atende ao pedido define o atributo de sessão `orderNumber`, que contém o número do pedido. Para obter o status do pedido, o usuário usa a intenção `GetOrderStatus`. O bot pode solicitar dados de slot ao usuário, como número do pedido e data do pedido. Quando o bot tem as informações necessárias, ele retorna o status do pedido.

Se você acha que seus usuários podem mudar de intenção durante uma sessão, você pode projetar seu bot para retornar o status do pedido mais recente. Em vez de pedir ao usuário as informações do pedido novamente, você usa o atributo de sessão `orderNumber` para compartilhar informações entre intenções e cumprir a intenção `GetOrderStatus`. O bot faz isso ao retornar o status do último pedido feito pelo usuário.

Para obter um exemplo de compartilhamento de informações entre intenções, consulte [Exemplo de bot: BookTrip \(p. 154\)](#).

## Configuração de atributos complexos

Os atributos de solicitação e de sessão são mapas de string para string de atributos e valores. Em muitos casos, você pode usar o mapa de string para transferir valores de atributo entre o aplicativo cliente e o bot. Em alguns casos, no entanto, pode ser necessário transferir uma estrutura complexa ou dados binários que não podem ser facilmente convertidos em um mapa de string. Por exemplo, o seguinte objeto JSON representa um arranjo com as três cidades mais populosas dos Estados Unidos:

```
{
  "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"  
    }  
  }  
]  
}
```

Esse arranjo de dados não funciona bem como mapa de string para string. Nesse caso, você pode transformar um objeto em uma string simples para que você possa enviá-lo ao seu bot com as operações [PostContent](#) (p. 347) e [PostText](#) (p. 355).

Por exemplo, se você estiver usando o JavaScript, você pode usar a operação `JSON.stringify` para converter um objeto para JSON e a operação `JSON.parse` para converter texto JSON para um objeto JavaScript:

```
// 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);
```

Para enviar atributos de sessão com a operação `PostContent`, você deve codificar os atributos em base64 antes de adicioná-los ao cabeçalho de solicitação, como mostrado no seguinte código JavaScript:

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

Você pode enviar dados binários para as operações `PostContent` e `PostText` convertendo os dados para uma string codificada em base64 e, em seguida, enviando a string como o valor nos atributos de sessão:

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

## Gerenciamento de sessões com a API do Amazon Lex

Quando um usuário inicia uma conversa com seu bot, o Amazon Lex cria uma sessão. As informações trocadas entre o aplicativo e o Amazon Lex compõem o estado da sessão para a conversa. Quando você faz uma solicitação, a sessão é identificada por uma combinação do nome do bot e um identificador de usuário especificado por você. Para obter mais informações sobre o identificador de usuário, consulte o campo `userId` na operação [PostContent](#) (p. 347) ou [PostText](#) (p. 355).

A resposta de uma operação de sessão inclui um identificador exclusivo que identifica uma sessão específica com um usuário. Você pode usar esse identificador durante o teste ou para ajudar a solucionar problemas no bot.

É possível modificar o estado da sessão enviado entre o aplicativo e o bot. Por exemplo, você pode criar e modificar os atributos que contêm informações personalizadas sobre a sessão e alterar o fluxo da conversa definindo o contexto de diálogo para interpretar o próximo enunciado.

Há duas maneiras de atualizar o estado da sessão. A primeira é usar uma função do Lambda com a operação `PostContent` ou `PostText` chamada após cada passagem da conversa. Para obter mais

informações, consulte [Uso de funções do Lambda \(p. 106\)](#). A outra é usar a API de tempo de execução do Amazon Lex no seu aplicativo para fazer alterações no estado da sessão.

A API de tempo de execução do Amazon Lex oferece operações que permitem gerenciar informações da sessão para uma conversa com o bot. As operações são [PutSession \(p. 361\)](#), [GetSession \(p. 344\)](#) e [DeleteSession \(p. 341\)](#). Você pode usar essas operações para obter informações sobre o estado da sessão do seu usuário com o bot e ter um controle apurado sobre o estado.

Use a operação `GetSession` quando desejar obter o estado atual da sessão. A operação retorna o estado atual da sessão, incluindo o estado do diálogo com o usuário, todos os atributos de sessão que foram definidos e valores de slot das três últimas intenções com as quais o usuário interagiu.

A operação `PutSession` permite manipular diretamente o estado da sessão atual. Você pode definir o tipo de ação de diálogo que o bot realizará em seguida. Isso oferece a você controle sobre o fluxo da conversa com o bot. Defina o campo de ação de diálogo `type` como `Delegate` para que o Amazon Lex determine a próxima ação do bot.

Você pode usar a operação `PutSession` para criar uma nova sessão com um bot e definir a intenção com a qual ele deve começar. Também é possível usar a operação `PutSession` para mudar de uma intenção para outra. Ao criar uma sessão ou alterar a intenção, você também pode definir o estado da sessão, como valores de slot e atributos de sessão. Quando a nova intenção é concluída, você tem a opção de reiniciar a intenção anterior. É possível usar a operação `GetSession` para obter o estado do diálogo da intenção anterior do Amazon Lex e usar as informações para definir o estado do diálogo da intenção.

A resposta da operação `PutSession` contém as mesmas informações que a operação `PostContent`. Você pode usar essas informações para solicitar a próxima informação ao usuário, da mesma forma que faria com a resposta da operação `PostContent`.

Use a operação `DeleteSession` para remover uma sessão existente e começar de novo com uma nova sessão. Por exemplo, ao testar seu bot, você pode usar a operação `DeleteSession` para remover as sessões de teste do seu bot.

As operações de sessão trabalham com as funções do Lambda de atendimento. Por exemplo, se sua função do Lambda retornar `Failed` como o estado de atendimento, você poderá usar a operação `PutSession` para definir o tipo de ação de diálogo como `close` e `fulfillmentState` como `ReadyForFulfillment` para repetir a etapa de atendimento.

Veja a seguir algumas ações que você pode executar com as operações de sessão:

- Fazer com que o bot inicie uma conversa em vez de esperar pelo usuário.
- Alternar entre as intenções durante uma conversa.
- Voltar para uma intenção anterior.
- Iniciar ou reiniciar uma conversa no meio da interação.
- Validar valores de slot e fazer com que o bot solicite novamente valores que não são válidos.

Cada uma delas é descrita com mais detalhes a seguir.

## Alternância entre intenções

Você pode usar a operação `PutSession` para alternar de uma intenção para outra. Também é possível usá-la para voltar para uma intenção anterior. Você pode usar a operação `PutSession` para definir atributos de sessão ou valores de slot para a nova intenção.

- Chame a operação `PutSession`. Defina o nome da intenção como o nome da nova intenção e defina a ação de diálogo como `Delegate`. Você também pode definir quaisquer valores de slot ou atributos de sessão necessários para a nova intenção.



- O Amazon Lex iniciará uma conversa com o usuário utilizando a nova intenção.

## Como retomar uma intenção anterior

Para retomar uma intenção anterior, use a operação `GetSession` para obter o resumo da intenção e, depois, use a operação `PutSession` para definir a intenção como seu estado de diálogo anterior.

- Chame a operação `GetSession`. A resposta da operação inclui um resumo do estado de diálogo das últimas três intenções com as quais o usuário interagiu.
- Usando as informações do resumo de intenção, chame a operação `PutSession`. Isso retornará o usuário para a intenção anterior no mesmo local na conversa.

Em alguns casos, pode ser necessário retomar a conversa do usuário com o bot. Por exemplo, digamos que você tenha criado um bot de atendimento ao cliente. Seu aplicativo determina que o usuário precisa conversar com um representante de atendimento ao cliente. Depois de falar com o usuário, o representante poderá direcionar a conversa de volta para o bot com as informações coletadas.

Para retomar uma sessão, use etapas semelhantes a estas:

- Seu aplicativo determina que o usuário precisa falar com um representante de atendimento ao cliente.
- Use a operação `GetSession` para obter o estado de diálogo atual da intenção.
- O representante de atendimento ao cliente se comunica com o usuário e resolve o problema.
- Use a operação `PutSession` para definir o estado de diálogo da intenção. Isso pode incluir definir valores de slot, definir atributos de sessão ou alterar a intenção.
- O bot retoma a conversa com o usuário.

## Como iniciar uma nova sessão

Se você desejar que o bot inicie a conversa com o usuário, use a operação `PutSession`.

- Crie uma intenção de boas-vindas sem slots e uma mensagem de conclusão solicitando que o usuário indique uma intenção. Por exemplo, "O que você gostaria de pedir? Você pode dizer "Pedir uma bebida" ou "Pedir uma pizza".
- Chame a operação `PutSession`. Defina o nome da intenção como o nome da intenção de boas-vindas e defina a ação de diálogo como `Delegate`.
- O Amazon Lex responderá com o prompt da sua intenção de boas-vindas para iniciar a conversa com o usuário.

## Validação de valores de slot

Você pode validar as respostas ao bot usando seu aplicativo cliente. Se a resposta não for válida, use a operação `PutSession` para obter uma nova resposta do usuário. Por exemplo, suponha que seu bot de pedido de flores só possa vender tulipas, rosas e lírios. Se o usuário pedir cravos, o aplicativo poderá fazer o seguinte:

- Examinar o valor do slot retornado pela resposta `PostText` ou `PostContent`.
- Se o valor do slot não for válido, chame a operação `PutSession`. Seu aplicativo deve limpar o valor do slot, definir o campo `slotToElicit` e definir o valor `dialogAction.type` como `elicitSlot`. Você também poderá definir os campos `messageFormat` e `message` se desejar alterar a mensagem usada pelo Amazon Lex para obter o valor do slot.



## Opções de implantação de bot

No momento, o Amazon Lex fornece as seguintes opções de implantação do bot:

- [AWS Mobile SDK](#) – você pode criar aplicativos móveis que se comunicam com o Amazon Lex usando os SDKs do AWS Mobile.
- Facebook Messenger – você pode integrar sua página do Facebook Messenger com o bot do Amazon Lex; para que os usuários finais no Facebook possam se comunicar com o bot. Na implementação atual, essa integração suporta apenas as mensagens de entrada de texto.
- Slack – você pode integrar seu bot do Amazon Lex com um aplicativo de sistema de mensagens Slack.
- Twilio – você pode integrar seu bot do Amazon Lex com o Twilio Simple Messaging Service (SMS).

Para ver exemplos, consulte [Implantação de bots do Amazon Lex \(p. 114\)](#).

## Intenções integradas e tipos de slot

Para facilitar a criação dos bots, o Amazon Lex permite que você use as intenções integradas e os tipos de slot padrão da Alexa.

Tópicos

- [Intenções integradas \(p. 28\)](#)
- [Tipos de slot integrados \(p. 28\)](#)

### Intenções integradas

Para ações comuns, você pode usar a biblioteca de intenções integradas padrão da Alexa. Para criar uma intenção de uma intenção integrada, escolha uma intenção no console e forneça um novo nome. A nova intenção tem a configuração de intenção básica, como os enunciados de amostra e slots.

Para obter uma lista de intenções integradas, consulte [Intenções integradas padrão](#) no Alexa Skills Kit.

Note

O Amazon Lex não oferece suporte às seguintes intenções:

- `AMAZON.YesIntent`
- `AMAZON.NoIntent`
- As intenções na [Biblioteca de intenções integradas](#) do Alexa Skills Kit

Na implementação atual, você não pode:

- Adicionar ou remover enunciados de amostra ou slots da intenção básica
- Configurar slots para intenções integradas

### Tipos de slot integrados

O Amazon Lex oferece suporte aos tipos de slot integrados do Alexa Skills Kit. É possível criar slots desses tipos em suas intenções. Isso elimina a necessidade de criar valores de enumeração para dados de slot comumente usados, como data, hora e local. Os tipos de slot integrados não têm versões.

Para obter uma lista de tipos de slot integrados disponíveis, consulte [Referência do tipo de slot](#) na documentação do Alexa Skills Kit.

#### Note

- O Amazon Lex não oferece suporte ao tipo de slot integrado `AMAZON.LITERAL` ou ao `AMAZON.SearchQuery`.
- O Amazon Lex amplia os tipos de slot `AMAZON.NUMBER` e `AMAZON.TIME`. Veja a seguir mais informações.

Além dos tipos de slot do Alexa Skills Kit, o Amazon Lex oferece suporte aos tipos de slot integrados a seguir. Os tipos de slot marcados como "Pré-visualização para desenvolvedores" estão em pré-visualização e podem ser alterados.

| Tipo de slot                                | Descrição breve  | Idiomas suportados | Disponibilidade                       |
|---|--|--------------------|---------------------------------------|
| <a href="#">AMAZON.EmailAddress (p. 30)</a> | Converte palavras que representam um endereço de e-mail em um endereço de e-mail padrão      | Inglês (EUA)       | Pré-visualização para desenvolvedores |
| <a href="#">AMAZON.Percentage (p. 30)</a>   | Converte palavras que representam uma porcentagem em um número e um sinal de porcentagem (%) | Inglês (EUA)       | Pré-visualização para desenvolvedores |
| <a href="#">AMAZON.PhoneNumber (p. 31)</a>  | Converte palavras que representam um número de telefone em uma string numérica               | Inglês (EUA)       | Pré-visualização para desenvolvedores |
| <a href="#">AMAZON.SpeedUnit (p. 31)</a>    | Converte palavras que representam uma unidade de velocidade em uma abreviatura padrão        | Inglês (EUA)       | Pré-visualização para desenvolvedores |
| <a href="#">AMAZON.WeightUnit (p. 32)</a>   | Converte palavras que representam uma unidade de peso em uma abreviatura padrão              | Inglês (EUA)       | Pré-visualização para desenvolvedores |

Quando usados com o Amazon Lex, os seguintes tipos de slot estendem o tipo de slot do Alexa Skill Kit.

| Tipo de slot                          | Descrição breve  | Idiomas suportados | Disponibilidade                       |
|---------------------------------------|--|--------------------|---------------------------------------|
| <a href="#">AMAZON.NUMBER (p. 30)</a> | Converte palavras numéricas em dígitos                       | Inglês (EUA)       | Pré-visualização para desenvolvedores |
| <a href="#">AMAZON.TIME (p. 32)</a>   | Converte palavras que indicam horários em um formato de hora | Inglês (EUA)       | Disponível                            |

## Tipos de slot do Alexa Skills Kit

Você pode usar qualquer um dos tipos de slot do Alexa Skills Kit em seus bots do Amazon Lex. Para usar um dos tipos de slot, especifique o nome do tipo de slot no console ou em uma chamada para a operação [the section called “PutIntent” \(p. 323\)](#). Para obter uma lista de tipos de slot integrados disponíveis, consulte [Referência do tipo de slot](#) no Alexa Skills Kit.

### Note

O Amazon Lex não oferece suporte ao tipo de slot integrado `AMAZON.LITERAL` ou ao `AMAZON.SearchQuery`.

## AMAZON.EmailAddress

Esse tipo de slot está na versão de pré-visualização do Amazon Lex e está sujeito a alterações.

Reconhece palavras que representam um endereço de e-mail fornecido, como `nomedeusuário@domínio`. Os endereços podem incluir os seguintes caracteres especiais em um nome de usuário: sublinhado (`_`), hífen (`-`), ponto (`.`) e o sinal de mais (`+`).

## AMAZON.NUMBER

Esse tipo de slot está na versão de pré-visualização do Amazon Lex e está sujeito a alterações.

Converte palavras numéricas em dígitos.

O Amazon Lex estende o tipo de slot `AMAZON.NUMBER` para converter palavras ou números que expressam um número em dígitos, incluindo números decimais. A tabela a seguir mostra como o tipo de slot `AMAZON.NUMBER` captura palavras numéricas.

| Entrada                                 | Resposta |
|---|----------|
| cento e vinte e três ponto quatro cinco | 123.45   |
| cento e vinte e três ponto quatro cinco | 123.45   |
| ponto quatro dois                       | 0.42     |
| ponto quarenta e dois                   | 0.42     |
| 232.998                                 | 232.998  |
| 50                                      | 50       |

## AMAZON.Percentage

Esse tipo de slot está na versão de pré-visualização do Amazon Lex e está sujeito a alterações.

Converte palavras e símbolos que representam uma porcentagem em um valor numérico com um sinal de porcentagem (%).

Se o usuário inserir um número sem um sinal de porcentagem ou as palavras "por cento", o valor do slot será definido para o número. A tabela a seguir mostra como o tipo de slot `AMAZON.Percentage` captura porcentagens.

| Entrada      | Resposta |
|--------------|----------|
| 50 por cento | 50%      |
| 0,4por cento | 0.4%     |
| 23,5%        | 23,5%    |
| 25           | 25       |

## AMAZON.PhoneNumber

Esse tipo de slot está na versão de pré-visualização do Amazon Lex e está sujeito a alterações.

Converte os números ou as palavras que representam um número de telefone em um formato de string sem a pontuação da seguinte forma.

### Note

Somente os números de telefone dos EUA são compatíveis.

| Type  | Descrição  | Entrada          | Result       |
|---|--|------------------|--------------|
| Número internacional com sinal de mais (+) à esquerda | Número de 11 dígitos com sinal de mais (+) à esquerda.         | +1 509 555-1212  | +15095551212 |
| Número internacional sem sinal de mais (+) à esquerda | Número de 11 dígitos sem sinal de mais (+) à esquerda          | 1 (509) 555-1212 | 15095551212  |
| Número nacional                                       | Número de 10 dígitos sem código internacional                  | (509) 555-1212   | 5095551212   |
| Número local  | Número de 7 dígitos sem código internacional ou código de área | 555-1212         | 5551212      |

## AMAZON.SpeedUnit

Esse tipo de slot está na versão de pré-visualização do Amazon Lex e está sujeito a alterações.

Converte palavras que representam unidades de velocidade na abreviatura correspondente.

Por exemplo, "milhas por hora" é convertido em mph.

Os exemplos a seguir mostram como o tipo de slot `AMAZON.SpeedUnit` captura unidades de velocidade.

| Unidade de velocidade                               | Abreviação |
|---|------------|
| milhas por hora, mph, MPH, m/h                      | mph        |
| quilômetros por hora, km por hora, kmph, KMPH, km/h | kmph       |
| metros por segundo, mps, MPS, m/s                   | mps        |
| milhas náuticas por hora, nós, nó                   | nó         |

## AMAZON.TIME

Converte palavras que representam horários em valores de hora.

O Amazon Lex estende o tipo de slot `AMAZON.TIME` para incluir resoluções para horários ambíguos. Quando o usuário insere um horário ambíguo, o Amazon Lex usa o atributo `slotDetails` de um evento do Lambda para passar resoluções para horários ambíguos à função do Lambda. Por exemplo, se seu bot solicitar um horário de entrega, o usuário poderá responder: "10 horas". Esse horário é ambíguo. Pode ser 10h ou 22h. Nesse caso, o valor no mapa `slots` é `null`, e a entidade `slotDetails` contém as duas possíveis resoluções do horário. Amazon Lex insere o seguinte na função do Lambda.

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

Quando o usuário responde com um horário não ambíguo, o Amazon Lex envia o horário para a função do Lambda no atributo `slots` do evento do Lambda, e o atributo `slotDetails` fica vazio. Por exemplo, se o usuário responder à solicitação para um horário de entrega com "10:00 PM (22h)", o Amazon Lex inserirá o seguinte na função do Lambda.

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

Para obter mais informações sobre os dados enviados do Amazon Lex para uma função do Lambda, consulte [Formato de eventos de entrada \(p. 106\)](#).

## AMAZON.WeightUnit

Esse tipo de slot está na versão de pré-visualização do Amazon Lex e está sujeito a alterações.

Converte palavras que representam uma unidade de peso na abreviatura correspondente. Por exemplo, "quilograma" é convertido para kg.

Os exemplos a seguir mostram como o tipo de slot `AMAZON.WeightUnit` captura unidades de peso:

| Unidade de peso             | Abreviação |
|-----------------------------|------------|
| quilogramas, quilos, kg, KG | kg         |
| gramas, g, G, g             | g          |
| miligramas, mg, mg          | mg         |
| libras, lb, LB              | lb         |
| onças, oz, OZ               | oz         |
| tonelada, tonelada, t       | t          |
| quilotonelada, kt           | kt         |

## Tipos de slot personalizados

Para cada intenção, você pode especificar parâmetros que indicam as informações de que a intenção precisa para cumprir a solicitação do usuário. Esses parâmetros ou slots têm um tipo. Um tipo de slot é uma lista de valores que o Amazon Lex usa para treinar o modelo de Machine Learning para reconhecer os valores de um slot. Por exemplo, você pode definir um tipo de slot chamado "Genres." Cada valor no tipo de slot é o nome de um gênero, "comédia", "aventura", "documentário", etc. Você pode definir um sinônimo para um valor de tipo de slot. Por exemplo, você pode definir os sinônimos "engraçado" e "humor" para o valor "comédia".

Você pode configurar o tipo de slot para restringir a resolução aos valores do slot. Os valores do slot serão usados como uma enumeração e o valor inserido pelo usuário será resolvido para o slot valor somente se ele for o mesmo que um dos valores de slot ou um sinônimo. Um sinônimo é resolvido para o valor de slot correspondente. Por exemplo, se o usuário inserir "engraçado", isso será resolvido para o valor do slot "comédia".

Como alternativa, você pode configurar o tipo de slot para expandir os valores. Os valores do slot serão usados como dados de treinamento e o slot será resolvido para o valor fornecido pelo usuário se ele for semelhante aos valores e sinônimos do slot. Esse é o comportamento padrão.

O Amazon Lex mantém uma lista de possíveis resoluções para um slot. Cada entrada na lista fornece um valor de resolução que o Amazon Lex reconhece como possibilidades adicionais para o slot. Um valor de resolução é o melhor esforço para corresponder ao valor do slot. A lista contém até cinco valores.

Quando o valor inserido pelo usuário é um sinônimo, a primeira entrada na lista de valores de resolução é o valor do tipo do slot. Por exemplo, se o usuário insere "engraçado", o campo `slots` contém "engraçado" e a primeira entrada no campo `slotDetails` é "comédia". Você pode configurar o `valueSelectionStrategy` quando cria ou atualiza um tipo de slot com a operação [PutSlotType](#) (p. 333) para que o valor do slot seja preenchido com o primeiro valor na lista de resolução.

Se você estiver usando uma função do Lambda, o evento de entrada para a função incluirá uma lista de resoluções chamada `slotDetails`. O exemplo a seguir mostra o slot e a seção de detalhes do slot da entrada para uma função do Lambda:

```
"slots": {
```

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

Para cada tipo de slot, você pode definir um máximo de 10.000 valores e sinônimos. Cada bot pode ter um número total de 50.000 valores e sinônimos de tipos de slots. Por exemplo, você pode ter cinco tipos de slot, cada um com 5.000 valores e 5.000 sinônimos, ou você pode ter 10 tipos de slot, cada um com 2.500 valores e 2.500 sinônimos. Se exceder esses limites, você receberá um `LimitExceededException` quando chamar a operação [PutBot](#) (p. 311).

# Conceitos básicos do Amazon Lex

O Amazon Lex fornece operações de API que você pode integrar com os aplicativos existentes. Para obter uma lista das operações compatíveis, consulte [API Reference \(p. 216\)](#). Você pode usar qualquer uma das opções a seguir:

- SDK da AWS — ao usar os SDKs, suas solicitações ao Amazon Lex são automaticamente assinadas e autenticadas usando as credenciais fornecidas por você. Esta é a escolha recomendada para a criação de suas aplicações.
- AWS CLI — você pode usar a AWS CLI para acessar qualquer recurso do Amazon Lex sem necessidade de escrever nenhum código.
- Console AWS — o console é a maneira mais fácil de começar a testar e usar o Amazon Lex

Se você é novo no Amazon Lex, recomendamos a leitura de [Amazon Lex: Como ele funciona \(p. 3\)](#) antes de continuar.

## Tópicos

- [Etapa 1: Configurar uma conta da AWS e criar um usuário administrador \(p. 35\)](#)
- [Etapa 2: Configurar a AWS Command Line Interface \(p. 36\)](#)
- [Etapa 3: Conceitos básicos \(console\) \(p. 37\)](#)
- [Etapa 4: Conceitos básicos \(AWS CLI\) \(p. 76\)](#)

## Etapa 1: Configurar uma conta da AWS e criar um usuário administrador

Antes de usar o Amazon Lex pela primeira vez, execute as seguintes tarefas:

1. [Cadastre-se na AWS \(p. 35\)](#)
2. [Criar um usuário da IAM \(p. 36\)](#)

## Cadastre-se na AWS

Se você já tiver uma conta da AWS, ignore esta etapa.

Ao se cadastrar na Amazon Web Services (AWS), sua conta da AWS é automaticamente cadastrada em todos os serviços da AWS, incluindo o Amazon Lex. Você será cobrado apenas pelos serviços que usar.

Com o Amazon Lex, você paga apenas pelos recursos que usa. Se você for um novo cliente da AWS, poderá começar o Amazon Lex gratuitamente. Para obter mais informações, consulte [Nível de uso gratuito da AWS](#).

Se já tiver uma conta da AWS, passe para a próxima tarefa. Se você ainda não possuir uma conta da AWS, use o procedimento a seguir para criar uma.



Para criar uma conta da AWS

1. Abra <https://portal.aws.amazon.com/billing/signup>.
2. Siga as instruções online.

Parte do procedimento de cadastro envolve uma chamada telefônica e a digitação de um código de verificação usando o teclado do telefone.

Anote o ID de sua conta da AWS, pois ele será necessário para a próxima tarefa.

## Criar um usuário da IAM

Os serviços da AWS, como o Amazon Lex, exigem que você forneça credenciais ao acessá-los, para que o serviço possa determinar se você tem permissões para acessar os recursos pertencentes a esse serviço. O console requer sua senha. Você pode criar chaves de acesso para a conta da AWS para acessar a AWS CLI ou a API.

No entanto, não recomendamos que você acesse a AWS usando as credenciais de sua conta da AWS. Em vez disso, recomendamos que você:

- Use o AWS Identity and Access Management (IAM) para criar um usuário do IAM.
- Adicione o usuário a um grupo do IAM com permissões de administrador
- Conceda permissões de administrador ao usuário do IAM que você criou.

Em seguida, você pode acessar a AWS usando uma URL especial e as credenciais do usuário do IAM.

Os exercícios de conceitos básicos deste guia pressupõem que você tenha um usuário (`adminuser`) com privilégios de administrador. Siga o procedimento para criar `adminuser` na conta.

Para criar um usuário administrador e fazer login no console

1. Crie um usuário administrador chamado `adminuser`, em sua conta da AWS. Para obter instruções, consulte [Criar o primeiro usuário e grupo de administradores do IAM](#) no Guia do usuário do IAM.
2. Como usuário, você pode fazer login no Console de gerenciamento da AWS usando uma URL especial. Para obter mais informações, consulte [Como os usuários fazem login em sua conta](#) no Guia do usuário do IAM.

Para obter mais informações sobre IAM, consulte o seguinte:

- [AWS Identity and Access Management \(IAM\)](#)
- [Conceitos básicos](#)
- [Guia do usuário do IAM](#)

## Próxima etapa

[Etapa 2: Configurar aAWS Command Line Interface](#) (p. 36)

# Etapa 2: Configurar aAWS Command Line Interface

Se você preferir usar o Amazon Lex com a AWS Command Line Interface (AWS CLI), faça download e configure-o.

## Important

Você não precisa da AWS CLI para executar as etapas nos exercícios de Conceitos básicos. No entanto, alguns dos últimos exercícios neste guia usam a AWS CLI. Se você preferir começar usando o console, ignore esta etapa e vá para [Etapa 3: Conceitos básicos \(console\) \(p. 37\)](#). Mais tarde, quando você precisar da AWS CLI, volte aqui para configurá-la.

### Para configurar a AWS CLI

1. Baixe e configure a AWS CLI. Para obter instruções, consulte os tópicos a seguir no Guia do usuário do AWS Command Line Interface:
  - [Começar a usar a AWS Command Line Interface](#)
  - [Configuração do AWS Command Line Interface](#)
2. Adicione um perfil nomeado para o usuário administrador no final do arquivo config da AWS CLI. Use esse perfil ao executar os comandos da AWS CLI. Para obter mais informações sobre perfis nomeados, consulte [Perfis nomeados](#) no Guia do usuário do AWS Command Line Interface.

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

Para obter uma lista das regiões da AWS disponíveis, consulte [Regiões e endpoints](#) no Referência geral do Amazon Web Services.

3. Verifique a configuração digitando o comando de ajuda na solicitação de comando:

```
aws help
```

### [Etapa 3: Conceitos básicos \(console\) \(p. 37\)](#)

## Etapa 3: Conceitos básicos (console)

A forma mais fácil de aprender a usar o Amazon Lex é usar o console. Para começar, criamos os seguintes exercícios que usam o console:

- Exercício 1 — Criar um bot do Amazon Lex usando um esquema, um bot predefinido que fornece toda a configuração de bot necessária. Você precisará de pouco trabalho para testar a configuração completa.

Além disso, você usa o esquema da função do Lambda fornecido pelo AWS Lambda para criar uma função do Lambda. A função é um hook de código que usa um código predefinido compatível com o bot.

- Exercício 2 — Criar um bot personalizado criando e configurando manualmente um bot. Você também pode criar uma função do Lambda como um gancho de código. Um código de exemplo é fornecido.
- Exercício 3 — Publicar um bot e, em seguida, criar uma nova versão. Como parte deste exercício, você criará um alias que aponte para a versão do bot.

### Tópicos

- [Exercício 1: Criar um bot do Amazon Lex usando um esquema \(console\) \(p. 38\)](#)
- [Exercício 2: Criar um bot personalizado do Amazon Lex \(p. 64\)](#)
- [Exercício 3: publique uma versão e crie um alias \(p. 75\)](#)

## Exercício 1: Criar um bot do Amazon Lex usando um esquema (console)

Neste exercício, você faz o seguinte:

- Crie seu primeiro bot do Amazon Lex e teste-o no console do Amazon Lex.

Para este exercício, você usará o esquema `OrderFlowers`. Para obter mais informações sobre esquemas, consulte [Esquemas do Amazon Lex e do AWS Lambda \(p. 113\)](#).

- Crie uma função do AWS Lambda e teste-a no console do Lambda. Ao processar uma solicitação, seu bot chama a função do Lambda. Para este exercício, você usa um esquema do Lambda (`lex-order-flowers-python`) fornecido no console do AWS Lambda para criar a função do Lambda. O código do esquema ilustra como você pode usar a mesma função do Lambda para executar a inicialização e a validação e atender à intenção `OrderFlowers`.
- Atualize o bot para adicionar a função do Lambda como o gancho de código para atender à intenção. Teste a experiência completa.

As seções a seguir explicam o que os esquemas fazem.

### Bot do Amazon Lex: visão geral do esquema

Você pode usar o esquema `OrderFlowers` para criar um bot do Amazon Lex. Para obter mais informações sobre a estrutura de um bot, consulte [Amazon Lex: Como ele funciona \(p. 3\)](#). O bot é pré-configurado da seguinte forma:

- Intenção – `OrderFlowers`
- Tipos de slot – um tipo de slot personalizado chamado `FlowerTypes` com valores de enumeração: `roses`, `lilies` e `tulips`.
- Slots – a intenção requer as seguintes informações (ou seja, slots) para que o bot possa atender à intenção.
  - `PickupTime` (tipo integrado `AMAZON.TIME`)
  - `FlowerType` (tipo personalizado `FlowerTypes`)
  - `PickupDate` (tipo integrado `AMAZON.DATE`)
- Enunciado – os seguintes enunciados de exemplo indicam a intenção do usuário:
  - "Gostaria de escolher flores."
  - "Gostaria de pedir algumas flores."
- Solicitações – depois que o bot identifica a intenção, ele usa as seguintes solicitações para preencher os slots:
  - Solicitação do slot `FlowerType` – "What type of flowers would you like to order? (Que tipo de flores você deseja encomendar?)"
  - Solicitação do slot `PickupDate` – "What day do you want the {FlowerType} to be picked up? (Em que dia você deseja que a flor seja retirada?)"
  - Solicitação do slot `PickupTime` – "At what time do you want the {FlowerType} to be picked up? (Em que hora você deseja que a flor seja retirada?)"
  - Declaração de confirmação – "Okay, your {FlowerType} will be ready for pickup by {PickupTime} on {PickupDate}. (OK, sua flor estará pronta para retirada no horário de retirada na data de retirada). Tudo bem?"

## Função do AWS Lambda: resumo do esquema

A função do Lambda neste exercício executa a inicialização e a validação e as tarefas de atendimento. Portanto, após criar a função do Lambda, você deve atualizar a configuração da intenção especificando a mesma função do Lambda como um gancho de código para lidar com a inicialização e a validação e as tarefas de atendimento.

- Como gancho de inicialização e validação, a função do Lambda executa uma validação básica. Por exemplo, se o usuário fornecer uma hora de retirada fora do horário comercial normal, a função do Lambda direcionará o Amazon Lex para solicitar que o usuário insira o horário novamente.
- Como parte do gancho de código de atendimento, a função do Lambda retorna uma mensagem de resumo indicando que o pedido de flores foi colocado (ou seja, a intenção é atendida).

Próxima etapa

[Etapa 1: Criar um bot do Amazon Lex \(console\) \(p. 39\)](#)

## Etapa 1: Criar um bot do Amazon Lex (console)

Para este exercício, crie um bot para pedir flores, chamado OrderFlowersBot.

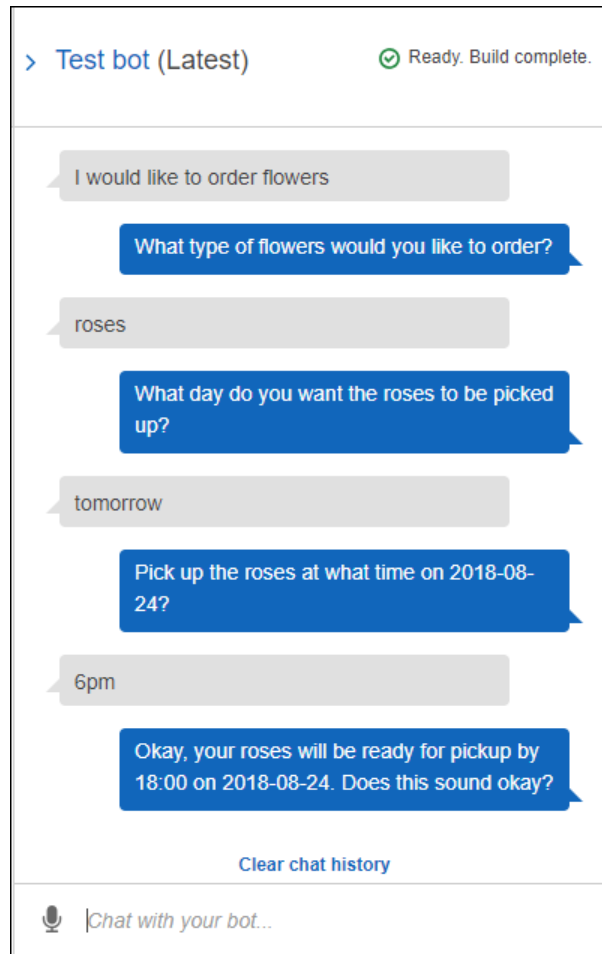
Para criar um bot do Amazon Lex (console)

1. Faça login no Console de gerenciamento da AWS e abra o console do Amazon Lex em <https://console.aws.amazon.com/lex/>.
2. Se este for o seu primeiro bot, selecione Get Started (Conceitos básicos); caso contrário, na página Bots, selecione Create (Criar).
3. Na página Create your Lex bot, forneça as informações a seguir e escolha Create.
  - Escolha o esquema OrderFlowers.
  - Deixe o nome do bot padrão (OrderFlowers).
  - Em COPPA, selecione **No**.
4. Escolha Criar. O console faz as solicitações necessárias ao Amazon Lex para salvar a configuração. Em seguida, o console exibe a janela do editor de bot.
5. Aguarde a confirmação de criação do bot.
6. Teste o bot.

### Note

Você pode testar o bot digitando o texto na janela de teste ou, em navegadores compatíveis, selecionando o botão de microfone na janela de teste e falando.

Use o texto de exemplo a seguir para começar uma conversa com o bot e encomendar flores:



Nesta entrada, o bot infere a intenção `OrderFlowers` e solicita os dados de slot. Quando você fornecer todas os dados de slot necessários, o bot cumprirá a intenção (`OrderFlowers`) retornando todas as informações para o aplicativo cliente (neste caso, o console). O console mostra as informações na janela de teste.

Especificamente:

- Na instrução "Em que dia você deseja que as rosas sejam entregues?", o termo "rosas" aparece porque a solicitação para o slot `pickupDate` é configurada usando substituições, `{FlowerType}`. Verifique isso no console.
- A instrução "Está bem, as rosas estarão prontas..." é a solicitação de confirmação que você configurou.
- A última instrução ("`FlowerType:roses...`") contém apenas os dados do slot que são retornados ao cliente, neste caso, na janela de teste. No próximo exercício, você usará uma função do Lambda para atender à intenção. Nesse caso, você recebe uma mensagem indicando que o pedido foi atendido.

Próxima etapa

[Etapa 2 \(opcional\): Revise os detalhes do fluxo de informações \(console\) \(p. 41\)](#)

## Etapa 2 (opcional): Revise os detalhes do fluxo de informações (console)

Esta seção explica o fluxo de informações entre um cliente e o Amazon Lex para cada entrada do usuário em nossa conversa de exemplo.

Para ver o fluxo de informações do conteúdo falado ou digitado, escolha o tópico apropriado.

### Tópicos

- [Etapa 2a \(opcional\): Revise os detalhes do fluxo de informações falado \(console\)](#) (p. 41)
- [Etapa 2b \(opcional\): Revise os detalhes do fluxo de informações digitado \(console\)](#) (p. 45)

## Etapa 2a (opcional): Revise os detalhes do fluxo de informações falado (console)

Esta seção explica o fluxo de informações entre o cliente e o Amazon Lex quando o cliente usa fala para enviar solicitações. Para obter mais informações, consulte [PostContent](#) (p. 347).

1. O usuário diz: Eu gostaria de encomendar flores.
  - a. O cliente (console) envia a seguinte solicitação [PostContent](#) (p. 347) ao Amazon Lex:

```
POST /bot/OrderFlowers/alias/$LATEST/user/4o9wwd hx6nlheferh6a73fuj d3118f5w/content
HTTP/1.1
x-amz-lex-session-attributes: "e30="
Content-Type: "audio/x-l16; sample-rate=16000; channel-count=1"
Accept: "audio/mpeg"

Request body

```

O URI e o corpo da solicitação fornecem informações ao Amazon Lex:

- URI da solicitação – fornece o nome do bot (*OrderFlowers*), o alias do bot (*\$LATEST*) e o nome do usuário (uma string aleatória que identifica o usuário). *content* indica que esta é uma solicitação da API *PostContent* (não uma solicitação *PostText*).
- Cabeçalhos de solicitação
  - *x-amz-lex-session-attributes* – o valor codificado em base64 representa "{}". Quando o cliente faz a primeira solicitação, não há atributos de sessão.
  - *Content-Type* – reflete o formato de áudio.
- Corpo da solicitação – o fluxo de áudio de entrada do usuário ("I would like to order some flowers. (Eu gostaria de encomendar flores.)").

### Note

Se o usuário escolher enviar texto ("Eu gostaria de encomendar flores") para a API *PostContent* em vez de falar, o corpo da solicitação será a entrada do usuário. O cabeçalho *Content-Type* é definido adequadamente:

```
POST /bot/OrderFlowers/alias/$LATEST/user/4o9wwd hx6nlheferh6a73fuj d3118f5w/
content HTTP/1.1
x-amz-lex-session-attributes: "e30="
Content-Type: "text/plain; charset=utf-8"
Accept: accept


```

```
Request body
input stream
```

- b. No fluxo de entrada, o Amazon Lex detecta a intenção (OrderFlowers). Em seguida, ele escolhe um dos slots da intenção (neste caso, o FlowerType) e um de seus prompts de seleção de valor e envia uma resposta com os seguintes cabeçalhos:

```
x-amz-lex-dialog-state:ElicitSlot
x-amz-lex-input-transcript:I would like to order some flowers.
x-amz-lex-intent-name:OrderFlowers
x-amz-lex-message:What type of flowers would you like to order?
x-amz-lex-session-attributes:e30=
x-amz-lex-slot-to-elicite:FlowerType
x-amz-lex-
slots:eyJQaWNrdXBuaW1lIjpudWxsLCJGbG93ZXJueXB1IjpudWxsLCJQaWNrdXBhYXR1IjpudWxsZQ==
```

Os valores do cabeçalho fornecem as seguintes informações:

- x-amz-lex-input-transcript – fornece a transcrição do áudio (entrada do usuário) da solicitação
- x-amz-lex-message – fornece a transcrição do áudio que o Amazon Lex retornou na resposta
- x-amz-lex-slots – a versão codificada em base64 dos slots e dos valores:

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

- x-amz-lex-session-attributes – a versão codificada em base64 dos atributos da sessão ({}).

O cliente reproduz o áudio no corpo da resposta.

## 2. O usuário diz: rosas

- a. O cliente (console) envia a seguinte solicitação [PostContent \(p. 347\)](#) ao Amazon Lex:

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

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

O corpo da solicitação é o stream de áudio de entrada do usuário (rosas). O sessionAttributes permanece vazio.

- b. O Amazon Lex interpreta o fluxo de entrada no contexto da intenção atual (ele lembra que pediu ao usuário informações a respeito do slot FlowerType). O Amazon Lex primeiro atualiza o valor do slot da intenção atual. Em seguida, ele escolhe outro slot (PickupDate), juntamente com uma das mensagens de solicitação (Quando você deseja pegar as rosas?) e retorna uma resposta com os seguintes cabeçalhos:

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

```
x-amz-lex-slot-to-elicite:PickupDate
x-amz-lex-
slots:eyJQaWNrdXBuaW1lIjpudWxsLCJGbg93ZXJueXB1Ijoicm9zaSdzIiwUGlja3VwRGF0ZSI6bnVsbH0=
```

Os valores do cabeçalho fornecem as seguintes informações:

- `x-amz-lex-slots` – a versão codificada em base64 dos slots e dos valores:

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

- `x-amz-lex-session-attributes` – a versão codificada em base64 dos atributos da sessão (`{}`)

O cliente reproduz o áudio no corpo da resposta.

3. O usuário diz: amanhã

- a. O cliente (console) envia a seguinte solicitação [PostContent \(p. 347\)](#) ao Amazon Lex:

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

Request body

```

O corpo da solicitação é o stream de áudio de entrada do usuário ("amanhã"). O `sessionAttributes` permanece vazio.

- b. O Amazon Lex interpreta o fluxo de entrada no contexto da intenção atual (ele lembra que pediu ao usuário informações a respeito do slot `PickupDate`). O Amazon Lex primeiro atualiza o valor do slot (`PickupDate`) da intenção atual. Em seguida, ele escolhe outro slot para escolher o valor para (`PickupTime`) e um dos prompts de seleção de valor (Quando você deseja receber as rosas em 18/03/2017?), e retorna uma resposta com os seguintes cabeçalhos:

```
x-amz-lex-dialog-state:ElicitSlot
x-amz-lex-input-transcript:tomorrow
x-amz-lex-intent-name:OrderFlowers
x-amz-lex-message:When do you want to pick up the roses on 2017-03-18?
x-amz-lex-session-attributes:e30=
x-amz-lex-slot-to-elicite:PickupTime
x-amz-lex-
slots:eyJQaWNrdXBuaW1lIjpudWxsLCJGbg93ZXJueXB1Ijoicm9zaSdzIiwUGlja3VwRGF0ZSI6IjIwMTctMDMtMTgif
x-amzn-RequestId:3a205b70-0b69-11e7-b447-eb69face3e6f
```

Os valores do cabeçalho fornecem as seguintes informações:

- `x-amz-lex-slots` – a versão codificada em base64 dos slots e dos valores:

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

- `x-amz-lex-session-attributes` – a versão codificada em base64 dos atributos da sessão (`{}`)

O cliente reproduz o áudio no corpo da resposta.



4. O usuário diz: 18h

- a. O cliente (console) envia a seguinte solicitação [PostContent \(p. 347\)](#) ao Amazon Lex:

```
POST /bot/OrderFlowers/alias/$LATEST/user/4o9wwd hx6nlheferh6a73fuj d3118f5w/content
HTTP/1.1
x-amz-lex-session-attributes: "e30="
Content-Type: "text/plain; charset=utf-8"
Accept: "audio/mpeg"

Request body

```

O corpo da solicitação é o stream de áudio de entrada do usuário ("18h"). O `sessionAttributes` permanece vazio.

- b. O Amazon Lex interpreta o fluxo de entrada no contexto da intenção atual (ele lembra que pediu ao usuário informações a respeito do slot `PickupTime`). Primeiro, ele atualiza o valor do slot para a intenção atual.

Agora, o Amazon Lex detecta que tem informações de todos os slots. No entanto, a intenção `OrderFlowers` é configurada com uma mensagem de confirmação. Portanto, o Amazon Lex precisa de uma confirmação explícita do usuário para poder prosseguir para o atendimento da intenção. Ele envia uma resposta com os seguintes cabeçalhos solicitando a confirmação antes de encomendar as flores:

```
x-amz-lex-dialog-state:ConfirmIntent
x-amz-lex-input-transcript:six p. m.
x-amz-lex-intent-name:OrderFlowers
x-amz-lex-message:Okay, your roses will be ready for pickup by 18:00 on 2017-03-18.
  Does this sound okay?
x-amz-lex-session-attributes:e30=
x-amz-lex-
slots:eyJQaWNrdXBuaW11IjoimTg6MDAiLCJGbg93ZXJUeXB1Ijoicm9zaSdzIiwUGlja3VwRGF0ZSI6IjIwMTctMDMtM
x-amzn-RequestId:083ca360-0b6a-11e7-b447-eb69face3e6f
```

Os valores do cabeçalho fornecem as seguintes informações:

- `x-amz-lex-slots` – a versão codificada em base64 dos slots e dos valores:

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

- `x-amz-lex-session-attributes` – a versão codificada em base64 dos atributos da sessão (`{}`)

O cliente reproduz o áudio no corpo da resposta.

5. O usuário diz: Sim

- a. O cliente (console) envia a seguinte solicitação [PostContent \(p. 347\)](#) ao Amazon Lex:

```
POST /bot/OrderFlowers/alias/$LATEST/user/4o9wwd hx6nlheferh6a73fuj d3118f5w/content
HTTP/1.1
x-amz-lex-session-attributes: "e30="
Content-Type: "audio/x-l16; sample-rate=16000; channel-count=1"
Accept: "audio/mpeg"

Request body
```

```
input stream ("Yes")
```

O corpo da solicitação é o stream de áudio de entrada do usuário ("Sim"). O `sessionAttributes` permanece vazio.

- b. O Amazon Lex interpreta o fluxo de entrada e entende que o usuário deseja prosseguir com o pedido. A intenção `OrderFlowers` é configurada com `ReturnIntent` como a atividade de cumprimento. Isso direciona o Amazon Lex a retornar todos os dados da intenção ao cliente. O Amazon Lex retorna uma mensagem com o seguinte:

```
x-amz-lex-dialog-state:ReadyForFulfillment
x-amz-lex-input-transcript:yes
x-amz-lex-intent-name:OrderFlowers
x-amz-lex-session-attributes:e30=
x-amz-lex-
slots:eyJQaWNrdXBuaW1lIjoimTg6MDAiLCJGbg93ZXJueXB1Ijoicm9zaSdzIiwUGlja3VwRGFOZSI6IjIwMTctMDMtM
```

O cabeçalho de resposta `x-amz-lex-dialog-state` é definido como `ReadyForFulfillment`. O cliente pode, então, cumprir a intenção.

6. Agora, teste novamente o bot. Para estabelecer um novo contexto (usuário), escolha o link `Clear` no console. Forneça os dados para a intenção `OrderFlowers` e inclua alguns dados inválidos. Por exemplo:
  - Jasmim como o tipo de flor (não é um dos tipos de flor suportados)
  - Ontem como o dia em que você deseja receber as flores

Observe que o bot aceita esses valores, pois você não tem nenhum código para inicializar e validar os dados do usuário. Na próxima seção, você adicionará uma função do Lambda para fazer isso. Observe o seguinte sobre a função Lambda:

- Ela valida dados de slot após cada entrada do usuário. Ela cumpre a intenção no final. Ou seja, o bot processa o pedido de flores e retorna uma mensagem para o usuário, em vez de simplesmente retornar dados do slot para o cliente. Para obter mais informações, consulte [Uso de funções do Lambda](#) (p. 106).
- Ela também define os atributos da sessão. Para obter mais informações sobre atributos de sessão, consulte [PostText](#) (p. 355).

Ao concluir a seção `Conceitos básicos`, você pode fazer os exercícios adicionais ([Exemplos adicionais: criação de bots do Amazon Lex](#) (p. 135)). O [Exemplo de bot: BookTrip](#) (p. 154) usa atributos de sessão para compartilhar informações entre intenções para iniciar uma conversa dinâmica com o usuário.

Próxima etapa

[Etapa 3: Criar uma função do Lambda \(console\)](#) (p. 50)

## Etapa 2b (opcional): Revise os detalhes do fluxo de informações digitado (console)

Esta seção explica o fluxo de informações entre o cliente e o Amazon Lex, em que o cliente usa a API `PostText` para enviar solicitações. Para obter mais informações, consulte [PostText](#) (p. 355).

1. O usuário digita: Eu gostaria de encomendar flores.
  - a. O cliente (console) envia a seguinte solicitação [PostText](#) (p. 355) ao Amazon Lex:

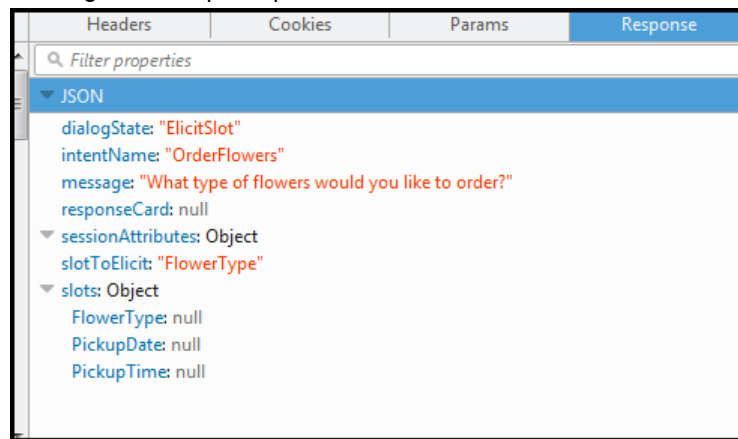
```
POST /bot/OrderFlowers/alias/$LATEST/user/4o9wwd hx6nlheferh6a73fuj d3118f5w/text
"Content-Type": "application/json"
"Content-Encoding": "amz-1.0"

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

O URI e o corpo da solicitação fornecem informações ao Amazon Lex:

- URI da solicitação – fornece o nome do bot (*OrderFlowers*), o alias do bot (*\$LATEST*) e o nome do usuário (uma string aleatória que identifica o usuário). O *text* final indica que esta é uma solicitação de API *PostText* (e não *PostContent*).
  - Corpo da solicitação – inclui a entrada do usuário (*inputText*) e *sessionAttributes* vazio. Quando o cliente faz a primeira solicitação, não há atributos de sessão. A função do Lambda os inicia posteriormente.
- b. No *inputText*, o Amazon Lex detecta a intenção (*OrderFlowers*). Essa intenção não tem nenhum gancho de código (ou seja, as funções do Lambda) para inicialização e validação de entrada do usuário ou de atendimento.

O Amazon Lex escolhe um dos slots (*FlowerType*) da intenção para obter o valor. Ele também seleciona uma das solicitações de escolha de valor para o slot (tudo parte da configuração de intenção) e, em seguida, envia a resposta a seguir de volta para o cliente. O console exibe a mensagem na resposta para o usuário.



O cliente exibe a mensagem na resposta.

2. O usuário digita: rosas
  - a. O cliente (console) envia a seguinte solicitação *PostText* (p. 355) para o Amazon Lex:

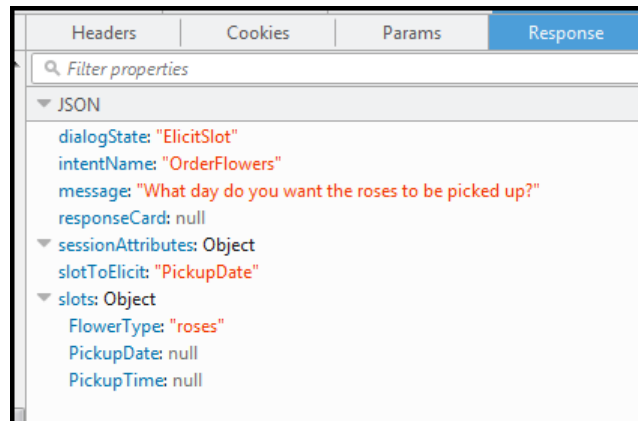
```
POST /bot/OrderFlowers/alias/$LATEST/user/4o9wwd hx6nlheferh6a73fuj d3118f5w/text
"Content-Type": "application/json"
"Content-Encoding": "amz-1.0"

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

O `inputText` no corpo da solicitação fornece entradas do usuário. O `sessionAttributes` permanece vazio.

- b. O Amazon Lex primeiro interpreta o `inputText` no contexto da intenção atual — o serviço lembra que pediu ao usuário específico informações sobre o slot `FlowerType`. O Amazon Lex primeiro atualiza o valor do slot para a intenção atual e escolhe outro slot (`PickupDate`) junto com uma de suas mensagens de solicitação — Em que dia você deseja retirar as rosas? — para o slot.

Em seguida, o Amazon Lex retorna a seguinte resposta:



O cliente exibe a mensagem na resposta.

3. O usuário digita: amanhã
  - a. O cliente (console) envia a seguinte solicitação [PostText \(p. 355\)](#) ao Amazon Lex:

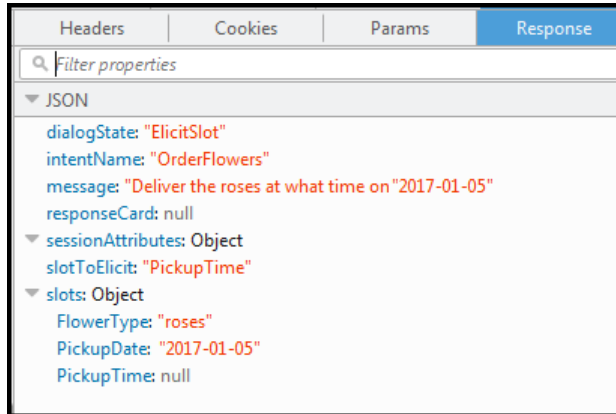
```
POST /bot/OrderFlowers/alias/$LATEST/user/4o9wwd hx6nlheferh6a73fuj d3118f5w/text
"Content-Type": "application/json"
"Content-Encoding": "amz-1.0"

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

O `inputText` no corpo da solicitação fornece entradas do usuário. O `sessionAttributes` permanece vazio.

- b. O Amazon Lex primeiro interpreta o `inputText` no contexto da intenção atual — o serviço lembra que pediu ao usuário específico informações sobre o slot `PickupDate`. O Amazon Lex atualiza o valor do slot (`PickupDate`) da intenção atual. Ele escolhe outro slot para obter o valor para (`PickupTime`). Ele retorna uma das solicitações de escolha de valor (Entregar as rosas a que horas em 01/05/2017?) para o cliente.

Em seguida, o Amazon Lex retorna a seguinte resposta:



O cliente exibe a mensagem na resposta.

4. O usuário digita: 18h

a. O cliente (console) envia a seguinte solicitação `PostText` (p. 355) ao Amazon Lex:

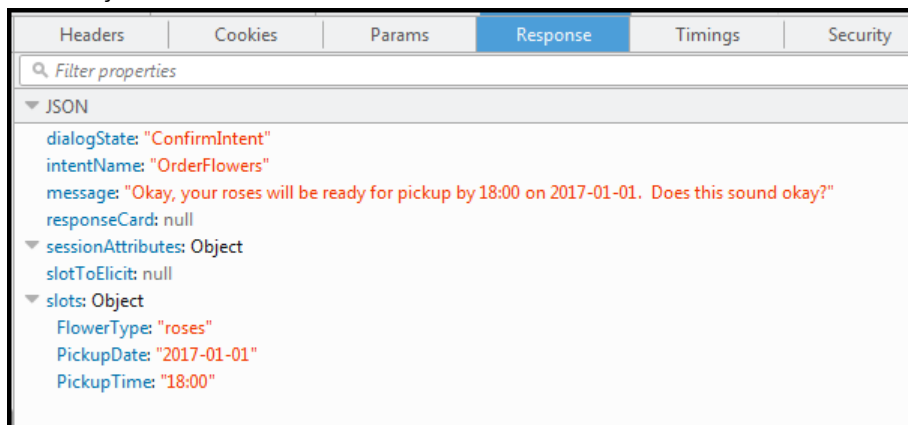
```
POST /bot/OrderFlowers/alias/$LATEST/user/4o9wwdxx6nlheferh6a73fujd3118f5w/text
"Content-Type": "application/json"
"Content-Encoding": "amz-1.0"

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

O `inputText` no corpo da solicitação fornece entradas do usuário. O `sessionAttributes` permanece vazio.

b. O Amazon Lex primeiro interpreta o `inputText` no contexto da intenção atual — o serviço lembra que pediu ao usuário específico informações sobre o slot `PickupTime`. O Amazon Lex primeiro atualiza o valor do slot da intenção atual. Agora, o Amazon Lex detecta que tem informações de todos os slots.

A intenção `OrderFlowers` é configurada com uma mensagem de confirmação. Portanto, o Amazon Lex precisa de uma confirmação explícita do usuário para poder prosseguir para o atendimento da intenção. O Amazon Lex envia a seguinte mensagem ao cliente solicitando confirmação antes de encomendar as flores:



O cliente exibe a mensagem na resposta.

5. O usuário digita: Sim

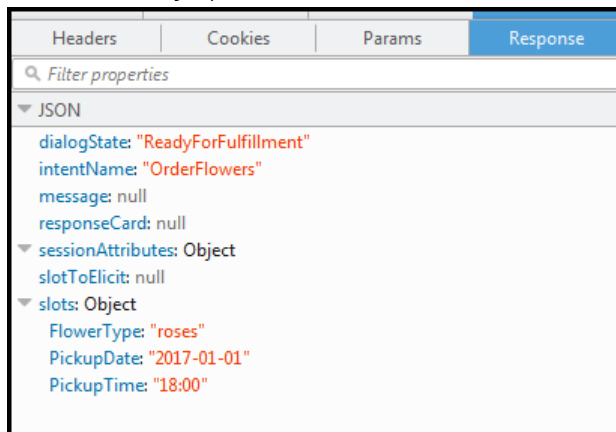
- a. O cliente (console) envia a seguinte solicitação [PostText \(p. 355\)](#) ao Amazon Lex:

```
POST /bot/OrderFlowers/alias/$LATEST/user/4o9wwdxx6nlheferh6a73fujd3118f5w/text
"Content-Type": "application/json"
"Content-Encoding": "amz-1.0"

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

O `inputText` no corpo da solicitação fornece entradas do usuário. O `sessionAttributes` permanece vazio.

- b. O Amazon Lex interpreta o `inputText` no contexto da confirmação da intenção atual. Ele entende que o usuário deseja prosseguir com o pedido. A intenção `OrderFlowers` é configurada com `ReturnIntent` como a atividade de atendimento (não há nenhuma função do Lambda para atender a intenção). Portanto, o Amazon Lex retorna os dados do slot ao cliente.



O Amazon Lex define o `dialogState` como `ReadyForFulfillment`. O cliente pode, então, cumprir a intenção.

6. Agora teste o bot novamente. Para isso, selecione o link **Clear** no console para estabelecer um novo contexto (de usuário). Agora, conforme você fornece dados para a intenção de encomenda de flores, tente fornecer dados inválidos. Por exemplo:

- Jasmim como o tipo de flor (não é um dos tipos de flor suportados).
- Ontem como o dia em que você deseja receber as flores.

Observe que o bot aceita esses valores, pois você não tem nenhum código para inicializar/validar os dados do usuário. Na próxima seção, você adicionará uma função do Lambda para fazer isso. Observe o seguinte sobre a função Lambda:

- A função do Lambda valida os dados do slot depois de cada entrada do usuário. Ela cumpre a intenção no final. Ou seja, o bot processa o pedido de flores e retorna uma mensagem para o usuário, em vez de simplesmente retornar dados de slot para o cliente. Para obter mais informações, consulte [Uso de funções do Lambda \(p. 106\)](#).
- A função do Lambda também define os atributos da sessão. Para obter mais informações sobre atributos de sessão, consulte [PostText \(p. 355\)](#).

Ao concluir a seção Conceitos básicos, você pode fazer os exercícios adicionais ([Exemplos adicionais: criação de bots do Amazon Lex \(p. 135\)](#)). O [Exemplo de bot: BookTrip \(p. 154\)](#) usa atributos de sessão para compartilhar informações entre intenções para iniciar uma conversa dinâmica com o usuário.

Próxima etapa

[Etapa 3: Criar uma função do Lambda \(console\) \(p. 50\)](#)

## Etapa 3: Criar uma função do Lambda (console)

Crie uma função do Lambda (usando o esquema `lex-order-flowers-python`) e execute a invocação de teste usando dados do evento de exemplo no console do AWS Lambda.

Você retorna ao console do Amazon Lex e adiciona a função do Lambda como o gancho de código para atender à intenção `OrderFlowers` no `OrderFlowersBot` que criou na seção anterior.

Para criar a função do Lambda (console)

1. Faça login no Console de gerenciamento da AWS e abra o console AWS Lambda em <https://console.aws.amazon.com/lambda/>.
2. Selecione **Create function** (Criar função).
3. Na página **Create function** (Criar função), selecione **Usar um blueprint** (Usar um esquema). Digite **lex-** na caixa de texto do filtro e pressione **Enter** para localizar o esquema e selecione o esquema `lex-order-flowers-python`.

Os esquemas da função do Lambda são fornecidos em Node.js e em Python. Para este exercício, use o esquema baseado em Python.

4. Na página **Basic information** (Informações básicas), faça o seguinte:
  - Digite um nome de função do Lambda (`OrderFlowersCodeHook`).
  - Para a função de execução, selecione **Create a new role with basic Lambda permissions** (Criar uma função com permissões básicas do Lambda).
  - Deixe os outros valores padrão.
5. Selecione **Create function** (Criar função).
6. Testar a função do Lambda.
  - a. Selecione **Select a test event** (Selecionar um evento de teste), **Configure test events** (Configurar eventos de teste).
  - b. Selecione **Amazon Lex Order Flowers** (Flores do pedido do Amazon Lex) da lista **Event template** (Modelo do evento). Este evento de exemplo corresponde ao modelo de solicitação/resposta do Amazon Lex (consulte [Uso de funções do Lambda \(p. 106\)](#)). Dê um nome ao evento de teste (`LexOrderFlowersTest`).
  - c. Escolha **Criar**.
  - d. Selecione **Test** (Testar) para testar o gancho de código.
  - e. Verifique se a função do Lambda foi executada com êxito. A resposta, neste caso, corresponde ao modelo de resposta do Amazon Lex.

Próxima etapa

[Etapa 4: Adicionar a função do Lambda como gancho de código \(console\) \(p. 51\)](#)

## Etapa 4: Adicionar a função do Lambda como gancho de código (console)

Nesta seção, você atualiza a configuração da intenção OrderFlowers para usar a função do Lambda da seguinte forma:

- Primeiro, use a função do Lambda como um gancho de código para executar o atendimento da intenção OrderFlowers. Você pode testar o bot e verificar se recebeu uma mensagem de atendimento da função do Lambda. O Amazon Lex invoca a função do Lambda somente depois que você fornece os dados de todos os slots necessários para solicitar flores.
- Configure a mesma função do Lambda como um gancho de código para executar a inicialização e a validação. Teste e verifique se a função do Lambda executa a validação (conforme você fornece dados do slot).

Para adicionar a função do Lambda; como um gancho de código (console)

1. No console do Amazon Lex, selecione o bot OrderFlowers. O console mostra a intenção OrderFlowers. Verifique se a versão da intenção está definida como \$LATEST, pois essa é a única versão que podemos modificar.
2. Adicione a função do Lambda como o gancho de código de atendimento e teste-a.
  - a. No Editor, escolha AWS Lambda function (Função do Lambda) como Fulfillment (Atendimento) e selecione a função do Lambda que você criou na etapa anterior (OrderFlowersCodeHook). Escolha OK para conceder permissão ao Amazon Lex para invocar a função do Lambda.

Você está configurando essa função do Lambda como um gancho de código para atender à intenção. O Amazon Lex invoca essa função apenas após ter todos os dados de slot necessários do usuário para atender à intenção.

- b. Especifique uma Goodbye message.
- c. Escolha Build.
- d. Teste o bot usando a conversa anterior.

A última declaração "Thanks, your order for roses..... (Obrigado, seu pedido de rosas...)" é uma resposta da função do Lambda que você configurou como um gancho de código. Na seção anterior, não havia nenhuma função do Lambda. Agora, você está usando uma função do Lambda para realmente atender à intenção OrderFlowers.

3. Adicione a função do Lambda como um gancho de código de inicialização e validação e teste.

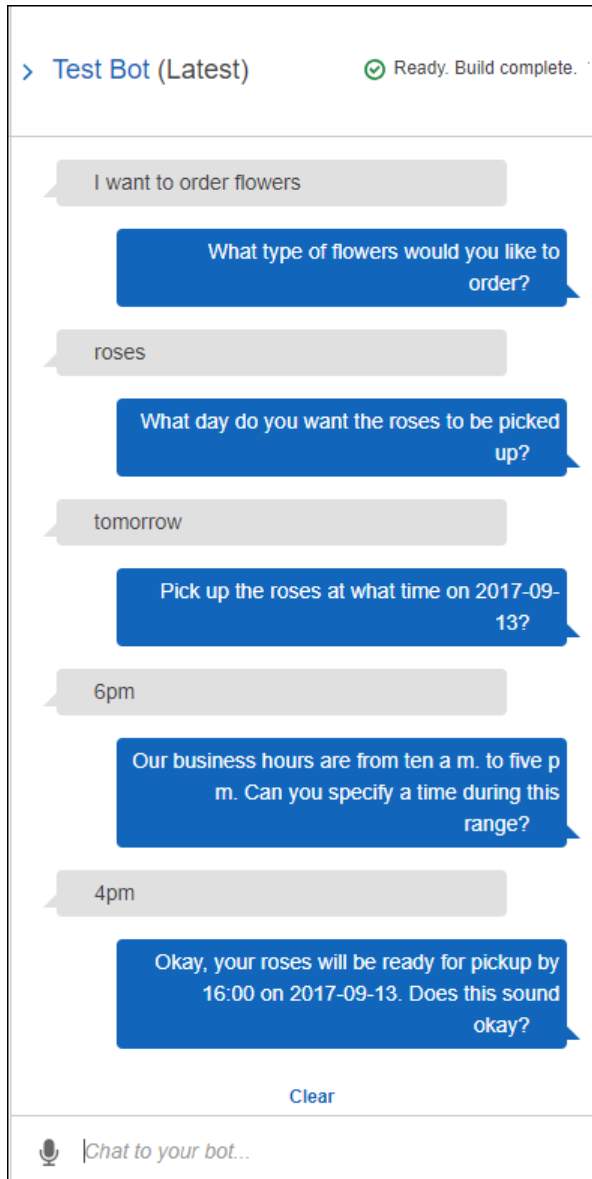
O exemplo de código da função do Lambda que você está usando pode executar a validação e o atendimento da entrada do usuário. O evento de entrada que a função do Lambda recebe tem um campo (invocationSource) que o código usa para determinar qual parte do código executar. Para obter mais informações, consulte [Evento de entrada de função do Lambda e formato de resposta \(p. 106\)](#).

- a. Selecione a versão \$LATEST da intenção OrderFlowers. Essa é a única versão que você pode atualizar.
- b. No Editor, escolha Initialization and validation em Options.
- c. Novamente, selecione a mesma função do Lambda.
- d. Escolha Build.
- e. Teste o bot.

Agora você está pronto para conversar com o Amazon Lex do seguinte modo. Para testar a parte da validação, escolha 18h, e a função do Lambda retornará uma resposta ("Our business



hours are from 10 AM to 5 PM. (Nosso horário de funcionamento é das 10h às 17h.")) e enviará a solicitação a você novamente. Depois que você fornecer todos os dados de slot válidos, a função do Lambda atenderá à ordem.



Próxima etapa

[Etapa 5 \(opcional\): Revise os detalhes do fluxo de informações \(console\) \(p. 52\)](#)

## Etapa 5 (opcional): Revise os detalhes do fluxo de informações (console)

Esta seção explica o fluxo de informações entre o cliente e Amazon Lex para cada entrada do usuário, incluindo a integração da função do Lambda.

## Note

A seção supõe que o cliente envia solicitações ao Amazon Lex usando a API de tempo de execução `PostText` e mostra os detalhes da solicitação e da resposta adequadamente. Para obter um exemplo do fluxo de informações entre o cliente e o Amazon Lex no qual o cliente usa a API `PostContent`, consulte [Etapa 2a \(opcional\): Revise os detalhes do fluxo de informações falado \(console\)](#) (p. 41).

Para obter mais informações sobre a API de runtime `PostText` e detalhes adicionais sobre as solicitações e respostas mostradas nas etapas a seguir, consulte [PostText](#) (p. 355).

1. Usuário: "Eu gostaria de encomendar flores."
  - a. O cliente (console) envia a seguinte solicitação [PostText](#) (p. 355) ao Amazon Lex:

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

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

O URI e o corpo da solicitação fornecem informações ao Amazon Lex:

- URI da solicitação – fornece o nome do bot (*OrderFlowers*), o alias do bot (*\$LATEST*) e o nome do usuário (uma string aleatória que identifica o usuário). O `text` final indica que esta é uma solicitação de API `PostText` (e não `PostContent`).
  - Corpo da solicitação – inclui a entrada do usuário (`inputText`) e `sessionAttributes` vazio. Quando o cliente faz a primeira solicitação, não há atributos de sessão. A função do Lambda as inicia posteriormente.
- b. No `inputText`, o Amazon Lex detecta a intenção (*OrderFlowers*). Essa intenção é configurada com uma função do Lambda como um gancho de código para inicialização e validação de dados do usuário. Portanto, o Amazon Lex invoca essa função do Lambda especificando as seguintes informações como dados do evento:

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

Para obter mais informações, consulte [Formato de eventos de entrada](#) (p. 106).

Além das informações enviadas pelo cliente, o Amazon Lex também inclui os seguintes dados adicionais:

- `messageVersion` – atualmente, o Amazon Lex oferece suporte apenas à versão 1.0.
  - `invocationSource` – indica o objetivo da chamada da função Lambda. Nesse caso, é para executar os dados de inicialização e validação do usuário. Neste ponto, o Amazon Lex sabe que o usuário não forneceu todos os dados do slot para atender à intenção.
  - Informações `currentIntent` com todos os valores de slot definidos como nulo.
- c. No momento, todos os valores de slot são nulos. Não há nada para a função do Lambda validar. A função do Lambda retorna a seguinte resposta ao Amazon Lex:

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

Para obter mais informações sobre o formato de resposta, consulte [Formato de resposta \(p. 109\)](#).

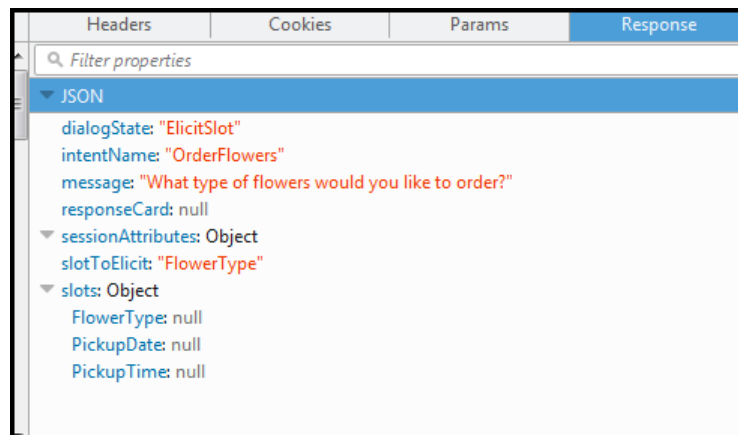
Observe o seguinte:

- `dialogAction.type` – definindo esse valor como `Delegate`, a função do Lambda delega a responsabilidade de decidir a próxima ação para o Amazon Lex.

#### Note

Se a função do Lambda detectar qualquer coisa na validação dos dados do usuário, ela instruirá o Amazon Lex sobre o que fazer em seguida, conforme mostrado nas próximas etapas.

- d. De acordo com o `dialogAction.type`, o Amazon Lex decide a próxima ação. Como nenhum dos slots estão preenchidos, ele decide escolher o valor para o slot `FlowerType`. Ele seleciona uma das solicitações de escolha de valor ("What type of flowers would you like to order? (Que tipo de flores você deseja encomendar?)") para este slot e envia a resposta a seguir de volta ao cliente:



O cliente exibe a mensagem na resposta.

2. Usuário: rosas

- a. O cliente envia a seguinte solicitação [PostText](#) (p. 355) ao Amazon Lex:

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

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

No corpo da solicitação, o `inputText` fornece entradas do usuário. O `sessionAttributes` permanece vazio.

- b. O Amazon Lex primeiro interpreta o `inputText` no contexto da intenção atual. O serviço lembra que havia solicitado ao usuário específico informações sobre o slot `FlowerType`. Ele atualiza o valor do slot na intenção atual e chama a função do Lambda com os seguintes dados do evento:

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

Observe o seguinte:

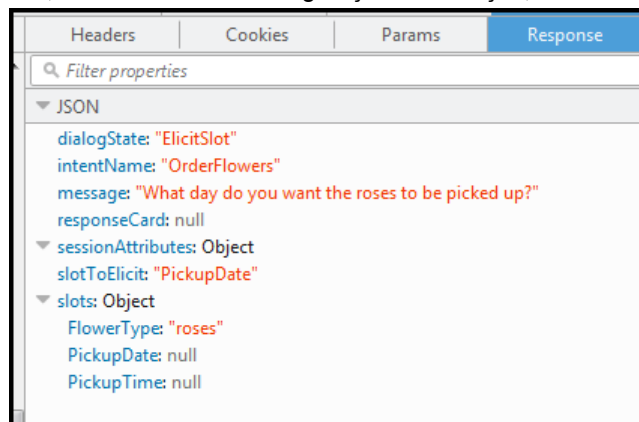
- `invocationSource` – continua sendo `DialogCodeHook` (estamos simplesmente validando os dados do usuário).
  - `currentIntent.slots` – o Amazon Lex atualizou o slot `FlowerType` para rosas.
- c. De acordo com o valor `invocationSource` de `DialogCodeHook`, a função do Lambda executa a validação dos dados do usuário. Ele reconhece `roses` como um valor de slot válido (e define `Price` como um atributo de sessão) e retorna a seguinte resposta ao Amazon Lex.

```
{
  "sessionAttributes": {
    "Price": 25
  },
  "dialogAction": {
    "type": "Delegate",
    "slots": {
```

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

Observe o seguinte:

- `sessionAttributes` – a função do Lambda adicionou `Price` (das rosas) como um atributo de sessão.
  - `dialogAction.type` – está definido como `Delegate`. Os dados do usuário eram válidos e a função do Lambda direciona o Amazon Lex a escolher a próxima ação.
- d. De acordo com o `dialogAction.type`, o Amazon Lex escolhe a próxima ação. O Amazon Lex sabe que precisa de mais dados de slot, portanto, escolhe o próximo slot não preenchido (`PickupDate`) com a prioridade mais alta de acordo com a configuração da intenção. O Amazon Lex seleciona uma das mensagens de solicitação de escolha de valor — "What day do you want the roses to be picked up? (Em que dia você deseja que as rosas sejam retiradas?)" — para esse slot, de acordo com a configuração da intenção, e envia a seguinte resposta de volta ao cliente:



O cliente simplesmente exibe a mensagem na resposta – "What day do you want the roses to be picked up? (Em que dia você deseja que as rosas sejam retiradas?)."

### 3. Usuário: amanhã

- a. O cliente envia a seguinte solicitação [PostText \(p. 355\)](#) ao Amazon Lex:

```
POST /bot/OrderFlowers/alias/$LATEST/user/ignw84y6seypre4xly5rimopuri2xwnd/text  
"Content-Type": "application/json"  
"Content-Encoding": "amz-1.0"  
  
{  
  "inputText": "tomorrow",  
  "sessionAttributes": {  
    "Price": "25"  
  }  
}
```

No corpo da solicitação, `inputText` fornece entradas do usuário e o cliente passa os atributos de sessão de volta ao serviço.

- b. O Amazon Lex lembra-se do contexto — para o qual estava escolhendo dados para o slot `PickupDate`. Nesse contexto, ele sabe que o valor de `inputText` é para o slot `PickupDate`. Em seguida, o Amazon Lex invoca a função do Lambda enviando o seguinte evento:

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

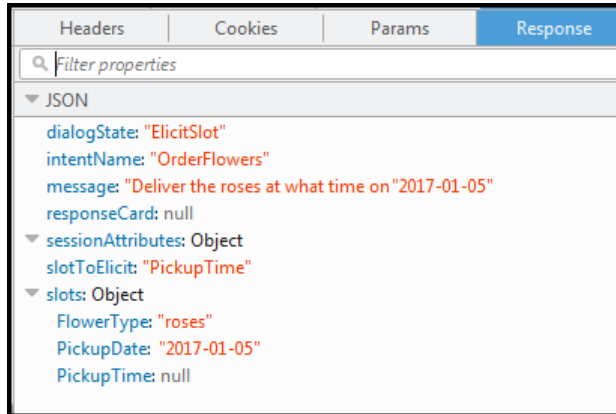
O Amazon Lex atualizou o `currentIntent.slots` definindo o valor de `PickupDate`. Observe também que o serviço passa o `sessionAttributes` como está para a função do Lambda.

- c. De acordo com o valor de `invocationSource` de `DialogCodeHook`, a função do Lambda executa a validação dos dados do usuário. Ela reconhece que o valor do slot `PickupDate` é válido e retorna a seguinte resposta ao Amazon Lex:

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

Observe o seguinte:

- `sessionAttributes` – sem alterações.
  - `dialogAction.type` – está definido como `Delegate`. Os dados do usuário eram válidos, e a função do Lambda direciona o Amazon Lex a escolher a próxima ação.
- d. De acordo com o `dialogAction.type`, o Amazon Lex escolhe a próxima ação. O Amazon Lex sabe que precisa de mais dados de slot, portanto, escolhe o próximo slot não preenchido (`PickupTime`) com a prioridade mais alta de acordo com a configuração da intenção. O Amazon Lex seleciona uma das mensagens de solicitação "Deliver the roses at what time on 2017-01-05? (Entregar as rosas em que horário no dia 01/01/2017?)" para esse slot, de acordo com a configuração da intenção, e envia a seguinte resposta de volta ao cliente:



O cliente exibe a mensagem na resposta – "Deliver the roses at what time on 2017-01-05?" (Entregar as rosas em que horário no dia 01/01/2017?)

4. Usuário: 16h

- a. O cliente envia a seguinte solicitação [PostText](#) (p. 355) ao Amazon Lex:

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

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

No corpo da solicitação, `inputText` fornece entradas do usuário. O cliente passa o `sessionAttributes` na solicitação.

- b. O Amazon Lex compreende o contexto. Ele entende que estava escolhendo dados para o slot `PickupTime`. Nesse contexto, ele sabe que o valor de `inputText` é para o slot `PickupTime`. Em seguida, o Amazon Lex invoca a função do Lambda enviando o seguinte evento:

```
{
  "messageVersion": "1.0",
  "invocationSource": "DialogCodeHook",
  "userId": "ignw84y6seypre4xly5rimopuri2xwnd",
  "sessionAttributes": {
    "Price": "25"
  },
  "bot": {
    "name": "OrderFlowersCustomWithRespCard",
    "alias": null,
    "version": "$LATEST"
  },
  "outputDialogMode": "Text",
  "currentIntent": {
    "name": "OrderFlowers",
    "slots": {
      "PickupTime": "16:00",
      "FlowerType": "roses",
      "PickupDate": "2017-01-05"
    }
  },
  "confirmationStatus": "None"
}
```

```
}  
}
```

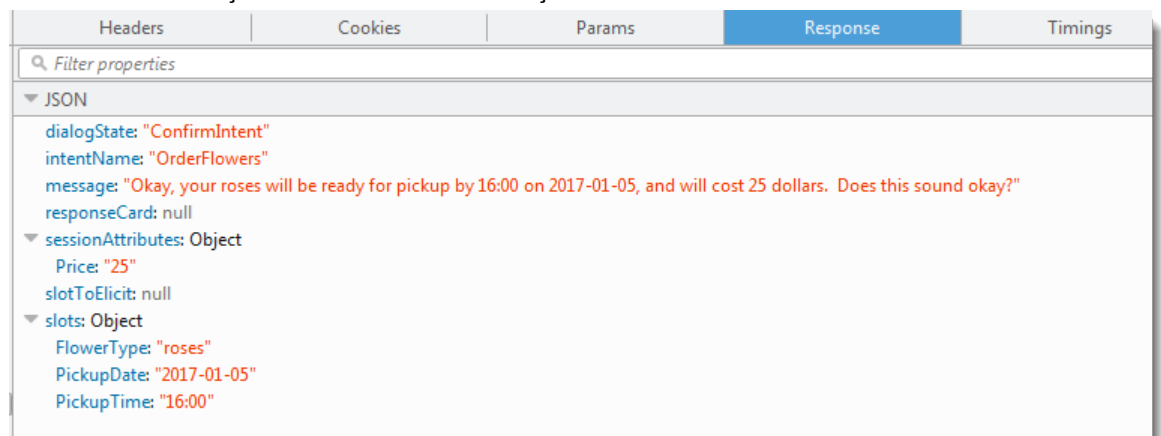
O Amazon Lex atualizou o `currentIntent.slots` definindo o valor de `PickupTime`.

- c. De acordo com o valor `invocationSource` de `DialogCodeHook`, a função do Lambda executa a validação dos dados do usuário. Ela reconhece que o valor do slot `PickupDate` é válido e retorna a seguinte resposta ao Amazon Lex.

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

Observe o seguinte:

- `sessionAttributes` – nenhuma alteração no atributo da sessão.
  - `dialogAction.type` – está definido como `Delegate`. Os dados do usuário eram válidos e a função do Lambda direciona o Amazon Lex a escolher a próxima ação.
- d. Neste ponto, o Amazon Lex sabe que tem todos os dados do slot. Essa intenção é configurada com um prompt de confirmação. Portanto, o Amazon Lex envia a seguinte resposta ao usuário solicitando confirmação antes de atender a intenção:



O cliente simplesmente exibe a mensagem na resposta e aguarda a resposta do usuário.

5. Usuário: Sim

- a. O cliente envia a seguinte solicitação [PostText](#) (p. 355) ao Amazon Lex:

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



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

- b. O Amazon Lex interpreta o `inputText` no contexto da confirmação da intenção atual. O Amazon Lex entende que o usuário deseja prosseguir com o pedido. Desta vez, o Amazon Lex chama a função do Lambda para atender à intenção enviando o seguinte evento, que define o `invocationSource` como `FulfillmentCodeHook` no evento que envia à função do Lambda. O Amazon Lex também define `confirmationStatus` como `Confirmed`.

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

Observe o seguinte:

- `invocationSource` – desta vez, o Amazon Lex define esse valor como `FulfillmentCodeHook`, direcionando a função do Lambda a atender à intenção.
  - `confirmationStatus` – está definido como `Confirmed`.
- c. Desta vez, a função do Lambda atende à intenção `OrderFlowers` e retorna a seguinte resposta:

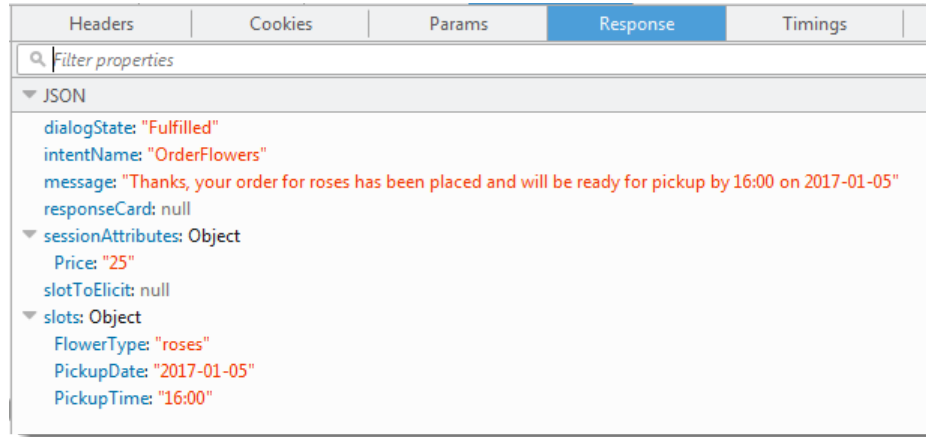
```
{  
  "sessionAttributes": {  
    "Price": "25"  
  },  
  "dialogAction": {  
    "type": "Close",  
    "fulfillmentState": "Fulfilled",  
    "message": {  
      "contentType": "PlainText",  
      "content": "Thanks, your order for roses has been placed and will be  
ready for pickup by 16:00 on 2017-01-05"  
    }  
  }  
}
```

Observe o seguinte:

- Define o `dialogAction.type` – a função do Lambda define esse valor como `Close` direcionando Amazon Lex para não esperar uma resposta do usuário.

- `dialogAction.fulfillmentState` – é definido como `Atendido` e inclui uma `message` adequada para transmitir para o usuário.
- d. O Amazon Lex analisa o `fulfillmentState` e envia a resposta a seguir de volta ao cliente:

Em seguida, o Amazon Lex retorna o seguinte ao cliente:



Observe que:

- `dialogState` – o Amazon Lex define esse valor como `fulfilled`.
- `message` – é a mesma mensagem que a função do Lambda forneceu.

O cliente exibe a mensagem.

6. Agora teste o bot novamente. Para estabelecer um novo contexto (de usuário), selecione o link `Clear` na janela de teste. Agora, forneça dados de slot inválidos para a intenção `OrderFlowers`. Desta vez, a função do Lambda executa a validação dos dados, redefine os valores dos dados do slot inválidos como nulos e pede ao Amazon Lex para solicitar dados válidos ao usuário. Por exemplo, tente o seguinte:
- `Jasmim` como o tipo de flor (não é um dos tipos de flor suportados).
  - `Ontem` como o dia em que você deseja receber as flores.
  - Após fazer o pedido, digite outro tipo de flor em vez de responder `"sim"` para confirmar o pedido. Em resposta, a função do Lambda atualiza o `Price` no atributo da sessão mantendo um total cumulativo de encomendas de flores.

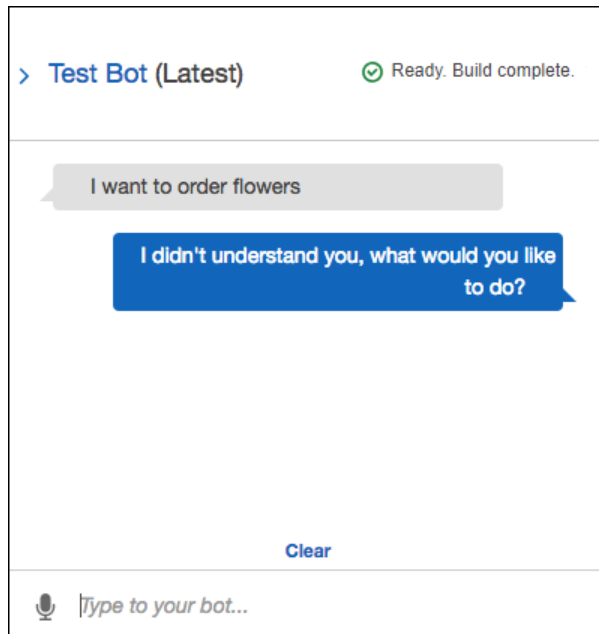
A função do Lambda também executa o atendimento da atividade.

Próxima etapa

[Etapa 6: Atualize a configuração de intenção para adicionar um enunciado \(console\) \(p. 61\)](#)

## Etapa 6: Atualize a configuração de intenção para adicionar um enunciado (console)

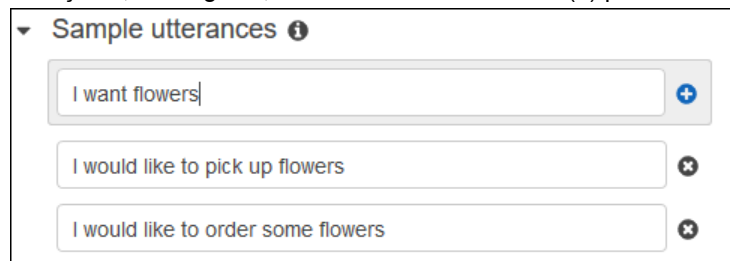
O bot `OrderFlowers` é configurado com apenas dois enunciados. Isso fornece informações limitadas ao Amazon Lex para criar um modelo de Machine Learning que reconhece e responde à intenção do usuário. Tente digitar `"I want to order flowers (Desejo encomendar flores)"` na janela de teste. O Amazon Lex não reconhece o texto e responde com `"I didn't understand you, what would you like to do? (Não entendi, o que você deseja fazer?)"` Você pode melhorar o modelo de Machine Learning adicionando mais enunciados.



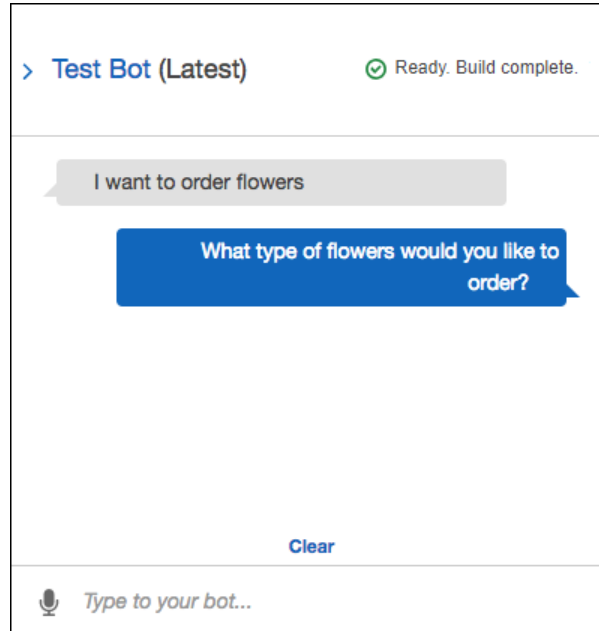
Cada enunciado que você adiciona fornece ao Amazon Lex mais informações sobre como responder aos usuários. Você não precisa adicionar um enunciado exato. O Amazon Lex generaliza a partir dos exemplos que você fornece para reconhecer correspondências exatas e entradas semelhantes.

Para adicionar um enunciado (console)

1. Adicione o enunciado "Quero flores" à intenção digitando-o na seção Sample utterances do editor de intenção e, em seguida, clicando no ícone de mais (+) próximo ao novo enunciado.



2. Crie seu bot para incorporar a alteração. Escolha Build e, em seguida, escolha Build novamente.
3. Teste o bot para confirmar se ele reconheceu o novo enunciado. Na janela de teste, digite "I want to order flowers. (Desejo encomendar flores.)" O Amazon Lex não reconhece a frase e responde com "What type of flowers would you like to order? (Que tipo de flores você deseja encomendar?)".



Próxima etapa

[Etapa 7 \(opcional\): Limpe \(console\) \(p. 63\)](#)

## Etapa 7 (opcional): Limpe (console)

Agora, exclua os recursos que você criou e limpe sua conta.

Você só pode excluir os recursos que não estão em uso. Em geral, você deve excluir recursos na seguinte ordem:

- Exclua bots para liberar recursos de intenção.
- Exclua intenções para liberar recursos de tipo de slot.
- Exclua tipos de slot por último.

Para limpar sua conta (console)

1. Faça login no Console de gerenciamento da AWS e abra o console do Amazon Lex em <https://console.aws.amazon.com/lex/>.
2. Na lista de bots, marque a caixa de seleção próxima a OrderFlowers.
3. Para excluir o bot, escolha Delete e, em seguida, escolha Continue na caixa de diálogo de confirmação.
4. No painel à esquerda, selecione Intents.
5. Na lista de intenções, escolha OrderFlowersIntent.
6. Para excluir a intenção, selecione Delete e, em seguida, escolha Continue na caixa de diálogo de confirmação.
7. No painel esquerdo, escolha Slot types.
8. Na lista de tipos de slot, escolha Flowers.
9. Para excluir o tipo de slot, selecione Delete e, em seguida, escolha Continue na caixa de diálogo de confirmação.

Você removeu todos os recursos do Amazon Lex que criou e limpou sua conta. Se desejar, você pode usar o [console do Lambda](#) para excluir a função Lambda usada neste exercício.

## Exercício 2: Criar um bot personalizado do Amazon Lex

Neste exercício, você usa o console do Amazon Lex para criar um bot personalizado para encomendar pizza (OrderPizzaBot). Você configura o bot adicionando uma intenção personalizada (OrderPizza), definindo tipos de slot personalizados e definindo os slots necessários para um pedido de pizza (massa da pizza, tamanho e assim por diante). Para obter mais informações sobre tipos de slot e slots, consulte [Amazon Lex: Como ele funciona](#) (p. 3).

### Tópicos

- [Etapa 1: Criar uma função do Lambda](#) (p. 64)
- [Etapa 2: crie um bot](#) (p. 66)
- [Etapa 3: crie e teste o bot](#) (p. 71)
- [Etapa 4 \(opcional\): Limpeza](#) (p. 74)

## Etapa 1: Criar uma função do Lambda

Primeiro, crie uma função do Lambda que atenda a um pedido de pizza. Essa função é específica ao bot do Amazon Lex que será criado na próxima seção.

Para criar uma função do Lambda

1. Faça login no Console de gerenciamento da AWS e abra o console AWS Lambda em <https://console.aws.amazon.com/lambda/>.
2. Escolha Create function.
3. Na página Create function, selecione Author from scratch.

Como você está usando um código personalizado fornecido neste exercício para criar uma função Lambda, selecione essa opção.

Faça o seguinte:

- a. Digite o nome: (PizzaOrderProcessor).
  - b. Em Runtime (Tempo de execução), selecione a versão mais recente do Node.js.
  - c. Em Role, selecione Create new role from template(s).
  - d. Insira um novo nome de função (PizzaOrderProcessorRole).
  - e. Escolha Create function.
4. Na página function, faça o seguinte:

Na seção Function code, selecione Edit code inline e, em seguida, copie o código da função Node.js a seguir e cole-o na janela.

```
'use strict';

// Close dialog with the customer, reporting fulfillmentState of Failed or Fulfilled
("Thanks, your pizza will arrive in 20 minutes")
function close(sessionAttributes, fulfillmentState, message) {
    return {
        sessionAttributes,
        dialogAction: {
```

```
        type: 'Close',
        fulfillmentState,
        message,
      },
    };
  }

// ----- Events -----

function dispatch(intentRequest, callback) {
  console.log(`request received for userId=${intentRequest.userId}, intentName=${intentRequest.currentIntent.name}`);
  const sessionAttributes = intentRequest.sessionAttributes;
  const slots = intentRequest.currentIntent.slots;
  const crust = slots.crust;
  const size = slots.size;
  const pizzaKind = slots.pizzaKind;

  callback(close(sessionAttributes, 'Fulfilled',
    {'contentType': 'PlainText', 'content': `Okay, I have ordered your ${size} ${pizzaKind} pizza on ${crust} crust`}));
}

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

// Route the incoming request based on intent.
// The JSON body of the request is provided in the event slot.
exports.handler = (event, context, callback) => {
  try {
    dispatch(event,
      (response) => {
        callback(null, response);
      });
  } catch (err) {
    callback(err);
  }
};
```

5. Escolha Salvar.

## Testar a função do Lambda usando dados de eventos de exemplo

No console, teste a função do Lambda usando os dados de eventos de exemplo para invocá-lo manualmente.

Para testar a função do Lambda.

1. Faça login no Console de gerenciamento da AWS e abra o console AWS Lambda em <https://console.aws.amazon.com/lambda/>.
2. Na página Lambda function (Função do Lambda), selecione a função do Lambda (PizzaOrderProcessor).
3. Na página da função, na lista de eventos de teste, selecione Configure test events.
4. Na página Configure test event, faça o seguinte:
  - a. Selecione Create new test event.
  - b. No campo Event name, insira um nome para o evento (PizzaOrderProcessorTest).
  - c. Copie o evento do Amazon Lex a seguir na janela.

```
{
```

```
"messageVersion": "1.0",
"invocationSource": "FulfillmentCodeHook",
"userId": "user-1",
"sessionAttributes": {},
"bot": {
  "name": "PizzaOrderingApp",
  "alias": "$LATEST",
  "version": "$LATEST"
},
"outputDialogMode": "Text",
"currentIntent": {
  "name": "OrderPizza",
  "slots": {
    "size": "large",
    "pizzaKind": "meat",
    "crust": "thin"
  },
  "confirmationStatus": "None"
}
}
```

#### 5. Escolha Criar.

O AWS Lambda cria o teste, e você retorna à página da função. Selecione Test (Testar), e o Lambda; executará a função do Lambda.

Na caixa de resultados, selecione Details. O console exibe a saída a seguir no painel Execution result.

```
{
  "sessionAttributes": {},
  "dialogAction": {
    "type": "Close",
    "fulfillmentState": "Fulfilled",
    "message": {
      "contentType": "PlainText",
      "content": "Okay, I have ordered your large meat pizza on thin crust."
    }
  }
}
```

## Próxima etapa

[Etapa 2: crie um bot \(p. 66\)](#)

## Etapa 2: crie um bot

Nesta etapa, você cria um bot para lidar com os pedidos de pizza.

### Tópicos

- [Criar o bot \(p. 66\)](#)
- [Criar uma intenção \(p. 67\)](#)
- [Criar tipos de slot \(p. 68\)](#)
- [Configurar a intenção \(p. 69\)](#)
- [Configurar o bot \(p. 70\)](#)

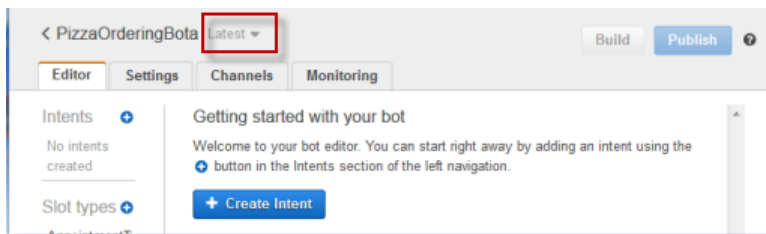
## Criar o bot

Crie o bot `PizzaOrderingBot` com o mínimo de informações necessárias. Você adiciona uma intenção, uma ação que o usuário deseja executar, para o bot mais tarde.

### Para criar o bot

1. Faça login no Console de gerenciamento da AWS e abra o console do Amazon Lex em <https://console.aws.amazon.com/lex/>.
2. Crie um bot.
  - a. Se você estiver criando seu primeiro bot, escolha Get Started. Caso contrário, escolha Bots e, em seguida, Create.
  - b. Na página Create your Lex bot escolha Custom bot e forneça as seguintes informações:
    - App name: PizzaOrderingBot
    - Output voice: Salli
    - Session timeout : 5 minutos.
    - Child-Directed: escolha a resposta apropriada.
  - c. Escolha Criar.

O console envia ao Amazon Lex uma solicitação para criar um novo bot. O Amazon Lex define a versão do bot como \$LATEST. Depois da criação do bot, o Amazon Lex mostra a guia Editor do bot:



- A versão Latest do bot encontra-se no console, ao lado do nome do bot. Os novos recursos do Amazon Lex têm a versão \$LATEST. Para obter mais informações, consulte [Controle de versão e aliases](#) (p. 102).
- Como você não criou nenhuma intenção ou tipos de slots, não há nenhum listado.
- Build e Publish são atividades no nível do bot. Depois de configurar todo o bot, você aprenderá mais sobre essas atividades.

### Próxima etapa

[Criar uma intenção](#) (p. 67)

### Criar uma intenção

Agora, crie a intenção `OrderPizza`, uma ação que o usuário deseja executar, com o mínimo de informações necessárias. Você adiciona tipos de slot para a intenção e, mais tarde, configura a intenção.

### Para criar uma intenção

1. No console do Amazon Lex; escolha o sinal adição (+) ao lado de Intents (Intenções) e, em seguida, escolha Create new intent (Criar nova intenção).
2. Na caixa de diálogo Create intent, digite o nome da intenção (`OrderPizza`) e escolha Add.

O console envia uma solicitação ao Amazon Lex para criar a intenção `OrderPizza`. Neste exemplo, você cria slots para a intenção depois de criar tipos de slots.



## Próxima etapa

[Criar tipos de slot \(p. 68\)](#)

## Criar tipos de slot

Crie tipos de slot ou valores de parâmetro que a intenção `OrderPizza` usa.

Para criar tipos de slot

1. No menu esquerdo, escolha o sinal de mais (+) ao lado de Slot types.
2. Na caixa de diálogo Add slot type, adicione o seguinte:
  - Slot type name (Nome do tipo de slot) – Massas
  - Description (Descrição) – Massas disponíveis
  - Escolha Restrict to Slot values and Synonyms
  - Value (Valor) – digite **thick**. Selecione a guia e, no campo Synonym (Sinônimo), digite **stuffed**. Escolha o sinal de adição (+). Digite **thin** e, em seguida, escolha o sinal de adição (+) novamente.

O diálogo deve ser semelhante ao seguinte:

The screenshot shows the 'Add slot type' dialog box. It has a title bar with a close button. The main content area includes:

- Slot type name:** A text input field containing 'Crusts'.
- Description:** A text input field containing 'Available crusts'.
- Slot Resolution:** Two radio buttons. 'Expand Values' is unselected, and 'Restrict to Slot values and Synonyms' is selected.
- Value:** A section with a list of values. Each value is in a text input field with a plus sign button to its right. The values are: 'e.g. Small', 'thick', 'thin', 'stuffed', and 'unstuffed'. Below the first value, there is a small text prompt 'Press Tab to add a synonym'.

At the bottom of the dialog are three buttons: 'Cancel', 'Save slot type', and 'Add slot to intent'.

3. Escolha Add slot to intent.
4. Na página Intent, escolha Required. Altere o nome do slot de **slotOne** para **crust**. Mude a solicitação para **What kind of crust would you like?**.
5. Repita [Step 1](#) a [Step 4](#) usando os valores da seguinte tabela:

| Nome  | Descrição            | Valores                 | Nome do slot | Solicitação            |
|-------|----------------------|-------------------------|--------------|------------------------|
| Sizes | Tamanhos disponíveis | pequena, média e grande | tamanho      | qual tamanho de pizza? |

| Nome      | Descrição          | Valores             | Nome do slot | Solicitação                              |
|-----------|--------------------|---------------------|--------------|--|
| PizzaKind | Pizzas disponíveis | vegetariana, queijo | pizzaKind    | Quer uma pizza de queijo ou vegetariana? |

### Próxima etapa

[Configurar a intenção \(p. 69\)](#)

## Configurar a intenção

Configure a intenção `OrderPizza` para atender à solicitação do usuário de pedir uma pizza.

Para configurar uma intenção

- Na página de configuração `OrderPizza`, configure a intenção da seguinte forma:
  - Sample utterances (Enunciados de exemplo) – digite as seguintes strings. As chaves `{}` contém nomes de slot.
    - Quero pedir pizza
    - Quero pedir uma pizza
    - Quero pedir uma pizza de `{pizzaKind}`
    - Quero pedir uma pizza `{size}` de `{pizzaKind}`
    - Quero uma pizza `{size}` de `{pizzaKind}` com massa `{crust}`
    - Posso pedir uma pizza
    - Posso pedir uma pizza de `{pizzaKind}`
    - Posso pedir uma pizza `{size}` de `{pizzaKind}`
  - Lambda initialization and validation (Inicialização de validação do Lambda) – mantenha a configuração padrão.
  - Confirmation prompt (Solicitação de confirmação) – mantenha a configuração padrão.
  - Fulfillment (Atendimento) – execute as seguintes tarefas:
    - Escolha AWS Lambda function (Função do Lambda).
    - Escolha **PizzaOrderProcessor**.
    - Se a caixa de diálogo Add permission to Lambda function (Adicionar permissão à função do Lambda) for mostrada, escolha OK para conceder à intenção `OrderPizza` permissão para chamar a função Lambda do `PizzaOrderProcessor`.
    - Deixe None selecionado.

A intenção deve ter a seguinte aparência:

OrderPizza Latest ▾

▼ Sample utterances ⓘ

+

✕

✕

✕

✕

✕

✕

✕

✕

▶ Lambda initialization and validation ⓘ

▼ Slots ⓘ

| Priority | Required                            | Name   | Slot type                                  | Prompt  |
|----------|-------------------------------------|--|--|---|
|          |                                     | <input type="text" value="e.g. Location"/>                     | <input type="text" value="e.g. AMAZO..."/> | <input type="text" value="e.g. What city?"/> ⚙️ + |
| 1. ▾     | <input checked="" type="checkbox"/> | <input type="text" value="crust"/>                             | Crusts ▾                                   | 1 ▾   |
|          |                                     | <input type="text" value="What kind of crust would you"/> ⚙️ ✕ |  |   |
| 2. ^ ▾   | <input checked="" type="checkbox"/> | <input type="text" value="size"/>                              | Sizes ▾                                    | 1 ▾   |
|          |                                     | <input type="text" value="What size pizza"/> ⚙️ ✕              |  |   |
| 3. ^     | <input checked="" type="checkbox"/> | <input type="text" value="pizzaKind"/>                         | PizzaKind ▾                                | 1 ▾   |
|          |                                     | <input type="text" value="Do you want a veg or chees"/> ⚙️ ✕   |  |   |

▶ Confirmation prompt ⓘ

▼ Fulfillment ⓘ

☒ AWS Lambda function ☐ Return parameters to client

▾

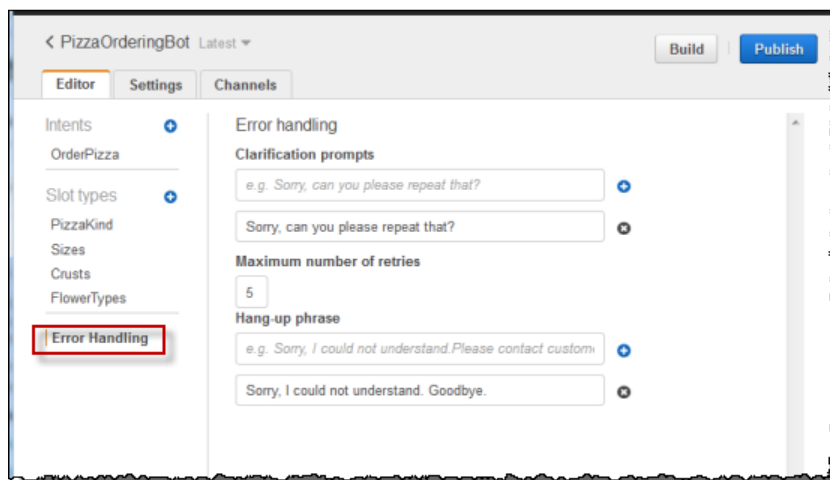
## Próxima etapa

Configurar o bot (p. 70)

## Configurar o bot

Configure o tratamento de erros para o bot PizzaOrderingBot.

1. Navegue até o bot PizzaOrderingBot. Escolha Editor e, em seguida, escolha Erros Handling (Tratamento de erros).



2. Use a guia Editor para configurar a manipulação de erros do bot.

- As informações que você fornece em Clarification Prompts (Solicitações de esclarecimento) são mapeadas para a configuração [clarificationPrompt](#) do bot.

Quando o Amazon Lex não pode determinar a intenção do usuário, o serviço retornará uma resposta com esta mensagem

- As informações que você fornece na frase Hang-up (Encerramento) são mapeadas para a configuração [abortStatement](#) do bot.

Se o serviço não puder determinar a intenção do usuário depois de um determinado número de solicitações consecutivas, o Amazon Lex retornará uma resposta com esta mensagem.

Deixe os valores padrão.

## Próxima etapa

[Etapa 3: crie e teste o bot \(p. 71\)](#)

## Etapa 3: crie e teste o bot

Crie e teste o bot para garantir que ele funciona.

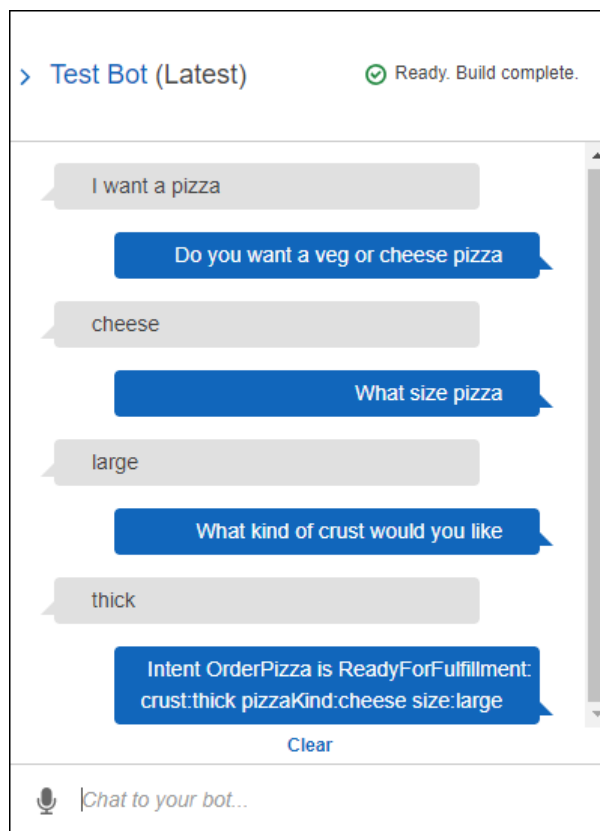
Para criar e testar o bot

1. Para criar o bot `PizzaOrderingBot`, escolha Build.

O Amazon Lex compila um modelo de Machine Learning para o bot. Quando você testa o bot no console, o console usa a API de tempo de execução para enviar a entrada do usuário de volta para o Amazon Lex. O Amazon Lex, por sua vez, usa o modelo de Machine Learning para interpretar a entrada do usuário.

Pode levar algum tempo para concluir a criação.

2. Para testar o bot, na janela Test bot (Testar bot), comece a se comunicar com o bot do Amazon Lex.
  - Por exemplo, você pode dizer ou digitar:



- Use o exemplo de utterances que você configurou na intenção `OrderPizza` para testar o bot. Por exemplo, a amostra a seguir é uma das amostras de utterances que você configurou para a intenção `PizzaOrder`:

```
I want a {size} {crust} crust {pizzaKind} pizza
```

Para testá-la, digite o seguinte:

```
I want a large thin crust cheese pizza
```

Quando você digita "I want to order a pizza (Quero encomendar uma pizza)", o Amazon Lex detecta a intenção (`OrderPizza`). Em seguida, o Amazon Lex solicita informações do slot.

Depois que você fornece todas as informações do slot, o Amazon Lex usa a função do Lambda que você configurou para a intenção.

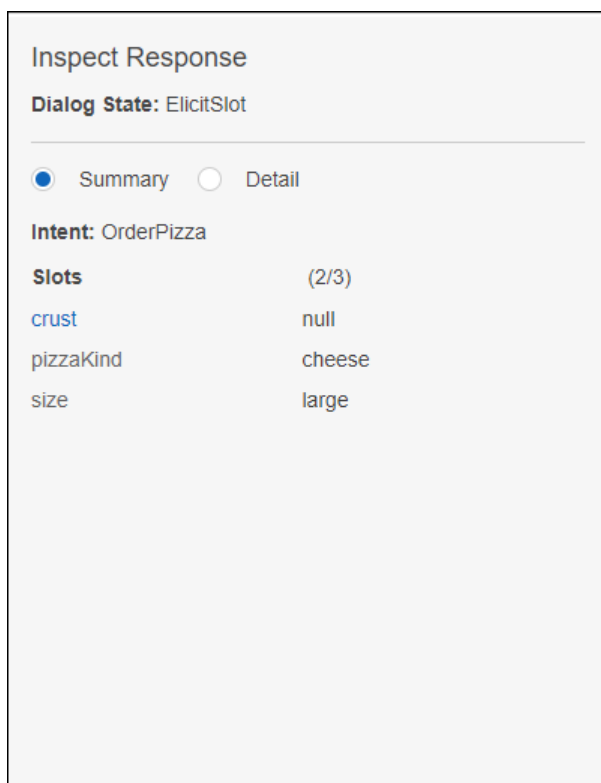
A função do Lambda retorna uma mensagem ("Okay, I have ordered your ... (OK, eu pedi sua ...)") para o Amazon Lex, que o Amazon Lex retorna para você.

## Inspeção da resposta

Abaixo da janela de bate-papo, há um painel que permite inspecionar a resposta do Amazon Lex. O painel fornece informações completas sobre o estado do seu bot, que muda conforme você interage com ele.

O conteúdo do painel mostra o estado atual da operação.

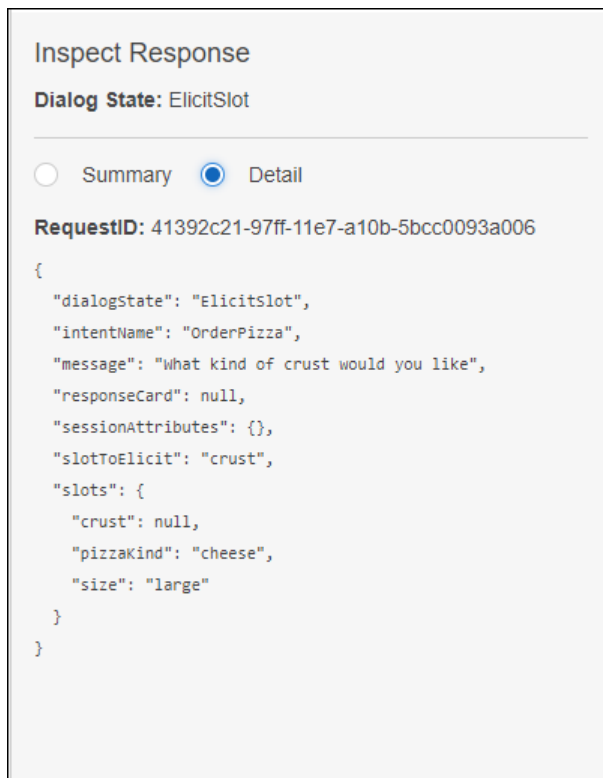
- Dialog State (Estado do diálogo) – o estado atual da conversa com o usuário. Ele pode ser `ElicitIntent`, `ElicitSlot`, `ConfirmIntent` ou `Fulfilled`.
- Summary (Resumo) – mostra uma visualização simplificada da caixa de diálogo que mostra os valores de slot para a intenção que está sendo atendida para que você possa controlar o fluxo de informações. Ele mostra o nome do método, o número de slots, o número de slots preenchidos e uma lista de slots e seus valores associados.



The image shows a screenshot of the 'Inspect Response' window in the Amazon Lex console. It displays the current state of a chatbot conversation. The 'Dialog State' is 'ElicitSlot'. There are two tabs: 'Summary' (selected) and 'Detail'. The 'Intent' is 'OrderPizza'. Under 'Slots', it shows '(2/3)' slots filled. The slots and their values are: 'crust' is 'null', 'pizzaKind' is 'cheese', and 'size' is 'large'.

| Slots     | (2/3)  |
|-----------|--------|
| crust     | null   |
| pizzaKind | cheese |
| size      | large  |

- Detail (Detalhes) – mostra a resposta JSON bruta do chatbot para fornecer uma visualização mais profunda da interação do bot e o estado atual da caixa de diálogo à medida que você testa e depura o chatbot. Se você digitar na janela de bate-papo, o painel de inspeção mostra a resposta JSON da operação [PostText](#) (p. 355). Se você falar com a janela de bate-papo, o painel de inspeção mostra o cabeçalho da resposta da operação [PostContent](#) (p. 347).



## Próxima etapa

Etapa 4 (opcional): Limpeza (p. 74)

## Etapa 4 (opcional): Limpeza

Exclua os recursos que você criou e limpe sua conta para evitar mais cobranças pelos os recursos que você criou.

Você só pode excluir os recursos que não estão em uso. Por exemplo, você não pode excluir um tipo de slot que seja referenciado por uma intenção. Você não pode excluir uma intenção que seja referenciada por um bot.

Exclua recursos na seguinte ordem:

- Exclua bots para liberar recursos de intenção.
- Exclua intenções para liberar recursos de tipo de slot.
- Exclua tipos de slot por último.

Para limpar sua conta

1. Faça login no Console de gerenciamento da AWS e abra o console do Amazon Lex em <https://console.aws.amazon.com/lex/>.
2. Na lista de bots, escolha PizzaOrderingBot.
3. Para excluir o bot, escolha Delete e, em seguida, escolha Continue.
4. No painel à esquerda, selecione Intents.

5. Na lista de intenções, escolha OrderPizza.
6. Para excluir a intenção, escolha Delete e, em seguida, escolha Continue.
7. No menu esquerdo, escolha Slot types.
8. Na lista de tipos de slot, escolha Crusts.
9. Para excluir o tipo de slot, escolha Delete e, em seguida, escolha Continue.
10. Repita [Step 8](#) e [Step 9](#) para os tipos de slot Sizes e PizzaKind.

Você removeu todos os recursos que criou e limpou sua conta.

## Próximas etapas

- [Publique uma versão e crie um alias.](#)
- [Criar um bot do Amazon Lex com a AWS Command Line Interface](#)

## Exercício 3: publique uma versão e crie um alias

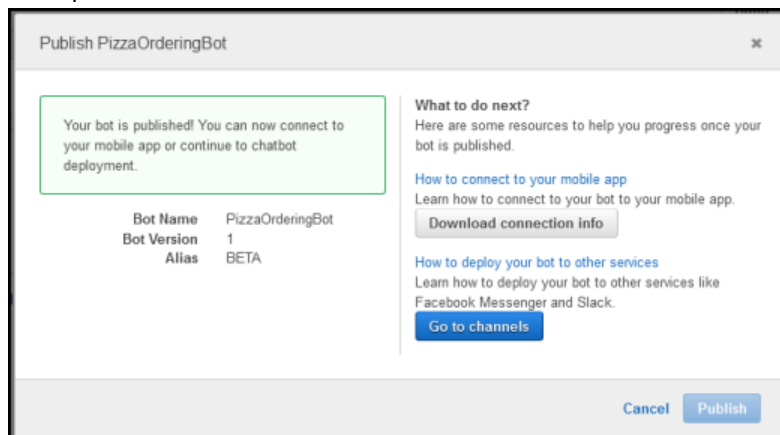
Nos Exercícios 1 e 2 de Conceitos básicos, você criou e testou um bot. Neste exercício, você faz o seguinte:

- Publica uma nova versão do bot. O Amazon Lex faz uma cópia do snapshot da versão `$LATEST` para publicar uma nova versão.
- Crie um alias que aponte para a versão nova.

Para obter mais informações sobre versionamento e aliases, consulte [Controle de versão e aliases](#) (p. 102).

Faça o seguinte para publicar uma versão de um bot criado neste exercício:

1. No console do Amazon Lex, escolha um dos bots que você criou.  
  
Verifique se o console mostra o `$LATEST` como a versão do bot ao lado do nome do bot.
2. Escolha Publish.
3. No assistente Publish **botname** (Publicar nome do bot), especifique o alias **BETA** e selecione Publish (Publicar).
4. Verifique se o console do Amazon Lex mostra a nova versão ao lado do nome do bot.





Agora que você tem um bot funcionando com versão publicada e um alias, pode implantá-lo (em sua aplicação móvel ou integrar o bot com o Facebook Messenger). Para ver um exemplo, consulte [Integração de um bot do Amazon Lex com o Facebook Messenger](#) (p. 116).

## Etapa 4: Conceitos básicos (AWS CLI)

Nesta etapa, você usa a AWS CLI para criar, testar e modificar um bot do Amazon Lex. Para concluir esses exercícios, você precisa estar familiarizado com o uso da CLI e tem um editor de texto. Para obter mais informações, consulte [Etapa 2: Configurar aAWS Command Line Interface](#) (p. 36)

- Exercício 1 — Criar e testar um bot do Amazon Lex. O exercício fornece todos os objetos JSON de que você precisa para criar um tipo de slot personalizado, uma intenção e um bot. Para obter mais informações, consulte [Amazon Lex: Como ele funciona](#) (p. 3)
- Exercício 2 — Atualizar o bot que você criou no Exercício 1 para adicionar mais um enunciado de exemplo. O Amazon Lex usa enunciados de amostra para criar o modelo de machine learning para o bot.
- Exercício 3 — Atualizar o bot que você criou no exercício 1 para adicionar uma função do Lambda para validar a entrada do usuário e atender à intenção.
- Exercício 4 — Publicar uma versão do tipo de slot, intenção e recursos do bot que você criou no Exercício 1. Uma versão é um snapshot de um recurso que não pode ser alterada.
- Exercício 5 — Criar um alias para o bot que você criou no Exercício 1.
- Exercício 6 — Limpar sua conta excluindo o tipo de slot, a intenção e o bot que você criou no Exercício 1, e o alias que você criou no Exercício 5.

### Tópicos

- [Exercício 1: Crie um bot do Amazon Lex \(AWS CLI\)](#) (p. 76)
- [Exercício 2: Adicionar um novo enunciado \(AWS CLI\)](#) (p. 88)
- [Exercício 3: Adicionar uma função do Lambda \(AWS CLI\)](#) (p. 92)
- [Exercício 4: Publicar uma versão \(AWS CLI\)](#) (p. 95)
- [Exercício 5: Criar um alias \(AWS CLI\)](#) (p. 99)
- [Exercício 6: Limpar \(AWS CLI\)](#) (p. 100)

## Exercício 1: Crie um bot do Amazon Lex (AWS CLI)

Em geral, quando você cria bots, você:

1. Cria tipos de slot para definir as informações com as quais seu bot vai trabalhar.
2. Cria as intenções que definem as ações do usuário a que o bot oferece suporte. Use os tipos de slot personalizados que você criou anteriormente para definir os slots, ou parâmetros, que a sua intenção requer.
3. Crie um bot que usa as intenções que você definiu.

Neste exercício, você cria e testa um novo bot do Amazon Lex usando a ILC. Use as estruturas JSON que fornecemos para criar o bot. Para executar os comandos neste exercício, você precisa saber em que região os comandos serão executados. Para obter uma lista de regiões, consulte [Limites de criação de modelos](#) (p. 212).

### Tópicos

- [Etapa 1: Criar uma função vinculada a serviço \(AWS CLI\)](#) (p. 77)

- [Etapa 2: Criar um tipo de slot personalizado \(AWS CLI\) \(p. 77\)](#)
- [Etapa 3: Criar uma intenção \(AWS CLI\) \(p. 79\)](#)
- [Etapa 4: Criar um bot \(AWS CLI\) \(p. 82\)](#)
- [Etapa 5: Testar um bot \(AWS CLI\) \(p. 84\)](#)

## Etapa 1: Criar uma função vinculada a serviço (AWS CLI)

O Amazon Lex assume funções vinculadas a serviço do AWS Identity and Access Management para chamar serviços da AWS em nome de seus bots. As funções, que estão em sua conta, são vinculadas aos casos de uso do Amazon Lex e têm permissões predefinidas. Para obter mais informações, consulte [Permissões de serviço \(p. 9\)](#).

Se você já tiver criado um bot do Amazon Lex usando o console, a função vinculada a serviço terá sido criada automaticamente. Vá para [Etapa 2: Criar um tipo de slot personalizado \(AWS CLI\) \(p. 77\)](#).

Para criar uma função vinculada a serviço (AWS CLI)

1. Na AWS CLI, digite o comando a seguir:

```
aws iam create-service-linked-role --aws-service-name lex.amazonaws.com
```

2. Verifique a política usando o seguinte comando:

```
aws iam get-role --role-name AWSServiceRoleForLexBots
```

A resposta é:

```
{
  "Role": {
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Action": "sts:AssumeRole",
          "Effect": "Allow",
          "Principal": {
            "Service": "lex.amazonaws.com"
          }
        }
      ]
    },
    "RoleName": "AWSServiceRoleForLexBots",
    "Path": "/aws-service-role/lex.amazonaws.com/",
    "Arn": "arn:aws:iam::account-id:role/aws-service-role/lex.amazonaws.com/AWSServiceRoleForLexBots"
  }
}
```

## Próxima etapa

[Etapa 2: Criar um tipo de slot personalizado \(AWS CLI\) \(p. 77\)](#)

## Etapa 2: Criar um tipo de slot personalizado (AWS CLI)

Crie um tipo de slot personalizado com valores de enumeração para as flores que podem ser solicitadas. Você vai usar esse tipo na próxima etapa quando criar a intenção `OrderFlowers`. Um tipo de slot define os valores possíveis para um slot, ou parâmetro, da intenção.

Para executar os comandos neste exercício, você precisa saber em que região os comandos serão executados. Para obter uma lista de regiões, consulte [Limites de criação de modelos](#) (p. 212).

Para criar um tipo de slot personalizado (AWS CLI)

1. Crie um arquivo de texto chamado **FlowerTypes.json**. Copie o seguinte código JSON de [FlowerTypes.json](#) (p. 78) para o arquivo de texto.
2. Chame a operação [PutSlotType](#) (p. 333) usando a AWS CLI para criar o tipo de slot. O exemplo é formatado para Unix, Linux e macOS. Para Windows, substitua o caractere de continuação Unix de barra invertida (\) no final de cada linha por um circunflexo (^).

```
aws lex-models put-slot-type \  
  --region region \  
  --name FlowerTypes \  
  --cli-input-json file://FlowerTypes.json
```

A resposta do servidor é:

```
{  
  "enumerationValues": [  
    {  
      "value": "tulips"  
    },  
    {  
      "value": "lilies"  
    },  
    {  
      "value": "roses"  
    }  
  ],  
  "name": "FlowerTypes",  
  "checksum": "checksum",  
  "version": "$LATEST",  
  "lastUpdatedDate": timestamp,  
  "createdDate": timestamp,  
  "description": "Types of flowers to pick up"  
}
```

## Próxima etapa

[Etapa 3: Criar uma intenção \(AWS CLI\)](#) (p. 79)

### FlowerTypes.json

O código a seguir são os dados JSON necessários para criar o tipo de slot personalizado FlowerTypes:

```
{  
  "enumerationValues": [  
    {  
      "value": "tulips"  
    },  
    {  
      "value": "lilies"  
    },  
    {  
      "value": "roses"  
    }  
  ],  
  "name": "FlowerTypes",  
  "description": "Types of flowers to pick up"
```

```
}
```

## Etapa 3: Criar uma intenção (AWS CLI)

Crie uma intenção para o bot `OrderFlowersBot` e forneça três slots, ou parâmetros. Os slots permitem que o bot cumpra a intenção:

- `FlowerType` é um tipo de slot personalizado que especifica quais tipos de flores podem ser solicitados.
- `AMAZON.DATE` e `AMAZON.TIME` são tipos de slot integrados usados para obter a data e a hora para entregar as flores do usuário.

Para executar os comandos neste exercício, você precisa saber em que região os comandos serão executados. Para obter uma lista de regiões, consulte [Limites de criação de modelos](#) (p. 212).

Para criar a intenção **OrderFlowers** (AWS CLI)

1. Crie um arquivo de texto chamado **OrderFlowers.json**. Copie o seguinte código JSON de [OrderFlowers.json](#) (p. 81) para o arquivo de texto.
2. Na AWS CLI, chame a operação [PutIntent](#) (p. 323) para criar a intenção. O exemplo é formatado para Unix, Linux e macOS. Para Windows, substitua o caractere de continuação Unix de barra invertida (`\`) no final de cada linha por um circunflexo (`^`).

```
aws lex-models put-intent \
  --region region \
  --name OrderFlowers \
  --cli-input-json file://OrderFlowers.json
```

O servidor responde com o seguinte:

```
{
  "confirmationPrompt": {
    "maxAttempts": 2,
    "messages": [
      {
        "content": "Okay, your {FlowerType} will be ready for pickup by {PickupTime} on {PickupDate}. Does this sound okay?",
        "contentType": "PlainText"
      }
    ]
  },
  "name": "OrderFlowers",
  "checksum": "checksum",
  "version": "$LATEST",
  "rejectionStatement": {
    "messages": [
      {
        "content": "Okay, I will not place your order.",
        "contentType": "PlainText"
      }
    ]
  },
  "createdDate": timestamp,
  "lastUpdatedDate": timestamp,
  "sampleUtterances": [
    "I would like to pick up flowers",
    "I would like to order some flowers"
  ],
  "slots": [
```

```
{
  "slotType": "AMAZON.TIME",
  "name": "PickupTime",
  "slotConstraint": "Required",
  "valueElicitationPrompt": {
    "maxAttempts": 2,
    "messages": [
      {
        "content": "Pick up the {FlowerType} at what time on
{PickupDate}?",
        "contentType": "PlainText"
      }
    ]
  },
  "priority": 3,
  "description": "The time to pick up the flowers"
},
{
  "slotType": "FlowerTypes",
  "name": "FlowerType",
  "slotConstraint": "Required",
  "valueElicitationPrompt": {
    "maxAttempts": 2,
    "messages": [
      {
        "content": "What type of flowers would you like to order?",
        "contentType": "PlainText"
      }
    ]
  },
  "priority": 1,
  "slotTypeVersion": "$LATEST",
  "sampleUtterances": [
    "I would like to order {FlowerType}"
  ],
  "description": "The type of flowers to pick up"
},
{
  "slotType": "AMAZON.DATE",
  "name": "PickupDate",
  "slotConstraint": "Required",
  "valueElicitationPrompt": {
    "maxAttempts": 2,
    "messages": [
      {
        "content": "What day do you want the {FlowerType} to be picked
up?",
        "contentType": "PlainText"
      }
    ]
  },
  "priority": 2,
  "description": "The date to pick up the flowers"
}
],
"fulfillmentActivity": {
  "type": "ReturnIntent"
},
"description": "Intent to order a bouquet of flowers for pick up"
}
```

## Próxima etapa

[Etapa 4: Criar um bot \(AWS CLI\) \(p. 82\)](#)

### OrderFlowers.json

O código a seguir são os dados JSON necessários para criar a intenção OrderFlowers:

```
{
  "confirmationPrompt": {
    "maxAttempts": 2,
    "messages": [
      {
        "content": "Okay, your {FlowerType} will be ready for pickup by {PickupTime} on {PickupDate}. Does this sound okay?",
        "contentType": "PlainText"
      }
    ]
  },
  "name": "OrderFlowers",
  "rejectionStatement": {
    "messages": [
      {
        "content": "Okay, I will not place your order.",
        "contentType": "PlainText"
      }
    ]
  },
  "sampleUtterances": [
    "I would like to pick up flowers",
    "I would like to order some flowers"
  ],
  "slots": [
    {
      "slotType": "FlowerTypes",
      "name": "FlowerType",
      "slotConstraint": "Required",
      "valueElicitationPrompt": {
        "maxAttempts": 2,
        "messages": [
          {
            "content": "What type of flowers would you like to order?",
            "contentType": "PlainText"
          }
        ]
      }
    },
    {
      "priority": 1,
      "slotTypeVersion": "$LATEST",
      "sampleUtterances": [
        "I would like to order {FlowerType}"
      ],
      "description": "The type of flowers to pick up"
    },
    {
      "slotType": "AMAZON.DATE",
      "name": "PickupDate",
      "slotConstraint": "Required",
      "valueElicitationPrompt": {
        "maxAttempts": 2,
        "messages": [
          {
            "content": "What day do you want the {FlowerType} to be picked up?",
            "contentType": "PlainText"
          }
        ]
      }
    }
  ]
}
```

```
    }
  ],
  "priority": 2,
  "description": "The date to pick up the flowers"
},
{
  "slotType": "AMAZON.TIME",
  "name": "PickupTime",
  "slotConstraint": "Required",
  "valueElicitationPrompt": {
    "maxAttempts": 2,
    "messages": [
      {
        "content": "Pick up the {FlowerType} at what time on {PickupDate}?",
        "contentType": "PlainText"
      }
    ]
  },
  "priority": 3,
  "description": "The time to pick up the flowers"
}
],
"fulfillmentActivity": {
  "type": "ReturnIntent"
},
"description": "Intent to order a bouquet of flowers for pick up"
}
```

## Etapa 4: Criar um bot (AWS CLI)

O bot `OrderFlowersBot` tem uma intenção, a intenção `OrderFlowers` que você criou na etapa anterior. Para executar os comandos neste exercício, você precisa saber em que região os comandos serão executados. Para obter uma lista de regiões, consulte [Limites de criação de modelos](#) (p. 212).

### Note

O exemplo da AWS CLI a seguir está formatado para Unix, Linux e macOS. Para Windows, altere `"\${LATEST}"` para `$LATEST`.

Para criar o bot **OrderFlowersBot** (AWS CLI)

1. Crie um arquivo de texto chamado **OrderFlowersBot.json**. Copie o seguinte código JSON de [OrderFlowersBot.json](#) (p. 83) para o arquivo de texto.
2. Na AWS CLI, chame a operação [PutBot](#) (p. 311) para criar o bot. O exemplo é formatado para Unix, Linux e macOS. Para Windows, substitua o caractere de continuação Unix de barra invertida (`\`) no final de cada linha por um circunflexo (`^`).

```
aws lex-models put-bot \
  --region region \
  --name OrderFlowersBot \
  --cli-input-json file://OrderFlowersBot.json
```

A resposta do servidor é a seguinte. Quando você cria ou atualiza o bot, o campo `status` é definido como `BUILDING`. Isso indica que o bot não está pronto para uso. Para determinar quando o bot está pronto para uso, use a operação [GetBot](#) (p. 252) na próxima etapa.

```
{
  "status": "BUILDING",
  "intents": [
```

```
{
  "intentVersion": "$LATEST",
  "intentName": "OrderFlowers"
},
{
  "name": "OrderFlowersBot",
  "locale": "en-US",
  "checksum": "checksum",
  "abortStatement": {
    "messages": [
      {
        "content": "Sorry, I'm not able to assist at this time",
        "contentType": "PlainText"
      }
    ]
  },
  "version": "$LATEST",
  "lastUpdatedDate": timestamp,
  "createdDate": timestamp,
  "clarificationPrompt": {
    "maxAttempts": 2,
    "messages": [
      {
        "content": "I didn't understand you, what would you like to do?",
        "contentType": "PlainText"
      }
    ]
  },
  "voiceId": "Salli",
  "childDirected": false,
  "idleSessionTTLInSeconds": 600,
  "processBehavior": "BUILD",
  "description": "Bot to order flowers on the behalf of a user"
}
```

3. Para determinar se o seu novo bot está pronto para uso, execute o comando a seguir. Repita esse comando até que o campo `status` retorne `READY`. O exemplo é formatado para Unix, Linux e macOS. Para Windows, substitua o caractere de continuação Unix de barra invertida (`\`) no final de cada linha por um circunflexo (`^`).

```
aws lex-models get-bot \
  --region region \
  --name OrderFlowersBot \
  --version-or-alias "$LATEST"
```

Procure pelo campo `status` na resposta:

```
{
  "status": "READY",
  ...
}
```

## Próxima etapa

[Etapa 5: Testar um bot \(AWS CLI\) \(p. 84\)](#)

## OrderFlowersBot.json

O código a seguir fornece os dados JSON necessários para criar o bot `OrderFlowers` do Amazon Lex:



```
{
  "intents": [
    {
      "intentVersion": "$LATEST",
      "intentName": "OrderFlowers"
    }
  ],
  "name": "OrderFlowersBot",
  "locale": "en-US",
  "abortStatement": {
    "messages": [
      {
        "content": "Sorry, I'm not able to assist at this time",
        "contentType": "PlainText"
      }
    ]
  },
  "clarificationPrompt": {
    "maxAttempts": 2,
    "messages": [
      {
        "content": "I didn't understand you, what would you like to do?",
        "contentType": "PlainText"
      }
    ]
  },
  "voiceId": "Salli",
  "childDirected": false,
  "idleSessionTTLInSeconds": 600,
  "description": "Bot to order flowers on the behalf of a user"
}
```

## Etapa 5: Testar um bot (AWS CLI)

Para testar o bot, você pode usar um arquivo baseado em texto ou um teste com base em voz.

### Tópicos

- [Testar o bot usando entrada de texto \(AWS CLI\) \(p. 84\)](#)
- [Testar o bot usando entrada de fala \(AWS CLI\) \(p. 86\)](#)

## Testar o bot usando entrada de texto (AWS CLI)

Para verificar se o bot funciona corretamente com a entrada de texto, use a operação [PostText \(p. 355\)](#). Para executar os comandos neste exercício, você precisa saber em que região os comandos serão executados. Para obter uma lista de regiões, consulte [Limites de serviço de tempo de execução \(p. 211\)](#).

### Note

O exemplo da AWS CLI a seguir está formatado para Unix, Linux e macOS. Para Windows, altere "`\$LATEST`" para `$LATEST` e substitua o caractere de continuação de barra invertida (`\`) no final de cada linha por um circunflexo (`^`).

### Para usar texto para testar o bot (AWS CLI)

1. Na AWS CLI, inicie uma conversa com o bot `OrderFlowersBot`. O exemplo é formatado para Unix, Linux e macOS. Para Windows, substitua o caractere de continuação Unix de barra invertida (`\`) no final de cada linha por um circunflexo (`^`).

```
aws lex-runtime post-text \
```

```
--region region \  
--bot-name OrderFlowersBot \  
--bot-alias "\$LATEST" \  
--user-id UserOne \  
--input-text "i would like to order flowers"
```

O Amazon Lex reconhece a intenção do usuário e inicia uma conversa retornando a seguinte resposta:

```
{  
  "slotToElicit": "FlowerType",  
  "slots": {  
    "PickupDate": null,  
    "PickupTime": null,  
    "FlowerType": null  
  },  
  "dialogState": "ElicitSlot",  
  "message": "What type of flowers would you like to order?",  
  "intentName": "OrderFlowers"  
}
```

2. Execute os seguintes comandos para concluir a conversa com o bot.

```
aws lex-runtime post-text \  
  --region region \  
  --bot-name OrderFlowersBot \  
  --bot-alias "\$LATEST" \  
  --user-id UserOne \  
  --input-text "roses"
```

```
aws lex-runtime post-text \  
  --region region \  
  --bot-name OrderFlowersBot \  
  --bot-alias "\$LATEST" \  
  --user-id UserOne \  
  --input-text "tuesday"
```

```
aws lex-runtime post-text \  
  --region region \  
  --bot-name OrderFlowersBot --bot-alias "\$LATEST" \  
  --user-id UserOne \  
  --input-text "10:00 a.m."
```

```
aws lex-runtime post-text \  
  --region region \  
  --bot-name OrderFlowersBot \  
  --bot-alias "\$LATEST" \  
  --user-id UserOne \  
  --input-text "yes"
```

Após a confirmação do pedido, o Amazon Lex envia uma resposta de atendimento para concluir a conversa:

```
{  
  "slots": {  
    "PickupDate": "2017-05-16",  
    "PickupTime": "10:00",  
    "FlowerType": "roses"  
  }
```

```
    },  
    "dialogState": "ReadyForFulfillment",  
    "intentName": "OrderFlowers"  
  }  
}
```

## Próxima etapa

[Testar o bot usando entrada de fala \(AWS CLI\) \(p. 86\)](#)

## Testar o bot usando entrada de fala (AWS CLI)

Para testar o bot usando arquivos de áudio, use a operação [PostContent \(p. 347\)](#). Você gera os arquivos de áudio usando operações de texto para fala do Amazon Polly.

Para executar os comandos neste exercício, você precisa saber em qual região os comandos do Amazon Lex e do Amazon Polly serão executados. Para obter uma lista das regiões do Amazon Lex, consulte [Limites de serviço de tempo de execução \(p. 211\)](#). Para obter uma lista de regiões do Amazon Polly, consulte [Regiões e endpoints do AWS](#) na Referência geral da Amazon Web Services.

### Note

O exemplo da AWS CLI a seguir está formatado para Unix, Linux e macOS. Para Windows, altere "\$LATEST" para \$LATEST e substitua o caractere de continuação de barra invertida (\) no final de cada linha por um circunflexo (^).

Para usar uma entrada de fala para testar o bot (AWS CLI)

1. Na AWS CLI, crie um arquivo de áudio usando o Amazon Polly. O exemplo é formatado para Unix, Linux e macOS. Para Windows, substitua o caractere de continuação Unix de barra invertida (\) no final de cada linha por um circunflexo (^).

```
aws polly synthesize-speech \  
  --region region \  
  --output-format pcm \  
  --text "i would like to order flowers" \  
  --voice-id "Kendra" \  
  IntentSpeech.mpg
```

2. Para enviar o arquivo de áudio para Amazon Lex, execute o seguinte comando. O Amazon Lex salva o áudio da resposta no arquivo de saída especificado.

```
aws lex-runtime post-content \  
  --region region \  
  --bot-name OrderFlowersBot \  
  --bot-alias "$LATEST" \  
  --user-id UserOne \  
  --content-type "audio/l16; rate=16000; channels=1" \  
  --input-stream IntentSpeech.mpg \  
  IntentOutputSpeech.mpg
```

O Amazon Lex responde com uma solicitação para o primeiro slot. Ele salva o áudio da resposta no arquivo de saída especificado.

```
{  
  "contentType": "audio/mpeg",  
  "slotToElicit": "FlowerType",  
  "dialogState": "ElicitSlot",  
  "intentName": "OrderFlowers",  
  "inputTranscript": "i would like to order some flowers",  
}
```

```
"slots": {  
  "PickupDate": null,  
  "PickupTime": null,  
  "FlowerType": null  
},  
"message": "What type of flowers would you like to order?"  
}
```

3. Para encomendar rosas, crie o seguinte arquivo de áudio e envie-o ao Amazon Lex:

```
aws polly synthesize-speech \  
  --region region \  
  --output-format pcm \  
  --text "roses" \  
  --voice-id "Kendra"  
FlowerTypeSpeech.mpg
```

```
aws lex-runtime post-content \  
  --region region \  
  --bot-name OrderFlowersBot \  
  --bot-alias "$LATEST" \  
  --user-id UserOne \  
  --content-type "audio/l16; rate=16000; channels=1" \  
  --input-stream FlowerTypeSpeech.mpg \  
  FlowerTypeOutputSpeech.mpg
```

4. Para definir a data de entrega, crie o seguinte arquivo de áudio e envie-o para o Amazon Lex:

```
aws polly synthesize-speech \  
  --region region \  
  --output-format pcm \  
  --text "tuesday" \  
  --voice-id "Kendra"  
DateSpeech.mpg
```

```
aws lex-runtime post-content \  
  --region region \  
  --bot-name OrderFlowersBot \  
  --bot-alias "$LATEST" \  
  --user-id UserOne \  
  --content-type "audio/l16; rate=16000; channels=1" \  
  --input-stream DateSpeech.mpg  
DateOutputSpeech.mpg
```

5. Para definir o horário de entrega, crie o seguinte arquivo de áudio e envie-o para o Amazon Lex:

```
aws polly synthesize-speech \  
  --region region \  
  --output-format pcm \  
  --text "10:00 a.m." \  
  --voice-id "Kendra"  
TimeSpeech.mpg
```

```
aws lex-runtime post-content \  
  --region region \  
  --bot-name OrderFlowersBot \  
  --bot-alias "$LATEST" \  
  --user-id UserOne \  
  --content-type "audio/l16; rate=16000; channels=1" \  
  --input-stream TimeSpeech.mpg
```

```
TimeOutputSpeech.mpg
```

6. Para confirmar a entrega, crie o seguinte arquivo de áudio e envie-o ao Amazon Lex:

```
aws polly synthesize-speech \  
  --region region \  
  --output-format pcm \  
  --text "yes" \  
  --voice-id "Kendra"  
ConfirmSpeech.mpg
```

```
aws lex-runtime post-content \  
  --region region \  
  --bot-name OrderFlowersBot \  
  --bot-alias "$LATEST" \  
  --user-id UserOne \  
  --content-type "audio/l16; rate=16000; channels=1" \  
  --input-stream ConfirmSpeech.mpg \  
ConfirmOutputSpeech.mpg
```

Depois que você confirmar a entrega, o Amazon Lex envia uma resposta que confirma o atendimento da intenção:

```
{  
  "contentType": "text/plain;charset=utf-8",  
  "dialogState": "ReadyForFulfillment",  
  "intentName": "OrderFlowers",  
  "inputTranscript": "yes",  
  "slots": {  
    "PickupDate": "2017-05-16",  
    "PickupTime": "10:00",  
    "FlowerType": "roses"  
  }  
}
```

### Próxima etapa

[Exercício 2: Adicionar um novo enunciado \(AWS CLI\) \(p. 88\)](#)

## Exercício 2: Adicionar um novo enunciado (AWS CLI)

Para melhorar o modelo de Machine Learning que o Amazon Lex usa para reconhecer solicitações dos usuários, adicione outro enunciado de exemplo ao bot.

A adição de um novo enunciado é um processo de quatro etapas.

1. Use a operação [GetIntent \(p. 288\)](#) para obter uma intenção do Amazon Lex.
2. Atualize a intenção.
3. Use a operação [PutIntent \(p. 323\)](#) para enviar a intenção atualizada de volta ao Amazon Lex.
4. Use as operações [GetBot \(p. 252\)](#) e [PutBot \(p. 311\)](#) para recriar qualquer bot que use a intenção.

Para executar os comandos neste exercício, você precisa saber em que região os comandos serão executados. Para obter uma lista de regiões, consulte [Limites de criação de modelos \(p. 212\)](#).

A resposta da operação `GetIntent` contém um campo chamado `checksum` que identifica uma revisão específica da intenção. Você deve fornecer o valor de soma de verificação quando usa a operação

[PutIntent \(p. 323\)](#) para atualizar uma intenção. Se não fizer isso, você receberá a seguinte mensagem de erro:

```
An error occurred (PreconditionFailedException) when calling
the PutIntent operation: Intent intent name already exists.
If you are trying to update intent name you must specify the
checksum.
```

#### Note

O exemplo da AWS CLI a seguir está formatado para Unix, Linux e macOS. Para Windows, altere "`\$LATEST`" para `$LATEST` e substitua o caractere de continuação de barra invertida (`\`) no final de cada linha por um circunflexo (`^`).

Para atualizar a intenção **OrderFlowers** (AWS CLI)

1. No AWS CLI, obtenha a intenção do Amazon Lex. O Amazon Lex envia a saída para um arquivo chamado **OrderFlowers-V2.json**.

```
aws lex-models get-intent \
  --region region \
  --name OrderFlowers \
  --intent-version "\$LATEST" > OrderFlowers-V2.json
```

2. Em um editor de texto, abra **OrderFlowers-V2.json**.

1. Encontre e exclua os campos `createdDate`, `lastUpdatedDate` e `version`.
2. Adicione o seguinte ao campo `sampleUtterances`:

```
I want to order flowers
```

3. Salve o arquivo.
3. Envie a intenção atualizada ao Amazon Lex com o seguinte comando:

```
aws lex-models put-intent \
  --region region \
  --name OrderFlowers \
  --cli-input-json file://OrderFlowers-V2.json
```

O Amazon Lex envia a seguinte resposta:

```
{
  "confirmationPrompt": {
    "maxAttempts": 2,
    "messages": [
      {
        "content": "Okay, your {FlowerType} will be ready for pickup by
{PickupTime} on {PickupDate}. Does this sound okay?",
        "contentType": "PlainText"
      }
    ]
  },
  "name": "OrderFlowers",
  "checksum": "checksum",
  "version": "$LATEST",
  "rejectionStatement": {
    "messages": [
      {
```

```
        "content": "Okay, I will not place your order.",
        "contentType": "PlainText"
    }
  ],
  "createdDate": timestamp,
  "lastUpdatedDate": timestamp,
  "sampleUtterances": [
    "I would like to pick up flowers",
    "I would like to order some flowers",
    "I want to order flowers"
  ],
  "slots": [
    {
      "slotType": "AMAZON.TIME",
      "name": "PickupTime",
      "slotConstraint": "Required",
      "valueElicitationPrompt": {
        "maxAttempts": 2,
        "messages": [
          {
            "content": "Pick up the {FlowerType} at what time on
{PickupDate}?",
            "contentType": "PlainText"
          }
        ]
      },
      "priority": 3,
      "description": "The time to pick up the flowers"
    },
    {
      "slotType": "FlowerTypes",
      "name": "FlowerType",
      "slotConstraint": "Required",
      "valueElicitationPrompt": {
        "maxAttempts": 2,
        "messages": [
          {
            "content": "What type of flowers would you like to order?",
            "contentType": "PlainText"
          }
        ]
      },
      "priority": 1,
      "slotTypeVersion": "$LATEST",
      "sampleUtterances": [
        "I would like to order {FlowerType}"
      ],
      "description": "The type of flowers to pick up"
    },
    {
      "slotType": "AMAZON.DATE",
      "name": "PickupDate",
      "slotConstraint": "Required",
      "valueElicitationPrompt": {
        "maxAttempts": 2,
        "messages": [
          {
            "content": "What day do you want the {FlowerType} to be picked
up?",
            "contentType": "PlainText"
          }
        ]
      },
      "priority": 2,
      "description": "The date to pick up the flowers"
    }
  ]
}
```

```
    },  
    ],  
    "fulfillmentActivity": {  
      "type": "ReturnIntent"  
    },  
    "description": "Intent to order a bouquet of flowers for pick up"  
  }  
}
```

Agora que você atualizou a intenção, recrie qualquer bot que a utilize.

Para recriar o bot **OrderFlowersBot** (AWS CLI)

1. Na AWS CLI, obtenha a definição do bot `OrderFlowersBot` e salve-a em um arquivo com o seguinte comando:

```
aws lex-models get-bot \  
  --region region \  
  --name OrderFlowersBot \  
  --version-or-alias "$LATEST" > OrderFlowersBot-V2.json
```

2. Em um editor de texto, abra **OrderFlowersBot-V2.json**. Remova os campos `createdDate`, `lastUpdatedDate`, `status` e `version`.
3. Em um editor de texto, adicione a seguinte linha à definição do bot:

```
"processBehavior": "BUILD",
```

4. Na AWS CLI, crie uma nova revisão do bot executando o seguinte comando para:

```
aws lex-models put-bot \  
  --region region \  
  --name OrderFlowersBot \  
  --cli-input-json file://OrderFlowersBot-V2.json
```

A resposta do servidor é:

```
{  
  "status": "BUILDING",  
  "intents": [  
    {  
      "intentVersion": "$LATEST",  
      "intentName": "OrderFlowers"  
    }  
  ],  
  "name": "OrderFlowersBot",  
  "locale": "en-US",  
  "checksum": "checksum",  
  "abortStatement": {  
    "messages": [  
      {  
        "content": "Sorry, I'm not able to assist at this time",  
        "contentType": "PlainText"  
      }  
    ]  
  },  
  "version": "$LATEST",  
  "lastUpdatedDate": timestamp,  
  "createdDate": timestamp,  
  "clarificationPrompt": {  
    "maxAttempts": 2,  
    "messages": [  

```



```
{
  "content": "I didn't understand you, what would you like to do?",
  "contentType": "PlainText"
},
"voiceId": "Salli",
"childDirected": false,
"idleSessionTTLInSeconds": 600,
"description": "Bot to order flowers on the behalf of a user"
}
```

## Próxima etapa

Exercício 3: Adicionar uma função do Lambda (AWS CLI) (p. 92)

## Exercício 3: Adicionar uma função do Lambda (AWS CLI)

Adicione uma função do Lambda que valide a entrada do usuário e atenda à intenção do usuário ao bot.

A adição de uma expressão do Lambda é um processo de cinco etapas.

1. Use a função [AddPermission](#) do Lambda para permitir que a intenção OrderFlowers chame a operação [Invoke](#) do Lambda.
2. Use a operação [GetIntent](#) (p. 288) para obter a intenção do Amazon Lex.
3. Atualize a intenção para adicionar a função do Lambda.
4. Use a operação [PutIntent](#) (p. 323) para enviar a intenção atualizada de volta ao Amazon Lex.
5. Use as operações [GetBot](#) (p. 252) e [PutBot](#) (p. 311) para recriar qualquer bot que use a intenção.

Para executar os comandos neste exercício, você precisa saber em que região os comandos serão executados. Para obter uma lista de regiões, consulte [Limites de criação de modelos](#) (p. 212).

Se você adicionar uma função do Lambda a uma intenção antes de adicionar a permissão `InvokeFunction`, verá a seguinte mensagem de erro:

```
An error occurred (BadRequestException) when calling the
PutIntent operation: Lex is unable to access the Lambda
function Lambda function ARN in the context of intent
intent ARN. Please check the resource-based policy on
the function.
```

A resposta da operação `GetIntent` contém um campo chamado `checksum` que identifica uma revisão específica da intenção. Quando usa a operação [PutIntent](#) (p. 323) para atualizar uma intenção, você deve fornecer o valor de soma de verificação. Se não fizer isso, você receberá a seguinte mensagem de erro:

```
An error occurred (PreconditionFailedException) when calling
the PutIntent operation: Intent intent name already exists.
If you are trying to update intent name you must specify the
checksum.
```

Este exercício usa a função do Lambda do [Exercício 1: Criar um bot do Amazon Lex usando um esquema \(console\)](#) (p. 38). Para obter instruções para criar a função do Lambda, consulte [Etapa 3: Criar uma função do Lambda \(console\)](#) (p. 50).

#### Note

O exemplo da AWS CLI a seguir está formatado para Unix, Linux e macOS. Para Windows, altere `"\${LATEST}"` para `$LATEST`.

Para adicionar uma função do Lambda a uma intenção

1. Na AWS CLI, adicione a permissão `InvokeFunction` para a intenção `OrderFlowers`:

```
aws lambda add-permission \  
  --region region \  
  --function-name OrderFlowersCodeHook \  
  --statement-id LexGettingStarted-OrderFlowersBot \  
  --action lambda:InvokeFunction \  
  --principal lex.amazonaws.com \  
  --source-arn "arn:aws:lex:region:account ID:intent:OrderFlowers:*
```

O Lambda envia a seguinte resposta:

```
{  
  "Statement": "{ \"Sid\": \"LexGettingStarted-OrderFlowersBot\",  
    \"Resource\": \"arn:aws:lambda:region:account ID:function:OrderFlowersCodeHook\",  
    \"Effect\": \"Allow\",  
    \"Principal\": { \"Service\": \"lex.amazonaws.com\" },  
    \"Action\": [ \"lambda:InvokeFunction\" ],  
    \"Condition\": { \"ArnLike\":  
      { \"AWS:SourceArn\":  
        \"arn:aws:lex:region:account ID:intent:OrderFlowers:*\" } } } } }"
```

2. Obtenha a intenção do Amazon Lex. O Amazon Lex envia a saída para um arquivo chamado **OrderFlowers-V3.json**.

```
aws lex-models get-intent \  
  --region region \  
  --name OrderFlowers \  
  --intent-version "\${LATEST}" > OrderFlowers-V3.json
```

3. Abra um editor de texto, abra o **OrderFlowers-V3.json**.

1. Encontre e exclua os campos `createdDate`, `lastUpdatedDate` e `version`.
2. Atualize o campo `fulfillmentActivity`:

```
"fulfillmentActivity": {  
  "type": "CodeHook",  
  "codeHook": {  
    "uri": "arn:aws:lambda:region:account ID:function:OrderFlowersCodeHook",  
    "messageVersion": "1.0"  
  }  
}
```

3. Salve o arquivo.
4. Na AWS CLI, envie a intenção atualizada para o Amazon Lex:

```
aws lex-models put-intent \  
  --region region \  
  --name OrderFlowers \  
  --intent-version "\${LATEST}"
```

```
--name OrderFlowers \  
--cli-input-json file://OrderFlowers-V3.json
```

Agora que você atualizou a intenção, recrie o bot.

#### Para recriar o bot **OrderFlowersBot**

1. Na AWS CLI, obtenha a definição do bot **OrderFlowersBot** e salve-a em um arquivo:

```
aws lex-models get-bot \  
--region region \  
--name OrderFlowersBot \  
--version-or-alias "$LATEST" > OrderFlowersBot-V3.json
```

2. Em um editor de texto, abra **OrderFlowersBot-V3.json**. Remova os campos `createdDate`, `lastUpdatedDate`, `status` e `version`.
3. No editor de texto, adicione a seguinte linha à definição do bot:

```
"processBehavior": "BUILD",
```

4. Na AWS CLI, crie uma nova revisão do bot:

```
aws lex-models put-bot \  
--region region \  
--name OrderFlowersBot \  
--cli-input-json file://OrderFlowersBot-V3.json
```

A resposta do servidor é:

```
{  
  "status": "READY",  
  "intents": [  
    {  
      "intentVersion": "$LATEST",  
      "intentName": "OrderFlowers"  
    }  
  ],  
  "name": "OrderFlowersBot",  
  "locale": "en-US",  
  "checksum": "checksum",  
  "abortStatement": {  
    "messages": [  
      {  
        "content": "Sorry, I'm not able to assist at this time",  
        "contentType": "PlainText"  
      }  
    ]  
  },  
  "version": "$LATEST",  
  "lastUpdatedDate": timestamp,  
  "createdDate": timestamp,  
  "clarificationPrompt": {  
    "maxAttempts": 2,  
    "messages": [  
      {  
        "content": "I didn't understand you, what would you like to do?",  
        "contentType": "PlainText"  
      }  
    ]  
  },  
}
```

```
"voiceId": "Salli",  
"childDirected": false,  
"idleSessionTTLInSeconds": 600,  
"description": "Bot to order flowers on the behalf of a user"  
}
```

## Próxima etapa

[Exercício 4: Publicar uma versão \(AWS CLI\) \(p. 95\)](#)

## Exercício 4: Publicar uma versão (AWS CLI)

Agora, crie uma versão do bot que você criou no Exercício 1. Uma versão é um snapshot do bot. Depois que criar uma versão, você não poderá alterá-la. A única versão de um bot que você pode atualizar é a versão `$LATEST`. Para obter mais informações sobre versões, consulte [Controle de versão e aliases \(p. 102\)](#).

Antes de poder publicar uma versão de um bot, você deve publicar a intenção que ele usa. Da mesma forma, você deve publicar os tipos de slot a que essas intenções se referem. No geral, para publicar uma versão de um bot, você faz o seguinte:

1. Publica uma versão de um tipo de slot com a operação [CreateSlotTypeVersion \(p. 230\)](#).
2. Publica uma versão de uma intenção com a operação [CreateIntentVersion \(p. 224\)](#).
3. Publica uma versão de um bot com a operação [CreateBotVersion \(p. 219\)](#).

Para executar os comandos neste exercício, você precisa saber em que região os comandos serão executados. Para obter uma lista de regiões, consulte [Limites de criação de modelos \(p. 212\)](#).

### Tópicos

- [Etapa 1: Publicar o tipo de slot \(AWS CLI\) \(p. 95\)](#)
- [Etapa 2: Publicar a intenção \(AWS CLI\) \(p. 96\)](#)
- [Etapa 3: Publicar o bot \(AWS CLI\) \(p. 98\)](#)

## Etapa 1: Publicar o tipo de slot (AWS CLI)

Antes de poder publicar uma versão de qualquer intenção que usa um tipo de slot, você deve publicar uma versão desse tipo de slot. Neste caso, você publica o tipo de slot `FlowerTypes`.

### Note

O exemplo da AWS CLI a seguir está formatado para Unix, Linux e macOS. Para Windows, altere `"\${LATEST}"` para `$LATEST` e substitua o caractere de continuação de barra invertida (`\`) no final de cada linha por um circunflexo (`^`).

Para publicar o tipo de slot (AWS CLI)

1. Na AWS CLI, obtenha a versão mais recente do tipo de slot:

```
aws lex-models get-slot-type \  
--region region \  
--name FlowerTypes \  
--slot-type-version "\${LATEST}"
```

A resposta do Amazon Lex segue. Registre a soma de verificação para a revisão atual da versão `$LATEST`.

```
{
  "enumerationValues": [
    {
      "value": "tulips"
    },
    {
      "value": "lilies"
    },
    {
      "value": "roses"
    }
  ],
  "name": "FlowerTypes",
  "checksum": "checksum",
  "version": "$LATEST",
  "lastUpdatedDate": timestamp,
  "createdDate": timestamp,
  "description": "Types of flowers to pick up"
}
```

2. Publique uma nova versão do tipo de slot. Use a soma de verificação que você registrou na etapa anterior.

```
aws lex-models create-slot-type-version \
  --region region \
  --name FlowerTypes \
  --checksum "checksum"
```

A resposta do Amazon Lex segue. Registre o número da versão para a próxima etapa.

```
{
  "version": "1",
  "enumerationValues": [
    {
      "value": "tulips"
    },
    {
      "value": "lilies"
    },
    {
      "value": "roses"
    }
  ],
  "name": "FlowerTypes",
  "createdDate": timestamp,
  "lastUpdatedDate": timestamp,
  "description": "Types of flowers to pick up"
}
```

## Próxima etapa

[Etapa 2: Publicar a intenção \(AWS CLI\) \(p. 96\)](#)

## Etapa 2: Publicar a intenção (AWS CLI)

Antes de poder publicar uma intenção, você tem que publicar todos os tipos de slot mencionados pela intenção. Os tipos de slot devem ser versões numeradas, não a versão `$LATEST`.

Primeiro, atualize a intenção `OrderFlowers` para usar a versão do tipo de slot `FlowerTypes` que você publicou na etapa anterior. Depois, publique uma nova versão da intenção `OrderFlowers`.

#### Note

O exemplo da AWS CLI a seguir está formatado para Unix, Linux e macOS. Para Windows, altere `"\${LATEST}"` para `$LATEST` e substitua o caractere de continuação de barra invertida (`\`) no final de cada linha por um circunflexo (`^`).

Para publicar uma versão de uma intenção (AWS CLI)

1. Na AWS CLI, obtenha a versão `$LATEST` da intenção `OrderFlowers` e salve-a em um arquivo:

```
aws lex-models get-intent \  
  --region region \  
  --name OrderFlowers \  
  --intent-version "\${LATEST}" > OrderFlowers_V4.json
```

2. Em um editor de texto, abra o arquivo **OrderFlowers\_V4.json**. Exclua os campos `createdDate`, `lastUpdatedDate` e `version`. Encontre o tipo de slot `FlowerTypes` e altere a versão para o número da versão que você registrou na etapa anterior. O seguinte fragmento do arquivo **OrderFlowers\_V4.json** mostra a localização da alteração:

```
{  
  "slotType": "FlowerTypes",  
  "name": "FlowerType",  
  "slotConstraint": "Required",  
  "valueElicitationPrompt": {  
    "maxAttempts": 2,  
    "messages": [  
      {  
        "content": "What type of flowers?",  
        "contentType": "PlainText"  
      }  
    ]  
  },  
  "priority": 1,  
  "slotTypeVersion": "version",  
  "sampleUtterances": []  
},
```

3. Na AWS CLI, salve a revisão da intenção:

```
aws lex-models put-intent \  
  --name OrderFlowers \  
  --cli-input-json file://OrderFlowers_V4.json
```

4. Obtenha a soma de verificação da última revisão da intenção:

```
aws lex-models get-intent \  
  --region region \  
  --name OrderFlowers \  
  --intent-version "\${LATEST}" > OrderFlowers_V4a.json
```

O seguinte fragmento de resposta mostra a soma de verificação da intenção. Registre isso para a próxima etapa.

```
"name": "OrderFlowers",  
"checksum": "checksum",  
"version": "$LATEST",
```

5. Publique uma nova versão da intenção:

```
aws lex-models create-intent-version \  
  --region region \  
  --name OrderFlowers \  
  --checksum "checksum"
```

O seguinte fragmento de resposta mostra a nova versão da intenção. Registre o número da versão para a próxima etapa.

```
"name": "OrderFlowers",  
"checksum": "checksum",  
"version": "version",
```

## Próxima etapa

[Etapa 3: Publicar o bot \(AWS CLI\) \(p. 98\)](#)

## Etapa 3: Publicar o bot (AWS CLI)

Depois que você tiver publicado todos os tipos de slot e intenções que são usados pelo seu bot, você pode publicar o bot.

Atualize o bot `OrderFlowersBot` para usar a intenção `OrderFlowers` que você atualizou na etapa anterior. Depois, publique uma nova versão do bot `OrderFlowersBot`.

### Note

O exemplo da AWS CLI a seguir está formatado para Unix, Linux e macOS. Para Windows, altere "`\$LATEST`" para `$LATEST` e substitua o caractere de continuação de barra invertida (`\`) no final de cada linha por um circunflexo (`^`).

Para publicar uma versão de um bot (AWS CLI)

1. Na AWS CLI, obtenha a versão `$LATEST` do bot `OrderFlowersBot` e salve-a em um arquivo:

```
aws lex-models get-bot \  
  --region region \  
  --name OrderFlowersBot \  
  --version-or-alias "\$LATEST" > OrderFlowersBot_V4.json
```

2. Em um editor de texto, abra o arquivo **OrderFlowersBot\_V4.json**. Exclua os campos `createdDate`, `lastUpdatedDate`, `status` e `version`. Encontre a intenção `OrderFlowers` e altere a versão para o número da versão que você registrou na etapa anterior. O seguinte fragmento de **OrderFlowersBot\_V4.json** mostra a localização da alteração.

```
"intents": [  
  {  
    "intentVersion": "version",  
    "intentName": "OrderFlowers"  
  }  
]
```

3. Na AWS CLI, salve a nova revisão do bot:

```
aws lex-models put-bot \  
  --name OrderFlowersBot \  
  --cli-input-json file://OrderFlowersBot_V4.json
```

4. Obtenha a soma de verificação da última revisão do bot:

```
aws lex-models get-bot \  
  --region region \  
  --name OrderFlowersBot > OrderFlowersBot_V4a.json
```

O seguinte fragmento de resposta mostra a soma de verificação do bot. Registre isso para a próxima etapa.

```
"name": "OrderFlowersBot",  
"locale": "en-US",  
"checksum": "checksum",
```

5. Publique uma nova versão do bot:

```
aws lex-models create-bot-version \  
  --region region \  
  --name OrderFlowersBot \  
  --checksum "checksum"
```

O seguinte fragmento de resposta mostra a nova versão do bot.

```
"checksum": "checksum",  
"abortStatement": {  
  ...  
},  
"version": "1",  
"lastUpdatedDate": timestamp,
```

## Próxima etapa

Exercício 5: Criar um alias (AWS CLI) (p. 99)

## Exercício 5: Criar um alias (AWS CLI)

Um alias é um ponteiro para uma versão específica de um bot. Com um alias, você pode atualizar com facilidade a versão que seus aplicativos cliente estão usando. Para obter mais informações, consulte [Controle de versão e aliases](#) (p. 102). Para executar os comandos neste exercício, você precisa saber em que região os comandos serão executados. Para obter uma lista de regiões, consulte [Limites de criação de modelos](#) (p. 212).

Para criar um alias (AWS CLI)

1. Na AWS CLI, obtenha a versão do bot OrderFlowersBot que você criou em [Exercício 4: Publicar uma versão \(AWS CLI\)](#) (p. 95).

```
aws lex-models get-bot \  
  --region region \  
  --name OrderFlowersBot \  
  --version-or-alias version > OrderFlowersBot_V5.json
```

2. Em um editor de texto, abra **OrderFlowersBot\_v5.json**. Encontre e registre o número da versão.  
3. Na AWS CLI, crie o alias do bot:

```
aws lex-models put-bot-alias \  
  --region region \  
  --name OrderFlowersBot \  
  --version-or-alias version > OrderFlowersBot_V5.json
```



```
--name PROD \  
--bot-name OrderFlowersBot \  
--bot-version version
```

A resposta do servidor é a seguinte:

```
{  
  "name": "PROD",  
  "createdDate": timestamp,  
  "checksum": "checksum",  
  "lastUpdatedDate": timestamp,  
  "botName": "OrderFlowersBot",  
  "botVersion": "1"  
}}
```

## Próxima etapa

[Exercício 6: Limpar \(AWS CLI\) \(p. 100\)](#)

## Exercício 6: Limpar (AWS CLI)

Exclua os recursos que você criou e limpe sua conta.

Você só pode excluir os recursos que não estão em uso. Em geral, você deve excluir recursos na seguinte ordem.

1. Exclua aliases para liberar recursos de bot.
2. Exclua bots para liberar recursos de intenção.
3. Exclua intenções para liberar recursos de tipo de slot.
4. Exclua tipos de slot.

Para executar os comandos neste exercício, você precisa saber em que região os comandos serão executados. Para obter uma lista de regiões, consulte [Limites de criação de modelos \(p. 212\)](#).

Para limpar sua conta (AWS CLI)

1. Na linha de comando da AWS CLI, exclua o alias:

```
aws lex-models delete-bot-alias \  
  --region region \  
  --name PROD \  
  --bot-name OrderFlowersBot
```

2. Na linha de comando da AWS CLI, exclua o bot:

```
aws lex-models delete-bot \  
  --region region \  
  --name OrderFlowersBot
```

3. Na linha de comando da AWS CLI, exclua a intenção:

```
aws lex-models delete-intent \  
  --region region \  
  --name OrderFlowers
```

4. Na linha de comando da AWS CLI, exclua o tipo de slot:

```
aws lex-models delete-slot-type \  
  --region region \  
  --name FlowerTypes
```

Você removeu todos os recursos que criou e limpou sua conta.

# Controle de versão e aliases

O Amazon Lex oferece suporte a versões de publicação de bots, intenções e tipos de slot para que você possa controlar a implementação usada por seus aplicativos cliente. Uma versão é um snapshot numerado de seu trabalho que você pode publicar para uso em diferentes partes de seu fluxo de trabalho, como desenvolvimento, implementação beta e produção.

Os bots do Amazon Lex também oferecem suporte a aliases. Um alias é um ponteiro para uma versão específica de um bot. Com um alias, você pode atualizar com facilidade a versão que seus aplicativos cliente estão usando. Por exemplo, você pode apontar um alias para a versão 1 do seu bot. Quando estiver pronto para atualizar o bot, você publica a versão 2 e altera o alias para apontar para a nova versão. Como suas aplicações usam o alias ao invés de uma versão específica, todos os seus clientes obtêm a nova funcionalidade sem a necessidade de atualizações.

Tópicos

- [Controle de versão \(p. 102\)](#)
- [Aliases \(p. 104\)](#)

## Controle de versão

Quando versiona um recurso do Amazon Lex, você cria um snapshot do recurso para que possa usar o recurso da forma como existia quando a versão foi criada. Após criar uma versão, ela permanecerá a mesma enquanto você continuar trabalhando no seu aplicativo.

### A versão \$LATEST

Quando você cria um bot, intenção ou tipo de slot do Amazon Lex, há apenas uma versão, a versão \$LATEST.



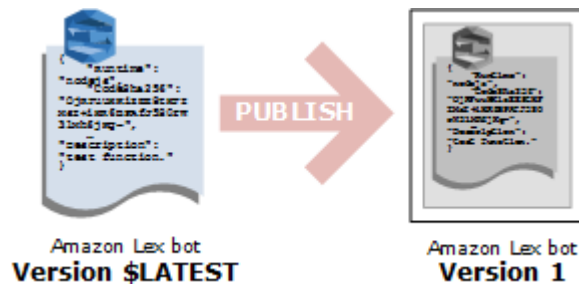
Amazon Lex bot  
**Version \$LATEST**

\$LATEST é a cópia de trabalho do seu recurso. Você pode atualizar apenas a versão \$LATEST, e até você publicar sua primeira versão, \$LATEST será a única versão do recurso.

Somente a versão \$LATEST de um recurso pode usar a versão \$LATEST do outro recurso. Por exemplo, a versão \$LATEST de um bot pode usar a versão \$LATEST de uma intenção, e a versão \$LATEST de uma intenção pode usar a versão \$LATEST de um tipo de slot.

## Publicação de uma versão de recurso do Amazon Lex

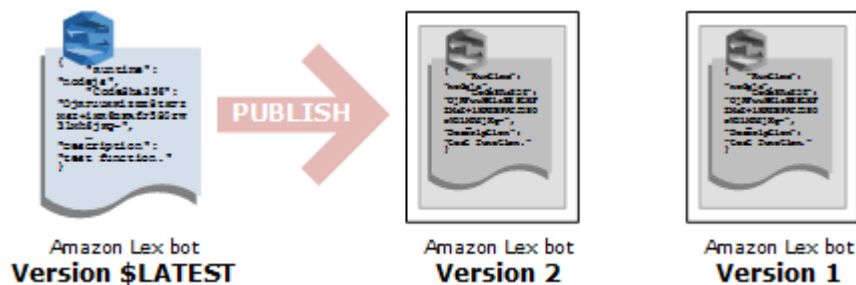
Quando você publica um recurso, o Amazon Lex faz uma cópia da versão \$LATEST e salva-a como uma versão numerada. A versão publicada não pode ser alterada.



Você cria e publica versões usando o console do Amazon Lex ou a operação [CreateBotVersion](#) (p. 219). Para ver um exemplo, consulte [Exercício 3: publique uma versão e crie um alias](#) (p. 75).

Quando você modifica a versão \$LATEST de um recurso, pode publicar a nova versão para disponibilizar as alterações para os aplicativos de seus clientes. Toda vez que você publica uma versão, o Amazon Lex copia a versão \$LATEST para criar a nova versão e incrementa 1 no número da versão. Os números de versão nunca são reutilizados. Por exemplo, se você remover a versão 10 numerada de um recurso e, em seguida, recriá-la, o próximo número de versão atribuído pelo Amazon Lex será a versão 11.

Para publicar um bot, você deve apontá-lo para uma versão numerada de qualquer intenção que ele use. Se você tentar publicar uma nova versão de um bot que usa a versão \$LATEST versão de uma intenção, O Amazon Lex retorna uma exceção HTTP 400: solicitação inválida. Para publicar uma versão numerada da intenção, você deve apontar a intenção de uma versão numerada de qualquer tipo de slot que ele use. Caso contrário, você receberá uma exceção HTTP 400: solicitação inválida.



#### Note

O Amazon Lex publica uma nova versão somente se a última versão publicada for diferente da versão \$LATEST. Se você tentar publicar a versão \$LATEST sem modificá-la, o Amazon Lex não criará nem publicará uma nova versão.

## Atualização de um recurso do Amazon Lex

Você pode atualizar apenas a versão \$LATEST de um bot, uma intenção ou um tipo de slot do Amazon Lex. As versões publicadas não podem ser alteradas. Você pode publicar uma nova versão a qualquer momento após atualizar um recurso no console ou com as operações [CreateBotVersion](#) (p. 219), [CreateIntentVersion](#) (p. 224) ou [CreateSlotTypeVersion](#) (p. 230).

## Exclusão de um recurso ou de uma versão do Amazon Lex

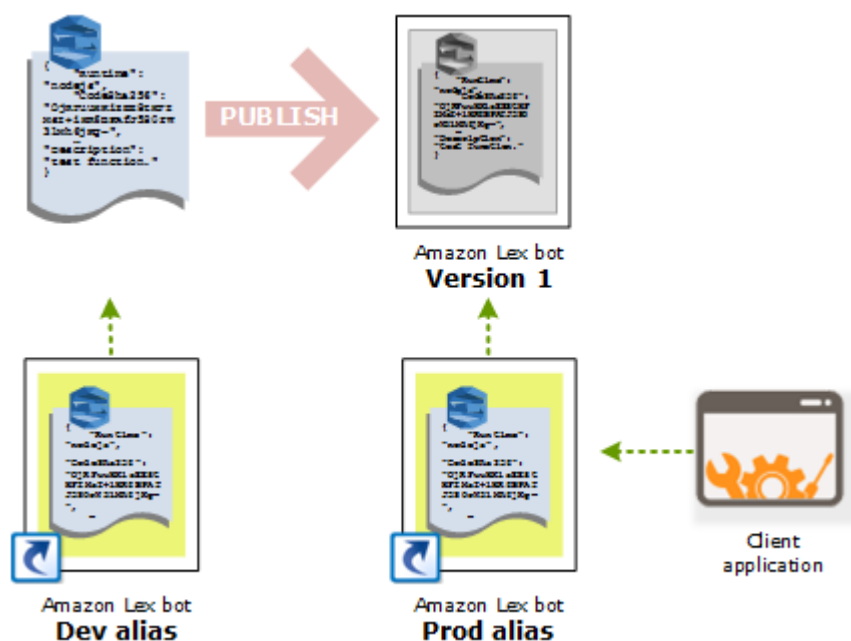
O Amazon Lex oferece suporte à exclusão de um recurso ou versão usando o console ou uma das operações da API:

- [DeleteBot](#) (p. 234)
- [DeleteBotVersion](#) (p. 240)
- [DeleteBotAlias](#) (p. 236)
- [DeleteBotChannelAssociation](#) (p. 238)
- [DeleteIntent](#) (p. 242)
- [DeleteIntentVersion](#) (p. 244)
- [DeleteSlotType](#) (p. 246)
- [DeleteSlotTypeVersion](#) (p. 248)

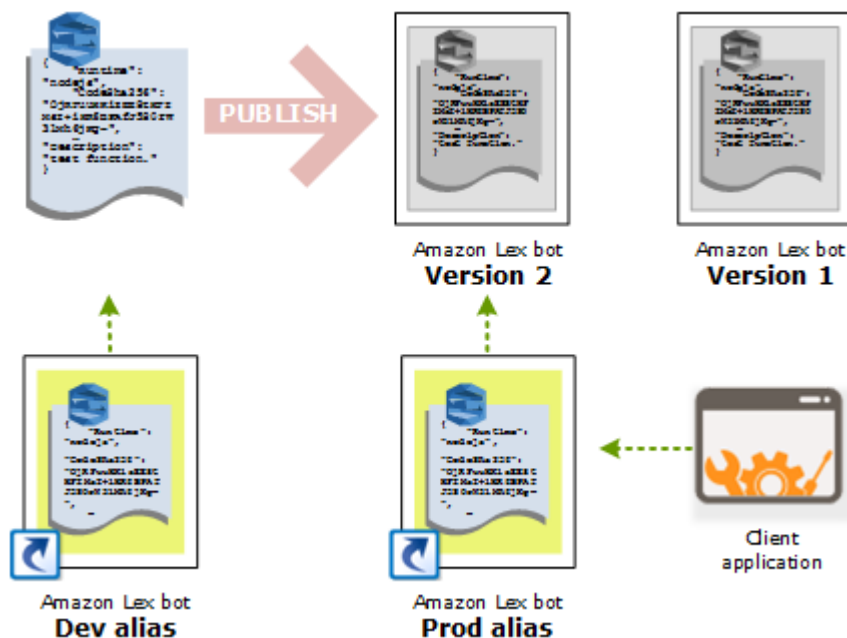
## Aliases

Um alias é um ponteiro para uma versão específica de um bot do Amazon Lex. Use um alias para permitir que os aplicativos de clientes usem uma versão específica do bot sem exigir que o aplicativo acompanhe qual é a versão.

O exemplo a seguir mostra duas versões de um bot do Amazon Lex, versão `$LATEST` e versão 1. Cada uma dessas versões de bot tem um alias associados, DEV e PROD, respectivamente. As aplicações dos clientes usam o alias PROD para acessar o bot.



Ao criar uma segunda versão do bot, você pode atualizar o alias para apontar para a nova versão do bot usando o console ou a operação [PutBot](#) (p. 311). Quando você altera o alias, todas as aplicações de seus clientes usam a nova versão. Se houver um problema com a nova versão, você poderá reverter para a versão anterior, simplesmente alterando o alias para apontar para essa versão.



#### Note

Embora você possa testar a versão `$LATEST` de um bot no console, recomendamos que, ao integrar um bot ao aplicativo do cliente, primeiro você publique uma versão e crie um alias que aponta para essa versão. Use o alias na aplicação de seus clientes pelos motivos explicados nesta seção. Quando você atualiza um alias, o Amazon Lex aguardará até que o tempo limite de sessão de todas as sessões atuais expire para começar a usar a nova versão. Para obter mais informações sobre tempo limite da sessão, consulte [the section called "Definição do tempo limite da sessão"](#) (p. 23)

# Uso de funções do Lambda

Você pode criar funções do AWS Lambda para usar como ganchos de código para seu bot do Amazon Lex. Você pode identificar funções do Lambda para executar a inicialização e a validação, o atendimento ou ambos na configuração da intenção.

Recomendamos usar uma função do Lambda como um gancho de código para seu bot. Sem uma função do Lambda, seu bot retorna as informações da intenção ao aplicativo cliente para atendimento.

## Tópicos

- [Evento de entrada de função do Lambda e formato de resposta \(p. 106\)](#)
- [Esquemas do Amazon Lex e do AWS Lambda \(p. 113\)](#)

## Evento de entrada de função do Lambda e formato de resposta

Esta seção descreve a estrutura dos dados de eventos que o Amazon Lex fornece para uma função do Lambda. Use essas informações para analisar a entrada em seu código do Lambda. Também explica o formato da resposta que o Amazon Lex espera que sua função do Lambda retorne.

## Tópicos

- [Formato de eventos de entrada \(p. 106\)](#)
- [Formato de resposta \(p. 109\)](#)

## Formato de eventos de entrada

O seguinte é o formato geral de um evento do Amazon Lex que é passado para uma função do Lambda. Use essas informações ao criar sua função do Lambda.

### Note

O formato de entrada pode mudar sem uma alteração correspondente em `messageVersion`. Seu código não deve gerar um erro se novos campos estiverem presentes.

```
{
  "currentIntent": {
    "name": "intent-name",
    "slots": {
      "slot name": "value",
      "slot name": "value"
    },
  },
  "slotDetails": {
    "slot name": {
      "resolutions" : [
        { "value": "resolved value" },
        { "value": "resolved value" }
      ],
      "originalValue": "original text"
    },
    "slot name": {
      "resolutions" : [
        { "value": "resolved value" },
        { "value": "resolved value" }
      ],
    },
  },
}
```

```
        "originalValue": "original text"
      }
    },
    "confirmationStatus": "None, Confirmed, or Denied (intent confirmation, if configured)"
  },
  "bot": {
    "name": "bot name",
    "alias": "bot alias",
    "version": "bot version"
  },
  "userId": "User ID specified in the POST request to Amazon Lex.",
  "inputTranscript": "Text used to process the request",
  "invocationSource": "FulfillmentCodeHook or DialogCodeHook",
  "outputDialogMode": "Text or Voice, based on ContentType request header in runtime API request",
  "messageVersion": "1.0",
  "sessionAttributes": {
    "key": "value",
    "key": "value"
  },
  "requestAttributes": {
    "key": "value",
    "key": "value"
  }
}
```

Observe as seguintes informações adicionais sobre os campos de evento:

- `currentIntent` – fornece os campos `name`, `slots`, `slotDetails` e `confirmationStatus` da intenção.

`slots` é um mapa de nomes de slots, configurado para a intenção, com valores de slot que o Amazon Lex reconheceu na conversa do usuário. Um valor de slot permanece nulo até o usuário fornecer um valor.

O valor de slot no evento de entrada pode não corresponder com um dos valores configurados para o slot. Por exemplo, se o usuário responder à solicitação "What color car would you like? (Você quer um carro de que cor?)" com "pizza", o Amazon Lex retornará "pizza" como o valor do slot. Sua função deve validar os valores de forma que façam sentido no contexto.

`slotDetails` fornece informações adicionais sobre um valor de slot. O arranjo `resolutions` contém uma lista de valores adicionais reconhecidos para o slot. Cada slot pode ter no máximo cinco valores.

O campo `originalValue` contém o valor que foi inserido pelo usuário para o slot. Quando o tipo de slot é configurado para retornar o primeiro valor de resolução como valor de slot, o `originalValue` pode ser diferente do valor no campo `slots`.

`confirmationStatus` fornece ao usuário resposta a um prompt de confirmação, se houver. Por exemplo, se o Amazon Lex perguntar "Do you want to order a large cheese pizza? (Você quer encomendar uma pizza grande de queijo?)", dependendo da resposta do usuário, o valor desse campo poderá ser `Confirmed` ou `Denied`. Caso contrário, o valor desse campo é `None`.



Se o usuário confirmar a intenção, o Amazon Lex definirá esse campo como `Confirmed`. Se o usuário negar a intenção, o Amazon Lex definirá esse valor como `Denied`.

Na resposta de confirmação, o utterance de um usuário pode fornecer atualizações de slot. Por exemplo, o usuário pode dizer "sim, mude o tamanho para médio". Nesse caso, o evento subsequente do Lambda terá o valor de slot atualizado, `PizzaSize` definido como `medium`. O Amazon Lex define o `confirmationStatus` como `None`, porque o usuário modificou alguns dados do slot, exigindo que a função do Lambda executasse a validação dos dados do usuário.

- `bot` – informações sobre o bot que processou a solicitação.
  - `name` – o nome do bot que processou a solicitação.
  - `alias` – o alias da versão do bot que processou a solicitação.
  - `version` – a versão do bot que processou a solicitação.
- `userId` – esse valor é fornecido pelo aplicativo cliente. O Amazon Lex o passa para a função do Lambda.
- `inputTranscript` – o texto usado para processar a solicitação.

Se a entrada foi um texto, o campo `inputTranscript` contém o texto que foi inserido pelo usuário.

Se a entrada foi um fluxo de áudio, o campo `inputTranscript` contém o texto extraído do fluxo de áudio. Esse é o texto que é processado para reconhecer as intenções e os valores de slot.

- `invocationSource` – para indicar por que o Amazon Lex está invocando a função Lambda, ele define isso como um dos seguintes valores:
  - `DialogCodeHook` – o Amazon Lex define esse valor para direcionar a função do Lambda para inicializar a função e validar a entrada de dados do usuário.

Quando a intenção é configurada para invocar uma função do Lambda como um gancho de código de inicialização e validação, o Amazon Lex invoca a função do Lambda especificada em cada entrada do usuário (enunciado) após o Amazon Lex entender a intenção.

#### Note

Se a intenção não estiver clara, o Amazon Lex não poderá invocar a função do Lambda.

- `FulfillmentCodeHook` – o Amazon Lex define esse valor para direcionar a função do Lambda para atender a uma intenção.

Se a intenção estiver configurada para invocar uma função do Lambda como um gancho de código de atendimento, o Amazon Lex definirá o `invocationSource` como esse valor somente depois de ter todos os dados de slot para atender à intenção.

Em sua configuração de intenção, você pode ter duas funções do Lambda separadas para inicializar e validar os dados do usuário e para atender à intenção. Você também pode usar uma função do Lambda para fazer as duas coisas. Nesse caso, sua função do Lambda pode usar o valor de `invocationSource` para seguir o caminho do código correto.

- `outputDialogMode` – para cada entrada do usuário, o cliente envia a solicitação ao Amazon Lex usando uma das operações da API de tempo de execução, [PostContent](#) (p. 347) ou [PostText](#) (p. 355). O Amazon Lex usa os parâmetros da solicitação Amazon Lex para determinar se a resposta ao cliente é por texto ou voz e define esse campo de forma adequada.

A função do Lambda pode usar essas informações para gerar uma mensagem apropriada. Por exemplo, se o cliente esperar uma resposta de voz, a função do Lambda poderá retornar Speech Synthesis Markup Language (SSML) em vez de texto.

- `messageVersion` – a versão da mensagem que identifica o formato dos dados de evento que estão indo para a função do Lambda e o formato esperado da resposta de uma função do Lambda.

#### Note

Você configura esse valor ao definir uma intenção. Na implementação atual, apenas a versão de mensagem 1.0 é suportada. Portanto, o console assume o valor padrão de 1.0 e não mostra a versão da mensagem.

- `sessionAttributes` – atributos de sessão específicos ao aplicativo que o cliente envia na solicitação. Se você quiser que o Amazon Lex os inclua na resposta ao cliente, sua função do Lambda deverá enviá-los de volta ao Amazon Lex na resposta. Para obter mais informações, consulte [Definição dos atributos da sessão](#) (p. 20)
- `requestAttributes` – atributos de sessão específicos à solicitação que o cliente envia na solicitação. Use atributos de solicitação para passar informações que não precisam ser mantidas durante toda a sessão. Se não houver atributos de solicitação, o valor será nulo. Para obter mais informações, consulte [Definição de atributos de solicitação](#) (p. 21)

## Formato de resposta

O Amazon Lex espera uma resposta de uma função do Lambda no seguinte formato:

```
{
  "sessionAttributes": {
    "key1": "value1",
    "key2": "value2"
    ...
  },
  "dialogAction": {
    "type": "ElicitIntent, ElicitSlot, ConfirmIntent, Delegate, or Close",
    Full structure based on the type field. See below for details.
  }
}
```

A resposta consiste em dois campos. O campo `sessionAttributes` é opcional, o campo `dialogAction` é obrigatório. O conteúdo do campo `dialogAction` depende do valor do `type` field. Para obter mais informações, consulte [dialogAction](#) (p. 110).

## sessionAttributes

Opcional. Se você incluir o campo `sessionAttributes`, ele pode ficar vazio. Se a função do Lambda não retornar os atributos da sessão, o último `sessionAttributes` conhecido transmitido pela API ou pela função do Lambda permanecerá. Para obter mais informações, consulte as operações [PostContent](#) (p. 347) e [PostText](#) (p. 355).

```
"sessionAttributes": {  
  "key1": "value1",  
  "key2": "value2"  
}
```

## dialogAction

Obrigatório. O campo `dialogAction` direciona o Amazon Lex para a próxima ação e descreve o que esperar do usuário depois que o Amazon Lex retornar uma resposta ao cliente.

O campo `type` indica a próxima ação. Ele também determina de quais outros campos a função do Lambda precisa fornecer como parte do valor de `dialogAction`.

- **Close** — informa ao Amazon Lex para não esperar uma resposta do usuário. Por exemplo, "Seu pedido de pizza foi feito" não requer uma resposta.

O `fulfillmentState` é obrigatório. O Amazon Lex usa esse valor para definir o campo `dialogState` na resposta [PostContent](#) (p. 347) ou [PostText](#) (p. 355) ao aplicativo cliente. Os campos `message` e `responseCard` são opcionais. Se você não especificar uma mensagem, o Amazon Lex usará a mensagem de despedida ou a mensagem de acompanhamento configurada para a intenção.

```
"dialogAction": {  
  "type": "Close",  
  "fulfillmentState": "Fulfilled or Failed",  
  "message": {  
    "contentType": "PlainText or SSML or CustomPayload",  
    "content": "Message to convey to the user. For example, Thanks, your pizza has been ordered."  
  },  
  "responseCard": {  
    "version": integer-value,  
    "contentType": "application/vnd.amazonaws.card.generic",  
    "genericAttachments": [  
      {  
        "title": "card-title",  
        "subTitle": "card-sub-title",  
        "imageUrl": "URL of the image to be shown",  
        "attachmentLinkUrl": "URL of the attachment to be associated with the card",  
        "buttons": [  
          {  
            "text": "button-text",  
            "value": "Value sent to server on button click"  
          }  
        ]  
      }  
    ]  
  }  
}
```

```
}
```

- **ConfirmIntent** — informa ao Amazon Lex que o usuário deve fornecer uma resposta Sim ou Não para confirmar ou negar a intenção atual.

Você deve incluir os campos `intentName` e `slots`. O campo `slots` deve conter uma entrada para cada um dos slots preenchidos para a intenção especificada. Não é necessário incluir uma entrada no campo `slots` para slots que não estão preenchidos. Você deve incluir o campo `message` se o campo `confirmationPrompt` de intenção for nulo. O conteúdo do campo `message` retornado pela função do Lambda tem precedência sobre o `confirmationPrompt` especificado na intenção. O campo `responseCard` é opcional.

```
"dialogAction": {
  "type": "ConfirmIntent",
  "message": {
    "contentType": "PlainText or SSML or CustomPayload",
    "content": "Message to convey to the user. For example, Are you sure you want a
large pizza?"
  },
  "intentName": "intent-name",
  "slots": {
    "slot-name": "value",
    "slot-name": "value",
    "slot-name": "value"
  },
  "responseCard": {
    "version": integer-value,
    "contentType": "application/vnd.amazonaws.card.generic",
    "genericAttachments": [
      {
        "title": "card-title",
        "subTitle": "card-sub-title",
        "imageUrl": "URL of the image to be shown",
        "attachmentLinkUrl": "URL of the attachment to be associated with the card",
        "buttons": [
          {
            "text": "button-text",
            "value": "Value sent to server on button click"
          }
        ]
      }
    ]
  }
}
```

- **Delegate** — direciona o Amazon Lex para escolher a próxima ação de acordo com a configuração do bot. Se a resposta não incluir nenhum atributo de sessão, o Amazon Lex manterá os atributos existentes. Se você deseja que o valor de um slot seja nulo, não é necessário incluir o campo do slot na solicitação. Você obterá uma exceção `DependencyFailedException` se sua função de cumprimento retornar a ação de diálogo `Delegate` sem remover nenhum slot.

```
"dialogAction": {
  "type": "Delegate",
  "slots": {
    "slot-name": "value",
    "slot-name": "value",
    "slot-name": "value"
  }
}
```

- **ElicitIntent** — informa ao Amazon Lex que o usuário deve responder com um enunciado que inclua uma intenção. Por exemplo, "Eu quero uma pizza grande", que indica a `OrderPizzaIntent`. O enunciado "large (grande)", por outro lado, não é suficiente para que o Amazon Lex infira a intenção do usuário.

Os campos `message` e `responseCard` são opcionais. Se você não fornecer uma mensagem, o Amazon Lex usará uma das solicitações de esclarecimento do bot.

```
{
  "dialogAction": {
    "type": "ElicitIntent",
    "message": {
      "contentType": "PlainText or SSML or CustomPayload",
      "content": "Message to convey to the user. For example, What can I help you with?"
    },
    "responseCard": {
      "version": integer-value,
      "contentType": "application/vnd.amazonaws.card.generic",
      "genericAttachments": [
        {
          "title": "card-title",
          "subTitle": "card-sub-title",
          "imageUrl": "URL of the image to be shown",
          "attachmentLinkUrl": "URL of the attachment to be associated with the card",
          "buttons": [
            {
              "text": "button-text",
              "value": "Value sent to server on button click"
            }
          ]
        }
      ]
    }
  }
}
```

- **ElicitSlot** — informa ao Amazon Lex que o usuário deve fornecer um valor de slot na resposta.

Os campos `intentName`, `slotToElicit` e `slots` são obrigatórios. Os campos `message` e `responseCard` são opcionais. Se você não especificar uma mensagem, o Amazon Lex usará uma das solicitações de seleção de valor de slot configuradas para o slot.

```
"dialogAction": {
  "type": "ElicitSlot",
  "message": {
    "contentType": "PlainText or SSML or CustomPayload",
    "content": "Message to convey to the user. For example, What size pizza would you like?"
  },
  "intentName": "intent-name",
  "slots": {
    "slot-name": "value",
    "slot-name": "value",
    "slot-name": "value"
  },
  "slotToElicit": "slot-name",
  "responseCard": {
    "version": integer-value,
    "contentType": "application/vnd.amazonaws.card.generic",
```

```
"genericAttachments": [
  {
    "title": "card-title",
    "subTitle": "card-sub-title",
    "imageUrl": "URL of the image to be shown",
    "attachmentLinkUrl": "URL of the attachment to be associated with the card",
    "buttons": [
      {
        "text": "button-text",
        "value": "Value sent to server on button click"
      }
    ]
  }
]
```

## Esquemas do Amazon Lex e do AWS Lambda

O console do Amazon Lex fornece bots de exemplo (chamados de esquemas de bot) pré-configurados para que você possa criar e testar rapidamente um bot no console. Para cada um desses esquemas de bot, também são fornecidos esquemas da função do Lambda. Esses esquemas fornecem um código de exemplo que funciona com os bots correspondentes. Com esses esquemas, você pode criar rapidamente um bot configurado com uma função do Lambda como um gancho de código e testar a configuração completa sem precisar escrever nenhum código.

Você pode usar os seguintes esquemas de bot do Amazon Lex e os esquemas das funções do AWS Lambda correspondentes como ganchos de código para os bots:

- Esquema do Amazon Lex — OrderFlowers
  - Esquemas do AWS Lambda — `lex-order-flowers` (código Node.js) e `lex-order-flowers-python`
- Esquema do Amazon Lex — ScheduleAppointment
  - Esquemas do AWS Lambda — `lex-make-appointment` (código Node.js) e `lex-make-appointment-python`
- Esquema do Amazon Lex — BookTrip
  - Esquemas do AWS Lambda — `lex-book-trip` (código Node.js) e `lex-book-trip-python`

Para criar um bot usando um esquema e configurá-lo para usar uma função do Lambda como um gancho de código, consulte [Exercício 1: Criar um bot do Amazon Lex usando um esquema \(console\) \(p. 38\)](#). Para obter um exemplo do uso de outros esquemas, consulte [Exemplos adicionais: criação de bots do Amazon Lex \(p. 135\)](#).

# Implantação de bots do Amazon Lex

Esta seção fornece exemplos de implantação de bots do Amazon Lex em várias plataformas de sistema de mensagens e em aplicativos para dispositivos móveis.

## Tópicos

- [Implantação de um bot do Amazon Lex em uma plataforma de sistema de mensagens \(p. 114\)](#)
- [Implantação de um bot do Amazon Lex em aplicativos móveis \(p. 128\)](#)

## Implantação de um bot do Amazon Lex em uma plataforma de sistema de mensagens

Esta seção explica como implantar bots do Amazon Lex nas plataformas de sistema de mensagens Facebook, Slack e Twilio.

### Note

Ao armazenar suas configurações do Facebook, do Slack ou do Twilio, o Amazon Lex usa chaves mestras de cliente (CMK) do AWS Key Management Service para criptografar as informações. Na primeira vez que você cria um canal para uma das seguintes plataformas de sistema de mensagens, o Amazon Lex cria uma CMK padrão (`aws/lex`). Como alternativa, você pode criar sua própria CMK com o AWS KMS. Isso lhe dá mais flexibilidade, incluindo a capacidade de criar, girar e desabilitar as chaves. Você também pode definir controles de acesso e auditar as chaves de criptografia usadas para proteger seus dados. Para obter mais informações, consulte [Guia do Desenvolvedor do AWS Key Management Service](#).

Quando uma plataforma de sistema de mensagens envia uma solicitação ao Amazon Lex, informações específicas à plataforma são incluídas como um atributo de solicitação para a função do Lambda. Use esses atributos para personalizar a forma como o seu bot se comporta. Para obter mais informações, consulte [Definição de atributos de solicitação \(p. 21\)](#).

Todos os atributos levam o namespace `x-amz-lex:` como prefixo. Por exemplo, o atributo `user-id` é chamado `x-amz-lex:user-id`. Existem atributos comuns que são enviados por todas as plataformas de mensagens, além de outros que são específicos de uma determinada plataforma. A tabela a seguir lista os atributos de solicitação que as plataformas de sistema de mensagens enviam à função do Lambda do bot.

### Atributos de solicitação comuns

| Atributo                  | Descrição   |
|---------------------------|---|
| <code>channel-id</code>   | O identificador do endpoint do canal do Amazon Lex.   |
| <code>channel-name</code> | O nome do canal do Amazon Lex.  |
| <code>channel-type</code> | Um dos seguintes valores: <ul style="list-style-type: none"><li>• <code>Facebook</code></li><li>• <code>Kik</code></li><li>• <code>Slack</code></li></ul> |

| Atributo             | Descrição  |
|----------------------|--|
|                      | <ul style="list-style-type: none"><li>• Twilio-SMS</li></ul> |
| webhook-endpoint-url | O endpoint do Amazon Lex para o canal.                       |

#### Atributos de solicitação do Facebook

| Atributo         | Descrição   |
|------------------|---|
| user-id          | O identificador do Facebook do remetente. Consulte <a href="https://developers.facebook.com/docs/messenger-platform/webhook-reference/message-received">https://developers.facebook.com/docs/messenger-platform/webhook-reference/message-received</a> .              |
| facebook-page-id | O identificador da página do Facebook do destinatário. Consulte <a href="https://developers.facebook.com/docs/messenger-platform/webhook-reference/message-received">https://developers.facebook.com/docs/messenger-platform/webhook-reference/message-received</a> . |

#### Atributos de solicitação do Kik

| Atributo         | Descrição   |
|------------------|---|
| kik-chat-id      | O identificador para a conversa no qual o bot está envolvido. Para obter mais informações, consulte <a href="https://dev.kik.com/#/docs/messaging#message-formats">https://dev.kik.com/#/docs/messaging#message-formats</a> . |
| kik-chat-type    | O tipo de conversa do qual a mensagem foi originada. Para obter mais informações, consulte <a href="https://dev.kik.com/#/docs/messaging#message-formats">https://dev.kik.com/#/docs/messaging#message-formats</a> .          |
| kik-message-id   | Um UUID que identifica a mensagem. Para obter mais informações, consulte <a href="https://dev.kik.com/#/docs/messaging#message-formats">https://dev.kik.com/#/docs/messaging#message-formats</a> .                            |
| kik-message-type | O tipo de mensagem. Para obter mais informações, consulte <a href="https://dev.kik.com/#/docs/messaging#message-types">https://dev.kik.com/#/docs/messaging#message-types</a> .   |

#### Atributos de solicitação do Twilio

| Atributo                   | Descrição   |
|----------------------------|---|
| user-id                    | O número de telefone do remetente ("De"). Consulte <a href="https://www.twilio.com/docs/api/rest/message">https://www.twilio.com/docs/api/rest/message</a> .      |
| twilio-target-phone-number | O número de telefone do destinatário ("Para"). Consulte <a href="https://www.twilio.com/docs/api/rest/message">https://www.twilio.com/docs/api/rest/message</a> . |

#### Atributos de solicitação do Slack

| Atributo      | Descrição   |
|---------------|---|
| user-id       | O identificador do usuário do Slack. Consulte <a href="https://api.slack.com/types/user">https://api.slack.com/types/user</a> .                           |
| slack-team-id | O identificador da equipe que enviou a mensagem. Consulte <a href="https://api.slack.com/methods/team.info">https://api.slack.com/methods/team.info</a> . |



| Atributo        | Descrição   |
|-----------------|---|
| slack-bot-token | O token de desenvolvedor que concede ao bot acesso às APIs do Slack. Consulte <a href="https://api.slack.com/docs/token-types">https://api.slack.com/docs/token-types</a> . |

## Integração de um bot do Amazon Lex com o Facebook Messenger

### Tópicos

- [Etapa 1: Criar um bot do Amazon Lex \(p. 116\)](#)
- [Etapa 2: crie um aplicativo do Facebook \(p. 116\)](#)
- [Etapa 3: Integrar o Facebook Messenger a seu bot do Amazon Lex \(p. 116\)](#)
- [Etapa 4: teste a integração \(p. 118\)](#)

Este exercício mostra como integrar o Facebook Messenger a seu bot do Amazon Lex. Você executa as seguintes etapas:

1. Criar um bot do Amazon Lex
2. Criação de um aplicativo do Facebook
3. Para integrar o Facebook Messenger a seu bot do Amazon Lex
4. Validação da integração

### Etapa 1: Criar um bot do Amazon Lex

Se você ainda não tem um bot do Amazon Lex, crie e implante um. Neste tópico, pressupomos que você esteja usando o bot que criou no Exercício 1 dos Conceitos básicos. No entanto, você pode usar qualquer um dos bots de exemplo fornecidos neste guia. Para o Exercício 1 dos Conceitos básicos, consulte [Exercício 1: Criar um bot do Amazon Lex usando um esquema \(console\) \(p. 38\)](#).

1. Crie um bot do Amazon Lex. Para obter instruções, consulte [Exercício 1: Criar um bot do Amazon Lex usando um esquema \(console\) \(p. 38\)](#).
2. Implante o bot e crie um alias. Para obter instruções, consulte [Exercício 3: publique uma versão e crie um alias \(p. 75\)](#).

### Etapa 2: crie um aplicativo do Facebook

No portal de desenvolvedor do Facebook, crie um aplicativo do Facebook e uma página do Facebook. Para obter instruções, consulte [Início rápido](#) na documentação da plataforma Facebook Messenger. Anote o seguinte:

- O App Secret para o aplicativo do Facebook
- O Page Access Token para a página do Facebook

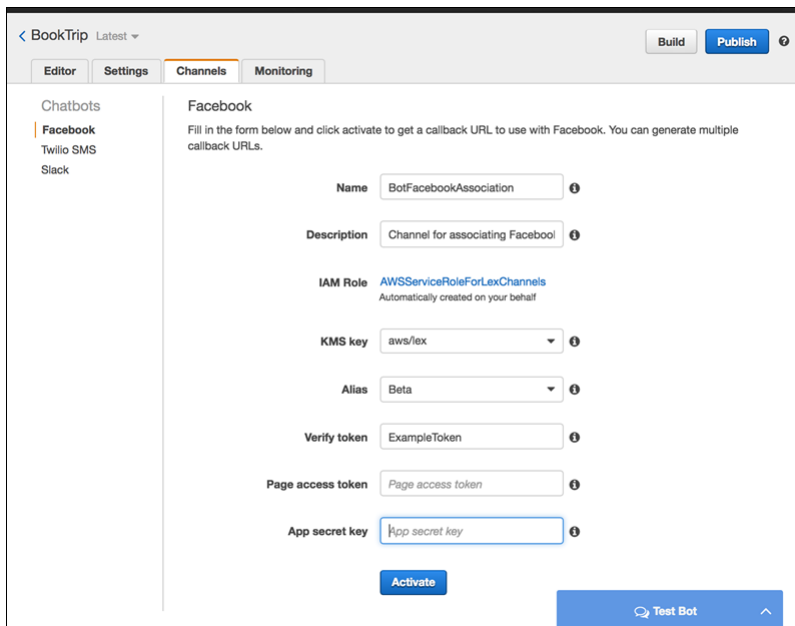
### Etapa 3: Integrar o Facebook Messenger a seu bot do Amazon Lex

Nesta seção, você integra o Facebook Messenger a seu bot do Amazon Lex.

Depois de concluir essa etapa, o console fornece um URL de retorno de chamada. Anote esse URL.

Para integrar o Facebook Messenger ao seu bot

1.
  - a. Faça login no Console de gerenciamento da AWS e abra o console do Amazon Lex em <https://console.aws.amazon.com/lex/>.
  - b. Escolha o bot do Amazon Lex.
  - c. Escolha Channels.
  - d. Escolha Facebook em Chatbots. O console exibe a página de integração do Facebook.
  - e. Na página de integração do Facebook:
    - Digite este nome: BotFacebookAssociation.
    - Para KMS key, escolha aws/lex.
    - Para Alias, escolha o alias bot.
    - Para Verify token, digite um token. Pode ser qualquer string que você escolher (por exemplo, ExampleToken). Use esse token posteriormente no portal de desenvolvedor do Facebook ao configurar o webhook.
    - Para acessar Page access token, digite o token que você obteve do Facebook na Etapa 2.
    - Para App secret key, digite a chave que você obteve do Facebook na Etapa 2.



- f. Selecione Ativar.

O console cria a associação de canal de bot e retorna um URL de retorno de chamada. Anote esse URL.

2. No portal de desenvolvedor do Facebook, escolha seu aplicativo.
3. Escolha o produto Messenger e selecione Setup webhooks na seção Webhook da página.

Para obter instruções, consulte [Início rápido](#) na documentação da plataforma Facebook Messenger.

4. Na página webhook do assistente de assinatura:
  - Em Callback URL (URL de retorno de chamada), digite a URL de retorno de chamada fornecida no console do Amazon Lex anteriormente no procedimento.

- Em Verify Token (Verificar token), digite o mesmo token usado no Amazon Lex.
  - Escolha Subscription Fields (messages, messaging\_postbacks e messaging\_optins).
  - Escolha Verify and Save. Isso inicia um handshake entre o Facebook e o Amazon Lex.
5. Ative a integração do Webhooks. Escolha a página que você criou e, em seguida, escolha subscribe.

#### Note

Se você atualizar ou recriar um webhook, deverá cancelar a assinatura e, em seguida, assinar a página novamente.

## Etapa 4: teste a integração

Agora, você pode começar uma conversa no Facebook Messenger com seu bot do Amazon Lex.

1. Abra a página do Facebook e escolha Message.
2. Na janela do Messenger, use os mesmos enunciados de teste fornecidos no [Etapa 1: Criar um bot do Amazon Lex \(console\)](#) (p. 39).

## Integração de um bot do Amazon Lex com o Kik

#### Tópicos

- [Etapa 1: Criar um bot do Amazon Lex](#) (p. 118)
- [Etapa 2: criar um bot do Kik](#) (p. 119)
- [Etapa 3: Integrar o bot do Kik com o bot do Amazon Lex](#) (p. 119)
- [Etapa 4: teste a integração](#) (p. 120)

Este exercício fornece instruções para integrar um bot do Amazon Lex com o aplicativo de sistema de mensagens Kik. Você executa as seguintes etapas:

1. Crie um bot do Amazon Lex.
2. Crie um bot do Kik usando o aplicativo e o site do sistema.
3. Integre o bot do Amazon Lex com o bot do Kik usando o console do Amazon Lex.
4. Inicie uma conversa com o bot do Amazon Lex usando o Kik para testar a associação entre o bot do Amazon Lex e o Kik.

## Etapa 1: Criar um bot do Amazon Lex

Se você ainda não tem um bot do Amazon Lex, crie e implante um. Neste tópico, pressupomos que você esteja usando o bot que criou no Exercício 1 dos Conceitos básicos. No entanto, você pode usar qualquer um dos bots de exemplo fornecidos neste guia. Para o Exercício 1 dos Conceitos básicos, consulte [Exercício 1: Criar um bot do Amazon Lex usando um esquema \(console\)](#) (p. 38)

1. Crie um bot do Amazon Lex. Para obter instruções, consulte [Exercício 1: Criar um bot do Amazon Lex usando um esquema \(console\)](#) (p. 38).
2. Implante o bot e crie um alias. Para obter instruções, consulte [Exercício 3: publique uma versão e crie um alias](#) (p. 75).

#### Próxima etapa

[Etapa 2: criar um bot do Kik](#) (p. 119)

## Etapa 2: criar um bot do Kik

Nesta etapa, use a interface de usuário do Kik para criar um bot. Use as informações geradas ao criar o bot para conectá-lo ao bot do Amazon Lex.

1. Se você ainda não tiver feito isso, baixe e instale o aplicativo do Kik e cadastre-se para criar uma conta. Se você já tem uma conta, faça login.
2. Acesse o site do Kik em <https://dev.kik.com/>. Deixe a janela do navegador aberta.
3. No aplicativo do Kik, selecione o ícone de engrenagem para abrir as configurações e, em seguida, selecione Your Kik Code.
4. Digitalize o código no site do Kik para abrir o chatbot do Botsworth. Selecione Yes para abrir o painel de bot.
5. No aplicativo do Kik, selecione Create a Bot. Siga as solicitações para criar o bot do Kik.
6. Assim que o bot for criado, selecione Configuration no seu navegador. Verifique se o novo bot está selecionado.
7. Anote o nome do bot e a chave de API para a próxima seção.

Próxima etapa

[Etapa 3: Integrar o bot do Kik com o bot do Amazon Lex \(p. 119\)](#)

## Etapa 3: Integrar o bot do Kik com o bot do Amazon Lex

Agora que criou os bots do Amazon Lex e do Kik, você está pronto para criar uma associação de canal entre eles no Amazon Lex. Quando a associação for ativada, o Amazon Lex configurará automaticamente uma URL de retorno de chamada com o Kik.

1. Faça login no Console de Gerenciamento da AWS e abra o console do Amazon Lex em <https://console.aws.amazon.com/lex/>.
2. Selecione o bot do Amazon Lex que você criou na Etapa 1.
3. Escolha a guia Channels.
4. Na seção Channels, selecione Kik.
5. Na página do Kik, insira as informações a seguir:
  - Digite um nome. Por exemplo, BotKikIntegration.
  - Digite uma descrição.
  - No menu suspenso KMS key, escolha "aws/lex".
  - Em Alias, selecione um alias da lista suspensa.
  - Em Kik bot user name, digite o nome que você deu ao bot no Kik.
  - Em Kik API key, digite a chave de API que foi atribuída ao bot no Kik.
  - Em User greeting, digite a saudação que você deseja que o bot envie na primeira vez em que um usuário iniciar um bate-papo com ele.
  - Em Error message, insira uma mensagem de erro que é exibida ao usuário quando parte da conversa não é compreendida.
  - Em Group chat behavior, selecione uma das opções:
    - Enable (Habilitar) – permite que todo o grupo de bate-papo interaja com o bot em uma única conversa.
    - Disable (Desabilitar) – restringe a conversa a um usuário do grupo de bate-papo.

### Kik

Fill in the form below and click activate to get a callback URL to use with Kik. You can generate multiple callback URLs. [Learn more](#) on steps to integrate with Kik.

Channel Name\*

KikBotIntegration

Channel Description

Integrate an Amazon Lex bot with Kik

IAM Role

[AWSServiceRoleForLexChannels](#)  
Automatically created on your behalf

KMS key

aws/lex

Alias\*

BETA

Kik Bot User Name\*

XXXXXXXX

Kik API Key\*

XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXX

User Greeting\*

Welcome to my first Amazon Lex bot on Kik

#### Advanced configuration

Error Message\*

There seems to be a problem.

Group Chat Behavior

☐ Enable

☒ Disable

\* Required Field

Activate

- Selecione Activate para criar a associação e vinculá-la ao bot do Kik.

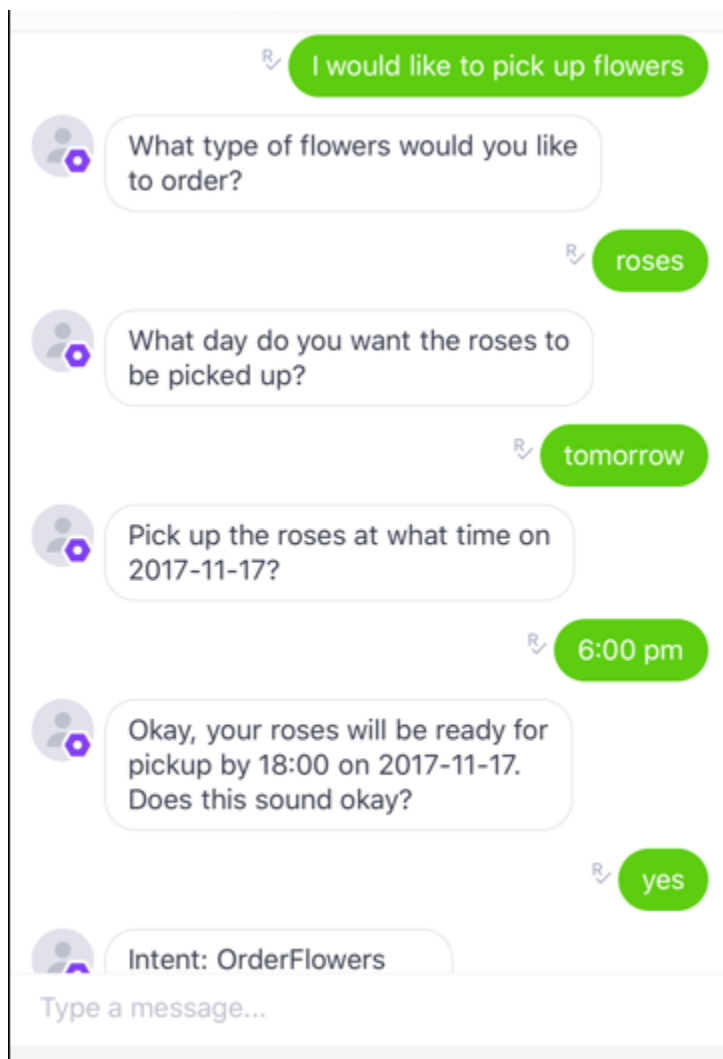
Próxima etapa

[Etapa 4: teste a integração \(p. 120\)](#)

## Etapa 4: teste a integração

Agora que você criou uma associação entre o bot do Amazon Lex e o do Kik, use o aplicativo do Kik para testá-la.

1. Inicie o aplicativo do Kik e faça login. Selecione o bot que você criou.
2. Teste o bot da seguinte forma:



Ao inserir cada frase, o bot do Amazon Lex responderá por meio do Kik com a solicitação que você criou para cada slot.

## Integração de um bot do Amazon Lex com o Slack

### Tópicos

- [Etapa 1: Criar um bot do Amazon Lex \(p. 122\)](#)
- [Etapa 2: cadastre-se no Slack e crie uma equipe do Slack \(p. 122\)](#)
- [Etapa 3: crie uma aplicação do Slack \(p. 122\)](#)
- [Etapa 4: Integrar o aplicativo do Slack com o bot do Amazon Lex \(p. 123\)](#)
- [Etapa 5: Completar a integração do Slack \(p. 124\)](#)
- [Etapa 6: teste a integração \(p. 125\)](#)

Este exercício fornece instruções para integrar um bot do Amazon Lex com o aplicativo de sistema de mensagens Slack. Você executa as seguintes etapas:

1. Crie um bot do Amazon Lex.
2. Crie um aplicativo de mensagem do Slack
3. Integre o aplicativo do Slack ao bot do Amazon Lex.
4. Teste a integração iniciando uma conversa com seu bot do Amazon Lex. Você envia mensagens com o aplicativo do Slack e testa em uma janela do navegador.

## Etapa 1: Criar um bot do Amazon Lex

Se você ainda não tem um bot do Amazon Lex, crie e implante um. Neste tópico, pressupomos que você esteja usando o bot que criou no Exercício 1 dos Conceitos básicos. No entanto, você pode usar qualquer um dos bots de exemplo fornecidos neste guia. Para o Exercício 1 dos Conceitos básicos, consulte [Exercício 1: Criar um bot do Amazon Lex usando um esquema \(console\) \(p. 38\)](#)

1. Crie um bot do Amazon Lex. Para obter instruções, consulte [Exercício 1: Criar um bot do Amazon Lex usando um esquema \(console\) \(p. 38\)](#).
2. Implante o bot e crie um alias. Para obter instruções, consulte [Exercício 3: publique uma versão e crie um alias \(p. 75\)](#).

Próxima etapa

[Etapa 2: cadastre-se no Slack e crie uma equipe do Slack \(p. 122\)](#)

## Etapa 2: cadastre-se no Slack e crie uma equipe do Slack

Cadastre-se em uma conta do Slack e crie uma equipe do Slack. Para obter instruções, consulte [Uso do Slack](#). Na próxima seção, você criará uma aplicação do Slack que qualquer equipe do Slack pode instalar.

Próxima etapa

[Etapa 3: crie uma aplicação do Slack \(p. 122\)](#)

## Etapa 3: crie uma aplicação do Slack

Nesta seção, faça o seguinte:

1. Crie uma aplicação do Slack no console da API do Slack
2. Configure o aplicativo para adicionar os seguintes recursos ao seu bot:
  - Um usuário de bot
  - Mensagens interativas

No final desta seção, você obterá as credenciais do aplicativo (ID do cliente, segredo do cliente e token de verificação). Na próxima seção, você usará essas informações para configurar a associação do canal do bot no console do Amazon Lex.

1. Faça login no console da API do Slack em [http://api.slack.com API](http://api.slack.com/API).
2. Crie uma aplicação.

Depois de que você cria o aplicativo com êxito, o Slack exibe a página Basic Information do aplicativo.

3. Configure os recursos da aplicação da seguinte forma:
  - a. No menu esquerdo, selecione Bot Users (Usuários de bot) e Add User (Adicionar usuário).

- Forneça um nome de exibição e um nome de usuário padrão.
  - Para Always Show My Bot as Online, escolha On.
  - Para salvar as alterações, escolha Add Bot User.
- b. No menu esquerdo, escolha Interactive Components (Componentes interativos).
- Selecione a alternância na qual executar os componentes interativos.
  - Na caixa Request URL especifique qualquer URL válido. Por exemplo, você pode usar o **`https://slack.com`**.
- Note
- Por enquanto, insira qualquer URL válido para obter o token de verificação necessário na próxima etapa. Você atualizará essa URL depois de adicionar a associação de canal do bot no console do Amazon Lex.
- Selecione Save Changes (Salvar alterações).
4. No menu esquerdo, em Settings, escolha Basic Information. Registre as seguintes credenciais do aplicativo:
- ID do cliente
  - Segredo do cliente
  - Token de verificação

Próxima etapa

[Etapa 4: Integrar o aplicativo do Slack com o bot do Amazon Lex \(p. 123\)](#)

## Etapa 4: Integrar o aplicativo do Slack com o bot do Amazon Lex

Agora que tem as credenciais do aplicativo do Slack, você pode integrar o aplicativo com o bot do Amazon Lex. Para associar o aplicativo do Slack a seu bot, adicione uma associação de canal do bot no Amazon Lex.

No console do Amazon Lex, ative uma associação de canal do bot para associar o bot ao aplicativo do Slack. Quando a associação de canal do bot está ativada, o Amazon Lex retorna duas URLs (URL do Postback e URL do OAuth). Anote esses URLs, pois serão necessários posteriormente.

Para integrar o aplicativo do Slack ao bot do Amazon Lex

1. Faça login no Console de Gerenciamento da AWS e abra o console do Amazon Lex em <https://console.aws.amazon.com/lex/>.
2. Selecione o bot do Amazon Lex que você criou na Etapa 1.
3. Escolha a guia Channels.
4. No menu esquerdo, selecione Slack.
5. Na página Slack, forneça os valores a seguir:
  - Digite um nome. Por exemplo, BotSlackIntegration.
  - No menu suspenso KMS key, escolha "aws/lex".
  - Para Alias, escolha o alias bot.
  - Digite o Client Id, o Client secrete o Verification Token, que você registrou na etapa anterior. Essas são as credenciais da aplicação do Slack.



The screenshot shows the 'Channels' tab in the Amazon Lex console for a bot named 'BookTrip'. The 'Slack' channel configuration is active. The form contains the following fields:

- Name:** BotSlackAssociation
- Description:** Channel for Slack
- IAM Role:** AWSRoleForLexChannels (Automatically created on your behalf)
- KMS key:** aws/lex
- Alias:** Beta
- Client Id:** Client Id
- Client secret:** Client secret
- Verification Token:** Verification Token
- Success Page URL:** Success Page URL

At the bottom right, there is a 'Test Bot' button.

6. Selecione Ativar.

O console cria a associação de canal do bot e retorna dois URLs (Postback URL e OAuth URL). Anote-os. Na próxima seção, você atualizará a configuração da sua aplicação do Slack para usar esses endpoints conforme o seguinte:

- A URL do Postback é o endpoint do bot do Amazon Lex que ouve aos eventos do Slack. Você usa este URL:
  - Como o URL de solicitação no recurso Event Subscriptions do aplicativo do Slack.
  - Para substituir o valor do espaço reservado para o URL solicitado no recurso Interactive Messages do aplicativo do Slack.
- A URL do OAuth é o endpoint do bot do Amazon Lex para um handshake do OAuth com o Slack.

Próxima etapa

[Etapa 5: Completar a integração do Slack \(p. 124\)](#)

## Etapa 5: Completar a integração do Slack

Nesta seção, use o console da API do Slack para completar a integração do aplicativo do Slack.

1. Faça login no console da API do Slack em <http://api.slack.com/API>. Selecione o aplicativo que você criou em [Etapa 3: crie uma aplicação do Slack \(p. 122\)](#).
2. Atualize o recurso OAuth & Permissions da seguinte forma:
  - a. No menu esquerdo, selecione OAuth & Permissions (OAuth e permissões).
  - b. Na seção Redirect URLs (URLs de redirecionamento), adicione a URL do OAuth fornecida pelo Amazon Lex na etapa anterior. Escolha Add a new Redirect URL e, em seguida, escolha Save URLs.
  - c. Na seção Scopes, escolha duas permissões no menu suspenso Select Permission Scopes. Filtre a lista com o seguinte texto:

- **chat:write:bot**
- **team:read**

Selecione Save Changes (Salvar alterações).

3. Atualize o recurso Interactive Components (Componentes interativos) atualizando o valor Request URL (URL da solicitação) para a URL do Postback que o Amazon Lex forneceu na etapa anterior. Insira o URL Postback que você salvou na etapa 4 e selecione Save Changes (Salvar alterações).
4. Assine o recurso Event Subscriptions da seguinte forma:
  - Habilite eventos escolhendo a opção On.
  - Defina o valor da Request URL (URL da solicitação) como a URL do Postback que o Amazon Lex forneceu na etapa anterior.
  - Na seção Subscribe to Bot Events, inscreva-se no evento de bot message.im para habilitar mensagens diretas entre o usuário final e o bot do Slack.
  - Salve as alterações.

Próxima etapa

[Etapa 6: teste a integração \(p. 125\)](#)

## Etapa 6: teste a integração

Agora, use uma janela do navegador para testar a integração do Slack com o bot do Amazon Lex.

1. Escolha Manage Distribution em Settings. Escolha Add to Slack para instalar o aplicativo. Autorize o bot a responder a mensagens.
2. Você é redirecionado para sua equipe do Slack. No menu esquerdo, na seção Mensagens diretas, escolha o bot. Se você não vir seu bot, escolha o ícone de mais (+) ao lado de Mensagens diretas para procurá-lo.
3. Inicie um bate-papo com o aplicativo do Slack, que está vinculado ao bot do Amazon Lex. Seu bot agora responde a mensagens.

Se você criou o bot usando o Exercício 1 dos Conceitos básicos, pode usar as conversas de exemplo fornecidas no exercício. Para obter mais informações, consulte [Etapa 4: Adicionar a função do Lambda como gancho de código \(console\) \(p. 51\)](#).

## Integração de um bot do Amazon Lex com o SMS programável do Twilio

Tópicos

- [Etapa 1: Criar um bot do Amazon Lex \(p. 126\)](#)
- [Etapa 2: Crie uma conta de SMS do Twilio \(p. 126\)](#)
- [Etapa 3: Integrar o endpoint do serviço de sistema de mensagens do Twilio com o bot do Amazon Lex \(p. 126\)](#)
- [Etapa 4: teste a integração \(p. 127\)](#)

Este exercício fornece instruções para integrar um bot do Amazon Lex com o serviço de mensagens simples (SMS) do Twilio. Você executa as seguintes etapas:

1. Criar um bot do Amazon Lex
2. Integrar o SMS programável do Twilio com seu bot do Amazon Lex
3. Iniciar uma interação com o bot do Amazon Lex e testar a configuração usando o serviço SMS no celular
4. Teste da integração

## Etapa 1: Criar um bot do Amazon Lex

Se você ainda não tem um bot do Amazon Lex, crie e implante um. Neste tópico, pressupomos que você esteja usando o bot que criou no Exercício 1 dos Conceitos básicos. No entanto, você pode usar qualquer um dos bots de exemplo fornecidos neste guia. Para o Exercício 1 dos Conceitos básicos, consulte [Exercício 1: Criar um bot do Amazon Lex usando um esquema \(console\) \(p. 38\)](#).

1. Crie um bot do Amazon Lex. Para obter instruções, consulte [Exercício 1: Criar um bot do Amazon Lex usando um esquema \(console\) \(p. 38\)](#).
2. Implante o bot e crie um alias. Para obter instruções, consulte [Exercício 3: publique uma versão e crie um alias \(p. 75\)](#).

## Etapa 2: Crie uma conta de SMS do Twilio

Cadastre-se para criar uma conta do Twilio e registre as seguintes informações de conta:

- ACCOUNT SID
- AUTH TOKEN

Para obter instruções de cadastro, consulte <https://www.twilio.com/console>.

## Etapa 3: Integrar o endpoint do serviço de sistema de mensagens do Twilio com o bot do Amazon Lex

Para integrar o Twilio com o bot do Amazon Lex

1. Para associar o bot do Amazon Lex com o endpoint de SMS programável do Twilio, ative a associação de canal do bot no console do Amazon Lex. Quando a associação de canal do bot estiver ativada, o Amazon Lex retornará uma URL de retorno de chamada. Anote o URL de retorno de chamada, pois você precisará dele mais tarde.
  - a. Faça login no Console de gerenciamento da AWS e abra o console do Amazon Lex em <https://console.aws.amazon.com/lex/>.
  - b. Selecione o bot do Amazon Lex que você criou na Etapa 1.
  - c. Escolha a guia Channels.
  - d. Na seção Chatbots, escolha Twilio SMS.
  - e. Na página Twilio SMS, forneça as seguintes informações:
    - Digite um nome. Por exemplo, BotTwilioAssociation.
    - Escolha "aws/lex" em KMS key.
    - Para Alias, escolha o alias bot.
    - Para Authentication Token, digite o TOKEN DE AUTORIZAÇÃO para sua conta do Twilio.
    - Em Account SID, digite o SID DA CONTA do Twilio.

The screenshot shows the Amazon Lex console interface for configuring a Twilio SMS channel. The 'Channels' tab is selected. On the left, there's a sidebar with 'Chatbots' and 'Twilio SMS' highlighted. The main area contains a form for 'Twilio SMS' with the following fields: 'Name' (BotTwilioAssociation), 'Description' (Channel for Twilio), 'IAM Role' (AWSRoleForLexChannels), 'KMS key' (aws/lex), 'Alias' (Beta), 'Authentication Token', and 'Account SID'. An 'Activate' button is located below the 'Account SID' field. At the bottom, there's a 'Callback URLs' section and a 'Test Bot' button.

- f. Selecione Ativar.

O console cria a associação de canal de bot e retorna um URL de retorno de chamada. Anote o URL.

2. No console do Twilio, conecte o endpoint de SMS do Twilio ao bot do Amazon Lex.
  - a. Faça login no console do Twilio em <https://www.twilio.com/console>.
  - b. Se você não tiver um endpoint de SMS do Twilio, crie um.
  - c. Atualize a configuração Inbound Settings (Configurações de entrada) do serviço de sistema de mensagens definindo o valor REQUEST URL (URL DE SOLICITAÇÃO) como a URL de retorno de chamada que o Amazon Lex forneceu na etapa anterior.

## Etapa 4: teste a integração

Use seu celular para testar a integração entre o SMS do Twilio e o bot.

Como testar a integração

1. Faça login no console do Twilio em <https://www.twilio.com/console> e faça o seguinte:
  - a. Verifique se você tem um número do Twilio associado ao serviço de mensagens em Manage Numbers.

Envie mensagens para esse número e inicie uma interação por SMS com o bot do Amazon Lex no celular.
  - b. Verifique se seu telefone celular está na lista branca como Verified Caller ID.

Se não estiver, siga as instruções no console do Twilio para permitir o celular que você planeja usar para testes.

Agora, você pode usar o celular para enviar mensagens ao endpoint de SMS do Twilio, que é mapeado para o bot do Amazon Lex.

2. Usando seu celular, envie mensagens para o número do Twilio.

O bot do Amazon Lex responde. Se você criou o bot usando o Exercício 1 dos Conceitos básicos, pode usar as conversas de exemplo fornecidas no exercício. Para obter mais informações, consulte [Etapa 4: Adicionar a função do Lambda como gancho de código \(console\)](#) (p. 51).

## Implantação de um bot do Amazon Lex em aplicativos móveis

Usando os SDKs da AWS, você pode integrar o bot do Amazon Lex com seus aplicativos móveis. Para obter mais informações, consulte [Adicionar bots de conversa ao aplicativo móvel com o Amazon Lex](#) no Guia do desenvolvedor do AWS Mobile.

Você também pode usar o AWS Mobile Hub para criar um aplicativo móvel quickstart que demonstre o uso do SDK do Amazon Lex em aplicativos móveis iOS e Android. Para obter mais informações, consulte [Bots de conversa do AWS Mobile Hub](#).

# Importação e exportação de bots, intenções e tipos de slot do Amazon Lex

Você pode importar ou exportar um bot, uma intenção ou um tipo de slot. Por exemplo, se quiser compartilhar um bot com um colega em uma conta da AWS diferente, você poderá exportá-lo e, em seguida, enviá-lo para ele. Se quiser adicionar vários enunciados a um bot, você pode exportá-lo, adicionar os enunciados e reimportá-lo para a conta.

Você pode exportar bots, intenções e tipos de slot no Amazon Lex (para compartilhar ou modificá-los) ou em um formato de habilidade do Alexa. Você só pode importar em formato do Amazon Lex.

Ao exportar um recurso, você precisa exportá-lo em um formato que seja compatível com o serviço para o qual está exportando, o Amazon Lex ou o Alexa Skills Kit. Se exportar um bot no formato do Amazon Lex, você poderá reimportá-lo para a conta, ou um usuário do Amazon Lex em outra conta poderá importá-lo para a respectiva conta. Você também pode exportar um bot em um formato compatível com uma habilidade do Alexa. Em seguida, você poderá importar o bot usando o Alexa Skills Kit para disponibilizá-lo com o Alexa. Para obter mais informações, consulte [Exportar para uma habilidade do Alexa \(p. 134\)](#).

Quando você exporta um bot, uma intenção ou um tipo de slot, os recursos são gravados em um arquivo JSON. Para exportar um bot, uma intenção ou um tipo de slot, você pode usar o console do Amazon Lex ou a operação [GetExport \(p. 282\)](#). Importe um bot, uma intenção ou um tipo de slot usando o [StartImport \(p. 338\)](#).

## Tópicos

- [Exportação e importação em formato do Amazon Lex \(p. 129\)](#)
- [Exportar para uma habilidade do Alexa \(p. 134\)](#)

## Exportação e importação em formato do Amazon Lex

Para exportar bots, intenções e tipos de slot do Amazon Lex com a intenção de reimportá-los no Amazon Lex, você cria um arquivo JSON no formato do Amazon Lex. Você pode editar os recursos neste arquivo e reimportá-los no Amazon Lex. Por exemplo, você pode adicionar enunciados a uma intenção e reimportar a intenção alterada para a conta. Você também pode usar o formato JSON para compartilhar um recurso. Por exemplo, você pode exportar um bot de uma região da AWS e, em seguida, importá-lo para outra região. Ou você pode enviar o arquivo JSON a um colega para compartilhar um bot.

## Tópicos

- [Exportação em formato do Amazon Lex \(p. 130\)](#)
- [Importação em formato do Amazon Lex \(p. 130\)](#)
- [Formato JSON para importação e exportação \(p. 131\)](#)

## Exportação em formato do Amazon Lex

Exporte os bots, as intenções e os tipos de slot do Amazon Lex para um formato que você possa importar para uma conta da AWS. Você pode exportar os seguintes recursos:

- Um bot, inclusive todas as intenções e os tipos de slot personalizados usados pelo bot
- Uma intenção, inclusive todos os tipos de slot personalizados usados pela intenção
- Um tipo de slot personalizado, inclusive todos os valores do tipo de slot

Você só pode exportar uma versão numerada de um recurso. Você não pode exportar uma versão `$LATEST` do recurso.

A exportação é um processo assíncrono. Quando a exportação é concluída, você obtém uma pre-signed URL do Amazon S3. O URL fornece o local de um arquivo .zip que contém o recurso exportado em formato JSON.

Você usa o console ou a operação [GetExport](#) (p. 282) para exportar bots, intenções e tipos de slot personalizados.

O processo para exportar um bot, uma intenção ou um tipo de slot é o mesmo. Nos procedimentos a seguir, substitua a intenção ou o tipo de slot do bot.

### Exportar um bot

Para exportar um bot

1. Faça login no Console de Gerenciamento da AWS e abra o console do Amazon Lex em <https://console.aws.amazon.com/lex/>.
2. Escolha Bots e o bot a ser exportado.
3. No menu Ações, escolha Export (Exportar).
4. Na caixa de diálogo Export Bot (Exportar bot), escolha a versão do bot a ser exportada. Em Plataforma, escolha Amazon Lex.
5. Escolha Export (Exportar).
6. Faça download e salve o arquivo .zip.

O Amazon Lex exporta o bot para um arquivo JSON contido no arquivo .zip. Para atualizar o bot, modifique o texto JSON e reimporte-o no Amazon Lex.

Próxima etapa

[Importação em formato do Amazon Lex](#) (p. 130)

## Importação em formato do Amazon Lex

Depois de exportar um recurso para um arquivo JSON no formato do Amazon Lex, você pode importar o arquivo JSON que contém o recurso em uma ou mais contas da AWS. Por exemplo, você pode exportar um bot e, em seguida, importá-lo para outra região da AWS. Ou você pode enviar o bot para uma colega, de maneira que ela possa importá-lo para a respectiva conta.

Ao importar um bot, uma intenção ou um tipo de slot, você deve decidir se deseja substituir a versão `$LATEST` de um recurso, como uma intenção ou um tipo de slot, durante a importação, ou se deseja que a importação falhe caso queira preservar o recurso que está na conta. Por exemplo, se estivesse fazendo upload de uma versão editada de um recurso para a conta, você optaria por substituir a versão `$LATEST`. Se estiver fazendo upload de um recurso enviado por um colega, você poderá optar por fazer a importação falhar se houver conflitos de recurso. Dessa maneira, os próprios recursos não são substituídos.

Durante a importação de um recurso, as permissões atribuídas ao usuário que está fazendo a solicitação de importação se aplicam. O usuário deve ter permissões para todos os recursos na conta afetada pela importação. O usuário também deve ter permissão para as operações [GetBot](#) (p. 252), [PutBot](#) (p. 311), [GetIntent](#) (p. 288), [PutIntent](#) (p. 323), [GetSlotType](#) (p. 299), [PutSlotType](#) (p. 333). Para obter mais informações sobre permissões, consulte [Como o Amazon Lex funciona com o IAM](#) (p. 188).

A importação relata erros ocorridos durante o processamento. Alguns erros são relatados antes do início da importação, outros são relatados durante o processo de importação. Por exemplo, se a conta que estiver importando uma intenção não tiver permissão para chamar uma função do Lambda usada pela intenção, a importação falhará antes das alterações serem feitas nos tipos de slot ou nas intenções. Se uma importação falhar durante o processo de importação, a versão `$LATEST` de qualquer intenção ou tipo de slot importado antes do processo ter falhado será modificada. Você não pode reverter alterações feitas na versão `$LATEST`.

Quando você importa um recurso, todos os recursos dependentes são importados para a versão `$LATEST` do recurso e, em seguida, recebem uma versão numerada. Por exemplo, se um bot usar uma intenção, ela receberá uma versão numerada. Se uma intenção usar um tipo de slot personalizado, ele receberá uma versão numerada.

Um recurso é importado somente uma vez. Por exemplo, se o bot contiver uma intenção `OrderPizza` e uma intenção `OrderDrink` em que ambas dependam do tipo de slot personalizado `Size`, o tipo de slot `Size` será importado uma vez e usado em ambas as intenções.

O processo para importar um bot, uma intenção ou um tipo de slot personalizado é o mesmo. Nos procedimentos a seguir, substitua a intenção ou o tipo de slot, conforme apropriado.

## Importar um bot

Para importar um bot

1. Faça login no Console de Gerenciamento da AWS e abra o console do Amazon Lex em <https://console.aws.amazon.com/lex/>.
2. Escolha Bots e o bot a ser importado. Para importar um novo bot, ignore esta etapa.
3. Em Ações, escolha Import (Importar).
4. Em Import Bot (Importar bot), escolha o arquivo .zip que contém o arquivo JSON que contém o bot a ser importado. Se você quiser ver conflitos de mesclagem antes de mesclar, escolha Notify me of merge conflicts (Notificar-me de conflitos de mesclagem). Se você desativar a verificação de conflitos, a versão `$LATEST` de todos os recursos usados pelo bot será substituída.
5. Escolha Import. Se você tiver optado por ser notificado sobre conflitos de mesclagem e houver conflitos, será exibida uma caixa de diálogo os listando. Para substituir a versão `$LATEST` de todos os recursos conflitantes, escolha Overwrite and continue (Substituir e continuar). Para interromper a importação, escolha Cancelar.

Você já pode testar o bot na conta.

## Formato JSON para importação e exportação

Os exemplos a seguir mostram a estrutura JSON para exportar e importar tipos de slot, intenções e bots em formato do Amazon Lex.

### Estrutura do tipo de slot

Esta é a estrutura JSON para tipos de slot personalizados. Use essa estrutura ao importar ou exportar tipos de slot, além de exportar intenções que dependam de tipos de slot personalizados.

```
{
```



```
"metadata": {
  "schemaVersion": "1.0",
  "importType": "LEX",
  "importFormat": "JSON"
},
"resource": {
  "name": "slot type name",
  "version": "version number",
  "enumerationValues": [
    {
      "value": "enumeration value",
      "synonyms": []
    },
    {
      "value": "enumeration value",
      "synonyms": []
    }
  ],
  "valueSelectionStrategy": "ORIGINAL_VALUE or TOP_RESOLUTION"
}
}
```

## Estrutura da intenção

Esta é a estrutura JSON para intenções. Use essa estrutura ao importar ou exportar intenções e bots que dependam de uma intenção.

```
{
  "metadata": {
    "schemaVersion": "1.0",
    "importType": "LEX",
    "importFormat": "JSON"
  },
  "resource": {
    "description": "intent description",
    "rejectionStatement": {
      "messages": [
        {
          "contentType": "PlainText or SSML or CustomPayload",
          "content": "string"
        }
      ]
    },
    "name": "intent name",
    "version": "version number",
    "fulfillmentActivity": {
      "type": "ReturnIntent or CodeHook"
    },
    "sampleUtterances": [
      "string",
      "string"
    ],
    "slots": [
      {
        "name": "slot name",
        "description": "slot description",
        "slotConstraint": "Required or Optional",
        "slotType": "slot type",
        "valueElicitationPrompt": {
          "messages": [
            {
              "contentType": "PlainText or SSML or CustomPayload",
              "content": "string"
            }
          ]
        }
      }
    ]
  }
}
```

```
    ],
    "maxAttempts": value
  },
  "priority": value,
  "sampleUtterances": []
}
],
"confirmationPrompt": {
  "messages": [
    {
      "contentType": "PlainText or SSML or CustomPayload",
      "content": "string"
    },
    {
      "contentType": "PlainText or SSML or CustomPayload",
      "content": "string"
    }
  ]
},
"maxAttempts": 2
},
"slotTypes": [
  List of slot type JSON structures.
  For more information, see Estrutura do tipo de slot.
]
}
}
```

## Estrutura do bot

Esta é a estrutura JSON para bots. Use essa estrutura ao importar ou exportar bots.

```
{
  "metadata": {
    "schemaVersion": "1.0",
    "importType": "LEX",
    "importFormat": "JSON"
  },
  "resource": {
    "name": "bot name",
    "version": "version number",
    "intents": [
      List of intent JSON structures.
      For more information, see Estrutura da intenção.
    ],
    "slotTypes": [
      List of slot type JSON structures.
      For more information, see Estrutura do tipo de slot.
    ],
    "voiceId": "output voice ID",
    "childDirected": boolean,
    "locale": "en-US",
    "idleSessionTTLInSeconds": timeout,
    "description": "bot description",
    "clarificationPrompt": {
      "messages": [
        {
          "contentType": "PlainText or SSML or CustomPayload",
          "content": "string"
        }
      ]
    },
    "maxAttempts": value
  },
  "abortStatement": {
    "messages": [
```

```
{
  "contentType": "PlainText or SSML or CustomPayload",
  "content": "string"
}
]
```

## Exportar para uma habilidade do Alexa

Exporte seu esquema de bot em um formato compatível com uma habilidade do Alexa. Depois de exportar o bot para um arquivo JSON, você poderá fazer upload dele para o Alexa usando o criador de habilidades.

Para exportar um bot e o esquema (modelo de interação)

1. Faça login no Console de gerenciamento da AWS e abra o console do Amazon Lex em <https://console.aws.amazon.com/lex/>.
2. Selecione o bot que você deseja exportar.
3. Em Ações, escolha Export (Exportar).
4. Escolha a versão do bot que você deseja exportar. Para o formato, escolha Alexa Skills Kit e Export (Exportar).
5. Se a caixa de diálogo de download for exibida, escolha um local para salvar o arquivo e selecione Save (Salvar).

O arquivo obtido por download é um arquivo .zip contendo um arquivo com o nome do bot exportado. Ele contém as informações necessárias para importar o bot como uma habilidade do Alexa.

### Note

O Amazon Lex e o Alexa Skills Kit são diferentes nos seguintes aspectos:

- Os atributos de sessão, indicados por colchetes ([ ]), não são compatíveis com o Alexa Skills Kit. Você precisa atualizar solicitações que usem atributos de sessão.
- Sinais de pontuação não são compatíveis com o Alexa Skills Kit. Você precisa atualizar enunciados que usem pontuação.

Para fazer upload do bot para uma habilidade do Alexa

1. Faça login no portal do desenvolvedor em <https://developer.amazon.com/>.
2. Na página Alexa Skills (Skills da Alexa), selecione Create Skill (Criar skill).
3. Na página Create a new skill (Criar uma skill), insira um nome de skill e a linguagem padrão para a skill. Verifique se a opção Custom (Personalizar) está selecionada para o modelo de skill e selecione Create skill (Criar skill).
4. Verifique se a opção Start from scratch (Começar do zero) está selecionada e selecione Choose (Escolher).
5. No menu à esquerda, selecione JSON Editor (Editor JSON). Arraste o arquivo JSON exportado do Amazon Lex para o editor JSON.
6. Selecione Save Model (Salvar modelo) para salvar o modelo de interação.

Depois de fazer upload do esquema para a habilidade do Alexa, faça alterações necessárias para executar a habilidade com o Alexa. Para obter mais informações sobre como criar um recurso do Alexa, consulte [Usar o Skill Builder \(Beta\)](#) no Alexa Skills Kit.

# Exemplos adicionais: criação de bots do Amazon Lex

As seções a seguir fornecem exercícios adicionais de Amazon Lex com instruções detalhadas.

## Tópicos

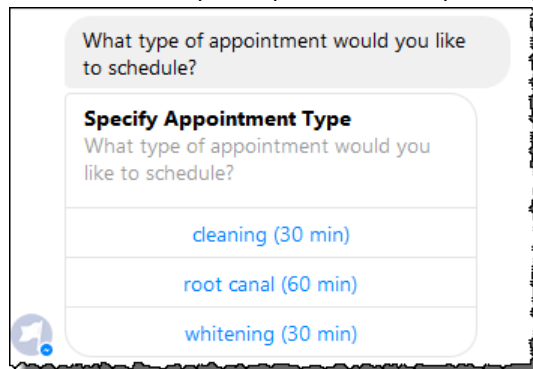
- [Exemplo de bot: ScheduleAppointment \(p. 135\)](#)
- [Exemplo de bot: BookTrip \(p. 154\)](#)
- [Exemplo: uso de um cartão de resposta \(p. 176\)](#)
- [Exemplo: Atualização de enunciados \(p. 180\)](#)
- [Exemplo: Integração com um site \(p. 181\)](#)

## Exemplo de bot: ScheduleAppointment

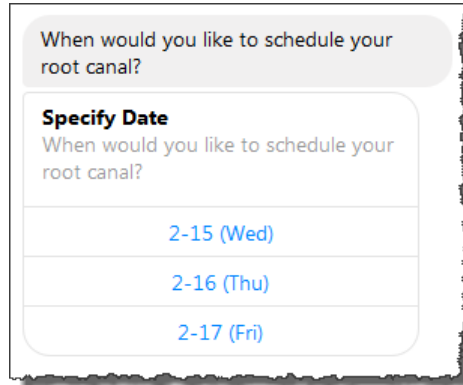
O exemplo de bot neste exercício marca consultas em uma clínica odontológica. O exemplo também ilustra o uso de cartões de resposta para obter entradas do usuário com botões. Especificamente, o exemplo ilustra a geração dinâmica de cartões de resposta em runtime.

Você pode configurar cartões de resposta em tempo de criação (também chamados de cartões de resposta estáticos) ou gerá-los dinamicamente em uma função do AWS Lambda. Neste exemplo, o bot usa os seguintes cartões de resposta:

- Um cartão de resposta que lista botões para o tipo de consulta. Por exemplo:



- Um cartão de resposta que lista botões para a data de consulta. Por exemplo:



When would you like to schedule your root canal?

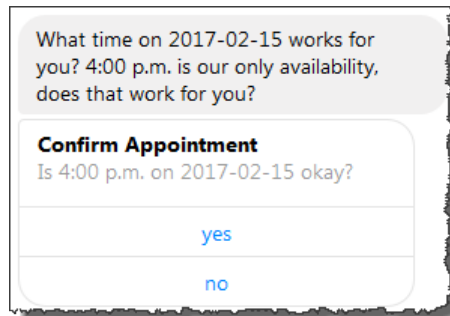
**Specify Date**  
When would you like to schedule your root canal?

2-15 (Wed)

2-16 (Thu)

2-17 (Fri)

- Um cartão de resposta que lista botões para confirmar um horário de consulta sugerido. Por exemplo:



What time on 2017-02-15 works for you? 4:00 p.m. is our only availability, does that work for you?

**Confirm Appointment**  
Is 4:00 p.m. on 2017-02-15 okay?

yes

no

As datas e os horários de consulta disponíveis variam, o que exige que você gere cartões de resposta em runtime. Você usa uma função do AWS Lambda para gerar dinamicamente esses cartões de resposta. A função do Lambda retorna cartões de resposta em sua resposta ao Amazon Lex. O Amazon Lex inclui o cartão de resposta em sua resposta ao cliente.

Se um cliente (por exemplo, Facebook Messenger) oferecer suporte a cartões de resposta, o usuário poderá escolher na lista de botões ou digitar a resposta. Caso contrário, o usuário simplesmente digitará a resposta.

Além do botão mostrado no exemplo anterior, você também pode incluir imagens, anexos e outras informações úteis a serem exibidas nos cartões de resposta. Para obter mais informações sobre cartões de resposta, consulte [Cartões de resposta](#) (p. 16).

Neste exercício, você faz o seguinte:

- Criação e teste de um bot (usando o esquema ScheduleAppointment). Para este exercício, você usará um esquema de bot para configurar e testar o bot rapidamente. Para obter uma lista de esquemas disponíveis, consulte [Esquemas do Amazon Lex e do AWS Lambda](#) (p. 113). Este bot está pré-configurado com uma intenção (MakeAppointment).
- Crie e teste uma função do Lambda (usando o esquema lex-make-appointment-python fornecido pelo Lambda). Você configura a intenção MakeAppointment para usar essa função do Lambda como um gancho de código para executar a inicialização, a validação e o atendimento de tarefas.

#### Note

A função do Lambda de exemplo fornecida exibe uma conversa dinâmica baseada na disponibilidade inventada de uma consulta odontológica. Em uma aplicação real, você pode usar um calendário real para definir uma consulta.

- Atualize a configuração da intenção `MakeAppointment` para usar a função do Lambda como um gancho de código. Em seguida, teste a experiência completa.
- Publique o bot de agendamento de consultas no Facebook Messenger para você possa ver os cartões de resposta em ação (atualmente, o cliente no console do Amazon Lex não oferece suporte a cartões de resposta).

As seções a seguir fornecem informações resumidas sobre os esquemas usados por você neste exercício.

#### Tópicos

- [Visão geral do esquema de bot \(ScheduleAppointment\) \(p. 137\)](#)
- [Visão geral do esquema da função do Lambda \(lex-make-appointment-python\) \(p. 138\)](#)
- [Etapa 1: Criar um bot do Amazon Lex \(p. 138\)](#)
- [Etapa 2: Criar uma função do Lambda \(p. 140\)](#)
- [Etapa 3: atualizar a intenção - configuração de um hook de código \(p. 141\)](#)
- [Etapa 4: implantar o bot na plataforma do Facebook Messenger \(p. 142\)](#)
- [Detalhes do fluxo de informações \(p. 142\)](#)

## Visão geral do esquema de bot (ScheduleAppointment)

O esquema `ScheduleAppointment` usado para criar um bot para este exercício está pré-configurado com o seguinte:

- Tipos de slot – um tipo de slot personalizado chamado `AppointmentTypeValue`, com os valores de enumeração `root`, `canal`, `cleaning` e `whitening`.
- Intenção – uma intenção (`MakeAppointment`), que está pré-configurada da seguinte forma:
  - Slots – a intenção está configurado com os seguintes slots:
    - Slot `AppointmentType`, do tipo personalizado `AppointmentTypes`.
    - Slot `Date`, do tipo de slot integrado `AMAZON.DATE`.
    - Slot `Time`, do tipo de slot integrado `AMAZON.TIME`.
  - Enunciados – a intenção está pré-configurada com os seguintes enunciados:
    - "Eu gostaria de marcar uma consulta"
    - "Marcar uma consulta"
    - "Marcar um {AppointmentType}"

Se o usuário enunciar qualquer um desses, o Amazon Lex determinará que `MakeAppointment` é a intenção e, em seguida, usará as solicitações para escolher os dados do slot.

- Solicitações – a intenção é pré-configurada com as seguintes solicitações:
  - Solicitação do slot `AppointmentType` – "What type of appointment would you like to schedule? (Que tipo de consulta você deseja agendar?)"
  - Solicitação do slot `Date` – "When should I schedule your {AppointmentType}? (Quando devo agendar seu {AppointmentType}?)"
  - Solicitação do slot `Time` – "At what time do you want to schedule the {AppointmentType}? (Em que horário você deseja agendar o {AppointmentType}?)"
- "Em que horário em {Date}?"
- Solicitação de confirmação – "{Time} is available, should I go ahead and book your appointment? ({Time} está disponível. Posso prosseguir e agendar sua consulta?)"

- Mensagem de cancelamento – "Okay, I will not schedule an appointment. (OK, não marcarei a consulta.)"

## Visão geral do esquema da função do Lambda (lex-make-appointment-python)

O esquema da função do Lambda (lex-make-appointment-python) é um gancho de código para bots que você cria usando o esquema de bot ScheduleAppointment.

Esse esquema de código da função do Lambda pode executar inicialização/validação e atendimento de tarefas.

- A função do Lambda exibe uma conversa dinâmica baseada em um exemplo de disponibilidade para uma consulta odontológica (em aplicativos reais, você pode usar um calendário). Para o dia ou a data que o usuário especificar, o código será configurado da seguinte forma:
  - Se não houver consultas disponíveis, a função do Lambda retornará uma resposta direcionando o Amazon Lex a solicitar outro dia ou data ao usuário (definindo o tipo `dialogAction` como `ElicitSlot`). Para obter mais informações, consulte [Formato de resposta](#) (p. 109).
  - Se houver apenas uma consulta disponível no dia ou data especificada, a função do Lambda sugerirá o horário disponível na resposta e direcionará o Amazon Lex a obter a confirmação do usuário definindo o `dialogAction` na resposta como `ConfirmIntent`. Isso ilustra como você pode melhorar a experiência do usuário sugerindo proativamente o horário disponível para uma consulta.
  - Se houver várias consultas disponíveis, a função do Lambda retornará uma lista dos horários disponíveis na resposta ao Amazon Lex. O Amazon Lex retorna uma resposta ao cliente com a mensagem da função do Lambda.
- Como o gancho de código de atendimento, a função do Lambda retorna um mensagem de resumo indicando que uma consulta está agendada (ou seja, a intenção está atendida).

### Note

Neste exemplo, mostramos como usar cartões de resposta. A função do Lambda cria e retorna um cartão de resposta ao Amazon Lex. As listas de cartão de resposta lista dias e horários disponíveis como botões para o usuário escolher. Ao testar o bot usando o cliente fornecido pelo console do Amazon Lex, não é possível ver o cartão de resposta. Para vê-lo, você deve integrar o bot com uma plataforma de mensagens, como Facebook Messenger. Para obter instruções, consulte [Integração de um bot do Amazon Lex com o Facebook Messenger](#) (p. 116). Para obter mais informações sobre cartões de resposta, consulte [Gerenciamento de mensagens](#) (p. 10).

Quando o Amazon Lex chama a função do Lambda, ela passa os dados do evento como entrada. Um dos campos do evento é `invocationSource`, que a função do Lambda usa para escolher entre uma validação de entrada e uma atividade de atendimento. Para obter mais informações, consulte [Formato de eventos de entrada](#) (p. 106).

### Próxima etapa

[Etapa 1: Criar um bot do Amazon Lex](#) (p. 138)

## Etapa 1: Criar um bot do Amazon Lex

Nesta seção, você cria um bot do Amazon Lex usando o esquema ScheduleAppointment, fornecido no console do Amazon Lex.

1. Faça login no Console de gerenciamento da AWS e abra o console do Amazon Lex em <https://console.aws.amazon.com/lex/>.

2. Na página Bots, selecione Create.
3. Na página Create your Lex bot:
  - Escolha o esquema ScheduleAppointment.
  - Deixe o nome do bot padrão (ScheduleAppointment).
4. Escolha Criar.

Esta etapa salva e cria o bot. O console envia as seguintes solicitações ao Amazon Lex durante o processo de criação:

- Crie uma nova versão dos tipos de slot (da versão \$LATEST). Para obter mais informações sobre tipos de slot neste esquema de bot, consulte [Visão geral do esquema de bot \(ScheduleAppointment\) \(p. 137\)](#).
- Crie uma versão da intenção MakeAppointment (da versão \$LATEST). Em alguns casos, o console envia uma solicitação para a operação da API `update` antes de criar uma nova versão.
- Atualize a versão \$LATEST do bot.

Neste ponto, o Amazon Lex cria um modelo de machine learning para o bot. Quando você testa o bot no console, o console usa a API de tempo de execução para enviar a entrada do usuário de volta para o Amazon Lex. O Amazon Lex, por sua vez, usa o modelo de Machine Learning para interpretar a entrada do usuário.

5. O console mostra o bot ScheduleAppointment. Na guia Editor, analise os detalhes da intenção pré-configurada (MakeAppointment).
6. Teste o bot na janela de teste. Use a seguinte captura de tela para ter uma conversa de teste com o bot:

The screenshot shows the 'Test Bot' interface in the Amazon Lex console. At the top, it says 'Build: Latest | Status: READY'. The conversation starts with the user input 'Book an appointment'. The bot responds with 'What type of appointment would you like to schedule?'. The user inputs 'Root canal'. The bot asks 'When should I schedule your root canal?'. The user inputs 'December 18'. The bot asks 'At what time do you want to schedule the root canal?'. The user inputs '4 pm'. The bot responds with '16:00 is available, should I go ahead and book your appointment?'. The user inputs 'Yes'. Finally, the bot outputs 'AppointmentType:root canal Date:2017-12-18 Time:16:00'. At the bottom, there is a 'Clear' button and a text input field with the placeholder 'Type to your bot...'.

Observe o seguinte:



- Na entrada inicial do usuário ("Marcar uma consulta"), o bot infere a intenção (MakeAppointment).
- Em seguida, ele usa os prompts configurados para obter dados de slot do usuário.
- O esquema de bot tem a intenção MakeAppointment configurada com o seguinte prompt de confirmação:

```
{Time} is available, should I go ahead and book your appointment?
```

Depois que o usuário fornece todos os dados do slot, o Amazon Lex retorna uma resposta ao cliente com uma solicitação de confirmação como a mensagem. O cliente exibe a mensagem para o usuário:

```
16:00 is available, should I go ahead and book your appointment?
```

Observe que o bot aceita quaisquer data e horário de consulta, pois você não tem nenhum código para inicializar ou validar os dados do usuário. Na próxima seção, você adicionará uma função do Lambda para fazer isso.

Próxima etapa

[Etapa 2: Criar uma função do Lambda \(p. 140\)](#)

## Etapa 2: Criar uma função do Lambda

Nesta seção, você cria uma função do Lambda usando um esquema (lex-make-appointment-python) fornecido no console do Lambda. Você também testa a função Lambda chamando-a usando dados do evento de exemplo do Amazon Lex fornecido pelo console.

1. Faça login no Console de gerenciamento da AWS e abra o console AWS Lambda em <https://console.aws.amazon.com/lambda/>.
2. Escolha Create a Lambda function (Criar uma função do Lambda).
3. Em Select blueprint (Selecionar esquema), digite **lex** para localizar o esquema e, em seguida, escolha o esquema lex-make-appointment-python.
4. Configure a função do Lambda da forma a seguir e, em seguida, escolha Create Function (Criar função).
  - Digite o nome da função do Lambda (MakeAppointmentCodeHook).
  - Para a função, escolha Create a new role from template(s) e, em seguida, digite um nome de função.
  - Deixe outros valores padrão.
5. Testar a função do Lambda.
  - a. Escolha Actions e, em seguida, escolha Configure test event.
  - b. Na lista Sample event template, escolha Lex-Make Appointment (preview). Este evento de exemplo usa o modelo de solicitação/resposta do Amazon Lex com valores definidos para corresponder a uma solicitação em seu bot do Amazon Lex. Para obter informações sobre o modelo de solicitação/resposta do Amazon Lex, consulte [Uso de funções do Lambda \(p. 106\)](#).
  - c. Escolha Save and test.
  - d. Verifique se a função do Lambda foi executada com êxito. A resposta, neste caso, corresponde ao modelo de resposta do Amazon Lex.

Próxima etapa

[Etapa 3: atualizar a intenção - configuração de um hook de código \(p. 141\)](#)

## Etapa 3: atualizar a intenção - configuração de um hook de código

Nesta seção, você atualiza a configuração da intenção `MakeAppointment` para usar a função do Lambda como um gancho de código para as atividades de validação e de atendimento.

1. No console do Amazon Lex, selecione o bot `ScheduleAppointment`. O console mostra a intenção `MakeAppointment`. Modifique a configuração de intenção da seguinte forma.

### Note

Você pode atualizar apenas as versões `$LATEST` de qualquer um dos recursos do Amazon Lex incluindo as intenções. Verifique se a versão da intenção está definida como `$LATEST`. Você ainda não publicou uma versão de seu bot, então a versão ainda deve ser `$LATEST` no console.

- a. Na seção **Options (Opções)**, escolha **Initialization and validation code hook (Gancho de código de inicialização e validação)** e, em seguida, escolha a função do Lambda na lista.
  - b. Na seção **Fulfillment (Atendimento)**, escolha **AWS Lambda function (Função do AWS Lambda)** e, em seguida, escolha a função do Lambda na lista.
  - c. Escolha **Goodbye message** e digite uma mensagem.
2. Escolha **Save** e, em seguida, **Build**.
  3. Teste o bot.

The screenshot shows the 'Test Bot' interface in the Amazon Lex console. At the top, it says 'Build: Latest | Status: READY'. The conversation starts with the user input 'Book appointment'. The bot responds with 'What type of appointment would you like to schedule?'. The user inputs 'root canal'. The bot responds with 'When would you like to schedule your root canal?'. The user inputs 'Tuesday'. The bot responds with 'We do not have any availability on that date, is there another day which works for you?'. The user inputs 'Wednesday'. The bot responds with 'What time on 2017-01-18 works for you? 4:00 p.m. is our only availability, does that work for you?'. The user inputs 'yes'. The bot responds with 'Okay, I have booked your appointment. We will see you at 4:00 p.m. on 2017-01-18'. At the bottom, there is a 'Clear' button and a text input field with the placeholder 'Type to your bot...'.

Próxima etapa

[Etapa 4: implantar o bot na plataforma do Facebook Messenger \(p. 142\)](#)

## Etapa 4: implantar o bot na plataforma do Facebook Messenger

Na seção anterior, você testou o bot `ScheduleAppointment` usando o cliente no console do Amazon Lex. No momento, o console do Amazon Lex não oferece suporte a cartões de resposta. Para testar os cartões de resposta gerados dinamicamente aos quais o bot oferece suporte, implante o bot na plataforma do Facebook Messenger e teste-o.

Para obter instruções, consulte [Integração de um bot do Amazon Lex com o Facebook Messenger \(p. 116\)](#).

Próxima etapa

[Detalhes do fluxo de informações \(p. 142\)](#)

## Detalhes do fluxo de informações

O esquema de bot `ScheduleAppointment` exibe, principalmente, o uso de cartões de resposta gerados dinamicamente. A função do Lambda neste exercício inclui cartões de resposta em sua resposta ao Amazon Lex. O Amazon Lex inclui os cartões de resposta em sua resposta ao cliente. Esta seção explica o seguinte:

- Fluxo de dados entre o cliente e o Amazon Lex.

A seção supõe que o cliente envia solicitações ao Amazon Lex usando a API de tempo de execução `PostText` e exibe os detalhes de solicitação/resposta adequadamente. Para obter mais informações sobre a API de runtime `PostText`, consulte [PostText \(p. 355\)](#).

### Note

Para obter um exemplo do fluxo de informações entre o cliente e o Amazon Lex em que o cliente usa a API `PostContent`, consulte [Etapa 2a \(opcional\): Revise os detalhes do fluxo de informações falado \(console\) \(p. 41\)](#).

- Fluxo de dados entre o Amazon Lex e a função do Lambda. Para obter mais informações, consulte [Evento de entrada de função do Lambda e formato de resposta \(p. 106\)](#).

### Note

O exemplo supõe que você está usando o cliente do Facebook Messenger, que não passa atributos de sessão na solicitação ao Amazon Lex. Portanto, os exemplos de solicitações desta seção mostram `sessionAttributes` vazio. Se você testar o bot usando o cliente fornecido no console do Amazon Lex, o cliente incluirá os atributos de sessão.

Esta seção descreve o que acontece após cada entrada do usuário.

#### 1. Usuário: tipos **Book an appointment**

- a. O cliente (console) envia a seguinte solicitação [PostContent \(p. 347\)](#) ao Amazon Lex:

```
POST /bot/ScheduleAppointment/alias/$LATEST/user/bijt6rovckwecnzsbthrr1d7lv3ja3n/text
```

```
"Content-Type": "application/json"
"Content-Encoding": "amz-1.0"

{
  "inputText": "book appointment",
  "sessionAttributes": {}
}
```

O URI e o corpo da solicitação fornecem informações ao Amazon Lex:

- URI da solicitação – fornece o nome do bot (ScheduleAppointment), o alias do bot (\$LATEST) e o ID do nome do usuário. O text final indica que esta é uma solicitação de API PostText (não PostContent).
  - Corpo da solicitação – inclui a entrada do usuário (inputText) e sessionAttributes vazio.
- b. No inputText, o Amazon Lex detecta a intenção (MakeAppointment). O serviço chama a função do Lambda, que está configurada como um gancho de código, para executar a inicialização e a validação passando o seguinte evento. Para obter mais detalhes, consulte [Formato de eventos de entrada \(p. 106\)](#).

```
{
  "currentIntent": {
    "slots": {
      "AppointmentType": null,
      "Date": null,
      "Time": null
    },
    "name": "MakeAppointment",
    "confirmationStatus": "None"
  },
  "bot": {
    "alias": null,
    "version": "$LATEST",
    "name": "ScheduleAppointment"
  },
  "userId": "bijt6rovckwecnzesbthrrld7lv3ja3n",
  "invocationSource": "DialogCodeHook",
  "outputDialogMode": "Text",
  "messageVersion": "1.0",
  "sessionAttributes": {}
}
```

Além das informações enviadas pelo cliente, o Amazon Lex; também inclui os seguintes dados:

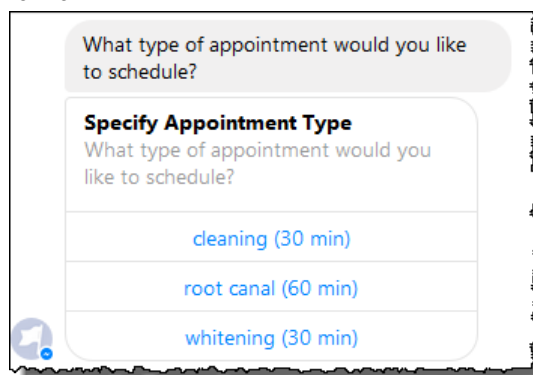
- currentIntent – fornece informações da intenção atual.
  - invocationSource – indica o objetivo da chamada da função do Lambda. Nesse caso, o objetivo é executar a inicialização e a validação dos dados do usuário (o Amazon Lex sabe que o usuário ainda não forneceu todos os dados do slot para atender à intenção).
  - messageVersion – atualmente, o Amazon Lex oferece suporte apenas à versão 1.0.
- c. No momento, todos os valores de slot são nulos (não há nada para validar). A função do Lambda retorna a seguinte resposta para o Amazon Lex direcionando o serviço a obter informações para o slot AppointmentType. Para obter mais informações sobre o formato de resposta, consulte [Formato de resposta \(p. 109\)](#).

```
{
  "dialogAction": {
    "slotToElicit": "AppointmentType",
    "intentName": "MakeAppointment",
    "responseCard": {
      "genericAttachments": [
```

```
{
  "buttons": [
    {
      "text": "cleaning (30 min)",
      "value": "cleaning"
    },
    {
      "text": "root canal (60 min)",
      "value": "root canal"
    },
    {
      "text": "whitening (30 min)",
      "value": "whitening"
    }
  ],
  "subTitle": "What type of appointment would you like to
schedule?",
  "title": "Specify Appointment Type"
},
{
  "version": 1,
  "contentType": "application/vnd.amazonaws.card.generic"
},
{
  "slots": {
    "AppointmentType": null,
    "Date": null,
    "Time": null
  },
  "type": "ElicitSlot",
  "message": {
    "content": "What type of appointment would you like to schedule?",
    "contentType": "PlainText"
  }
},
{
  "sessionAttributes": {}
}
```

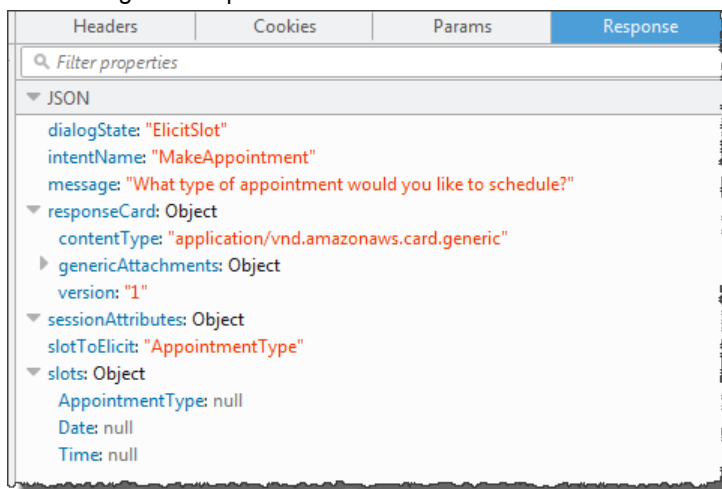
A resposta inclui os campos `dialogAction` e `sessionAttributes`. Dentre outras coisas, o campo `dialogAction` retorna os seguintes campos:

- `type` – definindo esse campo como `ElicitSlot`, a função do Lambda direciona o Amazon Lex a escolher o valor para o slot especificado no campo `slotToElicit`. A função do Lambda também fornece uma `message` para transmitir ao usuário.
- `responseCard` – identifica uma lista de valores possíveis para o slot `AppointmentType`. Um cliente que oferece suporte a cartões de resposta (por exemplo, Facebook Messenger) exibe um cartão de resposta para permitir que o usuário escolha um tipo de consulta da seguinte forma:



The screenshot shows a chat interface with a light gray background. At the top, a speech bubble contains the text "What type of appointment would you like to schedule?". Below this, a card titled "Specify Appointment Type" is displayed. The card has a subtitle "What type of appointment would you like to schedule?" and three buttons: "cleaning (30 min)", "root canal (60 min)", and "whitening (30 min)". The buttons are arranged vertically and have a light blue background with a darker blue border. A small blue speech bubble icon is visible in the bottom left corner of the card.

- d. Conforme indicado pelo `dialogAction.type` na resposta da função do Lambda, o Amazon Lex envia a seguinte resposta de volta ao cliente:



O cliente lê a resposta e, em seguida, exibe a mensagem: "What type of appointment would you like to schedule? (Que tipo de consulta você deseja agendar?)" e o cartão de resposta (se o cliente oferecer suporte a cartões de resposta).

2. Usuário: dependendo do cliente, o usuário tem duas opções:

- Se o cartão de resposta for exibido, escolha root canal (60 min) (canal de raiz (60 min)) ou digite **root canal**.
- Se o cliente não oferecer suporte a cartões de resposta, digite **root canal**.

- a. O cliente envia a seguinte solicitação `PostText` ao Amazon Lex (quebras de linha foram adicionadas para legibilidade):

```
POST /bot/BookTrip/alias/$LATEST/user/bijt6rovckwecnzeshrr1d7lv3ja3n/text
"Content-Type": "application/json"
"Content-Encoding": "amz-1.0"

{
  "inputText": "root canal",
  "sessionAttributes": {}
}
```

- b. O Amazon Lex chama a função do Lambda para validação dos dados do usuário enviando o seguinte evento como parâmetro:

```
{
  "currentIntent": {
    "slots": {
      "AppointmentType": "root canal",
      "Date": null,
      "Time": null
    },
    "name": "MakeAppointment",
    "confirmationStatus": "None"
  },
  "bot": {
    "alias": null,
    "version": "$LATEST",
    "name": "ScheduleAppointment"
  }
}
```

```
    },
    "userId": "bijt6rovckwecnzeshthrrid7lv3ja3n",
    "invocationSource": "DialogCodeHook",
    "outputDialogMode": "Text",
    "messageVersion": "1.0",
    "sessionAttributes": {}
}
```

Nos dados de evento, observe o seguinte:

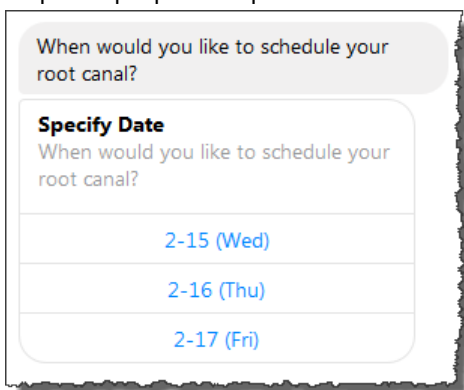
- `invocationSource` continua sendo `DialogCodeHook`. Nesta etapa, estamos apenas validando os dados do usuário.
  - O Amazon Lex define o campo `AppointmentType` no slot `currentIntent.slots` como `root canal`.
  - O Amazon Lex simplesmente passa o campo `sessionAttributes` entre o cliente e a função do Lambda.
- c. A função do Lambda valida a entrada do usuário e retorna a seguinte resposta ao Amazon Lex, direcionando o serviço a escolher um valor para a data da consulta.

```
{
  "dialogAction": {
    "slotToElicit": "Date",
    "intentName": "MakeAppointment",
    "responseCard": {
      "genericAttachments": [
        {
          "buttons": [
            {
              "text": "2-15 (Wed)",
              "value": "Wednesday, February 15, 2017"
            },
            {
              "text": "2-16 (Thu)",
              "value": "Thursday, February 16, 2017"
            },
            {
              "text": "2-17 (Fri)",
              "value": "Friday, February 17, 2017"
            },
            {
              "text": "2-20 (Mon)",
              "value": "Monday, February 20, 2017"
            },
            {
              "text": "2-21 (Tue)",
              "value": "Tuesday, February 21, 2017"
            }
          ],
          "subTitle": "When would you like to schedule your root canal?",
          "title": "Specify Date"
        }
      ],
      "version": 1,
      "contentType": "application/vnd.amazonaws.card.generic"
    },
    "slots": {
      "AppointmentType": "root canal",
      "Date": null,
      "Time": null
    },
    "type": "ElicitSlot",
    "message": {
```

```
        "content": "When would you like to schedule your root canal?",  
        "contentType": "PlainText"  
    },  
    "sessionAttributes": {}  
}
```

Novamente, a resposta inclui os campos `dialogAction` e `sessionAttributes`. Dentre outras coisas, o campo `dialogAction` retorna os seguintes campos:

- `type` – definindo esse campo como `ElicitSlot`, a função do Lambda direciona o Amazon Lex a escolher o valor para o slot especificado no campo `slotToElicit`. A função do Lambda também fornece uma `message` para transmitir ao usuário.
- `responseCard` – identifica uma lista de valores possíveis para o slot `Date`. Um cliente que oferece suporte a cartões de resposta (por exemplo, Facebook Messenger) exibe um cartão de resposta que permite que o usuário escolha uma data de consulta:



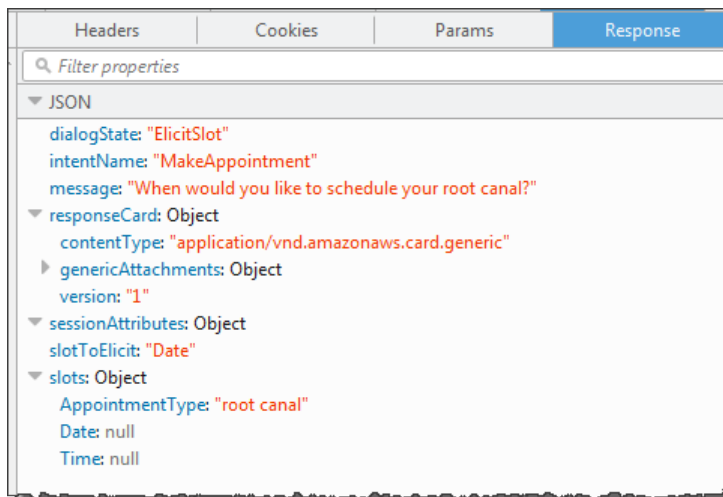
The image shows a screenshot of a response card from Facebook Messenger. The card has a light gray header with the text "When would you like to schedule your root canal?". Below the header is a section titled "Specify Date" with the same question. Underneath, there are three selectable date options: "2-15 (Wed)", "2-16 (Thu)", and "2-17 (Fri)". The card is displayed on a dark background, and the entire screenshot is framed with a rough, torn-edge border.

Apesar de a função do Lambda ter retornado cinco datas, o cliente (Facebook Messenger) tem um limite de três botões por cartão de resposta. Portanto, você verá apenas os primeiros três valores na captura de tela.

Essas datas são codificadas na função do Lambda. Em uma aplicação de produção, você pode usar um calendário para obter datas disponíveis em tempo real. Como as datas são dinâmicas, você deve gerar o cartão de resposta dinamicamente na função do Lambda.

- d. O Amazon Lex observa o `dialogAction.type` e retorna uma resposta ao cliente que inclui informações da resposta da função do Lambda.





O cliente exibe a mensagem: When would you like to schedule your root canal? (Quando você deseja agendar seu canal de raiz?) e o cartão de resposta (se o cliente oferecer suporte a cartões de resposta).

### 3. Usuário: tipos **Thursday**

- a. O cliente envia a seguinte solicitação PostText ao Amazon Lex (quebras de linha foram adicionadas para legibilidade):

```
POST /bot/BookTrip/alias/$LATEST/user/bijt6rovckwecnzeshrr1d7lv3ja3n/text
"Content-Type": "application/json"
"Content-Encoding": "amz-1.0"

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

- b. O Amazon Lex chama a função do Lambda para validação dos dados do usuário enviando o seguinte evento como parâmetro:

```
{
  "currentIntent": {
    "slots": {
      "AppointmentType": "root canal",
      "Date": "2017-02-16",
      "Time": null
    },
    "name": "MakeAppointment",
    "confirmationStatus": "None"
  },
  "bot": {
    "alias": null,
    "version": "$LATEST",
    "name": "ScheduleAppointment"
  },
  "userId": "u3fpr9gghj02zts7y5tpq5mm4din2xqy",
  "invocationSource": "DialogCodeHook",
  "outputDialogMode": "Text",
  "messageVersion": "1.0",
  "sessionAttributes": {}
}
```

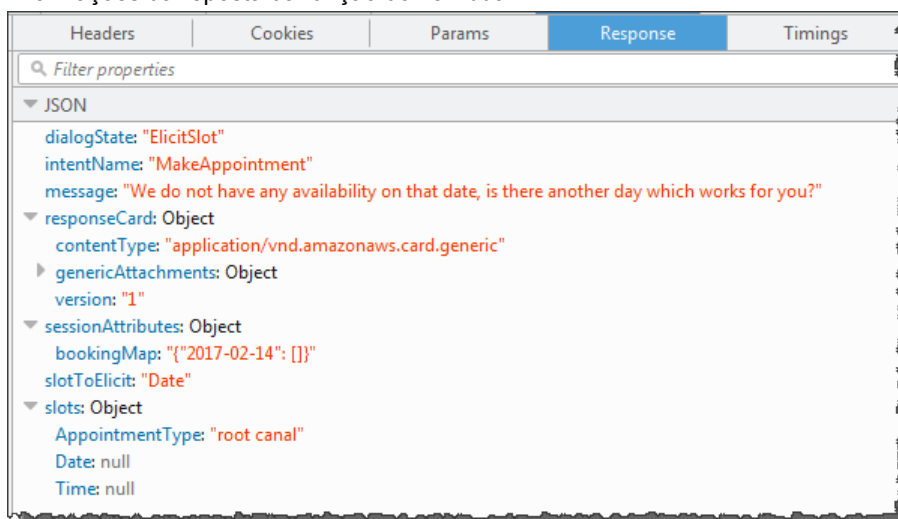
Nos dados de evento, observe o seguinte:

- `invocationSource` continua sendo `DialogCodeHook`. Nesta etapa, estamos apenas validando os dados do usuário.
  - O Amazon Lex define o campo `Date` no slot `currentIntent.slots` como `2017-02-16`.
  - O Amazon Lex simplesmente passa o `sessionAttributes` entre o cliente e a função do Lambda.
- c. A função do Lambda valida a entrada do usuário. Desta vez, a função do Lambda determina que não há consultas disponíveis na data especificada. Ela retorna a seguinte resposta ao Amazon Lex direcionando o serviço a escolher novamente um valor para a data da consulta.

```
{
  "dialogAction": {
    "slotToElicit": "Date",
    "intentName": "MakeAppointment",
    "responseCard": {
      "genericAttachments": [
        {
          "buttons": [
            {
              "text": "2-15 (Wed)",
              "value": "Wednesday, February 15, 2017"
            },
            {
              "text": "2-17 (Fri)",
              "value": "Friday, February 17, 2017"
            },
            {
              "text": "2-20 (Mon)",
              "value": "Monday, February 20, 2017"
            },
            {
              "text": "2-21 (Tue)",
              "value": "Tuesday, February 21, 2017"
            }
          ],
          "subTitle": "When would you like to schedule your root canal?",
          "title": "Specify Date"
        }
      ],
      "version": 1,
      "contentType": "application/vnd.amazonaws.card.generic"
    },
    "slots": {
      "AppointmentType": "root canal",
      "Date": null,
      "Time": null
    },
    "type": "ElicitSlot",
    "message": {
      "content": "We do not have any availability on that date, is there another day which works for you?",
      "contentType": "PlainText"
    }
  },
  "sessionAttributes": {
    "bookingMap": "{\"2017-02-16\": []}"
  }
}
```

Novamente, a resposta inclui os campos `dialogAction` e `sessionAttributes`. Dentre outras coisas, o `dialogAction` retorna os seguintes campos:

- `dialogAction` field:
    - `type` – a função do Lambda define esse valor como `ElicitSlot` e redefine o campo `slotToElicit` como `Date`. A função do Lambda também fornece uma `message` adequada para transmitir ao usuário.
    - `responseCard` – retorna uma lista de valores para o slot `Date`.
  - `sessionAttributes` - desta vez, a função do Lambda inclui o atributo de sessão `bookingMap`. Seu valor é a data solicitada da consulta e das consultas disponíveis (um objeto vazio indica que não há consultas disponíveis).
- d. O Amazon Lex observa o `dialogAction.type` e retorna uma resposta ao cliente que inclui informações da resposta da função do Lambda.



O cliente exibe a mensagem: We do not have any availability on that date, is there another day which works for you? (Não temos nenhuma disponibilidade nessa data, há outro dia que funcione para você?) e o cartão de resposta (se o cliente oferecer suporte a cartões de resposta).

4. Usuário: dependendo do cliente, o usuário tem duas opções:
- Se o cartão de resposta for exibido, escolha 2-15 (Wed) (15/2 (quarta)) ou digite **Wednesday**.
  - Se o cliente não oferecer suporte a cartões de resposta, digite **Wednesday**.
- a. O cliente envia a seguinte solicitação `PostText` ao Amazon Lex:

```
POST /bot/BookTrip/alias/$LATEST/user/bi jt6rovckwecnzesbthrr1d7lv3ja3n/text
"Content-Type": "application/json"
"Content-Encoding": "amz-1.0"

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

## Note

O cliente Facebook Messenger não define atributos de sessão. Se desejar manter estados de sessão entre as solicitações, você deverá fazer isso na função do Lambda. Em uma aplicação real, talvez você precise manter esses atributos de sessão em banco de dados de back-end.

- b. O Amazon Lex chama a função do Lambda para validação dos dados do usuário enviando o seguinte evento como parâmetro:

```
{
  "currentIntent": {
    "slots": {
      "AppointmentType": "root canal",
      "Date": "2017-02-15",
      "Time": null
    },
    "name": "MakeAppointment",
    "confirmationStatus": "None"
  },
  "bot": {
    "alias": null,
    "version": "$LATEST",
    "name": "ScheduleAppointment"
  },
  "userId": "u3fpr9gghj02zts7y5tpq5mm4din2xqy",
  "invocationSource": "DialogCodeHook",
  "outputDialogMode": "Text",
  "messageVersion": "1.0",
  "sessionAttributes": {
  }
}
```

O Amazon Lex atualizou `currentIntent.slots` definindo o slot `Date` como 2017-02-15.

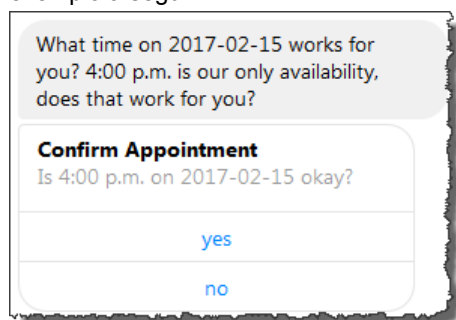
- c. A função do Lambda valida a entrada do usuário e retorna a seguinte resposta ao Amazon Lex, direcionando-o a escolher o valor para o horário da consulta.

```
{
  "dialogAction": {
    "slots": {
      "AppointmentType": "root canal",
      "Date": "2017-02-15",
      "Time": "16:00"
    },
    "message": {
      "content": "What time on 2017-02-15 works for you? 4:00 p.m. is our only availability, does that work for you?",
      "contentType": "PlainText"
    },
    "type": "ConfirmIntent",
    "intentName": "MakeAppointment",
    "responseCard": {
      "genericAttachments": [
        {
          "buttons": [
            {
              "text": "yes",
              "value": "yes"
            },
            {
              "text": "no",
              "value": "no"
            }
          ]
        }
      ]
    }
  }
}
```

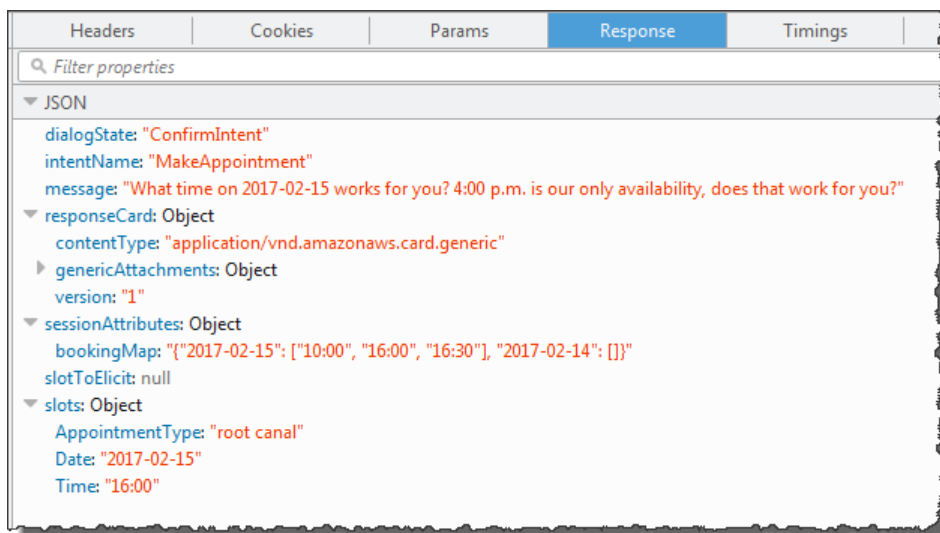
```
        },
        ],
        "subTitle": "Is 4:00 p.m. on 2017-02-15 okay?",
        "title": "Confirm Appointment"
    }
  ],
  "version": 1,
  "contentType": "application/vnd.amazonaws.card.generic"
},
"sessionAttributes": {
  "bookingMap": "{\"2017-02-15\": [\"10:00\", \"16:00\", \"16:30\"]}"
}
}
```

Novamente, a resposta inclui os campos `dialogAction` e `sessionAttributes`. Dentre outras coisas, o `dialogAction` retorna os seguintes campos:

- `dialogAction` field:
  - `type` – a função do Lambda define esse valor como `ConfirmIntent` direcionando o Amazon Lex a obter a confirmação do usuário para o horário da consulta sugerido na `message`.
  - `responseCard` – retorna uma lista de valores sim/não para o usuário escolher. Se o cliente for compatível com cartões de resposta, ele exibirá o cartão de resposta, como mostrado no exemplo a seguir:



- `sessionAttributes` - a função do Lambda define o atributo de sessão `bookingMap` com o valor definido como a data da consulta e as consultas disponíveis naquela data. Neste exemplo, são consultas de 30 minutos. Para um canal de raiz que leva uma hora, apenas 16h pode ser marcada.
- d. Como indicado no `dialogAction.type` na resposta da função do Lambda, o Amazon Lex retorna a seguinte resposta ao cliente:



O cliente exibe a mensagem: What time on 2017-02-15 works for you? 4:00 p.m. is our only availability, does that work for you?

5. Usuário: selecione **yes**.

O Amazon Lex chama a função do Lambda com os seguintes dados de evento. Como o usuário respondeu **yes**, o Amazon Lex define `confirmationStatus` como `Confirmed` e define o campo `Time` no `currentIntent.slots` como 4 p.m.

```
{
  "currentIntent": {
    "slots": {
      "AppointmentType": "root canal",
      "Date": "2017-02-15",
      "Time": "16:00"
    },
    "name": "MakeAppointment",
    "confirmationStatus": "Confirmed"
  },
  "bot": {
    "alias": null,
    "version": "$LATEST",
    "name": "ScheduleAppointment"
  },
  "userId": "u3fpr9gghj02zts7y5tpq5mm4din2xqy",
  "invocationSource": "FulfillmentCodeHook",
  "outputDialogMode": "Text",
  "messageVersion": "1.0",
  "sessionAttributes": {
  }
}
```

Como o `confirmationStatus` é confirmado, a função do Lambda processa a intenção (marca uma consulta odontológica) e retorna a seguinte resposta ao Amazon Lex:

```
{
  "dialogAction": {
    "message": {
      "content": "Okay, I have booked your appointment. We will see you at 4:00 p.m. on 2017-02-15",
      "contentType": "PlainText"
    }
  }
}
```

```
    },
    "type": "Close",
    "fulfillmentState": "Fulfilled"
  },
  "sessionAttributes": {
    "formattedTime": "4:00 p.m.",
    "bookingMap": "{\"2017-02-15\": [\"10:00\"]}"
  }
}
```

Observe o seguinte:

- A função Lambda atualizou o `sessionAttributes`.
- `dialogAction.type` é definido como `Close`, o que direciona o Amazon Lex a não esperar uma resposta do usuário.
- `dialogAction.fulfillmentState` é definido como `Fulfilled`, indicando que a intenção foi cumprida com êxito.

O cliente exibe a mensagem: Okay, I have booked your appointment. We will see you at 4:00 p.m. on 2017-02-15.

## Exemplo de bot: BookTrip

Este exemplo mostra como criar um bot configurado para oferecer suporte a várias intenções. O exemplo também ilustra como é possível usar atributos de sessão para o compartilhamento de informações entre intenções. Depois de criar o bot, você usa um cliente de teste no console do Amazon Lex para testar o bot (BookTrip). O cliente usa a operação da API de tempo de execução do [PostText \(p. 355\)](#) para enviar solicitações ao Amazon Lex para cada entrada do usuário.

O bot BookTrip, neste exemplo, está configurado com duas intenções (BookHotel e BookCar). Por exemplo, suponha que um usuário reserve um hotel primeiro. Durante a interação, o usuário fornece informações como datas de check-in, local e número de diárias. Depois de a intenção ser cumprida, o cliente pode manter essas informações usando atributos de sessão. Para obter mais informações sobre atributos de sessão, consulte [PostText \(p. 355\)](#).

Agora, suponha que o usuário prossiga para reservar um carro. Usando as informações que o usuário forneceu na intenção anterior BookHotel (ou seja, cidade de destino e datas de check-in e check-out), o hook de código (função Lambda) que você configurou para inicializar e validar a intenção BookCar slot inicializa os dados de slot para a intenção BookCar (ou seja, destino, cidade de retirada, data de retirada e data de devolução). Isso mostra como o compartilhamento de informações entre intenções permite que você crie bots que podem participar de conversas dinâmicas com o usuário.

Neste exemplo, usamos os seguintes atributos de sessão. Apenas o cliente e a função do Lambda podem definir e atualizar os atributos de sessão. O Amazon Lex só os passa entre o cliente e a função do Lambda. O Amazon Lex não mantém nem modifica os atributos de sessão.

- `currentReservation` – contém dados de slot para uma reserva em andamento e outras informações relevantes. A seguir, veja um exemplo de solicitação do cliente para o Amazon Lex. Ele mostra o atributo de sessão `currentReservation` no corpo da solicitação.

```
POST /bot/BookTrip/alias/$LATEST/user/wch89kjqcpkds8seny7dly5x3otq68j3/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
  "inputText":"Chicago",
  "sessionAttributes":{
```

```
"currentReservation": "{\n  \"ReservationType\": \"Hotel\",\n  \"Location\": \"Moscow\",\n  \"RoomType\": null,\n  \"CheckInDate\": null,\n  \"Nights\": null\n}"
}
```

- `lastConfirmedReservation` – contém informações semelhantes para uma intenção anterior, se houver. Por exemplo, se o usuário reservou um hotel e está no processo de reserva de um carro, esse atributo de sessão armazena dados de slot para a intenção anterior `BookHotel`.
- `confirmationContext` – a função do Lambda define isso como `AutoPopulate` ao preencher automaticamente alguns dos dados de slot com base nos dados de slot da reserva anterior (se houver). Isso permite o compartilhamento de informações entre intenções. Por exemplo, se o usuário tiver reservado um hotel anteriormente e agora desejar reservar um carro, o Amazon Lex poderá solicitar que o usuário confirme (ou negue) se o carro está sendo reservado para a mesma cidade e as mesmas datas que a reserva de hotel

Neste exercício, você usa esquemas para criar um bot do Amazon Lex e uma função do Lambda. Para obter mais informações sobre esquemas, consulte [Esquemas do Amazon Lex e do AWS Lambda \(p. 113\)](#).

Próxima etapa

[Etapa 1: revisão dos esquemas usados neste exercício \(p. 155\)](#)

## Etapa 1: revisão dos esquemas usados neste exercício

Tópicos

- [Visão geral do esquema de bot \(BookTrip\) \(p. 155\)](#)
- [Visão geral do esquema da função do Lambda \(lex-book-trip-python\) \(p. 157\)](#)

### Visão geral do esquema de bot (BookTrip)

O esquema (`BookTrip`) que você usa para criar um bot fornece as seguintes pré-configurações:

- Tipos de slot – dois tipos de slot personalizados:
  - `RoomTypes` com valores de enumeração: `king`, `queen` e `deluxe`, para uso na intenção `BookHotel`.
  - `CarTypes` com valores de enumeração: `economy`, `standard`, `midsize`, `full size`, `luxury` e `minivan`, para uso na intenção `BookCar`.
- Intenção 1 (`BookHotel`) – é pré-configurada da seguinte forma:
  - Slots pré-configurados
    - `RoomType`, do tipo de slot personalizado `RoomTypes`
    - `Location`, do tipo de slot integrado `AMAZON.US_CITY`
    - `CheckInDate`, do tipo de slot integrado `AMAZON.DATE`
    - `Nights`, do tipo de slot integrado `AMAZON.NUMBER`
  - Enunciados pré-configurados



- "Reservar um hotel"
- "Eu gostaria de fazer reservas de hotéis"
- "Reservar uma estadia de {Nights} em {Location}"

Se o usuário enunciar qualquer um desses, o Amazon Lex determinará que `BookHotel` é a intenção e, em seguida, solicitará os dados de slot ao usuário.

- Solicitações pré-configuradas
  - Solicitação do slot `Location` – "What city will you be staying in? (Em que cidade você se hospedará?)"
  - Solicitação do slot `CheckInDate` – "What day do you want to check in? (Em que dia você deseja fazer check-in?)"
  - Prompt do slot `Nights` – "How many nights will you be staying? (Por quantas noites você se hospedará?)"
  - Prompt do slot `RoomType` – "What type of room would you like, queen, king, or deluxe? (Que tipo de quarto você deseja: queen, king ou deluxe?)"
  - Declaração de confirmação – "Okay, I have you down for a {Nights} night stay in {Location} starting {CheckInDate}. (OK, reservei para você uma estadia de {Nights} em {Location} a partir de {CheckInDate}). Posso fazer a reserva?"
  - Negação – "Okay, I have cancelled your reservation in progress. (OK, cancelei sua reserva em andamento.)"
- Intenção 2 (`BookCar`) – é pré-configurada da seguinte forma:
  - Slots pré-configurados
    - `PickUpCity`, do tipo integrado `AMAZON.US_CITY`
    - `PickUpDate`, do tipo integrado `AMAZON.DATE`
    - `ReturnDate`, do tipo integrado `AMAZON.DATE`
    - `DriverAge`, do tipo integrado `AMAZON.NUMBER`
    - `CarType`, do tipo personalizado `CarTypes`
  - Enunciados pré-configurados
    - "Reservar um carro"
    - "Reservar um carro"
    - "Fazer uma reserva de carro"

Se o usuário enunciar qualquer um desses, o Amazon Lex determinará que `BookCar` é a intenção e solicitará os dados de slot do usuário.

- Solicitações pré-configuradas
  - Solicitação do slot `PickUpCity` – "In what city do you need to rent a car? (Em que cidade você precisa alugar um carro?)"
  - Solicitação do slot `PickUpDate` – "What day do you want to start your rental? (Em que dia você deseja iniciar o aluguel?)"
  - Solicitação do slot `ReturnDate` – "What day do you want to return this car? (Em que dia você deseja devolver o carro?)"
  - Solicitação do slot `DriverAge` – "How old is the driver for this rental? (Qual é a idade do motorista deste carro?)"
  - Solicitação do slot `CarType` – "What type of car would you like to rent? (Que tipo de carro você deseja alugar?)" Nossas opções mais populares são econômico, médio e luxo"
  - Declaração de confirmação – "Okay, I have you down for a {CarType} rental in {PickUpCity} from {PickUpDate} to {ReturnDate} (OK, reservei para você um aluguel de {CarType} em {PickUpCity} de {PickUpDate} a {ReturnDate}). Devo fazer a reserva?"

- Negação – "Okay, I have cancelled your reservation in progress. (OK, cancelei sua reserva em andamento.)"

## Visão geral do esquema da função do Lambda (lex-book-trip-python)

Além do esquema de bot, o AWS Lambda fornece um esquema (lex-book-trip-python) que você pode usar como um gancho de código com o esquema de bot. Para obter uma lista de esquemas de bot e dos esquemas da função do Lambda correspondentes, consulte [Esquemas do Amazon Lex e do AWS Lambda \(p. 113\)](#).

Quando cria um bot usando o esquema BookTrip, você atualiza a configuração das duas intenções (BookCar e BookHotel) adicionando essa função do Lambda como um gancho de código para a inicialização/validação dos dados do usuário e o atendimento das intenções.

O código de função Lambda fornecido demonstra uma conversa dinâmica usando informações conhecidas anteriormente (mantidas nos atributos de sessão) sobre um usuário para inicializar os valores de slot para uma intenção. Para obter mais informações, consulte [Gerenciar contexto da conversa \(p. 20\)](#).

Próxima etapa

[Etapa 2: Criar um bot do Amazon Lex \(p. 157\)](#)

## Etapa 2: Criar um bot do Amazon Lex

Nesta seção, você cria um bot do Amazon Lex (BookTrip).

1. Faça login no Console de gerenciamento da AWS e abra o console do Amazon Lex em <https://console.aws.amazon.com/lex/>.
2. Na página Bots, selecione Create.
3. Na página Create your Lex bot,
  - Escolha o esquema BookTrip.
  - Deixe o nome do bot padrão (BookTrip).
4. Escolha Criar. O console envia uma série de solicitações ao Amazon Lex para criar o bot. Observe o seguinte:
5. O console mostra o bot BookTrip. Na guia Editor, analise os detalhes das intenções pré-configuradas (BookCar e BookHotel).
6. Teste o bot na janela de teste. Use o seguinte para ter uma conversa de teste com o bot:

The screenshot shows a chat interface titled "Test Bot". The conversation is as follows:

- User: Book a hotel
- Bot: What city will you be staying in?
- User: Chicago
- Bot: What day do you want to check in?
- User: December 18th
- Bot: How many nights will you be staying?
- User: 4
- Bot: What type of room would you like, queen, king or deluxe?
- User: Queen
- Bot: Okay, I have you down for a 4 night stay in Chicago starting 2016-12-18. Shall I book the reservation?
- User: Yes
- Bot: CheckInDate:2016-12-18 Location:Chicago Nights:4 RoomType:queen

Below the bot's final message is a "Clear" button. At the bottom of the interface is a text input field with the placeholder "Type to your bot..." and a send button.

Da entrada inicial do usuário ("Book a hotel (Reservar um hotel)"), o Amazon Lex deduz a intenção (BookHotel). Em seguida, o bot solicita as informações pré-configuradas nessa intenção para escolher os dados de slot do usuário. Depois de o usuário fornecer todos os dados de slot, o Amazon Lex retorna uma resposta ao cliente com uma mensagem que inclui todas as entradas do usuário como uma mensagem. O cliente exibe a mensagem na resposta, conforme exibido.

```
CheckInDate:2016-12-18 Location:Chicago Nights:5 RoomType:queen
```

Agora, continue a conversa e tente reservar um carro.

The screenshot shows the 'Test Bot' interface for an Amazon Lex bot. The conversation is as follows:

- User: Also book a car
- Bot: In what city do you need to rent a car?
- User: Chicago
- Bot: What day do you want to start your rental?
- User: December 18th
- Bot: What day do you want to return the car?
- User: December 22nd
- Bot: How old is the driver for this rental?
- User: 35
- Bot: What type of car would you like to rent? Our most popular options are economy, midsize, and luxury
- User: economy
- Bot: The price of this economy rental in Chicago from 2016-12-18 to 2016-12-22 is 556 dollars. Shall I book the reservation?
- User: Yes
- Bot: CarType:economy DriverAge:35 PickUpCity:Chicago  
PickUpDate:2016-12-18 ReturnDate:2016-12-22

At the bottom, there is a 'Clear' button and a text input field with the placeholder 'Type to your bot...'.

Observe que,

- Não há validação de dados do usuário dessa vez. Por exemplo, você pode fornecer qualquer cidade para reservar um hotel.
- Você está fornecendo algumas das mesmas informações novamente (destino, cidade de retirada, data de retirada e data de devolução) para reservar um carro. Em uma conversa dinâmica, seu bot deve inicializar algumas dessas informações com base na entrada anterior que o usuário forneceu para reservar o hotel.

Na próxima seção, você criará uma função Lambda para executar a validação dos dados do usuário e a inicialização usando o compartilhamento de informações entre intenções por meio de atributos de sessão. Em seguida, use a configuração de intenção, adicionando a função Lambda como hook de código para executar a inicialização/validação de entradas do usuário e cumprir intenções.

Próxima etapa

[Etapa 3: Criar uma função do Lambda \(p. 160\)](#)

## Etapa 3: Criar uma função do Lambda

Nesta seção, você cria uma função do Lambda usando um esquema (lex-book-trip-python) fornecido no console do AWS Lambda. Você também testa a função do Lambda ao chamá-la usando dados de eventos de exemplo fornecidos pelo console.

Essa função do Lambda é escrita em Python.

1. Faça login no Console de gerenciamento da AWS e abra o console AWS Lambda em <https://console.aws.amazon.com/lambda/>.
2. Escolha Create function.
3. Selecione Use a blueprint (Usar um esquema). Digite **lex** para localizar o esquema e escolha o esquema `lex-book-trip-python`.
4. Configure a função do Lambda da forma a seguir e, em seguida, escolha Create Function (Criar função).
  - Digite um nome de função do Lambda (`BookTripCodeHook`).
  - Para a função, escolha Create a new role from template(s) e, em seguida, digite um nome de função.
  - Deixe os outros valores padrão.
5. Testar a função do Lambda. Chame a função do Lambda duas vezes usando dados de exemplo para reservar um carro e um hotel.
  - a. Escolha Actions, Configure test event.
  - b. Escolha Lex-Book Hotel (preview) na lista Sample event template.

Esse evento de exemplo corresponde ao modelo de solicitação/resposta do Amazon Lex. Para obter mais informações, consulte [Uso de funções do Lambda \(p. 106\)](#).
  - c. Escolha Save and test.
  - d. Verifique se a função do Lambda foi executada com êxito. A resposta, neste caso, corresponde ao modelo de resposta do Amazon Lex.
  - e. Repita a etapa. Dessa vez, você escolhe Lex-Book Car (preview) na lista Sample event template. A função do Lambda processa a reserva do carro.

Próxima etapa

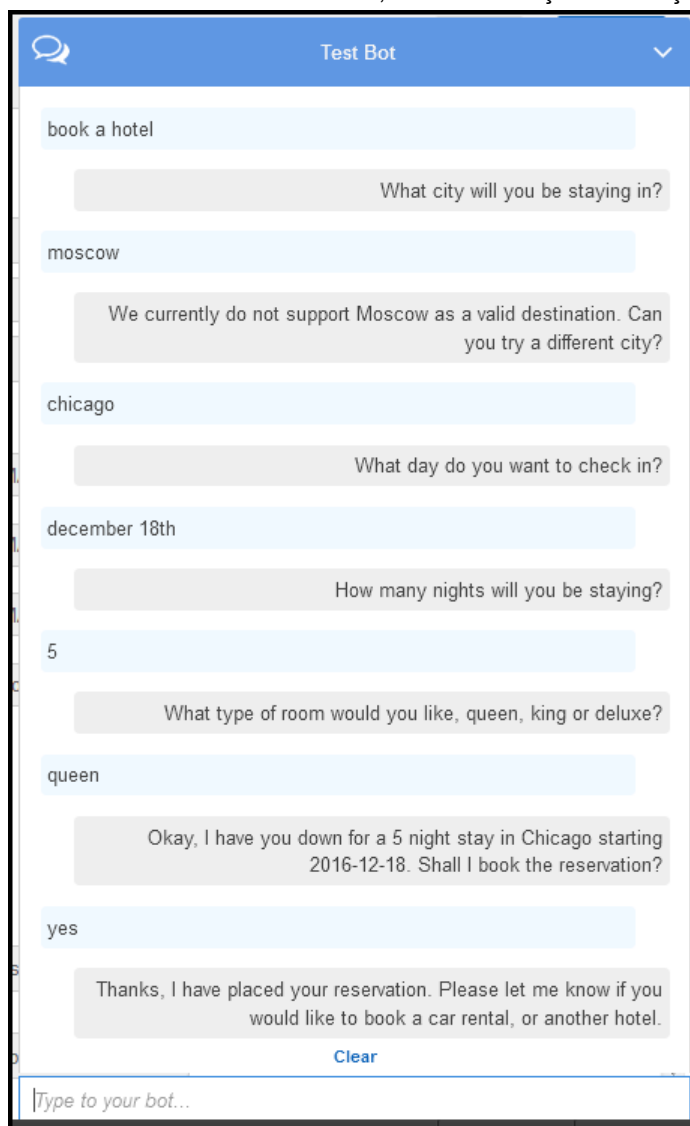
[Etapa 4: Adicionar a função do Lambda como um gancho de código \(p. 160\)](#)

## Etapa 4: Adicionar a função do Lambda como um gancho de código

Nesta seção, você atualiza as configurações das intenções BookCar e BookHotel adicionando a função do Lambda como um gancho de código para as atividades de inicialização/validação e de atendimento. Verifique se você escolheu a versão \$LATEST das intenções, pois só é possível atualizar a versão \$LATEST de seus recursos do Amazon Lex.

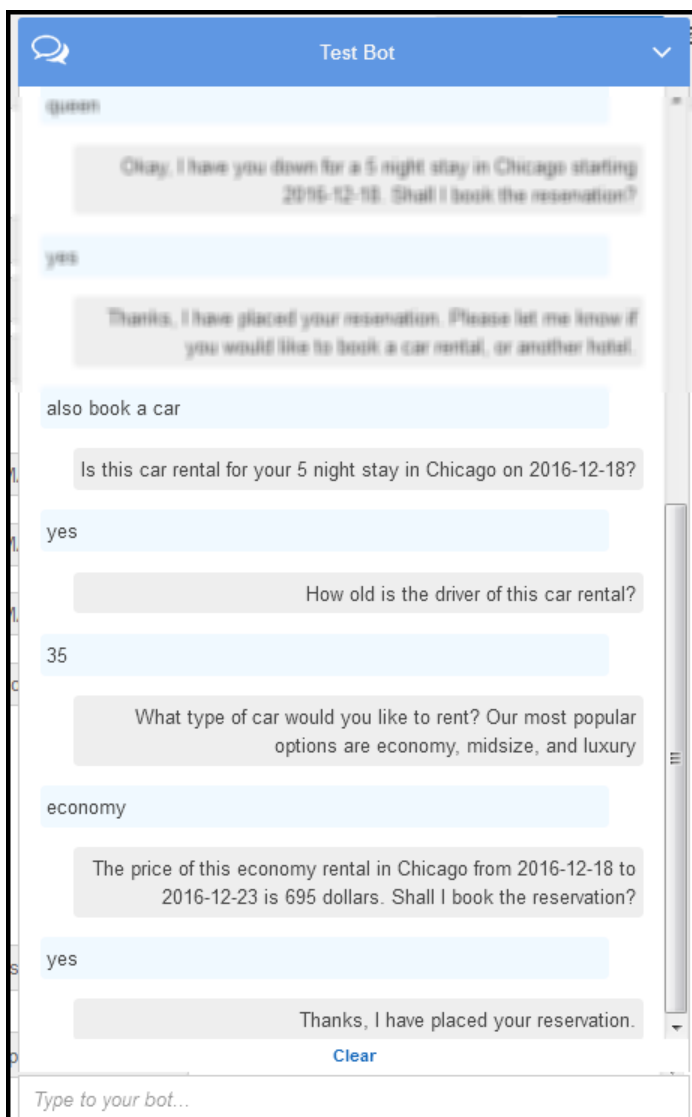
1. No console do Amazon Lex, escolha o bot BookTrip.
2. Na guia Editor, escolha a intenção BookHotel. Atualize a configuração de intenção da seguinte forma:
  - a. Verifique se a versão da intenção (ao lado do nome da intenção) é \$LATEST.
  - b. Adicione a função do Lambda como um gancho de código de inicialização e validação da seguinte forma:

- Em Opções, escolha Initialization and validation code hook.
  - Escolha a função do Lambda na lista.
- c. Adicione a função do Lambda como um gancho de código de atendimento da seguinte forma:
- Em Fulfillment, escolha AWS Lambda function.
  - Escolha a função do Lambda na lista.
  - Escolha Goodbye message e digite uma mensagem.
- d. Escolha Salvar.
3. Na guia Editor, escolha a intenção BookCar. Siga a etapa anterior para adicionar sua função Lambda como hook de código de validação e cumprimento.
4. Escolha Build. O console envia uma série de solicitações ao Amazon Lex para salvar as configurações.
5. Teste o bot. Agora que você tem uma função do Lambda executando a inicialização, a validação dos dados do usuário e o atendimento, verá a diferença na interação do usuário.



Para obter mais informações sobre o fluxo de dados do cliente (console) para o Amazon Lex, e do Amazon Lex para a função do Lambda, consulte [Fluxo de dados: intenção Book Hotel \(p. 163\)](#).

6. Continue a conversa e reserve um carro conforme mostrado a seguir:



Quando você escolhe reservar um carro, o cliente (console) envia uma solicitação ao Amazon Lex que inclui os atributos da sessão (da conversa anterior, BookHotel). O Amazon Lex passa essas informações para a função do Lambda, que, por sua vez, inicializa (ou seja, preenche automaticamente) alguns dos dados do slot de BookCar (ou seja, PickUpDate, ReturnDate e PickUpCity).

#### Note

Isso ilustra como atributos de sessão podem ser usados para manter o contexto nas intenções. O cliente do console fornece o link Clear na janela de teste que um usuário pode usar para limpar atributos de sessão anterior.

Para obter mais informações sobre o fluxo de dados do cliente (console) para o Amazon Lex, e do Amazon Lex para a função do Lambda, consulte [Fluxo de dados: intenção Book Car \(p. 171\)](#).

## Detalhes do fluxo de informações

Neste exercício, você participou de uma conversa com o bot BookTrip do Amazon Lex usando o cliente da janela de teste fornecido no console do Amazon Lex. Esta seção explica o seguinte:

- O fluxo de dados entre o cliente e o Amazon Lex.

A seção supõe que o cliente envia solicitações ao Amazon Lex usando a API de tempo de execução `PostText` e mostra os detalhes da solicitação e da resposta adequadamente. Para obter mais informações sobre a API de runtime `PostText`, consulte [PostText \(p. 355\)](#).

### Note

Para obter um exemplo do fluxo de informações entre o cliente e o Amazon Lex no qual o cliente usa a API `PostContent`, consulte [Etapa 2a \(opcional\): Revise os detalhes do fluxo de informações falado \(console\) \(p. 41\)](#).

- O fluxo de dados entre o Amazon Lex e a função Lambda. Para obter mais informações, consulte [Evento de entrada de função do Lambda e formato de resposta \(p. 106\)](#).

### Tópicos

- [Fluxo de dados: intenção Book Hotel \(p. 163\)](#)
- [Fluxo de dados: intenção Book Car \(p. 171\)](#)

## Fluxo de dados: intenção Book Hotel

Esta seção explica o que acontece após cada entrada do usuário.

1. Usuário: "reservar um hotel"
  - a. O cliente (console) envia a seguinte solicitação [PostText \(p. 355\)](#) ao Amazon Lex:

```
POST /bot/BookTrip/alias/$LATEST/user/wch89kjcpkds8seny7dly5x3otq68j3/text
"Content-Type": "application/json"
"Content-Encoding": "amz-1.0"

{
  "inputText": "book a hotel",
  "sessionAttributes": {}
}
```

O URI e o corpo da solicitação fornecem informações ao Amazon Lex:

- O URI da solicitação – fornece o nome do bot (BookTrip), o alias do bot (`$LATEST`) e o nome do usuário. O `text` final indica que esta é uma solicitação de API `PostText` (e não `PostContent`).
  - Corpo da solicitação – inclui a entrada do usuário (`inputText`) e `sessionAttributes` vazio. Inicialmente, esse é um objeto vazio e a função do Lambda primeiro define os atributos de sessão.
- b. No `inputText`, o Amazon Lex detecta a intenção (BookHotel). Essa intenção é configurada com uma função do Lambda como um gancho de código para inicialização/validação de dados



do usuário. Portanto, o Amazon Lex chama essa função do Lambda passando as seguintes informações como o parâmetro do evento (consulte [Formato de eventos de entrada \(p. 106\)](#)):

```
{
  "messageVersion": "1.0",
  "invocationSource": "DialogCodeHook",
  "userId": "wch89kjgcpkds8seny7dly5x3otq68j3",
  "sessionAttributes": {
  },
  "bot": {
    "name": "BookTrip",
    "alias": null,
    "version": "$LATEST"
  },
  "outputDialogMode": "Text",
  "currentIntent": {
    "name": "BookHotel",
    "slots": {
      "RoomType": null,
      "CheckInDate": null,
      "Nights": null,
      "Location": null
    },
    "confirmationStatus": "None"
  }
}
```

Além das informações enviadas pelo cliente, o Amazon Lex também inclui os seguintes dados adicionais:

- `messageVersion` – atualmente, o Amazon Lex oferece suporte apenas à versão 1.0.
  - `invocationSource` – indica o objetivo da chamada da função Lambda. Nesse caso, o objetivo é executar os dados de inicialização e validação do usuário (nesse momento, o Amazon Lex sabe que o usuário não forneceu todos os dados de slot para atender à intenção).
  - `currentIntent` – todos os valores do slot são definidos como nulos.
- c. No momento, todos os valores de slot são nulos. Não há nada para a função do Lambda validar. A função do Lambda retorna a seguinte resposta ao Amazon Lex. Para obter mais informações sobre o formato de resposta, consulte [Formato de resposta \(p. 109\)](#).

```
{
  "sessionAttributes": {
    "currentReservation": "{ \"ReservationType\": \"Hotel\", \"Location\": null, \"RoomType\": null, \"CheckInDate\": null, \"Nights\": null }"
  },
  "dialogAction": {
    "type": "Delegate",
    "slots": {
      "RoomType": null,
      "CheckInDate": null,
      "Nights": null,
      "Location": null
    }
  }
}
```

#### Note

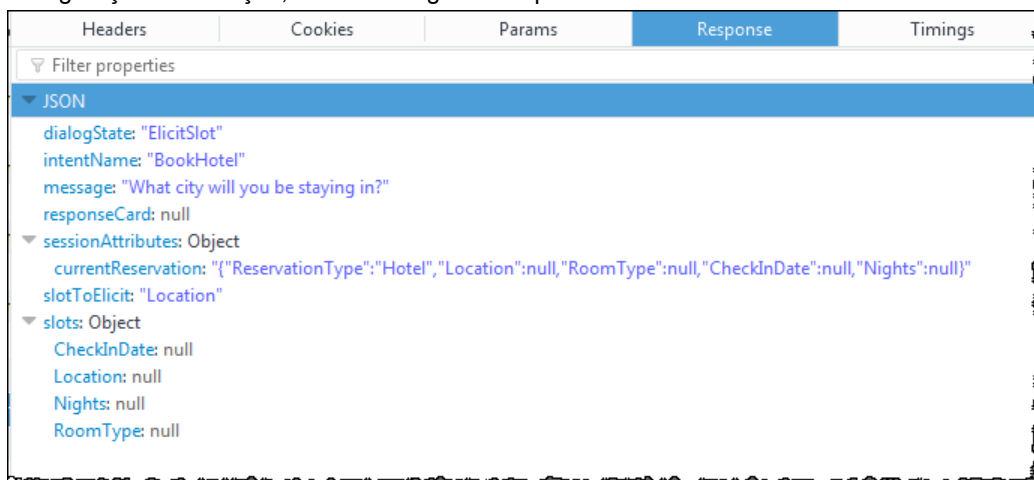
- `currentReservation` – A função do Lambda inclui este atributo de sessão. Seu valor é uma cópia das informações do slot atual e do tipo de reserva.

Somente a função do Lambda e o cliente podem definir e atualizar esses atributos de sessão. O Amazon Lex simplesmente passa esses valores.

- `dialogAction.type` – ao definir esse valor como `Delegate`, a função do Lambda delega a responsabilidade para a próxima ação ao Amazon Lex.

Se a função do Lambda tiver detectado algo na validação dos dados do usuário, ela instruirá o Amazon Lex sobre o que fazer a seguir.

- d. De acordo com o `dialogAction.type`, o Amazon Lex decide a próxima ação — obter dados do usuário para o slot `Location`. Ele seleciona uma das mensagens de solicitação ("What city will you be staying in? (Em que cidade você se hospedará?)" para esse slot, de acordo com a configuração da intenção, e envia a seguinte resposta de volta ao usuário:



Os atributos de sessão são passados para o cliente.

O cliente lê a resposta e, em seguida, exibe a mensagem: "Em qual cidade você se hospedará?"

## 2. Usuário: "Moscou"

- a. O cliente envia a seguinte solicitação `PostText` ao Amazon Lex (quebras de linha adicionadas para legibilidade):

```
POST /bot/BookTrip/alias/$LATEST/user/wch89kjcpkds8seny7dly5x3otq68j3/text
"Content-Type": "application/json"
"Content-Encoding": "amz-1.0"

{
  "inputText": "Moscow",
  "sessionAttributes": {
    "currentReservation": {
      "ReservationType": "Hotel",
      "Location": null,
      "RoomType": null,
      "CheckInDate": null,
      "Nights": null
    }
  }
}
```

Além do `inputText`, o cliente inclui os mesmos atributos de sessão `currentReservation` que recebeu.

- b. Primeiro, o Amazon Lex interpreta o `inputText` no contexto da intenção atual (o serviço lembra que pediu ao usuário específico informações sobre o slot `Location`). Ele atualiza o valor do slot para a intenção atual e chama a função do Lambda usando o evento a seguir:

```
{
  "messageVersion": "1.0",
  "invocationSource": "DialogCodeHook",
  "userId": "wch89kjqcpkds8seny7dly5x3otq68j3",
  "sessionAttributes": {
    "currentReservation": "{\\"ReservationType\\":\\"Hotel\\",\\"Location\\":null,\\\\"RoomType\\":null,\\"CheckInDate\\":null,\\"Nights\\":null}"
  },
  "bot": {
    "name": "BookTrip",
    "alias": null,
    "version": "$LATEST"
  },
  "outputDialogMode": "Text",
  "currentIntent": {
    "name": "BookHotel",
    "slots": {
      "RoomType": null,
      "CheckInDate": null,
      "Nights": null,
      "Location": "Moscow"
    },
    "confirmationStatus": "None"
  }
}
```

#### Note

- `invocationSource` continua sendo `DialogCodeHook`. Nesta etapa, estamos apenas validando os dados do usuário.
  - O Amazon Lex está apenas passando o atributo de sessão para a função do Lambda.
  - Para `currentIntent.slots`, o Amazon Lex atualizou o slot `Location` para `Moscow`.
- c. A função do Lambda executa a validação dos dados do usuário e determina que `Moscow` é um local inválido.

#### Note

A função do Lambda neste exercício tem uma lista simples de cidades válidas e `Moscow` não está na lista. Em uma aplicação de produção, você pode usar um banco de dados de back-end para obter essas informações.

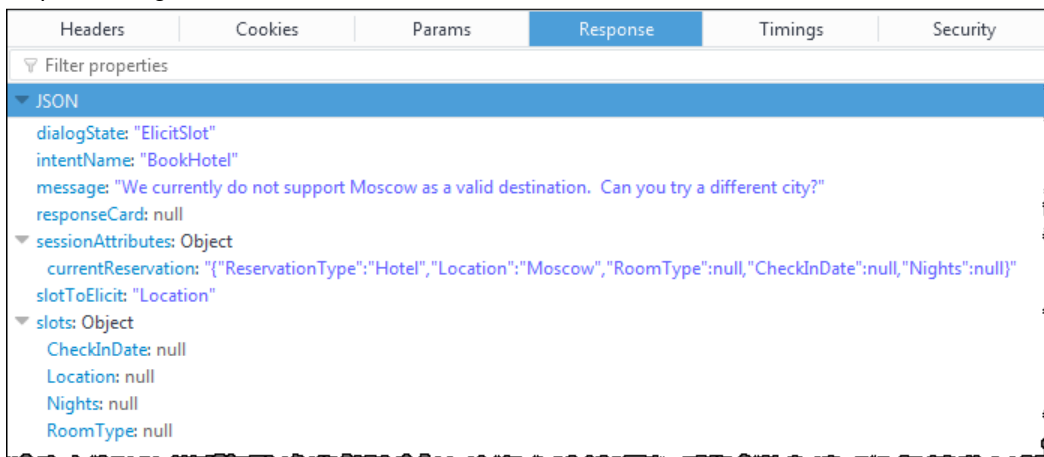
Ele redefine o valor do slot novamente como nulo e direciona o Amazon Lex a solicitar outro valor ao usuário enviando a seguinte resposta:

```
{
  "sessionAttributes": {
    "currentReservation": "{\\"ReservationType\\":\\"Hotel\\",\\"Location\\":\\"Moscow\\",\\"RoomType\\":null,\\"CheckInDate\\":null,\\"Nights\\":null}"
  },
  "dialogAction": {
    "type": "ElicitSlot",
    "intentName": "BookHotel",
    "slots": {
      "RoomType": null,
      "CheckInDate": null,
      "Nights": null,
      "Location": null
    },
    "slotToElicit": "Location",
  }
}
```

```
    "message": {
      "contentType": "PlainText",
      "content": "We currently do not support Moscow as a valid destination.
Can you try a different city?"
    }
  }
}
```

#### Note

- `currentIntent.slots.Location` é redefinido como nulo.
  - `dialogAction.type` é definido como `ElicitSlot`, que direciona o Amazon Lex a solicitar informações ao usuário novamente fornecendo o seguinte:
    - `dialogAction.slotToElicit` – slot para o qual obter dados do usuário.
    - `dialogAction.message` – uma message para transmitir ao usuário.
- d. O Amazon Lex observa o `dialogAction.type` e passa as informações para o cliente na resposta a seguir:



O cliente simplesmente exibe a mensagem: "No momento, não oferecemos suporte a Moscou como um destino válido. Poderia tentar outra cidade?"

### 3. Usuário: "Chicago"

- a. O cliente envia a seguinte solicitação `PostText` ao Amazon Lex:

```
POST /bot/BookTrip/alias/$LATEST/user/wch89kjcpkds8seny7dly5x3otq68j3/text
"Content-Type": "application/json"
"Content-Encoding": "amz-1.0"

{
  "inputText": "Chicago",
  "sessionAttributes": {
    "currentReservation": "{\"ReservationType\": \"Hotel\",
                          \"Location\": \"Moscow\",
                          \"RoomType\": null,
                          \"CheckInDate\": null,
                          \"Nights\": null}"
  }
}
```

- b. O Amazon Lex conhece o contexto para o qual estava escolhendo dados para o slot `Location`. Nesse contexto, ele sabe que o valor `inputText` é para o slot `Location`. Em seguida, ele chama a função do Lambda enviando o seguinte evento:

```
{
  "messageVersion": "1.0",
  "invocationSource": "DialogCodeHook",
  "userId": "wch89kjqcpkds8seny7dly5x3otq68j3",
  "sessionAttributes": {
    "currentReservation": "{\\"ReservationType\\":\\"Hotel\\",\\"Location\\":Moscow,\\\"RoomType\\":null,\\\"CheckInDate\\":null,\\\"Nights\\":null}"
  },
  "bot": {
    "name": "BookTrip",
    "alias": null,
    "version": "$LATEST"
  },
  "outputDialogMode": "Text",
  "currentIntent": {
    "name": "BookHotel",
    "slots": {
      "RoomType": null,
      "CheckInDate": null,
      "Nights": null,
      "Location": "Chicago"
    },
    "confirmationStatus": "None"
  }
}
```

O Amazon Lex atualizou o `currentIntent.slots` definindo o slot `Location` como `Chicago`.

- c. De acordo com o valor `invocationSource` de `DialogCodeHook`, a função do Lambda executa a validação dos dados do usuário. Ela reconhece `Chicago` como um valor de slot válido, atualiza o atributo de sessão adequadamente e, em seguida, retorna a seguinte resposta ao Amazon Lex.

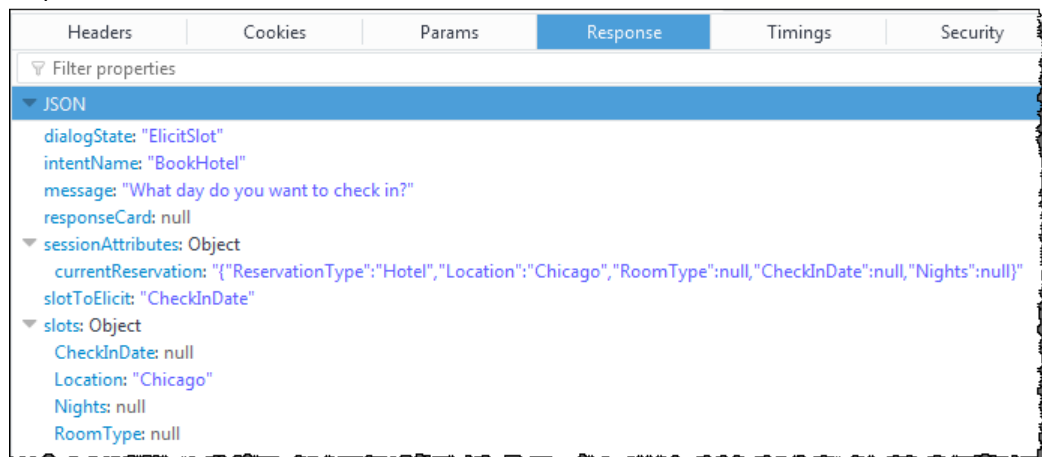
```
{
  "sessionAttributes": {
    "currentReservation": "{\\"ReservationType\\":\\"Hotel\\",\\"Location\\":\\\"Chicago\\",\\"RoomType\\":null,\\\"CheckInDate\\":null,\\\"Nights\\":null}"
  },
  "dialogAction": {
    "type": "Delegate",
    "slots": {
      "RoomType": null,
      "CheckInDate": null,
      "Nights": null,
      "Location": "Chicago"
    }
  }
}
```

#### Note

- `currentReservation` – a função do Lambda atualiza esse atributo de sessão definindo o `Location` como `Chicago`.
- `dialogAction.type` – é definido como `Delegate`. Os dados do usuário eram válidos e a função do Lambda direciona o Amazon Lex a escolher a próxima ação.

- d. De acordo com `dialogAction.type`, o Amazon Lex escolhe a próxima ação. O Amazon Lex sabe que precisa de mais dados de slot e escolhe o próximo slot não preenchido (`CheckInDate`) com a mais alta prioridade de acordo com a configuração de intenção. Ele seleciona uma das

mensagens de solicitação ("What day do you want to check in? (Em que dia você deseja fazer check-in?)" para esse slot, de acordo com a configuração da intenção, e envia a seguinte resposta de volta ao cliente:



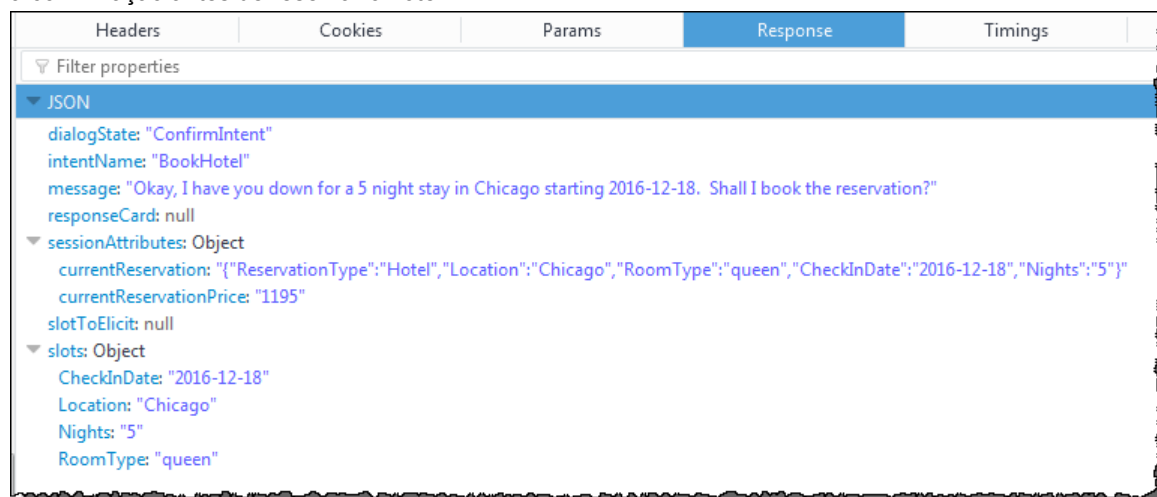
O cliente exibe a mensagem: "Em que dia você deseja fazer check-in?"

4. A interação do usuário continua — o usuário fornece os dados, a função do Lambda valida os dados e, em seguida, delega a próxima ação para o Amazon Lex. Por fim, o usuário fornece todos os dados do slot, a função do Lambda valida todas as entradas do usuário e, em seguida, o Amazon Lex reconhece que tem todos os dados do slot.

#### Note

Neste exercício, depois que o usuário fornece todos os dados do slot, a função do Lambda calcula o preço da reserva de hotel e o retorna como outro atributo de sessão (`currentReservationPrice`).

Neste momento, a intenção está pronta para ser atendida, mas a intenção `BookHotel` está configurada com uma solicitação de confirmação que exige a confirmação do usuário para que o Amazon Lex possa atender à intenção. Portanto, o Amazon Lex envia a seguinte mensagem ao cliente solicitando a confirmação antes de reservar o hotel:



O cliente exibe a mensagem: "OK, reservarei para você 5 diárias em Chicago a partir de 18/12/2016. Posso fazer a reserva?"

5. Usuário: "sim"

- a. O cliente envia a seguinte solicitação PostText ao Amazon Lex:

```
POST /bot/BookTrip/alias/$LATEST/user/wch89kjqcpkds8seny7dly5x3otq68j3/text
"Content-Type": "application/json"
"Content-Encoding": "amz-1.0"

{
  "inputText": "Yes",
  "sessionAttributes": {
    "currentReservation": {
      "ReservationType": "Hotel",
      "Location": "Chicago",
      "RoomType": "queen",
      "CheckInDate": "2016-12-18",
      "Nights": "5"
    },
    "currentReservationPrice": "1195"
  }
}
```

- b. O Amazon Lex interpreta o inputText no contexto da confirmação da intenção atual. O Amazon Lex entende que o usuário deseja prosseguir com a reserva. Dessa vez, o Amazon Lex chama a função do Lambda para atender à intenção enviando o seguinte evento. Ao definir invocationSource como FulfillmentCodeHook no evento, ele envia à função do Lambda. O Amazon Lex também define o confirmationStatus como Confirmed.

```
{
  "messageVersion": "1.0",
  "invocationSource": "FulfillmentCodeHook",
  "userId": "wch89kjqcpkds8seny7dly5x3otq68j3",
  "sessionAttributes": {
    "currentReservation": {
      "ReservationType": "Hotel",
      "Location": "Chicago",
      "RoomType": "queen",
      "CheckInDate": "2016-12-18",
      "Nights": "5"
    },
    "currentReservationPrice": "956"
  },
  "bot": {
    "name": "BookTrip",
    "alias": null,
    "version": "$LATEST"
  },
  "outputDialogMode": "Text",
  "currentIntent": {
    "name": "BookHotel",
    "slots": {
      "RoomType": "queen",
      "CheckInDate": "2016-12-18",
      "Nights": "5",
      "Location": "Chicago"
    }
  },
  "confirmationStatus": "Confirmed"
}
```

Note

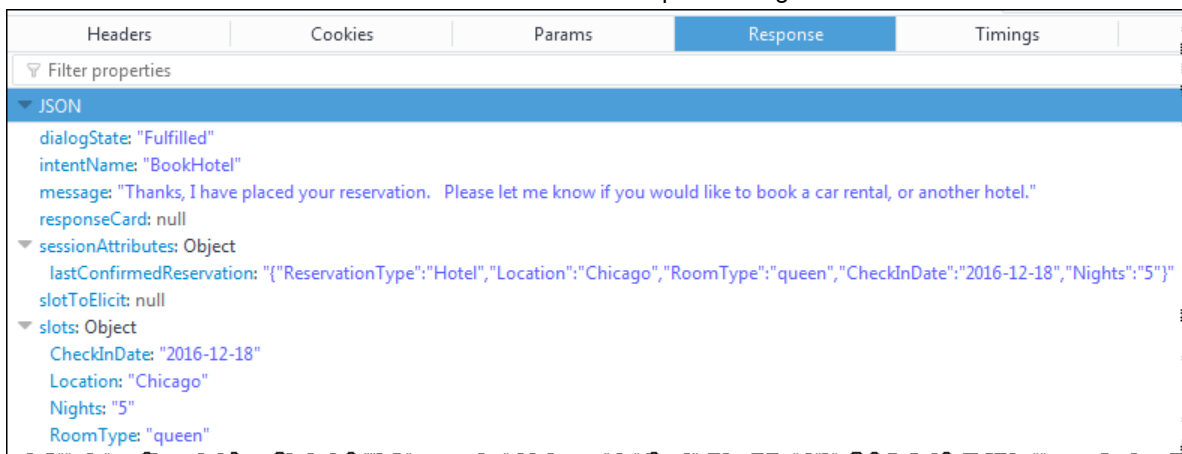
- invocationSource – dessa vez, o Amazon Lex define esse valor como FulfillmentCodeHook, direcionando a função do Lambda a atender à intenção.
  - confirmationStatus – é definido como Confirmed.
- c. Dessa vez, a função do Lambda atende à intenção BookHotel, o Amazon Lex conclui a reserva e, em seguida, retorna a seguinte resposta:

```
{
  "sessionAttributes": {
    "lastConfirmedReservation": "{\"ReservationType\":\"Hotel\",\"Location\":\"Chicago\",\"RoomType\":\"queen\",\"CheckInDate\":\"2016-12-18\",\"Nights\":\"5\"}"
  },
  "dialogAction": {
    "type": "Close",
    "fulfillmentState": "Fulfilled",
    "message": {
      "contentType": "PlainText",
      "content": "Thanks, I have placed your reservation. Please let me know if you would like to book a car rental, or another hotel."
    }
  }
}
```

#### Note

- `lastConfirmedReservation` – é um novo atributo de sessão que a função do Lambda adicionou (em vez de `currentReservation`, `currentReservationPrice`).
- `dialogAction.type` – a função do Lambda define esse valor como `Close` indicando ao Amazon Lex para não esperar uma resposta do usuário.
- `dialogAction.fulfillmentState` – é definido como `Fulfilled` e inclui uma mensagem adequada para transmitir para o usuário.

d. O Amazon Lex analisa o `fulfillmentState` e envia a resposta a seguir ao cliente:



#### Note

- `dialogState` – o Amazon Lex define esse valor como `Fulfilled`.
- `message` – é a mesma mensagem que a função do Lambda forneceu.

O cliente exibe a mensagem.

## Fluxo de dados: intenção Book Car

O bot BookTrip neste exercício oferece suporte a duas intenções (BookHotel e BookCar). Depois de reservar um hotel, o usuário pode continuar a conversa para reservar um carro. Desde que a sessão não



tenha expirado, em cada solicitação subsequente o cliente continua a enviar os atributos de sessão (neste exemplo, o `lastConfirmedReservation`). A função do Lambda pode usar essas informações para inicializar os dados do slot para a intenção `BookCar`. Isso mostra como é possível usar atributos de sessão no compartilhamento de informações entre intenções.

Especificamente, quando o usuário escolhe a intenção `BookCar`, a função do Lambda usa as informações relevantes no atributo de sessão para preencher os slots automaticamente (`PickUpDate`, `ReturnDate` e `PickUpCity`) para a intenção `BookCar`.

#### Note

O console do Amazon Lex fornece o link `Clear` (Limpar) que você pode usar para limpar os atributos da sessão anterior.

Siga as etapas neste procedimento para continuar a conversa.

1. Usuário: "também reservar um carro"
  - a. O cliente envia a seguinte solicitação `PostText` ao Amazon Lex.

```
POST /bot/BookTrip/alias/$LATEST/user/wch89kjqcpkds8seny7dly5x3otq68j3/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
  "inputText":"also book a car",
  "sessionAttributes":{
    "lastConfirmedReservation":{"ReservationType\":"Hotel\","Location\":"Chicago\","RoomType\":"queen\","CheckInDate\":"2016-12-18\","Nights\":"5\"}
  }
}
```

O cliente inclui o atributo de sessão `lastConfirmedReservation`.

- b. O Amazon Lex detecta a intenção (`BookCar`) a partir do `inputText`. Essa intenção também está configurada para chamar a função do Lambda para executar a inicialização e a validação dos dados do usuário. O Amazon Lex invoca a função do Lambda com o seguinte evento:

```
{
  "messageVersion": "1.0",
  "invocationSource": "DialogCodeHook",
  "userId": "wch89kjqcpkds8seny7dly5x3otq68j3",
  "sessionAttributes": {
    "lastConfirmedReservation": "{\\"ReservationType\\":\\"Hotel\\",\\"Location\\":\\"Chicago\\",\\"RoomType\\":\\"queen\\",\\"CheckInDate\\":\\"2016-12-18\\",\\"Nights\\":\\"5\\"}"
  },
  "bot": {
    "name": "BookTrip",
    "alias": null,
    "version": "$LATEST"
  },
  "outputDialogMode": "Text",
  "currentIntent": {
    "name": "BookCar",
    "slots": {
      "PickUpDate": null,
      "ReturnDate": null,
      "DriverAge": null,
      "CarType": null,

```

```
        "PickUpCity": null
      },
      "confirmationStatus": "None"
    }
  }
}
```

#### Note

- `messageVersion` – atualmente, o Amazon Lex oferece suporte apenas à versão 1.0.
  - `invocationSource` – indica que a finalidade de invocação é executar a inicialização e a validação dos dados do usuário.
  - `currentIntent` – inclui o nome da intenção e os slots. No momento, todos os valores de slot são nulos.
- c. A função do Lambda observa todos os valores de slot nulos com nada para validar. No entanto, ela usa atributos de sessão para inicializar alguns dos valores de slot (`PickUpDate`, `ReturnDate` e `PickUpCity`) e, em seguida, retorna a seguinte resposta:

```
{
  "sessionAttributes": {
    "lastConfirmedReservation": "{\n\"ReservationType\": \"Hotel\", \"Location\n\n\": \"Chicago\", \"RoomType\": \"queen\", \"CheckInDate\": \"2016-12-18\", \"Nights\":\n\n\"5\"}",
    "currentReservation": "{\n\"ReservationType\": \"Car\", \"PickUpCity\": null,\n\n\"PickUpDate\": null, \"ReturnDate\": null, \"CarType\": null}",
    "confirmationContext": "AutoPopulate"
  },
  "dialogAction": {
    "type": "ConfirmIntent",
    "intentName": "BookCar",
    "slots": {
      "PickUpCity": "Chicago",
      "PickUpDate": "2016-12-18",
      "ReturnDate": "2016-12-22",
      "CarType": null,
      "DriverAge": null
    },
    "message": {
      "contentType": "PlainText",
      "content": "Is this car rental for your 5 night stay in Chicago on\n\n2016-12-18?"
    }
  }
}
```

#### Note

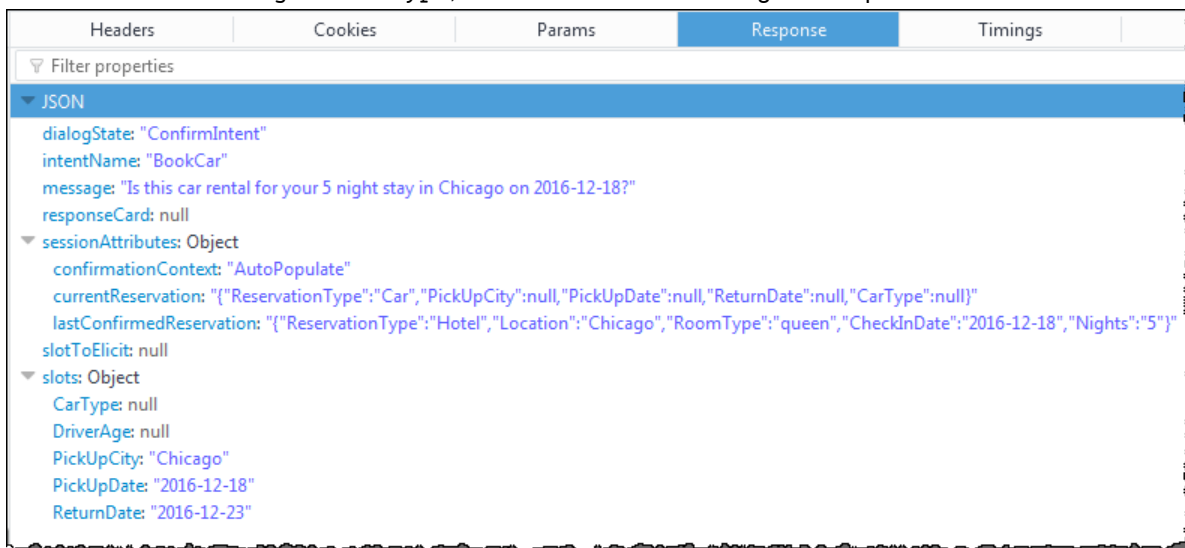
- Além do `lastConfirmedReservation`, a função do Lambda inclui mais atributos de sessão (`currentReservation` e `confirmationContext`).
- `dialogAction.type` é definido como `ConfirmIntent`, que informa ao Amazon Lex que uma resposta sim ou não é esperada do usuário (o `confirmationContext` definido como `AutoPopulate`, a função Lambda sabe que a resposta sim/não do usuário é para obter a confirmação do usuário da inicialização que a função do Lambda executou (dados de slot preenchidos automaticamente)).

A função do Lambda também inclui uma mensagem informativa na resposta na `dialogAction.message` para o Amazon Lex retornar ao cliente.

## Note

O termo `ConfirmIntent` (valor do `dialogAction.type`) não é relacionado a nenhuma intenção de bot. No exemplo, a função do Lambda usa esse termo para direcionar o Amazon Lex para obter uma resposta sim/não do usuário.

- d. De acordo com o `dialogAction.type`, o Amazon Lex retorna a seguinte resposta ao cliente:



O cliente exibe a mensagem: "Este aluguel de carro é para sua estadia de 5 diárias em Chicago em 18/12/2016?"

## 2. Usuário: "sim"

- a. O cliente envia a seguinte solicitação `PostText` ao Amazon Lex.

```
POST /bot/BookTrip/alias/$LATEST/user/wch89kjqcpkds8seny7dly5x3otq68j3/text
"Content-Type": "application/json"
"Content-Encoding": "amz-1.0"

{
  "inputText": "yes",
  "sessionAttributes": {
    "confirmationContext": "AutoPopulate",
    "currentReservation": "{ \"ReservationType\": \"Car\",
                          \"PickUpCity\": null,
                          \"PickUpDate\": null,
                          \"ReturnDate\": null,
                          \"CarType\": null }",
    "lastConfirmedReservation": "{ \"ReservationType\": \"Hotel\",
                                \"Location\": \"Chicago\",
                                \"RoomType\": \"queen\",
                                \"CheckInDate\": \"2016-12-18\",
                                \"Nights\": \"5\" }"
  }
}
```

- b. O Amazon Lex lê o `inputText` e conhece o contexto (solicitou ao usuário para confirmar o preenchimento automático). O Amazon Lex invoca a função do Lambda enviando o seguinte evento:

```
{
  "messageVersion": "1.0",
  "invocationSource": "DialogCodeHook",
  "userId": "wch89kjqcpkds8seny7dly5x3otq68j3",
  "sessionAttributes": {
    "confirmationContext": "AutoPopulate",
    "currentReservation": "{\\"ReservationType\\":\\"Car\\",\\"PickUpCity\\":null,\\\\"PickUpDate\\":null,\\"ReturnDate\\":null,\\"CarType\\":null}",
    "lastConfirmedReservation": "{\\"ReservationType\\":\\"Hotel\\",\\"Location\\":\\"Chicago\\",\\"RoomType\\":\\"queen\\",\\"CheckInDate\\":\\"2016-12-18\\",\\"Nights\\":\\"5\\"}"
  },
  "bot": {
    "name": "BookTrip",
    "alias": null,
    "version": "$LATEST"
  },
  "outputDialogMode": "Text",
  "currentIntent": {
    "name": "BookCar",
    "slots": {
      "PickUpDate": "2016-12-18",
      "ReturnDate": "2016-12-22",
      "DriverAge": null,
      "CarType": null,
      "PickUpCity": "Chicago"
    },
    "confirmationStatus": "Confirmed"
  }
}
```

Como o usuário respondeu Sim, o Amazon Lex define o `confirmationStatus` como `Confirmed`.

- c. A partir do `confirmationStatus`, a função do Lambda sabe que os valores preenchidos automaticamente estão corretos. A função do Lambda faz o seguinte:
- Atualiza o atributo de sessão `currentReservation` para o valor de slot que tinha preenchido automaticamente.
  - Define o `dialogAction.type` como `ElicitSlot`
  - Define o valor de `slotToElicit` como `DriverAge`.

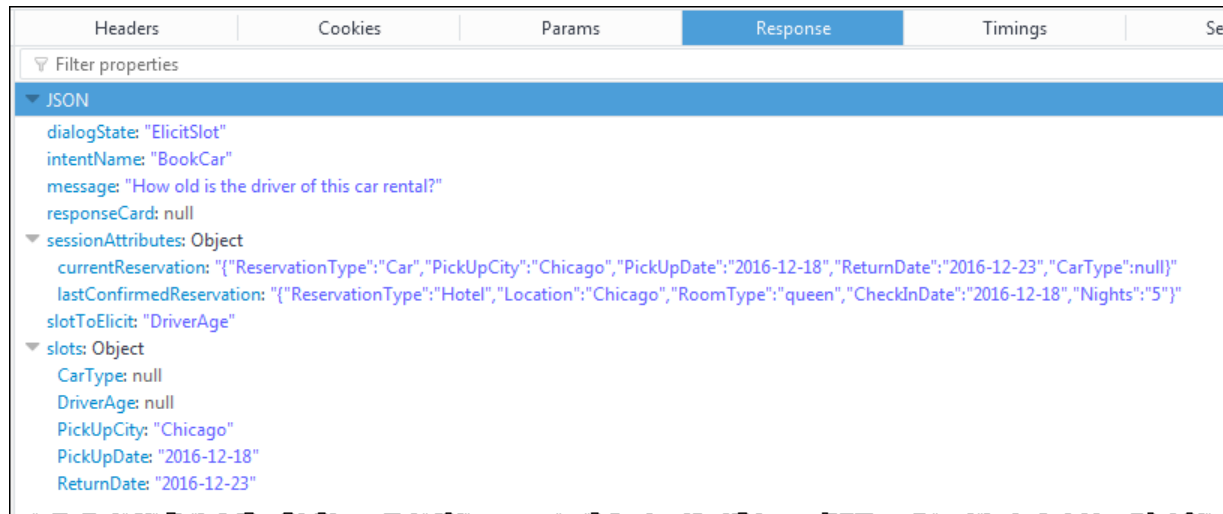
A resposta seguinte é enviada:

```
{
  "sessionAttributes": {
    "currentReservation": "{\\"ReservationType\\":\\"Car\\",\\"PickUpCity\\":\\"Chicago\\",\\"PickUpDate\\":\\"2016-12-18\\",\\"ReturnDate\\":\\"2016-12-22\\",\\"CarType\\":null}",
    "lastConfirmedReservation": "{\\"ReservationType\\":\\"Hotel\\",\\"Location\\":\\"Chicago\\",\\"RoomType\\":\\"queen\\",\\"CheckInDate\\":\\"2016-12-18\\",\\"Nights\\":\\"5\\"}"
  },
  "dialogAction": {
    "type": "ElicitSlot",
    "intentName": "BookCar",
    "slots": {
      "PickUpDate": "2016-12-18",
      "ReturnDate": "2016-12-22",
      "DriverAge": null,
      "CarType": null,

```

```
        "PickUpCity": "Chicago"
      },
      "slotToElicit": "DriverAge",
      "message": {
        "contentType": "PlainText",
        "content": "How old is the driver of this car rental?"
      }
    }
  }
}
```

- d. O Amazon Lex retorna a seguinte resposta:



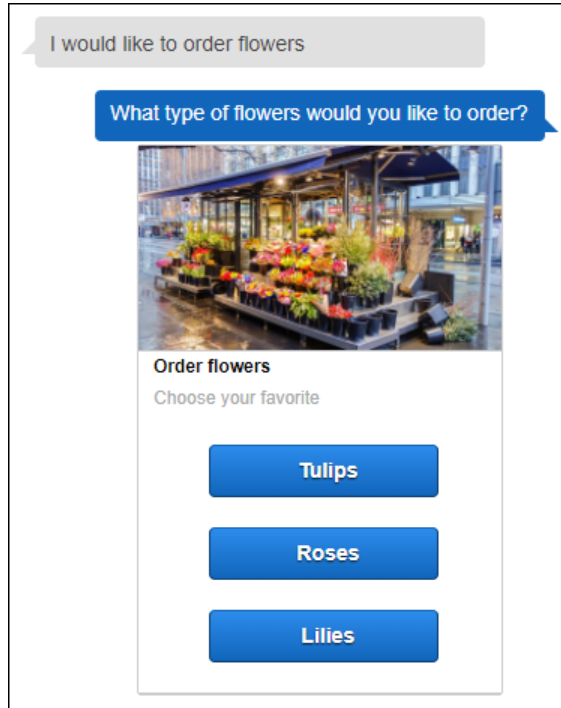
O cliente exibe a mensagem "How old is the driver of this car rental? (Qual é a idade do motorista do carro alugado?)" e a conversa continua.

## Exemplo: uso de um cartão de resposta

Neste exercício, o Exercício 1 dos Conceitos básicos é estendido adicionando um cartão de resposta. Crie um bot que oferece suporte à intenção `OrderFlowers` e, em seguida, atualize a intenção adicionando um cartão de resposta para o slot `FlowerType`. Além do seguinte prompt para o slot `FlowerType`, o usuário pode escolher os tipos de flor do cartão de resposta:

What type of flowers would you like to order?

O cartão de resposta está a seguir:



O usuário do bot pode digitar o texto ou escolher a partir de uma lista de tipos de flor. Este cartão de resposta é configurado com uma imagem, que aparece no cliente como mostrado. Para obter mais informações sobre cartões de resposta, consulte [Cartões de resposta \(p. 16\)](#).

Como criar e testar um bot com um cartão de resposta:

1. Siga o Exercício 1 dos Conceitos básicos para criar e testar um bot de OrderFlowers. Você deve concluir as etapas 1, 2 e 3. Não é necessário adicionar uma função do Lambda para testar o cartão de resposta. Para obter instruções, consulte [Exercício 1: Criar um bot do Amazon Lex usando um esquema \(console\) \(p. 38\)](#).
2. Atualize o bot adicionando o cartão de resposta e, em seguida, publique uma versão. Quando for publicar uma versão, especifique um alias (BETA) para apontar para ela.
  - a. No console do Amazon Lex, escolha seu bot.
  - b. Escolha a intenção `OrderFlowers`.
  - c. Escolha o ícone de engrenagem de configuração próximo à Prompt "What type of flowers" para configurar um cartão de resposta para o `FlowerType`.

The screenshot shows the Amazon Lex console interface for configuring an intent named "OrderFlowersWithRC". The interface has tabs for "Editor", "Settings", "Channels", and "Monitoring". The "Editor" tab is selected, showing the "OrderFlowers" intent. Under "Sample utterances", there are three examples: "e.g. I would like to book a flight.", "I would like to pick up flowers", and "I would like to order some flowers". Below this, the "Slots" section is displayed as a table with columns for Priority, Required, Name, Slot type, and Prompt. The first slot is "Location" (e.g. Location), which is required and has a slot type of "e.g. A...". The second slot is "FlowerType" (FlowerType), which is required and has a slot type of "Flower...". The third slot is "PickupDate" (PickupDate), which is required and has a slot type of "AMAZ...". The fourth slot is "PickupTime" (PickupTime), which is required and has a slot type of "AMAZ...". A red box highlights the settings icon for the first slot, "Location".

| Priority | Required | Name          | Slot type | Prompt          |
|----------|----------|---------------|-----------|-----------------|
|          |          | e.g. Location | e.g. A... | e.g. What city? |
| 1.       | ✓        | FlowerType    | Flower... | Latest          |
| 2.       | ✓        | PickupDate    | AMAZ...   | Built-in        |
| 3.       | ✓        | PickupTime    | AMAZ...   | Built-in        |

- d. Forneça um título ao cartão e configure três botões conforme mostrado na captura de tela a seguir. Opcionalmente, você pode adicionar uma imagem ao cartão de resposta, desde que você tenha um URL da imagem. Se estiver implantando um bot usando SMS do Twilio, deverá fornecer um URL da imagem.

Prompt response cards

Card 1 ⓘ

Preview as: Facebook ▼

🗑️

Image URL\* *e.g. http://www.example.com/image.png*

Title\* What type of flowers?

Subtitle\*

Button title\* Tulips ✕

Button value\* tulips ▼

Button title Lilies ✕

Button value lilies ▼

Button title Roses ✕

Button value roses ▼

+

 Add Card

- e. Escolha Save (Salvar) para salvar o cartão de resposta.
  - f. Escolha Save intent (Salvar intenção) para salvar a configuração da intenção.
  - g. Para criar o bot, escolha Build.
  - h. Para publicar uma versão do bot, escolha Publish. Especifique BETA como um alias que aponta para a versão do bot. Para obter informações sobre versionamento, consulte [Controle de versão e aliases](#) (p. 102).
3. Implante o bot em uma plataforma de mensagens:
- Implante o bot na plataforma do Facebook Messenger e teste a integração. Para obter instruções, consulte [Integração de um bot do Amazon Lex com o Facebook Messenger](#) (p. 116). Quando você encomenda flores, a janela de mensagem mostra o cartão de resposta para que você possa escolher um tipo de flor.
  - Implante o bot na plataforma Slack e teste a integração. Para obter instruções, consulte [Integração de um bot do Amazon Lex com o Slack](#) (p. 121). Quando você encomenda flores, a janela de mensagem mostra o cartão de resposta para que você possa escolher um tipo de flor.
  - Implante o bot na plataforma de SMS Twilio. Para obter instruções, consulte [Integração de um bot do Amazon Lex com o SMS programável do Twilio](#) (p. 125). Ao encomendar flores, a mensagem do Twilio mostra a imagem do cartão de resposta. O SMS do Twilio não comporta botões na resposta.



## Exemplo: Atualização de enunciados

Neste exercício, adicione enunciados adicionais aos que você criou no Exercício 1 de Conceitos básicos. Use a guia Monitoring (Monitoramento) no console do Amazon Lex para visualizar os enunciados que seu bot não reconheceu. Para melhorar a experiência dos usuários, adicione esses enunciados ao bot.

### Note

As estatísticas de enunciado são geradas uma vez por dia. Você pode ver o enunciado que não foi reconhecido, quantas vezes ele foi ouvido, e a última data e hora em que foi ouvido. Pode levar até 24 horas para o enunciado perdido aparecer no console.

Você pode ver enunciados para diferentes versões do bot. Para alterar a versão do bot para a qual você está vendo os enunciados, escolha uma versão diferente na lista vertical ao lado do nome do bot.

Para visualizar e adicionar os enunciados perdidos a um bot:

1. Siga a primeira etapa do Exercício 1 de Conceitos básicos para criar e testar um bot `OrderFlowers`. Para obter instruções, consulte [Exercício 1: Criar um bot do Amazon Lex usando um esquema \(console\)](#) (p. 38).
2. Teste o bot digitando o seguinte enunciado na janela Test Bot. Digite cada enunciado várias vezes. O bot de exemplo não reconhece os seguintes enunciados:
  - Pedir flores
  - Obtenha flores para mim
  - Peça flores
  - Obtenha algumas flores para mim
3. Aguarde o Amazon Lex coletar os dados de uso sobre os enunciados perdidos. Os dados do enunciado são gerados uma vez por dia, geralmente durante a noite.
4. Faça login no Console de gerenciamento da AWS e abra o console do Amazon Lex em <https://console.aws.amazon.com/lex/>.
5. Escolha o bot `OrderFlowers`.
6. Escolha a guia Monitoring, selecione Utterances no menu esquerdo e, em seguida, escolha o botão Missed. O painel mostra um máximo de 100 enunciados perdidos.

| Utterances                  |                      |         |        |                              |
|-----------------------------|----------------------|---------|--------|------------------------------|
| Add utterance to Intent ▼   |                      |         |        |                              |
| Filter: 🔍 Filter by keyword |                      |         |        |                              |
| Detected Missed             |                      |         |        |                              |
| <input type="checkbox"/>    | Utterances ▼         | Count ▼ | Status | Last said date ▼             |
| <input type="checkbox"/>    | I want flowers       | 5       | Missed | April 21, 2017 at 10:28:13 A |
| <input type="checkbox"/>    | Order flowers        | 4       | Missed | April 21, 2017 at 10:28:05 A |
| <input type="checkbox"/>    | Get me some flowers  | 2       | Missed | April 21, 2017 at 10:27:49 A |
| <input type="checkbox"/>    | Get me flowers       | 2       | Missed | April 21, 2017 at 10:27:25 A |
| <input type="checkbox"/>    | Please order flowers | 1       | Missed | April 21, 2017 at 10:26:55 A |
| <input type="checkbox"/>    | get me some flowers  | 1       | Missed | April 21, 2017 at 10:27:18 A |

7. Para escolher os enunciados perdidos que você deseja adicionar ao bot, selecione a caixa de seleção próxima a eles. Para adicionar o enunciado à versão `$LATEST` da intenção, escolha a seta para baixo ao lado da lista suspensa Add utterance to intent e, em seguida, escolha a intenção.
8. Para recriar o bot, escolha Build e, em seguida, Build novamente.
9. Para verificar se o seu bot reconhece o novo enunciado, use o painel Test Bot.

## Exemplo: Integração com um site

Neste exemplo, você integra um bot com um site usando texto e voz. Você usa o JavaScript e os serviços da AWS para criar uma experiência interativa para os visitantes do site. Você pode escolher entre esses exemplos documentados no [Blog de IA da AWS](#):

- [Implantar uma interface de usuário da Web para seu Chatbot](#) — demonstra uma interface de usuário da Web completa que fornece um cliente da Web para chatbots do Amazon Lex. Você pode usar isso para saber mais sobre clientes da Web, ou como um bloco de criação para sua própria aplicação.
- ["Saudações, visitante!" — interaja com os usuários da web com o Amazon Lex](#). O — demonstra o uso do Amazon Lex, o AWS SDK para JavaScript no navegador e o Amazon Cognito para criar uma experiência de conversa no site.
- [Captura de entrada de voz em um navegador e seu envio para o Amazon Lex](#) — demonstra como incorporar um chatbot com base em voz em um site usando o SDK para JavaScript no navegador. O aplicativo grava o áudio, envia o áudio ao Amazon Lex e, em seguida, reproduz a resposta.

# Segurança no Amazon Lex

A segurança da nuvem na AWS é a nossa maior prioridade. Como cliente da AWS, você se beneficiará de um datacenter e de uma arquitetura de rede criados para atender aos requisitos das empresas com as maiores exigências de segurança.

A segurança é uma responsabilidade compartilhada entre a AWS e você. O [modelo de responsabilidade compartilhada](#) descreve a segurança da nuvem e a segurança na nuvem:

- **Segurança da nuvem** – a AWS é responsável pela proteção da infraestrutura que executa serviços da AWS na nuvem da AWS. A AWS também fornece serviços que você pode usar com segurança. A eficácia da nossa segurança é regularmente testada e verificada por auditores terceirizados como parte dos [programas de conformidade da AWS](#). Para saber mais sobre os programas de conformidade que se aplicam ao Amazon Lex, consulte [Serviços da AWS no escopo por programa de conformidade](#).
- **Segurança na nuvem** – Sua responsabilidade é determinada pelo serviço da AWS que você usa. Você também é responsável por outros fatores, incluindo a confidencialidade de seus dados, os requisitos da sua organização e as leis e regulamentos aplicáveis.

Esta documentação ajudará você a entender como aplicar o modelo de responsabilidade compartilhada ao usar o Amazon Lex. Os tópicos a seguir mostram como configurar o Amazon Lex para cumprir os objetivos de segurança e conformidade. Você também aprenderá como usar outros serviços da AWS que podem ajudar a monitorar e proteger seus recursos do Amazon Lex.

## Tópicos

- [Proteção de dados no Amazon Lex \(p. 182\)](#)
- [Identity and Access Management para Amazon Lex \(p. 184\)](#)
- [Monitorar o Amazon Lex \(p. 199\)](#)
- [Validação de conformidade do Amazon Lex \(p. 207\)](#)
- [Resiliência no Amazon Lex \(p. 208\)](#)
- [Segurança da infraestrutura no Amazon Lex \(p. 208\)](#)

## Proteção de dados no Amazon Lex

O Amazon Lex coleta o conteúdo do cliente para solução de problemas e para ajudar a melhorar o serviço. O conteúdo do cliente é protegido por padrão. Você pode excluir o conteúdo para clientes individuais usando a API do Amazon Lex.

O Amazon Lex armazena quatro tipos de conteúdo:

- Enunciados de amostra, que são usados para criar e treinar um bot
- Enunciados do cliente de usuários que interagem com o bot
- Atributos de sessão, que fornecem informações específicas do aplicativo para a duração de uma interação do usuário com um bot
- Atributos de solicitação, que contêm informações que se aplicam a uma única solicitação a um bot

Qualquer bot do Amazon Lex criado para uso por crianças é regido pelo Children's Online Privacy Protection Act (COPPA). Você informa o Amazon Lex que o bot está sujeito a COPPA usando o

console ou a API do Amazon Lex para definir o campo `childDirected` como `true`. Quando o campo `childDirected` é definido como `true`, nenhum enunciado de usuário está armazenado.

#### Tópicos

- [Criptografia em repouso \(p. 183\)](#)
- [Criptografia em trânsito \(p. 184\)](#)
- [Gerenciamento de chaves \(p. 184\)](#)

## Criptografia em repouso

O Amazon Lex criptografa os enunciados de usuário que armazena.

#### Tópicos

- [Enunciados de amostra \(p. 183\)](#)
- [Enunciados do cliente \(p. 183\)](#)
- [Atributos da sessão \(p. 183\)](#)
- [Atributos de solicitação \(p. 184\)](#)

## Enunciados de amostra

Ao desenvolver um bot, você pode fornecer enunciados de amostra para cada intenção e slot. Você também pode fornecer sinônimos e valores personalizados para slots. Essas informações são usadas apenas para criar o bot e criar a experiência do usuário. Elas não são criptografadas.

## Enunciados do cliente

O Amazon Lex criptografa enunciados que os usuários enviam para seu bot, a menos que o campo `childDirected` seja definido como `true`.

Quando o campo `childDirected` é definido como `true`, nenhum enunciado de usuário está armazenado.

Quando o campo `childDirected` é definido como `false` (o padrão), enunciados do usuário são criptografados e armazenados por 15 dias para uso com a operação [GetUtterancesView \(p. 308\)](#). Para excluir enunciados armazenados para um usuário específico, use a operação [DeleteUtterances \(p. 250\)](#).

Quando seu bot aceita a entrada de voz, ela é armazenada indefinidamente. O Amazon Lex a usa para melhorar a capacidade do seu bot para responder à entrada do usuário.

Use a operação [DeleteUtterances \(p. 250\)](#) para excluir enunciados armazenados por um usuário específico.

## Atributos da sessão

Os atributos de sessão contêm informações específicas do aplicativo que são passadas entre o Amazon Lex e aplicativos cliente. O Amazon Lex passa atributos da sessão para todas as funções AWS Lambda configuradas para um bot. Se uma função do Lambda adicionar ou atualizar atributos da sessão, o Amazon Lex passará as novas informações de volta para o aplicativo cliente.

Os atributos da sessão permanecem em um armazenamento criptografado durante a sessão. Você pode configurar a sessão para permanecer ativa por um mínimo de 1 minuto e até 24 horas após o último enunciado do usuário. O padrão, a duração da sessão é de 5 minutos.

## Atributos de solicitação

Atributos de solicitação contêm informações específicas da solicitação e aplicam-se apenas à solicitação atual. Um aplicativo cliente usa atributos de solicitação para enviar informações para o Amazon Lex em tempo de execução.

Você usa atributos de solicitação para passar informações que não precisam ser mantidas durante toda a sessão. Como os atributos de solicitação não são mantidos entre solicitações, eles não são armazenados.

## Criptografia em trânsito

O Amazon Lex usa o protocolo HTTPS para se comunicar com o aplicativo cliente. Ele usa HTTPS e assinaturas da AWS para se comunicar com outros serviços, como o Amazon Polly e o AWS Lambda em nome do aplicativo.

## Gerenciamento de chaves

O Amazon Lex protege o conteúdo de uso não autorizado com chaves internas.

# Identity and Access Management para Amazon Lex

O AWS Identity and Access Management (IAM) é um serviço da AWS que ajuda um administrador a controlar com segurança o acesso aos recursos da AWS. Os administradores do IAM controlam quem pode ser autenticado (conectado) e autorizado (ter permissões) para usar os recursos do Amazon Lex. O IAM é um serviço da AWS que pode ser usado sem custo adicional.

### Tópicos

- [Público \(p. 184\)](#)
- [Autenticação com identidades \(p. 185\)](#)
- [Gerenciamento do acesso usando políticas \(p. 187\)](#)
- [Saiba mais \(p. 188\)](#)
- [Como o Amazon Lex funciona com o IAM \(p. 188\)](#)
- [Exemplos de políticas baseadas em identidade do Amazon Lex \(p. 191\)](#)
- [Exemplo de política baseada em recursos do Amazon Lex \(p. 196\)](#)
- [Solução de problemas de identidade e acesso do Amazon Lex \(p. 197\)](#)

## Público

O uso do AWS Identity and Access Management (IAM) varia, dependendo do trabalho que você realiza no Amazon Lex.

**Usuário do serviço** – se você usar o Amazon Lex para fazer sua tarefa, o administrador fornecerá as credenciais e as permissões de que você precisa. À medida que usar mais recursos do Amazon Lex para fazer seu trabalho, você poderá precisar de permissões adicionais. Entender como o acesso é gerenciado pode ajudar você a solicitar as permissões corretas ao seu administrador. Se não for possível acessar um recurso no Amazon Lex, consulte [Solução de problemas de identidade e acesso do Amazon Lex \(p. 197\)](#).

**Administrador do serviço** – se você for o responsável pelos recursos do Amazon Lex em sua empresa, você provavelmente terá acesso total ao Amazon Lex. Seu trabalho é determinar quais recursos do Amazon Lex seus funcionários devem acessar. Assim, é necessário enviar solicitações ao administrador

do IAM para alterar as permissões dos usuários de seu serviço. Revise as informações nesta página para entender os conceitos básicos do IAM. Para saber mais sobre como sua empresa pode usar o IAM com o Amazon Lex, consulte [Como o Amazon Lex funciona com o IAM](#) (p. 188).

Administrador do IAM – se você é um administrador do IAM, talvez queira saber detalhes sobre como pode escrever políticas para gerenciar o acesso ao Amazon Lex. Para visualizar exemplos de políticas baseadas em identidade do Amazon Lex que podem ser usadas no IAM, consulte [Exemplos de políticas baseadas em identidade do Amazon Lex](#) (p. 191).

## Autenticação com identidades

A autenticação é a forma como você faz login na AWS usando suas credenciais de identidade. Para obter mais informações sobre como fazer login usando o Console de gerenciamento da AWS, consulte [A página de login e do console do IAM](#) no Guia do usuário do IAM.

Você deve ser autenticado (fazer login na AWS) como o Usuário raiz da conta da AWS, um usuário do IAM, ou assumindo uma função do IAM. Também é possível usar a autenticação de logon único da sua empresa, ou até mesmo fazer login usando o Google ou o Facebook. Nesses casos, seu administrador configurou anteriormente a federação de identidades usando funções do IAM. Ao acessar a AWS usando credenciais de outra empresa, você estará assumindo uma função indiretamente.

Para fazer login diretamente no [Console de gerenciamento da AWS](#), use sua senha com o e-mail do usuário raiz ou seu nome de usuário do IAM. É possível acessar a AWS de maneira programática usando seu usuário raiz ou as chaves de acesso do usuário do IAM. A AWS fornece ferramentas do SDK ou da linha de comando para assinar sua solicitação de forma criptográfica usando suas credenciais. Se você não utilizar as ferramentas da AWS, você deverá assinar a solicitação por conta própria. Faça isso usando o Signature versão 4, um protocolo para autenticação de solicitações de API de entrada. Para obter mais informações sobre a autenticação de solicitações, consulte [Processo de cadastramento do Signature versão 4](#) na AWS General Reference.

Independentemente do método de autenticação usado, também pode ser exigido que você forneça informações adicionais de segurança. Por exemplo, a AWS recomenda o uso da autenticação multifator (MFA) para aumentar a segurança de sua conta. Para saber mais, consulte [Usar a autenticação multifator \(MFA\) na AWS](#) no Guia do usuário do IAM.

## Usuário raiz da conta da AWS

Ao criar uma conta da AWS, você começa com uma única identidade de login que tenha acesso total a todos os recursos e serviços da AWS na conta. Essa identidade é chamada de AWS da conta da usuário raiz e é acessada pelo login com o endereço de e-mail e a senha que você usou para criar a conta. Recomendamos que não use o usuário raiz para suas tarefas diárias, nem mesmo as administrativas. Em vez disso, siga as [melhores práticas de uso do usuário raiz somente para criar seu primeiro usuário do IAM](#). Depois, armazene as credenciais usuário raiz com segurança e use-as para executar apenas algumas tarefas de gerenciamento de contas e de serviços.

## Grupos e usuários do IAM

Um [usuário do IAM](#) é uma identidade em sua conta da AWS que tem permissões específicas para uma única pessoa ou um único aplicativo. Um usuário do IAM pode ter credenciais de longo prazo, como um nome de usuário e uma senha ou um conjunto de chaves de acesso. Para saber como gerar chaves de acesso, consulte [Gerenciar chaves de acesso para usuários do IAM](#) no Guia do usuário do IAM. Ao gerar chaves de acesso para um usuário do IAM, visualize e salve o par de chaves de maneira segura. Não será possível recuperar a chave de acesso secreta futuramente. Em vez disso, você deverá gerar outro par de chaves de acesso.

Um [grupo do IAM](#) é uma identidade que especifica uma coleção de usuários do IAM. Não é possível fazer login como um grupo. É possível usar grupos para especificar permissões para vários usuários de uma vez. Os grupos facilitam o gerenciamento de permissões para grandes conjuntos de usuários. Por

exemplo, você pode ter um grupo chamado Administradores do IAM e atribuir a esse grupo permissões para administrar recursos do IAM.

Usuários são diferentes de funções. Um usuário é exclusivamente associado a uma pessoa ou a um aplicativo, mas uma função pode ser assumida por qualquer pessoa que precisar dela. Os usuários têm credenciais permanentes de longo prazo, mas as funções fornecem credenciais temporárias. Para saber mais, consulte [Quando criar um usuário do IAM \(em vez de uma função\)](#) no Guia do usuário do IAM.

## Funções do IAM

Uma [função do IAM](#) é uma identidade dentro de sua conta da AWS que tem permissões específicas. Ela é semelhante a um usuário do IAM, mas não está associada a uma pessoa específica. É possível assumir temporariamente uma função do IAM no Console de gerenciamento da AWS [alternando funções](#). É possível assumir uma função chamando uma operação de API da AWS CLI ou da AWS, ou usando um URL personalizado. Para obter mais informações sobre os métodos para o uso de funções, consulte [Usar funções do IAM](#) no Guia do usuário do IAM.

As funções do IAM com credenciais temporária são úteis nas seguintes situações:

- Permissões temporárias para usuários do IAM – um usuário do IAM pode assumir uma função do IAM para obter temporariamente permissões diferentes para uma tarefa específica.
- Acesso de usuário federado – Em vez de criar um usuário do IAM, você pode usar identidades existentes do AWS Directory Service, do diretório de usuário da sua empresa ou de um provedor de identidades da Web. Estes são conhecidos como usuários federados. A AWS atribui uma função a um usuário federado quando o acesso é solicitado por meio de um [provedor de identidades](#). Para obter mais informações sobre usuários federados, consulte [Usuários federados e funções](#) no Guia do usuário do IAM.
- Acesso entre contas – é possível usar uma função do IAM para permitir que alguém (um principal confiável) em outra conta acesse recursos em sua conta. As funções são a principal forma de conceder acesso entre contas. No entanto, alguns serviços da AWS permitem que você anexe uma política diretamente a um recurso (em vez de usar uma função como proxy). Para saber a diferença entre funções e políticas baseadas em recurso para acesso entre contas, consulte [Como as funções do IAM diferem das políticas baseadas em recurso](#) no Guia do usuário do IAM.
- Acesso a serviços da AWS – Uma função de serviço é uma função do IAM que um serviço assume para realizar ações em seu nome na sua conta. Ao configurar alguns ambientes de serviço da AWS, você deve definir uma função a ser assumida pelo serviço. Essa função de serviço deve incluir todas as permissões necessárias para o serviço acessar os recursos da AWS de que precisa. As funções de serviço variam de acordo com o serviço, mas muitas permitem que você escolha as permissões, desde que atenda aos requisitos documentados para esse serviço. As funções de serviço fornecem acesso apenas dentro de sua conta e não podem ser usadas para conceder acesso a serviços em outras contas. Você pode criar, modificar e excluir uma função de serviço no IAM. Por exemplo, você pode criar uma função que permita que Amazon Redshift acesse um bucket do Amazon S3 em seu nome e carregue dados desse bucket em um cluster Amazon Redshift. Para obter mais informações, consulte [Criar uma função para delegar permissões a um serviço da AWS](#) no Guia do usuário do IAM.
- Aplicativos em execução no Amazon EC2 –Você pode usar uma função do IAM para gerenciar credenciais temporárias para aplicativos que estão sendo executados em uma instância do EC2 e que fazem solicitações de API da AWS CLI ou AWS. É preferível fazer isso do que armazenar chaves de acesso na instância do EC2. Para atribuir uma função da AWS a uma instância do EC2 e disponibilizá-la para todos os seus aplicativos, crie um perfil de instância que esteja anexado à instância. Um perfil de instância contém a função e permite que programas que estão em execução na instância do EC2 obtenham credenciais temporárias. Para mais informações, consulte [Uso de uma função do IAM para conceder permissões aos aplicativos em execução nas instâncias do Amazon EC2](#) no Guia do usuário do IAM.

Para saber se você deve usar funções do IAM, consulte [Quando criar uma função do IAM \(em vez de um usuário\)](#) no Guia do usuário do IAM.



## Gerenciamento do acesso usando políticas

Você controla o acesso na AWS criando políticas e anexando-as às identidades do IAM ou aos recursos da AWS. Uma política é um objeto na AWS que, quando associado a uma identidade ou a um recurso, define suas permissões. A AWS avalia essas políticas quando uma entidade (usuário raiz, usuário do IAM ou função do IAM) faz uma solicitação. As permissões nas políticas determinam se a solicitação será permitida ou negada. A maioria das políticas são armazenadas na AWS como documentos JSON. Para obter mais informações sobre a estrutura e o conteúdo de documentos de políticas JSON, consulte [Visão geral de políticas JSON](#) no Guia do usuário do IAM.

Um administrador do IAM pode usar políticas para especificar quem tem acesso aos recursos da AWS e quais ações essas pessoas podem executar nesses recursos. Cada entidade do IAM (usuário ou função) começa sem permissões. Em outras palavras, por padrão, os usuários não podem fazer nada, nem mesmo alterar sua própria senha. Para dar permissão a um usuário para fazer algo, um administrador deve anexar uma política de permissões ao usuário. Ou o administrador pode adicionar o usuário a um grupo que tenha as permissões pretendidas. Quando um administrador concede permissões a um grupo, todos os usuários desse grupo recebem essas permissões.

As políticas do IAM definem permissões para uma ação, independentemente do método usado para executar a operação. Por exemplo, suponha que você tenha uma política que permite a ação `iam:GetRole`. Um usuário com essa política pode obter informações de funções do Console de gerenciamento da AWS, da AWS CLI ou da API da AWS.

### Políticas baseadas em identidade

As políticas baseadas em identidade são documentos JSON de políticas de permissões que você pode anexar a uma entidade, como um usuário, função ou grupo do IAM. Essas políticas controlam quais ações cada identidade pode realizar, em quais recursos e em que condições. Para saber como criar uma política baseada em identidade, consulte [Criar políticas do IAM](#) no Guia do usuário do IAM.

As políticas baseadas em identidade podem ser categorizadas ainda mais como políticas em linha ou políticas gerenciadas. As políticas em linha são anexadas diretamente a um único usuário, grupo ou função. As políticas gerenciadas são políticas independentes que podem ser anexadas a vários usuários, grupos e funções em sua conta da AWS. As políticas gerenciadas incluem políticas gerenciadas pela AWS e políticas gerenciadas pelo cliente. Para saber como escolher entre uma política gerenciada ou uma política em linha, consulte [Escolher entre políticas gerenciadas e políticas em linha](#) no Guia do usuário do IAM.

### Políticas com base em recurso

Políticas baseadas em recursos são documentos de política JSON que você anexa a um recurso, como um bot do Amazon Lex. Os administradores do serviço podem usar essas políticas para definir quais ações um principal especificado (função, usuário ou membro da conta) pode executar nesse recurso e sob quais condições. As políticas baseadas em recurso são políticas em linha. Não há políticas baseadas em recurso gerenciadas.

### Outros tipos de política

A AWS oferece suporte a tipos de política menos comuns. Esses tipos de política podem definir o máximo de permissões concedidas a você pelos tipos de política mais comuns.

- **Limites de permissões** – um limite de permissões é um recurso avançado no qual você define o máximo de permissões que uma política baseada em identidade pode conceder a uma entidade do IAM (usuário ou função do IAM). É possível definir um limite de permissões para uma entidade. As permissões resultantes são a interseção das políticas baseadas em identidade da entidade e seus limites de permissões. As políticas baseadas em recurso que especificam o usuário ou a função no campo `Principal` não são limitadas pelo limite de permissões. Uma negação explícita em qualquer



uma dessas políticas substitui a permissão. Para obter mais informações sobre limites de permissões, consulte [Limites de permissões para entidades do IAM](#) no Guia do usuário do IAM.

- Políticas de controle de serviço (SCPs – Service control policies) – SCPs são políticas JSON que especificam o máximo de permissões para uma organização ou unidade organizacional (UO) no AWS Organizations. O AWS Organizations é um serviço para agrupamento e gerenciamento central das várias contas da AWS que sua empresa possui. Se você habilitar todos os recursos em uma organização, poderá aplicar políticas de controle de serviço (SCPs) a qualquer uma ou a todas as contas. O SCP limita as permissões para entidades em contas-membro, incluindo cada Usuário raiz da conta da AWS. Para obter mais informações sobre Organizações e SCPs, consulte [Como SCPs funcionam](#) no Guia do usuário do AWS Organizations.
- Políticas de sessão – as políticas de sessão são políticas avançadas que você transmite como um parâmetro quando cria de forma programática uma sessão temporária para uma função ou um usuário federado. As permissões da sessão resultante são a interseção das políticas baseadas em identidade do usuário ou da função e das políticas de sessão. As permissões também podem ser provenientes de uma política baseada em recurso. Uma negação explícita em qualquer uma dessas políticas substitui a permissão. Para obter mais informações, consulte [Políticas de sessão](#) no Guia do usuário do IAM.

## Vários tipos de política

Quando vários tipos de política são aplicáveis a uma solicitação, é mais complicado compreender as permissões resultantes. Para saber como a AWS determina se deve permitir uma solicitação quando vários tipos de política estão envolvidos, consulte [Lógica de avaliação de políticas](#) no Guia do usuário do IAM.

## Saiba mais

Para obter mais informações sobre o Identity and Access Management para o Amazon Lex, consulte as seguintes páginas:

- [Como o Amazon Lex funciona com o IAM](#) (p. 188)
- [Solução de problemas de identidade e acesso do Amazon Lex](#) (p. 197)

## Como o Amazon Lex funciona com o IAM

Antes de usar o AWS Identity and Access Management (IAM) para gerenciar o acesso ao Amazon Lex, você deve entender quais recursos do IAM estão disponíveis para uso com o Amazon Lex. Para obter uma visão de alto nível de como o Amazon Lex e outros serviços da AWS funcionam com o IAM, consulte [Serviços da AWS compatíveis com o IAM](#) no Guia do usuário do IAM.

### Tópicos

- [Políticas baseadas em identidade do Amazon Lex](#) (p. 188)
- [Políticas baseadas em recursos do Amazon Lex](#) (p. 190)
- [Autorização baseada em tags do Amazon Lex](#) (p. 190)
- [Funções do IAM do Amazon Lex](#) (p. 191)

## Políticas baseadas em identidade do Amazon Lex

Para especificar ações permitidas ou negadas e recursos, bem como as condições nas quais as ações são permitidas ou negadas, use as políticas baseadas em identidade do IAM. O Amazon Lex oferece suporte a ações, recursos e chaves de condição específicos. Para saber mais sobre todos os elementos que você usa em uma política JSON, consulte [Referência de elementos de política JSON do IAM](#) no Guia do usuário do IAM.

## Ações

O elemento `Action` de uma política baseada em identidade do IAM descreve a ação ou ações específicas que serão permitidas ou negadas pela política. As ações de política geralmente têm o mesmo nome que a operação de API da AWS associada. A ação é usada em uma política para conceder permissões para executar a operação associada.

No Amazon Lex, as ações de políticas usam o seguinte prefixo antes da ação: `lex:`. Por exemplo, para conceder a alguém permissão para chamar um bot do Amazon Lex com a operação `PostContent`, inclua a ação `lex:PostContent` na política da pessoa. As declarações de política devem incluir um elemento `Action` ou `NotAction`. O Amazon Lex define ações que descrevem as tarefas que podem ser executadas com esse serviço. Para obter uma lista de ações do Amazon Lex, consulte [Actions Defined by Amazon Lex](#) no Guia do usuário do IAM.

Para especificar várias ações em uma única declaração, separe-as com vírgulas, conforme o seguinte.

```
"Action": [
    "lex:action1",
    "lex:action2"
```

Você também pode especificar várias ações usando caracteres curinga (\*). Por exemplo, para especificar todas as ações que começam com a palavra `Put`, inclua a seguinte ação.

```
"Action": "lex:Put*"
```

## Recursos

O elemento `Resource` especifica o objeto ou os objetos aos quais a ação se aplica. As instruções devem incluir um elemento `Resource` ou um elemento `NotResource`. Você especifica um recurso usando um ARN ou, para indicar que a instrução se aplica a todos os recursos, usando o caractere curinga (\*).

O ARN de um recurso de bot do Amazon Lex tem o seguinte formato.

```
arn:aws:lex:${Region}:${Account}:bot/${Bot-Name}
```

Para obter mais informações sobre o formato de ARNs, consulte [Nomes de recursos da Amazon \(ARNs\) e namespaces de serviços da AWS](#).

Por exemplo, para especificar o bot `OrderFlowers` em sua instrução, use o seguinte ARN.

```
"Resource": "arn:aws:lex:us-east-2:123456789012:bot/OrderFlowers"
```

Para especificar todos os bots que pertencem a uma conta específica, use o caractere curinga (\*):

```
"Resource": "arn:aws:lex:us-east-2:123456789012:bot/*"
```

Algumas ações do Amazon Lex, como as ações para a criação de recursos, não podem ser executadas em um recurso específico. Nesses casos, é necessário usar o caractere curinga (\*).

```
"Resource": ""
```

Para ver uma lista de tipos de recurso do Amazon Lex e seus ARNs, consulte [Resources Defined by Amazon Lex](#) no Guia do usuário do IAM. Para saber com quais ações você pode especificar o ARN de cada recurso, consulte [Actions Defined by Amazon Lex](#).

## Chaves de condição

Use o elemento `Condition` (ou um bloco `Condition` para especificar condições nas quais uma instrução está em vigor. O elemento `Condition` é opcional. É possível criar expressões condicionais que usam [operadores de condição](#), como "igual a" ou "menor que", para fazer a condição da política corresponder aos valores na solicitação.

Se você especificar vários elementos `Condition` em uma instrução ou várias chaves em um único elemento `Condition`, a AWS os avaliará usando uma operação lógica AND. Se você especificar vários valores para uma única chave de condição, a AWS avaliará a condição usando uma operação lógica OR. Todas as condições devem ser atendidas para que as permissões da instrução sejam concedidas.

Você também pode usar variáveis de espaço reservado ao especificar as condições. Por exemplo, você pode conceder a um usuário do IAM permissão para acessar um recurso somente se ele estiver marcado com seu nome de usuário do IAM. Para obter mais informações, consulte [Elementos de política do IAM: variáveis e tags](#) no Guia do usuário do IAM.

O Amazon Lex define seu próprio conjunto de chaves de condição e também oferece suporte ao uso de algumas chaves de condição globais. Para obter uma lista de todas as chaves de condição globais da AWS, consulte [Chaves de contexto de condição globais da AWS](#) no Guia do usuário do IAM.

A tabela a seguir mostra as chaves de condição do Amazon Lex aplicáveis aos recursos do Amazon Lex. Você pode incluir essas chaves em elementos `Condition` de uma política de permissões do IAM.

## Exemplos

Para obter exemplos de políticas baseadas em identidade do Amazon Lex, consulte [Exemplos de políticas baseadas em identidade do Amazon Lex](#) (p. 191).

## Políticas baseadas em recursos do Amazon Lex

As políticas baseadas em recursos são documentos de política JSON que especificam quais ações uma entidade principal pode realizar no recurso Amazon Lex e em que condições. O Amazon Lex oferece suporte a políticas de permissões com base em recursos para bots, intenções e tipos de slot. Use as políticas baseadas em recursos para conceder permissão de uso a outras contas por recurso. Você também pode usar uma política baseada em recursos para permitir que um serviço da AWS acesse seus recursos do Amazon Lex.

Para permitir o acesso entre contas, você pode especificar uma conta inteira ou especificar entidades do IAM em outra conta como o [principal em uma política baseada em recurso](#). Adicionar um principal entre contas à política baseada em recurso é apenas o primeiro passo para estabelecer a relação de confiança. Quando o principal e o recurso estão em diferentes contas da AWS, você também deve conceder à entidade principal permissão para acessar o recurso. Conceda permissão anexando uma política baseada em identidade para a entidade. No entanto, se uma política baseada em recurso conceder acesso a um principal na mesma conta, nenhuma política baseada em identidade adicional será necessária. Para obter mais informações, consulte [Como as funções do IAM diferem de políticas baseadas em recursos](#) no Guia do usuário do IAM.

## Exemplos

Para obter exemplos de políticas baseadas em recursos do Amazon Lex, consulte [Exemplo de política baseada em recursos do Amazon Lex](#) (p. 196),

## Autorização baseada em tags do Amazon Lex

O Amazon Lex não é compatível com recursos de marcação ou de controle de acesso com base em tags.

## Funções do IAM do Amazon Lex

Uma [função do IAM](#) é uma entidade dentro de sua conta da AWS que tem permissões específicas.

### Usar credenciais temporárias com o Amazon Lex

Você pode usar credenciais temporárias para fazer login com federação, assumir uma função do IAM ou assumir uma função entre contas. As credenciais de segurança temporárias são obtidas chamando operações da API do AWS STS, como [AssumeRole](#) ou [GetFederationToken](#).

O Amazon Lex oferece suporte ao uso de credenciais temporárias.

### Funções vinculadas ao serviço

[Funções vinculadas ao serviço](#) permitem que os serviços da AWS acessem recursos em outros serviços para concluir uma ação em seu nome. As funções vinculadas ao serviço aparecem em sua conta do IAM e são de propriedade do serviço. Um administrador do IAM pode visualizar, mas não pode editar as permissões para funções vinculadas ao serviço.

O Amazon Lex oferece suporte a funções vinculadas ao serviço. Para obter detalhes sobre como criar ou gerenciar funções vinculadas ao serviço do Amazon Lex, consulte [Etapa 1: Criar uma função vinculada a serviço \(AWS CLI\)](#) (p. 77).

### Funções de serviço

Um serviço pode assumir uma [função de serviço](#) em seu nome. A função permite que o serviço acesse recursos em outros serviços para concluir uma ação em seu nome. As funções de serviço aparecem em sua conta do IAM e são de propriedade da conta. Isso significa que um administrador do IAM pode alterar as permissões para essa função. No entanto, isso pode impedir que o serviço funcione conforme o esperado.

O Amazon Lex oferece suporte às funções de serviço.

### Escolha de uma função do IAM no Amazon Lex

Ao criar um recurso de bot no Amazon Lex, é necessário escolher uma função para permitir que o Amazon Lex acesse o AWS Lambda, o AWS Key Management Service, o Amazon CloudWatch e o Amazon Polly em seu nome. Caso já tenha criado uma função de serviço ou função vinculada ao serviço, o console do Amazon Lex fornecerá uma lista das funções para sua escolha. Para obter mais informações, consulte [Permissões de serviço](#) (p. 9).

## Exemplos de políticas baseadas em identidade do Amazon Lex

Por padrão, os usuários e funções do IAM não têm permissão para criar ou modificar recursos do Amazon Lex. Eles também não podem executar tarefas usando o Console de gerenciamento da AWS, a AWS CLI ou uma API da AWS. Um administrador do IAM deve criar políticas do IAM que concedam aos usuários e funções permissão para executarem operações da API específicas nos recursos especificados de que precisam. O administrador deve anexar essas políticas aos usuários ou grupos do IAM que exigem essas permissões.

Para saber como criar uma política baseada em identidade do IAM usando esses exemplos de documentos de política JSON, consulte [Criar políticas no guia JSON](#) no Guia do usuário do IAM.

#### Tópicos

- [Melhores práticas de políticas](#) (p. 192)
- [Usar o console do Amazon Lex](#) (p. 192)
- [Políticas gerenciadas \(predefinidas\) pela AWS para o Amazon Lex](#) (p. 194)

- [Exemplo: permitir que os usuários visualizem suas próprias permissões \(p. 195\)](#)
- [Exemplo: excluir todos os bots do Amazon Lex \(p. 196\)](#)

## Melhores práticas de políticas

As políticas baseadas em identidade são muito eficientes. Elas determinam se alguém pode criar, acessar ou excluir recursos do Amazon Lex em sua conta. Essas ações podem incorrer em custos para sua conta da AWS. Ao criar ou editar políticas baseadas em identidade, siga estas diretrizes e recomendações:

- Comece usando políticas gerenciadas pela AWS – para começar a usar o Amazon Lex rapidamente, use as políticas gerenciadas pela AWS para conceder a seus funcionários as permissões de que precisam. Essas políticas já estão disponíveis em sua conta e são mantidas e atualizadas pela AWS. Para obter mais informações, consulte [Conceitos básicos do uso de permissões com políticas gerenciadas pela AWS](#) no Guia do usuário do IAM.
- Conceder privilégio mínimo – ao criar políticas personalizadas, conceda apenas as permissões necessárias para executar uma tarefa. Comece com um conjunto mínimo de permissões e conceda permissões adicionais conforme necessário. Fazer isso é mais seguro do que começar com permissões que são muito lenientes e tentar restringi-las posteriormente. Para obter mais informações, consulte [Conceder privilégio mínimo](#), no Guia do usuário do IAM.
- Habilitar o MFA para operações confidenciais – para segurança adicional, exija que os usuários do IAM usem a autenticação multifator (MFA) para acessar recursos ou operações de API confidenciais. Para obter mais informações, consulte [Como usar a autenticação multifator \(MFA\) na AWS](#) no Guia do usuário do IAM.
- Usar condições de política para segurança adicional – na medida do possível, defina as condições sob as quais suas políticas baseadas em identidade permitem o acesso a um recurso. Por exemplo, você pode gravar condições para especificar um intervalo de endereços IP permitidos do qual a solicitação deve partir. Você também pode escrever condições para permitir somente solicitações em uma data especificada ou período ou para exigir o uso de SSL ou MFA. Para obter mais informações, consulte [Elementos da política JSON do IAM](#) no Guia do usuário do IAM.

## Usar o console do Amazon Lex

Para acessar o console do Amazon Lex, você deve ter um conjunto mínimo de permissões. Essas permissões devem permitir que você liste e visualize detalhes sobre os recursos do Amazon Lex em sua conta da AWS. Se você criar uma política baseada em identidade que seja mais restritiva que as permissões mínimas necessárias, o console não funcionará como pretendido para entidades (usuários ou funções do IAM) com essa política.

Veja a seguir as permissões mínimas necessárias para usar o console do Amazon Lex.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudwatch:GetMetricStatistics",
        "cloudwatch:DescribeAlarms",
        "cloudwatch:DescribeAlarmsForMetric",
        "kms:DescribeKey",
        "kms:ListAliases",
        "lambda:GetPolicy",
        "lambda:ListFunctions",
        "lex:*",
        "polly:DescribeVoices",
        "polly:SynthesizeSpeech"
      ]
    }
  ]
}
```

```
    ],
    "Resource": [
        "*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "lambda:AddPermission",
        "lambda:RemovePermission"
    ],
    "Resource": "*",
    "Condition": {
        "StringLike": {
            "lambda:Principal": "lex.amazonaws.com"
        }
    }
},
{
    "Effect": "Allow",
    "Action": [
        "iam:GetRole",
        "iam:DeleteRole"
    ],
    "Resource": [
        "arn:aws:iam::*:role/aws-service-role/lex.amazonaws.com/
AWSServiceRoleForLexBots",
        "arn:aws:iam::*:role/aws-service-role/channels.lex.amazonaws.com/
AWSServiceRoleForLexChannels"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "iam:CreateServiceLinkedRole"
    ],
    "Resource": [
        "arn:aws:iam::*:role/aws-service-role/lex.amazonaws.com/
AWSServiceRoleForLexBots"
    ],
    "Condition": {
        "StringLike": {
            "iam:AWSServiceName": "lex.amazonaws.com"
        }
    }
},
{
    "Effect": "Allow",
    "Action": [
        "iam:DetachRolePolicy"
    ],
    "Resource": [
        "arn:aws:iam::*:role/aws-service-role/lex.amazonaws.com/
AWSServiceRoleForLexBots"
    ],
    "Condition": {
        "StringLike": {
            "iam:PolicyArn": "arn:aws:iam::aws:policy/aws-service-role/
AmazonLexBotPolicy"
        }
    }
},
{
    "Effect": "Allow",
    "Action": [
        "iam:CreateServiceLinkedRole"
    ]
}
```

```
    ],
    "Resource": [
        "arn:aws:iam::*:role/aws-service-role/channels.lex.amazonaws.com/
AWSServiceRoleForLexChannels"
    ],
    "Condition": {
        "StringLike": {
            "iam:AWSServiceName": "channels.lex.amazonaws.com"
        }
    }
},
{
    "Effect": "Allow",
    "Action": [
        "iam:DetachRolePolicy"
    ],
    "Resource": [
        "arn:aws:iam::*:role/aws-service-role/channels.lex.amazonaws.com/
AWSServiceRoleForLexChannels"
    ],
    "Condition": {
        "StringLike": {
            "iam:PolicyArn": "arn:aws:iam::aws:policy/aws-service-role/
LexChannelPolicy"
        }
    }
}
]
```

O console do Amazon Lex precisa dessas permissões adicionais pelas seguintes razões:

- As permissões `cloudwatch` permitem visualizar o desempenho e informações de monitoramento no console.
- Ações `iam` permitem que o Amazon Lex assuma funções do IAM para fazer chamadas para funções Lambda e processar dados para uma associação de canal de bot.
- Ações `kms` permitem que você gerencie as chaves do AWS Key Management Service usadas para criptografar dados ao criar uma associação de canal de bot.
- Ações `lambda` permitem que você visualize funções Lambda que seu bot pode usar e conceda ao Amazon Lex as permissões necessárias para seu bot invocar essas funções.
- Ações `lex` permitem que o console mostre os recursos do Amazon Lex na conta.
- Ações `polly` permitem que o console mostre as vozes do Amazon Polly disponíveis e converter texto em fala.
- Ações `iam` permitem que você use o console para gerenciar funções vinculadas ao servidor que concedem permissão para usar outros recursos da AWS.

Não é necessário conceder permissões mínimas do console para usuários que fazem chamadas somente à AWS CLI ou à API da AWS. Em vez disso, permita o acesso somente às ações que correspondem à operação da API que você está tentando executar.

Para obter mais informações, consulte [Adicionar permissões a um usuário](#) no Guia do usuário do IAM:

## Políticas gerenciadas (predefinidas) pela AWS para o Amazon Lex

A AWS resolve muitos casos de uso comuns, fornecendo políticas do IAM autônomas criadas e administradas pela AWS. Essas políticas são chamadas de políticas gerenciadas pela AWS. As políticas

gerenciadas pela AWS facilitam a atribuição de permissões apropriadas a usuários, grupos e funções em comparação com a elaboração de suas próprias políticas. Para obter mais informações [Políticas gerenciadas pela AWS](#) no Guia do usuário do IAM.

As seguintes políticas gerenciadas pela AWS, que você pode associar a grupos e funções na sua conta, são específicas do Amazon Lex:

- **ReadOnly** — concede acesso somente leitura a recursos do Amazon Lex.
- **RunBotsOnly** — concede acesso para executar bots de conversa do Amazon Lex.
- **FullAccess** — concede acesso total para criar, ler, atualizar, excluir e executar todos os recursos do Amazon Lex. Também concede a capacidade de associar funções Lambda com nomes que começam com `AmazonLex` com intenções do Amazon Lex.

#### Note

Você pode analisar essas políticas de permissões fazendo login no console do IAM e pesquisando políticas específicas.

Além disso, você pode criar políticas personalizadas do IAM para conceder permissões a ações da API do Amazon Lex. Você pode anexar essas políticas personalizadas a funções ou grupos do IAM que exijam essas permissões.

## Exemplo: permitir que os usuários visualizem suas próprias permissões

Este exemplo de política permite que os usuários do IAM visualizem as políticas gerenciadas e em linha anexadas à identidade de usuário. Essa política inclui permissões para concluir essa ação no console ou de forma programática usando a AWS CLI ou a API da AWS.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ViewOwnUserInfo",
      "Effect": "Allow",
      "Action": [
        "iam:GetUserPolicy",
        "iam:ListGroupForUser",
        "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": "*"
    }
  ]
}
```



```
}  
]  
}
```

## Exemplo: excluir todos os bots do Amazon Lex

Este exemplo de política concede a um usuário do IAM em sua conta da AWS permissão para excluir qualquer bot em sua conta.

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": [  
        "lex:DeleteBot"  
      ],  
      "Resource": [  
        "*" ]  
    }  
  ]  
}
```

## Exemplo de política baseada em recursos do Amazon Lex

As políticas baseadas em recursos são documentos de políticas JSON que especificam quais ações uma entidade principal pode executar no recurso do Amazon Lex e sob quais condições. Para saber como criar um bot, consulte [Conceitos básicos do Amazon Lex \(p. 35\)](#).

### Permitir que um usuário gerencie um bot específico

A política de permissões a seguir concede ao usuário permissões para criar e testar um bot de pedido de pizza. Isso permite que o usuário use apenas a intenção `OrderPizza` e o tipo de slot `Toppings` ao editar o bot na região `us-east-2`.

O bloco `Condition` usa a condição `ForAllValues:StringEqualsIfExists` e as chaves de condição do Amazon Lex `lex:associatedIntents` e `lex:associatedSlotType` para limitar a intenção e os tipos de slot que o usuário pode usar para esse bot.

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": [  
        "lex:Create*Version",  
        "lex:Post*",  
        "lex:Put*",  
        "lex>Delete*" ]  
      ],  
      "Resource": [  
        "arn:aws:lex:us-east-2::bot:PizzaBot:*",  
        "arn:aws:lex:us-east-2::intent:OrderPizza:*",  
        "arn:aws:lex:us-east-2::slottype:Toppings:*" ]  
      ],  
      "Condition": {  
        "ForAllValues:StringEqualsIfExists": {  
          "lex:associatedIntents": "OrderPizza",  
          "lex:associatedSlotType": "Toppings"  
        }  
      }  
    }  
  ]  
}
```

```
        "lex:associatedIntents": [
            "OrderPizza"
        ],
        "lex:associatedSlotTypes": [
            "Toppings"
        ]
    }
},
{
    "Effect": "Allow",
    "Action": [
        "lex:Get*"
    ],
    "Resource": [
        "arn:aws:lex:us-east-2::bot:*",
        "arn:aws:lex:us-east-2::intent:*",
        "arn:aws:lex:us-east-2::slottype:*"
    ]
}
]
```

## Solução de problemas de identidade e acesso do Amazon Lex

Use as seguintes informações para ajudar a diagnosticar e corrigir problemas comuns que podem ser encontrados ao trabalhar com o Amazon Lex e o AWS Identity and Access Management (IAM).

### Tópicos

- [Não tenho autorização para executar uma ação no Amazon Lex \(p. 197\)](#)
- [Não estou autorizado a executar iam:PassRole \(p. 197\)](#)
- [Quero visualizar minhas chaves de acesso \(p. 198\)](#)
- [Sou administrador e desejo conceder acesso ao Amazon Lex para outros usuários. \(p. 198\)](#)
- [Quero permitir que as pessoas fora da minha conta da AWS acessem meus recursos do Amazon Lex \(p. 198\)](#)

## Não tenho autorização para executar uma ação no Amazon Lex

Se o Console de gerenciamento da AWS informar que você não está autorizado a executar uma ação, entre em contato com o administrador para obter assistência. O administrador é a pessoa que forneceu a você o seu nome de usuário e senha.

Por exemplo, o erro a seguir ocorre quando o usuário `mateojackson` do IAM tenta usar o console para visualizar detalhes sobre um bot, mas não tem permissões `lex:GetBot`:

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform: lex:GetBot
on resource: OrderPizza
```

Neste caso, Mateo deve pedir ao administrador para atualizar suas políticas para permitir que ele tenha acesso ao bot `OrderPizza` usando a ação `lex:GetBot`.

## Não estou autorizado a executar iam:PassRole

Se você receber uma mensagem de erro informando que você não está autorizado a executar a ação `iam:PassRole`, entre em contato com o administrador para obter assistência. O administrador é a pessoa

que forneceu a você o seu nome de usuário e senha. Peça a essa pessoa para atualizar suas políticas para permitir que você passe uma função para o Amazon Lex.

Alguns serviços da AWS permitem que você passe uma função existente para o serviço, em vez de criar uma nova função de serviço ou função vinculada ao serviço. Para fazer isso, um usuário deve ter permissões para passar a função para o serviço.

O erro de exemplo a seguir ocorre quando uma usuária do IAM chamada `marymajor` tenta usar o console para executar uma ação no Amazon Lex. No entanto, a ação exige que o serviço tenha permissões concedidas por uma função de serviço. Mary não tem permissões para passar a função para o serviço.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform: iam:PassRole
```

Neste caso, Mary pede ao administrador para atualizar suas políticas para permitir que ela execute a ação `iam:PassRole`.

## Quero visualizar minhas chaves de acesso

Depois de criar suas chaves de acesso de usuário do IAM, você pode visualizar seu ID de chave de acesso a qualquer momento. No entanto, você não pode visualizar sua chave de acesso secreta novamente. Se você perder sua chave secreta, crie um novo par de chaves de acesso.

As chaves de acesso consistem em duas partes: um ID de chave de acesso (por exemplo, `AKIAIOSFODNN7EXAMPLE`) e uma chave de acesso secreta (por exemplo, `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`). Como um nome de usuário e uma senha, você deve usar o ID da chave de acesso e a chave de acesso secreta em conjunto para autenticar suas solicitações. Gerencie suas chaves de acesso de forma tão segura quanto você gerencia seu nome de usuário e sua senha.

### Important

Não forneça as chaves de acesso a terceiros, mesmo que seja para ajudar a [encontrar seu ID de usuário canônico](#). Ao fazer isso, você pode dar a alguém acesso permanente à sua conta.

Ao criar um par de chaves de acesso, você é solicitado a guardar o ID da chave de acesso e a chave de acesso secreta em um local seguro. A chave de acesso secreta só está disponível no momento em que é criada. Se você perder sua chave de acesso secreta, você deverá adicionar novas chaves de acesso para seu usuário do IAM. Você pode ter no máximo duas chaves de acesso. Se você já tiver duas, você deverá excluir um par de chaves para poder criar um novo. Para visualizar as instruções, consulte [Gerenciar chaves de acesso](#) no Guia do usuário do IAM.

## Sou administrador e desejo conceder acesso ao Amazon Lex para outros usuários.

Para permitir que outros usuários acessem o Amazon Lex, crie uma entidade do IAM (usuário ou função) para a pessoa ou o aplicativo que precisa do acesso. Eles usarão as credenciais dessa entidade para acessar a AWS. Você deve anexar uma política à entidade que concede a eles as permissões corretas no Amazon Lex.

Para começar a usar imediatamente, consulte [Criar os primeiros usuário e grupo delegados do IAM](#) no Guia do usuário do IAM.

## Quero permitir que as pessoas fora da minha conta da AWS acessem meus recursos do Amazon Lex

Você pode criar uma função que os usuários de outras contas ou pessoas fora da sua organização podem usar para acessar seus recursos. Você pode especificar quem é confiável para assumir a função. Para

serviços que oferecem suporte a políticas baseadas em recursos ou listas de controle de acesso (ACLs), você pode usar essas políticas para conceder às pessoas acesso a seus recursos.

Para saber mais, consulte o seguinte:

- Para saber se o Amazon Lex oferece suporte a esses recursos, consulte [Como o Amazon Lex funciona com o IAM \(p. 188\)](#).
- Para saber como conceder acesso aos seus recursos em todas as contas da AWS pertencentes a você, consulte [Conceder acesso a um usuário do IAM em outra conta da AWS pertencente a você](#) no Guia do usuário do IAM.
- Para saber como conceder acesso aos seus recursos para contas da AWS de terceiros, consulte [Conceder acesso a contas da AWS pertencentes a terceiros](#) no Guia do usuário do IAM.
- Para saber como fornecer acesso por meio de federação de identidades, consulte [Fornecer acesso a usuários autenticados externamente \(federação de identidades\)](#) no Guia do usuário do IAM.
- Para saber a diferença entre usar funções e políticas baseadas em recurso para acesso entre contas, consulte [Como as funções do IAM diferem de políticas baseadas em recursos](#) no Guia do usuário do IAM.

## Monitorar o Amazon Lex

O monitoramento é importante para manter a confiabilidade, a disponibilidade e o desempenho dos chatbots do Amazon Lex. Este tópico descreve como usar o Amazon CloudWatch Logs e o AWS CloudTrail para monitorar o Amazon Lex e descreve as métricas de associação de canal e o tempo de execução do Amazon Lex.

Tópicos

- [Monitoramento do Amazon Lex com o Amazon CloudWatch \(p. 199\)](#)
- [Monitoramento de chamadas da API do Amazon Lex com logs do AWS CloudTrail \(p. 204\)](#)

## Monitoramento do Amazon Lex com o Amazon CloudWatch

Para acompanhar a integridade dos bots do Amazon Lex, use o Amazon CloudWatch. Com o CloudWatch, você pode obter métricas para operações Amazon Lex individuais ou operações globais do Amazon Lex para a sua conta. Você também pode configurar os alarmes do CloudWatch para ser notificado quando uma ou mais métricas excederem um limite definido por você. Por exemplo, você pode monitorar o número de solicitações feitas a um bot durante um determinado período de tempo, visualizar a latência de solicitações bem-sucedidas ou gerar um alarme quando erros excederem um limite.

### Métricas do CloudWatch para Amazon Lex

Para obter as métricas para as operações do Amazon Lex, você deve especificar as seguintes informações:

- A dimensão da métrica. Uma dimensão é um conjunto de pares de nome-valor que você usa para identificar uma métrica. O Amazon Lex tem três dimensões:
  - BotAlias, BotName, Operation
  - BotAlias, BotName, InputMode, Operation
  - BotName, BotVersion, InputMode, Operation
- O nome da métrica, como MissedUtteranceCount ou RuntimeRequestCount.

Você pode obter as métricas para o Amazon Lex com o Console de gerenciamento da AWS, a AWS CLI ou a API do CloudWatch. Você pode usar a API do CloudWatch por meio de um dos Kits de desenvolvimento de software (SDKs) da Amazon AWS ou das ferramentas da API do CloudWatch. O console do Amazon Lex exibe gráficos com base nos dados brutos da API do CloudWatch.

Você deve ter as permissões do CloudWatch apropriadas para monitorar o Amazon Lex com o CloudWatch. Para obter mais informações, consulte [Controle de acesso e autenticação para o Amazon CloudWatch](#) no Guia do usuário do Amazon CloudWatch.

## Visualizar métricas do Amazon Lex

Visualize métricas do Amazon Lex usando o console do Amazon Lex ou o console do CloudWatch.

Para visualizar métricas (console do Amazon Lex)

1. Faça login no Console de gerenciamento da AWS e abra o console do Amazon Lex em <https://console.aws.amazon.com/lex/>.
2. Na lista de bots, escolha aqueles cujas métricas você deseja ver.
3. Escolha Monitoring. As métricas serão exibidos em gráficos.

Para visualizar métricas (console do CloudWatch)

1. Faça login no Console de gerenciamento da AWS e abra o console da CloudWatch em <https://console.aws.amazon.com/cloudwatch/>.
2. Escolha Metrics, All Metrics e, em seguida, selecione AWS/Lex.
3. Escolha a dimensão, informe um nome de métrica e selecione Add to graph (Adicionar ao gráfico).
4. Escolha um valor para o intervalo de datas. A contagem da métrica para o intervalo de datas selecionado é exibida no gráfico.

## Criar um alarme

Um alarme do CloudWatch observa uma única métrica por um período especificado e executa uma ou mais ações: envio de uma notificação a um tópico do Amazon Simple Notification Service (Amazon SNS) ou política do Auto Scaling. A ação ou ações são baseadas no valor da métrica relativa a um limite determinado por um número de períodos que você especifica. O CloudWatch também pode enviar uma mensagem do Amazon SNS quando o alarme mudar de estado.

Os alarmes do CloudWatch invocam ações somente quando o estado mudar e tiver persistido pelo período especificado por você.

Para definir um alarme

1. Faça login no Console de gerenciamento da AWS e abra o console da CloudWatch em <https://console.aws.amazon.com/cloudwatch/>.
2. Escolha Alarms e, em seguida, Create Alarm.
3. Escolha AWS/Lex Metrics e, em seguida, escolha uma métrica.
4. Para Time Range, escolha um intervalo de tempo para monitorar e, em seguida, selecione Next.
5. Insira um Name (Nome) e uma Description (Descrição).
6. Para Whenever, escolha  $\geq$  e digite um valor máximo.
7. Se você quiser que o CloudWatch envie um e-mail quando o estado do alarme for atingido, na seção Actions (Ações), em Whenever this alarm (Sempre que este alarme), escolha State is ALARM (Estado é ALARME). Em Send notification to (Enviar notificação para), selecione uma lista de correspondência ou selecione New list (Nova lista) e crie uma nova.

8. Visualize o alarme na seção Alarm Preview. Se você estiver satisfeito com o alarme, selecione Create Alarm.

## Métricas do CloudWatch para o tempo de execução do Amazon Lex

A tabela a seguir descreve as métricas de tempo de execução do Amazon Lex.

| Métrica                                    | Descrição  |
|--|--|
| <code>RuntimeInvalidLambdaResponses</code> | <p>O número de respostas inválidas do AWS Lambda (Lambda) no período especificado.</p> <p>Dimensão válida para a operação <code>PostContent</code> com o <code>InputMode</code> <code>Text</code> ou <code>Speech</code>:</p> <ul style="list-style-type: none"><li>• <code>BotName</code>, <code>BotAlias</code>, <code>Operation</code>, <code>InputMode</code></li></ul> <p>Dimensão válida para a operação <code>PostText</code>:</p> <ul style="list-style-type: none"><li>• <code>BotName</code>, <code>BotAlias</code>, <code>Operation</code></li></ul>  |
| <code>RuntimeLambdaErrors</code>           | <p>O número de erros de tempo de execução do Lambda no período especificado.</p> <p>Dimensão válida para a operação <code>PostContent</code> com o <code>InputMode</code> <code>Text</code> ou <code>Speech</code>:</p> <ul style="list-style-type: none"><li>• <code>BotName</code>, <code>BotAlias</code>, <code>Operation</code>, <code>InputMode</code></li></ul> <p>Dimensão válida para a operação <code>PostText</code>:</p> <ul style="list-style-type: none"><li>• <code>BotName</code>, <code>BotAlias</code>, <code>Operation</code></li></ul>  |
| <code>MissedUtteranceCount</code>          | <p>O número de enunciados que não foram reconhecidos no período especificado.</p> <p>Dimensões válidas para a operação <code>PostContent</code> com o <code>InputMode</code> <code>Text</code> ou <code>Speech</code>:</p> <ul style="list-style-type: none"><li>• <code>BotName</code>, <code>BotVersion</code>, <code>Operation</code>, <code>InputMode</code></li><li>• <code>BotName</code>, <code>BotAlias</code>, <code>Operation</code>, <code>InputMode</code></li></ul> <p>Dimensões válidas para a operação <code>PostText</code>:</p> <ul style="list-style-type: none"><li>• <code>BotName</code>, <code>BotVersion</code>, <code>Operation</code></li><li>• <code>BotName</code>, <code>BotAlias</code>, <code>Operation</code></li></ul> |
| <code>RuntimePollyErrors</code>            | <p>O número de respostas inválidas do Amazon Polly no período especificado.</p> <p>Dimensão válida para a operação <code>PostContent</code> com o <code>InputMode</code> <code>Text</code> ou <code>Speech</code>:</p> <ul style="list-style-type: none"><li>• <code>BotName</code>, <code>BotAlias</code>, <code>Operation</code>, <code>InputMode</code></li></ul>   |

| Métrica                   | Descrição   |
|---------------------------|---|
|                           | Dimensão válida para a operação PostText: <ul style="list-style-type: none"><li>• BotName, BotAlias, Operation</li></ul>  |
| RuntimeRequestCount       | <p>O número de solicitações de tempo de execução no período especificado.</p> <p>Dimensões válidas para a operação PostContent com o InputMode Text ou Speech:</p> <ul style="list-style-type: none"><li>• BotName, BotVersion, Operation, InputMode</li><li>• BotName, BotAlias, Operation, InputMode</li></ul> <p>Dimensões válidas para a operação PostText:</p> <ul style="list-style-type: none"><li>• BotName, BotVersion, Operation</li><li>• BotName, BotAlias, Operation</li></ul> <p>Unidade: contagem</p>  |
| RuntimeSuccessfulRequests | <p>Atenção: O número de solicitações bem-sucedidas entre o horário em que a solicitação foi feita e a resposta foi passada.</p> <p>Dimensões válidas para a operação PostContent com o InputMode Text ou Speech:</p> <ul style="list-style-type: none"><li>• BotName, BotVersion, Operation, InputMode</li><li>• BotName, BotAlias, Operation, InputMode</li></ul> <p>Dimensões válidas para a operação PostText:</p> <ul style="list-style-type: none"><li>• BotName, BotVersion, Operation</li><li>• BotName, BotAlias, Operation</li></ul> <p>Unidade: milissegundos</p> |
| RuntimeSystemErrors       | <p>O número de erros do sistema no período especificado. O intervalo de códigos de resposta para um erro do sistema vai de 500 até 599.</p> <p>Dimensão válida para a operação PostContent com o InputMode Text ou Speech:</p> <ul style="list-style-type: none"><li>• BotName, BotAlias, Operation, InputMode</li></ul> <p>Dimensão válida para a operação PostText:</p> <ul style="list-style-type: none"><li>• BotName, BotAlias, Operation</li></ul> <p>Unidade: contagem</p>   |

| Métrica                             | Descrição   |
|-------------------------------------|---|
| <code>RuntimeThrottledEvents</code> | <p>O número de solicitações limitadas. O Amazon Lex limita uma solicitação ao receber mais solicitações do que o limite de transações por segundo definido para a conta. Se o limite definido para a conta for frequentemente excedido, você poderá solicitar um aumento no limite. Para solicitar um aumento, consulte <a href="#">Limites de serviço da AWS</a>.</p> <p>Dimensão válida para a operação <code>PostContent</code> com o <code>InputMode Text</code> ou <code>Speech</code>:</p> <ul style="list-style-type: none"><li>• <code>BotName</code>, <code>BotAlias</code>, <code>Operation</code>, <code>InputMode</code></li></ul> <p>Dimensão válida para a operação <code>PostText</code>:</p> <ul style="list-style-type: none"><li>• <code>BotName</code>, <code>BotAlias</code>, <code>Operation</code></li></ul> <p>Unidade: contagem</p> |
| <code>RuntimeUserErrors</code>      | <p>O número de erros de usuário no período especificado. O intervalo de códigos de resposta para um erro de usuário vai de 400 até 499.</p> <p>Dimensão válida para a operação <code>PostContent</code> com <code>InputMode Text</code> ou <code>Speech</code>:</p> <ul style="list-style-type: none"><li>• <code>BotName</code>, <code>BotAlias</code>, <code>Operation</code>, <code>InputMode</code></li></ul> <p>Dimensão válida para a operação <code>PostText</code>:</p> <ul style="list-style-type: none"><li>• <code>BotName</code>, <code>BotAlias</code>, <code>Operation</code></li></ul> <p>Unidade: contagem</p>  |

As métricas de tempo de execução do Amazon Lex usam o namespace do AWS/Lex e fornecem métricas nas dimensões a seguir. Você pode agrupar as métricas por dimensões no console do CloudWatch:

| Dimensão   | Descrição  |
|--|--|
| <code>BotName</code> , <code>BotAlias</code> , <code>Operation</code> , <code>InputMode</code>   | Agrupar as métricas por alias do bot, nome do bot, operação ( <code>PostContent</code> ) e se a entrada foi de texto ou fala.  |
| <code>BotName</code> , <code>BotVersion</code> , <code>Operation</code> , <code>InputMode</code> | Agrupar as métricas por nome do bot, versão do bot, operação ( <code>PostContent</code> ) e se a entrada foi de texto ou fala. |
| <code>BotName</code> , <code>BotVersion</code> , <code>Operation</code>                          | Agrupar as métricas por nome do bot, versão do bot e operação, <code>PostText</code> .   |
| <code>BotName</code> , <code>BotAlias</code> , <code>Operation</code>                            | Agrupar as métricas por nome do bot, alias do bot e operação, <code>PostText</code> .  |



## Métricas do CloudWatch para associações de canal do Amazon Lex

Uma associação de canal é aquela entre o Amazon Lex e um canal de mensagens, como o Facebook. A tabela a seguir descreve as métricas de associação de canal do Amazon Lex.

| Métrica                       | Descrição   |
|-------------------------------|---|
| BotChannelAuthErrors          | O número de erros de autenticação retornado pelo canal de mensagens no período especificado. Um erro de autenticação indica que o token secreto fornecido durante a criação de canal é inválido ou expirou. |
| BotChannelConfigurationErrors | O número de erros de configuração no período especificado. Um erro de configuração indica que uma ou mais entradas de configuração para o canal são inválidas.  |
| BotChannelInboundThrottles    | O número de vezes que as mensagens que foram enviadas pelo canal de mensagens foram limitadas pelo Amazon Lex no período especificado.  |
| BotChannelOutboundThrottles   | O número de vezes que os eventos de saída do Amazon Lex para o canal de mensagens foram limitados no período de tempo especificado.   |
| BotChannelRequestCount        | O número de solicitações feitas em um canal no período especificado.  |
| BotChannelResponseCardErrors  | O número de vezes que o Amazon Lex não conseguiu publicar cartões de resposta no período especificado.  |
| BotChannelSystemErrors        | O número de erros internos que ocorreram no Amazon Lex para um canal no período especificado.   |

As métricas de associação de canal do Amazon Lex usam o namespace do AWS/Lex e fornecem métricas para a dimensão a seguir. Você pode agrupar as métricas por dimensões no console do CloudWatch:

| Dimensão  | Descrição   |
|---|---|
| BotAlias,<br>BotChannelName,<br>BotName, Source | Agrupam as métricas por alias do bot, nome do canal, nome do bot e origem do tráfego. |

## Monitoramento de chamadas da API do Amazon Lex com logs do AWS CloudTrail

O Amazon Lex é integrado com o AWS CloudTrail, um serviço que fornece um registro das ações executadas por um usuário, função ou um serviço do AWS em Amazon Lex faz. O CloudTrail captura um subconjunto das chamadas à API do Amazon Lex como eventos, incluindo as chamadas do console do Amazon Lex e as chamadas de código das APIs do Amazon Lex. Se você criar uma trilha, poderá habilitar a entrega contínua de eventos do CloudTrail para um bucket do Amazon S3, incluindo eventos para o Amazon Lex. Se não configurar uma trilha, você ainda poderá visualizar os eventos mais recentes no console do CloudTrail em Event history. Com as informações coletadas pelo CloudTrail, determine a solicitação feita para o Amazon Lex, o endereço IP do qual a solicitação foi feita, quem fez a solicitação, quando ela foi feita e detalhes adicionais.

Para saber mais sobre o CloudTrail, incluindo como configurá-lo e habilitá-lo, consulte o [AWS CloudTrail User Guide](#).

## Informações sobre o Amazon Lex no CloudTrail

O CloudTrail está habilitado na sua conta da AWS ao criá-la. Quando a atividade do evento com suporte ocorre no Amazon Lex, ela é registrada em um evento do CloudTrail junto com outros eventos de serviços da AWS em Event history (Histórico de eventos). Você pode visualizar, pesquisar e fazer download de eventos recentes em sua conta da AWS. Para obter mais informações, consulte [Visualizar eventos com o histórico de eventos do CloudTrail](#).

Para obter um registro contínuo de eventos em sua conta da AWS, incluindo eventos para o Amazon Lex, crie uma trilha. Uma trilha habilita o CloudTrail para fornecer arquivos de log a um bucket do Amazon Simple Storage Service (Amazon S3). Por padrão, quando você cria uma trilha no console, ela é aplicada a todas as regiões da AWS. A trilha registra eventos de todas as regiões na partição da AWS e fornece os arquivos de log ao bucket do S3 que você especificar. Além disso, é possível configurar outros serviços da AWS para analisar mais profundamente e agir sobre os dados de evento coletados nos logs do CloudTrail. Para obter mais informações, consulte:

- [Visão geral da criação de uma trilha](#)
- [CloudTrail Serviços compatíveis e integrações do](#)
- [Configuração de notificações do Amazon SNS para o CloudTrail](#)
- [Recebimento de arquivos de log do CloudTrail de várias regiões](#) e [Recebimento de arquivos de log do CloudTrail de várias contas](#)

O Amazon Lex oferece suporte ao registro em log das seguintes operações como eventos nos arquivos de log do CloudTrail:

- [CreateBotVersion](#) (p. 219)
- [CreateIntentVersion](#) (p. 224)
- [CreateSlotTypeVersion](#) (p. 230)
- [PutBot](#) (p. 311)
- [PutBotAlias](#) (p. 319)
- [PutIntent](#) (p. 323)
- [PutSlotType](#) (p. 333)

Cada entrada de log ou evento contém informações sobre quem gerou a solicitação. As informações ajudam a identificar:

- Se a solicitação foi feita com credenciais de usuário raiz ou do IAM
- Se a solicitação foi feita com credenciais de segurança temporárias de uma função ou de um usuário federado
- Se a solicitação foi feita por outro serviço da AWS

Para obter mais informações, consulte [Elemento userIdentity do CloudTrail](#).

Para obter informações sobre as ações do Amazon Lex que foram registradas em logs do CloudTrail, consulte [Serviço de criação de modelo do Amazon Lex](#). Por exemplo, as chamadas para as operações [PutBot](#) (p. 311), [GetBot](#) (p. 252) e [DeleteBot](#) (p. 234) geram entradas nos arquivos no log do CloudTrail. As ações documentadas no [Serviço de tempo de execução do Amazon Lex](#), [PostContent](#) (p. 347) e [PostText](#) (p. 355), não são registradas.

## Exemplo: entradas do arquivo de log do Amazon Lex

Uma trilha é uma configuração que permite a entrega de eventos como arquivos de log em um bucket do S3 que você especificar. Os arquivos de log do CloudTrail contêm uma ou mais entradas de log. Um

evento representa uma única solicitação de qualquer origem e inclui informações sobre a ação solicitada, a data e hora da ação, os parâmetros da solicitação e assim por diante. Os arquivos de log do CloudTrail não são um rastreamento de pilha ordenada de chamadas da API pública. Assim, eles não são exibidos em nenhuma ordem específica.

O exemplo de entrada de log do CloudTrail a seguir mostra o resultado de uma chamada à operação PutBot.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole | FederatedUser | IAMUser | Root | SAMLUser |
WebIdentityUser",
    "principalId": "principal ID",
    "arn": "ARN",
    "accountId": "account ID",
    "accessKeyId": "access key ID",
    "userName": "user name"
  },
  "eventTime": "timestamp",
  "eventSource": "lex.amazonaws.com",
  "eventName": "PutBot",
  "awsRegion": "region",
  "sourceIPAddress": "source IP address",
  "userAgent": "user agent",
  "requestParameters": {
    "name": "CloudTrailBot",
    "intents": [
      {
        "intentVersion": "11",
        "intentName": "TestCloudTrail"
      }
    ],
    "voiceId": "Salli",
    "childDirected": false,
    "locale": "en-US",
    "idleSessionTTLInSeconds": 500,
    "processBehavior": "BUILD",
    "description": "CloudTrail test bot",
    "clarificationPrompt": {
      "messages": [
        {
          "contentType": "PlainText",
          "content": "I didn't understand you. wWat would you like to
do?"
        }
      ],
      "maxAttempts": 2
    },
    "abortStatement": {
      "messages": [
        {
          "contentType": "PlainText",
          "content": "Sorry. I'm not able to assist at this time."
        }
      ]
    }
  },
  "responseElements": {
    "voiceId": "Salli",
    "locale": "en-US",
    "childDirected": false,
    "abortStatement": {
      "messages": [
```

```
        {
            "contentType": "PlainText",
            "content": "Sorry. I'm not able to assist at this time."
        }
    ]
},
"status": "BUILDING",
"createdDate": "timestamp",
"lastUpdatedDate": "timestamp",
"idleSessionTTLInSeconds": 500,
"intents": [
    {
        "intentVersion": "11",
        "intentName": "TestCloudTrail"
    }
],
"clarificationPrompt": {
    "messages": [
        {
            "contentType": "PlainText",
            "content": "I didn't understand you. What would you like to
do?"
        }
    ],
    "maxAttempts": 2
},
"version": "$LATEST",
"description": "CloudTrail test bot",
"checksum": "checksum",
"name": "CloudTrailBot"
},
"requestID": "request ID",
"eventID": "event ID",
"eventType": "AwsApiCall",
"recipientAccountId": "account ID"
}
}
```

## Validação de conformidade do Amazon Lex

Audidores externos avaliam a segurança e a conformidade do Amazon Lex como parte de vários programas de conformidade da AWS. O Amazon Lex não está no escopo de nenhum dos programas de conformidade da AWS.

Para obter uma lista de serviços da AWS no escopo de programas de conformidade específicos, consulte [Serviços da AWS no escopo do programa de conformidade](#). Para obter informações gerais, consulte [Programas de conformidade da AWS](#).

Os relatórios de auditoria de terceiros estão disponíveis para download por meio do AWS Artifact. Para obter mais informações, consulte [Baixar relatórios no AWS Artifact](#).

Para obter mais informações sobre programas de conformidade da AWS, consulte [Programas de conformidade da AWS](#).

Sua responsabilidade de conformidade ao usar o Amazon Lex é determinada pela confidencialidade de seus dados, pelas metas de conformidade de sua empresa e pelos regulamentos e leis aplicáveis. Se o seu uso do Amazon Lex estiver sujeito à conformidade com padrões, como HIPAA, PCI ou FedRAMP, a AWS fornecerá os seguintes recursos para ajudar:

- [Guias de Início Rápido de segurança e conformidade](#) – Guias de implantação que abordam as considerações de arquitetura e fornecem etapas para implantação de ambientes de linha de base focados em conformidade e segurança na AWS.
- [Whitepaper Arquitetura de segurança e conformidade da HIPAA](#) – um whitepaper que descreve como as empresas podem usar a AWS para criar aplicativos em conformidade com a HIPAA
- [Recursos de conformidade da AWS](#) – Uma coleção de manuais e guias que pode ser aplicada ao seu setor e local
- [AWS Config](#) – Um serviço que avalia até que ponto suas configurações de recursos estão em conformidade com práticas internas e diretrizes e regulamentações do setor.
- [AWS Security Hub](#) – Uma visão abrangente do estado da segurança na AWS que ajuda você a verificar a conformidade com os padrões e as melhores práticas do setor de segurança.

## Resiliência no Amazon Lex

A infraestrutura global da AWS é criada com base em regiões e zonas de disponibilidade da AWS. As regiões da AWS fornecem várias zonas de disponibilidade separadas e isoladas fisicamente, que são conectadas com baixa latência, altas taxas de transferência e redes altamente redundantes. Com as zonas de disponibilidade, você pode projetar e operar aplicativos e bancos de dados que executam o failover automaticamente entre as zonas de disponibilidade sem interrupção. As zonas de disponibilidade são mais altamente disponíveis, tolerantes a falhas e escaláveis que uma ou várias infraestruturas de data center tradicionais.

Para obter mais informações sobre regiões e zonas de disponibilidade da AWS, consulte [Infraestrutura global da AWS](#).

Além da infraestrutura global da AWS, o Amazon Lex oferece vários recursos para ajudar a oferecer suporte às suas necessidades de resiliência de dados e backup.

## Segurança da infraestrutura no Amazon Lex

Como serviço gerenciado, o Amazon Lex é protegido pelos procedimentos de segurança da rede global da AWS que estão descritos no whitepaper [Amazon Web Services: visão geral dos processos de segurança](#).

Você usa chamadas de API publicadas pela AWS para acessar o Amazon Lex por meio da rede. Os clientes devem ter suporte ao TLS (Transport Layer Security) 1.0. Recomendamos TLS 1.2 ou posterior. Os clientes também devem ter suporte a pacotes de criptografia com Perfect Forward Secrecy (PFS — Sigilo de encaminhamento perfeito) como Ephemeral Diffie-Hellman (DHE) ou Ephemeral Elliptic Curve Diffie-Hellman (ECDHE). A maioria dos sistemas modernos, como Java 7 e versões posteriores, oferece suporte a esses modos. Além disso, as solicitações devem ser assinadas usando um ID da chave de acesso e uma chave de acesso secreta associada a uma entidade principal do IAM. Ou você pode usar o [AWS Security Token Service](#) (AWS STS) para gerar credenciais de segurança temporárias e assinar solicitações.

Você pode chamar essas operações de API de qualquer local da rede, mas o Amazon Lex oferece suporte a políticas de acesso com base em recursos, que podem incluir restrições com base no endereço IP de origem. Você também pode usar políticas específicas do Amazon Lex para controlar o acesso de Amazon Virtual Private Cloud (Amazon VPC) endpoints ou VPCs específicas. Efetivamente, isso isola o acesso à rede para determinado recurso do Amazon Lex apenas da VPC específica dentro da rede da AWS.

# Diretrizes e limites no Amazon Lex

As seções a seguir fornecem as diretrizes e os limites ao usar o Amazon Lex.

## Tópicos

- [Diretrizes gerais \(p. 209\)](#)
- [Limites \(p. 211\)](#)

## Diretrizes gerais

Esta seção descreve as diretrizes gerais ao usar o Amazon Lex.

- Assinatura de solicitações – todas as operações de API de tempo de execução e de criação de modelo do Amazon Lex na [API Reference \(p. 216\)](#) usam o Signature V4 para solicitações de autenticação. Para mais informações sobre como autenticar solicitações, consulte [Processo de assinatura do Signature versão 4](#) no Referência geral do Amazon Web Services.

Para [PostContent \(p. 347\)](#), o Amazon Lex usa a opção de carga não assinada descrita em [Cálculos de assinatura para o cabeçalho de autorização: transferência de carga em um único bloco \(AWS Signature versão 4\)](#) na Referência da API do Amazon Simple Storage Service (S3).

Quando você usar a opção de carga não assinada, não inclua o hash da carga na solicitação canônica. Em vez disso, use a string literal "UNSIGNED-PAYLOAD" como o hash de carga. Inclua também um cabeçalho com o nome `x-amz-content-sha256` e o valor `UNSIGNED-PAYLOAD` na solicitação `PostContent`.

- Observe o seguinte sobre como o Amazon Lex captura valores de slot de enunciados de usuário:

O Amazon Lex usa os valores de enumeração fornecidos em uma definição de tipo de slot para treinar seus modelos de Machine Learning. Suponha que você defina uma intenção chamada `GetPredictionIntent` com o seguinte utterance de amostra:

```
"Tell me the prediction for {Sign}"
```

Onde `{Sign}` é um slot de `ZodiacSign` do tipo personalizado. Ele tem 12 valores de enumeração, de `Aries` a `Pisces`. Do enunciado do usuário "Tell me the prediction for ... (Informe a previsão para ...)", o Amazon Lex entende que o que vem a seguir é um signo do zodiaco.

Quando o campo `valueSelectionStrategy` é definido como `ORIGINAL_VALUE` usando a operação [PutSlotType \(p. 333\)](#), ou se Expand values (Expandir valores) for selecionado no console, se o usuário disser "Tell me the prediction for earth (Informe-me a previsão da terra)", o Amazon Lex inferirá que "earth (terra)" é um `ZodiacSign` e passará isso para seu aplicativo cliente ou para funções do Lambda. Você deve verificar quais valores do slot são válidos antes de usá-los em sua atividade de cumprimento.

Se você define o campo `valueSelectionStrategy` para `TOP_RESOLUTION` usando a operação [PutSlotType](#) (p. 333), ou se `Restrict to slot values and synonyms` está selecionada no console, os valores retornados são limitados aos valores que você definiu para o tipo de slot. Por exemplo, se o usuário diz "Fale a previsão para terra", o valor não seria reconhecido, porque não é um dos valores definidos para o tipo de slot. Ao definir sinônimos para valores de slot, eles são reconhecidos da mesma forma que um valor de slot. No entanto, o valor de slot é retornado ao invés do sinônimo.

Quando o Amazon Lex chama uma função do Lambda ou retorna o resultado de uma interação de fala com o aplicativo cliente, a capitalização dos valores do slot não é garantida. Por exemplo, se você estiver inferindo valores para o tipo de slot integrado `AMAZON.Movie` e um usuário disser ou digitar "Gone with the wind (E o vento levou)", o Amazon Lex poderá retornar "Gone with the Wind", "gone with the wind" ou "Gone With The Wind". Em interações de texto, a capitalização dos valores de slot corresponde ao texto inserido ou ao valor de slot, dependendo do valor do campo `valueResolutionStrategy`.

- O Amazon Lex não oferece suporte ao tipo de slot integrado `AMAZON.LITERAL`, como o Alexa Skills Kit oferece. No entanto, o Amazon Lex oferece suporte à criação de tipos de slot personalizados que você pode usar para implementar essa funcionalidade. Como mencionado na observação anterior, você pode capturar valores fora da definição de tipo de slot personalizado. Adicione mais e diversificados valores de enumeração para aumentar a precisão de reconhecimento automático de voz (ASR) e compreensão de linguagem natural (NLU).
- Os tipos de slot integrado `AMAZON.DATE` e `AMAZON.TIME` capturam as datas e horas absolutas e relativas. As datas e horas relativas são resolvidas na região em que o Amazon Lex está processando a solicitação.

Para o tipo de slot integrado `AMAZON.TIME`, se o usuário não especificar que um horário é antes ou depois do meio-dia, o horário será ambíguo e o Amazon Lex perguntará ao usuário novamente. Recomendamos que um horário absoluto seja escolhido para as solicitações. Por exemplo, use uma solicitação como "Em qual horário você deseja que a pizza seja entregue? Você pode dizer 18h ou 6 da tarde".

- O fornecimento de dados de treinamento confusos em seu bot reduz a capacidade do Amazon Lex de compreender a entrada do usuário. Considere estes exemplos:

Suponha que você tenha duas intenções (`OrderPizza` e `OrderDrink`) no seu bot e ambas estejam configuradas com um enunciado "Quero pedir". Esse enunciado não é mapeado para uma intenção específica da qual o Amazon Lex possa aprender ao criar o modelo de linguagem para o bot no tempo de criação. Como resultado, quando um usuário insere esse enunciado em tempo de execução, o Amazon Lex não pode escolher uma intenção com alto grau de confiança.

Considere outro exemplo, em que você define uma intenção personalizada para obter uma confirmação do usuário (por exemplo, `MyCustomConfirmationIntent`) e configura a intenção com os enunciados "Sim" e "Não". Observe que o Amazon Lex também tem um modelo de linguagem para entender

confirmações de usuário. Isso pode criar uma situação de conflito. Quando o usuário responde com um "Sim", isso significa que é uma confirmação da intenção em andamento ou que o usuário está solicitando a intenção personalizada que você criou?

Em geral, os utterances de amostra fornecidos devem ser mapeados para uma intenção específica e, opcionalmente, para valores de slot específicos.

- As operações de API de runtime [PostContent \(p. 347\)](#) e [PostText \(p. 355\)](#) reconhecem um ID de usuário como o parâmetro obrigatório. Os desenvolvedores podem configurar isso para qualquer valor que atenda às restrições descritas na API. Recomendamos que você não use esse parâmetro para enviar informações confidenciais como logins de usuário, e-mails ou números de seguro social. Esse ID é usado principalmente para identificar exclusivamente a conversa com um bot (pode haver vários usuários pedindo pizza).
- Se o aplicativo cliente usar o Amazon Cognito para autenticação, você poderá usar o ID de usuário do Amazon Cognito como o ID de usuário do Amazon Lex. Observe que qualquer função do Lambda configurada para o bot deve ter seu próprio mecanismo de autenticação para identificar o usuário em cujo nome o Amazon Lex está invocando a função do Lambda.
- Recomendamos que você defina uma intenção que capte a intenção de um usuário de interromper a conversa. Por exemplo, você pode definir uma intenção (`NothingIntent`) com enunciados de exemplo ("I don't want anything (Eu não quero nada)", "exit (sair)", "bye bye (tchau tchau)"), nenhum slot e nenhuma função do Lambda configurada como um gancho de código. Isso permite que os usuários fechem uma conversa com tranquilidade.

## Limites

Esta seção descreve os limites atuais no Amazon Lex. Esses limites são agrupados por categorias.

### Tópicos

- [Limites gerais \(p. 211\)](#)
- [Limites de serviço de tempo de execução \(p. 211\)](#)
- [Limites de criação de modelos \(p. 212\)](#)

## Limites gerais

Para obter uma lista de regiões da AWS onde o Amazon Lex está disponível, consulte [Regiões e endpoints da AWS](#) na Referência geral da Amazon Web Services.

Atualmente, o Amazon Lex oferece suporte somente ao idioma inglês dos EUA. Ou seja, o Amazon Lex treina seus bots para entender apenas inglês dos EUA.

## Limites de serviço de tempo de execução

Além dos limites descritos na referência de API, observe que:



## Limites do API

- A entrada de voz na operação [PostContent \(p. 347\)](#) pode ter até 15 segundos.
- Em ambas as operações da API de runtime, [PostContent \(p. 347\)](#) e [PostText \(p. 355\)](#), o tamanho do texto inserido pode ser de até 1.024 caracteres Unicode.
- O tamanho máximo de cabeçalhos `PostContent` é 16 KB. O tamanho máximo combinado de solicitações e cabeçalhos de sessão é 12 KB.
- O tamanho máximo de entradas para uma função do Lambda é 12 KB. O tamanho máximo de saída é 25 KB, dos quais 12 KB podem ser atributos de sessão.

## Usar a versão `$LATEST`

- A versão `$LATEST` de seu bot deve ser usada apenas para teste manual. O Amazon Lex limita o número de solicitações em tempo de execução que podem ser feitas à versão `$LATEST` do bot.
- Quando você atualizar a versão `$LATEST` do bot, o Amazon Lex encerra todas as conversas em andamento para qualquer aplicativo cliente usando a versão `$LATEST` do bot. Em geral, você não deve usar a versão `$LATEST` de um bot em produção, pois a versão `$LATEST` pode ser atualizada. Em vez disso, você deve publicar uma versão e usá-la.
- Quando você atualiza um alias, o Amazon Lex leva alguns minutos para incorporar a alteração. Quando você modificar a `$LATEST` versão do bot, a alteração será capturada imediatamente.

## Tempo limite de sessão

- Esse tempo limite de sessão definido quando o bot foi criado determina por quanto tempo o bot retém o contexto de conversa, como a intenção e os dados de slot do usuário atual.
- Depois que um usuário inicia a conversa com o bot e até que a sessão expire, o Amazon Lex usa a mesma versão do bot, mesmo que você atualize o alias do bot para apontar para outra versão.

## Limites de criação de modelos

A criação de modelos se refere à criação e ao gerenciamento de bots. Isso inclui a criação e o gerenciamento de bots, intenções, tipos de slot, slots e associações de canal de bot.

### Tópicos

- [Limites de bots \(p. 213\)](#)

- [Limites de intenção \(p. 214\)](#)
- [Limites de tipo de slot \(p. 215\)](#)

## Limites de bots

- Os prompts e as instruções são configurados em toda a API de criação de modelos. Cada um desses prompts ou instruções pode ter até cinco mensagens, sendo que cada mensagem pode conter de 1 a 1.000 caracteres UTF-8.
- Ao usar grupos de mensagens, você pode definir até cinco grupos de mensagens para cada mensagem. Cada grupo de mensagens pode conter no máximo cinco mensagens e existe um limite de 15 mensagens em todos os grupos de mensagens.
- Você pode definir enunciados de amostra para intenções e slots. É possível usar um máximo de 200.000 caracteres para todos os enunciados.
- Cada tipo de slot pode definir um máximo de 10.000 valores e sinônimos. Cada bot pode conter um máximo de 50.000 valores e sinônimos de tipos de slots.
- Os nomes de bots, aliases e associações de canal de bot não diferenciam maiúsculas de minúsculas no momento da criação. Se você criar `PizzaBot` e, em seguida, tentar criar `pizzaBot`, você receberá um erro. No entanto, ao acessar um recurso, os nomes dos recursos diferenciam maiúsculas de minúsculas (você deve especificar que é `PizzaBot`, e não `pizzaBot`). Esses nomes devem ter entre 2 e 50 caracteres ASCII.
- O número máximo de versões que você pode publicar para todos os tipos de recurso é 100. Observe que não há versionamento para aliases.
- Em um bot, os nomes de intenção e de slot devem ser exclusivos, ou seja, você não pode ter uma intenção e um slot com o mesmo nome.
- Você pode criar um bot configurado para oferecer suporte a várias intenções. Se duas intenções tiverem um slot com o mesmo nome, o tipo de slot correspondente deve ser o mesmo.

Por exemplo, suponha que você crie um bot para oferecer suporte a duas intenções (`OrderPizza` e `OrderDrink`). Se essas duas intenções tiverem o slot `size`, então o tipo de slot deve ser o mesmo em ambos os locais.

Além disso, os enunciados de amostra fornecidos para um slot em uma das intenções se aplicam a um slot com mesmo nome em outras intenções.

- Você pode associar um máximo de 100 intenções a um bot.

- Ao criar um bot, especifique um tempo limite de sessão. O tempo limite de sessão pode ser entre um minuto e um dia. O padrão é cinco minutos.
- É possível criar até cinco aliases para um bot.
- Você pode criar até 100 bots por conta da AWS.
- Não é possível criar várias intenções que se estendam a partir da mesma intenção integrada.

## Limites de intenção

- Os nomes de slot e de intenção não diferenciam maiúsculas de minúsculas no momento da criação. Ou seja, se você criar a intenção `OrderPizza` e, em seguida, tentar criar outra intenção `orderPizza`, receberá uma mensagem de erro. No entanto, ao acessar esses recursos, os nomes dos recursos diferenciam maiúsculas de minúsculas; especifique `OrderPizza`, e não `orderPizza`. Esses nomes devem ter entre 1 e 100 caracteres ASCII.
- Uma intenção pode ter até 1,500 utterances de amostra. É necessário no mínimo um enunciado de amostra. Cada enunciado de amostra pode ter até 200 caracteres UTF-8. Você pode usar até 200.000 caracteres para todas as intenções e enunciados de slot em um bot. Um enunciado de amostra para uma intenção:
  - Pode fazer referência a zero ou mais nomes de slot.
  - Pode fazer referência a um nome de slot apenas uma vez.

Por exemplo:

```
I want a pizza
I want a {pizzaSize} pizza
I want a {pizzaSize} {pizzaTopping} pizza
```

- Embora cada intenção ofereça suporte a até 1.500 enunciados, se você usar menos enunciados, o Amazon Lex poderá ter uma capacidade melhor de reconhecer entradas fora do conjunto fornecido.
- Você pode criar até cinco grupos de mensagens para cada mensagem em uma intenção. Pode haver um total de 15 mensagens em todos os grupos de mensagens para uma mensagem.
- O console só pode criar grupos de mensagens para as mensagens `conclusionStatement` e `followUpPrompt`. Você pode criar grupos de mensagens para qualquer outra mensagem usando a API do Amazon Lex.
- Cada slot pode ter até 10 utterances de amostra. Cada utterance de amostra deve fazer referência ao nome do slot exatamente uma vez. Por exemplo:

```
{pizzaSize} please
```

- Cada bot pode ter, no máximo, 200.000 caracteres no total para enunciados de intenção e de slot.
- Você não pode fornecer utterances para intenções que se estendam a partir de intenções integradas. Para todas as outras intenções, você deve fornecer pelo menos um utterance de amostra. As intenções contêm slots, mas os utterances de amostra a nível de slot são opcionais.
- Intenções integradas
  - No momento, o Amazon Lex não oferece suporte à inferência de slots para intenções integradas. Não é possível criar funções do Lambda para retornar a diretiva `ElicitSlot` na resposta com uma intenção derivada de intenções integradas. Para obter mais informações, consulte [Formato de resposta \(p. 109\)](#).
  - O serviço não é compatível com a adição de utterances de amostra a intenções integradas. Da mesma forma, você não pode adicionar nem remover slots de intenções integradas.
- Você pode criar até 1.000 intenções por conta da AWS. Você pode criar até 100 slots em uma intenção.

## Limites de tipo de slot

- Os nomes de tipo de slot não diferenciam maiúsculas de minúsculas no momento da criação. Se você criar o tipo de slot `PizzaSize` e, em seguida, tentar criar o tipo de slot `pizzaSize`, receberá uma mensagem de erro. No entanto, ao acessar esses recursos, os nomes dos recursos diferenciam maiúsculas de minúsculas (você deve especificar que é `PizzaSize`, e não `pizzaSize`). Os nomes devem ter entre 1 e 100 caracteres ASCII.
- Um tipo de slot personalizado que você criar pode ter no máximo 10.000 valores de enumeração e sinônimos. Cada valor pode ter até 140 caracteres UTF-8. Os valores de enumeração e sinônimos não contêm duplicados.
- Para um valor de tipo de slot, especifique tanto as letras maiúsculas quanto as minúsculas, onde for apropriado. Por exemplo, para um tipo de slot chamado `Procedure`, se o valor for `MRI`, especifique "MRI" e "mir" como valores.
- Tipos de slot integrado – no momento, o Amazon Lex não oferece suporte à adição de valores de enumeração ou sinônimos para os tipos de slot integrado.

# API Reference

Esta seção fornece a documentação das operações da API do Amazon Lex. Para obter uma lista de regiões da AWS onde o Amazon Lex está disponível, consulte [Regiões e endpoints da AWS](#) na Referência geral da Amazon Web Services.

## Tópicos

- [Actions](#) (p. 216)
- [Data Types](#) (p. 365)

## Actions

The following actions are supported by Amazon Lex Model Building Service:

- [CreateBotVersion](#) (p. 219)
- [CreateIntentVersion](#) (p. 224)
- [CreateSlotTypeVersion](#) (p. 230)
- [DeleteBot](#) (p. 234)
- [DeleteBotAlias](#) (p. 236)
- [DeleteBotChannelAssociation](#) (p. 238)
- [DeleteBotVersion](#) (p. 240)
- [DeleteIntent](#) (p. 242)
- [DeleteIntentVersion](#) (p. 244)
- [DeleteSlotType](#) (p. 246)
- [DeleteSlotTypeVersion](#) (p. 248)
- [DeleteUtterances](#) (p. 250)
- [GetBot](#) (p. 252)
- [GetBotAlias](#) (p. 257)
- [GetBotAliases](#) (p. 260)
- [GetBotChannelAssociation](#) (p. 263)
- [GetBotChannelAssociations](#) (p. 267)
- [GetBots](#) (p. 270)
- [GetBotVersions](#) (p. 273)
- [GetBuiltinIntent](#) (p. 276)
- [GetBuiltinIntents](#) (p. 278)
- [GetBuiltinSlotTypes](#) (p. 280)
- [GetExport](#) (p. 282)
- [GetImport](#) (p. 285)
- [GetIntent](#) (p. 288)
- [GetIntents](#) (p. 293)
- [GetIntentVersions](#) (p. 296)
- [GetSlotType](#) (p. 299)
- [GetSlotTypes](#) (p. 302)

- [GetSlotTypeVersions](#) (p. 305)
- [GetUtterancesView](#) (p. 308)
- [PutBot](#) (p. 311)
- [PutBotAlias](#) (p. 319)
- [PutIntent](#) (p. 323)
- [PutSlotType](#) (p. 333)
- [StartImport](#) (p. 338)

The following actions are supported by Amazon Lex Runtime Service:

- [DeleteSession](#) (p. 341)
- [GetSession](#) (p. 344)
- [PostContent](#) (p. 347)
- [PostText](#) (p. 355)
- [PutSession](#) (p. 361)

## Amazon Lex Model Building Service

The following actions are supported by Amazon Lex Model Building Service:

- [CreateBotVersion](#) (p. 219)
- [CreateIntentVersion](#) (p. 224)
- [CreateSlotTypeVersion](#) (p. 230)
- [DeleteBot](#) (p. 234)
- [DeleteBotAlias](#) (p. 236)
- [DeleteBotChannelAssociation](#) (p. 238)
- [DeleteBotVersion](#) (p. 240)
- [DeleteIntent](#) (p. 242)
- [DeleteIntentVersion](#) (p. 244)
- [DeleteSlotType](#) (p. 246)
- [DeleteSlotTypeVersion](#) (p. 248)
- [DeleteUtterances](#) (p. 250)
- [GetBot](#) (p. 252)
- [GetBotAlias](#) (p. 257)
- [GetBotAliases](#) (p. 260)
- [GetBotChannelAssociation](#) (p. 263)
- [GetBotChannelAssociations](#) (p. 267)
- [GetBots](#) (p. 270)
- [GetBotVersions](#) (p. 273)
- [GetBuiltinIntent](#) (p. 276)
- [GetBuiltinIntents](#) (p. 278)
- [GetBuiltinSlotTypes](#) (p. 280)
- [GetExport](#) (p. 282)
- [GetImport](#) (p. 285)
- [GetIntent](#) (p. 288)
- [GetIntents](#) (p. 293)

- [GetIntentVersions](#) (p. 296)
- [GetSlotType](#) (p. 299)
- [GetSlotTypes](#) (p. 302)
- [GetSlotTypeVersions](#) (p. 305)
- [GetUtterancesView](#) (p. 308)
- [PutBot](#) (p. 311)
- [PutBotAlias](#) (p. 319)
- [PutIntent](#) (p. 323)
- [PutSlotType](#) (p. 333)
- [StartImport](#) (p. 338)

## CreateBotVersion

Service: Amazon Lex Model Building Service

Creates a new version of the bot based on the `$LATEST` version. If the `$LATEST` version of this resource hasn't changed since you created the last version, Amazon Lex doesn't create a new version. It returns the last created version.

### Note

You can update only the `$LATEST` version of the bot. You can't update the numbered versions that you create with the `CreateBotVersion` operation.

When you create the first version of a bot, Amazon Lex sets the version to 1. Subsequent versions increment by 1. For more information, see [Controle de versão \(p. 102\)](#).

This operation requires permission for the `lex:CreateBotVersion` action.

## Request Syntax

```
POST /bots/name/versions HTTP/1.1
Content-type: application/json

{
  "checksum": "string"
}
```

## URI Request Parameters

The request requires the following URI parameters.

[name \(p. 219\)](#)

The name of the bot that you want to create a new version of. The name is case sensitive.

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^[A-Za-z_?]+$`

## Request Body

The request accepts the following data in JSON format.

[checksum \(p. 219\)](#)

Identifies a specific revision of the `$LATEST` version of the bot. If you specify a checksum and the `$LATEST` version of the bot has a different checksum, a `PreconditionFailedException` exception is returned and Amazon Lex doesn't publish a new version. If you don't specify a checksum, Amazon Lex publishes the `$LATEST` version.

Type: String

Required: No

## Response Syntax

```
HTTP/1.1 201
Content-type: application/json
```



```
{
  "abortStatement": {
    "messages": [
      {
        "content": "string",
        "contentType": "string",
        "groupNumber": number
      }
    ],
    "responseCard": "string"
  },
  "checksum": "string",
  "childDirected": boolean,
  "clarificationPrompt": {
    "maxAttempts": number,
    "messages": [
      {
        "content": "string",
        "contentType": "string",
        "groupNumber": number
      }
    ],
    "responseCard": "string"
  },
  "createdDate": number,
  "description": "string",
  "failureReason": "string",
  "idleSessionTTLInSeconds": number,
  "intents": [
    {
      "intentName": "string",
      "intentVersion": "string"
    }
  ],
  "lastUpdatedDate": number,
  "locale": "string",
  "name": "string",
  "status": "string",
  "version": "string",
  "voiceId": "string"
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 201 response.

The following data is returned in JSON format by the service.

### [abortStatement \(p. 219\)](#)

The message that Amazon Lex uses to abort a conversation. For more information, see [PutBot \(p. 311\)](#).

Type: [Statement \(p. 390\)](#) object

### [checksum \(p. 219\)](#)

Checksum identifying the version of the bot that was created.

Type: String

### [childDirected \(p. 219\)](#)

For each Amazon Lex bot created with the Amazon Lex Model Building Service, you must specify whether your use of Amazon Lex is related to a website, program, or other application that is directed

or targeted, in whole or in part, to children under age 13 and subject to the Children's Online Privacy Protection Act (COPPA) by specifying `true` or `false` in the `childDirected` field. By specifying `true` in the `childDirected` field, you confirm that your use of Amazon Lex is related to a website, program, or other application that is directed or targeted, in whole or in part, to children under age 13 and subject to COPPA. By specifying `false` in the `childDirected` field, you confirm that your use of Amazon Lex is not related to a website, program, or other application that is directed or targeted, in whole or in part, to children under age 13 and subject to COPPA. You may not specify a default value for the `childDirected` field that does not accurately reflect whether your use of Amazon Lex is related to a website, program, or other application that is directed or targeted, in whole or in part, to children under age 13 and subject to COPPA.

If your use of Amazon Lex relates to a website, program, or other application that is directed in whole or in part, to children under age 13, you must obtain any required verifiable parental consent under COPPA. For information regarding the use of Amazon Lex in connection with websites, programs, or other applications that are directed or targeted, in whole or in part, to children under age 13, see the [Amazon Lex FAQ](#).

Type: Boolean

[clarificationPrompt](#) (p. 219)

The message that Amazon Lex uses when it doesn't understand the user's request. For more information, see [PutBot](#) (p. 311).

Type: [Prompt](#) (p. 384) object

[createdDate](#) (p. 219)

The date when the bot version was created.

Type: Timestamp

[description](#) (p. 219)

A description of the bot.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

[failureReason](#) (p. 219)

If `status` is `FAILED`, Amazon Lex provides the reason that it failed to build the bot.

Type: String

[idleSessionTTLInSeconds](#) (p. 219)

The maximum time in seconds that Amazon Lex retains the data gathered in a conversation. For more information, see [PutBot](#) (p. 311).

Type: Integer

Valid Range: Minimum value of 60. Maximum value of 86400.

[intents](#) (p. 219)

An array of `Intent` objects. For more information, see [PutBot](#) (p. 311).

Type: Array of [Intent](#) (p. 380) objects

[lastUpdatedDate](#) (p. 219)

The date when the `$LATEST` version of this bot was updated.

Type: Timestamp

#### [locale \(p. 219\)](#)

Specifies the target locale for the bot.

Type: String

Valid Values: en-US

#### [name \(p. 219\)](#)

The name of the bot.

Type: String

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: ^([A-Za-z]\_?)+\$

#### [status \(p. 219\)](#)

When you send a request to create or update a bot, Amazon Lex sets the `status` response element to `BUILDING`. After Amazon Lex builds the bot, it sets `status` to `READY`. If Amazon Lex can't build the bot, it sets `status` to `FAILED`. Amazon Lex returns the reason for the failure in the `failureReason` response element.

Type: String

Valid Values: BUILDING | READY | READY\_BASIC\_TESTING | FAILED | NOT\_BUILT

#### [version \(p. 219\)](#)

The version of the bot.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: \\$(LATEST|[0-9])+

#### [voiceId \(p. 219\)](#)

The Amazon Polly voice ID that Amazon Lex uses for voice interactions with the user.

Type: String

## Errors

### BadRequestException

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

### ConflictException

There was a conflict processing the request. Try your request again.

HTTP Status Code: 409

### InternalFailureException

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

#### LimitExceededException

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

#### NotFoundException

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

#### PreconditionFailedException

The checksum of the resource that you are trying to change does not match the checksum in the request. Check the resource's checksum and try again.

HTTP Status Code: 412

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## CreateIntentVersion

Service: Amazon Lex Model Building Service

Creates a new version of an intent based on the `$LATEST` version of the intent. If the `$LATEST` version of this intent hasn't changed since you last updated it, Amazon Lex doesn't create a new version. It returns the last version you created.

### Note

You can update only the `$LATEST` version of the intent. You can't update the numbered versions that you create with the `CreateIntentVersion` operation.

When you create a version of an intent, Amazon Lex sets the version to 1. Subsequent versions increment by 1. For more information, see [Controle de versão \(p. 102\)](#).

This operation requires permissions to perform the `lex:CreateIntentVersion` action.

## Request Syntax

```
POST /intents/name/versions HTTP/1.1
Content-type: application/json

{
  "checksum": "string"
}
```

## URI Request Parameters

The request requires the following URI parameters.

[name \(p. 224\)](#)

The name of the intent that you want to create a new version of. The name is case sensitive.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[A-Za-z_?]+$`

## Request Body

The request accepts the following data in JSON format.

[checksum \(p. 224\)](#)

Checksum of the `$LATEST` version of the intent that should be used to create the new version. If you specify a checksum and the `$LATEST` version of the intent has a different checksum, Amazon Lex returns a `PreconditionFailedException` exception and doesn't publish a new version. If you don't specify a checksum, Amazon Lex publishes the `$LATEST` version.

Type: String

Required: No

## Response Syntax

```
HTTP/1.1 201
```

Content-type: application/json

```
{
  "checksum": "string",
  "conclusionStatement": {
    "messages": [
      {
        "content": "string",
        "contentType": "string",
        "groupNumber": number
      }
    ],
    "responseCard": "string"
  },
  "confirmationPrompt": {
    "maxAttempts": number,
    "messages": [
      {
        "content": "string",
        "contentType": "string",
        "groupNumber": number
      }
    ],
    "responseCard": "string"
  },
  "createdAt": number,
  "description": "string",
  "dialogCodeHook": {
    "messageVersion": "string",
    "uri": "string"
  },
  "followUpPrompt": {
    "prompt": {
      "maxAttempts": number,
      "messages": [
        {
          "content": "string",
          "contentType": "string",
          "groupNumber": number
        }
      ],
      "responseCard": "string"
    },
    "rejectionStatement": {
      "messages": [
        {
          "content": "string",
          "contentType": "string",
          "groupNumber": number
        }
      ],
      "responseCard": "string"
    }
  },
  "fulfillmentActivity": {
    "codeHook": {
      "messageVersion": "string",
      "uri": "string"
    },
    "type": "string"
  },
  "lastUpdatedDate": number,
  "name": "string",
  "parentIntentSignature": "string",
  "rejectionStatement": {
    "messages": [

```

```
        {
          "content": "string",
          "contentType": "string",
          "groupNumber": number
        }
      ],
      "responseCard": "string"
    },
    "sampleUtterances": [ "string" ],
    "slots": [
      {
        "description": "string",
        "name": "string",
        "priority": number,
        "responseCard": "string",
        "sampleUtterances": [ "string" ],
        "slotConstraint": "string",
        "slotType": "string",
        "slotTypeVersion": "string",
        "valueElicitationPrompt": {
          "maxAttempts": number,
          "messages": [
            {
              "content": "string",
              "contentType": "string",
              "groupNumber": number
            }
          ],
          "responseCard": "string"
        }
      }
    ],
    "version": "string"
  }
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 201 response.

The following data is returned in JSON format by the service.

### [checksum \(p. 224\)](#)

Checksum of the intent version created.

Type: String

### [conclusionStatement \(p. 224\)](#)

After the Lambda function specified in the `fulfillmentActivity` field fulfills the intent, Amazon Lex conveys this statement to the user.

Type: [Statement \(p. 390\)](#) object

### [confirmationPrompt \(p. 224\)](#)

If defined, the prompt that Amazon Lex uses to confirm the user's intent before fulfilling it.

Type: [Prompt \(p. 384\)](#) object

### [createdDate \(p. 224\)](#)

The date that the intent was created.

Type: Timestamp

[description \(p. 224\)](#)

A description of the intent.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

[dialogCodeHook \(p. 224\)](#)

If defined, Amazon Lex invokes this Lambda function for each user input.

Type: [CodeHook \(p. 376\)](#) object

[followUpPrompt \(p. 224\)](#)

If defined, Amazon Lex uses this prompt to solicit additional user activity after the intent is fulfilled.

Type: [FollowUpPrompt \(p. 378\)](#) object

[fulfillmentActivity \(p. 224\)](#)

Describes how the intent is fulfilled.

Type: [FulfillmentActivity \(p. 379\)](#) object

[lastUpdatedDate \(p. 224\)](#)

The date that the intent was updated.

Type: Timestamp

[name \(p. 224\)](#)

The name of the intent.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[A-Za-z_?]+$`

[parentIntentSignature \(p. 224\)](#)

A unique identifier for a built-in intent.

Type: String

[rejectionStatement \(p. 224\)](#)

If the user answers "no" to the question defined in `confirmationPrompt`, Amazon Lex responds with this statement to acknowledge that the intent was canceled.

Type: [Statement \(p. 390\)](#) object

[sampleUtterances \(p. 224\)](#)

An array of sample utterances configured for the intent.

Type: Array of strings

Array Members: Minimum number of 0 items. Maximum number of 1500 items.

Length Constraints: Minimum length of 1. Maximum length of 200.

[slots \(p. 224\)](#)

An array of slot types that defines the information required to fulfill the intent.



Type: Array of [Slot \(p. 386\)](#) objects

Array Members: Minimum number of 0 items. Maximum number of 100 items.  
[version \(p. 224\)](#)

The version number assigned to the new version of the intent.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `\$LATEST|[0-9]+`

## Errors

### BadRequestException

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

### ConflictException

There was a conflict processing the request. Try your request again.

HTTP Status Code: 409

### InternalFailureException

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

### LimitExceededException

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

### NotFoundException

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

### PreconditionFailedException

The checksum of the resource that you are trying to change does not match the checksum in the request. Check the resource's checksum and try again.

HTTP Status Code: 412

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)

- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## CreateSlotTypeVersion

Service: Amazon Lex Model Building Service

Creates a new version of a slot type based on the `$LATEST` version of the specified slot type. If the `$LATEST` version of this resource has not changed since the last version that you created, Amazon Lex doesn't create a new version. It returns the last version that you created.

### Note

You can update only the `$LATEST` version of a slot type. You can't update the numbered versions that you create with the `CreateSlotTypeVersion` operation.

When you create a version of a slot type, Amazon Lex sets the version to 1. Subsequent versions increment by 1. For more information, see [Controle de versão \(p. 102\)](#).

This operation requires permissions for the `lex:CreateSlotTypeVersion` action.

## Request Syntax

```
POST /slottypes/name/versions HTTP/1.1
Content-type: application/json

{
  "checksum": "string"
}
```

## URI Request Parameters

The request requires the following URI parameters.

[name \(p. 230\)](#)

The name of the slot type that you want to create a new version for. The name is case sensitive.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^([A-Za-z]_?)+$`

## Request Body

The request accepts the following data in JSON format.

[checksum \(p. 230\)](#)

Checksum for the `$LATEST` version of the slot type that you want to publish. If you specify a checksum and the `$LATEST` version of the slot type has a different checksum, Amazon Lex returns a `PreconditionFailedException` exception and doesn't publish the new version. If you don't specify a checksum, Amazon Lex publishes the `$LATEST` version.

Type: String

Required: No

## Response Syntax

```
HTTP/1.1 201
Content-type: application/json
```

```
{
  "checksum": "string",
  "createdDate": number,
  "description": "string",
  "enumerationValues": [
    {
      "synonyms": [ "string" ],
      "value": "string"
    }
  ],
  "lastUpdatedDate": number,
  "name": "string",
  "valueSelectionStrategy": "string",
  "version": "string"
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 201 response.

The following data is returned in JSON format by the service.

### [checksum \(p. 230\)](#)

Checksum of the `$LATEST` version of the slot type.

Type: String

### [createdDate \(p. 230\)](#)

The date that the slot type was created.

Type: Timestamp

### [description \(p. 230\)](#)

A description of the slot type.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

### [enumerationValues \(p. 230\)](#)

A list of `EnumerationValue` objects that defines the values that the slot type can take.

Type: Array of [EnumerationValue \(p. 377\)](#) objects

Array Members: Minimum number of 1 item. Maximum number of 10000 items.

### [lastUpdatedDate \(p. 230\)](#)

The date that the slot type was updated. When you create a resource, the creation date and last update date are the same.

Type: Timestamp

### [name \(p. 230\)](#)

The name of the slot type.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[A-Za-z_?]+$`

[valueSelectionStrategy \(p. 230\)](#)

The strategy that Amazon Lex uses to determine the value of the slot. For more information, see [PutSlotType \(p. 333\)](#).

Type: String

Valid Values: `ORIGINAL_VALUE` | `TOP_RESOLUTION`

[version \(p. 230\)](#)

The version assigned to the new slot type version.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `\$LATEST|[0-9]+`

## Errors

### BadRequestException

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

### ConflictException

There was a conflict processing the request. Try your request again.

HTTP Status Code: 409

### InternalFailureException

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

### LimitExceededException

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

### NotFoundException

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

### PreconditionFailedException

The checksum of the resource that you are trying to change does not match the checksum in the request. Check the resource's checksum and try again.

HTTP Status Code: 412

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## DeleteBot

Service: Amazon Lex Model Building Service

Deletes all versions of the bot, including the `$LATEST` version. To delete a specific version of the bot, use the [DeleteBotVersion \(p. 240\)](#) operation. The `DeleteBot` operation doesn't immediately remove the bot schema. Instead, it is marked for deletion and removed later.

Amazon Lex stores utterances indefinitely for improving the ability of your bot to respond to user inputs. These utterances are not removed when the bot is deleted. To remove the utterances, use the [DeleteUtterances \(p. 250\)](#) operation.

If a bot has an alias, you can't delete it. Instead, the `DeleteBot` operation returns a `ResourceInUseException` exception that includes a reference to the alias that refers to the bot. To remove the reference to the bot, delete the alias. If you get the same exception again, delete the referring alias until the `DeleteBot` operation is successful.

This operation requires permissions for the `lex:DeleteBot` action.

## Request Syntax

```
DELETE /bots/name HTTP/1.1
```

## URI Request Parameters

The request requires the following URI parameters.

[name \(p. 234\)](#)

The name of the bot. The name is case sensitive.

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^[A-Za-z_?]+$`

## Request Body

The request does not have a request body.

## Response Syntax

```
HTTP/1.1 204
```

## Response Elements

If the action is successful, the service sends back an HTTP 204 response with an empty HTTP body.

## Errors

**BadRequestException**

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

**ConflictException**

There was a conflict processing the request. Try your request again.

HTTP Status Code: 409

InternalFailureException

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

LimitExceededException

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

NotFoundException

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

ResourceInUseException

The resource that you are attempting to delete is referred to by another resource. Use this information to remove references to the resource that you are trying to delete.

The body of the exception contains a JSON object that describes the resource.

```
{ "resourceType": BOT | BOTALIAS | BOTCHANNEL | INTENT,  
  "resourceReference": {  
    "name": string, "version": string } }
```

HTTP Status Code: 400

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)



## DeleteBotAlias

Service: Amazon Lex Model Building Service

Deletes an alias for the specified bot.

You can't delete an alias that is used in the association between a bot and a messaging channel. If an alias is used in a channel association, the `DeleteBot` operation returns a `ResourceInUseException` exception that includes a reference to the channel association that refers to the bot. You can remove the reference to the alias by deleting the channel association. If you get the same exception again, delete the referring association until the `DeleteBotAlias` operation is successful.

### Request Syntax

```
DELETE /bots/botName/aliases/name HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

**botName** (p. 236)

The name of the bot that the alias points to.

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^[A-Za-z_?]+$`

**name** (p. 236)

The name of the alias to delete. The name is case sensitive.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[A-Za-z_?]+$`

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 204
```

### Response Elements

If the action is successful, the service sends back an HTTP 204 response with an empty HTTP body.

### Errors

#### BadRequestException

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

#### ConflictException

There was a conflict processing the request. Try your request again.

HTTP Status Code: 409

InternalFailureException

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

LimitExceededException

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

NotFoundException

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

ResourceInUseException

The resource that you are attempting to delete is referred to by another resource. Use this information to remove references to the resource that you are trying to delete.

The body of the exception contains a JSON object that describes the resource.

```
{ "resourceType": BOT | BOTALIAS | BOTCHANNEL | INTENT,  
  "resourceReference": {  
    "name": string, "version": string } }
```

HTTP Status Code: 400

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## DeleteBotChannelAssociation

Service: Amazon Lex Model Building Service

Deletes the association between an Amazon Lex bot and a messaging platform.

This operation requires permission for the `lex:DeleteBotChannelAssociation` action.

### Request Syntax

```
DELETE /bots/botName/aliases/aliasName/channels/name HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

[aliasName \(p. 238\)](#)

An alias that points to the specific version of the Amazon Lex bot to which this association is being made.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[A-Za-z_?]+$`

[botName \(p. 238\)](#)

The name of the Amazon Lex bot.

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^[A-Za-z_?]+$`

[name \(p. 238\)](#)

The name of the association. The name is case sensitive.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[A-Za-z_?]+$`

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 204
```

### Response Elements

If the action is successful, the service sends back an HTTP 204 response with an empty HTTP body.

### Errors

`BadRequestException`

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

ConflictException

There was a conflict processing the request. Try your request again.

HTTP Status Code: 409

InternalFailureException

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

LimitExceededException

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

NotFoundException

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## DeleteBotVersion

Service: Amazon Lex Model Building Service

Deletes a specific version of a bot. To delete all versions of a bot, use the [DeleteBot \(p. 234\)](#) operation.

This operation requires permissions for the `lex:DeleteBotVersion` action.

### Request Syntax

```
DELETE /bots/name/versions/version HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

**name** (p. 240)

The name of the bot.

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^[A-Za-z_?]+$`

**version** (p. 240)

The version of the bot to delete. You cannot delete the `$LATEST` version of the bot. To delete the `$LATEST` version, use the [DeleteBot \(p. 234\)](#) operation.

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `[0-9]+`

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 204
```

### Response Elements

If the action is successful, the service sends back an HTTP 204 response with an empty HTTP body.

### Errors

#### BadRequestException

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

#### ConflictException

There was a conflict processing the request. Try your request again.

HTTP Status Code: 409

#### InternalFailureException

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

#### LimitExceededException

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

#### NotFoundException

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

#### ResourceInUseException

The resource that you are attempting to delete is referred to by another resource. Use this information to remove references to the resource that you are trying to delete.

The body of the exception contains a JSON object that describes the resource.

```
{ "resourceType": BOT | BOTALIAS | BOTCHANNEL | INTENT,  
  "resourceReference": {  
    "name": string, "version": string } }
```

HTTP Status Code: 400

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## DeleteIntent

Service: Amazon Lex Model Building Service

Deletes all versions of the intent, including the `$LATEST` version. To delete a specific version of the intent, use the [DeleteIntentVersion](#) (p. 244) operation.

You can delete a version of an intent only if it is not referenced. To delete an intent that is referred to in one or more bots (see [Amazon Lex: Como ele funciona](#) (p. 3)), you must remove those references first.

### Note

If you get the `ResourceInUseException` exception, it provides an example reference that shows where the intent is referenced. To remove the reference to the intent, either update the bot or delete it. If you get the same exception when you attempt to delete the intent again, repeat until the intent has no references and the call to `DeleteIntent` is successful.

This operation requires permission for the `lex:DeleteIntent` action.

## Request Syntax

```
DELETE /intents/name HTTP/1.1
```

## URI Request Parameters

The request requires the following URI parameters.

[name](#) (p. 242)

The name of the intent. The name is case sensitive.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[A-Za-z_?]+$`

## Request Body

The request does not have a request body.

## Response Syntax

```
HTTP/1.1 204
```

## Response Elements

If the action is successful, the service sends back an HTTP 204 response with an empty HTTP body.

## Errors

### BadRequestException

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

### ConflictException

There was a conflict processing the request. Try your request again.

HTTP Status Code: 409

InternalFailureException

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

LimitExceededException

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

NotFoundException

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

ResourceInUseException

The resource that you are attempting to delete is referred to by another resource. Use this information to remove references to the resource that you are trying to delete.

The body of the exception contains a JSON object that describes the resource.

```
{ "resourceType": BOT | BOTALIAS | BOTCHANNEL | INTENT,  
  "resourceReference": {  
    "name": string, "version": string } }
```

HTTP Status Code: 400

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)



## DeleteIntentVersion

Service: Amazon Lex Model Building Service

Deletes a specific version of an intent. To delete all versions of a intent, use the [DeleteIntent](#) (p. 242) operation.

This operation requires permissions for the `lex:DeleteIntentVersion` action.

### Request Syntax

```
DELETE /intents/name/versions/version HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

[name](#) (p. 244)

The name of the intent.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[A-Za-z_?]+$`

[version](#) (p. 244)

The version of the intent to delete. You cannot delete the `$LATEST` version of the intent. To delete the `$LATEST` version, use the [DeleteIntent](#) (p. 242) operation.

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `[0-9]+`

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 204
```

### Response Elements

If the action is successful, the service sends back an HTTP 204 response with an empty HTTP body.

### Errors

**BadRequestException**

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

**ConflictException**

There was a conflict processing the request. Try your request again.

HTTP Status Code: 409

InternalFailureException

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

LimitExceededException

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

NotFoundException

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

ResourceInUseException

The resource that you are attempting to delete is referred to by another resource. Use this information to remove references to the resource that you are trying to delete.

The body of the exception contains a JSON object that describes the resource.

```
{ "resourceType": BOT | BOTALIAS | BOTCHANNEL | INTENT,  
  "resourceReference": {  
    "name": string, "version": string } }
```

HTTP Status Code: 400

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## DeleteSlotType

Service: Amazon Lex Model Building Service

Deletes all versions of the slot type, including the `$LATEST` version. To delete a specific version of the slot type, use the [DeleteSlotTypeVersion](#) (p. 248) operation.

You can delete a version of a slot type only if it is not referenced. To delete a slot type that is referred to in one or more intents, you must remove those references first.

### Note

If you get the `ResourceInUseException` exception, the exception provides an example reference that shows the intent where the slot type is referenced. To remove the reference to the slot type, either update the intent or delete it. If you get the same exception when you attempt to delete the slot type again, repeat until the slot type has no references and the `DeleteSlotType` call is successful.

This operation requires permission for the `lex:DeleteSlotType` action.

## Request Syntax

```
DELETE /slottypes/name HTTP/1.1
```

## URI Request Parameters

The request requires the following URI parameters.

[name](#) (p. 246)

The name of the slot type. The name is case sensitive.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^([A-Za-z]_?)+$`

## Request Body

The request does not have a request body.

## Response Syntax

```
HTTP/1.1 204
```

## Response Elements

If the action is successful, the service sends back an HTTP 204 response with an empty HTTP body.

## Errors

### BadRequestException

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

#### ConflictException

There was a conflict processing the request. Try your request again.

HTTP Status Code: 409

#### InternalFailureException

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

#### LimitExceededException

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

#### NotFoundException

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

#### ResourceInUseException

The resource that you are attempting to delete is referred to by another resource. Use this information to remove references to the resource that you are trying to delete.

The body of the exception contains a JSON object that describes the resource.

```
{ "resourceType": BOT | BOTALIAS | BOTCHANNEL | INTENT,  
  "resourceReference": {  
    "name": string, "version": string } }
```

HTTP Status Code: 400

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## DeleteSlotTypeVersion

Service: Amazon Lex Model Building Service

Deletes a specific version of a slot type. To delete all versions of a slot type, use the [DeleteSlotType \(p. 246\)](#) operation.

This operation requires permissions for the `lex:DeleteSlotTypeVersion` action.

### Request Syntax

```
DELETE /slottypes/name/version/version HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

[name \(p. 248\)](#)

The name of the slot type.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[A-Za-z_?]+$`

[version \(p. 248\)](#)

The version of the slot type to delete. You cannot delete the `$LATEST` version of the slot type. To delete the `$LATEST` version, use the [DeleteSlotType \(p. 246\)](#) operation.

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `[0-9]+`

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 204
```

### Response Elements

If the action is successful, the service sends back an HTTP 204 response with an empty HTTP body.

### Errors

**BadRequestException**

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

**ConflictException**

There was a conflict processing the request. Try your request again.

HTTP Status Code: 409

InternalFailureException

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

LimitExceededException

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

NotFoundException

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

ResourceInUseException

The resource that you are attempting to delete is referred to by another resource. Use this information to remove references to the resource that you are trying to delete.

The body of the exception contains a JSON object that describes the resource.

```
{ "resourceType": BOT | BOTALIAS | BOTCHANNEL | INTENT,  
  "resourceReference": {  
    "name": string, "version": string } }
```

HTTP Status Code: 400

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## DeleteUtterances

Service: Amazon Lex Model Building Service

Deletes stored utterances.

Amazon Lex stores the utterances that users send to your bot. Utterances are stored for 15 days for use with the [GetUtterancesView](#) (p. 308) operation, and then stored indefinitely for use in improving the ability of your bot to respond to user input.

Use the `DeleteUtterances` operation to manually delete stored utterances for a specific user. When you use the `DeleteUtterances` operation, utterances stored for improving your bot's ability to respond to user input are deleted immediately. Utterances stored for use with the `GetUtterancesView` operation are deleted after 15 days.

This operation requires permissions for the `lex:DeleteUtterances` action.

### Request Syntax

```
DELETE /bots/botName/utterances/userId HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

[botName](#) (p. 250)

The name of the bot that stored the utterances.

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^[A-Za-z_?]+$`

[userId](#) (p. 250)

The unique identifier for the user that made the utterances. This is the user ID that was sent in the [PostContent](#) or [PostText](#) operation request that contained the utterance.

Length Constraints: Minimum length of 2. Maximum length of 100.

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 204
```

### Response Elements

If the action is successful, the service sends back an HTTP 204 response with an empty HTTP body.

### Errors

`BadRequestException`

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

InternalFailureException

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

LimitExceededException

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

NotFoundException

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)



## GetBot

Service: Amazon Lex Model Building Service

Returns metadata information for a specific bot. You must provide the bot name and the bot version or alias.

This operation requires permissions for the `lex:GetBot` action.

### Request Syntax

```
GET /bots/name/versions/versionoralias HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

`name` (p. 252)

The name of the bot. The name is case sensitive.

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^[A-Za-z_?]+$`

`versionoralias` (p. 252)

The version or alias of the bot.

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "abortStatement": {
    "messages": [
      {
        "content": "string",
        "contentType": "string",
        "groupNumber": number
      }
    ],
    "responseCard": "string"
  },
  "checksum": "string",
  "childDirected": boolean,
  "clarificationPrompt": {
    "maxAttempts": number,
    "messages": [
      {
        "content": "string",
        "contentType": "string",
        "groupNumber": number
      }
    ],
    "responseCard": "string"
  },
}
```

```
"createdDate": number,
"description": "string",
"failureReason": "string",
"idleSessionTTLInSeconds": number,
"intents": [
  {
    "intentName": "string",
    "intentVersion": "string"
  }
],
"lastUpdatedDate": number,
"locale": "string",
"name": "string",
"status": "string",
"version": "string",
"voiceId": "string"
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

### [abortStatement \(p. 252\)](#)

The message that Amazon Lex returns when the user elects to end the conversation without completing it. For more information, see [PutBot \(p. 311\)](#).

Type: [Statement \(p. 390\)](#) object

### [checksum \(p. 252\)](#)

Checksum of the bot used to identify a specific revision of the bot's `$LATEST` version.

Type: String

### [childDirected \(p. 252\)](#)

For each Amazon Lex bot created with the Amazon Lex Model Building Service, you must specify whether your use of Amazon Lex is related to a website, program, or other application that is directed or targeted, in whole or in part, to children under age 13 and subject to the Children's Online Privacy Protection Act (COPPA) by specifying `true` or `false` in the `childDirected` field. By specifying `true` in the `childDirected` field, you confirm that your use of Amazon Lex is related to a website, program, or other application that is directed or targeted, in whole or in part, to children under age 13 and subject to COPPA. By specifying `false` in the `childDirected` field, you confirm that your use of Amazon Lex is not related to a website, program, or other application that is directed or targeted, in whole or in part, to children under age 13 and subject to COPPA. You may not specify a default value for the `childDirected` field that does not accurately reflect whether your use of Amazon Lex is related to a website, program, or other application that is directed or targeted, in whole or in part, to children under age 13 and subject to COPPA.

If your use of Amazon Lex relates to a website, program, or other application that is directed in whole or in part, to children under age 13, you must obtain any required verifiable parental consent under COPPA. For information regarding the use of Amazon Lex in connection with websites, programs, or other applications that are directed or targeted, in whole or in part, to children under age 13, see the [Amazon Lex FAQ](#).

Type: Boolean

### [clarificationPrompt \(p. 252\)](#)

The message Amazon Lex uses when it doesn't understand the user's request. For more information, see [PutBot \(p. 311\)](#).

Type: [Prompt \(p. 384\)](#) object

[createdDate \(p. 252\)](#)

The date that the bot was created.

Type: Timestamp

[description \(p. 252\)](#)

A description of the bot.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

[failureReason \(p. 252\)](#)

If `status` is `FAILED`, Amazon Lex explains why it failed to build the bot.

Type: String

[idleSessionTTLInSeconds \(p. 252\)](#)

The maximum time in seconds that Amazon Lex retains the data gathered in a conversation. For more information, see [PutBot \(p. 311\)](#).

Type: Integer

Valid Range: Minimum value of 60. Maximum value of 86400.

[intents \(p. 252\)](#)

An array of `intent` objects. For more information, see [PutBot \(p. 311\)](#).

Type: Array of [Intent \(p. 380\)](#) objects

[lastUpdatedDate \(p. 252\)](#)

The date that the bot was updated. When you create a resource, the creation date and last updated date are the same.

Type: Timestamp

[locale \(p. 252\)](#)

The target locale for the bot.

Type: String

Valid Values: `en-US`

[name \(p. 252\)](#)

The name of the bot.

Type: String

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^[A-Za-z_?]+$`

[status \(p. 252\)](#)

The status of the bot.

When the status is `BUILDING` Amazon Lex is building the bot for testing and use.

If the status of the bot is `READY_BASIC_TESTING`, you can test the bot using the exact utterances specified in the bot's intents. When the bot is ready for full testing or to run, the status is `READY`.

If there was a problem with building the bot, the status is `FAILED` and the `failureReason` field explains why the bot did not build.

If the bot was saved but not built, the status is `NOT_BUILT`.

Type: String

Valid Values: `BUILDING` | `READY` | `READY_BASIC_TESTING` | `FAILED` | `NOT_BUILT`

[version \(p. 252\)](#)

The version of the bot. For a new bot, the version is always `$LATEST`.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `\$LATEST|[0-9]+`

[voiceId \(p. 252\)](#)

The Amazon Polly voice ID that Amazon Lex uses for voice interaction with the user. For more information, see [PutBot \(p. 311\)](#).

Type: String

## Errors

### BadRequestException

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

### InternalFailureException

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

### LimitExceededException

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

### NotFoundException

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## GetBotAlias

Service: Amazon Lex Model Building Service

Returns information about an Amazon Lex bot alias. For more information about aliases, see [Controle de versão e aliases \(p. 102\)](#).

This operation requires permissions for the `lex:GetBotAlias` action.

### Request Syntax

```
GET /bots/botName/aliases/name HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

[botName \(p. 257\)](#)

The name of the bot.

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^[A-Za-z_?]+$`

[name \(p. 257\)](#)

The name of the bot alias. The name is case sensitive.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[A-Za-z_?]+$`

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "botName": "string",
  "botVersion": "string",
  "checksum": "string",
  "createdDate": number,
  "description": "string",
  "lastUpdatedDate": number,
  "name": "string"
}
```

### Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[botName \(p. 257\)](#)

The name of the bot that the alias points to.

Type: String

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^[A-Za-z_?]+$`

[botVersion \(p. 257\)](#)

The version of the bot that the alias points to.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `\$LATEST|[0-9]+`

[checksum \(p. 257\)](#)

Checksum of the bot alias.

Type: String

[createdDate \(p. 257\)](#)

The date that the bot alias was created.

Type: Timestamp

[description \(p. 257\)](#)

A description of the bot alias.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

[lastUpdatedDate \(p. 257\)](#)

The date that the bot alias was updated. When you create a resource, the creation date and the last updated date are the same.

Type: Timestamp

[name \(p. 257\)](#)

The name of the bot alias.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[A-Za-z_?]+$`

## Errors

### BadRequestException

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

### InternalFailureException

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

LimitExceededException

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

NotFoundException

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)



## GetBotAliases

Service: Amazon Lex Model Building Service

Returns a list of aliases for a specified Amazon Lex bot.

This operation requires permissions for the `lex:GetBotAliases` action.

### Request Syntax

```
GET /bots/botName/aliases/?  
maxResults=maxResults&nameContains=nameContains&nextToken=nextToken HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

#### `botName` (p. 260)

The name of the bot.

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^([A-Za-z_?])+`

#### `maxResults` (p. 260)

The maximum number of aliases to return in the response. The default is 50. .

Valid Range: Minimum value of 1. Maximum value of 50.

#### `nameContains` (p. 260)

Substring to match in bot alias names. An alias will be returned if any part of its name matches the substring. For example, "xyz" matches both "xyzabc" and "abcxyz."

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^([A-Za-z_?])+`

#### `nextToken` (p. 260)

A pagination token for fetching the next page of aliases. If the response to this call is truncated, Amazon Lex returns a pagination token in the response. To fetch the next page of aliases, specify the pagination token in the next request.

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 200  
Content-type: application/json  
  
{  
  "BotAliases": [  
    {  
      "botName": "string",  
      "botVersion": "string",  
      "checksum": "string",
```

```
        "createdDate": number,  
        "description": "string",  
        "lastUpdatedDate": number,  
        "name": "string"  
    }  
  ],  
  "nextToken": "string"  
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

### [BotAliases \(p. 260\)](#)

An array of `BotAliasMetadata` objects, each describing a bot alias.

Type: Array of [BotAliasMetadata \(p. 367\)](#) objects

### [nextToken \(p. 260\)](#)

A pagination token for fetching next page of aliases. If the response to this call is truncated, Amazon Lex returns a pagination token in the response. To fetch the next page of aliases, specify the pagination token in the next request.

Type: String

## Errors

### `BadRequestException`

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

### `InternalFailureException`

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

### `LimitExceededException`

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)

- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## GetBotChannelAssociation

Service: Amazon Lex Model Building Service

Returns information about the association between an Amazon Lex bot and a messaging platform.

This operation requires permissions for the `lex:GetBotChannelAssociation` action.

### Request Syntax

```
GET /bots/botName/aliases/aliasName/channels/name HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

[aliasName](#) (p. 263)

An alias pointing to the specific version of the Amazon Lex bot to which this association is being made.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[A-Za-z_?]+$`

[botName](#) (p. 263)

The name of the Amazon Lex bot.

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^[A-Za-z_?]+$`

[name](#) (p. 263)

The name of the association between the bot and the channel. The name is case sensitive.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[A-Za-z_?]+$`

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "botAlias": "string",
  "botConfiguration": {
    "string": "string"
  },
  "botName": "string",
  "createdDate": number,
  "description": "string",
  "failureReason": "string",
  "name": "string",
  "status": "string",
  "type": "string"
```

```
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

### [botAlias \(p. 263\)](#)

An alias pointing to the specific version of the Amazon Lex bot to which this association is being made.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[A-Za-z_?]+$`

### [botConfiguration \(p. 263\)](#)

Provides information that the messaging platform needs to communicate with the Amazon Lex bot.

Type: String to string map

### [botName \(p. 263\)](#)

The name of the Amazon Lex bot.

Type: String

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^[A-Za-z_?]+$`

### [createdDate \(p. 263\)](#)

The date that the association between the bot and the channel was created.

Type: Timestamp

### [description \(p. 263\)](#)

A description of the association between the bot and the channel.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

### [failureReason \(p. 263\)](#)

If `status` is `FAILED`, Amazon Lex provides the reason that it failed to create the association.

Type: String

### [name \(p. 263\)](#)

The name of the association between the bot and the channel.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[A-Za-z_?]+$`

### [status \(p. 263\)](#)

The status of the bot channel.

- **CREATED** - The channel has been created and is ready for use.
- **IN\_PROGRESS** - Channel creation is in progress.
- **FAILED** - There was an error creating the channel. For information about the reason for the failure, see the `failureReason` field.

Type: String

Valid Values: `IN_PROGRESS` | `CREATED` | `FAILED`

[type \(p. 263\)](#)

The type of the messaging platform.

Type: String

Valid Values: `Facebook` | `Slack` | `Twilio-Sms` | `Kik`

## Errors

### BadRequestException

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

### InternalFailureException

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

### LimitExceededException

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

### NotFoundException

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)



## GetBotChannelAssociations

Service: Amazon Lex Model Building Service

Returns a list of all of the channels associated with the specified bot.

The GetBotChannelAssociations operation requires permissions for the `lex:GetBotChannelAssociations` action.

### Request Syntax

```
GET /bots/botName/aliases/aliasName/channels/?  
maxResults=maxResults&nameContains=nameContains&nextToken=nextToken HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

[aliasName](#) (p. 267)

An alias pointing to the specific version of the Amazon Lex bot to which this association is being made.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^(-|^[A-Za-z]_?)+$`

[botName](#) (p. 267)

The name of the Amazon Lex bot in the association.

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^[A-Za-z]_?)+$`

[maxResults](#) (p. 267)

The maximum number of associations to return in the response. The default is 50.

Valid Range: Minimum value of 1. Maximum value of 50.

[nameContains](#) (p. 267)

Substring to match in channel association names. An association will be returned if any part of its name matches the substring. For example, "xyz" matches both "xyzabc" and "abcxyz." To return all bot channel associations, use a hyphen ("-") as the `nameContains` parameter.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[A-Za-z]_?)+$`

[nextToken](#) (p. 267)

A pagination token for fetching the next page of associations. If the response to this call is truncated, Amazon Lex returns a pagination token in the response. To fetch the next page of associations, specify the pagination token in the next request.

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 200
```



Content-type: application/json

```
{
  "botChannelAssociations": [
    {
      "botAlias": "string",
      "botConfiguration": {
        "string": "string"
      },
      "botName": "string",
      "createdDate": number,
      "description": "string",
      "failureReason": "string",
      "name": "string",
      "status": "string",
      "type": "string"
    }
  ],
  "nextToken": "string"
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

### [botChannelAssociations \(p. 267\)](#)

An array of objects, one for each association, that provides information about the Amazon Lex bot and its association with the channel.

Type: Array of [BotChannelAssociation \(p. 369\)](#) objects

### [nextToken \(p. 267\)](#)

A pagination token that fetches the next page of associations. If the response to this call is truncated, Amazon Lex returns a pagination token in the response. To fetch the next page of associations, specify the pagination token in the next request.

Type: String

## Errors

### BadRequestException

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

### InternalFailureException

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

### LimitExceededException

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## GetBots

Service: Amazon Lex Model Building Service

Returns bot information as follows:

- If you provide the `nameContains` field, the response includes information for the `$LATEST` version of all bots whose name contains the specified string.
- If you don't specify the `nameContains` field, the operation returns information about the `$LATEST` version of all of your bots.

This operation requires permission for the `lex:GetBots` action.

## Request Syntax

```
GET /bots/?maxResults=maxResults&nameContains=nameContains&nextToken=nextToken HTTP/1.1
```

## URI Request Parameters

The request requires the following URI parameters.

### `maxResults` (p. 270)

The maximum number of bots to return in the response that the request will return. The default is 10.

Valid Range: Minimum value of 1. Maximum value of 50.

### `nameContains` (p. 270)

Substring to match in bot names. A bot will be returned if any part of its name matches the substring. For example, "xyz" matches both "xyzabc" and "abcxyz."

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^([A-Za-z]_?)+$`

### `nextToken` (p. 270)

A pagination token that fetches the next page of bots. If the response to this call is truncated, Amazon Lex returns a pagination token in the response. To fetch the next page of bots, specify the pagination token in the next request.

## Request Body

The request does not have a request body.

## Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "bots": [
    {
      "createdDate": number,
      "description": "string",
      "lastUpdatedDate": number,
      "name": "string",
      "status": "string",
```

```
    "version": "string"
  },
  "nextToken": "string"
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

### [bots \(p. 270\)](#)

An array of `botMetadata` objects, with one entry for each bot.

Type: Array of [BotMetadata \(p. 371\)](#) objects

### [nextToken \(p. 270\)](#)

If the response is truncated, it includes a pagination token that you can specify in your next request to fetch the next page of bots.

Type: String

## Errors

### `BadRequestException`

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

### `InternalFailureException`

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

### `LimitExceededException`

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

### `NotFoundException`

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)

- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## GetBotVersions

Service: Amazon Lex Model Building Service

Gets information about all of the versions of a bot.

The `GetBotVersions` operation returns a `BotMetadata` object for each version of a bot. For example, if a bot has three numbered versions, the `GetBotVersions` operation returns four `BotMetadata` objects in the response, one for each numbered version and one for the `$LATEST` version.

The `GetBotVersions` operation always returns at least one version, the `$LATEST` version.

This operation requires permissions for the `lex:GetBotVersions` action.

## Request Syntax

```
GET /bots/name/versions/?maxResults=maxResults&nextToken=nextToken HTTP/1.1
```

## URI Request Parameters

The request requires the following URI parameters.

[maxResults](#) (p. 273)

The maximum number of bot versions to return in the response. The default is 10.

Valid Range: Minimum value of 1. Maximum value of 50.

[name](#) (p. 273)

The name of the bot for which versions should be returned.

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^[A-Za-z_?]+$`

[nextToken](#) (p. 273)

A pagination token for fetching the next page of bot versions. If the response to this call is truncated, Amazon Lex returns a pagination token in the response. To fetch the next page of versions, specify the pagination token in the next request.

## Request Body

The request does not have a request body.

## Response Syntax

```
HTTP/1.1 200
Content-type: application/json
```

```
{
  "bots": [
    {
      "createdDate": number,
      "description": "string",
      "lastUpdatedDate": number,
      "name": "string",
      "status": "string",
      "version": "string"
    }
  ]
}
```

```
  ],  
  "nextToken": "string"  
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

### [bots \(p. 273\)](#)

An array of `BotMetadata` objects, one for each numbered version of the bot plus one for the `$LATEST` version.

Type: Array of [BotMetadata \(p. 371\)](#) objects

### [nextToken \(p. 273\)](#)

A pagination token for fetching the next page of bot versions. If the response to this call is truncated, Amazon Lex returns a pagination token in the response. To fetch the next page of versions, specify the pagination token in the next request.

Type: String

## Errors

### `BadRequestException`

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

### `InternalFailureException`

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

### `LimitExceededException`

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

### `NotFoundException`

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)

- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)



## GetBuiltinIntent

Service: Amazon Lex Model Building Service

Returns information about a built-in intent.

This operation requires permission for the `lex:GetBuiltinIntent` action.

### Request Syntax

```
GET /builtins/intents/signature HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

[signature \(p. 276\)](#)

The unique identifier for a built-in intent. To find the signature for an intent, see [Standard Built-in Intents](#) in the Alexa Skills Kit.

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "signature": "string",
  "slots": [
    {
      "name": "string"
    }
  ],
  "supportedLocales": [ "string" ]
}
```

### Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[signature \(p. 276\)](#)

The unique identifier for a built-in intent.

Type: String

[slots \(p. 276\)](#)

An array of `BuiltinIntentSlot` objects, one entry for each slot type in the intent.

Type: Array of [BuiltinIntentSlot \(p. 374\)](#) objects

[supportedLocales \(p. 276\)](#)

A list of locales that the intent supports.

Type: Array of strings

Valid Values: en-US

## Errors

### BadRequestException

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

### InternalFailureException

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

### LimitExceededException

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

### NotFoundException

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## GetBuiltinIntents

Service: Amazon Lex Model Building Service

Gets a list of built-in intents that meet the specified criteria.

This operation requires permission for the `lex:GetBuiltinIntents` action.

### Request Syntax

```
GET /builtins/intents/?
locale=locale&maxResults=maxResults&nextToken=nextToken&signatureContains=signatureContains
HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

[locale](#) (p. 278)

A list of locales that the intent supports.

Valid Values: `en-US`

[maxResults](#) (p. 278)

The maximum number of intents to return in the response. The default is 10.

Valid Range: Minimum value of 1. Maximum value of 50.

[nextToken](#) (p. 278)

A pagination token that fetches the next page of intents. If this API call is truncated, Amazon Lex returns a pagination token in the response. To fetch the next page of intents, use the pagination token in the next request.

[signatureContains](#) (p. 278)

Substring to match in built-in intent signatures. An intent will be returned if any part of its signature matches the substring. For example, "xyz" matches both "xyzabc" and "abcxyz." To find the signature for an intent, see [Standard Built-in Intents](#) in the Alexa Skills Kit.

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "intents": [
    {
      "signature": "string",
      "supportedLocales": [ "string" ]
    }
  ],
  "nextToken": "string"
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[intents \(p. 278\)](#)

An array of `BuiltinIntentMetadata` objects, one for each intent in the response.

Type: Array of [BuiltinIntentMetadata \(p. 373\)](#) objects

[nextToken \(p. 278\)](#)

A pagination token that fetches the next page of intents. If the response to this API call is truncated, Amazon Lex returns a pagination token in the response. To fetch the next page of intents, specify the pagination token in the next request.

Type: String

## Errors

`BadRequestException`

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

`InternalFailureException`

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

`LimitExceededException`

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## GetBuiltinSlotTypes

Service: Amazon Lex Model Building Service

Gets a list of built-in slot types that meet the specified criteria.

For a list of built-in slot types, see [Slot Type Reference](#) in the Alexa Skills Kit.

This operation requires permission for the `lex:GetBuiltinSlotTypes` action.

### Request Syntax

```
GET /builtins/slottypes/?  
locale=locale&maxResults=maxResults&nextToken=nextToken&signatureContains=signatureContains  
HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

[locale](#) (p. 280)

A list of locales that the slot type supports.

Valid Values: en-US

[maxResults](#) (p. 280)

The maximum number of slot types to return in the response. The default is 10.

Valid Range: Minimum value of 1. Maximum value of 50.

[nextToken](#) (p. 280)

A pagination token that fetches the next page of slot types. If the response to this API call is truncated, Amazon Lex returns a pagination token in the response. To fetch the next page of slot types, specify the pagination token in the next request.

[signatureContains](#) (p. 280)

Substring to match in built-in slot type signatures. A slot type will be returned if any part of its signature matches the substring. For example, "xyz" matches both "xyzabc" and "abcxyz."

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 200  
Content-type: application/json  
  
{  
  "nextToken": "string",  
  "slotTypes": [  
    {  
      "signature": "string",  
      "supportedLocales": [ "string" ]  
    }  
  ]  
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[nextToken](#) (p. 280)

If the response is truncated, the response includes a pagination token that you can use in your next request to fetch the next page of slot types.

Type: String

[slotTypes](#) (p. 280)

An array of `BuiltInSlotTypeMetadata` objects, one entry for each slot type returned.

Type: Array of [BuiltInSlotTypeMetadata](#) (p. 375) objects

## Errors

`BadRequestException`

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

`InternalFailureException`

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

`LimitExceededException`

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## GetExport

Service: Amazon Lex Model Building Service

Exports the contents of a Amazon Lex resource in a specified format.

### Request Syntax

```
GET /exports/?exportType=exportType&name=name&resourceType=resourceType&version=version
HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

[exportType](#) (p. 282)

The format of the exported data.

Valid Values: ALEXA\_SKILLS\_KIT | LEX

[name](#) (p. 282)

The name of the bot to export.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: [a-zA-Z\_]+

[resourceType](#) (p. 282)

The type of resource to export.

Valid Values: BOT | INTENT | SLOT\_TYPE

[version](#) (p. 282)

The version of the bot to export.

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: [0-9]+

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "exportStatus": "string",
  "exportType": "string",
  "failureReason": "string",
  "name": "string",
  "resourceType": "string",
  "url": "string",
  "version": "string"
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

### [exportStatus \(p. 282\)](#)

The status of the export.

- `IN_PROGRESS` - The export is in progress.
- `READY` - The export is complete.
- `FAILED` - The export could not be completed.

Type: String

Valid Values: `IN_PROGRESS` | `READY` | `FAILED`

### [exportType \(p. 282\)](#)

The format of the exported data.

Type: String

Valid Values: `ALEXA_SKILLS_KIT` | `LEX`

### [failureReason \(p. 282\)](#)

If `status` is `FAILED`, Amazon Lex provides the reason that it failed to export the resource.

Type: String

### [name \(p. 282\)](#)

The name of the bot being exported.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `[a-zA-Z_]+`

### [resourceType \(p. 282\)](#)

The type of the exported resource.

Type: String

Valid Values: `BOT` | `INTENT` | `SLOT_TYPE`

### [url \(p. 282\)](#)

An S3 pre-signed URL that provides the location of the exported resource. The exported resource is a ZIP archive that contains the exported resource in JSON format. The structure of the archive may change. Your code should not rely on the archive structure.

Type: String

### [version \(p. 282\)](#)

The version of the bot being exported.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.



Pattern: [0-9]+

## Errors

### BadRequestException

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

### InternalFailureException

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

### LimitExceededException

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

### NotFoundException

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## GetImport

Service: Amazon Lex Model Building Service

Gets information about an import job started with the `StartImport` operation.

### Request Syntax

```
GET /imports/importId HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

[importId](#) (p. 285)

The identifier of the import job information to return.

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "createdDate": number,
  "failureReason": [ "string" ],
  "importId": "string",
  "importStatus": "string",
  "mergeStrategy": "string",
  "name": "string",
  "resourceType": "string"
}
```

### Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[createdDate](#) (p. 285)

A timestamp for the date and time that the import job was created.

Type: Timestamp

[failureReason](#) (p. 285)

A string that describes why an import job failed to complete.

Type: Array of strings

[importId](#) (p. 285)

The identifier for the specific import job.

Type: String

#### [importStatus \(p. 285\)](#)

The status of the import job. If the status is `FAILED`, you can get the reason for the failure from the `failureReason` field.

Type: String

Valid Values: `IN_PROGRESS` | `COMPLETE` | `FAILED`

#### [mergeStrategy \(p. 285\)](#)

The action taken when there was a conflict between an existing resource and a resource in the import file.

Type: String

Valid Values: `OVERWRITE_LATEST` | `FAIL_ON_CONFLICT`

#### [name \(p. 285\)](#)

The name given to the import job.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `[a-zA-Z_]+`

#### [resourceType \(p. 285\)](#)

The type of resource imported.

Type: String

Valid Values: `BOT` | `INTENT` | `SLOT_TYPE`

## Errors

### BadRequestException

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

### InternalFailureException

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

### LimitExceededException

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

### NotFoundException

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## GetIntent

Service: Amazon Lex Model Building Service

Returns information about an intent. In addition to the intent name, you must specify the intent version.

This operation requires permissions to perform the `lex:GetIntent` action.

## Request Syntax

```
GET /intents/name/versions/version HTTP/1.1
```

## URI Request Parameters

The request requires the following URI parameters.

**name** (p. 288)

The name of the intent. The name is case sensitive.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[A-Za-z_?]+$`

**version** (p. 288)

The version of the intent.

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `\$LATEST|[0-9]+`

## Request Body

The request does not have a request body.

## Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "checksum": "string",
  "conclusionStatement": {
    "messages": [
      {
        "content": "string",
        "contentType": "string",
        "groupNumber": number
      }
    ],
    "responseCard": "string"
  },
  "confirmationPrompt": {
    "maxAttempts": number,
    "messages": [
      {
        "content": "string",
        "contentType": "string",
        "groupNumber": number
      }
    ]
  }
}
```

```

    },
    "responseCard": "string"
  },
  "createdDate": number,
  "description": "string",
  "dialogCodeHook": {
    "messageVersion": "string",
    "uri": "string"
  },
  "followUpPrompt": {
    "prompt": {
      "maxAttempts": number,
      "messages": [
        {
          "content": "string",
          "contentType": "string",
          "groupNumber": number
        }
      ],
      "responseCard": "string"
    },
    "rejectionStatement": {
      "messages": [
        {
          "content": "string",
          "contentType": "string",
          "groupNumber": number
        }
      ],
      "responseCard": "string"
    }
  },
  "fulfillmentActivity": {
    "codeHook": {
      "messageVersion": "string",
      "uri": "string"
    },
    "type": "string"
  },
  "lastUpdatedDate": number,
  "name": "string",
  "parentIntentSignature": "string",
  "rejectionStatement": {
    "messages": [
      {
        "content": "string",
        "contentType": "string",
        "groupNumber": number
      }
    ],
    "responseCard": "string"
  },
  "sampleUtterances": [ "string" ],
  "slots": [
    {
      "description": "string",
      "name": "string",
      "priority": number,
      "responseCard": "string",
      "sampleUtterances": [ "string" ],
      "slotConstraint": "string",
      "slotType": "string",
      "slotTypeVersion": "string",
      "valueElicitationPrompt": {
        "maxAttempts": number,

```

```
        "messages": [
            {
                "content": "string",
                "contentType": "string",
                "groupNumber": number
            }
        ],
        "responseCard": "string"
    }
},
"version": "string"
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

### [checksum \(p. 288\)](#)

Checksum of the intent.

Type: String

### [conclusionStatement \(p. 288\)](#)

After the Lambda function specified in the `fulfillmentActivity` element fulfills the intent, Amazon Lex conveys this statement to the user.

Type: [Statement \(p. 390\)](#) object

### [confirmationPrompt \(p. 288\)](#)

If defined in the bot, Amazon Lex uses prompt to confirm the intent before fulfilling the user's request. For more information, see [PutIntent \(p. 323\)](#).

Type: [Prompt \(p. 384\)](#) object

### [createdDate \(p. 288\)](#)

The date that the intent was created.

Type: Timestamp

### [description \(p. 288\)](#)

A description of the intent.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

### [dialogCodeHook \(p. 288\)](#)

If defined in the bot, Amazon Amazon Lex invokes this Lambda function for each user input. For more information, see [PutIntent \(p. 323\)](#).

Type: [CodeHook \(p. 376\)](#) object

### [followUpPrompt \(p. 288\)](#)

If defined in the bot, Amazon Lex uses this prompt to solicit additional user activity after the intent is fulfilled. For more information, see [PutIntent \(p. 323\)](#).

Type: [FollowUpPrompt](#) (p. 378) object  
[fulfillmentActivity](#) (p. 288)

Describes how the intent is fulfilled. For more information, see [PutIntent](#) (p. 323).

Type: [FulfillmentActivity](#) (p. 379) object  
[lastUpdatedDate](#) (p. 288)

The date that the intent was updated. When you create a resource, the creation date and the last updated date are the same.

Type: Timestamp  
[name](#) (p. 288)

The name of the intent.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[A-Za-z_?]+$`

[parentIntentSignature](#) (p. 288)

A unique identifier for a built-in intent.

Type: String  
[rejectionStatement](#) (p. 288)

If the user answers "no" to the question defined in `confirmationPrompt`, Amazon Lex responds with this statement to acknowledge that the intent was canceled.

Type: [Statement](#) (p. 390) object  
[sampleUtterances](#) (p. 288)

An array of sample utterances configured for the intent.

Type: Array of strings

Array Members: Minimum number of 0 items. Maximum number of 1500 items.

Length Constraints: Minimum length of 1. Maximum length of 200.

[slots](#) (p. 288)

An array of intent slots configured for the intent.

Type: Array of [Slot](#) (p. 386) objects

Array Members: Minimum number of 0 items. Maximum number of 100 items.

[version](#) (p. 288)

The version of the intent.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `\$LATEST|[0-9]+`



## Errors

### BadRequestException

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

### InternalFailureException

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

### LimitExceededException

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

### NotFoundException

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## GetIntents

Service: Amazon Lex Model Building Service

Returns intent information as follows:

- If you specify the `nameContains` field, returns the `$LATEST` version of all intents that contain the specified string.
- If you don't specify the `nameContains` field, returns information about the `$LATEST` version of all intents.

The operation requires permission for the `lex:GetIntents` action.

## Request Syntax

```
GET /intents/?maxResults=maxResults&nameContains=nameContains&nextToken=nextToken HTTP/1.1
```

## URI Request Parameters

The request requires the following URI parameters.

### `maxResults` (p. 293)

The maximum number of intents to return in the response. The default is 10.

Valid Range: Minimum value of 1. Maximum value of 50.

### `nameContains` (p. 293)

Substring to match in intent names. An intent will be returned if any part of its name matches the substring. For example, "xyz" matches both "xyzabc" and "abcxyz."

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^([A-Za-z_?])+`

### `nextToken` (p. 293)

A pagination token that fetches the next page of intents. If the response to this API call is truncated, Amazon Lex returns a pagination token in the response. To fetch the next page of intents, specify the pagination token in the next request.

## Request Body

The request does not have a request body.

## Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "intents": [
    {
      "createdDate": number,
      "description": "string",
      "lastUpdatedDate": number,
      "name": "string",
      "version": "string"
    }
  ]
}
```

```
    }  
  ],  
  "nextToken": "string"  
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[intents \(p. 293\)](#)

An array of `Intent` objects. For more information, see [PutBot \(p. 311\)](#).

Type: Array of [IntentMetadata \(p. 381\)](#) objects

[nextToken \(p. 293\)](#)

If the response is truncated, the response includes a pagination token that you can specify in your next request to fetch the next page of intents.

Type: String

## Errors

`BadRequestException`

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

`InternalFailureException`

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

`LimitExceededException`

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

`NotFoundException`

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)

- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## GetIntentVersions

Service: Amazon Lex Model Building Service

Gets information about all of the versions of an intent.

The `GetIntentVersions` operation returns an `IntentMetadata` object for each version of an intent. For example, if an intent has three numbered versions, the `GetIntentVersions` operation returns four `IntentMetadata` objects in the response, one for each numbered version and one for the `$LATEST` version.

The `GetIntentVersions` operation always returns at least one version, the `$LATEST` version.

This operation requires permissions for the `lex:GetIntentVersions` action.

### Request Syntax

```
GET /intents/name/versions/?maxResults=maxResults&nextToken=nextToken HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

`maxResults` (p. 296)

The maximum number of intent versions to return in the response. The default is 10.

Valid Range: Minimum value of 1. Maximum value of 50.

`name` (p. 296)

The name of the intent for which versions should be returned.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[A-Za-z_?]+$`

`nextToken` (p. 296)

A pagination token for fetching the next page of intent versions. If the response to this call is truncated, Amazon Lex returns a pagination token in the response. To fetch the next page of versions, specify the pagination token in the next request.

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "intents": [
    {
      "createdDate": number,
      "description": "string",
      "lastUpdatedDate": number,
      "name": "string",
      "version": "string"
    }
  ]
}
```

```
  ],  
  "nextToken": "string"  
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[intents \(p. 296\)](#)

An array of [IntentMetadata](#) objects, one for each numbered version of the intent plus one for the `$LATEST` version.

Type: Array of [IntentMetadata \(p. 381\)](#) objects

[nextToken \(p. 296\)](#)

A pagination token for fetching the next page of intent versions. If the response to this call is truncated, Amazon Lex returns a pagination token in the response. To fetch the next page of versions, specify the pagination token in the next request.

Type: String

## Errors

**BadRequestException**

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

**InternalFailureException**

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

**LimitExceededException**

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

**NotFoundException**

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)

- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## GetSlotType

Service: Amazon Lex Model Building Service

Returns information about a specific version of a slot type. In addition to specifying the slot type name, you must specify the slot type version.

This operation requires permissions for the `lex:GetSlotType` action.

### Request Syntax

```
GET /slottypes/name/versions/version HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

**name** (p. 299)

The name of the slot type. The name is case sensitive.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[A-Za-z_?]+$`

**version** (p. 299)

The version of the slot type.

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `\$LATEST|[0-9]+`

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "checksum": "string",
  "createdDate": number,
  "description": "string",
  "enumerationValues": [
    {
      "synonyms": [ "string" ],
      "value": "string"
    }
  ],
  "lastUpdatedDate": number,
  "name": "string",
  "valueSelectionStrategy": "string",
  "version": "string"
}
```

### Response Elements

If the action is successful, the service sends back an HTTP 200 response.



The following data is returned in JSON format by the service.

[checksum \(p. 299\)](#)

Checksum of the `$LATEST` version of the slot type.

Type: String

[createdDate \(p. 299\)](#)

The date that the slot type was created.

Type: Timestamp

[description \(p. 299\)](#)

A description of the slot type.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

[enumerationValues \(p. 299\)](#)

A list of `EnumerationValue` objects that defines the values that the slot type can take.

Type: Array of [EnumerationValue \(p. 377\)](#) objects

Array Members: Minimum number of 1 item. Maximum number of 10000 items.

[lastUpdatedDate \(p. 299\)](#)

The date that the slot type was updated. When you create a resource, the creation date and last update date are the same.

Type: Timestamp

[name \(p. 299\)](#)

The name of the slot type.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[A-Za-z_?]+$`

[valueSelectionStrategy \(p. 299\)](#)

The strategy that Amazon Lex uses to determine the value of the slot. For more information, see [PutSlotType \(p. 333\)](#).

Type: String

Valid Values: `ORIGINAL_VALUE` | `TOP_RESOLUTION`

[version \(p. 299\)](#)

The version of the slot type.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `\$LATEST|[0-9]+`

## Errors

### BadRequestException

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

### InternalFailureException

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

### LimitExceededException

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

### NotFoundException

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## GetSlotTypes

Service: Amazon Lex Model Building Service

Returns slot type information as follows:

- If you specify the `nameContains` field, returns the `$LATEST` version of all slot types that contain the specified string.
- If you don't specify the `nameContains` field, returns information about the `$LATEST` version of all slot types.

The operation requires permission for the `lex:GetSlotTypes` action.

## Request Syntax

```
GET /slottypes/?maxResults=maxResults&nameContains=nameContains&nextToken=nextToken
HTTP/1.1
```

## URI Request Parameters

The request requires the following URI parameters.

[maxResults](#) (p. 302)

The maximum number of slot types to return in the response. The default is 10.

Valid Range: Minimum value of 1. Maximum value of 50.

[nameContains](#) (p. 302)

Substring to match in slot type names. A slot type will be returned if any part of its name matches the substring. For example, "xyz" matches both "xyzabc" and "abcxyz."

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[A-Za-z_?]+$`

[nextToken](#) (p. 302)

A pagination token that fetches the next page of slot types. If the response to this API call is truncated, Amazon Lex returns a pagination token in the response. To fetch next page of slot types, specify the pagination token in the next request.

## Request Body

The request does not have a request body.

## Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "nextToken": "string",
  "slotTypes": [
    {
      "createdDate": number,
      "description": "string",
      "lastUpdatedDate": number,
```

```
        "name": "string",  
        "version": "string"  
    }  
]  
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[nextToken](#) (p. 302)

If the response is truncated, it includes a pagination token that you can specify in your next request to fetch the next page of slot types.

Type: String

[slotTypes](#) (p. 302)

An array of objects, one for each slot type, that provides information such as the name of the slot type, the version, and a description.

Type: Array of [SlotTypeMetadata](#) (p. 388) objects

## Errors

**BadRequestException**

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

**InternalFailureException**

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

**LimitExceededException**

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

**NotFoundException**

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)

- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## GetSlotTypeVersions

Service: Amazon Lex Model Building Service

Gets information about all versions of a slot type.

The GetSlotTypeVersions operation returns a SlotTypeMetadata object for each version of a slot type. For example, if a slot type has three numbered versions, the GetSlotTypeVersions operation returns four SlotTypeMetadata objects in the response, one for each numbered version and one for the \$LATEST version.

The GetSlotTypeVersions operation always returns at least one version, the \$LATEST version.

This operation requires permissions for the lex:GetSlotTypeVersions action.

### Request Syntax

```
GET /slottypes/name/versions/?maxResults=maxResults&nextToken=nextToken HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

**maxResults** (p. 305)

The maximum number of slot type versions to return in the response. The default is 10.

Valid Range: Minimum value of 1. Maximum value of 50.

**name** (p. 305)

The name of the slot type for which versions should be returned.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: ^([A-Za-z]\_?)+\$

**nextToken** (p. 305)

A pagination token for fetching the next page of slot type versions. If the response to this call is truncated, Amazon Lex returns a pagination token in the response. To fetch the next page of versions, specify the pagination token in the next request.

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "nextToken": "string",
  "slotTypes": [
    {
      "createdDate": number,
      "description": "string",
      "lastUpdatedDate": number,
      "name": "string",
      "version": "string"
    }
  ]
}
```

```
}  
  ]  
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[nextToken](#) (p. 305)

A pagination token for fetching the next page of slot type versions. If the response to this call is truncated, Amazon Lex returns a pagination token in the response. To fetch the next page of versions, specify the pagination token in the next request.

Type: String

[slotTypes](#) (p. 305)

An array of `SlotTypeMetadata` objects, one for each numbered version of the slot type plus one for the `$LATEST` version.

Type: Array of [SlotTypeMetadata](#) (p. 388) objects

## Errors

`BadRequestException`

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

`InternalFailureException`

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

`LimitExceededException`

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

`NotFoundException`

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)

- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)



## GetUtterancesView

Service: Amazon Lex Model Building Service

Use the `GetUtterancesView` operation to get information about the utterances that your users have made to your bot. You can use this list to tune the utterances that your bot responds to.

For example, say that you have created a bot to order flowers. After your users have used your bot for a while, use the `GetUtterancesView` operation to see the requests that they have made and whether they have been successful. You might find that the utterance "I want flowers" is not being recognized. You could add this utterance to the `OrderFlowers` intent so that your bot recognizes that utterance.

After you publish a new version of a bot, you can get information about the old version and the new so that you can compare the performance across the two versions.

Utterance statistics are generated once a day. Data is available for the last 15 days. You can request information for up to 5 versions of your bot in each request. Amazon Lex returns the most frequent utterances received by the bot in the last 15 days. The response contains information about a maximum of 100 utterances for each version.

If you set `childDirected` field to true when you created your bot, or if you opted out of participating in improving Amazon Lex, utterances are not available.

This operation requires permissions for the `lex:GetUtterancesView` action.

## Request Syntax

```
GET /bots/botname/utterances?
view=aggregation&bot_versions=botVersions&status_type=statusType HTTP/1.1
```

## URI Request Parameters

The request requires the following URI parameters.

[botname](#) (p. 308)

The name of the bot for which utterance information should be returned.

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^[A-Za-z_?]+$`

[botVersions](#) (p. 308)

An array of bot versions for which utterance information should be returned. The limit is 5 versions per request.

Array Members: Minimum number of 1 item. Maximum number of 5 items.

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `\$LATEST|[0-9]+`

[statusType](#) (p. 308)

To return utterances that were recognized and handled, use `Detected`. To return utterances that were not recognized, use `Missed`.

Valid Values: `Detected` | `Missed`

## Request Body

The request does not have a request body.

## Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "botName": "string",
  "utterances": [
    {
      "botVersion": "string",
      "utterances": [
        {
          "count": number,
          "distinctUsers": number,
          "firstUtteredDate": number,
          "lastUtteredDate": number,
          "utteranceString": "string"
        }
      ]
    }
  ]
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

### [botName \(p. 309\)](#)

The name of the bot for which utterance information was returned.

Type: String

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: ^([A-Za-z\_?])+ \$

### [utterances \(p. 309\)](#)

An array of [UtteranceList \(p. 392\)](#) objects, each containing a list of [UtteranceData \(p. 391\)](#) objects describing the utterances that were processed by your bot. The response contains a maximum of 100 [UtteranceData](#) objects for each version. Amazon Lex returns the most frequent utterances received by the bot in the last 15 days.

Type: Array of [UtteranceList \(p. 392\)](#) objects

## Errors

### BadRequestException

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

#### InternalFailureException

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

#### LimitExceededException

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## PutBot

Service: Amazon Lex Model Building Service

Creates an Amazon Lex conversational bot or replaces an existing bot. When you create or update a bot you are only required to specify a name, a locale, and whether the bot is directed toward children under age 13. You can use this to add intents later, or to remove intents from an existing bot. When you create a bot with the minimum information, the bot is created or updated but Amazon Lex returns the response `FAILED`. You can build the bot after you add one or more intents. For more information about Amazon Lex bots, see [Amazon Lex: Como ele funciona](#) (p. 3).

If you specify the name of an existing bot, the fields in the request replace the existing values in the `$LATEST` version of the bot. Amazon Lex removes any fields that you don't provide values for in the request, except for the `idleTTLInSeconds` and `privacySettings` fields, which are set to their default values. If you don't specify values for required fields, Amazon Lex throws an exception.

This operation requires permissions for the `lex:PutBot` action. For more information, see [Identity and Access Management para Amazon Lex](#) (p. 184).

## Request Syntax

```
PUT /bots/name/versions/$LATEST HTTP/1.1
Content-type: application/json
```

```
{
  "abortStatement": {
    "messages": [
      {
        "content": "string",
        "contentType": "string",
        "groupNumber": number
      }
    ],
    "responseCard": "string"
  },
  "checksum": "string",
  "childDirected": boolean,
  "clarificationPrompt": {
    "maxAttempts": number,
    "messages": [
      {
        "content": "string",
        "contentType": "string",
        "groupNumber": number
      }
    ],
    "responseCard": "string"
  },
  "createVersion": boolean,
  "description": "string",
  "idleSessionTTLInSeconds": number,
  "intents": [
    {
      "intentName": "string",
      "intentVersion": "string"
    }
  ],
  "locale": "string",
  "processBehavior": "string",
  "voiceId": "string"
}
```

## URI Request Parameters

The request requires the following URI parameters.

### [name \(p. 311\)](#)

The name of the bot. The name is not case sensitive.

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^([A-Za-z]_?)+$`

## Request Body

The request accepts the following data in JSON format.

### [abortStatement \(p. 311\)](#)

When Amazon Lex can't understand the user's input in context, it tries to elicit the information a few times. After that, Amazon Lex sends the message defined in `abortStatement` to the user, and then aborts the conversation. To set the number of retries, use the `valueElicitationPrompt` field for the slot type.

For example, in a pizza ordering bot, Amazon Lex might ask a user "What type of crust would you like?" If the user's response is not one of the expected responses (for example, "thin crust, "deep dish," etc.), Amazon Lex tries to elicit a correct response a few more times.

For example, in a pizza ordering application, `OrderPizza` might be one of the intents. This intent might require the `CrustType` slot. You specify the `valueElicitationPrompt` field when you create the `CrustType` slot.

Type: [Statement \(p. 390\)](#) object

Required: No

### [checksum \(p. 311\)](#)

Identifies a specific revision of the `$LATEST` version.

When you create a new bot, leave the `checksum` field blank. If you specify a checksum you get a `BadRequestException` exception.

When you want to update a bot, set the `checksum` field to the checksum of the most recent revision of the `$LATEST` version. If you don't specify the `checksum` field, or if the checksum does not match the `$LATEST` version, you get a `PreconditionFailedException` exception.

Type: String

Required: No

### [childDirected \(p. 311\)](#)

For each Amazon Lex bot created with the Amazon Lex Model Building Service, you must specify whether your use of Amazon Lex is related to a website, program, or other application that is directed or targeted, in whole or in part, to children under age 13 and subject to the Children's Online Privacy Protection Act (COPPA) by specifying `true` or `false` in the `childDirected` field. By specifying `true` in the `childDirected` field, you confirm that your use of Amazon Lex is related to a website, program, or other application that is directed or targeted, in whole or in part, to children under age 13 and subject to COPPA. By specifying `false` in the `childDirected` field, you confirm that your use of Amazon Lex is not related to a website, program, or other application that is directed or targeted, in whole or in part, to children under age 13 and subject to COPPA. You may not specify a default

value for the `childDirected` field that does not accurately reflect whether your use of Amazon Lex is related to a website, program, or other application that is directed or targeted, in whole or in part, to children under age 13 and subject to COPPA.

If your use of Amazon Lex relates to a website, program, or other application that is directed in whole or in part, to children under age 13, you must obtain any required verifiable parental consent under COPPA. For information regarding the use of Amazon Lex in connection with websites, programs, or other applications that are directed or targeted, in whole or in part, to children under age 13, see the [Amazon Lex FAQ](#).

Type: Boolean

Required: Yes

#### [clarificationPrompt \(p. 311\)](#)

When Amazon Lex doesn't understand the user's intent, it uses this message to get clarification. To specify how many times Amazon Lex should repeat the clarification prompt, use the `maxAttempts` field. If Amazon Lex still doesn't understand, it sends the message in the `abortStatement` field.

When you create a clarification prompt, make sure that it suggests the correct response from the user. for example, for a bot that orders pizza and drinks, you might create this clarification prompt: "What would you like to do? You can say 'Order a pizza' or 'Order a drink.'"

Type: [Prompt \(p. 384\)](#) object

Required: No

#### [createVersion \(p. 311\)](#)

When set to `true` a new numbered version of the bot is created. This is the same as calling the `CreateBotVersion` operation. If you do not specify `createVersion`, the default is `false`.

Type: Boolean

Required: No

#### [description \(p. 311\)](#)

A description of the bot.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

Required: No

#### [idleSessionTTLInSeconds \(p. 311\)](#)

The maximum time in seconds that Amazon Lex retains the data gathered in a conversation.

A user interaction session remains active for the amount of time specified. If no conversation occurs during this time, the session expires and Amazon Lex deletes any data provided before the timeout.

For example, suppose that a user chooses the `OrderPizza` intent, but gets sidetracked halfway through placing an order. If the user doesn't complete the order within the specified time, Amazon Lex discards the slot information that it gathered, and the user must start over.

If you don't include the `idleSessionTTLInSeconds` element in a `PutBot` operation request, Amazon Lex uses the default value. This is also true if the request replaces an existing bot.

The default is 300 seconds (5 minutes).

Type: Integer

Valid Range: Minimum value of 60. Maximum value of 86400.

Required: No

[intents \(p. 311\)](#)

An array of `Intent` objects. Each intent represents a command that a user can express. For example, a pizza ordering bot might support an `OrderPizza` intent. For more information, see [Amazon Lex: Como ele funciona \(p. 3\)](#).

Type: Array of [Intent \(p. 380\)](#) objects

Required: No

[locale \(p. 311\)](#)

Specifies the target locale for the bot. Any intent used in the bot must be compatible with the locale of the bot.

The default is `en-US`.

Type: String

Valid Values: `en-US`

Required: Yes

[processBehavior \(p. 311\)](#)

If you set the `processBehavior` element to `BUILD`, Amazon Lex builds the bot so that it can be run. If you set the element to `SAVE` Amazon Lex saves the bot, but doesn't build it.

If you don't specify this value, the default value is `BUILD`.

Type: String

Valid Values: `SAVE` | `BUILD`

Required: No

[voiceId \(p. 311\)](#)

The Amazon Polly voice ID that you want Amazon Lex to use for voice interactions with the user. The locale configured for the voice must match the locale of the bot. For more information, see [Voices in Amazon Polly](#) in the Amazon Polly Developer Guide.

Type: String

Required: No

## Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "abortStatement": {
    "messages": [
      {
        "content": "string",
        "contentType": "string",
        "groupNumber": number
      }
    ]
  }
}
```

```
    ],
    "responseCard": "string"
  },
  "checksum": "string",
  "childDirected": boolean,
  "clarificationPrompt": {
    "maxAttempts": number,
    "messages": [
      {
        "content": "string",
        "contentType": "string",
        "groupNumber": number
      }
    ]
  },
  "responseCard": "string"
},
"createdDate": number,
"createVersion": boolean,
"description": "string",
"failureReason": "string",
"idleSessionTTLInSeconds": number,
"intents": [
  {
    "intentName": "string",
    "intentVersion": "string"
  }
],
"lastUpdatedDate": number,
"locale": "string",
"name": "string",
"status": "string",
"version": "string",
"voiceId": "string"
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

### [abortStatement \(p. 314\)](#)

The message that Amazon Lex uses to abort a conversation. For more information, see [PutBot \(p. 311\)](#).

Type: [Statement \(p. 390\)](#) object

### [checksum \(p. 314\)](#)

Checksum of the bot that you created.

Type: String

### [childDirected \(p. 314\)](#)

For each Amazon Lex bot created with the Amazon Lex Model Building Service, you must specify whether your use of Amazon Lex is related to a website, program, or other application that is directed or targeted, in whole or in part, to children under age 13 and subject to the Children's Online Privacy Protection Act (COPPA) by specifying `true` or `false` in the `childDirected` field. By specifying `true` in the `childDirected` field, you confirm that your use of Amazon Lex is related to a website, program, or other application that is directed or targeted, in whole or in part, to children under age 13 and subject to COPPA. By specifying `false` in the `childDirected` field, you confirm that your use of Amazon Lex is not related to a website, program, or other application that is directed or targeted,



in whole or in part, to children under age 13 and subject to COPPA. You may not specify a default value for the `childDirected` field that does not accurately reflect whether your use of Amazon Lex is related to a website, program, or other application that is directed or targeted, in whole or in part, to children under age 13 and subject to COPPA.

If your use of Amazon Lex relates to a website, program, or other application that is directed in whole or in part, to children under age 13, you must obtain any required verifiable parental consent under COPPA. For information regarding the use of Amazon Lex in connection with websites, programs, or other applications that are directed or targeted, in whole or in part, to children under age 13, see the [Amazon Lex FAQ](#).

Type: Boolean

[clarificationPrompt](#) (p. 314)

The prompts that Amazon Lex uses when it doesn't understand the user's intent. For more information, see [PutBot](#) (p. 311).

Type: [Prompt](#) (p. 384) object

[createdDate](#) (p. 314)

The date that the bot was created.

Type: Timestamp

[createVersion](#) (p. 314)

True if a new version of the bot was created. If the `createVersion` field was not specified in the request, the `createVersion` field is set to false in the response.

Type: Boolean

[description](#) (p. 314)

A description of the bot.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

[failureReason](#) (p. 314)

If status is `FAILED`, Amazon Lex provides the reason that it failed to build the bot.

Type: String

[idleSessionTTLInSeconds](#) (p. 314)

The maximum length of time that Amazon Lex retains the data gathered in a conversation. For more information, see [PutBot](#) (p. 311).

Type: Integer

Valid Range: Minimum value of 60. Maximum value of 86400.

[intents](#) (p. 314)

An array of `Intent` objects. For more information, see [PutBot](#) (p. 311).

Type: Array of [Intent](#) (p. 380) objects

[lastUpdatedDate](#) (p. 314)

The date that the bot was updated. When you create a resource, the creation date and last updated date are the same.

Type: Timestamp

[locale \(p. 314\)](#)

The target locale for the bot.

Type: String

Valid Values: en-US

[name \(p. 314\)](#)

The name of the bot.

Type: String

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: ^([A-Za-z]\_?)+\$

[status \(p. 314\)](#)

When you send a request to create a bot with `processBehavior` set to `BUILD`, Amazon Lex sets the `status` response element to `BUILDING`.

In the `READY_BASIC_TESTING` state you can test the bot with user inputs that exactly match the utterances configured for the bot's intents and values in the slot types.

If Amazon Lex can't build the bot, Amazon Lex sets `status` to `FAILED`. Amazon Lex returns the reason for the failure in the `failureReason` response element.

When you set `processBehavior` to `SAVE`, Amazon Lex sets the status code to `NOT_BUILT`.

When the bot is in the `READY` state you can test and publish the bot.

Type: String

Valid Values: `BUILDING` | `READY` | `READY_BASIC_TESTING` | `FAILED` | `NOT_BUILT`

[version \(p. 314\)](#)

The version of the bot. For a new bot, the version is always `$LATEST`.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: \(\$LATEST|[0-9])+

[voiceId \(p. 314\)](#)

The Amazon Polly voice ID that Amazon Lex uses for voice interaction with the user. For more information, see [PutBot \(p. 311\)](#).

Type: String

## Errors

### BadRequestException

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

#### ConflictException

There was a conflict processing the request. Try your request again.

HTTP Status Code: 409

#### InternalFailureException

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

#### LimitExceededException

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

#### PreconditionFailedException

The checksum of the resource that you are trying to change does not match the checksum in the request. Check the resource's checksum and try again.

HTTP Status Code: 412

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## PutBotAlias

Service: Amazon Lex Model Building Service

Creates an alias for the specified version of the bot or replaces an alias for the specified bot. To change the version of the bot that the alias points to, replace the alias. For more information about aliases, see [Controle de versão e aliases \(p. 102\)](#).

This operation requires permissions for the `lex:PutBotAlias` action.

### Request Syntax

```
PUT /bots/botName/aliases/name HTTP/1.1
Content-type: application/json

{
  "botVersion": "string",
  "checksum": "string",
  "description": "string"
}
```

### URI Request Parameters

The request requires the following URI parameters.

[botName \(p. 319\)](#)

The name of the bot.

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^[A-Za-z_?]+$`

[name \(p. 319\)](#)

The name of the alias. The name is not case sensitive.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[A-Za-z_?]+$`

### Request Body

The request accepts the following data in JSON format.

[botVersion \(p. 319\)](#)

The version of the bot.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `\$LATEST|[0-9]+`

Required: Yes

[checksum \(p. 319\)](#)

Identifies a specific revision of the `$LATEST` version.

When you create a new bot alias, leave the `checksum` field blank. If you specify a checksum you get a `BadRequestException` exception.

When you want to update a bot alias, set the `checksum` field to the checksum of the most recent revision of the `$LATEST` version. If you don't specify the `checksum` field, or if the checksum does not match the `$LATEST` version, you get a `PreconditionFailedException` exception.

Type: String

Required: No

[description \(p. 319\)](#)

A description of the alias.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

Required: No

## Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "botName": "string",
  "botVersion": "string",
  "checksum": "string",
  "createdDate": number,
  "description": "string",
  "lastUpdatedDate": number,
  "name": "string"
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[botName \(p. 320\)](#)

The name of the bot that the alias points to.

Type: String

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^[A-Za-z_?]+$`

[botVersion \(p. 320\)](#)

The version of the bot that the alias points to.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `\$LATEST|[0-9]+`

#### [checksum \(p. 320\)](#)

The checksum for the current version of the alias.

Type: String

#### [createdDate \(p. 320\)](#)

The date that the bot alias was created.

Type: Timestamp

#### [description \(p. 320\)](#)

A description of the alias.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

#### [lastUpdatedDate \(p. 320\)](#)

The date that the bot alias was updated. When you create a resource, the creation date and the last updated date are the same.

Type: Timestamp

#### [name \(p. 320\)](#)

The name of the alias.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[A-Za-z_?]+$`

## Errors

### BadRequestException

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

### ConflictException

There was a conflict processing the request. Try your request again.

HTTP Status Code: 409

### InternalFailureException

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

### LimitExceededException

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

### PreconditionFailedException

The checksum of the resource that you are trying to change does not match the checksum in the request. Check the resource's checksum and try again.

HTTP Status Code: 412

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## PutIntent

Service: Amazon Lex Model Building Service

Creates an intent or replaces an existing intent.

To define the interaction between the user and your bot, you use one or more intents. For a pizza ordering bot, for example, you would create an `OrderPizza` intent.

To create an intent or replace an existing intent, you must provide the following:

- Intent name. For example, `OrderPizza`.
- Sample utterances. For example, "Can I order a pizza, please." and "I want to order a pizza."
- Information to be gathered. You specify slot types for the information that your bot will request from the user. You can specify standard slot types, such as a date or a time, or custom slot types such as the size and crust of a pizza.
- How the intent will be fulfilled. You can provide a Lambda function or configure the intent to return the intent information to the client application. If you use a Lambda function, when all of the intent information is available, Amazon Lex invokes your Lambda function. If you configure your intent to return the intent information to the client application.

You can specify other optional information in the request, such as:

- A confirmation prompt to ask the user to confirm an intent. For example, "Shall I order your pizza?"
- A conclusion statement to send to the user after the intent has been fulfilled. For example, "I placed your pizza order."
- A follow-up prompt that asks the user for additional activity. For example, asking "Do you want to order a drink with your pizza?"

If you specify an existing intent name to update the intent, Amazon Lex replaces the values in the `$LATEST` version of the intent with the values in the request. Amazon Lex removes fields that you don't provide in the request. If you don't specify the required fields, Amazon Lex throws an exception. When you update the `$LATEST` version of an intent, the `status` field of any bot that uses the `$LATEST` version of the intent is set to `NOT_BUILT`.

For more information, see [Amazon Lex: Como ele funciona \(p. 3\)](#).

This operation requires permissions for the `lex:PutIntent` action.

## Request Syntax

```
PUT /intents/name/versions/$LATEST HTTP/1.1
Content-type: application/json
```

```
{
  "checksum": "string",
  "conclusionStatement": {
    "messages": [
      {
        "content": "string",
        "contentType": "string",
        "groupNumber": number
      }
    ],
    "responseCard": "string"
  },
  "confirmationPrompt": {
    "maxAttempts": number,
```



```
    "messages": [
      {
        "content": "string",
        "contentType": "string",
        "groupNumber": number
      }
    ],
    "responseCard": "string"
  },
  "createVersion": boolean,
  "description": "string",
  "dialogCodeHook": {
    "messageVersion": "string",
    "uri": "string"
  },
  "followUpPrompt": {
    "prompt": {
      "maxAttempts": number,
      "messages": [
        {
          "content": "string",
          "contentType": "string",
          "groupNumber": number
        }
      ],
      "responseCard": "string"
    },
    "rejectionStatement": {
      "messages": [
        {
          "content": "string",
          "contentType": "string",
          "groupNumber": number
        }
      ],
      "responseCard": "string"
    }
  },
  "fulfillmentActivity": {
    "codeHook": {
      "messageVersion": "string",
      "uri": "string"
    },
    "type": "string"
  },
  "parentIntentSignature": "string",
  "rejectionStatement": {
    "messages": [
      {
        "content": "string",
        "contentType": "string",
        "groupNumber": number
      }
    ],
    "responseCard": "string"
  },
  "sampleUtterances": [ "string" ],
  "slots": [
    {
      "description": "string",
      "name": "string",
      "priority": number,
      "responseCard": "string",
      "sampleUtterances": [ "string" ],
      "slotConstraint": "string",
      "slotType": "string",
```

```
    "slotTypeVersion": "string",
    "valueElicitationPrompt": {
      "maxAttempts": number,
      "messages": [
        {
          "content": "string",
          "contentType": "string",
          "groupNumber": number
        }
      ],
      "responseCard": "string"
    }
  ]
}
```

## URI Request Parameters

The request requires the following URI parameters.

### [name \(p. 323\)](#)

The name of the intent. The name is not case sensitive.

The name can't match a built-in intent name, or a built-in intent name with "AMAZON." removed. For example, because there is a built-in intent called `AMAZON.HelpIntent`, you can't create a custom intent called `HelpIntent`.

For a list of built-in intents, see [Standard Built-in Intents](#) in the Alexa Skills Kit.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^([A-Za-z_?])+ $`

## Request Body

The request accepts the following data in JSON format.

### [checksum \(p. 323\)](#)

Identifies a specific revision of the `$LATEST` version.

When you create a new intent, leave the `checksum` field blank. If you specify a checksum you get a `BadRequestException` exception.

When you want to update a intent, set the `checksum` field to the checksum of the most recent revision of the `$LATEST` version. If you don't specify the `checksum` field, or if the checksum does not match the `$LATEST` version, you get a `PreconditionFailedException` exception.

Type: String

Required: No

### [conclusionStatement \(p. 323\)](#)

The statement that you want Amazon Lex to convey to the user after the intent is successfully fulfilled by the Lambda function.

This element is relevant only if you provide a Lambda function in the `fulfillmentActivity`. If you return the intent to the client application, you can't specify this element.

#### Note

The `followUpPrompt` and `conclusionStatement` are mutually exclusive. You can specify only one.

Type: [Statement \(p. 390\)](#) object

Required: No

[confirmationPrompt \(p. 323\)](#)

Prompts the user to confirm the intent. This question should have a yes or no answer.

Amazon Lex uses this prompt to ensure that the user acknowledges that the intent is ready for fulfillment. For example, with the `orderPizza` intent, you might want to confirm that the order is correct before placing it. For other intents, such as intents that simply respond to user questions, you might not need to ask the user for confirmation before providing the information.

#### Note

You must provide both the `rejectionStatement` and the `confirmationPrompt`, or neither.

Type: [Prompt \(p. 384\)](#) object

Required: No

[createVersion \(p. 323\)](#)

When set to `true` a new numbered version of the intent is created. This is the same as calling the `CreateIntentVersion` operation. If you do not specify `createVersion`, the default is `false`.

Type: Boolean

Required: No

[description \(p. 323\)](#)

A description of the intent.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

Required: No

[dialogCodeHook \(p. 323\)](#)

Specifies a Lambda function to invoke for each user input. You can invoke this Lambda function to personalize user interaction.

For example, suppose your bot determines that the user is John. Your Lambda function might retrieve John's information from a backend database and prepopulate some of the values. For example, if you find that John is gluten intolerant, you might set the corresponding intent slot, `GlutenIntolerant`, to `true`. You might find John's phone number and set the corresponding session attribute.

Type: [CodeHook \(p. 376\)](#) object

Required: No

[followUpPrompt \(p. 323\)](#)

Amazon Lex uses this prompt to solicit additional activity after fulfilling an intent. For example, after the `orderPizza` intent is fulfilled, you might prompt the user to order a drink.

The action that Amazon Lex takes depends on the user's response, as follows:

- If the user says "Yes" it responds with the clarification prompt that is configured for the bot.
- If the user says "Yes" and continues with an utterance that triggers an intent it starts a conversation for the intent.
- If the user says "No" it responds with the rejection statement configured for the follow-up prompt.
- If it doesn't recognize the utterance it repeats the follow-up prompt again.

The `followUpPrompt` field and the `conclusionStatement` field are mutually exclusive. You can specify only one.

Type: [FollowUpPrompt \(p. 378\)](#) object

Required: No

[fulfillmentActivity \(p. 323\)](#)

Required. Describes how the intent is fulfilled. For example, after a user provides all of the information for a pizza order, `fulfillmentActivity` defines how the bot places an order with a local pizza store.

You might configure Amazon Lex to return all of the intent information to the client application, or direct it to invoke a Lambda function that can process the intent (for example, place an order with a pizzeria).

Type: [FulfillmentActivity \(p. 379\)](#) object

Required: No

[parentIntentSignature \(p. 323\)](#)

A unique identifier for the built-in intent to base this intent on. To find the signature for an intent, see [Standard Built-in Intents](#) in the Alexa Skills Kit.

Type: String

Required: No

[rejectionStatement \(p. 323\)](#)

When the user answers "no" to the question defined in `confirmationPrompt`, Amazon Lex responds with this statement to acknowledge that the intent was canceled.

#### Note

You must provide both the `rejectionStatement` and the `confirmationPrompt`, or neither.

Type: [Statement \(p. 390\)](#) object

Required: No

[sampleUtterances \(p. 323\)](#)

An array of utterances (strings) that a user might say to signal the intent. For example, "I want {PizzaSize} pizza", "Order {Quantity} {PizzaSize} pizzas".

In each utterance, a slot name is enclosed in curly braces.

Type: Array of strings

Array Members: Minimum number of 0 items. Maximum number of 1500 items.

Length Constraints: Minimum length of 1. Maximum length of 200.

Required: No

### slots (p. 323)

An array of intent slots. At runtime, Amazon Lex elicits required slot values from the user using prompts defined in the slots. For more information, see [Amazon Lex: Como ele funciona \(p. 3\)](#).

Type: Array of [Slot \(p. 386\)](#) objects

Array Members: Minimum number of 0 items. Maximum number of 100 items.

Required: No

## Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "checksum": "string",
  "conclusionStatement": {
    "messages": [
      {
        "content": "string",
        "contentType": "string",
        "groupNumber": number
      }
    ],
    "responseCard": "string"
  },
  "confirmationPrompt": {
    "maxAttempts": number,
    "messages": [
      {
        "content": "string",
        "contentType": "string",
        "groupNumber": number
      }
    ],
    "responseCard": "string"
  },
  "createdDate": number,
  "createVersion": boolean,
  "description": "string",
  "dialogCodeHook": {
    "messageVersion": "string",
    "uri": "string"
  },
  "followUpPrompt": {
    "prompt": {
      "maxAttempts": number,
      "messages": [
        {
          "content": "string",
          "contentType": "string",
          "groupNumber": number
        }
      ]
    },
    "responseCard": "string"
  },
  "rejectionStatement": {
    "messages": [
      {
        "content": "string",
        "contentType": "string",

```

```

        "groupNumber": number
      }
    ],
    "responseCard": "string"
  }
},
"fulfillmentActivity": {
  "codeHook": {
    "messageVersion": "string",
    "uri": "string"
  },
  "type": "string"
},
"lastUpdatedDate": number,
"name": "string",
"parentIntentSignature": "string",
"rejectionStatement": {
  "messages": [
    {
      "content": "string",
      "contentType": "string",
      "groupNumber": number
    }
  ],
  "responseCard": "string"
},
"sampleUtterances": [ "string" ],
"slots": [
  {
    "description": "string",
    "name": "string",
    "priority": number,
    "responseCard": "string",
    "sampleUtterances": [ "string" ],
    "slotConstraint": "string",
    "slotType": "string",
    "slotTypeVersion": "string",
    "valueElicitationPrompt": {
      "maxAttempts": number,
      "messages": [
        {
          "content": "string",
          "contentType": "string",
          "groupNumber": number
        }
      ],
      "responseCard": "string"
    }
  }
],
"version": "string"
}

```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[checksum \(p. 328\)](#)

Checksum of the \$LATEST version of the intent created or updated.

Type: String

[conclusionStatement \(p. 328\)](#)

After the Lambda function specified in the `fulfillmentActivity` intent fulfills the intent, Amazon Lex conveys this statement to the user.

Type: [Statement \(p. 390\)](#) object

[confirmationPrompt \(p. 328\)](#)

If defined in the intent, Amazon Lex prompts the user to confirm the intent before fulfilling it.

Type: [Prompt \(p. 384\)](#) object

[createdAt \(p. 328\)](#)

The date that the intent was created.

Type: Timestamp

[createVersion \(p. 328\)](#)

True if a new version of the intent was created. If the `createVersion` field was not specified in the request, the `createVersion` field is set to false in the response.

Type: Boolean

[description \(p. 328\)](#)

A description of the intent.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

[dialogCodeHook \(p. 328\)](#)

If defined in the intent, Amazon Lex invokes this Lambda function for each user input.

Type: [CodeHook \(p. 376\)](#) object

[followUpPrompt \(p. 328\)](#)

If defined in the intent, Amazon Lex uses this prompt to solicit additional user activity after the intent is fulfilled.

Type: [FollowUpPrompt \(p. 378\)](#) object

[fulfillmentActivity \(p. 328\)](#)

If defined in the intent, Amazon Lex invokes this Lambda function to fulfill the intent after the user provides all of the information required by the intent.

Type: [FulfillmentActivity \(p. 379\)](#) object

[lastUpdatedDate \(p. 328\)](#)

The date that the intent was updated. When you create a resource, the creation date and last update dates are the same.

Type: Timestamp

[name \(p. 328\)](#)

The name of the intent.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[A-Za-z_?]+$`

[parentIntentSignature \(p. 328\)](#)

A unique identifier for the built-in intent that this intent is based on.

Type: String

[rejectionStatement \(p. 328\)](#)

If the user answers "no" to the question defined in `confirmationPrompt` Amazon Lex responds with this statement to acknowledge that the intent was canceled.

Type: [Statement \(p. 390\)](#) object

[sampleUtterances \(p. 328\)](#)

An array of sample utterances that are configured for the intent.

Type: Array of strings

Array Members: Minimum number of 0 items. Maximum number of 1500 items.

Length Constraints: Minimum length of 1. Maximum length of 200.

[slots \(p. 328\)](#)

An array of intent slots that are configured for the intent.

Type: Array of [Slot \(p. 386\)](#) objects

Array Members: Minimum number of 0 items. Maximum number of 100 items.

[version \(p. 328\)](#)

The version of the intent. For a new intent, the version is always `$LATEST`.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `\$LATEST|[0-9]+`

## Errors

### BadRequestException

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

### ConflictException

There was a conflict processing the request. Try your request again.

HTTP Status Code: 409

### InternalFailureException

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

### LimitExceededException

The request exceeded a limit. Try your request again.



HTTP Status Code: 429

PreconditionFailedException

The checksum of the resource that you are trying to change does not match the checksum in the request. Check the resource's checksum and try again.

HTTP Status Code: 412

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## PutSlotType

Service: Amazon Lex Model Building Service

Creates a custom slot type or replaces an existing custom slot type.

To create a custom slot type, specify a name for the slot type and a set of enumeration values, which are the values that a slot of this type can assume. For more information, see [Amazon Lex: Como ele funciona](#) (p. 3).

If you specify the name of an existing slot type, the fields in the request replace the existing values in the `$LATEST` version of the slot type. Amazon Lex removes the fields that you don't provide in the request. If you don't specify required fields, Amazon Lex throws an exception. When you update the `$LATEST` version of a slot type, if a bot uses the `$LATEST` version of an intent that contains the slot type, the bot's `status` field is set to `NOT_BUILT`.

This operation requires permissions for the `lex:PutSlotType` action.

### Request Syntax

```
PUT /slottypes/name/versions/$LATEST HTTP/1.1
Content-type: application/json

{
  "checksum": "string",
  "createVersion": boolean,
  "description": "string",
  "enumerationValues": [
    {
      "synonyms": [ "string" ],
      "value": "string"
    }
  ],
  "valueSelectionStrategy": "string"
}
```

### URI Request Parameters

The request requires the following URI parameters.

[name](#) (p. 333)

The name of the slot type. The name is not case sensitive.

The name can't match a built-in slot type name, or a built-in slot type name with "AMAZON." removed. For example, because there is a built-in slot type called `AMAZON.DATE`, you can't create a custom slot type called `DATE`.

For a list of built-in slot types, see [Slot Type Reference](#) in the Alexa Skills Kit.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[A-Za-z_?]+$`

### Request Body

The request accepts the following data in JSON format.

[checksum](#) (p. 333)

Identifies a specific revision of the `$LATEST` version.

When you create a new slot type, leave the `checksum` field blank. If you specify a checksum you get a `BadRequestException` exception.

When you want to update a slot type, set the `checksum` field to the checksum of the most recent revision of the `$LATEST` version. If you don't specify the `checksum` field, or if the checksum does not match the `$LATEST` version, you get a `PreconditionFailedException` exception.

Type: String

Required: No

[createVersion \(p. 333\)](#)

When set to `true` a new numbered version of the slot type is created. This is the same as calling the `CreateSlotTypeVersion` operation. If you do not specify `createVersion`, the default is `false`.

Type: Boolean

Required: No

[description \(p. 333\)](#)

A description of the slot type.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

Required: No

[enumerationValues \(p. 333\)](#)

A list of `EnumerationValue` objects that defines the values that the slot type can take. Each value can have a list of `synonyms`, which are additional values that help train the machine learning model about the values that it resolves for a slot.

When Amazon Lex resolves a slot value, it generates a resolution list that contains up to five possible values for the slot. If you are using a Lambda function, this resolution list is passed to the function. If you are not using a Lambda function you can choose to return the value that the user entered or the first value in the resolution list as the slot value. The `valueSelectionStrategy` field indicates the option to use.

Type: Array of [EnumerationValue \(p. 377\)](#) objects

Array Members: Minimum number of 1 item. Maximum number of 10000 items.

Required: No

[valueSelectionStrategy \(p. 333\)](#)

Determines the slot resolution strategy that Amazon Lex uses to return slot type values. The field can be set to one of the following values:

- `ORIGINAL_VALUE` - Returns the value entered by the user, if the user value is similar to the slot value.
- `TOP_RESOLUTION` - If there is a resolution list for the slot, return the first value in the resolution list as the slot type value. If there is no resolution list, null is returned.

If you don't specify the `valueSelectionStrategy`, the default is `ORIGINAL_VALUE`.

Type: String

Valid Values: `ORIGINAL_VALUE` | `TOP_RESOLUTION`

Required: No

## Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "checksum": "string",
  "createdDate": number,
  "createVersion": boolean,
  "description": "string",
  "enumerationValues": [
    {
      "synonyms": [ "string" ],
      "value": "string"
    }
  ],
  "lastUpdatedDate": number,
  "name": "string",
  "valueSelectionStrategy": "string",
  "version": "string"
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

### [checksum \(p. 335\)](#)

Checksum of the `$LATEST` version of the slot type.

Type: String

### [createdDate \(p. 335\)](#)

The date that the slot type was created.

Type: Timestamp

### [createVersion \(p. 335\)](#)

True if a new version of the slot type was created. If the `createVersion` field was not specified in the request, the `createVersion` field is set to false in the response.

Type: Boolean

### [description \(p. 335\)](#)

A description of the slot type.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

### [enumerationValues \(p. 335\)](#)

A list of `EnumerationValue` objects that defines the values that the slot type can take.

Type: Array of [EnumerationValue \(p. 377\)](#) objects

Array Members: Minimum number of 1 item. Maximum number of 10000 items.

#### [lastUpdatedDate \(p. 335\)](#)

The date that the slot type was updated. When you create a slot type, the creation date and last update date are the same.

Type: Timestamp

#### [name \(p. 335\)](#)

The name of the slot type.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[A-Za-z_?]+$`

#### [valueSelectionStrategy \(p. 335\)](#)

The slot resolution strategy that Amazon Lex uses to determine the value of the slot. For more information, see [PutSlotType \(p. 333\)](#).

Type: String

Valid Values: ORIGINAL\_VALUE | TOP\_RESOLUTION

#### [version \(p. 335\)](#)

The version of the slot type. For a new slot type, the version is always `$LATEST`.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `\$LATEST|[0-9]+`

## Errors

### BadRequestException

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

### ConflictException

There was a conflict processing the request. Try your request again.

HTTP Status Code: 409

### InternalFailureException

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

### LimitExceededException

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

### PreconditionFailedException

The checksum of the resource that you are trying to change does not match the checksum in the request. Check the resource's checksum and try again.

HTTP Status Code: 412

### See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## StartImport

Service: Amazon Lex Model Building Service

Starts a job to import a resource to Amazon Lex.

### Request Syntax

```
POST /imports/ HTTP/1.1
Content-type: application/json

{
  "mergeStrategy": "string",
  "payload": blob,
  "resourceType": "string"
}
```

### URI Request Parameters

The request does not use any URI parameters.

### Request Body

The request accepts the following data in JSON format.

#### [mergeStrategy \(p. 338\)](#)

Specifies the action that the `StartImport` operation should take when there is an existing resource with the same name.

- `FAIL_ON_CONFLICT` - The import operation is stopped on the first conflict between a resource in the import file and an existing resource. The name of the resource causing the conflict is in the `failureReason` field of the response to the `GetImport` operation.

`OVERWRITE_LATEST` - The import operation proceeds even if there is a conflict with an existing resource. The `$LATEST` version of the existing resource is overwritten with the data from the import file.

Type: String

Valid Values: `OVERWRITE_LATEST` | `FAIL_ON_CONFLICT`

Required: Yes

#### [payload \(p. 338\)](#)

A zip archive in binary format. The archive should contain one file, a JSON file containing the resource to import. The resource should match the type specified in the `resourceType` field.

Type: Base64-encoded binary data object

Required: Yes

#### [resourceType \(p. 338\)](#)

Specifies the type of resource to export. Each resource also exports any resources that it depends on.

- A bot exports dependent intents.
- An intent exports dependent slot types.

Type: String

Valid Values: `BOT` | `INTENT` | `SLOT_TYPE`

Required: Yes

## Response Syntax

```
HTTP/1.1 201
Content-type: application/json

{
  "createdDate": number,
  "importId": "string",
  "importStatus": "string",
  "mergeStrategy": "string",
  "name": "string",
  "resourceType": "string"
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 201 response.

The following data is returned in JSON format by the service.

### [createdDate \(p. 339\)](#)

A timestamp for the date and time that the import job was requested.

Type: Timestamp

### [importId \(p. 339\)](#)

The identifier for the specific import job.

Type: String

### [importStatus \(p. 339\)](#)

The status of the import job. If the status is `FAILED`, you can get the reason for the failure using the `GetImport` operation.

Type: String

Valid Values: `IN_PROGRESS` | `COMPLETE` | `FAILED`

### [mergeStrategy \(p. 339\)](#)

The action to take when there is a merge conflict.

Type: String

Valid Values: `OVERWRITE_LATEST` | `FAIL_ON_CONFLICT`

### [name \(p. 339\)](#)

The name given to the import job.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `[a-zA-Z_]+`

### [resourceType \(p. 339\)](#)

The type of resource to import.



Type: String

Valid Values: BOT | INTENT | SLOT\_TYPE

## Errors

### BadRequestException

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

### InternalFailureException

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

### LimitExceededException

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## Amazon Lex Runtime Service

The following actions are supported by Amazon Lex Runtime Service:

- [DeleteSession](#) (p. 341)
- [GetSession](#) (p. 344)
- [PostContent](#) (p. 347)
- [PostText](#) (p. 355)
- [PutSession](#) (p. 361)

## DeleteSession

Service: Amazon Lex Runtime Service

Removes session information for a specified bot, alias, and user ID.

### Request Syntax

```
DELETE /bot/botName/alias/botAlias/user/userId/session HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

[botAlias \(p. 341\)](#)

The alias in use for the bot that contains the session data.

[botName \(p. 341\)](#)

The name of the bot that contains the session data.

[userId \(p. 341\)](#)

The identifier of the user associated with the session data.

Length Constraints: Minimum length of 2. Maximum length of 100.

Pattern: `[0-9a-zA-Z._:-]+`

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "botAlias": "string",
  "botName": "string",
  "sessionId": "string",
  "userId": "string"
}
```

### Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[botAlias \(p. 341\)](#)

The alias in use for the bot associated with the session data.

Type: String

[botName \(p. 341\)](#)

The name of the bot associated with the session data.

Type: String

[sessionId \(p. 341\)](#)

The unique identifier for the session.

Type: String

[userId \(p. 341\)](#)

The ID of the client application user.

Type: String

Length Constraints: Minimum length of 2. Maximum length of 100.

Pattern: `[0-9a-zA-Z._:-]+`

## Errors

### BadRequestException

Request validation failed, there is no usable message in the context, or the bot build failed, is still in progress, or contains unbuilt changes.

HTTP Status Code: 400

### ConflictException

Two clients are using the same AWS account, Amazon Lex bot, and user ID.

HTTP Status Code: 409

### InternalFailureException

Internal service error. Retry the call.

HTTP Status Code: 500

### LimitExceededException

Exceeded a limit.

HTTP Status Code: 429

### NotFoundException

The resource (such as the Amazon Lex bot or an alias) that is referred to is not found.

HTTP Status Code: 404

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)

- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## GetSession

Service: Amazon Lex Runtime Service

Returns session information for a specified bot, alias, and user ID.

### Request Syntax

```
GET /bot/botName/alias/botAlias/user/userId/session HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

*botAlias* (p. 344)

The alias in use for the bot that contains the session data.

*botName* (p. 344)

The name of the bot that contains the session data.

*userId* (p. 344)

The ID of the client application user. Amazon Lex uses this to identify a user's conversation with your bot.

Length Constraints: Minimum length of 2. Maximum length of 100.

Pattern: [0-9a-zA-Z.\_:-]+

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "dialogAction": {
    "fulfillmentState": "string",
    "intentName": "string",
    "message": "string",
    "messageFormat": "string",
    "slots": {
      "string" : "string"
    },
    "slotToElicit": "string",
    "type": "string"
  },
  "recentIntentSummaryView": [
    {
      "confirmationStatus": "string",
      "dialogActionType": "string",
      "fulfillmentState": "string",
      "intentName": "string",
      "slots": {
        "string" : "string"
      },
    },
  ]
}
```

```
        "slotToElicit": "string"
    }
  ],
  "sessionAttributes": {
    "string": "string"
  },
  "sessionId": "string"
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

### [dialogAction \(p. 344\)](#)

Describes the current state of the bot.

Type: [DialogAction \(p. 394\)](#) object

### [recentIntentSummaryView \(p. 344\)](#)

An array of information about the intents used in the session. The array can contain a maximum of three summaries. If more than three intents are used in the session, the `recentIntentSummaryView` operation contains information about the last three intents used.

Type: Array of [IntentSummary \(p. 398\)](#) objects

Array Members: Minimum number of 0 items. Maximum number of 3 items.

### [sessionAttributes \(p. 344\)](#)

Map of key/value pairs representing the session-specific context information. It contains application information passed between Amazon Lex and a client application.

Type: String to string map

### [sessionId \(p. 344\)](#)

A unique identifier for the session.

Type: String

## Errors

### `BadRequestException`

Request validation failed, there is no usable message in the context, or the bot build failed, is still in progress, or contains unbuilt changes.

HTTP Status Code: 400

### `InternalFailureException`

Internal service error. Retry the call.

HTTP Status Code: 500

### `LimitExceededException`

Exceeded a limit.

HTTP Status Code: 429

### NotFoundException

The resource (such as the Amazon Lex bot or an alias) that is referred to is not found.

HTTP Status Code: 404

### See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## PostContent

Service: Amazon Lex Runtime Service

Sends user input (text or speech) to Amazon Lex. Clients use this API to send text and audio requests to Amazon Lex at runtime. Amazon Lex interprets the user input using the machine learning model that it built for the bot.

The `PostContent` operation supports audio input at 8kHz and 16kHz. You can use 8kHz audio to achieve higher speech recognition accuracy in telephone audio applications.

In response, Amazon Lex returns the next message to convey to the user. Consider the following example messages:

- For a user input "I would like a pizza," Amazon Lex might return a response with a message eliciting slot data (for example, `PizzaSize`): "What size pizza would you like?".
- After the user provides all of the pizza order information, Amazon Lex might return a response with a message to get user confirmation: "Order the pizza?".
- After the user replies "Yes" to the confirmation prompt, Amazon Lex might return a conclusion statement: "Thank you, your cheese pizza has been ordered."

Not all Amazon Lex messages require a response from the user. For example, conclusion statements do not require a response. Some messages require only a yes or no response. In addition to the message, Amazon Lex provides additional context about the message in the response that you can use to enhance client behavior, such as displaying the appropriate client user interface. Consider the following examples:

- If the message is to elicit slot data, Amazon Lex returns the following context information:
  - `x-amz-lex-dialog-state` header set to `ElicitSlot`
  - `x-amz-lex-intent-name` header set to the intent name in the current context
  - `x-amz-lex-slot-to-elicite` header set to the slot name for which the message is eliciting information
  - `x-amz-lex-slots` header set to a map of slots configured for the intent with their current values
- If the message is a confirmation prompt, the `x-amz-lex-dialog-state` header is set to `Confirmation` and the `x-amz-lex-slot-to-elicite` header is omitted.
- If the message is a clarification prompt configured for the intent, indicating that the user intent is not understood, the `x-amz-dialog-state` header is set to `ElicitIntent` and the `x-amz-slot-to-elicite` header is omitted.

In addition, Amazon Lex also returns your application-specific `sessionAttributes`. For more information, see [Managing Conversation Context](#).

## Request Syntax

```
POST /bot/botName/alias/botAlias/user/userId/content HTTP/1.1
x-amz-lex-session-attributes: sessionAttributes
x-amz-lex-request-attributes: requestAttributes
Content-Type: contentType
Accept: accept

inputStream
```

## URI Request Parameters

The request requires the following URI parameters.



#### [accept \(p. 347\)](#)

You pass this value as the `Accept` HTTP header.

The message Amazon Lex returns in the response can be either text or speech based on the `Accept` HTTP header value in the request.

- If the value is `text/plain; charset=utf-8`, Amazon Lex returns text in the response.
- If the value begins with `audio/`, Amazon Lex returns speech in the response. Amazon Lex uses Amazon Polly to generate the speech (using the configuration you specified in the `Accept` header). For example, if you specify `audio/mpeg` as the value, Amazon Lex returns speech in the MPEG format.
- If the value is `audio/pcm`, the speech returned is `audio/pcm` in 16-bit, little endian format.
- The following are the accepted values:
  - `audio/mpeg`
  - `audio/ogg`
  - `audio/pcm`
  - `text/plain; charset=utf-8`
  - `audio/*` (defaults to `mpeg`)

#### [botAlias \(p. 347\)](#)

Alias of the Amazon Lex bot.

#### [botName \(p. 347\)](#)

Name of the Amazon Lex bot.

#### [contentType \(p. 347\)](#)

You pass this value as the `Content-Type` HTTP header.

Indicates the audio format or text. The header value must start with one of the following prefixes:

- PCM format, audio data must be in little-endian byte order.
  - `audio/l16; rate=16000; channels=1`
  - `audio/x-l16; sample-rate=16000; channel-count=1`
  - `audio/lpcm; sample-rate=8000; sample-size-bits=16; channel-count=1; is-big-endian=false`
- Opus format
  - `audio/x-cbr-opus-with-preamble; preamble-size=0; bit-rate=256000; frame-size-milliseconds=4`
- Text format
  - `text/plain; charset=utf-8`

#### [requestAttributes \(p. 347\)](#)

You pass this value as the `x-amz-lex-request-attributes` HTTP header.

Request-specific information passed between Amazon Lex and a client application. The value must be a JSON serialized and base64 encoded map with string keys and values. The total size of the `requestAttributes` and `sessionAttributes` headers is limited to 12 KB.

The namespace `x-amz-lex:` is reserved for special attributes. Don't create any request attributes with the prefix `x-amz-lex:`.

For more information, see [Setting Request Attributes](#).

#### [sessionAttributes \(p. 347\)](#)

You pass this value as the `x-amz-lex-session-attributes` HTTP header.

Application-specific information passed between Amazon Lex and a client application. The value must be a JSON serialized and base64 encoded map with string keys and values. The total size of the `sessionAttributes` and `requestAttributes` headers is limited to 12 KB.

For more information, see [Setting Session Attributes](#).

#### [userId \(p. 347\)](#)

The ID of the client application user. Amazon Lex uses this to identify a user's conversation with your bot. At runtime, each request must contain the `userId` field.

To decide the user ID to use for your application, consider the following factors.

- The `userId` field must not contain any personally identifiable information of the user, for example, name, personal identification numbers, or other end user personal information.
- If you want a user to start a conversation on one device and continue on another device, use a user-specific identifier.
- If you want the same user to be able to have two independent conversations on two different devices, choose a device-specific identifier.
- A user can't have two independent conversations with two different versions of the same bot. For example, a user can't have a conversation with the PROD and BETA versions of the same bot. If you anticipate that a user will need to have conversation with two different versions, for example, while testing, include the bot alias in the user ID to separate the two conversations.

Length Constraints: Minimum length of 2. Maximum length of 100.

Pattern: `[0-9a-zA-Z._:-]+`

## Request Body

The request accepts the following binary data.

#### [inputStream \(p. 347\)](#)

User input in PCM or Opus audio format or text format as described in the `Content-Type` HTTP header.

You can stream audio data to Amazon Lex or you can create a local buffer that captures all of the audio data before sending. In general, you get better performance if you stream audio data rather than buffering the data locally.

## Response Syntax

```
HTTP/1.1 200
Content-Type: contentType
x-amz-lex-intent-name: intentName
x-amz-lex-slots: slots
x-amz-lex-session-attributes: sessionAttributes
x-amz-lex-message: message
x-amz-lex-message-format: messageFormat
x-amz-lex-dialog-state: dialogState
x-amz-lex-slot-to-elicite: slotToElicite
x-amz-lex-input-transcript: inputTranscript

audioStream
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The response returns the following HTTP headers.

#### [contentType \(p. 349\)](#)

Content type as specified in the `Accept` HTTP header in the request.

#### [dialogState \(p. 349\)](#)

Identifies the current state of the user interaction. Amazon Lex returns one of the following values as `dialogState`. The client can optionally use this information to customize the user interface.

- `ElicitIntent` - Amazon Lex wants to elicit the user's intent. Consider the following examples:

For example, a user might utter an intent ("I want to order a pizza"). If Amazon Lex cannot infer the user intent from this utterance, it will return this dialog state.

- `ConfirmIntent` - Amazon Lex is expecting a "yes" or "no" response.

For example, Amazon Lex wants user confirmation before fulfilling an intent. Instead of a simple "yes" or "no" response, a user might respond with additional information. For example, "yes, but make it a thick crust pizza" or "no, I want to order a drink." Amazon Lex can process such additional information (in these examples, update the crust type slot or change the intent from `OrderPizza` to `OrderDrink`).

- `ElicitSlot` - Amazon Lex is expecting the value of a slot for the current intent.

For example, suppose that in the response Amazon Lex sends this message: "What size pizza would you like?". A user might reply with the slot value (e.g., "medium"). The user might also provide additional information in the response (e.g., "medium thick crust pizza"). Amazon Lex can process such additional information appropriately.

- `Fulfilled` - Conveys that the Lambda function has successfully fulfilled the intent.
- `ReadyForFulfillment` - Conveys that the client has to fulfill the request.
- `Failed` - Conveys that the conversation with the user failed.

This can happen for various reasons, including that the user does not provide an appropriate response to prompts from the service (you can configure how many times Amazon Lex can prompt a user for specific information), or if the Lambda function fails to fulfill the intent.

Valid Values: `ElicitIntent` | `ConfirmIntent` | `ElicitSlot` | `Fulfilled` | `ReadyForFulfillment` | `Failed`

#### [inputTranscript \(p. 349\)](#)

The text used to process the request.

If the input was an audio stream, the `inputTranscript` field contains the text extracted from the audio stream. This is the text that is actually processed to recognize intents and slot values. You can use this information to determine if Amazon Lex is correctly processing the audio that you send.

#### [intentName \(p. 349\)](#)

Current user intent that Amazon Lex is aware of.

#### [message \(p. 349\)](#)

The message to convey to the user. The message can come from the bot's configuration or from a Lambda function.

If the intent is not configured with a Lambda function, or if the Lambda function returned `Delegate` as the `dialogAction.type` in its response, Amazon Lex decides on the next course of action and selects an appropriate message from the bot's configuration based on the current interaction context. For example, if Amazon Lex isn't able to understand user input, it uses a clarification prompt message.

When you create an intent you can assign messages to groups. When messages are assigned to groups Amazon Lex returns one message from each group in the response. The message field is an escaped JSON string containing the messages. For more information about the structure of the JSON string returned, see [Formatos de mensagem suportados \(p. 15\)](#).

If the Lambda function returns a message, Amazon Lex passes it to the client in its response.

Length Constraints: Minimum length of 1. Maximum length of 1024.

[messageFormat \(p. 349\)](#)

The format of the response message. One of the following values:

- `PlainText` - The message contains plain UTF-8 text.
- `CustomPayload` - The message is a custom format for the client.
- `SSML` - The message contains text formatted for voice output.
- `Composite` - The message contains an escaped JSON object containing one or more messages from the groups that messages were assigned to when the intent was created.

Valid Values: `PlainText` | `CustomPayload` | `SSML` | `Composite`

[sessionAttributes \(p. 349\)](#)

Map of key/value pairs representing the session-specific context information.

[slots \(p. 349\)](#)

Map of zero or more intent slots (name/value pairs) Amazon Lex detected from the user input during the conversation. The field is base-64 encoded.

Amazon Lex creates a resolution list containing likely values for a slot. The value that it returns is determined by the `valueSelectionStrategy` selected when the slot type was created or updated. If `valueSelectionStrategy` is set to `ORIGINAL_VALUE`, the value provided by the user is returned, if the user value is similar to the slot values. If `valueSelectionStrategy` is set to `TOP_RESOLUTION` Amazon Lex returns the first value in the resolution list or, if there is no resolution list, null. If you don't specify a `valueSelectionStrategy`, the default is `ORIGINAL_VALUE`.

[slotToElicit \(p. 349\)](#)

If the `dialogState` value is `ElicitSlot`, returns the name of the slot for which Amazon Lex is eliciting a value.

The response returns the following as the HTTP body.

[audioStream \(p. 349\)](#)

The prompt (or statement) to convey to the user. This is based on the bot configuration and context. For example, if Amazon Lex did not understand the user intent, it sends the `clarificationPrompt` configured for the bot. If the intent requires confirmation before taking the fulfillment action, it sends the `confirmationPrompt`. Another example: Suppose that the Lambda function successfully fulfilled the intent, and sent a message to convey to the user. Then Amazon Lex sends that message in the response.

## Errors

`BadGatewayException`

Either the Amazon Lex bot is still building, or one of the dependent services (Amazon Polly, AWS Lambda) failed with an internal service error.

HTTP Status Code: 502

**BadRequestException**

Request validation failed, there is no usable message in the context, or the bot build failed, is still in progress, or contains unbuilt changes.

HTTP Status Code: 400

**ConflictException**

Two clients are using the same AWS account, Amazon Lex bot, and user ID.

HTTP Status Code: 409

**DependencyFailedException**

One of the dependencies, such as AWS Lambda or Amazon Polly, threw an exception. For example,

- If Amazon Lex does not have sufficient permissions to call a Lambda function.
- If a Lambda function takes longer than 30 seconds to execute.
- If a fulfillment Lambda function returns a `Delegat`e dialog action without removing any slot values.

HTTP Status Code: 424

**InternalFailureException**

Internal service error. Retry the call.

HTTP Status Code: 500

**LimitExceededException**

Exceeded a limit.

HTTP Status Code: 429

**LoopDetectedException**

This exception is not used.

HTTP Status Code: 508

**NotAcceptableException**

The accept header in the request does not have a valid value.

HTTP Status Code: 406

**NotFoundException**

The resource (such as the Amazon Lex bot or an alias) that is referred to is not found.

HTTP Status Code: 404

**RequestTimeoutException**

The input speech is too long.

HTTP Status Code: 408

**UnsupportedMediaTypeException**

The Content-Type header (`PostContent` API) has an invalid value.

HTTP Status Code: 415

### Example 1

In the response, the `x-amz-lex-message` header shows the response that Amazon Lex returned. The client can then send this response to the user. The same message is sent in audio/MPEG format through chunked encoding (as requested).

## Sample Request

### Sample Response

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## PostText

Service: Amazon Lex Runtime Service

Sends user input (text or SSML) to Amazon Lex. Client applications can use this API to send requests to Amazon Lex at runtime. Amazon Lex then interprets the user input using the machine learning model it built for the bot.

In response, Amazon Lex returns the next message to convey to the user an optional `responseCard` to display. Consider the following example messages:

- For a user input "I would like a pizza", Amazon Lex might return a response with a message eliciting slot data (for example, `PizzaSize`): "What size pizza would you like?"
- After the user provides all of the pizza order information, Amazon Lex might return a response with a message to obtain user confirmation "Proceed with the pizza order?"
- After the user replies to a confirmation prompt with a "yes", Amazon Lex might return a conclusion statement: "Thank you, your cheese pizza has been ordered."

Not all Amazon Lex messages require a user response. For example, a conclusion statement does not require a response. Some messages require only a "yes" or "no" user response. In addition to the message, Amazon Lex provides additional context about the message in the response that you might use to enhance client behavior, for example, to display the appropriate client user interface. These are the `slotToElicit`, `dialogState`, `intentName`, and `slots` fields in the response. Consider the following examples:

- If the message is to elicit slot data, Amazon Lex returns the following context information:
  - `dialogState` set to `ElicitSlot`
  - `intentName` set to the intent name in the current context
  - `slotToElicit` set to the slot name for which the message is eliciting information
  - `slots` set to a map of slots, configured for the intent, with currently known values
- If the message is a confirmation prompt, the `dialogState` is set to `ConfirmIntent` and `slotToElicit` is set to null.
- If the message is a clarification prompt (configured for the intent) that indicates that user intent is not understood, the `dialogState` is set to `ElicitIntent` and `slotToElicit` is set to null.

In addition, Amazon Lex also returns your application-specific `sessionAttributes`. For more information, see [Managing Conversation Context](#).

## Request Syntax

```
POST /bot/botName/alias/botAlias/user/userId/text HTTP/1.1
Content-type: application/json

{
  "inputText": "string",
  "requestAttributes": {
    "string" : "string"
  },
  "sessionAttributes": {
    "string" : "string"
  }
}
```

## URI Request Parameters

The request requires the following URI parameters.



#### [botAlias \(p. 355\)](#)

The alias of the Amazon Lex bot.

#### [botName \(p. 355\)](#)

The name of the Amazon Lex bot.

#### [userId \(p. 355\)](#)

The ID of the client application user. Amazon Lex uses this to identify a user's conversation with your bot. At runtime, each request must contain the `userId` field.

To decide the user ID to use for your application, consider the following factors.

- The `userId` field must not contain any personally identifiable information of the user, for example, name, personal identification numbers, or other end user personal information.
- If you want a user to start a conversation on one device and continue on another device, use a user-specific identifier.
- If you want the same user to be able to have two independent conversations on two different devices, choose a device-specific identifier.
- A user can't have two independent conversations with two different versions of the same bot. For example, a user can't have a conversation with the PROD and BETA versions of the same bot. If you anticipate that a user will need to have conversation with two different versions, for example, while testing, include the bot alias in the user ID to separate the two conversations.

Length Constraints: Minimum length of 2. Maximum length of 100.

Pattern: `[0-9a-zA-Z._:-]+`

## Request Body

The request accepts the following data in JSON format.

#### [inputText \(p. 355\)](#)

The text that the user entered (Amazon Lex interprets this text).

When you are using the AWS CLI, you can't pass a URL in the `--input-text` parameter. Pass the URL using the `--cli-input-json` parameter instead.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Required: Yes

#### [requestAttributes \(p. 355\)](#)

Request-specific information passed between Amazon Lex and a client application.

The namespace `x-amz-lex:` is reserved for special attributes. Don't create any request attributes with the prefix `x-amz-lex:`.

For more information, see [Setting Request Attributes](#).

Type: String to string map

Required: No

#### [sessionAttributes \(p. 355\)](#)

Application-specific information passed between Amazon Lex and a client application.

For more information, see [Setting Session Attributes](#).

Type: String to string map

Required: No

## Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "dialogState": "string",
  "intentName": "string",
  "message": "string",
  "messageFormat": "string",
  "responseCard": {
    "contentType": "string",
    "genericAttachments": [
      {
        "attachmentLinkUrl": "string",
        "buttons": [
          {
            "text": "string",
            "value": "string"
          }
        ],
        "imageUrl": "string",
        "subTitle": "string",
        "title": "string"
      }
    ],
    "version": "string"
  },
  "sessionAttributes": {
    "string": "string"
  },
  "slots": {
    "string": "string"
  },
  "slotToElicit": "string"
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

### [dialogState](#) (p. 357)

Identifies the current state of the user interaction. Amazon Lex returns one of the following values as `dialogState`. The client can optionally use this information to customize the user interface.

- `ElicitIntent` - Amazon Lex wants to elicit user intent.

For example, a user might utter an intent ("I want to order a pizza"). If Amazon Lex cannot infer the user intent from this utterance, it will return this `dialogState`.

- `ConfirmIntent` - Amazon Lex is expecting a "yes" or "no" response.

For example, Amazon Lex wants user confirmation before fulfilling an intent.

Instead of a simple "yes" or "no," a user might respond with additional information. For example, "yes, but make it thick crust pizza" or "no, I want to order a drink". Amazon Lex can process such additional information (in these examples, update the crust type slot value, or change intent from OrderPizza to OrderDrink).

- `ElicitSlot` - Amazon Lex is expecting a slot value for the current intent.

For example, suppose that in the response Amazon Lex sends this message: "What size pizza would you like?". A user might reply with the slot value (e.g., "medium"). The user might also provide additional information in the response (e.g., "medium thick crust pizza"). Amazon Lex can process such additional information appropriately.

- `Fulfilled` - Conveys that the Lambda function configured for the intent has successfully fulfilled the intent.
- `ReadyForFulfillment` - Conveys that the client has to fulfill the intent.
- `Failed` - Conveys that the conversation with the user failed.

This can happen for various reasons including that the user did not provide an appropriate response to prompts from the service (you can configure how many times Amazon Lex can prompt a user for specific information), or the Lambda function failed to fulfill the intent.

Type: String

Valid Values: `ElicitIntent` | `ConfirmIntent` | `ElicitSlot` | `Fulfilled` | `ReadyForFulfillment` | `Failed`

[intentName \(p. 357\)](#)

The current user intent that Amazon Lex is aware of.

Type: String

[message \(p. 357\)](#)

The message to convey to the user. The message can come from the bot's configuration or from a Lambda function.

If the intent is not configured with a Lambda function, or if the Lambda function returned `Delegate` as the `dialogAction.type` its response, Amazon Lex decides on the next course of action and selects an appropriate message from the bot's configuration based on the current interaction context. For example, if Amazon Lex isn't able to understand user input, it uses a clarification prompt message.

When you create an intent you can assign messages to groups. When messages are assigned to groups Amazon Lex returns one message from each group in the response. The message field is an escaped JSON string containing the messages. For more information about the structure of the JSON string returned, see [Formatos de mensagem suportados \(p. 15\)](#).

If the Lambda function returns a message, Amazon Lex passes it to the client in its response.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

[messageFormat \(p. 357\)](#)

The format of the response message. One of the following values:

- `PlainText` - The message contains plain UTF-8 text.
- `CustomPayload` - The message is a custom format defined by the Lambda function.
- `SSML` - The message contains text formatted for voice output.
- `Composite` - The message contains an escaped JSON object containing one or more messages from the groups that messages were assigned to when the intent was created.

Type: String

Valid Values: `PlainText` | `CustomPayload` | `SSML` | `Composite`

[responseCard \(p. 357\)](#)

Represents the options that the user has to respond to the current prompt. Response Card can come from the bot configuration (in the Amazon Lex console, choose the settings button next to a slot) or from a code hook (Lambda function).

Type: [ResponseCard \(p. 400\)](#) object

[sessionAttributes \(p. 357\)](#)

A map of key-value pairs representing the session-specific context information.

Type: String to string map

[slots \(p. 357\)](#)

The intent slots that Amazon Lex detected from the user input in the conversation.

Amazon Lex creates a resolution list containing likely values for a slot. The value that it returns is determined by the `valueSelectionStrategy` selected when the slot type was created or updated. If `valueSelectionStrategy` is set to `ORIGINAL_VALUE`, the value provided by the user is returned, if the user value is similar to the slot values. If `valueSelectionStrategy` is set to `TOP_RESOLUTION` Amazon Lex returns the first value in the resolution list or, if there is no resolution list, null. If you don't specify a `valueSelectionStrategy`, the default is `ORIGINAL_VALUE`.

Type: String to string map

[slotToElicit \(p. 357\)](#)

If the `dialogState` value is `ElicitSlot`, returns the name of the slot for which Amazon Lex is eliciting a value.

Type: String

## Errors

### BadGatewayException

Either the Amazon Lex bot is still building, or one of the dependent services (Amazon Polly, AWS Lambda) failed with an internal service error.

HTTP Status Code: 502

### BadRequestException

Request validation failed, there is no usable message in the context, or the bot build failed, is still in progress, or contains unbuilt changes.

HTTP Status Code: 400

### ConflictException

Two clients are using the same AWS account, Amazon Lex bot, and user ID.

HTTP Status Code: 409

### DependencyFailedException

One of the dependencies, such as AWS Lambda or Amazon Polly, threw an exception. For example,

- If Amazon Lex does not have sufficient permissions to call a Lambda function.

- If a Lambda function takes longer than 30 seconds to execute.
- If a fulfillment Lambda function returns a `Delegat` dialog action without removing any slot values.

HTTP Status Code: 424

`InternalFailureException`

Internal service error. Retry the call.

HTTP Status Code: 500

`LimitExceededException`

Exceeded a limit.

HTTP Status Code: 429

`LoopDetectedException`

This exception is not used.

HTTP Status Code: 508

`NotFoundException`

The resource (such as the Amazon Lex bot or an alias) that is referred to is not found.

HTTP Status Code: 404

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## PutSession

Service: Amazon Lex Runtime Service

Creates a new session or modifies an existing session with an Amazon Lex bot. Use this operation to enable your application to set the state of the bot.

For more information, see [Managing Sessions](#).

## Request Syntax

```
POST /bot/botName/alias/botAlias/user/userId/session HTTP/1.1
Accept: accept
Content-type: application/json

{
  "dialogAction": {
    "fulfillmentState": "string",
    "intentName": "string",
    "message": "string",
    "messageFormat": "string",
    "slots": {
      "string" : "string"
    },
    "slotToElicit": "string",
    "type": "string"
  },
  "sessionAttributes": {
    "string" : "string"
  }
}
```

## URI Request Parameters

The request requires the following URI parameters.

[accept \(p. 361\)](#)

The message that Amazon Lex returns in the response can be either text or speech based depending on the value of this field.

- If the value is `text/plain; charset=utf-8`, Amazon Lex returns text in the response.
- If the value begins with `audio/`, Amazon Lex returns speech in the response. Amazon Lex uses Amazon Polly to generate the speech in the configuration that you specify. For example, if you specify `audio/mpeg` as the value, Amazon Lex returns speech in the MPEG format.
- If the value is `audio/pcm`, the speech is returned as `audio/pcm` in 16-bit, little endian format.
- The following are the accepted values:
  - `audio/mpeg`
  - `audio/ogg`
  - `audio/pcm`
  - `audio/*` (defaults to `mpeg`)
  - `text/plain; charset=utf-8`

[botAlias \(p. 361\)](#)

The alias in use for the bot that contains the session data.

[botName \(p. 361\)](#)

The name of the bot that contains the session data.

### [userId \(p. 361\)](#)

The ID of the client application user. Amazon Lex uses this to identify a user's conversation with your bot.

Length Constraints: Minimum length of 2. Maximum length of 100.

Pattern: `[0-9a-zA-Z._:-]+`

## Request Body

The request accepts the following data in JSON format.

### [dialogAction \(p. 361\)](#)

Sets the next action that the bot should take to fulfill the conversation.

Type: [DialogAction \(p. 394\)](#) object

Required: No

### [sessionAttributes \(p. 361\)](#)

Map of key/value pairs representing the session-specific context information. It contains application information passed between Amazon Lex and a client application.

Type: String to string map

Required: No

## Response Syntax

```
HTTP/1.1 200
Content-Type: contentType
x-amz-lex-intent-name: intentName
x-amz-lex-slots: slots
x-amz-lex-session-attributes: sessionAttributes
x-amz-lex-message: message
x-amz-lex-message-format: messageFormat
x-amz-lex-dialog-state: dialogState
x-amz-lex-slot-to-elicit: slotToElicit
x-amz-lex-session-id: sessionId

audioStream
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The response returns the following HTTP headers.

### [contentType \(p. 362\)](#)

Content type as specified in the Accept HTTP header in the request.

### [dialogState \(p. 362\)](#)

- `ConfirmIntent` - Amazon Lex is expecting a "yes" or "no" response to confirm the intent before fulfilling an intent.
- `ElicitIntent` - Amazon Lex wants to elicit the user's intent.

- `ElicitSlot` - Amazon Lex is expecting the value of a slot for the current intent.
- `Failed` - Conveys that the conversation with the user has failed. This can happen for various reasons, including the user does not provide an appropriate response to prompts from the service, or if the Lambda function fails to fulfill the intent.
- `Fulfilled` - Conveys that the Lambda function has successfully fulfilled the intent.
- `ReadyForFulfillment` - Conveys that the client has to fulfill the intent.

Valid Values: `ElicitIntent` | `ConfirmIntent` | `ElicitSlot` | `Fulfilled` | `ReadyForFulfillment` | `Failed`

[intentName \(p. 362\)](#)

The name of the current intent.

[message \(p. 362\)](#)

The next message that should be presented to the user.

Length Constraints: Minimum length of 1. Maximum length of 1024.

[messageFormat \(p. 362\)](#)

The format of the response message. One of the following values:

- `PlainText` - The message contains plain UTF-8 text.
- `CustomPayload` - The message is a custom format for the client.
- `SSML` - The message contains text formatted for voice output.
- `Composite` - The message contains an escaped JSON object containing one or more messages from the groups that messages were assigned to when the intent was created.

Valid Values: `PlainText` | `CustomPayload` | `SSML` | `Composite`

[sessionAttributes \(p. 362\)](#)

Map of key/value pairs representing session-specific context information.

[sessionId \(p. 362\)](#)

A unique identifier for the session.

[slots \(p. 362\)](#)

Map of zero or more intent slots Amazon Lex detected from the user input during the conversation.

Amazon Lex creates a resolution list containing likely values for a slot. The value that it returns is determined by the `valueSelectionStrategy` selected when the slot type was created or updated. If `valueSelectionStrategy` is set to `ORIGINAL_VALUE`, the value provided by the user is returned, if the user value is similar to the slot values. If `valueSelectionStrategy` is set to `TOP_RESOLUTION` Amazon Lex returns the first value in the resolution list or, if there is no resolution list, null. If you don't specify a `valueSelectionStrategy` the default is `ORIGINAL_VALUE`.

[slotToElicit \(p. 362\)](#)

If the `dialogState` is `ElicitSlot`, returns the name of the slot for which Amazon Lex is eliciting a value.

The response returns the following as the HTTP body.

[audioStream \(p. 362\)](#)

The audio version of the message to convey to the user.



## Errors

### BadGatewayException

Either the Amazon Lex bot is still building, or one of the dependent services (Amazon Polly, AWS Lambda) failed with an internal service error.

HTTP Status Code: 502

### BadRequestException

Request validation failed, there is no usable message in the context, or the bot build failed, is still in progress, or contains unbuilt changes.

HTTP Status Code: 400

### ConflictException

Two clients are using the same AWS account, Amazon Lex bot, and user ID.

HTTP Status Code: 409

### DependencyFailedException

One of the dependencies, such as AWS Lambda or Amazon Polly, threw an exception. For example,

- If Amazon Lex does not have sufficient permissions to call a Lambda function.
- If a Lambda function takes longer than 30 seconds to execute.
- If a fulfillment Lambda function returns a `Delegat`e dialog action without removing any slot values.

HTTP Status Code: 424

### InternalFailureException

Internal service error. Retry the call.

HTTP Status Code: 500

### LimitExceededException

Exceeded a limit.

HTTP Status Code: 429

### NotAcceptableException

The accept header in the request does not have a valid value.

HTTP Status Code: 406

### NotFoundException

The resource (such as the Amazon Lex bot or an alias) that is referred to is not found.

HTTP Status Code: 404

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)

- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## Data Types

The following data types are supported by Amazon Lex Model Building Service:

- [BotAliasMetadata](#) (p. 367)
- [BotChannelAssociation](#) (p. 369)
- [BotMetadata](#) (p. 371)
- [BuiltinIntentMetadata](#) (p. 373)
- [BuiltinIntentSlot](#) (p. 374)
- [BuiltinSlotTypeMetadata](#) (p. 375)
- [CodeHook](#) (p. 376)
- [EnumerationValue](#) (p. 377)
- [FollowUpPrompt](#) (p. 378)
- [FulfillmentActivity](#) (p. 379)
- [Intent](#) (p. 380)
- [IntentMetadata](#) (p. 381)
- [Message](#) (p. 383)
- [Prompt](#) (p. 384)
- [ResourceReference](#) (p. 385)
- [Slot](#) (p. 386)
- [SlotTypeMetadata](#) (p. 388)
- [Statement](#) (p. 390)
- [UtteranceData](#) (p. 391)
- [UtteranceList](#) (p. 392)

The following data types are supported by Amazon Lex Runtime Service:

- [Button](#) (p. 393)
- [DialogAction](#) (p. 394)
- [GenericAttachment](#) (p. 396)
- [IntentSummary](#) (p. 398)
- [ResponseCard](#) (p. 400)

## Amazon Lex Model Building Service

The following data types are supported by Amazon Lex Model Building Service:

- [BotAliasMetadata](#) (p. 367)
- [BotChannelAssociation](#) (p. 369)

- [BotMetadata](#) (p. 371)
- [BuiltinIntentMetadata](#) (p. 373)
- [BuiltinIntentSlot](#) (p. 374)
- [BuiltinSlotTypeMetadata](#) (p. 375)
- [CodeHook](#) (p. 376)
- [EnumerationValue](#) (p. 377)
- [FollowUpPrompt](#) (p. 378)
- [FulfillmentActivity](#) (p. 379)
- [Intent](#) (p. 380)
- [IntentMetadata](#) (p. 381)
- [Message](#) (p. 383)
- [Prompt](#) (p. 384)
- [ResourceReference](#) (p. 385)
- [Slot](#) (p. 386)
- [SlotTypeMetadata](#) (p. 388)
- [Statement](#) (p. 390)
- [UtteranceData](#) (p. 391)
- [UtteranceList](#) (p. 392)

## BotAliasMetadata

Service: Amazon Lex Model Building Service

Provides information about a bot alias.

### Contents

#### botName

The name of the bot to which the alias points.

Type: String

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^[A-Za-z_?]+$`

Required: No

#### botVersion

The version of the Amazon Lex bot to which the alias points.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `\$LATEST|[0-9]+`

Required: No

#### checksum

Checksum of the bot alias.

Type: String

Required: No

#### createdDate

The date that the bot alias was created.

Type: Timestamp

Required: No

#### description

A description of the bot alias.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

Required: No

#### lastUpdatedDate

The date that the bot alias was updated. When you create a resource, the creation date and last updated date are the same.

Type: Timestamp

Required: No

#### name

The name of the bot alias.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[A-Za-z_?]+$`

Required: No

### See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## BotChannelAssociation

Service: Amazon Lex Model Building Service

Represents an association between an Amazon Lex bot and an external messaging platform.

### Contents

#### botAlias

An alias pointing to the specific version of the Amazon Lex bot to which this association is being made.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[A-Za-z_?]+$`

Required: No

#### botConfiguration

Provides information necessary to communicate with the messaging platform.

Type: String to string map

Required: No

#### botName

The name of the Amazon Lex bot to which this association is being made.

##### Note

Currently, Amazon Lex supports associations with Facebook and Slack, and Twilio.

Type: String

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^[A-Za-z_?]+$`

Required: No

#### createdDate

The date that the association between the Amazon Lex bot and the channel was created.

Type: Timestamp

Required: No

#### description

A text description of the association you are creating.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

Required: No

#### failureReason

If `status` is `FAILED`, Amazon Lex provides the reason that it failed to create the association.

Type: String

Required: No

name

The name of the association between the bot and the channel.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[A-Za-z_?]+$`

Required: No

status

The status of the bot channel.

- **CREATED** - The channel has been created and is ready for use.
- **IN\_PROGRESS** - Channel creation is in progress.
- **FAILED** - There was an error creating the channel. For information about the reason for the failure, see the `failureReason` field.

Type: String

Valid Values: `IN_PROGRESS` | `CREATED` | `FAILED`

Required: No

type

Specifies the type of association by indicating the type of channel being established between the Amazon Lex bot and the external messaging platform.

Type: String

Valid Values: `Facebook` | `Slack` | `Twilio-Sms` | `Kik`

Required: No

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## BotMetadata

Service: Amazon Lex Model Building Service

Provides information about a bot. .

### Contents

#### createdDate

The date that the bot was created.

Type: Timestamp

Required: No

#### description

A description of the bot.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

Required: No

#### lastUpdatedDate

The date that the bot was updated. When you create a bot, the creation date and last updated date are the same.

Type: Timestamp

Required: No

#### name

The name of the bot.

Type: String

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^[A-Za-z_?]+$`

Required: No

#### status

The status of the bot.

Type: String

Valid Values: `BUILDING` | `READY` | `READY_BASIC_TESTING` | `FAILED` | `NOT_BUILT`

Required: No

#### version

The version of the bot. For a new bot, the version is always `$LATEST`.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `\$LATEST|[0-9]+`



Required: No

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## BuiltinIntentMetadata

Service: Amazon Lex Model Building Service

Provides metadata for a built-in intent.

### Contents

signature

A unique identifier for the built-in intent. To find the signature for an intent, see [Standard Built-in Intents](#) in the Alexa Skills Kit.

Type: String

Required: No

supportedLocales

A list of identifiers for the locales that the intent supports.

Type: Array of strings

Valid Values: en-US

Required: No

### See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## BuiltinIntentSlot

Service: Amazon Lex Model Building Service

Provides information about a slot used in a built-in intent.

### Contents

name

A list of the slots defined for the intent.

Type: String

Required: No

### See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## BuiltinSlotTypeMetadata

Service: Amazon Lex Model Building Service

Provides information about a built in slot type.

### Contents

signature

A unique identifier for the built-in slot type. To find the signature for a slot type, see [Slot Type Reference](#) in the Alexa Skills Kit.

Type: String

Required: No

supportedLocales

A list of target locales for the slot.

Type: Array of strings

Valid Values: en-US

Required: No

### See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## CodeHook

Service: Amazon Lex Model Building Service

Specifies a Lambda function that verifies requests to a bot or fulfills the user's request to a bot..

### Contents

messageVersion

The version of the request-response that you want Amazon Lex to use to invoke your Lambda function. For more information, see [Uso de funções do Lambda \(p. 106\)](#).

Type: String

Length Constraints: Minimum length of 1. Maximum length of 5.

Required: Yes

uri

The Amazon Resource Name (ARN) of the Lambda function.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws:lambda:[a-z]+-[a-z]+-[0-9]:[0-9]{12}:function:[a-zA-Z0-9-_.]+(/[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12})?([a-zA-Z0-9-_.]+)?`

Required: Yes

### See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## EnumerationValue

Service: Amazon Lex Model Building Service

Each slot type can have a set of values. Each enumeration value represents a value the slot type can take.

For example, a pizza ordering bot could have a slot type that specifies the type of crust that the pizza should have. The slot type could include the values

- thick
- thin
- stuffed

## Contents

synonyms

Additional values related to the slot type value.

Type: Array of strings

Length Constraints: Minimum length of 1. Maximum length of 140.

Required: No

value

The value of the slot type.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 140.

Required: Yes

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## FollowUpPrompt

Service: Amazon Lex Model Building Service

A prompt for additional activity after an intent is fulfilled. For example, after the `OrderPizza` intent is fulfilled, you might prompt the user to find out whether the user wants to order drinks.

### Contents

`prompt`

Prompts for information from the user.

Type: [Prompt \(p. 384\)](#) object

Required: Yes

`rejectionStatement`

If the user answers "no" to the question defined in the `prompt` field, Amazon Lex responds with this statement to acknowledge that the intent was canceled.

Type: [Statement \(p. 390\)](#) object

Required: Yes

### See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## FulfillmentActivity

Service: Amazon Lex Model Building Service

Describes how the intent is fulfilled after the user provides all of the information required for the intent. You can provide a Lambda function to process the intent, or you can return the intent information to the client application. We recommend that you use a Lambda function so that the relevant logic lives in the Cloud and limit the client-side code primarily to presentation. If you need to update the logic, you only update the Lambda function; you don't need to upgrade your client application.

Consider the following examples:

- In a pizza ordering application, after the user provides all of the information for placing an order, you use a Lambda function to place an order with a pizzeria.
- In a gaming application, when a user says "pick up a rock," this information must go back to the client application so that it can perform the operation and update the graphics. In this case, you want Amazon Lex to return the intent data to the client.

## Contents

codeHook

A description of the Lambda function that is run to fulfill the intent.

Type: [CodeHook \(p. 376\)](#) object

Required: No

type

How the intent should be fulfilled, either by running a Lambda function or by returning the slot data to the client application.

Type: String

Valid Values: `ReturnIntent` | `CodeHook`

Required: Yes

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)



## Intent

Service: Amazon Lex Model Building Service

Identifies the specific version of an intent.

## Contents

intentName

The name of the intent.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[A-Za-z_?]+$`

Required: Yes

intentVersion

The version of the intent.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `\$LATEST|[0-9]+`

Required: Yes

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## IntentMetadata

Service: Amazon Lex Model Building Service

Provides information about an intent.

### Contents

#### createdDate

The date that the intent was created.

Type: Timestamp

Required: No

#### description

A description of the intent.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

Required: No

#### lastUpdatedDate

The date that the intent was updated. When you create an intent, the creation date and last updated date are the same.

Type: Timestamp

Required: No

#### name

The name of the intent.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[A-Za-z_?]+$`

Required: No

#### version

The version of the intent.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `\$LATEST|[0-9]+`

Required: No

### See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)

- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## Message

Service: Amazon Lex Model Building Service

The message object that provides the message text and its type.

### Contents

content

The text of the message.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1000.

Required: Yes

contentType

The content type of the message string.

Type: String

Valid Values: `PlainText` | `SSML` | `CustomPayload`

Required: Yes

groupName

Identifies the message group that the message belongs to. When a group is assigned to a message, Amazon Lex returns one message from each group in the response.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 5.

Required: No

### See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## Prompt

Service: Amazon Lex Model Building Service

Obtains information from the user. To define a prompt, provide one or more messages and specify the number of attempts to get information from the user. If you provide more than one message, Amazon Lex chooses one of the messages to use to prompt the user. For more information, see [Amazon Lex: Como ele funciona](#) (p. 3).

## Contents

### maxAttempts

The number of times to prompt the user for information.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 5.

Required: Yes

### messages

An array of objects, each of which provides a message string and its type. You can specify the message string in plain text or in Speech Synthesis Markup Language (SSML).

Type: Array of [Message](#) (p. 383) objects

Array Members: Minimum number of 1 item. Maximum number of 15 items.

Required: Yes

### responseCard

A response card. Amazon Lex uses this prompt at runtime, in the `PostText` API response. It substitutes session attributes and slot values for placeholders in the response card. For more information, see [Exemplo: uso de um cartão de resposta](#) (p. 176).

Type: String

Length Constraints: Minimum length of 1. Maximum length of 50000.

Required: No

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## ResourceReference

Service: Amazon Lex Model Building Service

Describes the resource that refers to the resource that you are attempting to delete. This object is returned as part of the `ResourceInUseException` exception.

### Contents

#### name

The name of the resource that is using the resource that you are trying to delete.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `[a-zA-Z_]+`

Required: No

#### version

The version of the resource that is using the resource that you are trying to delete.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `\$LATEST|[0-9]+`

Required: No

### See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## Slot

Service: Amazon Lex Model Building Service

Identifies the version of a specific slot.

### Contents

#### description

A description of the slot.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

Required: No

#### name

The name of the slot.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[A-Za-z](-|_|.)*$`

Required: Yes

#### priority

Directs Lex the order in which to elicit this slot value from the user. For example, if the intent has two slots with priorities 1 and 2, AWS Lex first elicits a value for the slot with priority 1.

If multiple slots share the same priority, the order in which Lex elicits values is arbitrary.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 100.

Required: No

#### responseCard

A set of possible responses for the slot type used by text-based clients. A user chooses an option from the response card, instead of using text to reply.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 50000.

Required: No

#### sampleUtterances

If you know a specific pattern with which users might respond to an Amazon Lex request for a slot value, you can provide those utterances to improve accuracy. This is optional. In most cases, Amazon Lex is capable of understanding user utterances.

Type: Array of strings

Array Members: Minimum number of 0 items. Maximum number of 10 items.

Length Constraints: Minimum length of 1. Maximum length of 200.

Required: No

slotConstraint

Specifies whether the slot is required or optional.

Type: String

Valid Values: `Required` | `Optional`

Required: Yes

slotType

The type of the slot, either a custom slot type that you defined or one of the built-in slot types.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^((AMAZON\ . )_? | [A-Za-z ]_? )+`

Required: No

slotTypeVersion

The version of the slot type.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `\$LATEST | [0-9 ]+`

Required: No

valueElicitationPrompt

The prompt that Amazon Lex uses to elicit the slot value from the user.

Type: [Prompt \(p. 384\)](#) object

Required: No

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)



## SlotTypeMetadata

Service: Amazon Lex Model Building Service

Provides information about a slot type..

### Contents

#### createdDate

The date that the slot type was created.

Type: Timestamp

Required: No

#### description

A description of the slot type.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

Required: No

#### lastUpdatedDate

The date that the slot type was updated. When you create a resource, the creation date and last updated date are the same.

Type: Timestamp

Required: No

#### name

The name of the slot type.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[A-Za-z_?]+$`

Required: No

#### version

The version of the slot type.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `\$LATEST|[0-9]+`

Required: No

### See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)

- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## Statement

Service: Amazon Lex Model Building Service

A collection of messages that convey information to the user. At runtime, Amazon Lex selects the message to convey.

## Contents

### messages

A collection of message objects.

Type: Array of [Message \(p. 383\)](#) objects

Array Members: Minimum number of 1 item. Maximum number of 15 items.

Required: Yes

### responseCard

At runtime, if the client is using the [PostText](#) API, Amazon Lex includes the response card in the response. It substitutes all of the session attributes and slot values for placeholders in the response card.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 50000.

Required: No

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## UtteranceData

Service: Amazon Lex Model Building Service

Provides information about a single utterance that was made to your bot.

### Contents

count

The number of times that the utterance was processed.

Type: Integer

Required: No

distinctUsers

The total number of individuals that used the utterance.

Type: Integer

Required: No

firstUtteredDate

The date that the utterance was first recorded.

Type: Timestamp

Required: No

lastUtteredDate

The date that the utterance was last recorded.

Type: Timestamp

Required: No

utteranceString

The text that was entered by the user or the text representation of an audio clip.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2000.

Required: No

### See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## UtteranceList

Service: Amazon Lex Model Building Service

Provides a list of utterances that have been made to a specific version of your bot. The list contains a maximum of 100 utterances.

### Contents

botVersion

The version of the bot that processed the list.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `\$LATEST|[0-9]+`

Required: No

utterances

One or more [UtteranceData](#) (p. 391) objects that contain information about the utterances that have been made to a bot. The maximum number of object is 100.

Type: Array of [UtteranceData](#) (p. 391) objects

Required: No

### See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## Amazon Lex Runtime Service

The following data types are supported by Amazon Lex Runtime Service:

- [Button](#) (p. 393)
- [DialogAction](#) (p. 394)
- [GenericAttachment](#) (p. 396)
- [IntentSummary](#) (p. 398)
- [ResponseCard](#) (p. 400)

## Button

Service: Amazon Lex Runtime Service

Represents an option to be shown on the client platform (Facebook, Slack, etc.)

### Contents

#### text

Text that is visible to the user on the button.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 15.

Required: Yes

#### value

The value sent to Amazon Lex when a user chooses the button. For example, consider button text "NYC." When the user chooses the button, the value sent can be "New York City."

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1000.

Required: Yes

### See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## DialogAction

Service: Amazon Lex Runtime Service

Describes the next action that the bot should take in its interaction with the user and provides information about the context in which the action takes place. Use the `DialogAction` data type to set the interaction to a specific state, or to return the interaction to a previous state.

### Contents

#### fulfillmentState

The fulfillment state of the intent. The possible values are:

- `Failed` - The Lambda function associated with the intent failed to fulfill the intent.
- `Fulfilled` - The intent has fulfilled by the Lambda function associated with the intent.
- `ReadyForFulfillment` - All of the information necessary for the intent is present and the intent ready to be fulfilled by the client application.

Type: String

Valid Values: `Fulfilled` | `Failed` | `ReadyForFulfillment`

Required: No

#### intentName

The name of the intent.

Type: String

Required: No

#### message

The message that should be shown to the user. If you don't specify a message, Amazon Lex will use the message configured for the intent.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Required: No

#### messageFormat

- `PlainText` - The message contains plain UTF-8 text.
- `CustomPayload` - The message is a custom format for the client.
- `SSML` - The message contains text formatted for voice output.
- `Composite` - The message contains an escaped JSON object containing one or more messages. For more information, see [Message Groups](#).

Type: String

Valid Values: `PlainText` | `CustomPayload` | `SSML` | `Composite`

Required: No

#### slots

Map of the slots that have been gathered and their values.

Type: String to string map

Required: No

slotToElicit

The name of the slot that should be elicited from the user.

Type: String

Required: No

type

The next action that the bot should take in its interaction with the user. The possible values are:

- `ConfirmIntent` - The next action is asking the user if the intent is complete and ready to be fulfilled. This is a yes/no question such as "Place the order?"
- `Close` - Indicates that there will not be a response from the user. For example, the statement "Your order has been placed" does not require a response.
- `Delegate` - The next action is determined by Amazon Lex.
- `ElicitIntent` - The next action is to determine the intent that the user wants to fulfill.
- `ElicitSlot` - The next action is to elicit a slot value from the user.

Type: String

Valid Values: `ElicitIntent` | `ConfirmIntent` | `ElicitSlot` | `Close` | `Delegate`

Required: Yes

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)



## GenericAttachment

Service: Amazon Lex Runtime Service

Represents an option rendered to the user when a prompt is shown. It could be an image, a button, a link, or text.

### Contents

#### attachmentLinkUrl

The URL of an attachment to the response card.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Required: No

#### buttons

The list of options to show to the user.

Type: Array of [Button \(p. 393\)](#) objects

Array Members: Minimum number of 0 items. Maximum number of 5 items.

Required: No

#### imageUrl

The URL of an image that is displayed to the user.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Required: No

#### subTitle

The subtitle shown below the title.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 80.

Required: No

#### title

The title of the option.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 80.

Required: No

### See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)

- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## IntentSummary

Service: Amazon Lex Runtime Service

Provides information about the state of an intent. You can use this information to get the current state of an intent so that you can process the intent, or so that you can return the intent to its previous state.

### Contents

#### confirmationStatus

The status of the intent after the user responds to the confirmation prompt. If the user confirms the intent, Amazon Lex sets this field to `Confirmed`. If the user denies the intent, Amazon Lex sets this value to `Denied`. The possible values are:

- `Confirmed` - The user has responded "Yes" to the confirmation prompt, confirming that the intent is complete and that it is ready to be fulfilled.
- `Denied` - The user has responded "No" to the confirmation prompt.
- `None` - The user has never been prompted for confirmation; or, the user was prompted but did not confirm or deny the prompt.

Type: String

Valid Values: `None` | `Confirmed` | `Denied`

Required: No

#### dialogActionType

The next action that the bot should take in its interaction with the user. The possible values are:

- `ConfirmIntent` - The next action is asking the user if the intent is complete and ready to be fulfilled. This is a yes/no question such as "Place the order?"
- `Close` - Indicates that there will not be a response from the user. For example, the statement "Your order has been placed" does not require a response.
- `ElicitIntent` - The next action is to determine the intent that the user wants to fulfill.
- `ElicitSlot` - The next action is to elicit a slot value from the user.

Type: String

Valid Values: `ElicitIntent` | `ConfirmIntent` | `ElicitSlot` | `Close` | `Delegate`

Required: Yes

#### fulfillmentState

The fulfillment state of the intent. The possible values are:

- `Failed` - The Lambda function associated with the intent failed to fulfill the intent.
- `Fulfilled` - The intent has fulfilled by the Lambda function associated with the intent.
- `ReadyForFulfillment` - All of the information necessary for the intent is present and the intent ready to be fulfilled by the client application.

Type: String

Valid Values: `Fulfilled` | `Failed` | `ReadyForFulfillment`

Required: No

#### intentName

The name of the intent.

Type: String

Required: No

slots

Map of the slots that have been gathered and their values.

Type: String to string map

Required: No

slotToElicit

The next slot to elicit from the user. If there is not slot to elicit, the field is blank.

Type: String

Required: No

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## ResponseCard

Service: Amazon Lex Runtime Service

If you configure a response card when creating your bots, Amazon Lex substitutes the session attributes and slot values that are available, and then returns it. The response card can also come from a Lambda function ( `dialogCodeHook` and `fulfillmentActivity` on an intent).

### Contents

#### `contentType`

The content type of the response.

Type: String

Valid Values: `application/vnd.amazonaws.card.generic`

Required: No

#### `genericAttachments`

An array of attachment objects representing options.

Type: Array of [GenericAttachment](#) (p. 396) objects

Array Members: Minimum number of 0 items. Maximum number of 10 items.

Required: No

#### `version`

The version of the response card format.

Type: String

Required: No

### See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Go - Pilot](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

# Histórico de documentos do Amazon Lex

- Última atualização da documentação: 8 de maio de 2018

A tabela a seguir descreve as alterações importantes em cada versão do Amazon Lex. Para receber notificações sobre atualizações dessa documentação, você pode se inscrever em um feed RSS.

| update-history-change                       | update-history-description   | update-history-date |
|---|--|---------------------|
| <a href="#">Novo recurso</a>                | O Amazon Lex permite que você gerencie informações da sessão para seus bots. Para obter mais informações, consulte <a href="#">Gerenciamento de sessões com a API do Amazon Lex</a> .                                  | August 8, 2019      |
| <a href="#">Expansão de região (p. 401)</a> | O Amazon Lex agora está disponível na região Oeste dos EUA (Oregon) (us-west-2).   | May 8, 2018         |
| <a href="#">Novo recurso (p. 401)</a>       | Adicionado suporte para exportação e importação em formato do Amazon Lex. Para obter mais informações, consulte <a href="#">Importação e exportação de bots, intenções e tipos de slot do Amazon Lex</a> .             | February 13, 2018   |
| <a href="#">Novo recurso (p. 401)</a>       | O Amazon Lex agora oferece suporte a outras mensagens de resposta para bots. Para obter mais informações, consulte <a href="#">Respostas</a> .   | February 8, 2018    |
| <a href="#">Expansão de região (p. 401)</a> | O Amazon Lex agora está disponível na região UE (Irlanda) (eu-west-1).   | November 21, 2017   |
| <a href="#">Novo recurso (p. 401)</a>       | Suporte adicionado para a implantação de bots do Amazon Lex no Kik. Para obter mais informações, consulte <a href="#">Integração de um bot do Amazon Lex com o Kik</a> .   | November 20, 2017   |
| <a href="#">Novo recurso (p. 401)</a>       | Suporte adicionado para novos tipos de slot e atributos de solicitação integrados. Para obter mais informações, consulte <a href="#">Tipos de slot integrados</a> e <a href="#">Definir atributos de solicitação</a> . | November 3, 2017    |

|   |   |                   |
|---|---|-------------------|
| <a href="#">Novo recurso (p. 401)</a>           | Exportação adicionada ao recurso Alexa Skills Kit. Para obter mais informações, consulte <a href="#">Exportar para uma habilidade do Alexa</a> .  | September 7, 2017 |
| <a href="#">Novo recurso (p. 401)</a>           | Suporte a sinônimo adicionado aos valores de tipo de slot. Para obter mais informações, consulte <a href="#">Tipos de slot personalizados</a> .   | August 31, 2017   |
| <a href="#">Novo recurso (p. 401)</a>           | Adicionada integração com o AWS CloudTrail. Para obter mais informações, consulte <a href="#">Monitoramento de chamadas de API do Amazon Lex com logs do AWS CloudTrail</a> .   | August 15, 2017   |
| <a href="#">Documentação expandida (p. 401)</a> | Exemplos de Conceitos básicos adicionados para a AWS CLI. Para obter mais informações, consulte <a href="https://docs.aws.amazon.com/lex/latest/dg/gs-cli.html">https://docs.aws.amazon.com/lex/latest/dg/gs-cli.html</a> | May 22, 2017      |
| <a href="#">Novo guia (p. 401)</a>              | Esta é a primeira versão do Guia do usuário do Amazon Lex.  | April 19, 2017    |

# AWS Glossary

For the latest AWS terminology, see the [AWS Glossary](#) in the AWS General Reference.